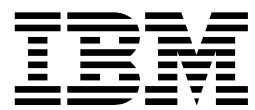


DB2 Universal Database for OS/390



# Руководство по прикладному программированию и SQL

*Версия 6*





DB2 Universal Database for OS/390



# Руководство по прикладному программированию и SQL

*Версия 6*

**Внимание!**

Перед тем как использовать данный документ и продукт, описанный в нем, прочтите общие сведения под заголовком Приложение I, "Замечания" на стр. 977.

**Первое издание (июнь 1999)**

Это издание применимо к Версии 6 DB2 Universal Database Server for OS/390, 5645-DB2, а также ко всем последующим выпускам, если в новых изданиях не будет указано иное. Убедитесь, что вы используете правильное издание для вашей версии продукта.

Конкретные изменения отмечены вертикальной чертой слева от измененного текста. Вертикальная черта слева от заголовка рисунка указывает на то, что рисунок был изменен. Редакторские исправления, не имеющие технического значения, не отмечены.

---

# Содержание

---

<b>Раздел 1. Введение</b>	1
<b>Глава 1. Введение в эту книгу и в библиотеку DB2® for OS/390®</b>	
Для кого предназначена эта книга	3
Структура книги	3
Другие полезные книги	5
Терминология и ссылки	5
Как читать синтаксические диаграммы	6
Как пользоваться библиотекой DB2	7
Как получить информацию о DB2	10
Сводка изменений Версии 6 DB2 UDB for OS/390	11
Сводка изменений в этой книге	16
<b>Раздел 2. Использование запросов SQL</b>	17
<b>Глава 2–1. Получение данных</b>	
Таблицы результатов	19
Типы данных	20
Выбор столбцов: SELECT	21
Выбор строк с указанием критериев поиска: WHERE	26
Использование функций и выражений	34
Упорядочивание строк: ORDER BY	47
Получение значений по группам: GROUP BY	51
Группировка с критериями: HAVING	51
Объединение списков значений: UNION	52
Специальные регистры	54
Поиск информации в каталоге DB2	54
<b>Глава 2–2. Работа с таблицами и изменение данных</b>	
Работа с таблицами	57
Работа с производными таблицами	64
Изменение данных DB2	66
<b>Глава 2–3. Объединение данных из нескольких таблиц</b>	
Внутреннее объединение	76
Полное внешнее объединение	78
Левое внешнее объединение	79
Правое внешнее объединение	79
Правила SQL для операторов, содержащих операции объединения	80
Использование нескольких типов объединения в одном операторе SQL	81
Использование вложенных табличных выражений и табличных пользовательских функций при объединении	82
<b>Глава 2–4. Использование подзапросов</b>	
Обзор основных понятий	85
Как кодировать подзапрос	87
Использование связанных подзапросов	89

<b>Глава 2–5. Исполнение SQL с вашего терминала при помощи SPUFI</b>	93
Размещение входного набора данных и использование SPUFI	93
Изменение умолчаний SPUFI (не обязательное)	96
Ввод операторов SQL	99
Обработка операторов SQL	100
Просмотр вывода	100
<b>Раздел 3. Кодирование SQL в прикладных программах хоста</b>	103
<b>Глава 3–1. Основы кодирования операторов SQL в прикладной программе</b>	107
Общие замечания по примерам написания операторов SQL	108
Разделители операторов SQL	109
Объявление определений таблицы и производной таблицы	109
Обращение к данным с помощью переменных хоста и структур хоста	110
Проверка выполнения операторов SQL	116
<b>Глава 3–2. Использование указателей для получения наборов строк</b>	123
Функции указателя	123
Пример использования указателя	124
Объявление указателя с опцией WITH HOLD	129
<b>Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN</b>	133
Вызов DCLGEN через DB2I	134
Включение объявлений данных в вашу программу	138
Поддержка DCLGEN для языков C, COBOL и PL/I	138
Пример: Добавление объявления таблицы и структуры переменных хоста к библиотеке	140
<b>Глава 3–4. Встроенные операторы SQL в языках хоста</b>	145
Кодирование операторов SQL в программе на языке ассемблер	145
Кодирование операторов SQL в программах на языках C или C <sup>++</sup>	160
Кодирование операторов SQL в программах на языке COBOL	181
Кодирование операторов SQL в программах на языке FORTRAN	204
Кодирование операторов SQL в программах на языке PL/I	215
<b>Глава 3–5. Использование триггеров для работы с активными данными</b>	233
Пример создания и использования триггера	233
Составные части триггера	235
Вызов хранимых процедур и пользовательских функций из триггеров	240
Каскадное срабатывание триггеров	242
Порядок активации триггеров	243
Взаимодействие между триггерами и реляционными связями	244
Создание триггеров, дающих надежные результаты	245
<b>Раздел 4. Использование объектно–реляционных расширений DB2</b>	249
<b>Глава 4–1. Введение в объектно–реляционные расширения DB2</b>	251
<b>Глава 4–2. Программирование больших объектов (LOB)</b>	253
Введение в LOB	253

Объявление переменных хоста больших объектов и локаторов больших объектов . . . . .	257
Материализация большого объекта . . . . .	262
Применение локаторов больших объектов для экономии памяти . . . . .	262
<b>Глава 4–3. Создание и использование пользовательских функций . . . . .</b>	<b>267</b>
Определение, реализация и вызов пользовательских функций . . . . .	267
Определение пользовательской функции . . . . .	270
Реализация внешней пользовательской функции . . . . .	274
Вызов пользовательской функции . . . . .	317
<b>Глава 4–4. Создание и применение пользовательских типов . . . . .</b>	<b>327</b>
Введение в пользовательские типы . . . . .	327
Применение пользовательских типов в прикладных программах . . . . .	328
Объединение пользовательских типов при помощи пользовательских функций и больших объектов . . . . .	334
<b>Раздел 5. Разработка прикладных программ баз данных DB2 . . . . .</b>	<b>339</b>
<b>Глава 5–1. Планирование прекомпиляции и связывания . . . . .</b>	<b>341</b>
Планирование прекомпиляции . . . . .	342
Планирование связывания . . . . .	342
<b>Глава 5–2. Планирование одновременности . . . . .</b>	<b>351</b>
Что такое одновременность? Что такое блокировки? . . . . .	351
Эффекты блокировок DB2 . . . . .	353
Основные рекомендации по обеспечению одновременности . . . . .	357
Атрибуты блокировок транзакций . . . . .	360
Настройка использования блокировок . . . . .	368
Блокировка больших объектов . . . . .	387
<b>Глава 5–3. Планирование восстановления . . . . .</b>	<b>393</b>
Единица работы TSO (пакетной и диалоговой) . . . . .	393
Единицы работы в CICS . . . . .	394
Единица работы в IMS (диалоговой) . . . . .	395
Единица работы в DL/I batch и в IMS batch . . . . .	401
<b>Глава 5–4. Планирование доступа к распределенным данным . . . . .</b>	<b>405</b>
Введение в доступ к распределенным данным . . . . .	405
Два метода программирования распределенных данных . . . . .	408
Особенности программирования методов доступа . . . . .	412
Подготовка программ для доступа DRDA . . . . .	413
Координация изменений нескольких источников данных . . . . .	416
Различные вопросы, связанные с распределенными данными . . . . .	418
<b>Раздел 6. Разработка прикладных программ . . . . .</b>	<b>433</b>
<b>Глава 6–1. Подготовка прикладной программы к запуску . . . . .</b>	<b>435</b>
Действия при подготовке программы . . . . .	435
Шаг 1: Прекомпиляция программы . . . . .	438
Шаг 2: Связывание прикладной программы . . . . .	450

Шаг 3: Компиляция (или ассемблирование) и компоновка прикладной программы . . . . .	463
Шаг 4: Запуск прикладной программы . . . . .	465
Использование процедур JCL для подготовки прикладных программ . . . . .	469
Использование ISPF и DB2 Interactive (DB2I) . . . . .	475
<b>Глава 6–2. Тестирование прикладной программы . . . . .</b>	<b>507</b>
Создание среды тестирования . . . . .	507
Тестирование операторов SQL при помощи SPUFI . . . . .	511
Отладка программы . . . . .	511
Поиск источника ошибки . . . . .	518
<b>Глава 6–3. Обработка программ среды DL/I batch . . . . .</b>	<b>525</b>
Планирование работы с DL/I batch . . . . .	525
Особенности разработки программы . . . . .	526
Наборы входных и выходных данных . . . . .	528
Особенности подготовки программы . . . . .	530
Перезапуск и восстановление . . . . .	533
<b>Раздел 7. Дополнительные приемы программирования . . . . .</b>	<b>537</b>
<b>Глава 7–1. Кодирование динамического SQL в прикладных программах . . . . .</b>	<b>543</b>
Выбор между статическим и динамическим SQL . . . . .	544
Кэширование операторов динамического SQL . . . . .	547
Ограничение динамического SQL при помощи утилиты ограничения ресурсов . . . . .	552
Выбор языка хоста для прикладных программ с динамическим SQL . . . . .	554
Динамический SQL без операторов SELECT . . . . .	555
Динамический SQL для операторов SELECT с фиксированным списком . . . . .	559
Динамический SQL для операторов SELECT с переменным списком . . . . .	562
Использование динамического SQL в языке COBOL . . . . .	577
<b>Глава 7–2. Использование хранимых процедур в системах клиент–сервер . . . . .</b>	<b>579</b>
Введение в хранимые процедуры . . . . .	579
Пример простой хранимой процедуры . . . . .	581
Настройка среды хранимых процедур . . . . .	584
Написание и подготовка хранимой процедуры . . . . .	593
Написание и подготовка прикладной программы, использующей хранимые процедуры . . . . .	607
Выполнение хранимой процедуры . . . . .	643
Тестирование хранимой процедуры . . . . .	649
<b>Глава 7–3. Настройка запросов . . . . .</b>	<b>655</b>
Общие советы и вопросы . . . . .	655
Написание эффективных предикатов . . . . .	658
Общие правила проверки предикатов . . . . .	662
Эффективное использование переменных хоста . . . . .	682
Написание эффективных подзапросов . . . . .	687
Особые способы повлиять на выбор пути доступа . . . . .	693

<b>Глава 7–4. Использование EXPLAIN для повышения производительности SQL</b>	701
Получение информации PLAN_TABLE с помощью EXPLAIN	702
Первые вопросы о доступе к данным	711
Интерпретация доступа к одной таблице	721
Объяснение доступа к нескольким таблицам	728
Предварительная выборка данных	738
Информация о сортировках	743
Обработка производных таблиц и вложенных табличных выражений	745
Оценка стоимости оператора	749
<b>Глава 7–5. Параллельные операции и производительность запросов</b>	753
Сравнение методов параллелизма	754
Разрешение параллельной обработки	757
Когда параллелизм не используется	758
Интерпретация выходных данных EXPLAIN	759
Настройка параллельной обработки	762
Запрещение параллелизма выполнения запроса	763
<b>Глава 7–6. Программирование для утилиты ISPF (Interactive System Productivity Facility – интерактивная утилита производительности системы)</b>	765
Использование ISPF и командного процессора DSN	765
Вызов одной программы SQL через ISPF и DSN	766
Вызов нескольких программ SQL через ISPF и DSN	767
Вызов нескольких программ SQL посредством ISPF и CAF	768
<b>Глава 7–7. Программирование для утилиты подключения по вызову (CAF)</b>	769
Возможности и ограничения утилиты подключения по вызову	769
Как использовать CAF	773
Примеры сценариев	794
Обработчики для прикладной программы	795
Сообщения об ошибках и набор данных DSNTRACE	796
Коды возврата и коды причины CAF	796
Примеры программ	797
<b>Глава 7–8. Программирование для утилиты подключения служб управления восстановимыми ресурсами (RRSAF)</b>	807
Возможности и ограничения RRSAF	807
Как использовать RRSAF	810
Примеры сценариев	841
Коды возврата и коды причин RRSAF	843
Примеры программ	844
<b>Глава 7–9. Особенности программирования для CICS</b>	849
Управление утилитой подключения CICS из прикладной программы	849
Повышение показателя повторного использования потоков	849
Проверка рабочего состояния утилиты подключения CICS	850
<b>Глава 7–10. Приемы программирования: Вопросы и ответы</b>	853
Задание уникального ключа для таблицы	853
Просмотр ранее полученных данных	853
Обновление ранее полученных данных	857

Обновление данных во время получения их из базы данных . . . . .	857
Обновление тысяч строк . . . . .	857
Получение тысяч строк . . . . .	858
Использование SELECT * . . . . .	858
Оптимизация получения небольшого количества строк . . . . .	858
Добавление данных в конец таблицы . . . . .	859
Перевод требований конечных пользователей в операторы SQL . . . . .	859
Изменение определения таблицы . . . . .	859
Хранение данных не в табличном формате . . . . .	860
Поиск нарушенного реляционного или проверочного ограничения . . . . .	860
<b>Приложения . . . . .</b>	861
<b>Приложение А. Таблицы примеров DB2 . . . . .</b>	863
Таблица работ (DSN8610.ACT) . . . . .	863
Таблица отделов (DSN8610.DEPT) . . . . .	864
Таблица сотрудников (DSN8610.EMP) . . . . .	866
Таблица фотографий и резюме сотрудников (DSN8610.EMP_PHOTO_RESUME) . . . . .	870
Таблица проектов (DSN8610.PROJ) . . . . .	871
Таблица работ по проектам (DSN8610.PROJECT) . . . . .	872
Таблица сотрудников работ по проектам (DSN8610.EMPPROJECT) . . . . .	873
Отношения между таблицами . . . . .	875
Производные таблицы для таблиц примеров . . . . .	875
Хранение таблиц программ примеров . . . . .	879
<b>Приложение В. Примеры прикладных программ . . . . .</b>	883
Типы примеров прикладных программ . . . . .	883
Использование прикладных программ . . . . .	885
<b>Приложение С. Как запускать примеры программ DSNTIAUL, DSNTIAD и DSNTEP2 . . . . .</b>	889
Запуск DSNTIAUL . . . . .	890
Запуск DSNTIAD . . . . .	894
Запуск DSNTEP2 . . . . .	896
<b>Приложение D. Примеры программирования . . . . .</b>	899
Пример программы на языке COBOL с динамическим SQL . . . . .	899
Пример динамического и статического SQL в программе на языке C . . . . .	913
Пример программы на языке COBOL с использованием доступа DRDA . . . . .	917
Пример программы на языке COBOL, использующей доступ по собственному протоколу DB2 . . . . .	925
Примеры использования хранимых процедур . . . . .	932
<b>Приложение Е. Подкоманды REBIND для списков планов или пакетов . . . . .</b>	957
Обзор процедуры генерации списка команд REBIND . . . . .	957
Примеры операторов SELECT для генерации команд REBIND . . . . .	958
Пример JCL для запуска списков команд REBIND . . . . .	961
<b>Приложение F. Зарезервированные слова SQL . . . . .</b>	965
<b>Приложение G. Характеристики операторов SQL в DB2 for OS/390 . . . . .</b>	967

Разрешенные действия для операторов SQL . . . . .	967
Операторы SQL, допустимые в функциях и в хранимых процедурах . . . . .	970
<b>Приложение Н. Опции подготовки программ для удаленных пакетов</b>	<b>973</b>
<b>Приложение I. Замечания</b> . . . . .	<b>977</b>
Информация об интерфейсе программирования . . . . .	978
Товарные знаки . . . . .	979
<b>Глоссарий</b> . . . . .	<b>981</b>
<b>Библиография</b> . . . . .	<b>1001</b>
<b>Индекс</b> . . . . .	<b>I-1</b>



---

## Раздел 1. Введение

<b>Глава 1. Введение в эту книгу и в библиотеку DB2® for OS/390®</b>	3
Для кого предназначена эта книга	3
Структура книги	3
Другие полезные книги	5
Терминология и ссылки	5
Как читать синтаксические диаграммы	6
Как пользоваться библиотекой DB2	7
Как получить информацию о DB2	10
DB2 в WWW	10
Публикации по DB2	10
Обучение DB2	11
Как заказать библиотеку DB2	11
Сводка изменений Версии 6 DB2 UDB for OS/390	11
Увеличение объема данных	11
Производительность и доступность	12
Улучшение совместного использования данных	13
Пользовательская производительность	13
Сетевые вычисления	14
Объектно–реляционные расширения и активные данные	15
Увеличение числа функций	15
Возможности DB2 for OS/390	16
Замечания по перенастройке	16
Сводка изменений в этой книге	16



---

# Глава 1. Введение в эту книгу и в библиотеку DB2® for OS/390®

---

## Для кого предназначена эта книга

В этой книге описана разработка и создание прикладных программ, которые обращаются к DB2 for OS/390 (DB2), гибкой реляционной СУБД (DBMS).

Книга предназначена для разработчиков прикладных программ DB2, знакомых с языком SQL (Structured Query Language – язык структурированных запросов) и с одним или несколькими языками программирования, которые поддерживает DB2.

## Структура книги

Эта книга состоит из следующих частей:

- В разделе “Раздел 1. Введение” на стр. 1 описывается эта книга и приводится общая информация о Версии 6 DB2 for OS/390.
- В разделе “Раздел 2. Использование запросов SQL” на стр. 17 приводится сводка элементов SQL, часто используемых в прикладных программах, а также описано тестирование операторов SQL при помощи интерактивного интерфейса SPUFI (SQL processor using file input – процессор SQL, использующий файловый ввод). Этот раздел можно использовать как введение в SQL или как краткий справочник. Начиная писать программы с операторами SQL, посмотрите главы 3, 4, 5 и 6.
- В разделе “Раздел 3. Кодирование SQL в прикладных программах хоста” на стр. 103 рассказано, как кодировать программы с операторами SQL и выполнять их под DB2, для следующих языков:
  - Ассемблер
  - C<sup>1</sup>
  - COBOL
  - FORTRAN
  - PL/I
- В разделе “Глава 3–5. Использование триггеров для работы с активными данными” на стр. 233 объясняется, как использовать в прикладных программах триггеры.
- В разделе “Раздел 4. Использование объектно–реляционных расширений DB2” на стр. 249 объясняется, как использовать в прикладных программах большие объекты, пользовательские функции и пользовательские типы.
- В разделе “Раздел 5. Разработка прикладных программ баз данных DB2” на стр. 339 описаны задачи планирования:
  - В разделе “Глава 5–1. Планирование прекомпиляции и связывания” на стр. 341 рассказано об основных процессах при создании программы.

---

<sup>1</sup> В данной книге С означает и язык C/370™, и язык C++, кроме тех мест, где это оговорено особо.

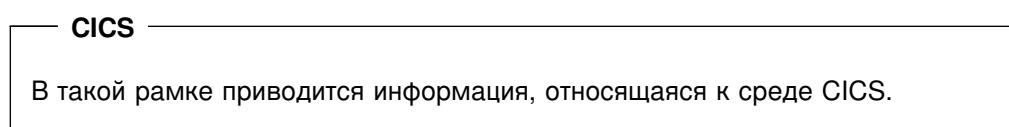
В разделе “Глава 6–1. Подготовка прикладной программы к запуску” на стр. 435 приводятся подробности этих процессов.

- В разделе “Глава 5–2. Планирование одновременности” на стр. 351 описано использование блокировок DB2.
- В разделе “Глава 5–3. Планирование восстановления” на стр. 393 описана разработка программ, которые с максимально возможной быстротой реагируют на прерывания.
- В разделе “Глава 5–4. Планирование доступа к распределенным данным” на стр. 405 рассказано, как методы DB2 для доступа к распределенным данным могут повлиять на структуру программы.
- В разделе “Раздел 6. Разработка прикладных программ” на стр. 433 рассказано, как готовить и тестировать прикладные программы работы с базами данных DB2 и как обрабатывать пакетные программы DL/I.
- В разделе “Раздел 7. Дополнительные приемы программирования” на стр. 537 описаны методы:
  - Кодирования динамического SQL в прикладных программах
  - Использования хранимых процедур
  - Улучшения производительности для запросов
  - Программирования для ISPF (Interactive System Productivity – продуктивность интерактивных систем), CAF (call attachment facility – возможность присоединения вызова) и утилиты подключения служб менеджера восстановимых ресурсов (RRSAF)
  - Особенности программирования для CICS®
- В этих приложениях приводятся:
  - Таблицы примеров DB2, которые используются в данной книге
  - Описания прикладных программ DB2 примеров
  - Примеры программирования, в том числе программы, использующие статический и динамический SQL, хранимые процедуры, и программы, вызывающие хранимые процедуры.
  - Примеры генерации списков для команд rebnd
  - Список зарезервированных слов SQL
  - Действия, разрешенные для операторов SQL DB2
  - Опции связывания для удаленных пакетов
  - Лицензионная информация и сведения о товарных знаках

После этих приложений расположены:

- Глоссарий терминов и сокращений, используемых в книге
- Библиография других книг, которые могут оказаться полезными
- Указатель

Помеченные рамки в данной книге содержат информацию, относящуюся к конкретной среде, например, CICS, IMS™ или TSO.



---

## Другие полезные книги

DB2 for OS/390 – одна из нескольких реляционных СУБД, разработанных IBM®. В каждой из таких систем используется своя разновидность SQL. В данной книге описывается только разновидность, используемая DB2 for OS/390. В других книгах IBM описаны другие разновидности. Список таких книг приводится в библиографии в конце данной книги.

Если вы собираетесь использовать только DB2, держите под рукой *DB2 SQL Reference* – энциклопедический справочник по синтаксису и семантике каждого оператора DB2 SQL. Фундаментальные понятия и идеи SQL описаны в разделе Глава 2 справочника *DB2 SQL Reference*.

Если вы намереваетесь разрабатывать прикладные программы, использующие определение BM SQL, дополнительную информациюсмотрите в справочнике *IBM SQL Reference*.

При подготовке программ к выполнению вам понадобится список опций команд BIND, REBIND PLAN и PACKAGE, который приводится в книге *DB2 Command Reference*.

---

## Терминология и ссылки

В этой книге DB2 Universal Database Server for OS/390 будет называться "DB2 for OS/390". Там, где ясно, какая система имеется в виду, DB2 for OS/390 будет называться просто "DB2". При ссылках на другие книги из библиотеки используется краткое название. (Например, "смотрите *DB2 SQL Reference*" – это ссылка на книгу *IBM DATABASE 2 Universal Database Server for OS/390 SQL Reference*.)

Ссылки в этой книге на "DB2 UDB" относятся к продукту DB2 Universal Database™, который доступен в операционных системах AIX®, OS/2® и Windows NT™. Если в этой книге приводятся ссылки на книги о продукте DB2 UDB, приводится полное название и номер книги.

Следующие термины используются, как указано:

**DB2®** Означает либо лицензированную программу DB2, либо отдельную подсистему DB2.

**C и язык C** Относятся к языку программирования C.

**CICS®** Означает CICS/ESA® и CICS Transaction Server for OS/390 Выпуск 1.

**IMS™** Означает IMS/ESA®.

**MVS** Означает MVS/Enterprise Systems Architecture (MVS/ESA®) – элемент OS/390.

## Как читать синтаксические диаграммы

В синтаксических диаграммах в этой книге используются следующие соглашения:

- Читайте синтаксические диаграммы слева направо и сверху вниз, следуя линии диаграммы.

Символ **►** означает начало оператора.

Символ **—→** означает, что описание оператора продолжается на следующей строке.

Символ **▶** означает, что описание оператора продолжается с предыдущей строки.

Символ **—▶** означает конец оператора.

Диаграммы синтаксических единиц, отличных от законченных операторов, начинаются с символа **►** и заканчиваются символом **—→**.

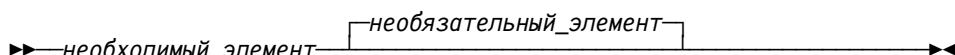
- Необходимые элементы указываются на главном пути.

**►—необходимый\_элемент** 

- Необязательные элементы выводятся под главным путем.

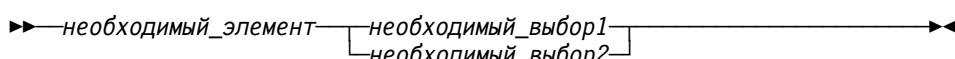
**►—необходимый\_элемент** 

Необязательный элемент может выводиться над главным путем; это не влияет на выполнение оператора и применяется только для удобства чтения.

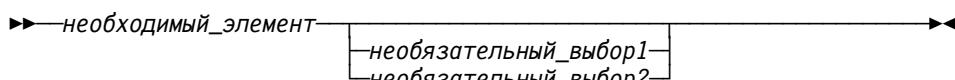
**►—необходимый\_элемент** 

- Если возможен выбор из двух или более элементов, они указываются один над другим в вертикальном ряду.

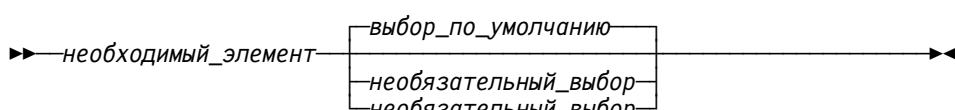
Если один из этих элементов выбрать *необходимо*, этот вертикальный ряд будет находиться на главном пути.

**►—необходимый\_элемент** 

Если выбирать один из элементов *необязательно*, весь вертикальный ряд размещается под главным путем.

**►—необходимый\_элемент** 

Если один из элементов выбирается по умолчанию, он записывается над главным путем, а остальные варианты – внизу.

**►—необходимый\_элемент** 

- Обратная стрелка над главным путем означает, что элемент можно повторить.



Если стрелка повтора содержит запятую, необходимо разделять повторяемые элементы запятой.



Стрелка повтора над вертикальным рядом означает, что можно выбрать несколько элементов.

- Ключевые слова показаны в верхнем регистре (например, FROM). Их надо писать в точности так, как на диаграмме. Переменные записываются в нижнем регистре (например, имя-столбца). Они представляют имена или значения, задаваемые пользователем.
- Если показаны знаки препинания, круглые скобки, арифметические операторы или другие подобные символы, необходимо ввести их, как часть синтаксиса.

## Как пользоваться библиотекой DB2

Названия книг в библиотеке начинаются со слов DB2 Universal Database for OS/390 Version 6 (или DB2 Universal Database for OS/390 Версия 6). Однако ссылки в библиотеке даются в сокращенном виде без имени продукта, версии и выпуска. В них может непосредственно указываться раздел, содержащий нужную информацию. Полный список книг библиотеки и разделов каждой книги смотрите в библиографии в конце этой книги.

Во всей библиотеке "DB2" может означать как лицензированную программу DB2 for OS/390, так и отдельную подсистему DB2 for MVS/ESA. Конкретное значение следует в каждом случае из контекста.

Наиболее практически важная задача, связанная с системой управления базой данных, состоит в формулировке запросов к ней и получении от нее ответов, эту задачу мы называем задачей *конечного пользователя*. Среди других необходимых задач – определение параметров системы, занесение данных в базы и т.д. Задачи, связанные с DB2, сгруппированы в следующие крупные категории (хотя дополнительную информацию, относящуюся ко всем перечисленным ниже задачам для новых версий DB2, можно найти в руководстве *DB2 Release Guide*):

**Установка:** Если вы только устанавливаете систему DB2, все необходимые сведения вы найдете в руководстве *DB2 Installation Guide*.

Если вы собираетесь работать с совместно используемыми данными, вам понадобится также книга *DB2 Data Sharing: Planning and Administration*, где описываются особенности установки для совместного использования данных.

**Задачи конечного пользователя:** Для получения данных конечные пользователи применяют операторы SQL. С помощью операторов SQL можно также выполнять вставку, изменение или удаление данных, Конечным

пользователям может понадобиться введение в SQL, подробные инструкции по пользованию SPUFI и алфавитный справочник типов операторов SQL. Эту информацию можно найти в этой книге и в книге *DB2 SQL Reference*.

Конечные пользователи также могут выполнять операторы SQL через утилиту управления запросами (Query Management Facility, QMF™) или другую программу; все необходимые им инструкции или ссылки можно найти в библиотеке для этой программы. Список книг библиотеки QMF приводится в библиографии в конце данной книги.

**Прикладное программирование:** Некоторые пользователи обращаются к DB2, сами не зная об этом, когда используют программы, содержащие операторы SQL. Такие программы пишут разработчики прикладных программ DB2. Поскольку они пишут операторы SQL, им необходимы книги *Руководство по прикладному программированию и языку SQL для DB2*, *DB2 SQL Reference* и *DB2 ODBC Guide and Reference*, как и конечным пользователям.

Разработчикам прикладных программ требуются также инструкции по многим другим темам:

- Как передавать данные между DB2 и программой хоста на языках COBOL, С или, например, FORTRAN
- Как подготовить к компиляции программы со встроенными операторами SQL
- Как одновременно обрабатывать данные из двух систем, например, DB2 и IMS или DB2 и CICS®
- Как писать распределенные прикладные программы, компоненты которых работают на разных платформах
- Как писать прикладные программы, использующие DB2 ODBC для доступа к серверам DB2
- Как писать прикладные программы, использующие Open Database Connectivity (ODBC) для доступа к серверам DB2
- Как писать прикладные программы для доступа к серверам DB2 на языке Java™

Материал, необходимый для написания программ хоста с SQL, содержится в книгах *Руководство по прикладному программированию и языку SQL для DB2* и *DB2 Application Programming Guide and Reference for Java™*. Материал, необходимый для написания прикладных программ, использующих для доступа к серверам DB2 DB2 ODBC или ODBC, приводится в книге *DB2 ODBC Guide and Reference*. Об обработке ошибок можно узнать из книги *DB2 Сообщения и коды*.

Информацию о написании прикладных программ, компоненты которых работают на разных plataформах, можно найти в книге *Distributed Relational Database Architecture™ : Application Programming Guide*.

**Управление системой и базами данных:** Управление охватывает почти все остальные аспекты. В книге *DB2 Administration Guide* описание задач управления разбито на следующие части:

- В Разделе 2 (Том 1) *DB2 Administration Guide* обсуждаются решения при проектировании базы данных и описывается, как реализовать проект, создавая объекты DB2, загружая данные и внося изменения.
- В Разделе 3 (Том 1) *DB2 Administration Guide* описываются способы управления доступом к системе DB2 и данными внутри DB2 для контроля за разными аспектами пользования DB2 и соответствия другим требованиям защиты и контроля.
- В Разделе 4 (Том 1) *DB2 Administration Guide* описываются действия при нормальной повседневной работе и обсуждаются шаги, необходимые при подготовке к восстановлению нормального функционирования в случае ошибки.
- В Разделе 5 (Том 1) *DB2 Administration Guide* объясняется, как следить за производительностью системы DB2 и ее частей. Здесь также перечислены приемы ускорения работы отдельных частей системы.

Кроме того, дополнения *DB2 Administration Guide* содержат ценную информацию о таблицах примера DB2, поддержке национальных языков (National Language Support, NLS), написании обработчиков пользователя, интерпретации вывода трассировки DB2 и преобразовании символов для распределенных данных.

Если вы только проектируете базу данных и планируете рабочие процедуры DB2, вам необходимо руководство *DB2 Administration Guide*. Если вы хотите также реализовать ваши собственные планы, создавая объекты DB2, предоставляя привилегии, запуская задания утилит и т.п., вам также потребуются:

- *DB2 SQL Reference*, где описываются операторы SQL для создания, изменения и отбрасывания объектов, предоставления и отзыва привилегий
- *DB2 Utility Guide and Reference*, где объясняется, как запускать утилиты
- *DB2 Command Reference*, где объясняется, как выполнять команды

Если вы планируете совместное использование данных, вам будет полезна книга

*DB2 Data Sharing: Planning and Administration*, где описывается, как планировать и осуществлять совместное использование данных.

Дополнительную информацию об управлении системой и базой данных можно найти в книге *DB2 Сообщения и коды*, где перечислены сообщения и коды, выдаваемые DB2, с объяснениями и рекомендуемыми действиями.

**Диагностика:** Специалисты по диагностике ищут и устраняют ошибки в программе DB2. Они могут также рекомендовать или применить соответствующие меры. Документация для этих целей приводится в книгах *DB2 Diagnosis Guide and Reference* и *DB2 Сообщения и коды*.

## Как получить информацию о DB2

### DB2 в WWW

Не забывайте смотреть свежую информацию по DB2. Посетите домашнюю страницу DB2 в WWW. В списках новостей сообщается о последних изменениях продукта. Объявления о продуктах, пресс-релизы, сводки и технические статьи помогут вам спланировать стратегию управления базой данных.

Можно искать и просматривать публикации по DB2 в WWW, а также загружать и печатать свежие версии большинства книг по DB2. По ссылкам на другие сайты WWW можно найти дополнительную информацию о семействе DB2 и о решениях для OS/390. Посмотрите информацию о DB2 в WWW по адресу:

<http://www.software.ibm.com/data/db2/os390>

### Публикации по DB2

Публикации по DB2 для DB2 Universal Database Server for OS/390 доступны как в печатном, так и в электронном виде.

#### Формат BookManager®

Электронные книги на компакт-диске позволяют читать, искать информацию, печатать текст частями и делать пометки в книгах BookManager. При помощи соответствующего продукта BookManager READ или программ IBM Library Reader можно просматривать эти книги в средах OS/390, VM, OS/2, DOS, AIX и Windows™. Многие книги DB2 BookManager можно просматривать и в WWW.

#### Формат PDF

Многие книги DB2 доступны в формате Portable Document Format (PDF) для просмотра или печати с компакт-диска или из WWW. Установите эти книги в своей интрасети, чтобы к ним можно было обращаться в вашей организации.

#### Компакт-диски

Книги для Версии 6 DB2 Universal Database Server for OS/390, доступные на компакт-дисках:

- *DB2 UDB for OS/390 Version 6 Licensed Online Book*, LK3T-3519, содержащая *DB2 UDB for OS/390 Version 6 Diagnosis Guide and Reference* в формате BookManager, для заказа с продуктом.
- *DB2 UDB Server for OS/390 Version 6 Online and PDF Library*, SK3T-3518, собрание книг для сервера DB2 в форматах BookManager и PDF.

Периодически на последующих изданиях этих дисков книги будут обновляться.

Книги для Версии 6 DB2 UDB Server for OS/390 доступны также в следующих наборах, где записаны электронные книги для многих продуктов IBM:

- *Online Library Omnibus Edition OS/390 Collection*, SK2T-6700, на английском языке
- *IBM Online Library MVS Collection Kit*, SK88-8002, на японском языке для просмотра в операционных системах DOS и Windows.

## Обучение DB2

Программа IBM Education and Training предлагает множество очных курсов для быстрого и эффективного приобретения опыта по DB2. Занятия проводятся во многих городах мира. Информацию о курсах по вашей стране можно найти на сайте IBM Learning Services:

<http://www.ibm.com/services/learning/>

Дополнительную информацию, в том числе о графиках занятий, можно узнать у вашего представителя IBM.

Можно также организовать курсы по месту вашей работы и во время, удобное для вас. Можно даже изменить программы занятий в соответствии с вашими потребностями. В каталоге *All-in-One Education and Training Catalog* описаны возможности обучения DB2 в Соединенных Штатах. Узнать об этих курсах можно, позвонив в США по телефону 1–800–IBM–TEACH (1–800–426–8322).

## Как заказать библиотеку DB2

Можно заказать публикации по DB2 и компакт–диски через вашего представителя IBM или через местное отделение IBM. Если вы находитесь в Соединенных Штатах или в Канаде, можно сделать заказ, позвонив по бесплатному телефону:

- В США – 1–800–879–2755.
- В Канаде – 1–800–565–1234.

Чтобы заказать дополнительные копии лицензированных публикаций, выберите опцию SOFTWARE. Чтобы заказать дополнительные публикации или диски CD–ROM, выберите опции PUBLICATIONS и SLSS. Будьте готовы назвать ваш номер заказчика, номер продукта и коды возможностей или номера заказов, которые вам нужны.

---

## Сводка изменений Версии 6 DB2 UDB for OS/390

DB2 UDB for OS/390 Версия 6 представляет собой усовершенствованный вариант сервера реляционной базы данных для OS/390. Основные особенности этого выпуска – повышение объема данных и производительности утилит и запросов, упрощение обслуживания баз данных, большая мощность сетевых вычислений и совместимость семейства DB2 с богатым набором новых объектно–ориентированных функций, поддержка триггеров, большее число встроенных функций.

## Увеличение объема данных

**16–терабайтные таблицы** обеспечивают значительное увеличение табличного объема для многораздельных табличных пространств и табличных пространств больших объектов и их индексов и для однораздельных индексов.

**Пулы буферов в пространствах данных** смягчают ограничения на виртуальную память для адресного пространства ssnmDBM1, и пространства данных увеличены до максимального доступного пространства пула виртуальных буферов.

## Производительность и доступность

Усовершенствованная перебалансировка разделов позволяет перераспределять многораздельные данные, минимально затрагивая их доступность. Одна операция REORG реорганизует и перебалансирует много разделов.

Можно **динамически менять частоту контрольных точек** при помощи новой команды SET LOG и инициировать контрольные точки в любой момент, когда доступна ваша подсистема.

**Утилиты стали быстрее и удобнее, больше используют параллельный режим:**

- Ускорение резервного копирования и восстановления позволяет COPY и RECOVER обрабатывать списки объектов параллельно, а также восстанавливать индексные и табличные пространства одновременно из копий образов и из журнала.
- Параллельное построение индексов сокращает время, затраченное на операции LOAD и REORG с табличными пространствами или разделения табличных пространств с несколькими индексами; время, затраченное на операции REBUILD INDEX, также сокращается.
- Тестирование показало **сокращение общего и процессорного времени для оперативного выполнения утилиты REORG**.
- Встроенная статистика реализует набор статистических функций в утилитах, что ускоряет доступ к табличным пространствам.
- Теперь можно **задать моменты запуска REORG**, указав пороговые значения для соответствующих статистик из каталога DB2.

**Усовершенствования, повышающие производительность запросов:**

- Повышена степень параллелизма запросов для сложных запросов, таких как внешние объединения и запросы, использующие однораздельные таблицы.
- Улучшена балансировка рабочей нагрузки в Parallel Sysplex®, что сокращает время обработки единичного запроса, распределенного между активными членами DB2.
- Улучшена передача данных, что позволяет затребовать несколько блоков запросов DRDA при выполнении операций большого объема.
- Можно использовать индекс для доступа к предикатам с несвязанными подзапросами IN.
- Ускорение обработки запросов для запросов на обработку объединений.

**Дополнительные улучшения производительности и доступности:**

- Более быстрый перезапуск и восстановление с возможностью откладывать менее срочную работу во время перезапуска и более быстрым процессом применения журнала.
- Повышение гибкости с возможностью использования размеров страниц 8 и 16 Кбайт для более эффективного балансирования требований различных рабочих нагрузок и управления трафиком через раздел обеспечения взаимодействия для некоторых рабочих нагрузок.

- **Прямой доступ к строке** при помощи нового типа данных ROWID, что позволяет непосредственно обращаться повторно к строке без использования индекса или просмотра таблицы.
- **Дополнительные возможности управления выбором пути доступа** с помощью нового метода, облегчающего указание DB2 советов по оптимизации.
- **Увеличенный размер буфера вывода журнала** (от 1000 до 100000 4-килобайтных буферов), что повышает производительность чтения журнала и записи в журнал.

## **Улучшение совместного использования данных**

**Больше опций кэширования** для использования разделом обеспечения взаимодействия повышает производительность в среде совместного использования данных для некоторых прикладных программ путем записи измененных страниц непосредственно в DASD.

**Управление обслуживанием копии отображения пространства** с помощью новой опции, позволяющей не следить за изменениями страниц, что повышает производительность прикладных программ, совместно использующих данные.

## **Пользовательская производительность**

**Прогностическое ограничение ресурсов** усиливает возможности утилиты ограничения ресурсов, что помогает оценивать потребление ресурсов для запросов с большими объемами данных.

**Оценка стоимости выполнения оператора** ресурса обработки, которая требуется для оператора SQL, позволяет задать пороги ошибки и предупреждения для ограничения и решить, какие операторы требуют настройки.

**Пул буферов по умолчанию** для данных пользователя и индексов изолирует данные пользователя от системных каталогов DB2 и, отделяя их от системных данных, способствует оптимизации настройки.

**Расширен объем сведений мониторинга DB2** – в них включены операции ввода/вывода наборов данных при трассировке, как для пакетного, так и для оперативного мониторинга.

**Лучшая интеграция сообщения о задержках между DB2 и Workload Manager** позволяет DB2 уведомлять Workload Manager о текущем состоянии рабочего требования.

**Допускается больше таблиц в операторах SQL** SELECT, UPDATE, INSERT и DELETE и в производных таблицах. DB2 увеличивает допустимый предел с 15 до 225 таблиц. Число таблиц и производных таблиц в подвыборке остается прежним.

**Улучшенная совместимость семейства DB2 UDB** включает расширения SQL:

- Условие VALUES оператора INSERT, где теперь допускаются любые выражения
- Новый оператор VALUES INTO

**Упрощено управление восстановлением данных**, что позволяет иметь единственную точку восстановления и легче восстанавливать данные на удаленном узле.

**Улучшенные команды базы данных** обеспечивают расширенную поддержку для поиска символов по шаблону (\*) и позволяют фильтровать вывод на дисплей.

Можно легко **обрабатывать динамический SQL в пакетном режиме** с новой объектной формой DSNTEP2, поставляемой с DB2 for OS/390.

## Сетевые вычисления

**SQLJ**, новейшая реализация Java для среды OS/390, поддерживает встроенный SQL в языке программирования Java. SQLJ дает программам на Java преимущества высокой производительности, простоты управления и авторизации, доступных статическому SQL, и облегчает их написание.

**Поддержка DRDA® для трехчастных имен** увеличивает функциональные возможности прикладных программ, использующих трехчастные имена для удаленного доступа, и повышает производительность прикладных программ клиент–сервер.

**Усовершенствования хранимых процедур** позволяют создавать и изменять определения хранимых процедур, строить вложенные вызовы хранимых процедур и пользовательских функций, и встраивать операторы CALL в прикладные программы или динамически вызывать операторы CALL из драйверов IBM ODBC и CLI.

**Расширения DB2 ODBC** включают новые и модифицированные интерфейсы прикладного программирования (API) и новые типы данных для поддержки объектно–реляционных расширений.

**Доступ ODBC к данным каталога DB2 for OS/390** повышает производительность ваших запросов к каталогу ODBC, перенаправляя их в теневые копии таблиц каталога DB2.

**Повышение производительности для прикладных программ ODBC** сокращает количество сетевых сообщений при выполнении операторов динамического SQL.

**Усовершенствования для динамически подготавливаемых операторов SQL** включают новый специальный регистр, что позволяет явно задавать имена пользовательских типов, пользовательских функций и хранимых процедур.

**Объединение соединений DDF в пул** использует новый тип неактивного потока, который повышает производительность для большого количества входящих соединений DDF.

## Объектно–реляционные расширения и активные данные

Расширения объектов DB2 реализуют преимущества объектно–ориентированной технологии, усиливая вашу реляционную базу данных богатым набором типов данных и функций. Эти расширения дополнены мощным механизмом триггеров, что позволяет использовать в базе данных логику прикладной программы и управлять следующими новыми структурами:

- **Большие объекты (LOB)** хорошо подходят для представления больших сложных структур в таблицах DB2. Теперь вы можете эффективно использовать мультимедиа и хранить в базе такие объекты, как сложные документы, видеозаписи, изображения и аудиозаписи. Ключевые элементы поддержки больших объектов:
  - Типы данных больших объектов для хранения байтовых строк объемом до 2 Гбайт
  - Локаторы больших объектов для облегчения действий со значениями больших объектов в виде удобных для обработки кусков
  - Вспомогательные таблицы (находящиеся в табличных пространствах больших объектов) для хранения значений больших объектов
- **Пользовательские типы** (иногда называемые особыми пользовательскими типами), которые, как и встроенные типы данных, описывают данные в столбцах таблицы, содержащей экземпляры (или объекты) этих типов данных. К значениям некоторого пользовательского типа применимы только те функции и операции, которые явно определены на нем.
- **Пользовательские функции**, как и встроенные функции или операции, поддерживают действия с экземплярами пользовательского типа (и встроенными типами данных) в запросах SQL.
- **Новые и расширенные встроенные функции** увеличивают возможности языка SQL, добавляя около 100 новых встроенных функций, расширений для существующих функций и пользовательских функций примера.

**Триггеры** автоматически выполняют набор операторов SQL, когда бы заданное событие ни произошло. Эти операторы проверяют или изменяют значения в базе данных, читают и модифицируют базу данных и вызывают функции, выполняющие операции в базе данных и вне ее.

Можно использовать **модули расширения DB2** для DB2 for OS/390 для хранения и использования изображений, аудио–, видеозаписей и текстовых объектов. Эти модули автоматически воспринимают и обрабатывают информацию об объекте и поддерживают богатый набор API.

## Увеличение числа функций

Некоторые функции и возможности доступны и для пользователей Версии 6, и для пользователей Версии 5. Узнать, как получить эти возможности до перенастройки в Версию 6, можно, посетив следующий Web–сайт:

<http://www.software.ibm.com/data/db2/os390/v5apar.html>

## **Возможности DB2 for OS/390**

DB2 for OS/390 Версии 6 предлагает множество дополнительных возможностей для сервера, поставляемых автоматически, когда вы приобретаете DB2 Universal Database for OS/390:

- DB2 Management Tools Package, состоящая из следующих компонентов:
  - Центр управления DB2 UDB
  - Построитель хранимых процедур DB2
  - Программа установки DB2
  - DB2 Visual Explain
  - DB2 Estimator
- Net.Data® for OS/390

Вы можете установить и использовать следующие возможности в рамках программы “Try and Buy,” в течение 90 дней бесплатно:

- утилита управления запросами
- DB2 DataPropagator™
- Монитор производительности DB2
- Инструмент пула буферов DB2
- Инструмент управления DB2

## **Замечания по перенастройке**

Перенастройка в Версию 6 удаляет все индексы типа 1, совместно используемые данные только для чтения, пароли наборов данных, переменные хоста, используемые без двоеточия, и операции RECOVER INDEX. Перенастройка в Версию 6 может осуществляться только из подсистемы Версии 5.

---

## **Сводка изменений в этой книге**

Основные изменения в этой книге:

- Глава 3–5. Использование триггеров для работы с активными данными – новая глава, описывающая написание прикладных программ SQL, которые используют триггеры.
- Раздел 4. Использование объектно–реляционных расширений DB2 – новая глава, описывающая написание прикладных программ SQL, которые используют большие объекты, пользовательские функции и пользовательские типы.
- Глава 5–4. Планирование доступа к распределенным данным содержит информацию об использовании трехчастных имен в прикладных программах DRDA.
- Глава 7–2. Использование хранимых процедур в системах клиент–сервер содержит информацию о новом методе определения хранимых процедур.
- Приложение С, “Как запускать примеры программ DSNTIAUL, DSNTIAD и DSNTEP2” на стр. 889 – новое приложение.

---

## Раздел 2. Использование запросов SQL

<b>Глава 2–1. Получение данных</b>	19
Таблицы результатов	19
Типы данных	20
Выбор столбцов: SELECT	21
Выбор всех столбцов: SELECT *	22
Выбор некоторых столбцов: SELECT имя–столбца	23
Выбор данных DB2, не являющихся таблицей: Использование SYSNULL	23
Выбор производных столбцов: SELECT выражение	23
Удаление повторяющихся строк: DISTINCT	24
Именование столбцов результата: AS	24
Правила SQL обработки оператора SELECT	26
Выбор строк с указанием критериев поиска: WHERE	26
Выбор строк с пустыми значениями	27
Выбор строк при помощи знаков равенства и неравенства	28
Выбор значений, подобных символьной строке	29
Выбор строк, удовлетворяющих нескольким критериям	31
Использование ключевого слова BETWEEN для задания диапазона выбора	32
Использование ключевого слова IN для задания значения из списка	33
Использование функций и выражений	34
Конкатенация строк: CONCAT	34
Вычисление значений в столбце или в нескольких столбцах	34
Использование функций столбца	37
Использование скалярных функций	38
Использование пользовательских функций	45
Использование выражений CASE	46
Упорядочивание строк: ORDER BY	47
Задание имен столбцов	47
Ссылки на производные столбцы	50
Получение значений по группам: GROUP BY	51
Группировка с критериями: HAVING	51
Объединение списков значений: UNION	52
Использование UNION для исключения повторений	53
Использование UNION ALL для сохранения повторений	53
Специальные регистры	54
Поиск информации в каталоге DB2	54
Вывод списка таблиц, которые вы можете использовать	54
Вывод списка столбцов в таблице	55
<b>Глава 2–2. Работа с таблицами и изменение данных</b>	57
Работа с таблицами	57
Создание ваших собственных таблиц: CREATE TABLE	57
Создание таблиц с родительскими ключами и внешними ключами	60
Изменение таблиц с проверочными ограничениями	61
Создание таблиц с триггерами	62
Создание временных таблиц	62
Отbrasывание таблиц: DROP TABLE	64
Работа с производными таблицами	64
Определение производной таблицы: CREATE VIEW	65

Изменение данных при помощи производной таблицы . . . . .	66
Отбрасывание производных таблиц: DROP VIEW . . . . .	66
<b>Изменение данных DB2 . . . . .</b>	<b>66</b>
Вставка строки: INSERT . . . . .	66
Изменение текущих значений: UPDATE . . . . .	71
Удаление строк: DELETE . . . . .	73
<b>Глава 2–3. Объединение данных из нескольких таблиц . . . . .</b>	<b>75</b>
Внутреннее объединение . . . . .	76
Полное внешнее объединение . . . . .	78
Левое внешнее объединение . . . . .	79
Правое внешнее объединение . . . . .	79
Правила SQL для операторов, содержащих операции объединения . . . . .	80
Использование нескольких типов объединения в одном операторе SQL . . . . .	81
Использование вложенных табличных выражений и табличных пользовательских функций при объединении . . . . .	82
<b>Глава 2–4. Использование подзапросов . . . . .</b>	<b>85</b>
Обзор основных понятий . . . . .	85
Связанные и несвязанные подзапросы . . . . .	86
Подзапросы и предикаты . . . . .	86
Таблица результатов подзапроса . . . . .	87
Подвыборы для UPDATE, DELETE и INSERT . . . . .	87
Как кодировать подзапрос . . . . .	87
Элементарный предикат . . . . .	87
Уточненные предикаты: ALL, ANY и SOME . . . . .	87
Использование ключевого слова IN . . . . .	88
Использование ключевого слова EXISTS . . . . .	88
Использование связанных подзапросов . . . . .	89
Пример связанного подзапроса . . . . .	89
Использование корреляционных имен в ссылках . . . . .	90
Использование связанных подзапросов в операторе UPDATE . . . . .	91
Использование связанных подзапросов в операторе DELETE . . . . .	91
<b>Глава 2–5. Исполнение SQL с вашего терминала при помощи SPUFI . . . . .</b>	<b>93</b>
Размещение входного набора данных и использование SPUFI . . . . .	93
Изменение умолчаний SPUFI (не обязательное) . . . . .	96
Ввод операторов SQL . . . . .	99
Обработка операторов SQL . . . . .	100
Просмотр вывода . . . . .	100
Формат результатов оператора SELECT . . . . .	101
Содержимое сообщений . . . . .	102

## Глава 2–1. Получение данных

Получать данные можно при помощи оператора SQL SELECT, в котором задается таблица результатов. В этой главе описано использование операторов SELECT в интерактивном режиме для исследования реляционных данных.

Дальнейшие вопросы использования операторов SELECT смотрите в разделах “Глава 2–4. Использование подзапросов” на стр. 85, “Глава 5–4. Планирование доступа к распределенным данным” на стр. 405 и в разделе Глава 5 книги *DB2 SQL Reference*.

Примеры операторов SQL иллюстрируют понятия, обсуждаемые в данной книге. Мы советуем вам построить свои операторы SQL, подобные описанным в примерах, и выполнить их динамически при помощи SPUFI или QMF (Query Management Facility).

---

### Таблицы результатов

Данные, возвращаемые SQL, всегда имеют форму таблицы. В книгах о DB2 эта таблица называется *таблицей результатов*. Как и у таблиц, из которых извлекаются данные, у этой таблицы есть строки и столбцы. Программа выбирает эти данные по строке за раз.

**Пример:** Следующий оператор SELECT:

```
SELECT LASTNAME, FIRSTNME, PHONENO  
      FROM DSN8610.EMP  
     WHERE WORKDEPT = 'D11'  
     ORDER BY LASTNAME;
```

дает такие результаты:

LASTNAME	FIRSTNME	PHONENO
ADAMSON	BRUCE	4510
BROWN	DAVID	4501
JOHN	REBA	0672
JONES	WILLIAM	0942
LUTZ	JENNIFER	0672
PIANKA	ELIZABETH	3782
SCOUTTEN	MARILYN	1682
STERN	IRVING	6423
WALKER	JAMES	2986
YAMAMOTO	KIYOSHI	2890
YOSHIMURA	MASATOSHI	2890

Таблица результатов выводится в такой форме после того, как SPUFI проводит выборку и форматирует ее. Формат ваших результатов может быть другим.

## Типы данных

При создании таблицы DB2 вы приписываете каждому столбцу определенный тип данных. Тип данных может быть встроенным типом или пользовательским типом. В этом разделе обсуждаются встроенные типы данных. Информацию о пользовательских типах данных смотрите в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327. Тип данных столбца определяет, что с ним можно и что нельзя делать. Когда вы выполняете операции над столбцами, данные должны быть совместимы по типу с данными соответствующего столбца. Например, нельзя вставить символьные данные, например, фамилию, в столбец, для которого указан числовый тип данных. Подобным же образом нельзя сравнивать столбцы с данными несовместимых типов.

Чтобы лучше понимать идеи, излагаемые в данной главе, надо знать типы данных столбцов, которые используются в примерах. Как показано на рис. 1, типы данных делятся на четыре основные категории: строковые, даты–времени, числовые и ROWID.

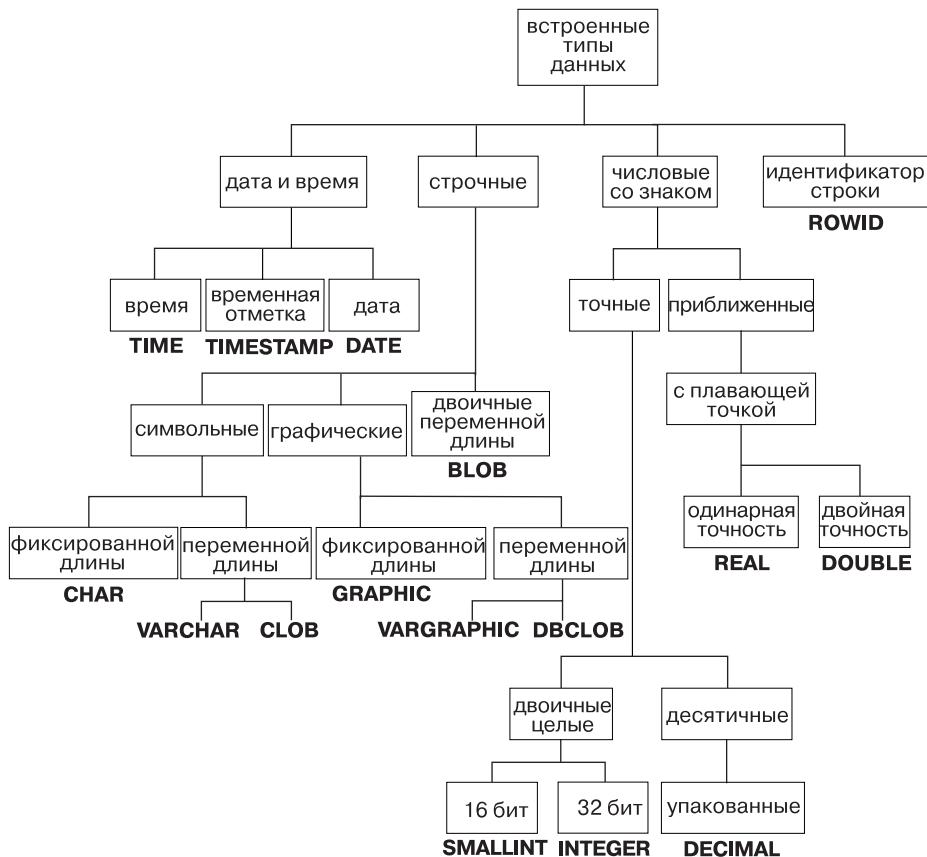


Рисунок 1. Типы данных DB2

Более подробное описание каждого типа данных смотрите в разделе Глава 3 книги *DB2 SQL Reference*.

В Табл. 1 на стр. 21 показано, операнды каких типов данных являются совместимыми (Да), а каких – нет (Нет).

Таблица 1. Совместимость типов данных для присвоения и сравнения. Д означает, что типы данных совместимы. Н означает несовместимость. Если в столбце стоит число, смотрите соответствующее примечание внизу.

Операнды	Число с Сим—										Польз. тип
	Двоичн. целое	Десят. число	плав. точкой	вольная строка	Графич. строка	Двоичная строка	Дата	Время	Врем. отметка	ID строки	
Двоичное целое	Д	Д	Д	Н	Н	Н	Н	Н	Н	Н	2
Десятичное число	Д	Д	Д	Н	Н	Н	Н	Н	Н	Н	2
Число с плавающей точкой	Д	Д	Д	Н	Н	Н	Н	Н	Н	Н	2
Символьная строка	Н	Н	Н	Д	Н	Н <sup>3</sup>	1	1	1	Н	2
Графическая строка	Н	Н	Н	Н	Д	Н	Н	Н	Н	Н	2
Двоичная строка	Н	Н	Н	Н <sup>3</sup>	Н	Д	Н	Н	Н	Н	2
Дата	Н	Н	Н	1	Н	Н	Д	Н	Н	Н	2
Время	Н	Н	Н	1	Н	Н	Н	Д	Н	Н	2
Временная— отметка	Н	Н	Н	1	Н	Н	Н	Н	Д	Н	2
ID строки	Н	Н	Н	Н	Н	Н	Н	Н	Н	Д	2
Польз. тип	2	2	2	2	2	2	2	2	2	2	Д <sup>2</sup>

#### Примечания:

- Совместимость значений даты–времени при назначении и сравнении ограничивается:
  - Значения даты–времени можно назначать столбцам и переменным с типом данных символьная строка.
  - Правильное строковое представление даты можно назначить столбцу даты или сравнить с датой.
  - Правильное строковое представление времени можно назначить столбцу времени или сравнить со временем.
  - Правильное строковое представление временной отметки можно назначить столбцу временной отметки или сравнить со временной отметкой.
- Значение пользовательского типа сравнимо только со значением того же пользовательского типа. Вообще говоря, DB2 поддерживает назначение между значением пользовательского типа и его исходного типа данных. Более подробную информацию смотрите в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.
- Никакие символьные строки, даже строки подтипа FOR BIT DATA, не сравнимы с двоичными строками.

## Выбор столбцов: SELECT

Для выбора столбцов из базы данных для таблицы результатов существует несколько возможностей. В данном разделе описаны различные методы выбора столбцов.

## Выбор всех столбцов: **SELECT \***

Для выбора данных DB2 нет необходимости знать имена столбцов. Звездочка (\*) в условии SELECT означает выбор “всех столбцов” каждой выбранной строки названной таблицы.

Следующий оператор SQL:

```
SELECT *  
  FROM DSN8610.DEPT;
```

дает такой результат:

DEPTNO	DEPTNAME	MGRNO	ADMDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	-----
B01	PLANNING	000020	A00	-----
C01	INFORMATION CENTER	000030	A00	-----
D01	DEVELOPMENT CENTER	-----	A00	-----
D11	MANUFACTURING SYSTEMS	000060	D01	-----
D21	ADMINISTRATION SYSTEMS	000070	D01	-----
E01	SUPPORT SERVICES	000050	A00	-----
E11	OPERATIONS	000090	E01	-----
E21	SOFTWARE SUPPORT	000100	E01	-----
F22	BRANCH OFFICE F2	-----	E01	-----
G22	BRANCH OFFICE G2	-----	E01	-----
H22	BRANCH OFFICE H2	-----	E01	-----
I22	BRANCH OFFICE I2	-----	E01	-----
J22	BRANCH OFFICE J2	-----	E01	-----

Оператор SELECT в примере возвращает данные всех столбцов (SELECT \*) каждой возвращаемой строки таблицы DSN8610.DEPT. Поскольку в примере не задано условие WHERE, оператор возвращает данные всех строк.

Прочерки в столбце MGRNO четвертой строки таблицы результатов означают, что соответствующее значение пусто. Значение пусто, поскольку не указан менеджер этого отдела. Пустые значения описываются в разделе “Выбор строк с пустыми значениями” на стр. 27.

SELECT \* рекомендуется использовать главным образом для динамического SQL и для определений производных таблиц. SELECT \* можно использовать в статическом SQL, однако это не рекомендуется; если вы добавите столбец к таблице, на которую ссылается SELECT \*, программа может обратиться к столбцу, для которого вы не задали принимающих переменных хоста. Дополнительную информацию о переменных хостасмотрите в разделе “Обращение к данным с помощью переменных хоста и структур хоста” на стр. 110.

Избежать этой проблемы можно, перечислив имена столбцов в статическом операторе SELECT вместо того, чтобы использовать в нем звездочку. Это позволит также видеть соответствия между принимающими переменными хоста и столбцами таблицы результатов.

## **Выбор некоторых столбцов: SELECT имя—столбца**

Выберите требуемый столбец или столбцы, перечислив их имена. Столбцы появятся в порядке перечисления, а не в порядке их следования в таблице.

Следующий оператор SQL:

```
SELECT MGRNO, DEPTNO  
      FROM DSN8610.DEPT;
```

дает такой результат:

MGRNO	DEPTNO
000010	A00
000020	B01
000030	C01
000040	D01
000050	E01
000060	D11
000070	D21
000080	E11
000090	E21
000100	F22
000110	G22
000120	H22
000130	I22
000140	J22

Оператор SELECT примера возвращает данные двух названных столбцов каждой строки таблицы DSN8610.DEPT. В одном операторе SELECT можно задать от одного до 750 столбцов.

## **Выбор данных DB2, не являющихся таблицей: Использование SYSUMMY1**

В DB2 предусмотрена таблица EBCDIC под названием SYSIBM.SYSDUMMY1, которую можно использовать для выбора данных DB2, не являющихся таблицей.

Например, если вы хотите применить встроенную функцию DB2 к переменной хоста, можно использовать следующий оператор SQL:

```
SELECT RAND(:HRAND)  
      FROM SYSIBM.SYSDUMMY1;
```

## **Выбор производных столбцов: SELECT выражение**

Можно выбрать столбцы, производные от константы, выражения или функции. Следующий оператор SQL:

```
SELECT EMPNO, (SALARY + BONUS + COMM)  
      FROM DSN8610.EMP;
```

выбирает данные всех строк таблицы DSN8610.EMP, вычисляет результат выражения и возвращает столбцы в порядке, указанном в операторе SELECT. В таблице результатов у производных столбцов, таких как (SALARY + BONUS + COMM), не будет имен. Условие AS позволяет давать имена столбцам, у которых их нет. Сведения об условии AS смотрите в разделе Именование столбцов результата: AS.

Если вы хотите упорядочить строки данных в таблице результатов, используйте условие ORDER BY, описанное в разделе “Упорядочивание строк: ORDER BY” на стр. 47.

## Удаление повторяющихся строк: DISTINCT

Ключевое слово DISTINCT означает удаление повторяющихся строк из результата, то есть данные в вашем результате будут уникальными.

Следующий оператор SELECT выводит список номеров отделов для административных отделов:

```
SELECT DISTINCT ADMRDEPT  
    FROM DSN8610.DEPT;
```

это даст следующий результат:

```
ADMRDEPT  
=====  
A00  
D01  
E01
```

Сравните результат предыдущего примера со следующим:

```
SELECT ADMRDEPT  
    FROM DSN8610.DEPT;
```

который даст такие результаты:

```
ADMRDEPT  
=====  
A00  
A00  
A00  
A00  
A00  
D01  
D01  
E01  
E01  
E01  
E01  
E01  
E01  
E01  
E01
```

Если ключевое слово DISTINCT опущено, возвращаются значения столбца ADMRDEPT каждой выбранной строки, даже если среди них есть повторяющиеся.

## Именование столбцов результата: AS

При помощи условия AS можно дать имена столбцам результатов в условии SELECT. Это особенно полезно для столбцов, определенных как выражение или функция. Синтаксис и дополнительную информацию смотрите в разделе Глава 3 книги *DB2 SQL Reference*.

В следующих примерах даны варианты использования условия AS.

**Пример 1:** Дадим выражению SALARY+BONUS+COMM имя TOTAL\_SAL.

```
SELECT SALARY+BONUS+COMM AS TOTAL_SAL  
      FROM DSN8610.EMP  
     ORDER BY TOTAL_SAL;
```

**Пример 2:** Можно дать имена столбцам результата в условии выбора оператора CREATE VIEW. Вам не требуется поддерживать список столбцов в CREATE VIEW, поскольку ключевое слово AS позволяет дать имена производным столбцам. Столбцы в производной таблице EMP\_SAL названы EMPNO и TOTAL\_SAL.

```
CREATE VIEW EMP_SAL AS  
  SELECT EMPNO, SALARY+BONUS+COMM AS TOTAL_SAL  
    FROM DSN8610.EMP;
```

**Пример 3:** При помощи условия AS можно дать совпадающие имена соответствующим столбцам при объединении. Третий столбец результата объединения двух таблиц получает имя TOTAL\_VALUE, несмотря на то, что он содержит данные из столбцов базы данных с различными именами. В следующем операторе SQL:

```
SELECT 'On hand' AS STATUS, PARTNO, QOH * COST AS TOTAL_VALUE  
      FROM PART_ON_HAND  
UNION ALL  
SELECT 'Ordered' AS STATUS, PARTNO, QORDER * COST AS TOTAL_VALUE  
      FROM ORDER_PART  
ORDER BY PARTNO, TOTAL_VALUE;
```

при объединении двух операторов SELECT считается, что имя столбца первой таблицы результатов совпадает с именем столбца второй таблицы результатов. Столбец STATUS и производный столбец TOTAL\_VALUE в первой и во второй таблице результатов имеют совпадающие имена, в результате при объединении двух таблиц результатов будет получено следующее:

STATUS	PARTNO	TOTAL_VALUE
On hand	00557	345.60
Ordered	00557	150.50
:		

Сведения об операции объединения смотрите в разделе “Объединение списков значений: UNION” на стр. 52.

**Пример 4:** Условие AS можно использовать в условии FROM, чтобы дать имя производному столбцу, на который вы хотите сослаться в условии GROUP BY. Использование условия AS в первом условии SELECT приведет к ошибке, поскольку имена, назначаемые в условии AS, еще не будут существовать к моменту выполнения GROUP BY. Однако условие AS можно использовать в подвыборе внешнего условия GROUP BY, поскольку подвыбор находится на более низком уровне по отношению к условию GROUP BY, которое ссылается на заданное имя. Следующий оператор SQL:

```
SELECT HIREYEAR, AVG(SALARY)  
      FROM (SELECT YEAR(HIREDATE) AS HIREYEAR, SALARY  
            FROM DSN8610.EMP) AS NEWEMP  
     GROUP BY HIREYEAR;
```

дает имя HIREYEAR табличному выражению в скобках, что позволяет использовать это имя для столбца результата в условии GROUP BY.

## Правила SQL обработки оператора SELECT

Правила SQL определяют, что оператор SELECT должен генерировать набор строк, который получался бы при вычислении условий в этом операторе в следующем порядке:

- FROM
- WHERE
- GROUP BY
- HAVING
- SELECT

Внутренняя обработка условий в DB2 не обязательно производится именно в этом порядке, однако полученные результаты всегда будут совпадать с результатами вычислений в описанном порядке.

Условия подвыбора обрабатываются от самого внутреннего подвыбора к внешнему.

Условие ORDER BY может появляться только в самом внешнем операторе SELECT.

Если вы используете условие AS для задания имени в самом внешнем условии SELECT, на это имя может ссылаться только условие ORDER BY. Если вы используете условие AS в подвыборе, на заданное имя можно ссылаться в любом выражении вне подвыбора.

Например, следующий оператор SQL недопустим:

```
SELECT EMPNO, (SALARY + BONUS + COMM) AS TOTAL_SAL  
      FROM DSN8610.EMP  
     WHERE TOTAL_SAL > 50000;
```

Однако допустим такой оператор SQL:

```
SELECT EMPNO, (SALARY + BONUS + COMM) AS TOTAL_SAL  
      FROM DSN8610.EMP  
     ORDER BY TOTAL_SAL;
```

---

## Выбор строк с указанием критериев поиска: WHERE

Условие WHERE применяется для выбора строк, соответствующих определенным критериям. Условие WHERE задает *критерии поиска*. Критерий поиска состоит из одного или нескольких *предикатов*. Предикат задает проверку, которую DB2 должна выполнить для каждой строки таблицы.

Когда DB2 вычисляет значение предиката для строки, она может получить результаты "истина", "ложь" или "неизвестно". Результат "неизвестно" может быть получен, только если один из операндов имеет пустое значение.

Если критерий поиска содержит столбец пользовательского типа, значение, с которым сравнивается этот столбец, должно иметь тот же самый пользовательский тип; если это не так, надо выполнить преобразование значения к этому пользовательскому типу. Дополнительную информацию

смотрите в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

В следующих разделах приводятся примеры различных операций сравнения, которые можно использовать в предикатах условия WHERE. Список операций сравнения приводится в следующей таблице.

Таблица 2. Операции сравнения, применяемые в критериях поиска

Тип сравнения	Задается...	Пример
Пусто	IS NULL	PHONENO IS NULL
Равно	=	DEPTNO = 'X01'
Не равно	<>	DEPTNO <> 'X01'
Меньше	<	AVG(SALARY) < 30000
Меньше или равно	<=	AGE <= 25
Не меньше	>=	AGE >= 21
Больше	>	SALARY > 2000
Больше или равно	>=	SALARY >= 5000
Не больше	<=	SALARY <= 5000
Подобно другому значению	LIKE	NAME LIKE '%SMITH%' или STATUS LIKE 'N_'
По крайней мере одно из двух	OR	HIREDATE < '1965-01-01' OR SALARY < 16000
И одно, и другое	AND	HIREDATE < '1965-01-01' AND SALARY < 16000
Между двумя значениями	BETWEEN	SALARY BETWEEN 20000 AND 40000
Равно одному из набора	IN (X, Y, Z)	DEPTNO IN ('B01', 'C01', 'D01')

Можно также искать строки, *не* удовлетворяющие какому-либо из перечисленных критериев; для этого перед критерием надо поставить ключевое слово NOT. Дополнительную информацию об использовании ключевого слова NOT смотрите в разделе “Использование ключевого слова NOT с операциями сравнения” на стр. 28.

## Выбор строк с пустыми значениями

Пустое значение (*null*) означает отсутствие значения в столбце данной строки. Пустое значение отличается от *нулевого* значения и от значения *из одних пробелов*.

Условие WHERE можно использовать для получения строк, которые содержат пустое значение в каком-либо столбце. Для этого надо задать:

WHERE *имя-столбца* IS NULL

При помощи предиката можно также отсеять пустые значения. Для этого надо задать:

WHERE *имя-столбца* IS NOT NULL

## Выбор строк при помощи знаков равенства и неравенства

Для задания критериев поиска в условии WHERE можно использовать знаки равенства (=), неравенства и ключевое слово NOT.

### Использование знака равенства (=)

Знак равенства (=) можно использовать для выбора строк, в которых в заданном столбце содержится определенное значение. Например, чтобы выбрать только строки, где задан номер отдела A00, используйте в операторе SQL условие WHERE WORKDEPT = 'A00':

```
SELECT FIRSTNAME, LASTNAME  
      FROM DSN8610.EMP  
     WHERE WORKDEPT = 'A00';
```

Этот оператор возвратит имя и фамилию каждого сотрудника отдела A00.

### Использование неравенств

Для задания критериев поиска можно использовать следующие неравенства:

- не равно (<>)
- меньше (<)
- меньше или равно (<=)
- больше (>)
- больше или равно (>=).

Чтобы выбрать всех сотрудников, нанятых до 1 января 1960 года, можно задать:

```
SELECT HIREDATE, FIRSTNAME, LASTNAME  
      FROM DSN8610.EMP  
     WHERE HIREDATE < '1960-01-01';
```

В этом примере возвращается дата приема на работу, имя и фамилия каждого сотрудника, нанятого до 1960 года.

При сравнении строк DB2 использует последовательность сортировки, соответствующую схеме кодировки для таблицы. Это значит, что если для таблицы задан CCSID EBCDIC, DB2 будет использовать последовательность сортировки EBCDIC. Если для таблицы задан CCSID ASCII, DB2 будет использовать последовательность сортировки ASCII. Последовательности сортировки EBCDIC и ASCII различаются. Например, в EBCDIC латинские буквы идут раньше цифр, а в ASCII – позже цифр.

### Использование ключевого слова NOT с операциями сравнения

Ключевое слово NOT используется для выбора всех строк, *кроме* строк, соответствующих критерию поиска. Ключевое слово NOT должно стоять перед критерием поиска. Чтобы выбрать всех руководителей с выплатами *не* больше 30000 долларов, задайте:

```
SELECT WORKDEPT, EMPNO  
      FROM DSN8610.EMP  
     WHERE NOT (SALARY + BONUS + COMM) > 30000 AND JOB = 'MANAGER'  
          ORDER BY WORKDEPT;
```

Следующие условия WHERE эквивалентны:

Таблица 3. Эквивалентные условия WHERE. Использующие ключевое слово NOT с операциями сравнения и использующие только операции сравнения.

С использованием NOT	Эквивалентное условие
WHERE NOT DEPTNO = 'A00'	WHERE DEPTNO <> 'A00'
WHERE NOT DEPTNO < 'A00'	WHERE DEPTNO >= 'A00'
WHERE NOT DEPTNO > 'A00'	WHERE DEPTNO <= 'A00'
WHERE NOT DEPTNO <> 'A00'	WHERE DEPTNO = 'A00'
WHERE NOT DEPTNO <= 'A00'	WHERE DEPTNO > 'A00'
WHERE NOT DEPTNO >= 'A00'	WHERE DEPTNO < 'A00'

Ключевое слово NOT нельзя использовать непосредственно у операции сравнения. Следующее условие WHERE ошибочно:

**Недопустимо:** WHERE DEPT NOT = 'A00'

Другие ключевые слова SQL можно использовать с NOT: допустимы записи NOT LIKE, NOT IN и NOT BETWEEN. Например, следующие два условия эквивалентны:

WHERE MGRNO NOT IN ('000010', '000020')

WHERE NOT MGRNO IN ('000010', '000020')

## Выбор значений, подобных символьной строке

Ключевое слово LIKE используется для задания символьной строки, которой должны быть подобны значения столбца в требуемых строках:

- Знак процента (%) используется для задания символьной строки (из нуля или большего числа символов)
- Знак подчеркивания (\_) означает один любой символ.
- Предикат LIKE можно использовать только с символьными или с графическими данными, но не с числовыми данными, данными типов дата–время или ROWID .

## Выбор значений, подобных строке с неопределенными символами

Знак процента (%) означает “любую строку, в частности, пустую строку.”

Следующий оператор SQL выбирает данные из каждой строки, где инициалы сотрудника – "E H".

```
SELECT FIRSTNAME, LASTNAME, WORKDEPT  
      FROM DSN8610.EMP  
     WHERE FIRSTNAME LIKE 'E%' AND LASTNAME LIKE 'H%';
```

Следующий оператор SQL выбирает данные из каждой строки таблицы отделов, где название отдела содержит в любом месте подстроку “CENTER.”

```
SELECT DEPTNO, DEPTNAME  
      FROM DSN8610.DEPT  
     WHERE DEPTNAME LIKE '%CENTER%';
```

Предположим, что столбец DEPTNO – столбец трехсимвольных строк фиксированной длины. Для него задание следующего критерия поиска:

```
...WHERE DEPTNO LIKE 'E%1';
```

вернет все строки, где номер отдела начинается с E и заканчивается на 1. Если номер какого-либо отдела – E1, третий символ будет пробелом, что не соответствует критерию поиска. Если же вы определили столбец DEPTNO как трехсимвольный столбец *переменной длины*, отдел E1 будет соответствовать условию поиска: в столбцы переменной длины можно записать значение любой длины вплоть до максимальной заданной при его создании.

Следующий оператор SQL выбирает данные из каждой строки таблицы отделов, где название отдела начинается с E и содержит в любом месте 1.

```
SELECT DEPTNO, DEPTNAME  
  FROM DSN8610.DEPT  
 WHERE DEPTNO LIKE 'E%1%';
```

### **Выбор значений, подобных строке с одним неопределенным символом**

Символ подчеркивания (\_) означает “один любой символ.” В следующем операторе SQL,

```
SELECT DEPTNO, DEPTNAME  
  FROM DSN8610.DEPT  
 WHERE DEPTNO LIKE 'E_1';
```

'E\_1' означает “E, за ним один любой символ, за ним 1.” (Внимание: '\_’ – подчеркивание, а не знак дефиса.) 'E\_1' выберет только трехсимвольные номера отделов, которые начинаются с E и заканчиваются 1; при этом 'E1:' выбран не будет.

Следующий оператор SQL выбирает данные из каждой строки по четырехзначному телефонному номеру, первые три цифры в котором должны быть 378.

```
SELECT LASTNAME, PHONENO  
  FROM DSN8610.EMP  
 WHERE PHONENO LIKE '378_';
```

### **Выбор значений, подобных строке с символами % или \_**

Если символы % или \_ надо задать как литеральную часть вашей строки, используйте условие ESCAPE и укажите в предикате LIKE перед % или \_ – заданный символ. В следующем примере ESCAPE '+' указывает, что плюс – особый символ в критерии поиска. Например:

```
...WHERE C1 LIKE 'AAAA+%BBB%' ESCAPE '+'
```

ищет строку, начинающуюся с AAAA%BBB. Символ плюс (+) перед первым % указывает, что % должен восприниматься как часть строки поиска. Второй %, перед которым не стоит плюс, задает, что за строкой может идти любое число (в частности, ноль) любых символов. В данном примере '++' в строке позволило бы задать один плюс (+) как часть строки поиска.

## Выбор строк, удовлетворяющих нескольким критериям

Для комбинирования критериев поиска можно использовать ключевые слова AND, OR и NOT. AND задает, что при поиске должны быть удовлетворены оба критерия. OR задает, что при поиске должен быть удовлетворен хотя бы один критерий.

**Пример 1:** В данном примере возвращается номер сотрудника, дата приема на работу и зарплата для каждого сотрудника, нанятого до 1965 года и получающего зарплату меньше 16000 долларов в год.

```
SELECT EMPNO, HIREDATE, SALARY  
      FROM DSN8610.EMP  
 WHERE HIREDATE < '1965-01-01' AND SALARY < 16000;
```

**Пример 2:** В данном примере возвращается номер сотрудника, дата приема на работу и зарплата для каждого сотрудника, кто либо нанят до 1965 года, либо получает зарплату меньше 16000 в год, либо и то, и другое.

```
SELECT EMPNO, HIREDATE, SALARY  
      FROM DSN8610.EMP  
 WHERE HIREDATE < '1965-01-01' OR SALARY < 16000;
```

### Использование скобок с AND и OR

Если вы задаете более двух критериев с операциями AND или OR, можно применить скобки, чтобы указать порядок, в котором DB2 должна производить вычисление. От положения скобок может зависеть смысл условия WHERE.

**Пример 1:** Чтобы выбрать строки для тех сотрудников, которые соответствуют по крайней мере одному из следующих критериев:

- Принятые на работу до 1965 года И имеющие зарплату меньше 20000 долларов.
- С уровнем образования меньше 13.

Надо задать следующее условие WHERE; оно вернет строки для сотрудников 000290, 000310 и 200310:

```
SELECT EMPNO  
      FROM DSN8610.EMP  
 WHERE (HIREDATE < '1965-01-01' AND SALARY < 20000) OR (EDLEVEL < 13);
```

**Пример 2:** Чтобы выбрать строки для тех сотрудников, которые соответствуют обоим следующим критериям:

- Принятые на работу до 1965 года
- Зарплата сотрудника меньше 20000 ИЛИ уровень образования меньше 13.

Надо задать следующее условие WHERE; оно вернет строки для сотрудников 000310 и 200310:

```
SELECT EMPNO  
      FROM DSN8610.EMP  
 WHERE HIREDATE < '1965-01-01' AND (SALARY < 20000 OR EDLEVEL < 13);
```

**Пример 3:** Показанный ниже оператор SQL вернет номера сотрудников, удовлетворяющих по крайней мере одному из следующих критериев:

- Принятые на работу до 1965 года и получающие зарплату меньше 20000 долларов.

- Принятые на работу после 1 января 1965 года и получающие зарплату больше 40000 долларов.

Условие WHERE вернет строки для сотрудников 000050, 000310 и 200310.

```
SELECT EMPNO
      FROM DSN8610.EMP
     WHERE (HIREDATE < '1965-01-01' AND SALARY < 20000)
       OR (HIREDATE > '1965-01-01' AND SALARY > 40000);
```

### **Использование NOT с AND и OR**

При использовании ключевого слова NOT с AND и OR важно учитывать положение скобок.

**Пример 1:** В данном примере NOT применяется только к первому критерию поиска ( $\text{SALARY} \geq 50000$ ):

```
SELECT EMPNO, EDLEVEL, JOB
      FROM DSN8610.EMP
     WHERE NOT (SALARY >= 50000) AND (EDLEVEL < 18);
```

Этот оператор SQL возвращает номер сотрудника, уровень образования и название должности для каждого сотрудника, удовлетворяющего *обоим* следующим критериям:

- Зарплата сотрудника меньше 50000 долларов.
- Уровень образования меньше 18.

**Пример 2:** Чтобы применить отрицание к набору предикатов, возьмите весь этот набор в скобки и поставьте ключевое слово NOT перед скобками.

```
SELECT EMPNO, EDLEVEL, JOB
      FROM DSN8610.EMP
     WHERE NOT (SALARY >= 50000 AND EDLEVEL >= 18);
```

Этот оператор SQL возвращает номер сотрудника, уровень образования и название должности для каждого сотрудника, удовлетворяющего *по крайней мере одному из* следующих критериев:

- Зарплата сотрудника меньше 50000 долларов.
- Уровень образования меньше 18.

### **Использование ключевого слова BETWEEN для задания диапазона выбора**

Ключевое слово BETWEEN используется для выбора строк, в которых значение столбца лежит между заданными пределами.

Первым задавайте нижний предел предиката BETWEEN, затем задавайте верхний предел. Пределы воспринимаются как *нестрогие*. Предположим, вы задали

`WHERE имя-столбца BETWEEN 6 AND 8`

где значения в столбце *имя-столбца* – целые. DB2 выберет все строки, в которых в столбце *имя-столбца* стоит 6, 7 или 8. Если вы зададите первым большее число, а вторым – меньшее (например, BETWEEN 8 AND 6), предикат всегда будет иметь значение "ложь".

**Пример 1:**

```
SELECT DEPTNO, MGRNO
  FROM DSN8610.DEPT
 WHERE DEPTNO BETWEEN 'C00' AND 'D31';
```

В этом примере возвращается номер отдела и номер руководителя для каждого отдела с номерами от C00 до D31.

#### **Пример 2:**

```
SELECT EMPNO, SALARY
  FROM DSN8610.EMP
 WHERE SALARY NOT BETWEEN 40000 AND 50000;
```

В этом примере возвращаются номера сотрудников и их зарплата для всех сотрудников, чья зарплата меньше 40000 долларов или больше 50000 долларов. Предикат BETWEEN можно использовать для задания допустимой ошибки при вычислениях с плавающей точкой. Числа с плавающей точкой – это приближения действительных чисел. Поэтому результат сравнения может быть вычислен как "ложь", даже если в столбцы COL1 и COL2 были записаны одинаковые числа:

```
...WHERE COL1 = COL2
```

В данном примере переменная хоста FUZZ используется как допустимая ошибка:

```
...WHERE COL1 BETWEEN (COL2 - :FUZZ) AND (COL2 + :FUZZ)
```

## **Использование ключевого слова IN для задания значения из списка**

Предикат IN используется для выбора всех строк, где значения в столбце совпадают с одним из перечисленных в списке.

В списке значений после ключевого слова IN порядок переменных несуществен и не влияет на порядок строк в результате. Весь список заключается в скобки, а отдельные элементы разделяются запятыми; пробелы не обязательны.

```
SELECT DEPTNO, MGRNO
  FROM DSN8610.DEPT
 WHERE DEPTNO IN ('B01', 'C01', 'D01');
```

В этом примере возвращается номер отдела и номер руководителя для отделов B01, C01 и D01.

Использование предиката IN дает те же результаты, что и более длинный набор критериев, разделенных ключевыми словами OR. Например, условие WHERE в этом операторе SELECT можно было бы записать так:

```
WHERE DEPTNO = 'B01' OR DEPTNO = 'C01' OR DEPTNO = 'D01'
```

Однако предикат IN экономит время кодирования и проще для понимания.

Следующий оператор SQL находит ошибочные значения в столбце SEX.

```
SELECT EMPNO, SEX
  FROM DSN8610.EMP
 WHERE SEX NOT IN ('F', 'M');
```

---

## Использование функций и выражений

Чтобы управлять видом и значениями строк и столбцов в таблице результатов, можно использовать операции и функции. В этом разделе описываются некоторые из таких операций и функций.

### Конкатенация строк: CONCAT

При помощи ключевого слова CONCAT можно выполнить конкатенацию нескольких строк. CONCAT можно использовать с любым строковым выражением. Например,

```
SELECT LASTNAME CONCAT ',' CONCAT FIRSTNAME  
      FROM DSN8610.EMP;
```

выдаст фамилию, запятую и имя из каждой строки результата.

Дополнительную информацию о выражениях конкатенациисмотрите в разделе Глава 3 книги *DB2 SQL Reference*.

### Вычисление значений в столбце или в нескольких столбцах

С числовыми данными или с данными типа дата–время можно выполнять вычисления. Подробную информацию о вычислениях с данными типов дата, время и временная отметка смотрите в разделе Глава 3 книги *DB2 SQL Reference*.

#### Использование числовых данных

Для выбранных строк можно, наряду со значениями столбцов, вывести и вычисленные значения.

Например, если вы напишете следующий оператор SQL:

```
SELECT EMPNO,  
       SALARY / 12 AS MONTHLY_SAL,  
       SALARY / 52 AS WEEKLY_SAL  
      FROM DSN8610.EMP  
     WHERE WORKDEPT = 'A00';
```

получатся следующие результаты

EMPNO	MONTHLY_SAL	WEEKLY_SAL
000010	4395.83333333	1014.42307692
000110	3875.00000000	894.23076923
000120	2437.50000000	562.50000000
200010	3875.00000000	894.23076923
200120	2437.50000000	562.50000000

Оператор SELECT в примере выводит месячную и недельную зарплату для сотрудников отдела A00.

Чтобы получить номер отдела, номер сотрудника, оклад, премию и комиссионные для тех сотрудников, у которых премия плюс комиссионные больше 5000, введите:

```
SELECT WORKDEPT, EMPNO, SALARY, BONUS, COMM  
      FROM DSN8610.EMP  
     WHERE BONUS + COMM > 5000;
```

будет получен следующий результат:

WORKDEPT	EMPNO	SALARY	BONUS	COMM
A00	000010	52750.00	1000.00	4220.00
A00	200010	46500.00	1000.00	4220.00

## **Выбор 15– или 31–значной точности для десятичных чисел**

DB2 допускает использование двух наборов правил для определения точности и масштаба результатов при действиях с десятичными числами.

- Правила DEC15 допускают максимальную точность результата действия 15 цифр. Эти правила применяются, если оба операнда имеют точность не больше 15 цифр, если не имеет место одно из условий, требующих применения правил DEC31.
- Правила DEC31 допускают максимальную точность результата действия 31 цифра. Эти правила применяются, если имеет место хотя бы одно из следующих условий:
  - Какой–либо операнд в действии имеет точность выше 15 цифр.
  - Операция входи в оператор динамического SQL и имеет место хотя бы одно из следующих условий:
    - Текущее значение специального регистра CURRENT PRECISION – DEC31.
    - Для опции установки DECIMAL ARITHMETIC на панели DSNTIPF выбрано значение DEC31 или 31, опция установки USE FOR DYNAMICRULES на панели DSNTIPF имеет значение YES, и программа не установила значение CURRENT PRECISION.
    - Для оператора SQL задано поведение связывания, определения или вызова, оператор находится в программе, препомпилированной с опцией DEC(31), опция установки USE FOR DYNAMICRULES на панели DSNTIPF имеет значение NO, и программа не установила значение CURRENT PRECISION. Объяснение поведений связывания, определения и вызова смотрите в разделе “Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES” на стр. 457.
    - Операция выполняется во встроенном (статическом) операторе SQL, который препомпилировался с опцией DEC(31) или же со значением этой опции по умолчанию при опции установки DECIMAL ARITHMETIC – DEC31 или 31. (Информацию о препомпилияции и список опций препомпилияции смотрите в разделе “Шаг 1: Препомпилияция программы” на стр. 438.)

Выбор между DEC15 и DEC31 определяется вашими задачами. Выбирайте:

- DEC15, чтобы избежать ошибок, возникающих, когда вычисленный масштаб результата простой операции умножения или деления оказывается меньше 0. Эта ошибка может встретиться при любом наборе правил, однако для правил DEC31 это происходит чаще.
- DEC31, чтобы избежать переполнения, или при работе с данными, точность которых больше 15.

**Что можно сделать:** Для операторов статического SQL простейший способ переопределить использование правил DEC31 – задать опцию прекомпилятора DEC(15). Это снизит вероятность ошибок для встроенных операторов в программе.

Если для операторов динамического SQL задано поведение связывания, определения или вызова и значение опции установки USE FOR DYNAMICRULES на панели DSNTIPF – NO, чтобы переопределить правила DEC31, используйте опцию прекомпиляции DEC(15).

Для динамического оператора или для отдельного статического оператора используйте скалярную функцию DECIMAL для задания значений точности и масштабирования результатов так, чтобы это не приводило к ошибкам.

Для динамического оператора перед его выполнением задайте для специального регистра CURRENT PRECISION значение DEC15.

### Использование данных даты–времени

Если вы используете даты, назначьте для всех столбцов, содержащих даты, типы данных даты–времени. Это не только позволит вам выполнять дополнительные операции над таблицами, но и поможет избежать некоторых проблем:

Предположим, при создании таблицы YEMP (она описана в разделе “Создание новой таблицы отделов” на стр. 59), вы назначили для столбца BIRTHDATE тип данных DECIMAL(8,0), а затем заполнили его датами в виде ггггммдд. Затем вы хотите найти людей не моложе 27 лет при помощи такого запроса:

```
SELECT EMPNO, FIRSTNAME, LASTNAME  
      FROM YEMP  
     WHERE YEAR(CURRENT DATE - BIRTHDATE) > 26;
```

Предположим, что к моменту выполнения запроса кому–либо из таблицы YEMP было 27 лет, 0 месяцев и 29 дней – в таком случае он не попадет в таблицу результатов. Вот почему это происходит:

Если тип данных в столбце – DECIMAL(8,0) DB2 будет рассматривать BIRTHDATE как продолжительность, и тем самым вычислит CURRENT DATE – BIRTHDATE как дату. (*Продолжительность* – это число, соответствующее интервалу времени. Дополнительную информацию о операндах типа дата–время и продолжительность смотрите в разделе Глава 3 книги *DB2 SQL Reference*.) Результат вычислений (27/00/29) как дата недопустим, он будет преобразован в 26/12/29. В результате этого ошибочного преобразования DB2 будет считать, что такому человеку 26, а не 27 лет. Чтобы получить правильный результат, надо при создании таблицы задать для BIRTHDATE тип DATE, и результат вычисления CURRENT DATE – BIRTHDATE станет продолжительностью.

Если данные дат хранились в столбцах с другими типами (не DATE и не TIMESTAMP), можно преобразовать их при помощи скалярных функций. Несколько вариантов преобразования показаны в следующих примерах:

- Допустим, данные хранились в виде ггггммдд в столбце C2 типа DECIMAL(8,0); тогда можно использовать:

- Функцию DIGITS для преобразования числового значения в символьный формат
- Функцию SUBSTR для разделения строки на год, месяц и день
- Операцию CONCAT для объединения фрагментов даты в формате ISO (с дефисами)
- Функцию DATE для перевода полученной символьной строки ('*гггг-мм-дд*') в формат даты DB2.

Например:

```
DATE(SUBSTR(DIGITS(C2),1,4) CONCAT
      '-'           CONCAT
      SUBSTR(DIGITS(C2),5,2) CONCAT
      '-'           CONCAT
      SUBSTR(DIGITS(C2),7,2))
```

- Для данных, хранящихся в виде *ггггннн* (где ннн – номер дня в году) в столбце C3 типа DECIMAL(7,0), используйте:
    - Функцию DIGITS для преобразования числового значения в символьный формат
    - Функцию DATE для перевода полученной символьной строки ('*ггггннн*') в формат даты DB2.
- DATE(DIGITS(C3))
- Для данных, хранящихся в виде *гннн* (где ннн – номер дня в году) в столбце C4 типа DECIMAL(5,0), используйте:
    - Функцию DIGITS для преобразования числового значения в символьный формат
    - Символьную константу '19' как первые две цифры года
    - Операцию CONCAT для объединения фрагментов даты в формате ISO
    - Функцию DATE для перевода полученной символьной строки ('19гннн') в формат даты DB2.
- DATE('19' CONCAT DIGITS(C4))

## Использование функций столбца

Функция столбца вычисляет одно значение для группы строк. Функции столбца SQL вычисляют значения по всему столбцу данных. Для вычисления используются только значения из выбранных строк (то есть тех строк, которые удовлетворяют условию WHERE).

Имеются следующие функции столбца:

<b>SUM</b>	Возвращает сумму значений.
<b>MIN</b>	Возвращает минимальное значение.
<b>AVG</b>	Возвращает среднее значение.
<b>MAX</b>	Возвращает максимальное значение.
<b>COUNT</b>	Возвращает число выбранных строк.
<b>STDDEV</b>	Возвращает среднеквадратическое отклонение значений в столбце.
<b>VARIANCE</b>	Возвращает вариацию значений в столбце.

Следующий оператор SQL вычисляет для отдела D11 сумму окладов сотрудников, минимальный оклад, средний оклад, среднеквадратическое отклонение, максимальный оклад и число сотрудников отдела:

```
SELECT SUM(SALARY) AS SUMSAL,
       MIN(SALARY) AS MINSAL,
       AVG(SALARY) AS AVGSAL,
       STDDEV(SALARY) AS SDSAL,
       MAX(SALARY) AS MAXSAL,
       COUNT(*) AS CNTSAL
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'D11';
```

Будет получен следующий результат:

SUMSAL	MINSAL	AVGSAL	SDSAL	MAXSAL	CNTSAL
=====	=====	=====	=====	=====	=====
276620.00	18270.00	25147.27272727	+0.4198694799164255E+04	32250.00	11

С функциями SUM, AVG и COUNT можно использовать ключевое слово DISTINCT. DISTINCT означает, что заданная функция применяется только к уникальным значениям столбца. Применение DISTINCT с функциями MAX и MIN не оказывает влияния на результат и не рекомендуется.

Функции SUM и AVG можно применять только к числам. Функции MIN, MAX и COUNT можно использовать для любых типов данных.

Следующий оператор SQL подсчитывает число сотрудников в таблице.

```
SELECT COUNT(*)
      FROM DSN8610.EMP;
```

Следующий оператор SQL вычисляет средний уровень образования в группе отделов.

```
SELECT AVG(EDLEVEL)
      FROM DSN8610.EMP
     WHERE WORKDEPT LIKE '_0_';
```

Следующий оператор SQL вычисляет число различных должностей в таблице DSN8610.EMP.

```
SELECT COUNT(DISTINCT JOB)
      FROM DSN8610.EMP;
```

## Использование скалярных функций

Скалярная функция также дает одно значение, однако в отличие от функции столбца аргумент скалярной функции – это единственное значение.

Следующий оператор SQL возвращает год приема на работу каждого сотрудника некоторого отдела:

```
SELECT YEAR(HIREDATE) AS HIYEAR
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'A00';
```

это даст следующие результаты:

```

HIREYEAR
=====
1972
1965
1965
1958
1963

```

Скалярная функция YEAR дает одно скалярное значение для каждой строки DSN8610.EMP, которая удовлетворяет критерию поиска. В этом примере критерию поиска удовлетворяют пять строк, и функция YEAR даст пять скалярных значений.

В Табл. 4 перечислены скалярные функции, которые вы можете использовать. Полное описание этих функций смотрите в разделе Глава 4 книги *DB2 SQL Reference*.

*Таблица 4 (Стр. 1 из 5). Скалярные функции*

Скалярная функция	Возвращает...	Пример
ABS или ABSVAL	абсолютную величину своего аргумента.	ABS(DIFFERENCE)
ACOS	арккосинус аргумента в радианах.	ACOS(COSOFANGLE)
ASIN	арксинус аргумента в радианах.	ASIN(SINOFANGLE)
ATAN	арктангенс аргумента в радианах.	ATAN(TANOFANGLE)
ATANH	гиперболический арктангенс аргумента в радианах.	ATANH(TANHOFANGLE)
ATAN2	гиперболический арктангенс, где два аргумента соответствуют координатам x и y точки прямоугольной гиперболы.	ATAN2(XCOORD,YCOORD)
BLOB	представление первого аргумента в виде двоичного большого объекта.	BLOB('This is a BLOB')
CEIL или CEILING	наименьшее целое число, большее или равное аргументу.	CEIL(SALARY)
CHAR	представление первого аргумента в виде строки.	CHAR(HIREDATE)
CLOB	представление первого аргумента в виде символьного большого объекта.	CLOB('This is a CLOB')
CONCAT	катенация двух аргументов.	CONCAT(FIRSTNME, LASTNAME)
COS	косинус аргумента, который воспринимается как угол в радианах.	COS(ANGLE)
COSH	гиперболический косинус аргумента, который воспринимается как угол в радианах.	COSH(ANGLE)
DATE	дата, соответствующая аргументу.	DATE('1989-03-02')
DAY	часть аргумента, соответствующая дням.	DAY(DATE1 - DATE2)
DAYOFMONTH	часть аргумента, соответствующая дням.	DAYOFMONTH(DATE1)
DAYOFWEEK	целое от 1 до 7, соответствующее дню недели даты своего аргумента.	DAYOFWEEK(HIREDATE)

Таблица 4 (Стр. 2 из 5). Скалярные функции

Скалярная функция	Возвращает...	Пример
DAYOFYEAR	целое от 1 до 366, соответствующее дню года даты своего аргумента.	DAYOFYEAR(HIREDATE)
DAYS	представление аргумента в виде целого числа.	DAY('1990-01-08') – DAY(HIREDATE) + 1
DEGREES	аргумент, преобразованный из радианов в градусы.	DEGREES(ANGLE)
DBCLOB	представление первого аргумента в виде большого объекта из двухбайтных символов.	DBCLOB(GRAPHCOL)
DECIMAL	представление первого аргумента в десятичном виде.	DECIMAL(AVG(SALARY), 8,2)
DIGITS	представление аргумента в виде строки символов.	DIGITS(COLUMNX)
DOUBLE	представление аргумента в виде числа с плавающей точкой двойной точности.	DOUBLE(SALARY)
EXP	е в степени, заданной аргументом.	EXP(DOUBLEVAL)
FLOAT	представление аргумента в виде числа с плавающей точкой.	FLOAT(SALARY)/COMM
FLOOR	целая часть аргумента.	FLOOR(SALARY)
GRAPHIC	представление первого аргумента в виде GRAPHIC.	GRAPHIC(DBCLOBCOL)
HEX	представление аргумента в шестнадцатеричном виде.	HEX(BCHARCOL)
HOUR	часть аргумента, соответствующая часам.	HOUR(TIMECOL) > 12
IFNULL	первый непустой аргумент.	IFNULL(SMLLINT1,100) + SMLLINT2 > 1000
INSERT	строка, которая получается из первого аргумента, если, начиная с позиции, заданной вторым аргументом, заменить число символов, заданное третьим аргументом, на строку, заданную четвертым аргументом.	INSERT(LASTNAME,1,3,'***)
INTEGER	представление аргумента в виде целого числа.	INTEGER(AVG(SALARY)+.5)
JULIAN_DAY	целое число, соответствующее количеству дней от 1 января 4712 года до н.э. до даты, заданной аргументом.	JULIAN_DAY(HIREDATE)
LEFT	первый аргумент, усеченный справа до длины, задаваемой вторым аргументом.	LEFT(LASTNAME,1)
LENGTH	длина своего аргумента.	LENGTH(ADDRESS)
LOG	натуральный логарифм аргумента.	LOG(NUMVAL)
LOG10	логарифм аргумента по основанию 10.	LOG10(NUMVAL)
LONG_VARCHAR	представление первого аргумента в виде символьной строки переменной длины.	LONG_VARCHAR(CLOBCOL)

Таблица 4 (Стр. 3 из 5). Скалярные функции

Скалярная функция	Возвращает...	Пример
LONG_VARGRAPHIC	представление первого аргумента в виде графической строки переменной длины.	LONG_VARGRAPHIC(DBCLOBCOL)
LOWER или LCASE	аргумент, преобразованный в нижний регистр	LOWER(DEPTNAME)
LTRIM	аргумент с удаленными пробелами слева	LTRIM(LASTNAME)
MICROSECOND	часть аргумента, соответствующая микросекундам.	MICROSECOND(TSTMPCOL) <> 0
MIDNIGHT_SECONDS	целое от 0 до 86400, соответствующее количеству секунд от полуночи до времени, заданного аргументом.	MIDNIGHT_SECONDS(TIMECOL)
MINUTE	часть аргумента, соответствующая минутам.	MINUTE(TIMECOL) = 0
MOD	остаток от деления первого аргумента на второй.	MOD(NUMCOL1,NUMCOL2)
MONTH	часть аргумента, соответствующая месяцу.	MONTH(BIRTHDATE) = 5
NULIF	NULL, если два аргумента равны. Первый аргумент, если они не равны.	NULIF(SALARY,0)
POSSTR	положение в первом аргументе строки, заданной вторым аргументом.	POSSTR(NOTE_TEXT,'Quintana')
POWER	первый аргумент, возведенный в степень, которая задается вторым аргументом.	POWER(DOUBLECOL,2)
QUARTER	целое от 1 до 4, соответствующее кварталу года даты своего аргумента.	QUARTER(HIREDATE)
RADIANS	аргумент, преобразованный из градусной меры в радианы.	RADIANS(ANGLE)
RAISE_ERROR	SQLSTATE в первом аргументе и сообщение об ошибке во втором аргументе в SQLCA для оператора SQL, который содержит функцию RAISE_ERROR.	RAISE_ERROR('70001', 'EDUCLVL value is greater than 20')
RAND	случайное число между 0 и 1, вычисленное с использованием заданного аргумента в качестве "затравки".	RAND(SEEDVAL)
REAL	представление аргумента в виде числа с плавающей точкой одинарной точности.	REAL(SALARY)
REPEAT	строка, состоящая из первого аргумента, повторенного количеством раз, которое задается вторым аргументом.	REPEAT('*',72)
REPLACE	результат замены в строке первого аргумента всех вхождений второго аргумента на третий аргумент.	REPLACE(DEPTNAME,'_','—')

Таблица 4 (Стр. 4 из 5). Скалярные функции

Скалярная функция	Возвращает...	Пример
RIGHT	первый аргумент, усеченный слева до длины, задаваемой вторым аргументом.	RIGHT(LASTNAME,5)
ROUND	первый аргумент, округленный до числа знаков после запятой, которое задается вторым аргументом.	ROUND(SALARY,0)
ROWID	представление аргумента в виде ROWID.	ROWID(:rowidvar)
RTRIM	аргумент с удаленными пробелами справа	RTRIM(LASTNAME)
SECOND	часть аргумента, соответствующая секундам.	SECOND(RECEIVED)
SIGN	целое (-1, 0 или 1), соответствующее знаку аргумента.	SIGN(NUMCOL)
SIN	синус аргумента, который воспринимается как угол в радианах.	SIN(ANGLE)
SINH	гиперболический синус аргумента, который воспринимается как угол в радианах.	SINH(ANGLE)
SMALLINT	представление аргумента в виде короткого целого числа.	SMALLINT(SALARY)
SPACE	строка из однобайтных пробелов, длина которой задается аргументом.	SPACE(3)
SQRT	квадратный корень аргумента.	SQRT(NUMCOL)
STRIP	строка с удаленными пробелами или заданными символами.	STRIP(LASTNAME,TRAILING)
SUBSTR	подстрока строки.	SUBSTR(FIRSTNAME,2,3)
TIME	время, соответствующее аргументу.	TIME(TSTMPCOL) < '13:00:00'
TIMESTAMP	временная отметка, соответствующая аргументу или аргументам.	TIMESTAMP(DATECOL, TIMECOL)
TRANSLATE	первый аргумент, в котором символы, перечисленные в третьем аргументе, заменены на символы, перечисленные во втором аргументе.	TRANSLATE(DEPTNAME,'_','')
TRUNCATE	первый аргумент, усеченный до числа знаков после запятой, которое задается вторым аргументом.	TRUNCATE(SALARY,0)
UPPER или UCASE	аргумент, преобразованный в верхний регистр	UPPER(DEPTNAME)
VALUE	первый непустой аргумент.	VALUE(SMALLINT1,100) + SMALLINT2 > 1000
VARCHAR	представление первого аргумента в символьном виде переменной длины.	VARCHAR(JOB)
VARGRAPHIC	представление первого аргумента в графическом виде переменной длины.	VARGRAPHIC('single-byte')
WEEK	целое от 1 до 54, соответствующее неделе года даты своего аргумента.	WEEK(BIRTHDATE)

Таблица 4 (Стр. 5 из 5). Скалярные функции

Скалярная функция	Возвращает...	Пример
YEAR	часть аргумента, соответствующая году.	YEAR(BIRTHDATE) = 1956

## Примеры CHAR

Функция CHAR возвращает строковое представление значения типа дата–время или десятичного числа. Это может быть полезным, если точность числа больше максимально возможной для языка хоста. Например, если ваше число имеет точность больше 18 знаков, можно записать его в переменную хоста при помощи функции CHAR. Более конкретно, если BIGDECIMAL – столбец, определенный как DECIMAL(33), можно определить строку с фиксированной длиной BIGSTRING CHAR(33), и выполнить следующий оператор:

```
SELECT CHAR(MAX(BIGDECIMAL))
      INTO :BIGSTRING
     FROM T;
```

CHAR возвращает также представление в виде символьной строки заданного формата для значения типа дата–время. Например:

```
SELECT CHAR(HIREDATE,USA)
      FROM DSN8610.EMP
     WHERE EMPNO='000010';
```

возвращает 01/01/1965.

## DECIMAL

Функция DECIMAL возвращает десятичное представление числового или символьного значения. В частности, DECIMAL может преобразовать целое значение, чтобы его можно было использовать как длительность. Предположим, что у нас есть переменная хоста PERIOD типа INTEGER. В следующем примере из таблицы DSN8610.PROJ выбираются начальные даты (PRSTDATE) и к ним добавляется период, заданный переменной хоста (PERIOD). Чтобы использовать значение PERIOD как длительность, сначала надо добиться, чтобы DB2 воспринимало его как DECIMAL(8,0):

```
EXEC SQL
SELECT PRSTDATE + DECIMAL(:PERIOD,8)
      FROM DSN8610.PROJ;
```

Функцию DECIMAL можно также использовать для преобразования символьной строки в десятичное число. Символьная строка при этом должна удовлетворять правилам записи целой или десятичной константы SQL. Сведения об этих правилахсмотрите в разделе Глава 3 книги *DB2 SQL Reference*.

Предположим, вы хотите отобрать всех сотрудников, номера телефонов которых делятся на 13. Однако в таблице PHONENO определен как CHAR(4). Чтобы отобрать таких сотрудников, надо выполнить запрос:

```
SELECT EMPNO, LASTNAME, PHONENO  
      FROM DSN8610.EMP  
     WHERE DECIMAL(PHONENO,4) = INTEGER(DECIMAL(PHONENO,4)/13)*13;
```

## VALUE

VALUE может вернуть заданное значение вместо пустого значения. Например, следующий оператор SQL выбирает значения из всех строк таблицы DSN8610.DEPT. Если руководитель отдела MGRNO не указан (то есть в столбце стоит *null*), возвращается значение 'ABSENT':

```
SELECT DEPTNO, DEPTNAME, VALUE(MGRNO, 'ABSENT')  
      FROM DSN8610.DEPT;
```

## NULIF

NULIF возвращает пустое значение, если два аргумента функции совпадают. Если они не равны, NULIF возвращает значение первого аргумента. NULIF можно использовать в вычислениях, обращающихся к столбцам, где некоторое значение указывает на отсутствие информации.

**Пример:** Допустим, вы хотите подсчитать средний доход для всех сотрудников, получающих премии. Для таких сотрудников значение премии больше 0. Следующий оператор SQL при подсчете среднего дохода учитывает только сотрудников, получающих премии.

```
SELECT AVG(SALARY+NULIF(BONUS,0)+COMM)  
          AS "AVERAGE EARNINGS"  
      FROM DSN8610.EMP;
```

## Вложенные функции столбцов и скалярные функции

Функции можно вкладывать одну в другую:

- Скалярные функции в скалярные функции

Например, вы хотите узнать месяц и день приема на работу конкретного сотрудника отдела E11 и вывести его в формате USA.

```
SELECT SUBSTR((CHAR(HIREDATE, USA)),1,5)  
      FROM DSN8610.EMP  
     WHERE LASTNAME = 'SMITH' AND WORKDEPT = 'E11';
```

даст такой результат:

06/19

- Скалярные функции в функции столбца

Аргумент функции столбца должен быть задан как столбец, поэтому если этот аргумент – скалярная функция, эта функция должна применяться к столбцу. Например, вы хотите узнать средний возраст приема на работу сотрудников отдела A00. Оператор:

```
SELECT AVG(DECIMAL(YEAR(HIREDATE - BIRTHDATE)))  
      FROM DSN8610.EMP  
     WHERE WORKDEPT = 'A00';
```

даст такой результат:

28.0

Реальная форма этого результата зависит от того, как определена переменная хоста, которой вы присваиваете результат (в данном случае DECIMAL(3,1)).

- Функции столбца в скалярные функции

Например, вы хотите узнать год приема на работу последнего сотрудника, принятого в отдел A00. Оператор:

```
SELECT YEAR(MAX(HIREDATE))
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'A00';
```

что дает такой результат:

1972

## Использование пользовательских функций

DB2 дает вам возможность определять собственные функции. Эти функции могут быть функциями с источниками, то есть основанными на существующих функциях, или же внешними, то есть написанными пользователем.

Внешние пользовательские функции могут возвращать отдельное значение или таблицу. Внешние функции, возвращающие таблицу, называют *табличными пользовательскими функциями*.

Если ни одна из встроенных скалярных функций DB2 не удовлетворяет ваших потребностей, вам надо определить и написать пользовательскую функцию, выполняющую требуемое действие. Пользовательскую функцию можно применять в любом месте, где можно применять встроенные функции.

Предположим, вы определили и написали функцию под названием REVERSE, которая обращает порядок символов в строке. Определение может выглядеть примерно так:

```
CREATE FUNCTION REVERSE(CHAR)
  RETURNS CHAR
  EXTERNAL NAME 'REVERSE'
  LANGUAGE C;
```

Теперь эту функцию можно использовать в операторе SQL в любом месте, где допустима встроенная функция с символьным аргументом. Например:

```
SELECT REVERSE(:CHARSTR)
  FROM SYSDUMMY1;
```

Хотя пользовательскую функцию столбца написать нельзя, можно определять пользовательские функции столбца на основе встроенных функций столбца. Допустим, например, что у вас есть таблица под названием EUROEMP со столбцом EUROSAL пользовательского типа EURO, который основан на типе DECIMAL(9,2). Встроенную функцию AVG нельзя использовать для поиска среднего значения EUROSAL, поскольку AVG применяется к числовым аргументам. Однако можно определить функцию AVG на основе встроенной функции AVG, принимающую аргументы типа EURO:

```
CREATE FUNCTION AVG(EURO)
  RETURNS EURO
  SOURCE AVG;
```

Эту функцию можно использовать для поиска среднего значения столбца EUROSAL:

```
SELECT AVG(EUROSAL) FROM EUROEMP;
```

Можно определить и написать табличную пользовательскую функцию, которую пользователи смогут вызывать из условия FROM оператора SELECT. Предположим, вы определили и написали функцию под названием BOOKS, которая возвращает таблицу с информацией о книгах на данную тему. Определение может выглядеть примерно так:

```
CREATE FUNCTION BOOKS(SUBJECT)
RETURNS TABLE (TITLE      VARCHAR(25),
                AUTHOR     VARCHAR(25),
                PUBLISHER VARCHAR(25),
                ISBNNUM   VARCHAR(20),
                PRICE      DECIMAL(5,2),
                CHAP1     CLOB(50K))
LANGUAGE COBOL
EXTERNAL NAME BOOKS;
```

Теперь эту функцию можно включить в условие FROM оператора SELECT, чтобы получать информацию о книгах. Например:

```
SELECT B.TITLE, B.AUTHOR, B.PUBLISHER, B.ISBNNUM
  FROM TABLE(BOOKS('Computers')) AS B
 WHERE B.TITLE LIKE '%COBOL%';
```

Информацию об определении и написании пользовательских функций смотрите в разделе “Глава 4–3. Создание и использование пользовательских функций” на стр. 267, а информацию об определении пользовательских типов – в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

## Использование выражений CASE

Выражение CASE позволяет выполнять оператор SQL несколькими способами в зависимости от значения критерия поиска.

Один из вариантов использования выражения CASE – замена значений в таблице результатов на более содержательные.

**Пример:** Допустим, мы хотим вывести номера сотрудников, их фамилии и уровни образования для всех клерков в таблице сотрудников. Уровни образования хранятся в столбце EDLEVEL как короткие целые, но вы хотели бы заменить их более понятными словосочетаниями. Эту задачу решает следующий оператор SQL:

```
SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
CASE
    WHEN EDLEVEL<=12 THEN 'HIGH SCHOOL OR LESS'
    WHEN EDLEVEL>12 AND EDLEVEL<=14 THEN 'JUNIOR COLLEGE'
    WHEN EDLEVEL>14 AND EDLEVEL<=17 THEN 'FOUR-YEAR COLLEGE'
    WHEN EDLEVEL>17 THEN 'GRADUATE SCHOOL'
    ELSE 'UNKNOWN'
END
AS EDUCATION
  FROM DSN8610.EMP
 WHERE JOB='CLERK';
```

Таблица результатов будет выглядеть примерно так:

FIRSTNAME	MIDDLEINIT	LASTNAME	EDUCATION
SEAN		O'CONNELL	JUNIOR COLLEGE
JAMES	J	JEFFERSON	JUNIOR COLLEGE
SALVATORE	M	MARINO	FOUR-YEAR COLLEGE
DANIEL	S	SMITH	FOUR-YEAR COLLEGE
SYBIL	V	JOHNSON	FOUR-YEAR COLLEGE
MARIA	L	PEREZ	FOUR-YEAR COLLEGE
GREG		ORLANDO	JUNIOR COLLEGE
ROBERT	M	MONTEVERDE	FOUR-YEAR COLLEGE

Выражение CASE заменяет каждое короткое целое в столбце EDLEVEL на словесное описание уровня образования. Если значение EDLEVEL пусто, выражение CASE заменит его словом UNKNOWN.

Другой вариант использования выражения CASE – предупредить нежелательные действия над значениями в столбцах, например, деление на ноль.

**Пример:** Чтобы определить отношение комиссионных сотрудников к их окладам, можно выполнить следующий оператор SQL:

```
SELECT EMPNO, WORKDEPT,
       COMM/SALARY AS "COMMISSION/SALARY",
    FROM DSN8610.EMP;
```

Однако это может вызвать некоторые проблемы. Если сотрудник не получает никакого оклада, при вычислении произойдет деление на ноль. Исправив оператор SELECT при помощи выражения CASE, вы избежите этого:

```
SELECT EMPNO, WORKDEPT,
       (CASE WHEN SALARY=0 THEN NULL
             ELSE COMM/SALARY
          END) AS "COMMISSION/SALARY"
    FROM DSN8610.EMP;
```

Выражение CASE определяет отношение комиссионных к окладу только, если оклад не равен нулю. Иначе DB2 устанавливает для отношения пустое значение (NULL).

## Упорядочивание строк: ORDER BY

Условие ORDER BY используется для получения строк в определенном порядке. Применение ORDER BY – единственный способ, гарантирующий вывод строк в том порядке, в котором вы хотите их получить. В следующих разделах объясняется, как использовать условие ORDER BY.

### Задание имен столбцов

Порядок выбранных строк определяется по столбцу, указанному в условии ORDER BY; этот столбец называется *столбцом упорядочивания*. Можно указать несколько столбцов для упорядочивания.

Строки можно упорядочивать в восходящем или в нисходящем порядке. Пустые значения будут последними при восходящем порядке и первыми при нисходящем порядке.

DB2 сортирует строки в последовательности упорядочивания, определяемой схемой кодировки для таблицы. Числа DB2 сортирует алгебраически, а значения даты—времени — хронологически.

### **Перечисление строк в восходящем порядке**

Чтобы получить результат в *восходящем* порядке, задайте ASC. Например, чтобы получить номера сотрудников, их фамилии и даты приема на работу для отдела A00 в *восходящем* порядке дат приема, задайте следующий оператор SQL:

```
SELECT EMPNO, LASTNAME, HIREDATE  
      FROM DSN8610.EMP  
     WHERE WORKDEPT = 'A00'  
     ORDER BY HIREDATE ASC;
```

Это даст такой результат:

EMPNO	LASTNAME	HIREDATE
000110	LUCCHESI	1958-05-16
000120	O'CONNELL	1963-12-05
000010	HAAS	1965-01-01
200010	HEMMINGER	1965-01-01
200120	ORLANDO	1972-05-05

В этом примере данные упорядочены по давности приема на работу. Порядок сортировки ASC используется по умолчанию.

### **Перечисление строк в нисходящем порядке**

Чтобы задать *нисходящий* порядок сортировки строк, укажите DESC. Например, чтобы получить номера отделов, фамилии и номера сотрудниц в *нисходящем* порядке номеров отделов, задайте следующий оператор SQL:

```
SELECT WORKDEPT, LASTNAME, EMPNO  
      FROM DSN8610.EMP  
     WHERE SEX = 'F'  
     ORDER BY WORKDEPT DESC;
```

Это даст такой результат:

WORKDEPT	LASTNAME	EMPNO
E21	WONG	200330
E11	HENDERSON	000090
E11	SCHNEIDER	000280
E11	SETRIGHT	000310
E11	SCHWARTZ	200280
E11	SPRINGER	200310
D21	PULASKI	000070
D21	JOHNSON	000260
D21	PEREZ	000270
D11	PIANKA	000160
D11	SCOUTTEN	000180
D11	LUTZ	000220
D11	JOHN	200220
C01	KWAN	000030
C01	QUINTANA	000130
C01	NICHOLLS	000140
C01	NATZ	200140
A00	HAAS	000010
A00	HEMMINGER	200010

### Упорядочивание по нескольким столбцам

Чтобы упорядочить результаты по значениям из нескольких столбцов, укажите в условии ORDER BY несколько имен столбцов.

Если у нескольких строк значения в *первом столбце упорядочивания* совпадают, эти строки будут идти в порядке значений во втором столбце, указанном в условии ORDER BY, затем в третьем и так далее. Например, два следующих оператора SELECT дадут разные результаты. Первый упорядочивает выбранные строки по должностям, а затем по уровням образования. Второй оператор SELECT упорядочивает выбранные строки по уровням образования, а затем уже по должностям.

#### Пример 1: Оператор SQL:

```
SELECT JOB, EDLEVEL, LASTNAME
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'E21'
    ORDER BY JOB, EDLEVEL;
```

дает такой результат:

JOB	EDLEVEL	LASTNAME
FIELDREP	14	LEE
FIELDREP	14	WONG
FIELDREP	16	GOUNOT
FIELDREP	16	ALONZO
FIELDREP	16	MEHTA
MANAGER	14	SPENSER

#### Пример 2: Оператор SQL:

```
SELECT JOB, EDLEVEL, LASTNAME
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'E21'
    ORDER BY EDLEVEL, JOB;
```

дает такой результат:

JOB	EDLEVEL	LASTNAME
FIELDREP	14	LEE
FIELDREP	14	WONG
MANAGER	14	SPENSER
FIELDREP	16	MEHTA
FIELDREP	16	GOUNOT
FIELDREP	16	ALONZO

Чтобы изменить обычную последовательность сортировки, можно использовать процедуру полей. Более подробную информацию о сортировке (сравнении строк)смотрите в книге *DB2 SQL Reference*, а о процедурах полей – в книге *DB2 Administration Guide*.

В следующих обстоятельствах условие ORDER BY может ссылаться на столбцы, которые не входят в условие SELECT:

- В запросе не указано UNION или UNION ALL
- Нет условия GROUP BY
- В списке SELECT нет функций столбца
- В списке SELECT нет DISTINCT

Если не выполняется какое-либо из перечисленных выше утверждений, в условии ORDER BY можно использовать только столбцы из условия SELECT.

**Пример 3:** В данном операторе SQL строки упорядочены по EDLEVEL, JOB и SALARY, хотя SALARY и не входит в список SELECT:

```
SELECT JOB, EDLEVEL, LASTNAME
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'E21'
    ORDER BY EDLEVEL, JOB, SALARY;
```

Таблица результатов будет выглядеть примерно так:

JOB	EDLEVEL	LASTNAME
FIELDREP	14	WONG
FIELDREP	14	LEE
MANAGER	14	SPENSER
FIELDREP	16	MEHTA
FIELDREP	16	GOUNOT
FIELDREP	16	ALONZO

## Ссылки на производные столбцы

Если вы использовали условие AS, чтобы дать имя столбцу в операторе SELECT, это имя можно использовать в условии ORDER BY. Например, следующий оператор SQL упорядочивает выбранную информацию по общей сумме выплат:

```
SELECT EMPNO, (SALARY + BONUS + COMM) AS TOTAL_SAL
      FROM DSN8610.EMP
    ORDER BY TOTAL_SAL;
```

---

## Получение значений по группам: GROUP BY

Условие GROUP BY используется для группировки строк по значениям в одном или нескольких столбцах. После группировки можно применять функции столбца к каждой группе.

Кроме столбцов, перечисленных в условии GROUP BY, в операторе SELECT должны быть указаны все остальные столбцы, используемые как операнды в одной из функций столбца.

Следующий оператор SQL создает списки наименьших и наибольших уровней образования сотрудников по всем отделам.

```
SELECT WORKDEPT, MIN(EDLEVEL), MAX(EDLEVEL)
      FROM DSN8610.EMP
    GROUP BY WORKDEPT;
```

Если в указанном в условии GROUP BY столбце есть пустые значения, DB2 рассматривает их как одинаковые. Таким образом, все пустые значения составляют единую группу.

Если условие GROUP BY используется, оно идет за условием FROM и любым условием WHERE, но перед условием ORDER BY.

Строки можно группировать по значениям в нескольких столбцах. Например, следующий оператор вычисляет средние оклады для мужчин и женщин в отделах A00 и C01:

```
SELECT WORKDEPT, SEX, AVG(SALARY) AS AVG_SALARY
      FROM DSN8610.EMP
     WHERE WORKDEPT IN ('A00', 'C01')
    GROUP BY WORKDEPT, SEX;
```

и дает следующий результат:

WORKDEPT	SEX	AVG_SALARY
=====	==	=====
A00	F	49625.00000000
A00	M	35000.00000000
C01	F	29722.50000000

DB2 группирует строки сначала по номеру отдела, а затем (внутри каждого отдела) по полу; после этого для каждой группы определяется среднее значение SALARY.

---

## Группировка с критериями: HAVING

Условие HAVING задает дополнительные критерии, которым должны удовлетворять полученные группы. Условие HAVING действует подобно условию WHERE, но применяется к группам; оно может содержать те же критерии поиска, которые задаются в условии WHERE. Критерий поиска в условии HAVING проверяет не свойства отдельных строк в группе, а свойства группы в целом.

Следующий оператор SQL:

```

SELECT WORKDEPT, AVG(SALARY) AS AVG_SALARY
      FROM DSN8610.EMP
     GROUP BY WORKDEPT
    HAVING COUNT(*) > 1
   ORDER BY WORKDEPT;

```

что дает такой результат:

WORKDEPT	AVG_SALARY
A00	40850.00000000
C01	29722.50000000
D11	25147.27272727
D21	25668.57142857
E11	21020.00000000
E21	24086.66666666

Сравните этот пример со вторым примером из раздела “Получение значений по группам: GROUP BY” на стр. 51. Условие HAVING COUNT(\*) > 1 говорит о том, что надо принимать во внимание только отделы, где больше одного сотрудника. (Это приводит к тому, что отделы B01 и E01 не выводятся.)

Условие HAVING проверяет свойства группы. Например, его можно использовать, чтобы получить среднюю зарплату и минимальный уровень образования для женщин в тех департаментах, где уровень образования всех сотрудниц не меньше 16. Если вас интересуют только отделы A00 и D11, можно использовать следующий оператор SQL, который проверяет свойство группы MIN(EDLEVEL):

```

SELECT WORKDEPT, AVG(SALARY) AS AVG_SALARY, MIN(EDLEVEL) AS MIN_EDLEVEL
      FROM DSN8610.EMP
     WHERE SEX = 'F' AND WORKDEPT IN ('A00', 'D11')
     GROUP BY WORKDEPT
    HAVING MIN(EDLEVEL) >= 16;

```

Этот оператор SQL дает такие результаты:

WORKDEPT	AVG_SALARY	MIN_EDLEVEL
A00	49625.00000000	18
D11	25817.50000000	17

Если вы задаете условия GROUP BY и HAVING, условие HAVING должно идти за условием GROUP BY. В функцию в условии HAVING можно включить ключевое слово DISTINCT, если вы не использовали DISTINCT где-либо еще в том же операторе SELECT. Несколько предикатов в условии HAVING можно соединять при помощи ключевых слов AND и OR, для любого предиката критерия поиска можно также использовать ключевое слово NOT. Более подробную информацию смотрите в разделе “Выбор строк, удовлетворяющих нескольким критериям” на стр. 31.

## Объединение списков значений: UNION

При помощи ключевого слова UNION можно объединять несколько операторов SELECT и получать единую таблицу результатов. Когда DB2 встречает ключевое слово UNION, она обрабатывает каждый оператор SELECT, формируя промежуточную таблицу результатов, а затем объединяет эти промежуточные таблицы. Если вы используете UNION для объединения

двоих столбцов с совпадающими именами, столбец в полученной таблице унаследует это имя.

Если вы используете оператор UNION, поле SQLNAME в SQLDA будет содержать имена столбцов первого операнда.

## Использование UNION для исключения повторений

Оператор UNION можно использовать для исключения повторений при объединении списков значений из различных таблиц. Например, вы можете получить объединенный список номеров сотрудников из двух источников:

- Сотрудники отдела D11
- Сотрудники, работающие над проектами MA2112, MA2113 и AD3111.

Следующий оператор SQL:

```
SELECT EMPNO
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'D11'
UNION
SELECT EMPNO
      FROM DSN8610.EMPPROJECT
     WHERE PROJNO = 'MA2112' OR
          PROJNO = 'MA2113' OR
          PROJNO = 'AD3111'
   ORDER BY EMPNO;
```

даст объединенную таблицу результатов с номерами сотрудников в восходящем порядке без повторений.

Если вы используете условие ORDER BY, оно должно идти после последнего оператора SELECT, входящего в объединение. В данном примере конечный порядок строк итоговой таблицы результатов определяется по первому столбцу.

## Использование UNION ALL для сохранения повторений

Если вы хотите сохранить повторения в результате оператора UNION, задайте после UNION необязательное ключевое слово ALL.

Следующий оператор SQL:

```
SELECT EMPNO
      FROM DSN8610.EMP
     WHERE WORKDEPT = 'D11'
UNION ALL
SELECT EMPNO
      FROM DSN8610.EMPPROJECT
     WHERE PROJNO = 'MA2112' OR
          PROJNO = 'MA2113' OR
          PROJNO = 'AD3111'
   ORDER BY EMPNO;
```

даст объединенную таблицу результатов с номерами сотрудников в восходящем порядке с сохранением повторений.

---

## Специальные регистры

Специальный регистр – это область хранения, которую DB2 определяет для процесса. Для изменения текущего значения регистра можно использовать оператор SET. Если имя регистра встречается в других операторах SQL, при выполнении оператора вместо него подставляется текущее значение регистра.

В операторах SQL можно использовать некоторые определенные специальные регистры. Дополнительную информацию о специальных регистрах смотрите в разделе Глава 3 книги *DB2 SQL Reference*.

- CURRENT DATE или CURRENT\_DATE
- CURRENT DEGREE
- CURRENT PACKAGESET
- CURRENT PATH
- CURRENT PRECISION
- CURRENT QUERY OPTIMIZATION
- CURRENT RULES
- CURRENT SERVER
- CURRENT SQLID
- CURRENT TIME или CURRENT\_TIME
- CURRENT TIMESTAMP или CURRENT\_TIMESTAMP
- CURRENT TIMEZONE
- USER

Если вы хотите узнать значение специального регистра, можно при помощи оператора SET *переменная–хоста* назначить значение специального регистра *переменной* в вашей программе. Подробности смотрите в описании оператора SET *переменная–хоста* в разделе Глава 6 книги *DB2 SQL Reference*.

---

## Поиск информации в каталоге DB2

Ниже на примерах показано, как обращаться к таблицам системного каталога DB2:

- Вывод списка таблиц, к которым вы можете обращаться
- Вывод списка имен столбцов в таблице

Содержание таблиц системного каталога DB2 может оказаться полезным справочным средством, когда вы начнете разрабатывать операторы SQL или прикладные программы.

## Вывод списка таблиц, которые вы можете использовать

В таблице каталога SYSIBM.SYSTABAUTH, перечисляются привилегии, данные идентификаторам (ID) авторизации. Чтобы вывести список таблиц, к которым вы можете обращаться (благодаря привилегиям, которые даны как вашему ID авторизации, так и PUBLIC), можно выполнить оператор SQL, подобный показанному в следующем примере. Для этого у вас должна быть привилегия SELECT для SYSIBM.SYSTABAUTH.

```
SELECT DISTINCT TCREATOR, TTNAME  
  FROM SYSIBM.SYSTABAUTH  
 WHERE GRANTEE IN (USER, 'PUBLIC', 'PUBLIC*') AND GRANTEETYPE = ' ';
```

**Если ваша подсистема DB2 использует внешнюю подпрограмму для авторизации доступа**, нельзя полагаться на результаты запроса к каталогу для определения того, какие таблицы вам доступны. Если такая внешняя подпрограмма установлена, доступом к таблицам управляет не только DB2, но и RACF®.

## Вывод списка столбцов в таблице

Все столбцы всех таблиц описаны в другой таблице каталога под названием SYSIBM.SYSCOLUMNS. Допустим, вы выполнили оператор из предыдущего примера (и получили список таблиц, к которым можете обращаться) и теперь хотите получить информацию о таблице DSN8610.DEPT. Для выполнения следующего примера у вас должна быть привилегия SELECT для SYSIBM.SYSCOLUMNS.

```
SELECT NAME, COLTYPE, SCALE, LENGTH
  FROM SYSIBM.SYSCOLUMNS
 WHERE TBNAME = 'DEPT'
   AND TBCREATOR = 'DSN8610';
```

Если в таблицу, информацию о столбцах которой вы запрашиваете, входят столбцы больших объектов или ROWID, поле LENGTH для таких столбцов содержит число байт, которые этот столбец занимает в таблице базы, а не реальную длину данных больших объектов или ROWID. Чтобы определить максимальную длину данных для столбцов больших объектов или ROWID, включите в запрос столбец LENGTH2. Например:

```
SELECT NAME, COLTYPE, LENGTH, LENGTH2
  FROM SYSIBM.SYSCOLUMNS
 WHERE TBNAME = 'EMP_PHOTO_RESUME'
   AND TBCREATOR = 'DSN8610';
```



## Глава 2–2. Работа с таблицами и изменение данных

В этой главе изложены следующие темы:

- Создание ваших собственных таблиц: `CREATE TABLE`
- “Создание таблиц с родительскими ключами и внешними ключами” на стр. 60
- “Изменение таблиц с проверочными ограничениями” на стр. 61
- “Создание таблиц с триггерами” на стр. 62
- “Создание временных таблиц” на стр. 62
- “Отбрасывание таблиц: `DROP TABLE`” на стр. 64
- “Определение производной таблицы: `CREATE VIEW`” на стр. 65
- “Изменение данных при помощи производной таблицы” на стр. 66
- “Отбрасывание производных таблиц: `DROP VIEW`” на стр. 66
- “Вставка строки: `INSERT`” на стр. 66
- “Изменение текущих значений: `UPDATE`” на стр. 71
- “Удаление строк: `DELETE`” на стр. 73

Дополнительную информацию о работе с таблицами и данными смотрите в книгах *DB2 SQL Reference* и в разделе Раздел 2 (Том 1) книги *DB2 Administration Guide*.

### Работа с таблицами

Вам может понадобиться создавать и отбрасывать таблицы, с которыми вы работаете. Можно создавать новые таблицы, копировать существующие таблицы, добавлять столбцы, добавлять или отбрасывать реляционные или проверочные ограничения, а также выполнять любое число изменений. В данном разделе описано, как создавать таблицы и работать с ними.

#### Создание ваших собственных таблиц: `CREATE TABLE`

Оператор `CREATE TABLE` используется для создания таблиц. Следующий оператор SQL создает таблицу под названием `PRODUCT`:

```
CREATE TABLE PRODUCT
  (SERIAL      CHAR(8)      NOT NULL,
   DESCRIPTION  VARCHAR(60)  DEFAULT,
   MFGCOST     DECIMAL(8,2),
   MFGDEPT    CHAR(3),
   MARKUP      SMALLINT,
   SALESDEPT   CHAR(3),
   CURDATE     DATE        DEFAULT);
```

Оператор `CREATE` состоит из следующих элементов:

- `CREATE TABLE`, где таблице дается имя `PRODUCT`.
- Списка столбцов, которые образуют таблицу. Для каждого столбца задайте:
  - Имя столбца (например, `SERIAL`).
  - Тип данных и атрибут длины (например, `CHAR(8)`). Дальнейшую информацию о типах данных смотрите в разделе “Типы данных” на стр. 20.

- Схему кодировки для таблицы.  
Задайте CCSID EBCDIC, чтобы использовать схему кодировки EBCDIC, или же CCSID ASCII, чтобы использовать схему кодировки ASCII. По умолчанию используется схема кодировки табличного пространства, где находится таблица.
- Значение по умолчанию (не обязательно). Смотрите раздел “Указание умолчаний.”
- Реляционное или проверочное ограничение таблицы (не обязательно). Смотрите разделы “Создание таблиц с родительскими ключами и внешними ключами” на стр. 60 и “Изменение таблиц с проверочными ограничениями” на стр. 61.

## Указание умолчаний

Если вы хотите наложить ограничения на ввод или задать значения по умолчанию, можете указать в описании столбцов:

- NOT NULL, если столбец не должен содержать пустых значений.
- UNIQUE, если значения в каждой строке должны быть уникальными и столбец не должен содержать пустых значений.
- DEFAULT, если столбец должен содержать одно из следующих принятых в DB2 значений по умолчанию:
  - Для числовых полей значение по умолчанию – ноль.
  - Для строк фиксированной длины значение по умолчанию – строки из пробелов.
  - Для строк переменной длины, включая строки больших объектов, значение по умолчанию – строка нулевой длины (пустая строка).
  - Для полей даты–времени значение по умолчанию – текущее значение соответствующего специального регистра.
- DEFAULT *значение*, если вы хотите задать следующие значения в качестве умолчаний:
  - Константу
  - USER – значение времени выполнения специального регистра USER
  - CURRENT SQLID – ID авторизации SQL для данного процесса
  - NULL
  - Имя функции преобразования типа для преобразования значения по умолчанию к пользовательскому типу столбца

Каждое описание столбца надо отделять от следующего запятой, а весь список описаний столбцов следует заключить в скобки.

## Создание рабочих таблиц

Перед проверкой операторов SQL, которые производят вставку, изменение и удаление строк, надо создать *рабочие таблицы* (копии таблиц DSN8610.EMP и DSN8610.DEPT), чтобы не испортить исходные таблицы примеров. В этом разделе показано, как создать две рабочие таблицы и как заполнить рабочую таблицу значениями, взятыми из другой таблицы.

В каждом из примеров этой главы предполагается, что вы зарегистрировались под своим ID авторизации. ID авторизации уточняет имя каждого

создаваемого вами объекта. Например, если ваш ID авторизации – SMITH, и вы создаете таблицу YDEPT, эта таблица получит имя SMITH.YDEPT. Если вы хотите обратиться к таблице DSN8610.DEPT, вам надо указать ее полное имя. Если вы хотите обратиться к вашей собственной таблице YDEPT, вам достаточно указать “YDEPT”.

### **Создание новой таблицы отделов**

Следующие операторы создают новую таблицу отделов под названием YDEPT по образцу существующей таблицы DSN8610.DEPT и индекс для YDEPT:

```
CREATE TABLE YDEPT  
    LIKE DSN8610.DEPT;  
  
CREATE UNIQUE INDEX YDEPTX  
    ON YDEPT (DEPTNO);
```

Если вы хотите, чтобы DEPTNO был первичным ключом, как в таблице примера, определите этот ключ явно. Для этого используется оператор ALTER TABLE:

```
ALTER TABLE YDEPT  
    PRIMARY KEY(DEPTNO);
```

Для копирования строк из одной таблицы в другую можно использовать оператор INSERT с условием SELECT. Следующий оператор копирует все строки из DSN8610.DEPT в вашу собственную рабочую таблицу YDEPT.

```
INSERT INTO YDEPT  
SELECT *  
FROM DSN8610.DEPT;
```

Сведения об операторе INSERT смотрите в разделе “Изменение данных DB2” на стр. 66.

### **Создание новой таблицы сотрудников**

Следующий оператор создает новую таблицу сотрудников под названием YEMP.

```
CREATE TABLE YEMP  
(EMPNO      CHAR(6)          PRIMARY KEY NOT NULL,  
 FIRSTNAME  VARCHAR(12)       NOT NULL,  
 MIDINIT    CHAR(1)          NOT NULL,  
 LASTNAME   VARCHAR(15)       NOT NULL,  
 WORKDEPT   CHAR(3)          REFERENCES YDEPT ON DELETE SET NULL,  
 PHONENO    CHAR(4)          UNIQUE NOT NULL,  
 HIREDATE   DATE            ,  
 JOB        CHAR(8)          ,  
 EDLEVEL    SMALLINT         ,  
 SEX        CHAR(1)          ,  
 BIRTHDATE  DATE            ,  
 SALARY     DECIMAL(9, 2)     ,  
 BONUS      DECIMAL(9, 2)     ,  
 COMM       DECIMAL(9, 2)     );
```

Этот оператор также создает реляционную связь между внешним ключом в YEMP (WORKDEPT) и первичным ключом в YDEPT (DEPTNO). Он также накладывает ограничение уникальности на телефонные номера.

Если вы хотите изменить определение таблицы после ее создания, используйте оператор ALTER TABLE.

Если вы хотите изменить имя таблицы после ее создания, используйте оператор RENAME TABLE. Подробное описание операторов ALTER TABLE и RENAME TABLE смотрите в разделе Глава 6 книги *DB2 SQL Reference*. Нельзя удалить столбец из таблицы или изменить определение столбца. Однако можно добавлять или отбрасывать ограничения для столбцов в таблице.

## Создание таблиц с родительскими ключами и внешними ключами

Говорят, что для ваших таблиц соблюдается *реляционная целостность*, если все ссылки от данных одного столбца таблицы на данные другого столбца той же или другой таблицы правильны. Если реляционная целостность нарушена, DB2 переводит табличное пространство или раздел в состояние отложенной проверки.

При использовании оператора CREATE TABLE для создания новой таблицы вы можете определить первичные ключи, уникальные ключи или внешние ключи. Для определения внешнего ключа, включающего несколько столбцов, используйте ключевое слово REFERENCES и необязательное условие FOREIGN KEY (для именованных реляционных связей). Определение внешнего ключа устанавливает реляционную связь между столбцами внешнего ключа в таблице и столбцами родительского ключа (первичного или уникального) в этой или в другой таблице. У родительской таблицы реляционной связи должен быть первичный ключ и первичный индекс или уникальный ключ и уникальный индекс. Непустые значения в столбце внешнего ключа должны совпадать со значениями в соответствующем столбце родительского ключа в родительской таблице.

Пример оператора CREATE TABLE, где определяется *и родительский ключ, и внешний ключ* для одних и тех же столбцов, приведен в разделе “Создание новой таблицы сотрудников” на стр. 59. Можно также с помощью отдельных условий PRIMARY KEY, UNIQUE или FOREIGN KEY в определении таблицы задать родительские и внешние ключи, использующие несколько столбцов. (В столбцах для родительских ключей *не могут* стоять пустые значения.)

Для временных таблиц родительские или внешние ключи определять нельзя. Более подробную информацию о временных таблицах смотрите в разделе “Создание временных таблиц” на стр. 62.

Если используется процессор схем, когда вы определяете первичный или уникальный ключ в операторе CREATE TABLE, DB2 создает уникальный индекс. В прочих случаях перед использованием таблицы с первичным или уникальным ключом вы должны создать уникальный индекс сами. Уникальный индекс накладывает на родительский ключ условие уникальности. Сведения о процессоре схем смотрите в разделе Раздел 2 книги *DB2 Administration Guide*.

Задание внешнего ключа задает реляционную связь с правилом удаления. Сведения о правилах удаления смотрите в разделе “Удаление из таблиц с реляционными и проверочными ограничениями” на стр. 73. Примеры создания таблиц с реляционными связями смотрите в разделе Приложение A, “Таблицы примеров DB2” на стр. 863.

Когда вы определяете реляционную связь, DB2 накладывает ограничения на каждую операцию SQL INSERT, DELETE и UPDATE, а также на использование утилиты LOAD. После создания таблицы вы можете управлять реляционными связями для этой таблицы, добавляя или отбрасывая их.

## Изменение таблиц с проверочными ограничениями

*Проверочное ограничение* позволяет задать, какие значения в столбце этой таблицы считаются допустимыми. Например, вместо того, чтобы писать подпрограмму проверки данных, вы можете использовать проверочное ограничение, задающее, что размер годового оклада в таблице не может быть меньше 15000 долларов.

Если значения в каждом столбце таблицы удовлетворяют проверочным ограничениям, определенным для этой таблицы, говорят, что для таблицы соблюдается *проверочная целостность*. Если DB2 не может гарантировать проверочную целостность, она переводит табличное пространство или раздел, содержащие эту таблицу, в состояние *отложенной проверки*, что не позволяет использовать эту таблицу в некоторых утилитах и операторах SQL.

На временные таблицы нельзя накладывать проверочные ограничения. Более подробную информацию о временных таблицах смотрите в разделе “Создание временных таблиц” на стр. 62.

Чтобы задать проверочные ограничения для одного или нескольких столбцов таблицы, используется условие CHECK и необязательное условие CONSTRAINT (для именованных проверочных ограничений). Проверочное ограничение может состоять как из одного предиката, так и из нескольких предикатов, соединенных операциями AND или OR. Первый operand каждого предиката должен быть именем столбца, второй может быть именем столбца или константой, и типы данных двух operandов должны быть совместимы.

Операторы CREATE TABLE или ALTER TABLE с условием CHECK могут задавать проверочные ограничения для базовой таблицы. Проверочные ограничения помогают вам управлять целостностью данных, определяя, какие значения могут содержать столбцы таблицы.

Следующие операторы SQL:

```
ALTER TABLE YEMP  
    ADD CHECK (WORKDEPT BETWEEN 1 and 100);  
ALTER TABLE YEMP  
    ADD CONSTRAINT BONUSCHK CHECK (BONUS <= SALARY);
```

накладывают на таблицу YEMP следующие проверочные ограничения:

- Номера отделов должны находиться в диапазоне от 1 до 100.
- Премия сотруднику не может быть больше его оклада.

BONUSCHK – именованное проверочное ограничение, что позволяет отбросить его позже при необходимости.

Хотя ограничение CHECK IS NOT NULL функционально эквивалентно NOT NULL, оно расходует больше места и не рекомендуется использовать его, если можно обойтись условием NOT NULL. Однако если вы хотите позже

снять ограничение на непустоту данных, его надо задать именно в виде условия CHECK IS NOT NULL.

## Создание таблиц с триггерами

Триггеры – это наборы операторов SQL, которые выполняются, когда в таблице DB2 происходит определенное событие. Таким событием может быть операция вставки, изменения или удаления. Как и ограничения, триггеры можно использовать для управления изменениями в таблицах DB2. Однако триггеры – более мощное средство, поскольку по сравнению с ограничениями они позволяют отслеживать более широкий спектр изменений и выполнять более широкий спектр действий. Например, проверочное ограничение позволяет гарантировать, что в таблице не будет окладов меньше 15000 долларов. Если используется триггер, при падении оклада ниже 15000 долларов может быть вызвана пользовательская функция, которая пошлет сообщение с изложением ситуации в отдел заработной платы.

Чтобы создать триггер для таблицы, используйте оператор CREATE TRIGGER. Например, чтобы известить отдел заработной платы о том, что оклад сотрудника составил меньше 15000 долларов, можно создать такой триггер:

```
CREATE TRIGGER LOWPAY AFTER UPDATE OF SALARY ON EMP
    REFERENCING NEW AS MODIFIED
    FOR EACH ROW MODE DB2SQL WHEN (MODIFIED.SALARY < 15000)
    BEGIN ATOMIC
        CALL LOWPAY_LIST(MODIFIED.EMPNO, MODIFIED.FIRSTNAME,
                          MODIFIED.MIDDLENAME, MODIFIED.LASTNAME,
                          MODIFIED.SALARY);
    END;
```

Этот триггер под названием LOWPAY активируется при изменении столбца оклада в таблице сотрудников. Триггер сравнивает новое значение оклада с 15000 и если новое значение меньше 15000 долларов, вызывает пользовательскую функцию LOWPAY\_LIST.

Если вы позже решите, что минимальный оклад надо повысить с 15000 до 20000 долларов, можно будет отбросить этот триггер при помощи оператора:

```
DROP TRIGGER LOWPAY;
```

Затем вы создадите новый триггер, который будет определять минимальный оклад 20000 долларов.

Дополнительную информацию о триггерахсмотрите в разделе “Глава 3–5. Использование триггеров для работы с активными данными” на стр. 233 и в разделе Раздел 2 (Том 1) книги *DB2 Administration Guide*.

## Создание временных таблиц

Если таблица требуется вам только на время работы процесса прикладной программы, вы можете создать ее как временную. Операторы SQL, использующие временные таблицы, могут работать быстрее, так как:

- DB2 не записывает в журнал изменения во временных таблицах.
- Для временных таблиц не бывает конфликтов блокировок.

Временные таблицы особенно полезны, когда требуется проводить сортировку или поиск в промежуточных наборах результатов с большим числом строк,

при том, что только малое подмножество этих строк требуется хранить постоянно.

Во временных таблицах могут также возвращаться наборы результатов для хранимых процедур. Более подробную информацию смотрите в разделе “Написание хранимой процедуры, возвращающей наборы результатов клиенту DRDA” на стр. 603.

Определить временную таблицу можно при помощи оператора SQL CREATE GLOBAL TEMPORARY TABLE.

**Пример 1:** Следующий оператор определяет таблицу под названием TEMPPROD:

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD
  (SERIAL      CHAR(8)      NOT NULL,
   DESCRIPTION  VARCHAR(60)  NOT NULL,
   MFGCOST     DECIMAL(8,2),
   MFGDEPT    CHAR(3),
   MARKUP      SMALLINT,
   SALESDEPT  CHAR(3),
   CURDATE     DATE        NOT NULL);
```

**Пример 2:**

Временную таблицу можно определить также, скопировав определение базовой таблицы:

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD LIKE PROD;
```

Операторы SQL в примерах 1 и 2 создают идентичные определения, несмотря на то, что в таблице PROD два столбца – DESCRIPTION и CURDATE – были определены с условиями NOT NULL WITH DEFAULT. Во временных таблицах WITH DEFAULT не поддерживается, и при применении второго метода создания TEMPPROD DB2 изменяет определения DESCRIPTION и CURDATE на NOT NULL.

После выполнения одного из двух приведенных выше операторов CREATE определение TEMPPROD будет создано, однако ни одного экземпляра этой таблицы еще не будет существовать.

Чтобы создать экземпляр TEMPPROD, надо использовать TEMPPROD в прикладной программе. DB2 создаст экземпляр таблицы, когда TEMPPROD появится в одном из следующих операторов SQL:

- OPEN
- SELECT
- INSERT
- DELETE

Экземпляр временной таблицы будет существовать на текущем сервере, пока не произойдет одно из следующих событий:

- Соединение с удаленным сервером, для которого был создан экземпляр, будет разорвано.
- Единица работы, под которой был создан экземпляр, будет завершена.

При выполнении оператора ROLLBACK DB2 удаляет экземпляр временной таблицы. При выполнении оператора COMMIT DB2 удаляет экземпляр временной таблицы, за исключением того случая, когда указатель для обращения к временной таблицы определен с условием WITH HOLD и открыт.

- Процесс прикладной программы завершается.

Допустим, вы определили TEMPPROD и затем выполняете программу, которая содержит следующие операторы:

```
EXEC SQL DECLARE C1 CURSOR FOR SELECT * FROM TEMPPROD;
EXEC SQL INSERT INTO TEMPPROD SELECT * FROM PROD;
  EXEC SQL OPEN C1;
:
EXEC SQL COMMIT;
:
EXEC SQL CLOSE C1;
```

При выполнении оператора INSERT DB2 создает экземпляр TEMPPROD и заполняет его строками из таблицы PROD. При выполнении оператора COMMIT DB2 удаляет все строки из TEMPPROD. Однако если изменить определение C1 на:

```
EXEC SQL DECLARE C1 CURSOR WITH HOLD FOR SELECT * FROM TEMPPROD;
```

DB2 не удалит содержимое TEMPPROD до завершения прикладной программы, поскольку указатель C1 определен WITH HOLD и будет открыт при выполнении оператора COMMIT. В любом случае DB2 отбрасывает экземпляр TEMPPROD при завершении прикладной программы.

## Отбрасывание таблиц: **DROP TABLE**

Следующий оператор SQL отбрасывает таблицу YEMP:

```
DROP TABLE YEMP;
```

### **Оператор *DROP TABLE* надо использовать с осторожностью:**

отбрасывание таблицы НЕ эквивалентно удалению всех ее строк. Отбрасывая таблицу, вы теряете не только данные и определение этой таблицы. Вы теряете все синонимы, производные таблицы, индексы, реляционные и проверочные ограничения, связанные с этой таблицей. Вы теряете также все полномочия, данные для этой таблицы.

Дополнительные сведения об операторе DROP смотрите в разделе Глава 6 книги *DB2 SQL Reference*.

---

## Работа с производными таблицами

В данном разделе описывается, как при помощи операторов CREATE VIEW и DROP VIEW управлять представлениями (производными таблицами) для существующих таблиц. Хотя существующую производную таблицу нельзя изменить, если ваши базовые таблицы изменились, затронув производную таблицу, ее можно отбросить и создать новую. Отбрасывание и создание производных таблиц не влияет на базовые таблицы и данные в них.

## Определение производной таблицы: CREATE VIEW

Производная таблица не содержит собственных данных; это просто записанное определение набора строк и столбцов. Производная таблица может включать любые данные (в частности, все) из одной или нескольких таблиц; в большинстве случаев производные таблицы можно использовать так же, как обычные. Использование производных таблиц позволяет упростить операторы SQL.

Оператор CREATE VIEW определяет производную таблицу и дает ей имя, подобно тому, как это делается для обычной таблицы.

```
CREATE VIEW VDEPTM AS  
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT  
    FROM DSN8610.DEPT, DSN8610.EMP  
   WHERE DSN8610.EMP.EMPNO = DSN8610.DEPT.MGRNO;
```

Эта производная таблица выводит фамилии руководителей каждого отдела вместе с данными отделов из таблицы DSN8610.DEPT.

Когда программа обращается к данным, определенным в производной таблице, DB2 по определению производной таблицы возвращает набор строк, к которым программа может обращаться при помощи операторов SQL.

Теперь, когда производная таблица VDEPTM определена, вы можете получать данные, используя эту производную таблицу. Чтобы посмотреть отделы, подчиненные отделу D01, и их руководителей, выполните следующий оператор:

```
SELECT DEPTNO, LASTNAME  
  FROM VDEPTM  
 WHERE ADMRDEPT = 'D01';
```

При создании производной таблицы в операторе CREATE VIEW можно использовать специальные регистры USER и CURRENT SQLID. Для производной таблицы DB2 использует значение USER или CURRENT SQLID, соответствующее пользователю оператора SQL (SELECT, UPDATE, INSERT или DELETE), а не создателю производной таблицы. Иными словами, ссылка на специальный регистр в определении вида означает использование значения времени выполнения.

Производные таблицы можно использовать, чтобы ограничивать доступ к данным определенного рода, таким как информация о доходах. Другие возможные цели использования производных таблиц:

- Сделать доступным для прикладной программы подмножество данных таблицы. Например, производная таблица, основанная на таблице сотрудников, может содержать только строки, относящиеся к определенному отделу.
- Скомбинировать данные из двух или нескольких таблиц и сделать полученные данные доступными для прикладной программы. При помощи оператора SELECT, который устанавливает соответствие значений из одной таблицы со значениями из другой, вы можете создать производную таблицу, содержащую данные из обеих таблиц. Однако для этого типа производных таблиц вы сможете только выбирать данные. В производной таблице, объединяющей несколько таблиц, нельзя изменять, удалять или вставлять данные.

- Выводить вычисленные данные и делать результаты вычислений доступными для прикладной программы. При вычислении можно использовать любые функции и операции, которые можно использовать в операторе SELECT.

## Изменение данных при помощи производной таблицы

Некоторые производные таблицы допускают только чтение, некоторые же допускают изменение или вставку с определенными ограничениями.

(Дополнительную информацию о производных таблицах только для чтения смотрите в разделе Глава 6 книги *DB2 SQL Reference*.) Если у производной таблицы нет ограничений на изменение, надо учитывать некоторые дополнительные соображения:

- Чтобы вставлять, изменять и удалять строки при помощи производной таблицы, надо иметь соответствующие полномочия.
- Если вставка строки в таблицу производится при помощи производной таблицы, в ней должны быть определены все столбцы базовой таблицы, для которых не задано значения по умолчанию. У вставляемой строки в каждом из таких столбцов должно быть задано значение.
- На производные таблицы, которые используются для изменения данных, накладываются те же реляционные и проверочные ограничения, что и на таблицы, используемые для определения этих производных таблиц.

## Отbrasывание производных таблиц: **DROP VIEW**

Отbrasывая производную таблицу, вы отbrasываете также все производные таблицы, определенные с ее помощью. Следующий оператор SQL отbrasывает производную таблицу VDEPTM:

```
DROP VIEW VDEPTM;
```

---

## Изменение данных DB2

В данном разделе описано, как добавлять или изменять данные в существующей таблице при помощи операторов INSERT, UPDATE и DELETE.

## Вставка строки: **INSERT**

Оператор INSERT используется для вставки новых строк в таблицу или производную таблицу. При использовании оператора INSERT можно:

- Задать значения для вставляемой строки. Можно задавать константы, переменные хоста, выражения, DEFAULT или NULL.
- Включить в оператор INSERT оператор SELECT, чтобы сообщить DB2, что данные для вставляемой строки или строк надо брать из другой таблицы или производной таблицы. В разделе “Заполнение таблицы значениями из другой таблицы: INSERT для нескольких строк” на стр. 69 объясняется, как использовать оператор SELECT в операторе INSERT для добавления строк в таблицу.

В любом случае для каждой вставляемой строки вы должны задать значения во всех столбцах, у которых нет значений по умолчанию. Для любого из столбцов, соответствующих одному из следующих условий, можно задать DEFAULT, и DB2 вставит в этот столбец значение по умолчанию:

- Допускающий пустые значения.
- Определенный со значением по умолчанию.
- С типом данных ROWID. (У столбцов ROWID всегда есть значения по умолчанию.)

Значения, которые вы можете вставить в столбец с типом данных ROWID, зависят от того, был ли этот столбец определен с условием GENERATED ALWAYS или GENERATED BY DEFAULT. Дополнительную информацию смотрите в разделе “Вставка данных в столбец ROWID” на стр. 70.

Можно задать названия столбцов, для которых вы указываете значения. Другой вариант – опустить список имен столбцов.

Для статических операторов вставки лучше задавать названия столбцов, для которых вы указываете значения, потому что:

- Ваши операторы вставки не будут зависеть от формата таблицы. (Скажем, если вы добавите к таблице новый столбец, оператор не придется менять.)
- Вы можете быть уверены, что значения попадают в правильные столбцы.
- Исходный текст оператора будет более понятным.

Если вы опустили названия столбцов в статическом операторе вставки, а к таблице, куда производится вставка, добавили столбец, при повторном связывании может произойти ошибка. Эта ошибка происходит при любом связывании оператора вставки, пока вы не измените этот оператор, включив в него значение для нового столбца. Это происходит, даже если для нового столбца есть значение по умолчанию.

Если вы задаете список названий столбцов, соответствующие значения надо указывать в том же порядке, что и имена столбцов в списке.

Например,

```
INSERT INTO YDEPT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION)
VALUES ('E31', 'DOCUMENTATION', '000010', 'E01', '');
```

После вставки новой строки отдела в таблицу YDEPT можно при помощи оператора SELECT посмотреть содержимое таблицы. Следующий оператор SQL:

```
SELECT *
  FROM YDEPT
 WHERE DEPTNO LIKE 'E%'
   ORDER BY DEPTNO;
```

выводит все новые строки отделов, которые вы вставили:

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
E01	SUPPORT SERVICES	000050	A00	
E11	OPERATIONS	000090	E01	
E21	SOFTWARE SUPPORT	000100	E01	
E31	DOCUMENTATION	000010	E01	

Вводить данные в таблицы можно двумя способами:

- Можно копировать одну таблицу в другую, как описано в разделе “Заполнение таблицы значениями из другой таблицы: INSERT для нескольких строк” на стр. 69.
- Для вставки больших объемов данных в таблицу можно написать прикладную программу. Подробности смотрите в разделе “Раздел 3. Кодирование SQL в прикладных программах хоста” на стр. 103.
- Для ввода данных из других источников можно использовать утилиту DB2 LOAD. Дополнительную информацию об утилите LOAD смотрите в разделе Раздел 2 книги *DB2 Utility Guide and Reference*.

## **Вставка строк в таблицы с реляционными связями**

При вставке строк в родительскую таблицу:

- Если уникальный индекс к данному моменту не существует, и вы не используете процессор схем, определите для родительского ключа уникальный индекс.
- Не вводите повторных значений для родительского ключа.
- Не вставляйте пустых значений ни в один из столбцов родительского ключа.

При вставке строк в зависимую таблицу:

- Каждое непустое значение, которое вы вставляете в столбец внешнего ключа, должно совпадать с одним из значений родительского ключа.
- Если одно из полей внешнего ключа пусто, весь внешний ключ считается пустым.
- Если вы отбросили индекс для родительского ключа родительской таблицы, вы не сможете вставлять строки ни в родительскую, ни в зависимую таблицы.

Например, у таблицы проектов из примера (PROJ) есть внешние ключи — номер отдела (DEPTNO), который ссылается на таблицу отделов, и номер сотрудника (RESPEMP), который ссылается на таблицу сотрудников. У каждой строки, которую вы вставляете в таблицу проектов, значение RESPEMP должно либо совпадать с некоторым значением EMPNO из таблицы сотрудников, либо быть пустым. Значение DEPTNO в строке должно совпадать с некоторым значением DEPTNO из таблицы отделов. (В этом случае пустые значения недопустимы, поскольку DEPTNO в таблице отделов определен, как NOT NULL.)

## **Вставка строк в таблицы с проверочными ограничениями**

Когда вы используете INSERT для вставки строки в таблицу, DB2 автоматически проверяет все проверочные ограничения для этой таблицы. Если данные нарушают какое-либо ограничение, определенное для этой таблицы, DB2 не вставляет эту строку.

Следующий оператор INSERT удовлетворяет всем ограничениям и будет выполнен успешно:

```
INSERT INTO YEMP
  (EMPNO, FIRSTNAME, LASTNAME, WORKDEPT, JOB, SALARY, BONUS)
VALUES (100125, 'MARY', 'SMITH', 55, 'SALES', 65000, 0);
```

Следующий оператор INSERT не будет выполнен:

```
INSERT INTO YEMP
  (EMPNO, FIRSTNME, LASTNAME, WORKDEPT, JOB, SALARY, BONUS)
  VALUES (120026, 'JOHN', 'SMITH', 25, 'MANAGER', 5000, 45000);
```

поскольку значение BONUS больше значения SALARY, а это нарушает проверочное ограничение, определенное для YEMP в разделе “Изменение таблиц с проверочными ограничениями” на стр. 61.

## **Заполнение таблицы значениями из другой таблицы: INSERT для нескольких строк**

Чтобы выбрать строки из одной таблицы и вставить их в другую, используйте подвыбор в операторе INSERT.

Следующий оператор SQL создает таблицу под названием TELE:

```
CREATE TABLE TELE
  (NAME2  VARCHAR(15)  NOT NULL,
   NAME1  VARCHAR(12)  NOT NULL,
   PHONE   CHAR(4) );
```

Следующий оператор копирует данные из DSN8610.EMP во вновь созданную таблицу:

```
INSERT INTO TELE
SELECT LASTNAME, FIRSTNME, PHONENO
  FROM DSN8610.EMP
 WHERE WORKDEPT = 'D21';
```

Два этих оператора создают и заполняют таблицу TELE, которая будет выглядеть так:

NAME2	NAME1	PHONE
PULASKI	EVA	7831
JEFFERSON	JAMES	2094
MARINO	SALVATORE	3780
SMITH	DANIEL	0961
JOHNSON	SYBIL	8953
PEREZ	MARIA	9001
MONTEVERDE	ROBERT	3780

Оператор CREATE TABLE из примера создает таблицу, которая в начальный момент пуста. В этой таблице есть столбцы для фамилий, имен и номеров телефонов, но нет строк.

Оператор INSERT заполняет эту таблицу данными, отобранными из таблицы DSN8610.EMP: фамилиями, именами и номерами телефонов сотрудников отдела D21.

**Пример:** Следующий оператор CREATE создает таблицу, которая содержит название отдела, где работает сотрудник, а также его номер телефона. Подвыбор заполняет таблицу DLIST данными из строк двух существующих таблиц, DSN8610.DEPT и DSN8610.EMP.

```

CREATE TABLE DLIST
  (DEPT    CHAR(3)      NOT NULL,
   DNAME   VARCHAR(36)   ,
   LNAME   VARCHAR(15)   NOT NULL,
   FNAME   VARCHAR(12)   NOT NULL,
   INIT    CHAR          ,
   PHONE   CHAR(4)  );

INSERT INTO DLIST
  SELECT DEPTNO, DEPTNAME, LASTNAME, FIRSTNAME, MIDINIT, PHONENO
    FROM DSN8610.DEPT, DSN8610.EMP
   WHERE DEPTNO = WORKDEPT;

```

## **Вставка данных в столбец ROWID**

Прежде чем вставлять данные в столбец ROWID, вы должны знать, как определен столбец ROWID. Столбцы ROWID можно определять как GENERATED ALWAYS или как GENERATED BY DEFAULT. GENERATED ALWAYS означает, что значения в столбце генерирует DB2; в такой столбец вставлять данные нельзя. Если столбец определен как GENERATED BY DEFAULT, вставка данных возможна, если же вы этого не сделаете, DB2 занесет в столбец значение по умолчанию. Предположим, что таблицы T1 и T2 содержат по два столбца: столбец целых чисел и столбец ROWID. Чтобы следующий оператор можно было успешно выполнить, ROWIDCOL2 должен быть определен как GENERATED BY DEFAULT.

```

INSERT INTO T2 (INTCOL2,ROWIDCOL2)
  SELECT INTCOL1, ROWIDCOL1 FROM T1;

```

Если ROWIDCOL2 определен как GENERATED ALWAYS, вставку данных столбца ROWID из T1 в T2 выполнить нельзя, однако можно вставить данные числового столбца. Чтобы вставить только целые данные, используйте один из следующих методов:

- Задайте в вашем операторе INSERT только столбец целых данных:

```

INSERT INTO T2 (INTCOL2)
  SELECT INTCOL1 FROM T1;

```

- Задайте в вашем операторе INSERT условие OVERRIDING USER VALUE, чтобы сообщить DB2, что надо игнорировать любые значения, которые вы задаете для генерируемых системой столбцов:

```

INSERT INTO T2 (INTCOL2,ROWIDCOL2) OVERRIDING USER VALUE
  SELECT INTCOL1, ROWIDCOL1 FROM T1;

```

## **Использование оператора INSERT в прикладной программе**

Если при выполнении оператора INSERT DB2 обнаруживает ошибку, вставка в таблицу не производится, а коды ошибок заносятся в переменные хоста SQLCODE и SQLSTATE или в соответствующие поля SQLCA. Если оператор INSERT выполнен успешно, SQLERRD(3) будет содержать число вставленных строк. Дополнительную информацию смотрите в разделе Приложение С книги *DB2 SQL Reference*.

**Примеры:** Следующий оператор вставляет в таблицу YEMP информацию о новом сотруднике. Поскольку у YEMP есть внешний ключ WORKDEPT, ссылающийся на первичный ключ DEPTNO в YDEPT, значение, вставляемое в столбец WORKDEPT (в данном случае E31) должно быть либо одним из DEPTNO в таблице YDEPT, либо пустым.

```
INSERT INTO YEMP
VALUES ('000400', 'RUTHERFORD', 'B', 'HAYES', 'E31',
'5678', '1983-01-01', 'MANAGER', 16, 'M', '1943-07-10', 24000,
500, 1900);
```

Следующий оператор также вставляет строку в таблицу YEMP. Однако в этом операторе не задаются значения для каждого столбца. В незаданных столбцах пустые значения допускаются, и DB2 вставит в эти столбцы пустые значения. Поскольку у YEMP есть внешний ключ WORKDEPT, ссылающийся на первичный ключ DEPTNO в YDEPT, значение, вставляемое в столбец WORKDEPT (в данном случае D11) должно быть либо одним из DEPTNO в таблице YDEPT, либо пустым.

```
INSERT INTO YEMP
(EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, JOB)
VALUES ('000410', 'MILLARD', 'K', 'FILLMORE', 'D11', '4888', 'MANAGER');
```

## Изменение текущих значений: UPDATE

Для изменения данных в таблице используется оператор UPDATE. Оператор UPDATE можно использовать также для удаления значения из столбца строки (без удаления самой строки); для этого надо изменить значение в столбце на NULL.

Допустим, например, что сотрудник назначен на новую должность. Чтобы исправить несколько элементов данных о сотруднике в рабочей таблице YEMP и отразить новое назначение, можно выполнить:

```
UPDATE YEMP
SET JOB = 'MANAGER',
PHONENO ='5678'
WHERE EMPNO = '000400';
```

Строки во временных таблицах изменять нельзя.

В условии SET указываются имена столбцов, значения которых надо изменить, и новые значения для них. Значения в столбце можно изменить на:

- Значение в столбце
- Константу
- Пустое значение

Если вы не определили столбец (при создании таблицы или при добавлении этого столбца) как допускающий нулевые значения, произойдет ошибка.

- Содержимое переменной хоста
- Специальный регистр. Текущее значение столбца можно заменить на значение указанного специального регистра.
- Выражение. Текущее значение столбца можно заменить на результат вычисления выражения.

Далее надо задать, в какие строки следует вносить изменения:

- Чтобы изменить одну строку, используйте условие WHERE, которому соответствует одна и только одна строка

- Чтобы изменить несколько строк, используйте условие WHERE, которому соответствуют только строки, подлежащие изменению.

Если условие WHERE опустить, DB2 изменит все строки таблицы или производной таблицы, занеся в них заданные вами значения.

Если DB2 обнаруживает ошибку при выполнении оператора UPDATE (например, новое значение слишком велико для столбца), выполнение оператора прекращается, а в переменные хоста SQLCODE и SQLSTATE или в соответствующие поля SQLCA заносятся коды ошибок. При этом никакие строки в таблице не меняются (если какие-либо строки уже были изменены, им возвращаются прежние значения). Если оператор UPDATE выполнен успешно, SQLERRD(3) будет содержать число измененных строк.

**Примеры:** Следующий оператор добавляет отсутствующий инициал и изменяет должность для сотрудника 000200.

```
UPDATE YEMP
  SET MIDINIT = 'H', JOB = 'FIELDREP'
 WHERE EMPNO = '000200';
```

Следующий оператор увеличивает каждому сотруднику отдела D11 оклад на 400 долларов. Этот оператор может изменять несколько строк.

```
UPDATE YEMP
  SET SALARY = SALARY + 400.00
 WHERE WORKDEPT = 'D11';
```

### **Изменение таблиц с реляционными связями**

При изменении родительской таблицы нельзя менять родительский ключ, для которого существуют зависимые строки.

Если вы изменяете зависимую таблицу, любые новые непустые значения внешнего ключа должны соответствовать родительскому ключу для каждой связи, в которой данная таблица является зависимой. Например, номера отделов в таблице работников зависят от номеров отделов в таблице отделов; вы можете не указать для сотрудника никакого отдела, но не можете указать для него несуществующий отдел.

Если при изменении таблицы с реляционными ограничениями обнаруживается ошибка, DB2 отменяет все изменения, выполненные при этом обновлении.

### **Изменение таблиц с проверочными ограничениями**

При выполнении оператора UPDATE для изменения строки таблицы DB2 автоматически проверяет все проверочные ограничения для таблицы. Если предлагаемое изменение нарушает какое-либо ограничение, определенное для этой таблицы, DB2 не изменяет эту строку.

Для таблицы YEMP, определенной в разделе “Создание новой таблицы сотрудников” на стр. 59, следующий оператор UPDATE удовлетворяет всем ограничениям и будет выполнен успешно:

```
UPDATE YEMP
  SET JOB = 'TECHNICAL'
 WHERE FIRSTNAME = 'MARY' AND LASTNAME= 'SMITH';
```

Следующий оператор UPDATE не будет выполнен:

```
UPDATE YEMP  
    SET WORKDEPT = 166  
 WHERE FIRSTNAME = 'MARY' AND LASTNAME= 'SMITH';
```

поскольку WORKDEPT должен находиться в диапазоне от 1 до 100.

## Удаление строк: **DELETE**

При помощи оператора **DELETE** можно удалять из таблицы целые строки. Оператор **DELETE** может удалить из таблицы некоторое число строк (в частности, ни одной) в зависимости от того, сколько строк удовлетворяют критерию поиска, заданному в условии **WHERE**. Если условие **WHERE** в операторе **DELETE** опущено, DB2 удаляет из указанной таблицы или производной таблицы *все строки*. Оператор **DELETE** не удаляет из строк определенные столбцы.

Оператор **DELETE** можно использовать для удаления всех строк из временной таблицы. Однако нельзя использовать оператор **DELETE** с условием **WHERE** для удаления из временной таблицы некоторых строк.

Следующий оператор **DELETE** удаляет из таблицы **YEMP** все строки, где номер сотрудника – 000060.

```
DELETE FROM YEMP  
 WHERE EMPNO = '000060';
```

При выполнении этого оператора DB2 удаляет из таблицы **YEMP** все строки, соответствующие критерию поиска.

Если при выполнении оператора **DELETE** DB2 обнаруживает ошибку, удаление не производится, а в переменные хоста **SQLCODE** и **SQLSTATE** или в соответствующие поля **SQLCA** заносятся коды ошибок. Данные в таблице не изменяются.

**SQLERRD(3)** в **SQLCA** содержит число удаленных строк. При этом учитываются только строки, удаленные в таблице, которая была указана в операторе **DELETE**. Строки, удаленные по каскадному правилу, не учитываются.

## Удаление из таблиц с реляционными и проверочными ограничениями

Удаление строки из таблиц, у которых есть родительский ключ и зависимые таблицы, подчиняется правилам удаления, заданным для этой таблицы. Чтобы удаление было успешным, оно должно удовлетворять всем правилам удаления для всех затронутых связей. Если реляционное ограничение нарушается, оператор **DELETE** не будет выполнен.

Убедитесь, что проверочные ограничения не затрагивают оператор **DELETE** косвенным образом. Допустим, например, что вы удаляете строку из родительской таблицы, что приводит к заданию нулевого значения в столбце зависимой таблицы. Если проверочное ограничение для зависимой таблицы определяет, что в столбце не должно быть пустых значений, удаление завершается неудачно и возникает ошибка.

## **Удаление всех строк из таблицы**

Оператор DELETE – мощное средство, которое удаляет из таблицы *все* строки, если условие WHERE не задает ограничений. (В сегментированных табличных пространствах удаление всех строк из таблицы происходит очень быстро.) Например, следующий оператор:

```
DELETE FROM YDEPT;
```

удаляет *все* строки из таблицы YDEPT. Если его выполнить, таблица будет существовать (в частности, в нее можно будет вставлять строки), но она будет пуста. Все существующие производные таблицы и полномочия для этой таблицы при использовании DELETE не будут затронуты. В отличие от этого, при использовании оператора DROP TABLE будут отброшены все производные таблицы и полномочия, что может сделать некорректными планы и пакеты. Сведения об операторе DROP смотрите в разделе “Отбрасывание таблиц: DROP TABLE” на стр. 64.

## Глава 2–3. Объединение данных из нескольких таблиц

Часто информация, которую вы хотите видеть, не находится в одной таблице. Чтобы сформировать строку таблицы результата, может потребоваться взять значения для некоторых столбцов из одной таблицы, а для некоторых – из другой таблицы. Можно использовать оператор `SELECT` для получения и объединения в одну строку значений столбцов из нескольких таблиц.

DB2 поддерживает следующие типы объединения: внутреннее объединение, левое внешнее объединение, правое внешнее объединение и полное внешнее объединение.

Вы можете задать объединения в условии `FROM` запроса: ниже на рис. 2 показаны способы комбинирования таблиц с использованием функций внешнего объединения.

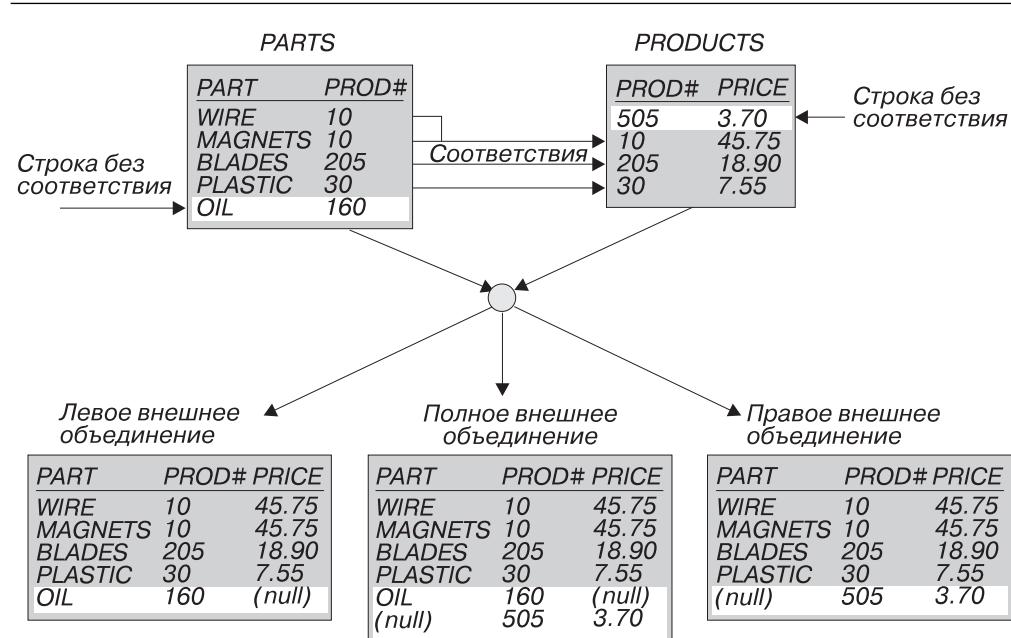


Рисунок 2. Внешние объединения двух таблиц. Каждое объединение производится по столбцу `PROD#`.

Таблица результата содержит объединенные данные из всех таблиц для строк, соответствующих критериям поиска.

Столбцы результата объединения будут иметь имена, если в списке внешнего `SELECT` указаны базовые столбцы. Однако если вы используете для построения столбца результата функцию (типа `COALESCE` или `VALUE`), этот столбец останется без имени, если вы не назовете его при помощи условия `AS` в списке `SELECT`.

В примерах в этом разделе, в которых показаны различные типы объединения, используются две следующие таблицы:

Таблица PARTS			Таблица PRODUCTS		
PART	PROD#	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	10	ACWF	505	SCREWDRIVER	3.70
OIL	160	WESTERN_CHEM	30	RELAY	7.55
MAGNETS	10	BATEMAN	205	SAW	18.90
PLASTIC	30	PLASTIK_CORP	10	GENERATOR	45.75
BLADES	205	ACE_STEEL			

## Внутреннее объединение

Чтобы произвести внутреннее объединение, выполните оператор SELECT, где в условии FROM заданы таблицы, а в условии WHERE или в условии ON – критерий объединения. Критерий объединения может быть любым простым или составным условием поиска, не содержащим ссылки на подзапрос. Глава 5 справочника *DB2 SQL Reference* описывает полный синтаксис условия объединения.

В простейшем случае внутреннего объединения критерий объединения имеет вид *столбец1=столбец2*. Например, можно объединить таблицы PARTS и PRODUCTS по столбцу PROD#, чтобы получить таблицу деталей с их поставщиками и продуктами, где эти детали используются.

Каждый из следующих примеров:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS, PRODUCTS
 WHERE PARTS.PROD# = PRODUCTS.PROD#;
```

или

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

даст результаты:

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_CORP	30	RELAY
BLADES	ACE_STEEL	205	SAW

В этом примере следует обратить внимание на три особенности:

- В таблице деталей есть деталь (OIL), продукт для которой (#160) отсутствует в таблице продуктов. В таблице продуктов есть продукт (SCREWDRIVER, #505), деталей для которого нет в таблице деталей. Ни OIL, ни SCREWDRIVER не войдут в результат объединения.
- Однако при *внешнем объединении* в результат войдут строки, где для значений в объединяемых столбцах нет соответствий.
- Синтаксис позволяет явно указать, что требуется именно внутреннее объединение. Вместо запятой в условии FROM можно написать INNER JOIN. Когда вы явным образом объединяете таблицы в условии FROM, используйте для задания критерия объединения ON (а не WHERE).

- Если вы не задаете условие WHERE в первой форме запроса, таблица результата будет содержать все возможные сочетания строк для таблиц, указанных в условии FROM. Тот же результат можно получить, указав критерий объединения, который всегда справедлив, во второй форме запроса. Например:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON 1=1;
```

В любом случае число строк в таблице результата – это произведение чисел строк в каждой из таблиц.

Можно задавать более сложные условия объединения, чтобы получать другие наборы результатов. Например, чтобы исключить из таблицы деталей, поставщиков и продуктов тех поставщиков, названия которых начинаются с буквы A, введите такой запрос:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
   AND SUPPLIER NOT LIKE 'A%';
```

Результатом этого запроса будут все строки, в которых название поставщика не начинается с буквы A:

PART	SUPPLIER	PROD#	PRODUCT
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY

**Пример объединения таблицы с этой же таблицей с использованием внутреннего объединения** В следующем примере таблица DSN8610.PROJ объединяется с этой же таблицей; возвращается номер и название каждого “главного” проекта с номером и названием проекта, который составляет часть этого главного проекта. В этом примере A указывает на первый экземпляр таблицы DSN8610.PROJ, а B – на второй экземпляр этой таблицы. Критерий объединения – значение в столбце PROJNO таблицы DSN8610.PROJ A должно быть равно значению в столбце MAJPROJ таблицы DSN8610.PROJ B.

Оператор SQL выглядит так:

```
SELECT A.PROJNO, A.PROJNAME, B.PROJNO, B.PROJNAME
  FROM DSN8610.PROJ A, DSN8610.PROJ B
 WHERE A.PROJNO = B.MAJPROJ;
```

Таблица результатов:

PROJNO	PROJNAME	PROJNO	PROJNAME
AD3100	ADMIN SERVICES	AD3110	GENERAL AD SYSTEMS
AD3110	GENERAL AD SYSTEMS	AD3111	PAYROLL PROGRAMMING
AD3110	GENERAL AD SYSTEMS	AD3112	PERSONNEL PROGRAMMG
:			
OP2010	SYSTEMS SUPPORT	OP2013	DB/DC SUPPORT

В этом примере запятая в условии FROM неявно задает внутреннее объединение; это эквивалентно использованию ключевых слов INNER JOIN. Если вы обозначаете внутреннее объединение запятой, критерий объединения

надо указывать в условии WHERE. Если вы используете ключевые слова INNER JOIN, критерий объединения надо указывать в условии ON.

## Полное внешнее объединение

Условие FULL OUTER JOIN означает включение несоответствующих строк из обеих таблиц. Отсутствующие значения в строках таблицы результатов — пустые.

Критерием объединения для полного внешнего объединения может быть простой критерий поиска, который сравнивает два столбца или функции преобразования типов, примененные к столбцам.

Например, следующий запрос выполняет полное внешнее объединение таблиц PARTS и PRODUCTS:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
      FROM PARTS FULL OUTER JOIN PRODUCTS
            ON PARTS.PROD# = PRODUCTS.PROD#;
```

Таблица результатов для этого запроса:

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
OIL	WESTERN_CHEM	160	-----
-----	-----	---	SCREWDRIVER

**Пример использования COALESCE (или VALUE):** “COALESCE” — ключевое слово, определенное в стандарте SQL как синоним функции VALUE. Эта функция (независимо от имени) может быть особенно полезна при использовании полного внешнего объединения, так как она возвращает первое не пустое значение.

Возможно, вы заметили, что в результате примера “Полное внешнее объединение” для SCREWDRIVER номер продукта был пустым, несмотря на то, что в таблице PRODUCTS номер продукта для SCREWDRIVER был. Если бы вместо PARTS.PROD# вы написали PRODUCTS.PROD#, PROD# был бы пустым для OIL. Если бы вы написали и PRODUCTS.PROD#, и PARTS.PROD#, в результат вошли бы два столбца, и в каждом некоторые значения были бы пустыми. При помощи функции COALESCE можно объединить два столбца в один и при этом избавиться от пустых значений.

Для тех же таблиц PARTS и PRODUCTS следующий пример:

```
SELECT PART, SUPPLIER,
       COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM, PRODUCT
      FROM PARTS FULL OUTER JOIN PRODUCTS
            ON PARTS.PROD# = PRODUCTS.PROD#;
```

даст такие результаты:

PART	SUPPLIER	PRODNUM	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
OIL	WESTERN_CHEM	160	-----
		505	SCREWDRIVER

Условие AS (AS PRODNUM) дает имя результату функции COALESCE.

## Левое внешнее объединение

Условие LEFT OUTER JOIN означает включение строк из таблицы, указанной перед этим условием, даже если значения в объединяющем столбце не соответствуют значениям в столбце таблицы, указанном за этим условием.

| Как и для внутреннего объединения, критерий объединения может быть любым простым или составным условием поиска, не содержащим ссылки на подзапрос.

Например, чтобы включить строки из таблицы PARTS, у которых нет соответствующих значений в таблице PRODUCTS, и оставить только цены больше 10.00, выполните запрос:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS LEFT OUTER JOIN PRODUCTS
    ON PARTS.PROD#=PRODUCTS.PROD#
   AND PRODUCTS.PRICE>10.00;
```

Результат этого запроса:

PART	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	ACWF	10	GENERATOR	45.75
MAGNETS	BATEMAN	10	GENERATOR	45.75
PLASTIC	PLASTIK_Corp	30	-----	-----
BLADES	ACE_STEEL	205	SAW	18.90
OIL	WESTERN_CHEM	160	-----	-----

Поскольку в таблице PARTS могут быть строки, для которых нет соответствия, а столбца PRICE в таблице PARTS нет, строки, где значение PRICE меньше 10.00, включены в результат объединения, но значение PRICE для них будет пустым (null).

## Правое внешнее объединение

Условие RIGHT OUTER JOIN означает включение строк из таблицы, указанной после этого условием, даже если значения в объединяющем столбце не соответствуют значениям в столбце таблицы, указанном перед этим условием.

| Как и для внутреннего объединения, критерий объединения может быть любым простым или составным условием поиска, не содержащим ссылки на подзапрос.

Например, чтобы включить строки из таблицы PRODUCTS, у которых нет соответствующих значений в таблице PARTS, и оставить только цены больше 10.00, выполните запрос:

```
SELECT PART, SUPPLIER, PRODUCTS.PROD#, PRODUCT
  FROM PARTS RIGHT OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
   AND PRODUCTS.PRICE > 10.00;
```

он даст такие результаты:

PART	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	ACWF	10	GENERATOR	45.75
MAGNETS	BATEMAN	10	GENERATOR	45.75
BLADES	ACE_STEEL	205	SAW	18.90

Поскольку в таблице PRODUCTS не может быть строк, для которых нет соответствия, а столбец PRICE находится в таблице PRODUCTS, строки, где значение PRICE меньше 10.00, не включены в результат объединения.

---

## Правила SQL для операторов, содержащих операции объединения

Правила SQL требуют, чтобы результат оператора SELECT был таким же, как если бы условия оценивались в следующем порядке:

- FROM
- WHERE
- GROUP BY
- HAVING
- SELECT

Операция объединения входит в условие FROM; таким образом, если вы хотите предсказать, какие строки будут возвращены оператором SELECT, содержащим операцию объединения, считайте, что операция объединения выполняется первой.

Предположим, например, что вы хотите получить список названий деталей, поставщиков, номеров продуктов и названий продуктов из таблиц PARTS и PRODUCTS. Этим категориям соответствуют столбцы PART, SUPPLIER, PROD# и PRODUCT. Вы хотите включать строки из каждой таблицы, даже если значение PROD# не соответствует значению PROD# в другой таблице, то есть вам нужно выполнить полное внешнее объединение. Вы хотите также исключить строки для продукта номер 10. Если вы напишете следующий оператор SELECT:

```
SELECT PART, SUPPLIER,
      VALUE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM, PRODUCT
     FROM PARTS FULL OUTER JOIN PRODUCTS
       ON PARTS.PROD# = PRODUCTS.PROD#
      WHERE PARTS.PROD# <> '10' AND PRODUCTS.PROD# <> '10';
```

вы получите такую таблицу:

PART	SUPPLIER	PRODNUM	PRODUCT
PLASTIC	PLASTIK_CORP	30	RELAY
BLADES	ACE_STEEL	205	SAW

то есть не то, что вам требовалось. DB2 сначала выполняет операцию объединения, а затем применяет условие WHERE. Условие WHERE исключает строки, где PROD# имеет пустое значение, и результат будет таким же, как при использовании внутреннего объединения.

Правильный оператор SELECT для получения списка выглядит так:

```
SELECT PART, SUPPLIER,
       VALUE(X.PROD#, Y.PROD#) AS PRODNUM, PRODUCT
  FROM
    (SELECT PART, SUPPLIER, PROD# FROM PARTS WHERE PROD# <> '10') X
 FULL OUTER JOIN
    (SELECT PROD#, PRODUCT FROM PRODUCTS WHERE PROD# <> '10') Y
      ON X.PROD# = Y.PROD#;
```

В этом случае DB2 применяет условие WHERE к каждой таблице по отдельности, и строки не будут исключаться на том основании, что PROD# пусто. Затем DB2 выполняет полное внешнее объединение, и вы получите требуемую таблицу.

PART	SUPPLIER	PRODNUM	PRODUCT
OIL	WESTERN_CHEM	160	-----
BLADES	ACE_STEEL	205	SAW
PLASTIC	PLASTIK_CORP	30	RELAY
-----	-----	505	SCREWDRIVER

---

## Использование нескольких типов объединения в одном операторе SQL

Если надо объединить больше двух таблиц, можно использовать в условии FROM несколько операций объединения, причем не обязательно одного типа. Предположим, вы хотите, чтобы в таблицу результатов вошли все сотрудники, названия их отделов и проекты, за которые они ответственны, если такие есть. Для получения всей этой информации вам понадобится объединить три таблицы. Например, можно использовать такой оператор SELECT:

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM DSN8610.EMP INNER JOIN DSN8610.DEPT
    ON WORKDEPT = DSN8610.DEPT.DEPTNO
  LEFT OUTER JOIN DSN8610.PROJ
    ON EMPNO = RESPEMP
   WHERE LASTNAME > 'S';
```

Таблица результатов:

EMPNO	LASTNAME	DEPTNAME	PROJNO
000020	THOMPSON	PLANNING	PL2100
000060	STERN	MANUFACTURING SYSTEMS	MA2110
000100	SPENSER	SOFTWARE SUPPORT	OP2010
000170	YOSHIMURA	MANUFACTURING SYSTEMS	-----
000180	SCOUTTEN	MANUFACTURING SYSTEMS	-----
000190	WALKER	MANUFACTURING SYSTEMS	-----
000250	SMITH	ADMINISTRATION SYSTEMS	AD3112
000280	SCHNEIDER	OPERATIONS	-----
000300	SMITH	OPERATIONS	-----
000310	SETRIGHT	OPERATIONS	-----
200170	YAMAMOTO	MANUFACTURING SYSTEMS	-----
200280	SCHWARTZ	OPERATIONS	-----
200310	SPRINGER	OPERATIONS	-----
200330	WONG	SOFTWARE SUPPORT	-----

---

## Использование вложенных табличных выражений и табличных пользовательских функций при объединении

Операнд объединения может быть не просто именем одной таблицы. Можно использовать:

- Вложенные табличные выражения

Вложенное табличное выражение – это подвыбор, заключенный в скобки, за которым идет корреляционное имя.

- Табличная пользовательская функция

Табличная пользовательская функция – это пользовательская функция, которая возвращает таблицу.

Следующий запрос содержит вложенное табличное выражение:

```
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
    FROM PROJECTS LEFT JOIN
(SELECT PART,
       COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
       PRODUCTS.PRODUCT
    FROM PARTS FULL OUTER JOIN PRODUCTS
      ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
  ON PROJECTS.PROD# = PRODNUM;
```

Вложенное табличное выражение:

```
(SELECT PART,
       COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
       PRODUCTS.PRODUCT
    FROM PARTS FULL OUTER JOIN PRODUCTS
      ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
```

Корреляционное имя здесь – TEMP.

**Пример использования простого вложенного табличного выражения:**

```

SELECT CHEAP_PARTS.PROD#, CHEAP_PARTS.PRODUCT
  FROM (SELECT PROD#, PRODUCT
         FROM PRODUCTS
        WHERE PRICE < 10) AS CHEAP_PARTS;

```

даст такие результаты:

PROD#	PRODUCT
505	SCREWDRIVER
30	RELAY

В этом примере корреляционное имя – CHEAP\_PARTS. На имя CHEAP\_PARTS в операторе есть две ссылки: CHEAP\_PARTS.PROD# и CHEAP\_PARTS.PRODUCT. Эти ссылки корректны, поскольку они не находятся в том же условии FROM, где определяется CHEAP\_PARTS.

#### ***Пример подвыбора как левого операнда объединения:***

```

SELECT PART, SUPPLIER, PRODNUM, PRODUCT
  FROM (SELECT PART, PROD# AS PRODNUM, SUPPLIER
         FROM PARTS
        WHERE PROD# < '200') AS PARTX
 LEFT OUTER JOIN PRODUCTS
   ON PRODNUM = PROD#;

```

даст такие результаты:

PART	SUPPLIER	PRODNUM	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
OIL	WESTERN_CHEM	160	-----

Поскольку PROD# – символьное поле, DB2 для определения множества строк в результате выполняет символьное сравнение. '30' при символьном сравнении идет после '200', поэтому строка, где PROD# – '30' в результат не войдет.

#### ***Пример объединения с табличной функцией:***

Результат табличной пользовательской функции можно объединить с таблицей, так же, как это делается для двух таблиц. Предположим, например, что CVTPRICE – табличная функция, которая преобразует цены в таблице PRODUCTS в заданную вами валюту и возвращает PRODUCTS с ценами в новой валюте. Вы можете получить таблицу деталей, поставщиков и цен продуктов в выбранной вами валюте, выполнив следующий запрос:

```

SELECT PART, SUPPLIER, PARTS.PROD#, Z.PRODUCT, Z.PRICE
  FROM PARTS, TABLE(CVTPRICE(:CURRENCY)) AS Z
 WHERE PARTS.PROD# = Z.PROD#;

```

#### ***Примеры ссылок для таблиц:***

Ссылки на таблицы можно использовать и во вложенных табличных выражениях, и как аргументы табличных функций. Основное правило для этих случаев – ссылка должна делаться из спецификации таблицы на более высоком уровне иерархии подзапросов. Можно использовать также ссылку, если спецификация таблицы, на которую производится ссылка, находится в

том же условии FROM левее ссылки и ссылка находится в одном из следующих условий:

- Во вложенном табличном выражении, перед которым стоит ключевое слово TABLE
- В аргументе табличной функции

| Табличные функции или табличные выражения, которые содержат корреляционные ссылки на другие таблицы в одном и том же условии FROM, нельзя использовать в полном внешнем объединении или в правом внешнем объединении.

В следующих примерах показано правильное использование ссылок в спецификации таблиц:

```
SELECT T.C1, Z.C5  
  FROM T, TABLE(TF3(T.C2)) AS Z  
 WHERE T.C3 = Z.C4;
```

Ссылка T.C2 корректна, поскольку спецификация таблицы, на которую делается ссылка – T – находится левее. Если бы вы задали объединение в другом порядке, где T шло бы за TABLE(TF3(T.C2)), ссылка T.C2 была бы некорректной.

```
SELECT D.DEPTNO, D.DEPTNAME,  
       EMPINFO.AVGSAL, EMPINFO.EMPCOUNT  
  FROM DEPT D,  
        TABLE(SELECT AVG(E.SALARY) AS AVGSAL,  
              COUNT(*) AS EMPCOUNT  
             FROM EMP E  
            WHERE E.WORKDEPT=D.DEPTNO) AS EMPINFO;
```

Ссылка D.DEPTNO корректна, поскольку перед вложенным табличным выражением, где она встречается, стоит ключевое слово TABLE и спецификация таблицы D стоит слева от вложенного табличного выражения в условии FROM. Если ключевое слово TABLE убрать, ссылка D.DEPTNO станет некорректной.

## Глава 2–4. Использование подзапросов

Подзапросы используются, когда надо уточнить критерий поиска на основе информации из промежуточной таблицы. Например, вам могут понадобиться номера всех сотрудников из одной таблицы, которые также встречаются для данного проекта в другом проекте.

В этой главе приводится обзор подзапросов, показано, как включать подзапросы в условия WHERE или HAVING, а также как использовать связанные подзапросы.

### Обзор основных понятий

Предположим, вы хотите получить список номеров сотрудников, их фамилий и комиссионных для всех, кто работает над определенным проектом, например проектом номер MA2111. Легко написать первую часть оператора SELECT:

```
SELECT EMPNO, LASTNAME, COMM  
      FROM DSN8610.EMP  
     WHERE EMPNO  
  
      :
```

Однако дальше вы встретите затруднение: в таблице DSN8610.EMP нет данных о номерах проектов. Вы не можете узнать, кто из сотрудников работает над проектом MA2111, не выполнив другой оператор SELECT для таблицы DSN8610.EMPPROJACT.

Чтобы разрешить эту проблему, можно использовать подвыбор. Подвыбор в условии WHERE называется *подзапросом*. Оператор SELECT, внутри которого используется подзапрос, называется *внешним оператором SELECT*.

```
SELECT EMPNO, LASTNAME, COMM  
      FROM DSN8610.EMP  
     WHERE EMPNO IN  
          (SELECT EMPNO  
             FROM DSN8610.EMPPROJACT  
            WHERE PROJNO = 'MA2111');
```

Чтобы лучше понять результаты выполнения этого оператора SQL, представьте себе, что DB2 выполняет следующий процесс:

1. DB2 обрабатывает подзапрос и получает список значений EMPNO:

```
(SELECT EMPNO  
      FROM DSN8610.EMPPROJACT  
     WHERE PROJNO = 'MA2111');
```

который образует промежуточную таблицу результатов:  
(Из EMPPROJACT)

000200
000200
000220

2. Эта промежуточная таблица используется как список в критерии поиска внешнего оператора SELECT. Это будет эквивалентно выполнению следующего оператора:

```
SELECT EMPNO, LASTNAME, COMM  
      FROM DSN8610.EMP  
     WHERE EMPNO IN  
          ('000200', '000220');
```

В итоге таблица результатов будет выглядеть так:

Выборка	EMPNO	LASTNAME	COMM
1 →	000200	BROWN	2217
2 →	000220	LUTZ	2387

## Связанные и несвязанные подзапросы

Подзапросы поставляют информацию для задания строки (в условии WHERE) или группы строк (в условии HAVING). Подзапрос дает таблицу результатов, используемую для задания выбираемой строки или группы строк. Если этот подзапрос одинаков для каждой строки или группы, он выполняется всего один раз.

Такой подзапрос называется *несвязанным*. В предыдущем запросе содержимое подзапроса одно и то же для любой строки таблицы DSN8610.EMP.

Подзапросы, содержимое которых различно для разных строк или групп строк, называются *связанными* подзапросами. Сведения о связанных подзапросах смотрите в разделе “Использование связанных подзапросов” на стр. 89. Вся информация вплоть до этого раздела применима как к связанным, так и к несвязанным подзапросам.

## Подзапросы и предикаты

Подзапрос всегда составляет часть предиката. Предикат имеет вид:  
*операнд операция (подзапрос)*

Этот предикат может быть частью условия WHERE или HAVING. Условие WHERE или HAVING может включать предикаты, содержащие подзапросы. Предикат с подзапросом, как и любой другой предикат поиска, можно заключать в скобки, ставить перед ним ключевое слово NOT и соединять с другими предикатами при помощи ключевых слов AND и OR. Например, условие WHERE запроса может выглядеть так:

```
WHERE X IN (подзапрос1) AND (Y > SOME (подзапрос2) OR Z IS NULL)
```

Подзапросы могут появляться также в предикатах других подзапросов. Такие подзапросы называются *вложенными* с некоторым уровнем вложения. Например, подзапрос в подзапросе внешнего оператора SELECT имеет уровень вложения 2. DB2 допускает уровни вложения до 15, но лишь немногие операторы требуют использования уровней вложения выше 1.

Отношения подзапроса к внешнему оператору SELECT те же, что и отношения вложенного подзапроса к подзапросу; к ним применимы те же правила, если не сказано иного.

## Таблица результатов подзапроса

Если не используется ключевое слово EXISTS, подзапрос должен возвращать таблицу результатов из одного столбца. Это значит, что в условии SELECT в подзапросе должен быть указан единственный столбец или единственное выражение. Например, любое из следующих условий SELECT допустимо:

```
SELECT AVG(SALARY)
SELECT EMPNO
```

Таблица результатов может включать несколько значений, если только подзапрос не является подзапросом элементарного предиката.

## Подвыборы для UPDATE, DELETE и INSERT

Если подвыбор используется в операторе UPDATE, DELETE и INSERT, в подзапросе нельзя обращаться к той же таблице, что и в операторе UPDATE, DELETE или INSERT.

---

## Как кодировать подзапрос

Задавать подзапрос в условии WHERE или HAVING можно различными способами. Ниже перечислены возможные варианты:

- Элементарный предикат
- Уточненные предикаты: ALL, ANY и SOME
- С ключевым словом IN
- С ключевым словом EXISTS

## Элементарный предикат

Подзапрос можно использовать сразу после любой операции сравнения. В таком случае подзапрос должен возвращать не более одного значения. DB2 сравнивает это значение со значением слева от оператора сравнения.

Например, следующий оператор SQL возвращает номера, фамилии и оклады тех сотрудников, чей уровень образования выше среднего по фирме.

```
SELECT EMPNO, LASTNAME, SALARY
  FROM DSN8610.EMP
 WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
      FROM DSN8610.EMP);
```

## Уточненные предикаты: ALL, ANY и SOME

Можно использовать подзапрос после операции сравнения, за которой идет ключевое слово ALL, ANY или SOME. В этом случае подзапрос может возвращать ноль, одно или больше значений, в том числе пустых.

- Ключевое слово ALL указывает на то, что первый operand должен сравниваться со **всеми** значениями, которые возвращает подзапрос. Например, предположим, что вы используете операцию сравнения 'больше' с ключевым словом ALL:

*WHERE выражение > ALL (подзапрос)*

Значение выражения, удовлетворяющее этому условию WHERE, должно быть больше всех значений, возвращенных подзапросом. Если подзапрос

возвратит пустую таблицу результатов, любое значение будет удовлетворять предикату.

- Ключевое слово ANY или SOME указывает на то, что сравнение значения должно быть успешным хотя бы с одним из значений, которые возвращает подзапрос. Например, предположим, что вы используете операцию сравнения 'больше' с ключевым словом ANY:

WHERE выражение > ANY (подзапрос)

Значение выражения, удовлетворяющее этому условию WHERE, должно быть больше по крайней мере одного из значений, возвращенных подзапросом (то есть больше минимального из этих значений). Если подзапрос возвратит пустую таблицу результатов, никакое значение не будет удовлетворять предикату.

Если подзапрос, который возвращает одно или несколько пустых значений, даст неожиданные для вас результаты, посмотрите описание уточненных предикатов в разделе Глава 3 книги *DB2 SQL Reference*.

## Использование ключевого слова IN

Ключевое слово IN указывает, что значение выражения должно находиться среди значений, возвращаемых подзапросом. Использование IN эквивалентно использованию "= ANY" или "= SOME."

## Использование ключевого слова EXISTS

В предыдущих примерах DB2 оценивала подзапросы и использовала их результаты как часть условия WHERE внешнего оператора SELECT. В отличие от этого при задании ключевого слова EXISTS DB2 просто проверяет, что подзапрос возвращает одну или несколько строк. Если возвращается хотя бы одна строка, критерий считается выполненным; если же ни одной строки не возвращается, критерий считается не выполненным. Например:

```
SELECT EMPNO, LASTNAME
  FROM DSN8610.EMP
 WHERE EXISTS
   (SELECT *
    FROM DSN8610.PROJ
     WHERE PRSTDATE > '1986-01-01');
```

В данном примере критерий поиска будет выполнен, если у одного из проектов в таблице DSN8610.PROJ проектная дата начала работ позже 1 января 1986 года. По этому примеру оценить полезность EXISTS трудно, поскольку результат будет один и тот же для каждой строки, проверяемой для внешнего оператора SELECT. Это значит, что в результате появятся либо все строки, либо ни одной из них. Для связанных подзапросов применение EXISTS более осмысленно, поскольку результат подзапроса может меняться от строки к строке.

Как показано в примере, в подзапросе условия EXISTS задавать имена столбцов не требуется. Вместо этого можно написать SELECT \*. Ключевое слово EXISTS можно сочетать с ключевым словом NOT, чтобы отобрать строки, для которых данных, задаваемых вашим критерием, не существует; то есть можно написать

WHERE NOT EXISTS (SELECT ...);

## Использование связанных подзапросов

В ранее описанных подзапросах DB2 обрабатывала подзапрос один раз, подставляя его результат в правую часть условия поиска и выполняла внешний оператор SELECT на основе значения критерия поиска. Можно также писать подзапросы, которые DB2 будет при выполнении внешнего оператора SELECT обрабатывать заново для каждой строки (в условии WHERE) или группы строк (в условии HAVING). Такие подзапросы называются **связанными**.

**Пользовательские функции в связанных подзапросах:** Будьте осторожны при вызове из связанных подзапросов пользовательских функций, которые используют временную память. DB2 не очищает временную память между вызовами подзапроса. Это может привести к неожиданным результатам, так как во временной памяти будут храниться значения, оставшиеся от предыдущего вызова подзапроса.

## Пример связанного подзапроса

Предположим, что вы хотите получить список сотрудников, у которых уровень образования выше среднего по отделу, где они работают. Для этого DB2 должна провести поиск в таблице DSN8610.EMP. Для каждого сотрудника в этой таблице DB2 должна сравнить его уровень образования со средним по его отделу.

Обратите внимание на отличие здесь связанных запросов от несвязанных. В примере несвязанного подзапроса выше уровень образования сравнивался со средним по всей компании, для вычисления которого требовалось просмотреть таблицу целиком. В связанном подзапросе оценивается только отдел, соответствующий определенному сотруднику.

В подзапросе вы сообщаете DB2, что надо вычислить средний уровень образования для отдела, указанного в текущей строке. Запрос, который это делает, будет выглядеть так:

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
  FROM DSN8610.EMP X
 WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
      FROM DSN8610.EMP
     WHERE WORKDEPT = X.WORKDEPT);
```

Связанный запрос выглядит так же, как несвязанный, но в нем есть одна или несколько **связанных ссылок**. В данном примере присутствует единственная связанные ссылка – X.WORKDEPT в условии WHERE подвыбора. В этом условии квалификатор X – **корреляционное имя**, определенное в условии FROM внешнего оператора SELECT. X означает, что строки берутся из первого экземпляра DSN8610.EMP. В каждый момент выполнения запроса X будет означать строку DSN8610.EMP, к которой применяется условие WHERE.

Рассмотрим, что происходит, когда этот подзапрос выполняется для некоторой строки DSN8610.EMP. Перед выполнением X.WORKDEPT принимает значение из столбца WORKDEPT этой строки. Допустим, например, что обрабатывается строка для CHRISTINE HAAS. Она работает в отделе A00, что записано в

столбце WORKDEPT этой строки. Таким образом, для этой строки подзапрос будет выглядеть так:

```
(SELECT AVG(EDLEVEL)
   FROM DSN8610.EMP
 WHERE WORKDEPT = 'A00');
```

В результате выполнения подзапроса мы получим средний уровень образования по отделу Кристины Хаас. Затем во внешнем подвыборе это значение сравнивается с собственным уровнем образования Кристины. Если в другой строке WORKDEPT будет иметь другое значение, это значение будет использоваться в подзапросе вместо A00. Например, для строки MICHAEL L THOMPSON, значение отдела – B01, и подзапрос даст средний уровень образования по отделу B01.

Таблица результатов, полученных по этому запросу, будет выглядеть так:  
(Из EMP)

Выборка	EMPNO	LASTNAME	WORKDEPT	EDLEVEL
1→	000010	HAAS	A00	18
2→	000030	KWAN	C01	20
3→	000070	PULASKI	D21	16
4→	000090	HENDERSON	E11	16
	.	.	.	.
	.	.	.	.
	.	.	.	.

## Использование корреляционных имен в ссылках

Связанные ссылки могут появляться в критериях поиска в подзапросах, вложенных табличных выражениях или как аргументы табличных пользовательских функций. Сведения о связанных ссылках во вложенных табличных выражениях и в табличных функциях смотрите в разделе “Использование вложенных табличных выражений и табличных пользовательских функций при объединении” на стр. 82. В подзапросе критерия поиска ссылка должна иметь вид X.C, где X – корреляционное имя, а C – имя столбца в таблице, обозначенной X.

Это корреляционное имя вводится в условии FROM некоторого запроса. Этот запрос может быть внешним SELECT или любым подзапросом, содержащим ссылку. Например, допустим, что запрос содержит подзапросы A, B и C, причем A содержит B, а B содержит C. Тогда в C можно использовать корреляционное имя, которое определено в B, в A или во внешнем SELECT.

Корреляционное имя можно задать для каждой таблицы, появляющейся в условии FROM. Для этого его надо просто указать после имени таблицы. Между именем таблицы и ее корреляционным именем надо оставить один или несколько пробелов. Чтобы улучшить читаемость оператора SQL, можно вставить между именем таблицы и ее корреляционным именем ключевое слово AS. Например:

```

SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
  FROM DSN8610.EMP AS X
 WHERE EDLEVEL >
      (SELECT AVG(EDLEVEL)
        FROM DSN8610.EMP
       WHERE WORKDEPT = X.WORKDEPT);

```

В запросе можно использовать сколько угодно корреляционных имен. На их применение нет ограничений. Например, можно определить одно корреляционное имя в ссылке во внешнем операторе SELECT, а другое – во вложенном подзапросе.

## Использование связанных подзапросов в операторе UPDATE

Когда вы используете связанный подзапрос в операторе UPDATE, корреляционное имя относится к строкам, которые вы изменяете. Допустим, если все действия по проекту должны завершиться до сентября 1997 года, ваш отдел рассматривает этот проект как приоритетный. Для оценки проектов из таблицы DSN8610.PROJ и записи единицы (как признака приоритетности) в столбец PRIORITY (столбец, добавленный для этой цели к DSN8610.PROJ) для каждого приоритетного проекта можно использовать такой оператор SQL:

```

UPDATE DSN8610.PROJ X
SET PRIORITY = 1
WHERE DATE('1997-09-01') >
      (SELECT MAX(ACENDATE)
        FROM DSN8610.PROJECT
       WHERE PROJNO = X.PROJNO);

```

Проверяя каждую строку таблицы DSN8610.PROJ, DB2 определяет максимальную дату завершения работ (ACENDATE) для всех действий по проекту (из таблицы DSN8610.PROJECT). Если дата завершения каждой работы, связанной с проектом, меньше чем сентябрь 1997 года, текущая строка таблицы DSN8610.PROJ будет выбрана, и DB2 изменит ее.

## Использование связанных подзапросов в операторе DELETE

Когда вы используете связанный подзапрос в операторе DELETE, корреляционное имя относится к строке, которую вы удаляете. DB2 оценивает связанный подзапрос один раз для каждой строки таблицы, названной в операторе DELETE, чтобы решить, надо ли удалять эту строку.

Допустим, например, что отдел рассматривает проект как завершенный, если сумма текущих значений рабочего времени для него не больше половины. В таком случае отдел удаляет строку этого проекта из таблицы DSN8610.PROJ. В примере оператора ниже PROJ и PROJECT – независимые таблицы; то есть для них не определены никакие реляционные связи.

```

DELETE FROM DSN8610.PROJ X
 WHERE .5 >
      (SELECT SUM(ACSTAFF)
        FROM DSN8610.PROJECT
       WHERE PROJNO = X.PROJNO);

```

При обработке этого оператора DB2 вычисляет для каждого проекта (то есть для каждой строки таблицы DSN8610.PROJ) суммарную занятость для этого

проекта и сравнивает ее с 0.5. Если она меньше 0.5, DB2 удаляет строку из таблицы DSN8610.PROJ.

В продолжение этого примера предположим, что DB2 удаляет строку из таблицы DSN8610.PROJ. Надо удалить строки, относящиеся к удаленному проекту, и из таблицы DSN8610.PROJECT. Для этого используется оператор:

```
DELETE FROM DSN8610.PROJECT X  
WHERE NOT EXISTS  
(SELECT *  
FROM DSN8610.PROJ  
WHERE PROJNO = X.PROJNO);
```

DB2 определяет для каждой строки таблицы DSN8610.PROJECT, существует ли проект с таким номером в таблице DSN8610.PROJ. Если такого проекта нет, DB2 удаляет строку из DSN8610.PROJECT.

Подзапрос в операторе DELETE не может ссылаться на ту таблицу, из которой удаляются строки. В прикладной программе примера некоторым отделам подчинены другие отделы. Рассмотрим следующий оператор, который должен был бы удалить все отделы, которым не подчинены другие отделы:

```
DELETE FROM DSN8610.DEPT X  
WHERE NOT EXISTS (SELECT * FROM DSN8610.DEPT  
WHERE ADMRDEPT = X.DEPTNO);
```

Результаты не должны зависеть от порядка, в котором DB2 обращается к столбцам таблицы. Если бы оператор можно было бы выполнить, его результат зависел бы от того, оценивает DB2 строку управляющего отдела до удаления отдела, который подчинен ему, или же после удаления. Поэтому DB2 запрещает такие операции.

То же правило распространяется на зависимые таблицы, участвующие в реляционных связях. Если в операторе DELETE есть подзапрос, который обращается к таблице, участвующей в процессе удаления, последнее правило удаления в пути к этой таблице должно быть RESTRICT или NO ACTION. Например, при отсутствии реляционных связей следующий оператор удаляет отделы, руководители которых не указаны правильно в таблице сотрудников:

```
DELETE FROM DSN8610.DEPT THIS  
WHERE NOT DEPTNO =  
(SELECT WORKDEPT  
FROM DSN8610.EMP  
WHERE EMPNO = THIS.MGRNO);
```

Если для таблиц этого примера определены реляционные связи, выполнение этого оператора приведет к ошибке. Удаление затрагивает таблицу, к которой обращается подзапрос (DSN8610.EMP – таблица, зависимая от DSN8610.DEPT), и последнее правило удаления в пути к EMP – SET NULL, а не RESTRICT и не NO ACTION. Если бы этот оператор был выполнен, результаты зависели бы от порядка, в котором DB2 обращается к столбцам.

---

## Глава 2–5. Исполнение SQL с вашего терминала при помощи SPUFI

В этой главе описано, как вводить и выполнять операторы SQL на терминале TSO при помощи утилиты SPUFI (SQL processor using file input – процессор SQL, использующий файловый ввод). Большинство интерактивных примеров SQL, показанных в разделе Раздел 2. Использование запросов SQL, можно выполнить, следуя инструкциям из этой главы и используя таблицы примеров, показанные в разделе Приложение А, “Таблицы примеров DB2” на стр. 863. В этих инструкциях предполагается, что ISPF для вас доступна.

---

### Размещение входного набора данных и использование SPUFI

Прежде чем использовать SPUFI, надо разместить входной набор данных, если его еще не существует. Этот набор данных будет содержать один или несколько операторов SQL, которые вы хотите выполнить. Информацию о ISPF и размещении наборов данных смотрите в книге *ISPF V4 User's Guide*.

Чтобы использовать SPUFI, выберите SPUFI из меню основных опций DB2I, как показано на рис. 3.

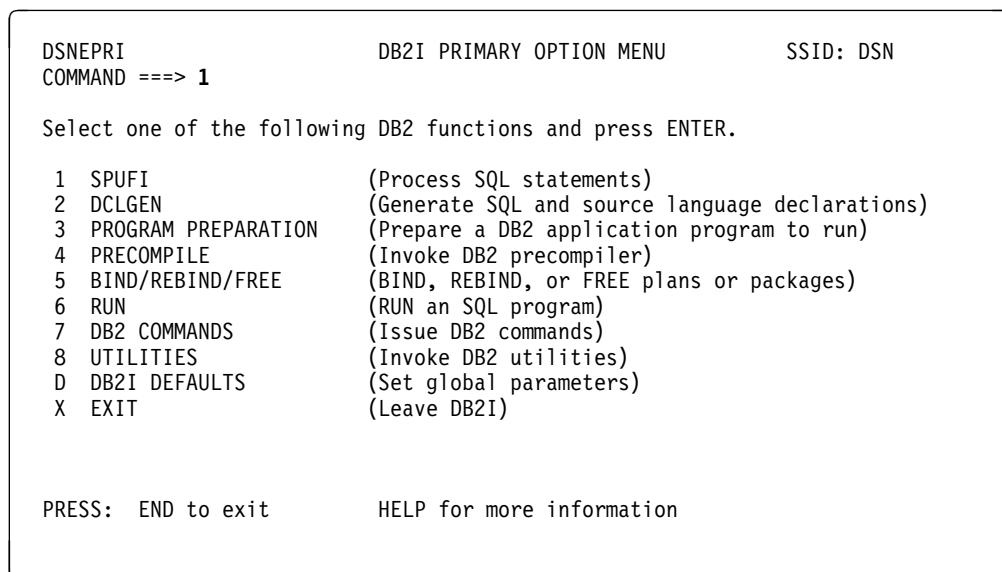


Рисунок 3. Меню основных опций DB2I с выбранной опцией 1

После этого выводится панель SPUFI, показанная на рис. 4 на стр. 94.

При этом на появившейся панели SPUFI поля ввода данных будут содержать значения, которые вы ввели в них ранее. При необходимости можно задавать имена наборов данных и опции обработки каждый раз, когда выводится панель SPUFI. Те значения, которые вы не меняете, остаются в силе.

```

DSNESP01                               SPUFI                               SSID: DSN
====>
Enter the input data set name:  (Can be sequential or partitioned)
 1 DATA SET NAME..... ==> EXAMPLES(XMP1)
 2 VOLUME SERIAL..... ==>          (Enter if not cataloged)
 3 DATA SET PASSWORD. ==>          (Enter if password protected)

Enter the output data set name:  (Must be a sequential data set)
 4 DATA SET NAME..... ==> RESULT

Specify processing options:
 5 CHANGE DEFAULTS... ==> Y      (Y/N - Display SPUFI defaults panel?)
 6 EDIT INPUT..... ==> Y        (Y/N - Enter SQL statements?)
 7 EXECUTE..... ==> Y         (Y/N - Execute SQL statements?)
 8 AUTOCOMMIT..... ==> Y       (Y/N - Commit after successful run?)
 9 BROWSE OUTPUT..... ==> Y     (Y/N - Browse output data set?)

For remote SQL processing:
10 CONNECT LOCATION ==>

PRESS: ENTER to process      END to exit      HELP for more information

```

*Рисунок 4. Заполненная панель SPUFI*

Заполните панель SPUFI следующим образом:

#### 1,2,3 INPUT DATA SET NAME (ИМЯ ВХОДНОГО НАБОРА ДАННЫХ)

В полях 1 – 3 укажите входной набор данных. Этот набор данных содержит один или несколько операторов SQL, которые вы хотите выполнить. Разместите этот набор данных до использования SPUFI, если его еще не существует.

- Имя должно соответствовать стандартным соглашениям об именовании TSO.
- Перед началом сеанса набор данных может быть пустым. Затем вы можете добавлять в него операторы SQL, редактируя этот набор данных из SPUFI.
- Этот набор данных может быть последовательным или многораздельным, но его характеристики DCB должны быть следующими:
  - Формат записи (RECFM) – либо F, либо FB.
  - Длина логической записи (LRECL) – 79 или 80. Для любого набора данных, кроме создаваемого командой EXPORT из QMF, используйте 80.
- Данные в наборе данных могут начинаться с позиции 1. Они могут идти до позиции 71 при длине логической записи 79 и до позиции 72 при длине логической записи 80. SPUFI полагает, что последние 8 байт каждой записи – последовательный номер.

Когда вы используете эту панель повторно, имя ранее использованного набора данных выводится в поле DATA SET NAME. Чтобы создать новый компонент существующего многораздельного набора данных, измените только имя компонента.

#### 4 OUTPUT DATA SET NAME (ИМЯ ВЫХОДНОГО НАБОРА ДАННЫХ)

Введите имя набора данных, куда будет направлен вывод оператора SQL. Размещать этот набор данных предварительно не требуется.

Если этот набор данных существует, новый вывод заменяет его прежнее содержание. Если набор данных не существует, DB2 размещает набор данных на устройстве с типом, указанным на панели CURRENT SPUFI DEFAULTS, и затем заносит новый набор данных в каталог. Это устройство должно быть устройством хранения данных прямого доступа, и у вас должны быть права занимать место на этом устройстве.

Для выходного набора данных необходимы следующие атрибуты:

- Организация: последовательная
- Формат записи: F, FB, FBA, V, VB или VBA
- Длина записи: от 80 до 32768 байт, не меньше длины записи входного набора данных

На рис. 4 на стр. 94 показан простейший вариант – ввод имени **RESULT**. SPUFI размещает набор данных с именем *userid.RESULT* и направляет весь вывод в этот набор данных. Если набор данных с названием *userid.RESULT* уже существует, SPUFI направляет вывод DB2 в него, заменяя все существующие данные.

#### 5 CHANGE DEFAULTS (ИЗМЕННИТЬ УМОЛЧАНИЯ)

Позволяет изменить управляющие значения и характеристики выходного набора данных и формата вашего сеанса SPUFI. Если задать в этом поле Y(YES), выводится панель умолчаний SPUFI. Дополнительную информацию о задаваемых значениях и их влиянии на работу SPUFI и характеристики вывода смотрите в разделе “Изменение умолчаний SPUFI (не обязательное)” на стр. 96. Для данного примера менять умолчания SPUFI не требуется.

#### 6 EDIT INPUT (РЕДАКТИРОВАТЬ ВВОД)

Если вы хотите редактировать входной набор данных, оставьте в строке 6 Y(YES). Для создания нового компонента входного набора данных и ввода операторов SQL можно использовать редактор ISPF. (Если вы хотите сразу начать обработку уже существующего набора операторов SQL, введите N(NO). В этом случае шаг, описанный в разделе “Ввод операторов SQL” на стр. 99, будет пропущен.)

#### 7 EXECUTE (ВЫПОЛНИТЬ)

Чтобы выполнить операторы SQL из входного набора данных, оставьте в строке 7 Y(YES).

SPUFI обработает операторы SQL, которые могут быть динамически подготовлены. Эти операторы SQL описаны в разделе Приложение G, “Характеристики операторов SQL в DB2 for OS/390” на стр. 967.

#### 8 AUTOCOMMIT (АВТОМАТИЧЕСКОЕ ПРИНЯТИЕ)

Чтобы изменения в данных DB2 стали постоянными, оставьте в строке 8 Y(YES). При этом SPUFI выполнит команду COMMIT, если все операторы будут выполнены успешно. Если не все операторы будут выполнены успешно, SPUFI выполнит команду ROLLBACK – отмену всех уже выполненных в файле изменений (возврат к последней точке принятия). Описание функций COMMIT или ROLLBACK смотрите в разделе “Единица работы TSO (пакетной и диалоговой)” на стр. 393 или в разделе Глава 6 книги *DB2 SQL Reference*.

Если вы зададите в этом поле N, DB2 после выполнения операторов SQL из входного набора данных выводит панель SPUFI COMMIT OR ROLLBACK. Эта панель предлагает вам принять (COMMIT), откатить (ROLLBACK) или отложить принятие (DEFER) изменений, произведенных операторами SQL. Если вы введете DEFER, не производится ни принятия, ни отката изменений.

#### 9 BROWSE OUTPUT (ПРОСМОТР ВЫВОДА)

Чтобы просмотреть результаты запроса, оставьте в строке 9 Y(YES). SPUFI сохраняет результаты в выходном наборе данных. Их можно посмотреть в любое время, пока вы не сотрете или перезапишете этот набор данных. Более подробную информациюсмотрите в разделе "Формат результатов оператора SELECT" на стр. 101.

#### 10 CONNECT LOCATION (СОЕДИНЬСЯ С ПОЛОЖЕНИЕМ)

Задает имя сервера прикладных программ, если это требуется, которому вы хотите передать операторы SQL. После этого SPUFI передает этому серверу оператор CONNECT типа 2 .

SPUFI – локально связываемый пакет. Операторы SQL во входных данных могут быть обработаны только если оператор CONNECT будет выполнен успешно. Если требование на соединение не будет успешно выполнено, выходной набор данных будет содержать полученные коды возврата SQL и сообщения об ошибках.

## Изменение умолчаний SPUFI (не обязательное)

Закончив работу с панелью SPUFI, нажмите клавишу ENTER. Если вы задали в строке панели SPUFI YES, вы увидите панель умолчаний SPUFI. При первом вызове SPUFI для всех опций, кроме имени подсистемы DB2, будут заданы значения по умолчанию. Любые изменения этих значений остаются в силе, пока вы не измените значения еще раз. На рис. 5 показаны начальные значения по умолчанию.

```
DSNESP02          CURRENT SPUFI DEFAULTS          SSID: DSN
====>
Enter the following to control your SPUFI session:
 1 SQL TERMINATOR    ==> ;      (SQL Statement Terminator)
 2 ISOLATION LEVEL   ==> RR     (RR=Repeatable Read, CS=Cursor Stability)
 3 MAX SELECT LINES  ==> 250   (Maximum number of lines to be
                                returned from a SELECT)

Output data set characteristics:
 4 RECORD LENGTH ... ==> 4092  (LRECL= logical record length)
 5 BLOCKSIZE .....   ==> 4096  (Size of one block)
 6 RECORD FORMAT.... ==> VB    (RECFM= F, FB, FBA, V, VB, or VB)
 7 DEVICE TYPE..... ==> SYSDA (Must be a DASD unit name)

Output format characteristics:
 8 MAX NUMERIC FIELD ==> 33   (Maximum width for numeric field)
 9 MAX CHAR FIELD .. ==> 80   (Maximum width for character field)
10 COLUMN HEADING .. ==> NAMES (NAMES, LABELS, ANY, or BOTH)

PRESS: ENTER to process    END to exit    HELP for more information
```

Рисунок 5. Панель SPUFI defaults (умолчания SPUFI)

На панели CURRENT SPUFI DEFAULTS задайте значения для следующих опций. Все поля должны быть заполнены.

1 SQL TERMINATOR (СИМВОЛ—ОГРАНИЧИТЕЛЬ SQL)

Позволяет задать символ, которым будет завершаться каждый оператор SQL. Можно задать любой символ, кроме перечисленных в Табл. 5. По умолчанию используется точка с запятой.

Таблица 5. Специальные символы, недопустимые как ограничители SQL

Имя	Символ	Шестнадцатеричный код
пробел		X'40'
запятая	,	X'5E'
двойная кавычка	"	X'7F'
левая скобка	(	X'4D'
правая скобка	)	X'5D'
апостроф	'	X'7D'
подчеркивание	_	X'6D'

Точку с запятой нельзя использовать, если будет выполняться оператор, сам содержащий точку с запятой. Предположим, вы выбрали в качестве ограничителя операторов символ #. Тогда оператор CREATE TRIGGER с вложенной точкой с запятой будет выглядеть следующим образом:

```
CREATE TRIGGER NEW_HIRe
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

При выборе символа—ограничителя оператора необходимо, чтобы этот символ не использовался в операторе.

2 ISOLATION LEVEL (УРОВЕНЬ ИЗОЛЯЦИИ)

Позволяет задавать уровень изоляции для ваших операторов SQL.

Дополнительную информациюсмотрите в разделе “Опция ISOLATION” на стр. 373.

3 MAX SELECT LINES (МАКСИМАЛЬНОЕ ЧИСЛО СТРОК, ВОЗВРАЩАЕМЫХ SELECT)

Максимальное число выходных строк, которое может возвратить оператор SELECT. Чтобы ограничить число возвращаемых строк, задайте новое максимальное число больше 1.

4 RECORD LENGTH (ДЛИНА ЗАПИСИ)

Длина записи должна быть не меньше 80 байт. Максимальная длина записи зависит от используемого устройства. Значение по умолчанию допускает записи длиной 4092 байт.

В каждой записи сможет содержаться одна строка вывода. Если строка длиннее записи, последние поля строки усекаются. SPUFI отвергает поля, размер которых превышает длину записи.

5 BLOCKSIZE (РАЗМЕР БЛОКА)

Следуйте обычным правилам выбора размера блока. Для формата записи F размер блока равен длине записи. Для FB и FBA размер блока

должен быть кратен удвоенной длине записи. Для VB и VBA размер блока должен быть на 4 байта больше размера блока для FB и FBA.

6 RECORD FORMAT (ФОРМАТ ЗАПИСИ)

Задайте F, FB, FBA, V, VB или VBA. В форматах FBA и VBA после числа строк, заданного в поле LINES/PAGE OF LISTING панели умолчаний DB2I, вставляется символ управления принтером. По умолчанию используется формат VB (блокированный, с переменной длиной).

7 DEVICE TYPE (ТИП УСТРОЙСТВА)

Позволяет задать стандартное имя MVS для устройства хранения прямого доступа. По умолчанию используется SYSDA. SYSDA указывает, что MVS должна выбрать подходящее устройство хранения прямого доступа.

8 MAX NUMERIC FIELD (МАКСИМАЛЬНОЕ ЧИСЛОВОЕ ПОЛЕ)

Максимальная ширина числового столбца в вашем выводе. Выберите значение больше нуля. По умолчанию IBM использует значение 20.

Более подробную информацию смотрите в разделе “Формат результатов оператора SELECT” на стр. 101.

9 MAX CHAR FIELD (МАКСИМАЛЬНОЕ СИМВОЛЬНОЕ ПОЛЕ)

Максимальная ширина символьного столбца в вашем выводе. Строки данных DATETIME и GRAPHIC при выводе представляются как символьные, и SPUFI использует для них те же значения по умолчанию, что и для символьных полей. Выберите значение больше нуля. По умолчанию IBM использует значение 80. Более подробную информацию смотрите в разделе “Формат результатов оператора SELECT” на стр. 101.

10 COLUMN HEADING (ЗАГОЛОВКИ СТОЛБЦА)

Можно задать для заголовков столбцов значения NAMES, LABELS, ANY или BOTH.

- NAME (используется по умолчанию) выводит только имена столбцов.
- LABEL использует в качестве заголовков надписи. Если у столбца нет надписи, заголовок останется пустым.
- ANY использует надписи столбцов или имена столбцов.
- BOTH создает двухстроковый заголовок с именем и надписью.

Имена столбцов – это те идентификаторы, которые вы используете в операторах SQL. Если в операторе SQL для столбца указано условие AS, SPUFI выводит в заголовке вместо имени столбца содержимое условия AS. Надписи для столбцов вы можете определять в операторах LABEL ON.

Введя опции SPUFI, нажмите клавишу ENTER, чтобы продолжить работу. Затем SPUFI обрабатывает следующую опцию, для которой вы задали YES. Если все оставшиеся опции обработки – NO, SPUFI выводит панель SPUFI.

Если вы нажмете клавишу END, вы вернетесь к панели SPUFI, но все изменения, внесенные на панели умолчаний SPUFI, будут утеряны. Если вы нажмете ENTER, SPUFI сохранит ваши изменения.

## Ввод операторов SQL

Далее SPUFI позволяет редактировать входной набор данных. Сначала вы вводите операторы SQL во входной набор данных. Можно также редактировать входной набор данных, содержащий операторы SQL, изменять, удалять и вставлять операторы SQL.

Редактор ISPF выводит пустую панель EDIT.

На этой панели при помощи программы ISPF EDIT введите операторы SQL, которые хотите выполнить, как показано на рис. 6.

Поставьте курсор на первую строку ввода и введите первую часть оператора SQL. Остальную часть оператора SQL можно ввести на следующих строках, как показано на рис. 6. Разбивка операторов по строкам облегчает чтение и не влияет на их обработку.

Во входной набор данных можно поместить несколько операторов SQL. Один оператор SQL можно записать на одной или на нескольких строчках. DB2 выполняет операторы в том порядке, в котором вы расположили их во входном наборе. Не записывайте на одной строке несколько операторов SQL. Первый из них будет выполнен, но остальные операторы SQL на той же строке DB2 проигнорирует.

При использовании SPUFI завершайте каждый оператор SQL точкой с запятой (;). Это говорит SPUFI, что оператор закончен.

После ввода операторов SQL нажмите клавишу END PF, чтобы сохранить файл и выполнить операторы SQL.

```
EDIT -----userid.EXAMPLES(XMP1) ----- COLUMNS 001 072
COMMAND INPUT ==> SAVE                                SCROLL ==> PAGE
*****TOP OF DATA*****                                *****
000100 SELECT LASTNAME, FIRSTNAME, PHONENO
000200   FROM DSN8610.EMP
000300 WHERE WORKDEPT= 'D11'
000400 ORDER BY LASTNAME;
*****BOTTOM OF DATA*****
```

Рисунок 6. Панель edit после ввода оператора SQL

Нажатие клавиши END PF сохраняет набор данных. Чтобы сохранить набор данных и продолжить редактирование, введите команду SAVE. Рекомендуем сохранять набор данных примерно после каждого 10 минут редактирования.

На рис. 6 показано, как выглядит панель после ввода оператора SQL примера и выполнения команды SAVE.

Этап редактирования можно пропустить, если изменить значение опции обработки EDIT INPUT:

```
EDIT INPUT ... ==> NO
```

Комментарии к операторам SQL можно добавлять как на отдельных строках, так и на той же строке. В каждом варианте комментарий должен начинаться с двух дефисов (—). Все, что стоит справа от двух дефисов, DB2 игнорирует.

---

## Обработка операторов SQL

SPUFI передает входной набор данных DB2 для обработки. DB2 выполняет операторы SQL во входном наборе данных EXAMPLES(XMP1) и направляет вывод в выходной набор данных *id\_пользователя.RESULT*.

Этап обработки операторов DB2 можно пропустить, если изменить значение опции обработки EXECUTE INPUT:

```
EXECUTE ..... ==> NO
```

Оператор SQL может выполняться довольно долго, в зависимости от величины таблицы DB2, где идет поиск, и от количества обрабатываемых DB2 строк. Чтобы прервать обработку DB2, нажмите клавишу PA1 и ответьте на вопрос, действительно ли вы хотите остановить обработку. При этом выполнение оператора SQL будет отменено и вы вернетесь в меню ISPF-PDF.

Что при этом произойдет с выходным набором данных? Это зависит от того, какую часть входного набора данных DB2 успела обработать к моменту прерывания обработки. DB2 могла еще не успеть открыть выходной набор данных, могла записать в него часть результатов или же все результаты.

---

## Просмотр вывода

SPUFI форматирует и выводит выходной набор данных при помощи программы ISPF Browse. На рис. 7 на стр. 101 показан вывод программы примера. Для каждого оператора SQL, выполненного DB2, выходной набор данных будет содержать:

- Выполняемый оператор SQL, скопированный из входного набора данных
- Результаты выполнения оператора SQL
- SQLCA в форматированном виде, если при выполнении произошли ошибки

В конце набора данных приводится статистическая сводка, описывающая обработку входного набора данных в целом.

При выполнении оператора SELECT через SPUFI сообщение “SQLCODE IS 100” указывает на отсутствие ошибок. Если, кроме сообщения SQLCODE IS 100, других результатов нет, DB2 не смогла найти никаких строк, удовлетворяющих заданным в операторе критериям.

Для всех прочих типов операторов SQL, выполняемых через SPUFI, на отсутствие ошибок указывает сообщение “SQLCODE IS 0.”

```

BROWSE-- id пользователя.RESULT          COLUMNS 001 072
COMMAND INPUT ==>                      SCROLL ==> PAGE
-----+-----+-----+-----+-----+-----+
SELECT LASTNAME, FIRSTNME, PHONENO      00010000
     FROM DSN8610.EMP                   00020000
      WHERE WORKDEPT = 'D11'            00030000
      ORDER BY LASTNAME;              00040000
-----+-----+-----+-----+-----+-----+
LASTNAME      FIRSTNME    PHONENO
ADAMSON        BRUCE       4510
BROWN          DAVID       4501
JOHN           REBA        0672
JONES          WILLIAM    0942
LUTZ           JENNIFER   0672
PIANKA         ELIZABETH  3782
SCOUTTEN       MARILYN    1682
STERN          IRVING     6423
WALKER         JAMES      2986
YAMAMOTO       KIYOSHI    2890
YOSHIMURA     MASATOSHI  2890
DSNE610I NUMBER OF ROWS DISPLAYED IS 11
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 4
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 30

```

Рисунок 7. Набор данных результатов для задачи примера

## Формат результатов оператора SELECT

Результаты операторов SELECT подчиняются следующим правилам:

- Если числовые или символьные данные столбца невозможно вывести полностью:
  - Слишком длинные символьные значения усекаются справа.
  - Слишком длинные числовые значения выводятся в виде звездочек (\*).
  - Для всех столбцов, кроме столбцов больших объектов, при усечении в выходной набор данных выводится предупреждающее сообщение. Поскольку длины столбцов больших объектов, как правило, больше значений, задаваемых в поле MAX CHAR FIELD панели CURRENT SPUFI DEFAULTS, SPUFI не выводит при их усечении предупреждений.

Изменить количество данных, выводимое для числовых и символьных столбцов, можно, задав новые значения на панели CURRENT SPUFI DEFAULTS, как описано в разделе “Изменение умолчаний SPUFI (не обязательное)” на стр. 96.

- Пустые значения выводятся в виде дефисов (–). (Пустые значения описаны в разделе “Выбор строк с пустыми значениями” на стр. 27.)
- Значения в столбцах ROWID и BLOB выводятся в шестнадцатеричном виде.
- Значения в столбцах CLOB выводятся так же, как значения в столбцах VARCHAR.

- Значения в столбцах DBCLOB выводятся так же, как значения в столбцах VARGRAPHIC.
- Каждый выбранный столбец обозначается своим заголовком; заголовки повторяются в начале каждой страницы вывода. Содержимое заголовков зависит от значения, заданного в поле COLUMN HEADING на панели CURRENT SPUFI DEFAULTS.

## Содержимое сообщений

Каждое сообщение содержит следующие элементы:

- SQLCODE, если оператор выполнен успешно
- SQLCA в форматированном виде, если при выполнении оператора произошли ошибки
- Какие символьные позиции входного набора данных SPUFI просматривает при поиске операторов SQL. Эта информация поможет проверить, правильно ли SPUFI определила положение номеров строк (если они были) во входном наборе данных.
- В статистическую сводку входит:
  - Число обработанных операторов SQL
  - Число прочитанных (из входного набора данных) входных записей
  - Число записанных (в выходной набор данных) выходных записей.

При обработке операторов SQL вы можете получать и другие сообщения:

- Число строк таблицы, обработанных DB2, то есть:
  - Возвращенных оператором SELECT, или
  - Измененных оператором UPDATE, или
  - Добавленных в таблицу оператором INSERT, или
  - Удаленных из таблицы оператором DELETE
- Сообщения об усечении данных в столбце из-за превышения ширины

---

## Раздел 3. Кодирование SQL в прикладных программах хоста

<b>Глава 3–1. Основы кодирования операторов SQL в прикладной программе</b>	107
Общие замечания по примерам написания операторов SQL	108
Разделители операторов SQL	109
Объявление определений таблицы и производной таблицы	109
Обращение к данным с помощью переменных хоста и структур хоста	110
Использование переменных хоста	111
Использование структур хоста	115
Проверка выполнения операторов SQL	116
SQLCODE и SQLSTATE	117
Оператор WHENEVER	117
Обработка арифметических ошибок и ошибок преобразования	118
Обработка кодов возврата ошибок SQL	119
<b>Глава 3–2. Использование указателей для получения наборов строк</b>	123
Функции указателя	123
Пример использования указателя	124
Шаг 1: Определение указателя	125
Шаг 2: Открытие указателя	126
Шаг 3: Задание реакции на конец данных	127
Шаг 4: Получение строки при помощи указателя	127
Шаг 5a: Изменение текущей строки	128
Шаг 5b: Удаление текущей строки	129
Шаг 6: Закрытие указателя	129
Объявление указателя с опцией WITH HOLD	129
<b>Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN</b>	133
Вызов DCLGEN через DB2I	134
Включение объявлений данных в вашу программу	138
Поддержка DCLGEN для языков C, COBOL и PL/I	138
Пример: Добавление объявления таблицы и структуры переменных хоста к библиотеке	140
Шаг 1. Задайте COBOL как язык хоста	140
Шаг 2. Создание объявления таблицы и структуры хоста	141
Шаг 3. Проверка результатов	142
<b>Глава 3–4. Встроенные операторы SQL в языках хоста</b>	145
Кодирование операторов SQL в программе на языке ассемблер	145
Задание области связи SQL	145
Определение областей дескрипторов SQL	146
Вставка операторов SQL	147
Использование переменных хоста	150
Объявление переменных хоста	151
Определение эквивалентных типов данных SQL и ассемблера	153
Определение совместимости типов данных SQL и ассемблера	157
Использование переменных–индикаторов	158
Обработка кодов возврата ошибок SQL	159
Макрокоманды для программ на ассемблере	160
Кодирование операторов SQL в программах на языках С или С <sup>++</sup>	160

Определение области связи SQL . . . . .	161
Определение областей дескрипторов SQL . . . . .	161
Встраивание операторов SQL . . . . .	162
Использование переменных хоста . . . . .	164
Объявление переменных хоста . . . . .	165
Применение структур хоста . . . . .	169
Определение эквивалентных типов данных в SQL и C . . . . .	171
Определение совместимости типов данных SQL и C . . . . .	177
Применение переменных–индикаторов . . . . .	178
Обработка кодов возврата ошибок SQL . . . . .	179
Особенности C++ . . . . .	181
Кодирование операторов SQL в программах на языке COBOL . . . . .	181
Задание области связи SQL . . . . .	181
Определение областей дескрипторов SQL . . . . .	183
Вставка операторов SQL . . . . .	183
Использование переменных хоста . . . . .	188
Объявление переменных хоста . . . . .	188
Использование структур хоста . . . . .	193
Определение эквивалентных типов данных SQL и COBOL . . . . .	195
Определение совместимости типов данных SQL и языка COBOL . . . . .	200
Использование переменных–индикаторов . . . . .	201
Обработка кодов возврата ошибок SQL . . . . .	202
Особенности объектно–ориентированных расширений в языке COBOL	204
Кодирование операторов SQL в программах на языке FORTRAN . . . . .	204
Задание области связи SQL . . . . .	204
Определение областей дескрипторов SQL . . . . .	205
Вставка операторов SQL . . . . .	206
Использование переменных хоста . . . . .	208
Объявление переменных хоста . . . . .	209
Определение эквивалентности типов данных SQL и FORTRAN . . . . .	210
Определение совместимости типов данных SQL и FORTRAN . . . . .	213
Использование переменных–индикаторов . . . . .	214
Обработка кодов возврата ошибок SQL . . . . .	215
Кодирование операторов SQL в программах на языке PL/I . . . . .	215
Задание области связи SQL . . . . .	216
Определение областей дескрипторов SQL . . . . .	216
Вставка операторов SQL . . . . .	217
Использование переменных хоста . . . . .	220
Объявление переменных хоста . . . . .	221
Использование структур хоста . . . . .	223
Определение эквивалентности типов данных SQL и PL/I . . . . .	224
Определение совместимости типов данных SQL и PL/I . . . . .	229
Использование переменных–индикаторов . . . . .	229
Обработка кодов возврата ошибок SQL . . . . .	230
<b>Глава 3–5. Использование триггеров для работы с активными данными . . . . .</b>	<b>233</b>
Пример создания и использования триггера . . . . .	233
Составные части триггера . . . . .	235
Вызов хранимых процедур и пользовательских функций из триггеров . . . . .	240
Передача таблиц переходов пользовательским функциям и хранимым процедурам . . . . .	241
Каскадное срабатывание триггеров . . . . .	242
Порядок активации триггеров . . . . .	243

Взаимодействие между триггерами и реляционными связями . . . . .	244
Создание триггеров, дающих надежные результаты . . . . .	245



---

## Глава 3–1. Основы кодирования операторов SQL в прикладной программе

Предположим, вы пишете прикладную программу, которая обращается к данным из базы данных DB2. Когда программа выполняет оператор SQL, она должна связаться с DB2. Когда DB2 завершает обработку оператора SQL, она посыпает код возврата; ваша программа должна проверить этот код возврата, чтобы определить результат операции.

Чтобы связаться с DB2, надо:

- Выбрать метод связи с DB2. Можно использовать один из следующих методов:
  - Статический SQL
  - Встроенный динамический SQL
  - Open Database Connectivity (ODBC)
  - Поддержка прикладных программ JDBC

В этой книге описывается встроенный SQL. Сравнение статического и встроенного динамического SQL и подробное обсуждение встроенного динамического SQL смотрите в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.

ODBC позволяет обращаться к данным при помощи вызовов функций ODBC в вашей программе. Вы выполняете операторы SQL, передавая их DB2 при помощи вызовов функций ODBC. ODBC позволяет избежать препомпилияции и связывания вашей прикладной программы и повышает переносимость программ за счет использования интерфейса ODBC.

Если вы пишете свои программы на Java, для обращения к DB2 можно использовать поддержку JDBC. JDBC подобна ODBC, но разработана специально для использования с Java, поэтому для вызовов DB2 из прикладных программ Java лучше использовать JDBC.

Дополнительную информацию об использовании JDBC смотрите в книге *DB2 ODBC Guide and Reference*.

- Задайте ограничитель операторов SQL, как описано в разделе “Разделители операторов SQL” на стр. 109.
- Объявите используемые вами таблицы, как описано в разделе “Объявление определений таблицы и производной таблицы” на стр. 109. (Это необязательный шаг.)
- Объявите элементы данных, используемые для передачи данных между DB2 и языком хоста, как описано в разделе “Обращение к данным с помощью переменных хоста и структур хоста” на стр. 110.
- Запишите операторы SQL, обращающиеся к данным DB2. Смотрите раздел “Обращение к данным с помощью переменных хоста и структур хоста” на стр. 110.

Информацию об использовании языка SQL смотрите в разделе “Раздел 2. Использование запросов SQL” на стр. 17 и в книге *DB2 SQL Reference*. Особенности использования операторов SQL в прикладной программе описаны в разделе “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145.

- Объявите область связи (SQLCA), или обрабатывайте исключительные ситуации, которые возвращает DB2 с кодами возврата, в SQLCA. Подробности смотрите в разделе “Проверка выполнения операторов SQL” на стр. 116.

Кроме этих основных шагов, надо рассмотреть также несколько специальных вопросов:

- В разделе “Глава 3–2. Использование указателей для получения наборов строк” на стр. 123 описано использование указателя в прикладной программе для выбора набора строк и обработке их по строке за раз.
- В разделе “Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN” на стр. 133 описано, как использовать генератор объявлений DB2 – DCLGEN – для генерации правильных операторов SQL DECLARE для таблиц и производных таблиц.

В этом разделе излагается информация об использовании SQL в прикладных программах на языках ассемблер, С, COBOL, FORTRAN и PL/I. SQL можно использовать также в прикладных программах на языках Ada, APL2®, BASIC и Prolog. Дополнительную информацию об этих языках смотрите в следующих публикациях:

<b>Ada</b>	<i>IBM Ada/370 SQL Module Processor for DB2 Database Manager User's Guide</i>
<b>APL2</b>	<i>APL2 Programming: Using Structured Query Language (SQL)</i>
<b>BASIC</b>	<i>IBM BASIC/MVS Language Reference</i>
<b>Prolog/MVS &amp; VM</b>	<i>IBM SAA AD/Cycle® Prolog/MVS &amp; VM Programmer's Guide</i>

## Общие замечания по примерам написания операторов SQL

Для операторов SQL, показанных в этом разделе, применяются следующие соглашения:

- Оператор SQL входит в прикладную программу на языке COBOL. Примеры SQL состоят из нескольких строк, причем каждое условие оператора занимает отдельную строку.
- Предполагается использование опций препроцессора APOST и APOSTSQL (хотя они не принимаются по умолчанию). Поэтому для ограничения символьных литералов в операторах SQL и языка хоста используются апострофы (').
- Операторы SQL обращаются к данным в таблицах примера, поставляемых вместе с DB2. В таблицах содержатся данные некоторой компании, производящей товары, о ее сотрудниках и текущих проектах. Описание таблиц смотрите в разделе Приложение А, “Таблицы примеров DB2” на стр. 863.
- В примерах SQL синтаксис операторов SQL может даваться не полностью. Глава 6 *DB2 SQL Reference* содержит подробное описание и сведения по синтаксису всех операторов, рассматриваемых в этой книге.
- Приведенные примеры не учитывают реляционные ограничения. Дополнительную информацию о влиянии реляционных ограничений на операторы SQL и примеры работы операторов SQL с реляционными

ограничениями смотрите в разделе “Глава 2–2. Работа с таблицами и изменение данных” на стр. 57.

Как исключение, примеры могут не удовлетворять перечисленным выше условиям. Такие случаи оговариваются специально.

---

## Разделители операторов SQL

Оператор SQL в вашей программе должен быть заключен между ключевым словом EXEC SQL и ограничителем оператора. В языках, описанных в этой книге, приняты следующие ограничители:

Язык	Ограничитель оператора SQL
<b>Ассемблер</b>	Конец строки или конец последней из строк продолжения
<b>C</b>	Точка с запятой (;
<b>COBOL</b>	END-EXEC
<b>FORTRAN</b>	Конец строки или конец последней из строк продолжения
<b>PL/I</b>	Точка с запятой (;

Например, для ограничения оператора SQL в программе на COBOL следует использовать ключевые слова EXEC SQL и END-EXEC:

```
EXEC SQL  
operator SQL  
      END-EXEC.
```

---

## Объявление определений таблицы и производной таблицы

До выполняемых операторов SQL для получения, изменения, удаления или вставки данных в вашей программе надо объявить таблицы и производные таблицы, к которым она обращается. Для этого включите в вашу программу оператор SQL DECLARE.

Объявлять таблицы или производные таблицы нет необходимости, но это дает определенные преимущества. Одно из преимуществ – возможность справок. Например, в операторе DECLARE указана структура таблицы или производной таблицы, с которыми вы работаете, а также тип данных для каждого столбца. Чтобы посмотреть имена столбцов и типы данных в таблице или производной таблице, достаточно посмотреть оператор DECLARE. Другое преимущество состоит в том, что ваши объявления используются препроцессором DB2 для проверки правильности имен столбцов и типов данных в ваших операторах SQL. Если имена столбцов и типы данных не соответствуют операторам SQL DECLARE в вашей программе, препроцессор DB2 выдает предупреждение.

Для объявления таблицы или производной таблицы надо вставить оператор DECLARE в раздел WORKING-STORAGE SECTION или LINKAGE SECTION части DATA DIVISION вашей программы на языке COBOL. Укажите имя нужной таблицы и перечислите все столбцы вместе с их типами данных. При объявлении таблицы или производной таблицы указывается DECLARE имя таблицы TABLE независимо от того, относится ли это имя к таблице или к производной таблице.

Например, оператор DECLARE TABLE для таблицы DSN8610.DEPT выглядит так:

```
EXEC SQL
DECLARE DSN8610.DEPT TABLE
  (DEPTNO  CHAR(3)          NOT NULL,
   DEPTNAME VARCHAR(36)      NOT NULL,
   MGRNO    CHAR(6)          ,
   ADMRDEPT CHAR(3)          NOT NULL,
   LOCATION  CHAR(16)        )
END-EXEC.
```

Вместо того, чтобы самостоятельно вставлять в программу оператор DECLARE, можно воспользоваться генератором объявлений DCLGEN, поставляемым с DB2. Дополнительную информацию о пользовании DCLGEN смотрите в разделе “Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN” на стр. 133.

При объявлении таблицы или производной таблицы, содержащей столбец с особым пользовательским типом, лучше объявить этот столбец с исходным типом пользовательского типа, а не с самим пользовательским типом. Когда столбец объявляется с исходным типом, DB2 может проверить встроенные операторы SQL, где данный столбец упоминается, во время прекомпиляции.

## Обращение к данным с помощью переменных хоста и структур хоста

Обращение к данным возможно с помощью переменных хоста и структур хоста.

*Переменная хоста* – это элемент данных, объявленный в языке хоста для использования в операторе SQL. Переменные хоста позволяют:

- Заносить данные в переменную хоста для вашей прикладной программы
- Помещать данные в переменную хоста для вставки в таблицу или для изменения содержимого строки
- Использовать данные в переменной хоста при оценке условия WHERE или HAVING
- Заносить значение переменной хоста в специальный регистр, например, CURRENT SQLID или CURRENT DEGREE
- Вставлять пустые значения в столбцы с помощью индикаторной переменной хоста, принимающей отрицательное значение
- Использовать данные в переменной хоста в операторах обработки динамического SQL, таких как EXECUTE, PREPARE и OPEN

*Структура хоста* – это группа переменных хоста, к которым оператор SQL может обращаться, используя только одно имя. Для определения структур хоста используются операторы языка хоста.

## Использование переменных хоста

В операторе SQL может использоваться любое допустимое имя переменной хоста. Прежде чем использовать это имя в программе хоста, необходимо объявить его. (Дополнительную информацию для каждого из рассмотренных языков смотрите в соответствующем месте в разделе “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145.)

Для повышения производительности проследите, чтобы объявление на языке хоста как можно точнее соответствовало типу ассоциированных с ним данных в базе данных; смотрите “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145. Дополнительные рекомендации по настройке производительности смотрите в разделе “Раздел 7. Дополнительные приемы программирования” на стр. 537.

С помощью переменной хоста можно представлять значения данных, но ее нельзя использовать для представления таблицы, производной таблицы или имени столбца. (Таблицу, производную таблицу, или имена столбцов можно задать во время выполнения с помощью динамического SQL.

Дополнительную информацию смотрите в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.)

Переменные хоста подчиняются условиям образования имен в языке хоста. Чтобы сообщить DB2, что данная переменная не является именем столбца, перед переменными хоста в SQL должно стоять двоеточие (:). Двоеточие не ставится перед переменными хоста вне операторов SQL.

За дополнительной информацией об объявлении переменных хоста в разных языках обращайтесь к соответствующим разделам:

- *Ассемблер*: “Использование переменных хоста” на стр. 150
- *C*: “Использование переменных хоста” на стр. 164
- *COBOL*: “Использование переменных хоста” на стр. 188
- *FORTRAN*: “Использование переменных хоста” на стр. 208
- *PL/I*: “Использование переменных хоста” на стр. 220.

## Занесение данных в переменную хоста

Переменную хоста можно использовать для указания области данных в программе, где должны содержаться значения столбцов полученных строк.

**Получение одной строки данных:** Условие INTO оператора SELECT дает имена одной или нескольким переменным хоста, где должны содержаться возвращенные значения столбцов. Имена переменных находятся во взаимно–однозначном соответствии со списком имен столбцов в операторе SELECT.

Предположим, например, что вы получаете значения столбцов EMPNO, LASTNAME, и WORKDEPT из строк в таблице DSN8610.EMP. Определите в вашей программе область данных для каждого столбца, затем с помощью условия INTO присвойте областям данных имена, как показано в следующем примере. (Обратите внимание, что перед каждой переменной хоста стоит двоеточие):

```

        EXEC SQL
SELECT EMPNO, LASTNAME, WORKDEPT
      INTO :CBLEMPNO, :CBLNAME, :CBLDEPT
     FROM DSN8610.EMP
    WHERE EMPNO = :EMPID
END-EXEC.

```

Чтобы переменные хоста CBLEMPNO, CBLNAME, и CBLDEPT были совместимы с типами данных в столбцах EMPNO, LASTNAME, и WORKDEPT таблицы DSN8610.EMP, их необходимо объявить в разделе данных вашей программы.

Если оператор SELECT возвращает несколько строк, происходит ошибка, и любые возвращенные данные будут не определены и непредсказуемы.

**Получение наборов строк данных:** Если число строк, которые должна возвратить DB2, неизвестно, или ожидается возврат нескольких строк, необходимо вместо оператора SELECT ... воспользоваться оператором INTO.

Указатель DB2 дает возможность прикладной программе обрабатывать набор строк, считывая из таблицы результатов по одной строке за раз. Информацию по пользованию указателямисмотрите в разделе “Глава 3–2. Использование указателей для получения наборов строк” на стр. 123.

**Перечисление элементов данных в условии выбора:** Задавая список элементов данных в условии SELECT, можно использовать не только имена столбцов в таблицах и производных таблицах. Можно затребовать также значения столбцов в одном наборе со значениями переменной хоста и константами. Например:

```

MOVE 4476 TO RAISE.
MOVE '000220' TO PERSON.
        EXEC SQL
SELECT EMPNO, LASTNAME, SALARY, :RAISE, SALARY + :RAISE
      INTO :EMP-NUM, :PERSON-NAME, :EMP-SAL, :EMP-RAISE, :EMP-TTL
     FROM DSN8610.EMP
    WHERE EMPNO = :PERSON
END-EXEC.

```

Приведенные ниже результаты помещены под заголовками столбцов, представляющих имена переменных хоста:

EMP-NUM	PERSON-NAME	EMP-SAL	EMP-RAISE	EMP-TTL
000220	LUTZ	29840	4476	34316

### **Вставка и изменение данных**

Переменную хоста можно использовать для задания или изменения значения в таблице DB2. Для этого следует использовать имя переменной хоста в условии SET оператора UPDATE или условие VALUES оператора INSERT. В приведенном примере у одного из служащих изменяется номер телефона:

```

        EXEC SQL
UPDATE DSN8610.EMP
  SET PHONENO = :NEWPHONE
 WHERE EMPNO = :EMPID
END-EXEC.

```

## Поиск данных

Переменную хоста можно использовать для указания значения в предикате критерия поиска или вместо константы в выражении. Например, определив поле EMPID, содержащее номер служащего, можно затем получить имя служащего с номером 000110:

```
MOVE '000110' TO EMPID.  
      EXEC SQL  
      SELECT LASTNAME  
        INTO :PGM-LASTNAME  
        FROM DSN8610.EMP  
       WHERE EMPNO = :EMPID  
      END-EXEC.
```

## Использование индикаторных переменных с переменными хоста

Индикаторные переменные – это числа типа small integer, с помощью которых можно:

- Установить, не является ли значение связанной с ним переменной хоста на выводе пустым, или указать на пустое значение переменной хоста на входе
- Определить исходную длину строки символов, усеченной при назначении переменной хоста
- Определить, что символьное значение не могло быть преобразовано при назначении переменной хоста
- Восстановить часть значения времени, соответствующую секундам и усеченную при назначении переменной хоста

**Занесение данных в переменные хоста:** Если в получаемом вами столбце содержится пустое значение, DB2 присваивает индикаторной переменной отрицательное значение. Если пустое значение появилось из-за ошибки преобразования числа или символа или ошибки в арифметическом выражении, DB2 присваивает индикаторной переменной значение –2. Дополнительную информациюсмотрите в разделе “Обработка арифметических ошибок и ошибок преобразования” на стр. 118.

Если индикаторная переменная не используется, а DB2 получает пустое значение, происходит ошибка.

Когда DB2 получает значение столбца, можно проверить индикаторную переменную. Отрицательное значение индикаторной переменной соответствует пустому значению (null) столбца. При пустом значении (null) столбца переменная хоста сохраняет предыдущее значение.

С помощью индикаторной переменной можно также проверить, не усечена ли при получении строка символов. Если индикаторная переменная принимает положительное целое значение, это целое представляет собой исходную длину строки.

Индикаторную переменную, следующую за двоеточием, можно задать сразу же после переменной хоста. При желании можно воспользоваться словом INDICATOR между переменной хоста и его индикаторной переменной. Таким образом, следующие два примера эквивалентны:

```

EXEC SQL
SELECT PHONENO
  INTO :CBLPHONE:INDNULL
   FROM DSN8610.EMP
  WHERE EMPNO = :EMPID
    END-EXEC.

EXEC SQL
SELECT PHONENO
  INTO :CBLPHONE INDICATOR :INDNULL
   FROM DSN8610.EMP
  WHERE EMPNO = :EMPID
    END-EXEC.

```

После этого следует проверить знак INDNULL. Если ее значение отрицательно, ему соответствует пустое значение PHONENO, а на содержимое CBLPHONE можно не обращать внимания.

Если для чтения значения столбца используется указатель, тем же методом можно определить, является ли значение столбца пустым (null).

**Вставка пустых значений (null) в столбцы с помощью переменных хоста:**  
Индикаторную переменную можно использовать для вставки пустого значения (null) из переменной хоста в столбец. При обработке операторов INSERT и UPDATE DB2 проверяет индикаторную переменную (если она существует). Если индикаторная переменная отрицательна, столбец содержит пустое значение (null). Если значение индикаторной переменной больше –1, связанная с ней переменная хоста содержит для данного столбца некоторое непустое значение.

Предположим, например, что ваша программа читает ID служащего и новый номер его телефона, который требуется занести в таблицу его данных вместо старого. Новый номер может отсутствовать, если старый номер был неверен, а новый еще не получен. Если новое значение для столбца PHONENO может быть пустым (null), можно написать:

```

EXEC SQL
UPDATE DSN8610.EMP
  SET PHONENO = :NEWPHONE:PHONEIND
 WHERE EMPNO = :EMPID
   END-EXEC.

```

Если NEWPHONE содержит непустое значение, задайте для PHONEIND нулевое значение, вставив перед оператором:

MOVE 0 TO PHONEIND.

Если NEWPHONE содержит пустое значение, задайте для PHONEIND отрицательное значение, вставив перед оператором:

MOVE -1 TO PHONEIND.

### **Назначения и сравнения при различных типах данных**

При назначениях и сравнениях, затрагивающих некоторый столбец DB2 и переменную хоста с различными типом и длиной данных, может потребоваться преобразование данных. Глава 3 *DB2 SQL Reference* описывает правила для операций назначения и сравнения данных.

## Использование структур хоста

Вместо структуры хоста можно использовать одну или несколько переменных хоста. Со структурами хоста можно также использовать индикаторные переменные (или структуры).

### Пример: Использование структуры хоста

В приведенном ниже примере предполагается, что в вашей программе на COBOL есть следующий оператор SQL:

```
EXEC SQL
SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT
  INTO :EMPNO, :FIRSTNAME, :MIDINIT, :LASTNAME, :WORKDEPT
    FROM DSN8610.VEMP
   WHERE EMPNO = :EMPID
END-EXEC.
```

Если вы не хотите перечислять переменные хоста, используйте имя структуры :PEMP, которая содержит переменные :EMPNO, :FIRSTNAME, :MIDINIT, :LASTNAME, и :WORKDEPT. Тогда данный пример будет выглядеть так:

```
EXEC SQL
SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT
  INTO :PEMP
    FROM DSN8610.VEMP
   WHERE EMPNO = :EMPID
END-EXEC.
```

Структуру хоста можно либо объявить самостоятельно, либо при помощи DCLGEN сгенерировать описание записи на COBOL, объявления структуры в PL/I или объявления структуры в C, которые соответствуют столбцам таблицы. Дополнительные подробности о создании структуры хоста в вашей программе смотрите в разделе “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145. Дополнительную информацию об использовании DCLGEN и ограничениях, касающихся языка C, смотрите в разделе “Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN” на стр. 133.

## Использование индикаторных переменных со структурами хоста

Для поддержки структуры хоста можно определить *индикаторную структуру* (массив переменных типа short integer). Индикаторные структуры определяются в части DATA DIVISION вашей программы на языке COBOL. Если значения столбцов, которые ваша программа заносит в структуру хоста, могут быть пустыми, можно присоединить имя индикаторной структуры к имени структуры хоста. Это позволит DB2 сообщать вашей программе о каждом пустом значении, возвращенном переменной хоста в структуре хоста. Например:

```

01 PEMP-ROW.
  10 EMPNO          PIC X(6).
  10 FIRSTNME.
    49 FIRSTNME-LEN  PIC S9(4) USAGE COMP.
    49 FIRSTNME-TEXT PIC X(12).
  10 MIDINIT        PIC X(1).
  10 LASTNAME.
    49 LASTNAME-LEN  PIC S9(4) USAGE COMP.
    49 LASTNAME-TEXT PIC X(15).
  10 WORKDEPT       PIC X(3).
  10 EMP-BIRTHDATE  PIC X(10).

01 INDICATOR-TABLE.
  02 EMP-IND        PIC S9(4) COMP OCCURS 6 TIMES.
  :
MOVE '000230' TO EMPNO.
  :
      EXEC SQL
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, BIRTHDATE
  INTO :PEMP-ROW:EMP-IND
  FROM DSN8610.EMP
  WHERE EMPNO = :EMPNO
  END-EXEC.

```

В этом примере EMP-IND – массив из шести значений, которые проверяются на знак. Если, например, EMP-IND(6) содержит отрицательное значение, соответствующая переменная хоста в структуре хоста (EMP-BIRTHDATE) содержит пустое значение.

Вследствие того, что в этом примере строки выбираются из таблицы DSN8610.EMP, некоторые из значений в массиве EMP-IND всегда равны нулю. Первые четыре столбца каждой строки определены как NOT NULL. В приведенном выше примере DB2 выбирает значения для строки данных, заносимой в структуру хоста. Чтобы определить, есть ли среди выбранных значений столбцов пустые (null) и если да, то какие именно, надо использовать соответствующую структуру для индикаторных переменных. Сведения об использовании ключевого слова IS NULL в условиях WHERE смотрите в разделе “Глава 2–1. Получение данных” на стр. 19.

## Проверка выполнения операторов SQL

Для программы, содержащей операторы SQL, необходима отдельная область для связи с DB2 — *область связи SQL* (SQLCA). При обработке оператора SQL в вашей программе DB2 помещает коды возврата либо в переменные хоста SQLCODE и SQLSTATE, либо в соответствующие поля SQLCA. Коды возврата показывают, успешно или неудачно завершилось исполнение оператора.

SQLCA – важный инструмент диагностики ошибок, и имеет смысл включать в прикладные программы инструкции для вывода на экран части содержимого SQLCA. Например, может быть полезным содержимое SQLERRD(3), где указывается количество строк, которые DB2 изменила, вставила или удалила. Если в SQLWARN0 содержится W, это значит, что DB2 установила по крайней мере один из флагов предупреждения (с SQLWARN1 по SQLWARNA). Приложение C *DB2 SQL Reference* содержит описание всех полей SQLCA.

## **SQLCODE и SQLSTATE**

При выполнении любого оператора SQL в поля SQLCODE и SQLSTATE области SQLCA записывается код возврата. Хотя в принципе оба поля выполняют одну и ту же роль (указывают, успешно ли прошло выполнение оператора), между ними существует определенная разница.

**SQLCODE:** В SQLCODE DB2 возвращает следующие коды:

- Если SQLCODE = 0, выполнение прошло успешно.
- Если SQLCODE > 0, выполнение прошло успешно, но получено предупреждение.
- Если SQLCODE < 0, выполнение завершилось неудачно.

SQLCODE 100 означает, что не обнаружено данных.

Значения кодов SQLCODE, отличных от 0 и 100, зависят от конкретной реализации SQL в том или ином продукте.

**SQLSTATE:** SQLSTATE дает прикладным программам возможность одинаковым образом обрабатывать ошибки разных систем управления базами данных IBM. Раздел Приложение С книги *DB2 Сообщения и коды* содержит полный список возможных значений SQLSTATE.

Преимущество использования поля SQLCODE заключается в том, что с его помощью можно получить более подробную информацию, чем с помощью SQLSTATE. Со многими кодами SQLCODE связаны элементы SQLCA, указывающие, например, на объект, вызвавший ошибку SQL.

В соответствии со стандартом SQL можно объявлять SQLCODE и SQLSTATE (SQLCOD и SQLSTA на языке FORTRAN) как отдельные переменные хоста. Если вы зададите опцию прокомпиляции STDSQL(YES), коды возврата будут помещаться в эти переменные хоста; в таком случае включать SQLCA в прикладную программу не нужно.

## **Оператор WHENEVER**

Оператор WHENEVER вызывает проверку системой DB2 области SQLCA, после чего либо исполнение программы продолжается, либо, в случае обнаружения ошибки, исключительной ситуации или предупреждения, вызванных исполнением какого-либо оператора SQL, выполняется переход в другую область программы. После этого ваша программа может обработать SQLCODE или SQLSTATE и выполнить необходимые действия в зависимости от типа ошибки или исключительной ситуации.

Оператор WHENEVER позволяет указать действия, которые необходимо выполнить при истинности общего условия. В программе можно использовать несколько операторов WHENEVER. В этом случае первый оператор WHENEVER распространяется на все последующие операторы SQL в исходном тексте программы до очередного оператора WHENEVER.

Оператор WHENEVER имеет следующий вид:

```
EXEC SQL  
  WHENEVER условие действие  
END-EXEC
```

*Условие* — это одно из следующих трех значений:

**SQLWARNING**

Указывает действие, которое должно быть выполнено, если SQLWARN0 = W или SQLCODE содержит положительное значение, отличное от 100. SQLWARN0 может устанавливаться по различным причинам — например, в случае округления значения столбца при записи в переменную хоста. Ваша программа может не рассматривать это как ошибку.

**SQLERROR** Указывает действие, которое должно быть выполнено, если DB2 в результате выполнения оператора SQL вернет ошибку (SQLCODE < 0).

**NOT FOUND** Указывает действие, которое должно быть выполнено, если DB2 не удастся найти строку, удовлетворяющую оператору SQL, либо в случае отсутствия строк, которые могут быть выбраны (SQLCODE = 100).

*Действие* — это одно из следующих двух значений:

**CONTINUE**

Указывает следующий последовательный оператор исходного текста программы.

**GOTO или GO TO метка хоста**

Указывает оператор, обозначаемый *меткой хоста*. Вместо *метки хоста* надо поставить единственный элемент с двоеточием перед ним. Вид этого элемента зависит от языка хоста. Например, для языка COBOL можно использовать *имя раздела* или *имя параграфа* без спецификатора.

Оператор WHENEVER должен стоять перед первым оператором SQL, на который должно распространяться его действие. Однако если ваша программа проверяет SQLCODE непосредственно, она должна проверять SQLCODE после выполнения оператора SQL.

## Обработка арифметических ошибок и ошибок преобразования

При ошибках числовых или знаковых преобразований и при арифметических ошибках возможно присвоение индикаторной переменной значения -2.

Например, при делении на ноль или арифметическом переполнении прекращать выполнение оператора SELECT не обязательно. Если используются индикаторные переменные, при возникновении ошибки в списке SELECT можно продолжить выполнение и возвратить корректные данные для тех строк, где ошибка не произошла.

В тех строках, где ошибка произошла, одному или нескольким выбранным элементам не присваивается никакого осмысленного значения. При этом соответствующей индикаторной переменной присваивается значение -2, а в поле SQLCODE в области SQLCA заносится значение +802 (SQLSTATE '01519').

## Обработка кодов возврата ошибок SQL

Следует проверять ошибки перед принятием данных и обрабатывать ошибки, обнаруженные в данных. Ассемблерная подпрограмма DSNTIAR позволяет получить форматированное представление SQLCA, а также текстовое сообщение на основании поля SQLCODE области SQLCA.

Синтаксис вызова DSNTIAR для конкретного языка программирования и подробности вызова приводятся на следующих страницах:

- Для ассемблера – страница 159
- Для языка C – страница 179
- Для языка COBOL – страница 202
- Для языка FORTRAN – страница 215
- Для языка PL/I – страница 230

DSNTIAR считывает данные из SQLCA, преобразует их в сообщение и помещает результат в область вывода сообщений, предоставленную прикладной программой. Каждый раз при вызове этой подпрограммы любое предшествующее сообщение в области вывода сообщений переписывается. Чтобы получить точное представление SQLCA, сообщения следует переместить или вывести на печать до повторного вызова DSNTIAR и до изменения содержимого SQLCA.

DSNTIAR предполагает, что SQLCA имеет определенный формат. Если ваша прикладная программа изменит формат SQLCA перед вызовом DSNTIAR, результат будет непредсказуем.

### Задание области вывода сообщений

Вызывающая программа должна выделить достаточно памяти в области вывода сообщений для текста сообщений. Скорее всего, для области вывода сообщений не потребуется более 10 строк по 80 байт. В прикладной программе допускается наличие только одной области вывода сообщений.

Область вывода сообщений должна быть задана в формате VARCHAR. Это символьный формат переменной длины; перед данными идет поле длины размером в два байта. Поле длины указывает DSNTIAR, сколько всего байт занимает область вывода сообщений; минимальное значение – 240.

На рис. 8 на стр. 120 показан формат области вывода сообщений; здесь *length* — двухбайтное поле полной длины, а длина каждой строки равна длине логической записи (*lrec*), которую вы указали для DSNTIAR.

Строка:

1	<input type="text"/>
2	<input type="text"/>

.

.

n-1

n

<input type="text"/>
<input type="text"/>

Размеры полей (в байтах):

← 2 → Длина логической записи →

Рисунок 8. Формат области вывода сообщений

При вызове DSNTIAR в качестве параметров необходимо задать SQLCA и область вывода сообщений. Кроме того, требуется указать длину логической записи в байтах (*lrec*) как значение от 72 до 240. DSNTIAR предполагает, что область вывода сообщений содержит записи постоянной длины, равной *lrec*.

DSNTIAR помещает в область вывода сообщений до 10 строк. Если текст сообщения оказывается больше длины записи, указанной в параметре DSNTIAR, выводимое сообщение разбивается на несколько записей, по возможности по словам. Разбитые записи начинаются с отступа. Все записи начинаются с пустого символа, означающего начало абзаца. Если число строк превышает размер области вывода сообщений, DSNTIAR выдает код возврата 4. Пустая запись означает конец области вывода сообщений.

### Возможные коды возврата DSNTIAR

Код	Значение
0	Успешное выполнение.
4	Данных было больше, чем помещается в отведенную область сообщений.
8	Длина логической записи находилась вне интервала от 72 до 240 включительно.
12	Размер области сообщений оказался недостаточным. Длина сообщения была 240 или больше.
16	Ошибка в подпрограмме сообщений TSO.
20	Не удалось загрузить модуль DSNTIA1.
24	Ошибка данных SQLCA.

### Подготовка к использованию DSNTIAR

DSNTIAR может исполняться либо выше, либо ниже границы 16 Мбайт виртуальной памяти. Модуль объектного кода DSNTIAR, поставляемый с DB2, имеет атрибуты AMODE(31) и RMODE(ANY). Во время установки DSNTIAR компонуется как AMODE(31) и RMODE(ANY). Таким образом, DSNTIAR исполняется в 31-разрядном режиме, если:

- Связывается с другими модулями, имеющими атрибуты AMODE(31) и RMODE(ANY),
- Связывается с прикладной программой, в JCL компоновки которой указаны атрибуты AMODE(31) и RMODE(ANY), или

- Загружается прикладной программой.

При загрузке DSNTIAR из другой программы надо соблюдать определенные правила вызова DSNTIAR. Например, если вызывающая программа исполняется в режиме 24-разрядной адресации, а DSNTIAR загружен выше 16-мегабайтной границы, нельзя использовать для вызова DSNTIAR инструкцию ассемблера BALR или макрос CALL, так как они предполагают, что DSNTIAR исполняется в 24-разрядном режиме. Вместо них следует использовать инструкцию, способную переключаться в 31-разрядный режим, например, BASSM.

Вы можете динамически компоновать (загружать) и вызывать DSNTIAR непосредственно из языка, не поддерживающего 31-разрядную адресацию (например, OS/VS COBOL). Для этого можно скомпоновать вторую версию DSNTIAR с атрибутами AMODE(24) и RMODE(24) в еще одну библиотеку модуля загрузки. Другая возможность – написать на ассемблере промежуточную программу вызова DSNTIAR в 31-разрядном режиме, и затем вызвать эту промежуточную программу из прикладной программы в 24-разрядном режиме.

Более подробная информация о допустимых значениях и значениях по умолчанию параметров AMODE и RMODE в конкретном языке приводится в руководстве по написанию прикладных программ на этом языке. Подробности задания атрибутов AMODE и RMODE прикладной программы приводятся в руководстве пользователя компоновщика и загрузчика для того языка, на котором написана программа.

## **Сценарий использования DSNTIAR**

Предположим, что вы хотите проверить прикладную программу для DB2 на языке COBOL на тупиковые ситуации и истечения срока, а также перед продолжением исполнения убедиться, что все указатели закрыты. Чтобы при получении программой отрицательного SQLCODE передать управление подпрограмме обработки ошибок, вы используете оператор WHENEVER SQLERROR.

В подпрограмме обработки ошибок вы пишете раздел, отслеживающий значения SQLCODE –911 и –913. При тупиковой ситуации или истечении срока вы можете получить либо тот, либо другой код SQLCODE. При возникновении одной из этих ошибок подпрограмма обработки ошибок закрывает указатели следующим оператором:

`EXEC SQL CLOSE имя указателя`

В случае успешного закрытия этот оператор возвращает SQLCODE, равный 0 или –501.

Чтобы сформировать полный текст сообщения, соответствующего отрицательным значениям SQLCODE, вы можете вызвать DSNTIAR в подпрограмме обработки ошибок.

1. Выберите длину логической записи, равную (*/rec*) строкам вывода.

Предположим, что в нашем примере */rec* равна 72 (чтобы сообщение умещалось на экране терминала) и хранится в переменной под названием ERROR-TEXT-LEN.

2. Задайте область сообщений в своей программе на языке COBOL.

Поскольку требуется получить область размером до 10 строк длиной по 72 знака, следует задать область размером 720 байт плюс 2-байтная область для длины области вывода сообщений.

```
01  ERROR-MESSAGE.  
    02  ERROR-LEN    PIC S9(4)  COMP VALUE +720.  
    02  ERROR-TEXT   PIC X(72)   OCCURS 10 TIMES  
                                INDEXED BY ERROR-INDEX.  
    77  ERROR-TEXT-LEN  PIC S9(9)  COMP VALUE +72.
```

В данном примере область сообщений называется ERROR-MESSAGE.

3. Не забудьте задать область SQLCA. В данном примере предполагается, что область SQLCA названа SQLCA.

Чтобы вывести на экран содержимое SQLCA, если код SQLCODE равен 0 или –501, следует сначала сформатировать сообщение при помощи вызова DSNTIAR после оператора SQL, устанавливающего SQLCODE 0 или –501:

```
CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
```

После этого можно напечатать область вывода сообщений так же, как любую другую переменную. Полученное сообщение может иметь следующий вид:

```
DSNT408I SQLCODE = -501, ERROR: THE CURSOR IDENTIFIED IN A FETCH OR  
CLOSE STATEMENT IS NOT OPEN  
DSNT418I SQLSTATE = 24501 SQLSTATE RETURN CODE  
DSNT415I SQLERRP = DSNXERT SQL PROCEDURE DETECTING ERROR  
DSNT416I SQLERRD = -315 0 0 -1 0 0 SQL DIAGNOSTIC INFORMATION  
DSNT416I SQLERRD = X'FFFFFEC5' X'00000000' X'00000000'  
                  X'FFFFFFF' X'00000000' X'00000000' SQL DIAGNOSTIC  
INFORMATION
```

---

## Глава 3–2. Использование указателей для получения наборов строк

Для получения прикладной программой набора строк в DB2 используется механизм, называемый **указатель**. При помощи указателя можно получать строки из таблицы или из набора результатов, возвращенного хранимой процедурой. В этой главе объясняется, как прикладная программа при помощи указателя может получать строки из таблицы. Информацию о том, как при помощи указателя получать строки из набора результатов, смотрите в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

---

### Функции указателя

Вы можете получить и обработать набор строк, удовлетворяющих критерию поиска в операторе SQL. Однако программа, выбирающая строки, не может обрабатывать их все одновременно. Программа должна обрабатывать по одной строке за раз.

Чтобы проиллюстрировать понятие указателя, предположим, что DB2 построила *таблицу результатов*<sup>2</sup>, содержащую все заданные оператором SELECT строки. DB2 использует указатель, чтобы сделать строки из таблицы результатов доступными вашей программе. Указатель определяет *текущую строку* таблицы результатов, заданной оператором SELECT. Когда вы используете указатель, ваша программа может получать последовательно по одной строке из таблицы результатов, пока не дойдет до конца данных (то есть не получит условие *не найдена*, SQLCODE=100 и SQLSTATE = '02000'). Набор строк, полученный в результате выполнения оператора SELECT, может состоять из нуля, одной или нескольких строк в зависимости от числа строк, удовлетворяющих критериям поиска в операторе SELECT.

Оператор SELECT, который упоминается в этом разделе, должен находиться внутри оператора DECLARE CURSOR и не должен включать условия INTO. Оператор DECLARE CURSOR определяет указатель, идентифицирующий набор строк, который возвращает оператор SELECT, и дает ему имя.

Таблица результатов с указателем обрабатывается подобно последовательному набору данных. Прежде чем вы начнете получать строки, указатель надо открыть (оператором OPEN). Для получения очередной строки указателя используется оператор FETCH. FETCH можно использовать последовательно, пока вы не получите все строки. Когда возникнет условие “конец данных”, указатель надо закрыть оператором CLOSE (подобно тому, как это делается при обработке условия “конец файла”).

В программе может использоваться несколько указателей. У каждого указателя есть свои:

- оператор DECLARE CURSOR, определяющий указатель

---

<sup>2</sup> DB2 строит таблицы результатов различными способами в зависимости от сложности оператора SELECT. Однако полученные результаты не зависят от способа, которым DB2 их получает.

- операторы OPEN и CLOSE, открывающие и закрывающие указатель
- операторы FETCH для получения строки из таблицы результатов этого указателя.

Переменные хоста, прежде чем ссылаться на них в операторе DECLARE CURSOR, надо объявить. Дополнительную информацию смотрите в разделе Глава 6 книги *DB2 SQL Reference*.

Указатели можно использовать для выборки, изменения или удаления строки из таблицы, но не для вставки строки в таблицу.

## Пример использования указателя

Предположим, ваша программа проверяет данные о сотрудниках отдела D11 и сохраняет данные в разделе DSN8610.EMP. Ниже показаны операторы SQL, которые следует вставить в программу на языке COBOL, чтобы определить и использовать указатель. В данном примере программа использует указатель для обработки набора строк из таблицы DSN8610.EMP.

*Таблица 6 (Стр. 1 из 2). Операторы SQL, определяющие и использующие указатель в программе на языке COBOL*

Оператор SQL	Описан в разделе
<pre>EXEC SQL DECLARE THISEMP CURSOR FOR   SELECT EMPNO, LASTNAME,          WORKDEPT, JOB     FROM DSN8610.EMP    WHERE WORKDEPT = 'D11' FOR UPDATE OF JOB END-EXEC.</pre>	"Шаг 1: Определение указателя" на стр. 125
<pre>EXEC SQL OPEN THISEMP END-EXEC.</pre>	"Шаг 2: Открытие указателя" на стр. 126
<pre>EXEC SQL WHENEVER NOT FOUND   GO TO CLOSE-THISEMP END-EXEC.</pre>	"Шаг 3: Задание реакции на конец данных" на стр. 127
<pre>EXEC SQL FETCH THISEMP   INTO :EMP-NUM, :NAME2,        :DEPT, :JOB-NAME END-EXEC.</pre>	"Шаг 4: Получение строки при помощи указателя" на стр. 127
... для определенных сотрудников в отделе D11 изменяя значение JOB:	"Шаг 5а: Изменение текущей строки" на стр. 128
<pre>EXEC SQL UPDATE DSN8610.EMP   SET JOB = :NEW-JOB  WHERE CURRENT OF THISEMP END-EXEC.</pre>	
... затем печатаем строку.	

Таблица 6 (Стр. 2 из 2). Операторы SQL, определяющие и использующие указатель в программе на языке COBOL

Оператор SQL	Описан в разделе
... для прочих сотрудников удаляем строку:	"Шаг 5b: Удаление текущей строки" на стр. 129
EXEC SQL DELETE FROM DSN8610.EMP WHERE CURRENT OF THISEMP END-EXEC.	
Возвращаемся к выборке и обрабатываем следующую строку.	
CLOSE-THISEMP. EXEC SQL CLOSE THISEMP END-EXEC.	"Шаг 6: Закрытие указателя" на стр. 129

## Шаг 1: Определение указателя

Чтобы определить и идентифицировать набор строк, к которому мы будем обращаться через указатель, используйте оператор DECLARE CURSOR. Оператор DECLARE CURSOR дает указателю имя и задает оператор SELECT. Оператор SELECT определяет критерий отбора строк, которые составят таблицу результатов. Оператор DECLARE CURSOR выглядит так:

```
EXEC SQL  
DECLARE имя-указателя CURSOR FOR  
  SELECT список-имен-столбцов  
    FROM имя-таблицы  
    WHERE критерий-поиска  
  FOR UPDATE OF имя-столбца  
  END-EXEC.
```

Здесь показан простой оператор SELECT. В операторе SELECT внутри оператора DECLARE CURSOR можно использовать и другие условия. В разделе Глава 5 книги *DB2 SQL Reference* показаны некоторые другие условия, которые можно использовать в операторе SELECT.

**Изменение столбца:** Если вы собираетесь изменять содержимое столбца в некоторых (или во всех) строках указанной таблицы, включите в оператор условие FOR UPDATE OF с именами всех столбцов, которые вы будете изменять. На использование условия FOR UPDATE OF действуют опции препроцессора NOFOR и STDSQL. Сведения об этих правилахсмотрите в Табл. 44 на стр. 441. Если вы не указали имена столбцов, которые хотите изменить, и при этом не задали опции препроцессора STDSQL(YES) или NOFOR, вы получите ошибку.

Столбец в указанной таблице можно изменить, даже если он не входит в таблицу результатов. В таком случае имя этого столбца можно не указывать в операторе SELECT (однако не забудьте указать его в условии FOR UPDATE OF). Когда указатель возвратит (при помощи оператора FETCH) строку со значением, которое вы хотите изменить, можно использовать оператор UPDATE ... WHERE CURRENT OF для изменения строки.

Предположим, например, что каждая строки таблицы результатов включает столбцы EMPNO, LASTNAME и WORKDEPT из таблицы DSN8610.EMP. Если вы хотите изменить значение в столбце JOB (один из столбцов таблицы DSN8610.EMP), оператор DECLARE CURSOR должен содержать FOR UPDATE OF JOB, несмотря на то, что в условии SELECT столбец JOB опущен.

FOR UPDATE OF можно использовать также для изменения значений в столбце одной таблицы с использованием информации из другой таблицы. Например, вы хотите поощрить сотрудников, отвечающих за определенные проекты. Для этого определим указатель так:

```
EXEC SQL
DECLARE C1 CURSOR FOR
    SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME, SALARY
    FROM DSN8610.EMP X
    WHERE EXISTS
        (SELECT *
         FROM DSN8610.PROJ Y
         WHERE X.EMPNO=Y.RESPEMP
         AND Y.PROJNO=:GOODPROJ)
    FOR UPDATE OF SALARY;
```

Пользователи вводят номера проектов, для которых вы хотите поощрить сотрудников, и вы записываете эти номера в переменную хоста GOODPROJ. Затем вы используете указатель и оператор UPDATE ... WHERE CURRENT OF, чтобы:

- Найти номера проектов и ответственных за эти проекты в таблице DSN8610.PROJ.
- Найти оклады соответствующих сотрудников в таблице DSN8610.EMP.
- Изменить оклады для этих сотрудников.

**Таблицы результатов только для чтения:** Некоторые таблицы результатов – например, результат объединения нескольких таблиц – нельзя изменять. Подробные спецификации таблиц результатов только для чтения описаны в разделе Глава 6 книги *DB2 SQL Reference*.

## Шаг 2: Открытие указателя

Чтобы сообщить DB2, что вы готовы к обработке первой строки из таблицы результатов, ваша программа должна выполнить оператор OPEN. После этого DB2 использует оператор SELECT внутри DECLARE CURSOR, чтобы задать набор строк. Если в операторе SELECT есть переменные хоста, DB2 будет использовать при выборе строк *текущие значения* этих переменных. Таблица результатов, удовлетворяющих критериям поиска, может содержать ноль, одну или несколько строк. Оператор OPEN выглядит так:

```
EXEC SQL
OPEN имя_указателя
END-EXEC.
```

При использовании указателей DB2 определяет значения специальных регистров CURRENT DATE, CURRENT TIME и CURRENT TIMESTAMP один раз, когда выполняется оператор OPEN. Полученные значения регистров DB2 используют во всех последующих операторах FETCH.

На время, требуемое для обработки DB2 оператора OPEN, влияют два фактора:

- Должна ли DB2 выполнять какую-либо сортировку перед получением строк из таблицы результатов
- Использует ли DB2 параллелизм для обработки оператора SELECT, связанного с указателем

Более подробную информацию смотрите в разделе "Влияние сортировки на оператор OPEN CURSOR" на стр. 744.

### Шаг 3: Задание реакции на конец данных

Чтобы определить, получила ли программа последнюю строку данных, надо проверить поле SQLCODE на значение 100 или поле SQLSTATE на значение '02000'. Эти коды будут получены, если по оператору FETCH была выбрана последняя строка таблицы результатов, а ваша программа выполняет последующий оператор FETCH. Например:

```
IF SQLCODE = 100 GO TO DATA-NOT-FOUND.
```

Другой вариант – использовать оператор WHENEVER NOT FOUND. Оператор WHENEVER NOT FOUND может передать управление другой части вашей программы, где будет выполнен оператор CLOSE. Оператор WHENEVER NOT FOUND выглядит так:

```
EXEC SQL  
WHENEVER NOT FOUND GO TO символический-адрес  
END-EXEC.
```

При любой операции выборки строки при помощи указателя в вашей программе должна быть предусмотрена возможность ситуации "конец данных" и ее обработка. Дополнительную информацию об операторе WHENEVER NOT FOUND смотрите в разделе "Проверка выполнения операторов SQL" на стр. 116.

### Шаг 4: Получение строки при помощи указателя

Оператор FETCH используется для занесения содержимого выбранной строки в переменные хоста вашей программы. Оператор SELECT в операторе DECLARE CURSOR задает строки с данными, которые требуются вашей программе, однако DB2 не передаст вам никаких данных, пока прикладная программа не выполнит оператор FETCH.

Когда ваша программа выполняет оператор FETCH, DB2 перемещает указатель на следующую строку в таблице результатов, делая ее *текущей строкой*. Затем DB2 заносит содержимое текущей строки в переменные хоста программы, заданные в условии INTO оператора FETCH. Эта последовательность повторяется при каждом выполнении оператора FETCH, пока вы не обработаете все строки в таблице результатов.

Оператор FETCH выглядит так:

```
EXEC SQL  
FETCH имя-указателя  
INTO :переменная-хоста1, :переменная-хоста2  
END-EXEC.
```

Если вы используете оператор FETCH при работе с удаленной подсистемой, производительность может снизиться. Чтобы преодолеть этот недостаток, можно использовать выборку блоками. Дополнительную информацию смотрите в разделе “Использование блочной выборки” на стр. 424. При выборке блоками обрабатываются строки за текущей строкой прикладной программы. Выборку блоками нельзя использовать, если указатель применяется для изменения или удаления строк.

## Шаг 5а: Изменение текущей строки

Когда ваша программа получает текущую строку, вы можете изменить данные в ней при помощи оператора UPDATE. Для этого надо выполнить оператор UPDATE...WHERE CURRENT OF, специально предназначенный для использования с указателем. Оператор UPDATE ... WHERE CURRENT OF выглядит так:

```
EXEC SQL
UPDATE имя-таблицы
  SET столбец1 = значение1, столбец2 = значение2
  WHERE CURRENT OF имя-указателя
    END-EXEC.
```

Оператор UPDATE с использованием указателя отличается от оператора, описанного в разделе “Глава 2–2. Работа с таблицами и изменение данных” на стр. 57.

- Изменяется только одна строка – текущая.
- Условие WHERE задает указатель на изменяемую строку.
- Прежде, чем использовать этот оператор UPDATE, вы должны назвать каждый изменяемый столбец в условии FOR UPDATE OF оператора SELECT в операторе DECLARE CURSOR.<sup>3</sup>

Оператор UPDATE нельзя использовать для изменения строк во временных таблицах.

После внесения изменений в строку указатель остается на текущей строке, пока не будет выполнен следующий оператор FETCH.

Нельзя изменять строку, если при этом будет нарушено ограничение уникальности, проверочное или реляционное ограничение. Дополнительную информацию смотрите в разделе “Изменение таблиц с реляционными связями” на стр. 72.

В разделе “Изменение текущих значений: UPDATE” на стр. 71 показано, как последовательно использовать оператор UPDATE для изменения всех столбцов, удовлетворяющих критерию поиска. Другой вариант – использовать последовательно оператор UPDATE...WHERE CURRENT OF, если вы хотите получить копию строки, изучить ее, а затем изменить.

---

<sup>3</sup> Если вы не указали имен столбцов, которые хотите изменить, и при этом не задали опции прекомпилятора STDSQL(YES) или NOFOR, при попытке изменения этих столбцов вы получите код ошибки в переменных хоста SQLCODE и SQLSTATE или в соответствующих полях SQLCA. Это относится только к случаю, когда вы не задали опции прекомпиляции STDSQL(YES) или NOFOR.

## Шаг 5b: Удаление текущей строки

Когда ваша программа получает текущую строку, вы можете удалить ее при помощи оператора DELETE. Для этого надо выполнить оператор DELETE...WHERE CURRENT OF, специально предназначенный для использования с указателем. Оператор DELETE ... WHERE CURRENT OF выглядит так:

```
EXEC SQL  
DELETE FROM имя-таблицы  
WHERE CURRENT OF имя-указателя  
END-EXEC.
```

Оператор DELETE с использованием указателя отличается от оператора, описанного в разделе “Глава 2–2. Работа с таблицами и изменение данных” на стр. 57.

- Можно удалить только одну строку – текущую.
- Условие WHERE задает указатель на удаляемую строку.

Оператор DELETE с указателем нельзя использовать для удаления строк из временных таблиц.

Удалив строку, вы не можете использовать указатель для изменения или удаления других строк, пока не будет выполнен оператор FETCH, который устанавливает указатель на следующую строку.

В разделе “Удаление строк: DELETE” на стр. 73 показано, как последовательно использовать оператор DELETE для удаления всех столбцов, удовлетворяющих критерию поиска. Другой вариант – использовать последовательно оператор DELETE...WHERE CURRENT OF, если вы хотите получить копию строки, изучить ее, а затем удалить.

Нельзя удалить строку, если при этом будут нарушены реляционные ограничения.

## Шаг 6: Закрытие указателя

Закончив обработку строк таблицы результатов, надо выполнить оператор CLOSE, чтобы закрыть указатель, перед тем как использовать его снова.

```
EXEC SQL  
CLOSE имя-указателя  
END-EXEC.
```

Если вы закончили обработку строк “таблицы результатов” и не хотите больше использовать указатель, можно не закрывать его явно – DB2 автоматически закроет указатель при завершении работы программы.

---

## Объявление указателя с опцией WITH HOLD

Если ваша программа завершает единицу работы (то есть выполняет принятие всех произведенных изменений), и вы не хотите, чтобы DB2 закрыла *все открытые указатели*, объягите указатель с опцией WITH HOLD.

Открытый указатель, определенный с опцией WITH HOLD, остается открытым после операции принятия. Этот указатель позиционируется после последней

выбранной строки и перед логически следующей строкой в полученном наборе результатов.

Следующее объявление указателя приводит к тому, что он сохранит свое положение в таблице DSN8610.EMP после точки принятия:

```
EXEC SQL
DECLARE EMPLUPDT CURSOR WITH HOLD FOR
    SELECT EMPNO, LASTNAME, PHONENO, JOB, SALARY, WORKDEPT
    FROM DSN8610.EMP
        WHERE WORKDEPT < 'D11'
        ORDER BY EMPNO
END-EXEC.
```

Указатель, объявленный таким образом, будет закрыт, когда:

- Будет выполнен оператор CLOSE указатель, ROLLBACK или CONNECT
- Будет выполнена функция CAF CLOSE
- Завершится выполнение прикладной программы.

При аварийном завершении программы положение указателя теряется; для подготовки к перезапуску программа переустанавливает указатель.

При объявлении указателей с опцией WITH HOLD действуют следующие ограничения:

Не используйте оператор DECLARE CURSOR WITH HOLD с подключением новых пользователей при помощи возможности подключения DB2, поскольку все открытые указатели при этом закрываются.  
Не объявляйте указатель с опцией WITH HOLD в потоке, который может стать неактивным. В этом случае ваши блокировки могут остаться неснятыми на неопределенное время.

#### Система IMS

*Нельзя использовать оператор DECLARE CURSOR...WITH HOLD в программах обработки сообщений (MPP) и в пакетной обработке сообщений, управляемой сообщениями (BMP). В этом случае каждое сообщение рассматривается как новый пользователь DB2; независимо от того, была ли при объявлении указана опция WITH HOLD, для нового пользователя все открытые указатели закрываются. WITH HOLD можно использовать в программах BMP, не управляемых сообщениями, и в пакетных программах DL/I.*

## CICS

В прикладных программах CICS можно использовать оператор DECLARE CURSOR...WITH HOLD, чтобы указать, что указатель не должен закрываться в точках принятия и синхронизации. Однако SYNCPOINT ROLLBACK закрывает все указатели, как и окончание задания (EOT), прежде чем DB2 использует повторно или завершит поток. Поскольку в псевододиалоговых транзакциях обычно используются несколько операторов EXEC CICS RETURN и тем самым происходит несколько EOT, область действия сохраняемого указателя будет ограничена. После EOT вам надо снова открыть и переустановить указатель WITH HOLD, как и объявленный без опции WITH HOLD.

Всегда следует закрывать ненужные вам указатели. Если вы оставите указатель подключения CICS незакрытым, DB2 не сможет закрыть его, пока утилита подключения CICS не будет вызвана снова или пока не будет завершен поток.



---

## Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN

DCLGEN, генератор объявлений, поставляемый с DB2, генерирует операторы DECLARE, которые можно использовать в программах на языках C, COBOL или PL/I, вместо того, чтобы писать этот оператор самому. Подробное описание синтаксиса DCLGEN смотрите в разделе Глава 2 справочника *DB2 Command Reference*.

DCLGEN генерирует объявления таблиц и записывает их как члены секционированного набора данных, который вы можете включать в свою программу. Когда вы используете DCLGEN для генерации объявления таблицы, DB2 получает соответствующую информацию из каталога DB2, где содержится информация об определении таблицы и определении каждого столбца в этой таблице. DCLGEN использует эту информацию для генерации полного оператора SQL DECLARE для таблицы или производной таблицы и соответствующего описания структуры PL/I или C или описания записи COBOL. DCLGEN можно использовать для создания описания таблицы только тогда, когда сама таблица уже существует.

DCLGEN надо использовать до прекомпиляции вашей программы. Вызовите DCLGEN с именем таблицы или производной таблицы перед тем как прекомпилировать вашу программу. Чтобы использовать объявления, сгенерированные DCLGEN, в вашей программе, используйте оператор SQL INCLUDE.

Чтобы можно было использовать DCLGEN, DB2 должна быть активна. DCLGEN можно запустить несколькими способами:

- Из ISPF через DB2I. Выберите опцию DCLGEN на панели меню основных опций DB2I . Затем заполните поля панели DCLGEN, введя информацию для построения объявлений. Нажмите клавишу ENTER.
- Прямо из TSO. Зарегистрируйтесь в TSO, введите команду TSO DSN, а затем – подкоманду DCLGEN.
- Из CLIST, запущенной в приоритетном или в фоновом режиме TSO, введите команду DSN, а затем – подкоманду DCLGEN.
- В JCL. Задайте требуемую информацию в JCL и запустите DCLGEN в пакетном задании.

Если вы хотите запустить DCLGEN на переднем плане и имена ваших таблиц содержат символы DBCS, вам потребуется вводить и выводить двухбайтные символы. Если у вас нет терминала, который позволяет выводить символы DBCS, можно вводить символы DBCS в шестнадцатеричном режиме редактора ISPF.

## Вызов DCLGEN через DB2I

Простейший способ запуска DCLGEN – через DB2I. На рис. 9 показана панель DCLGEN, которая выводится при выборе опции 2, DCLGEN, в меню основных опций DB2I. Дополнительные инструкции по использованию DB2I смотрите в разделе “Использование ISPF и DB2 Interactive (DB2I)” на стр. 475.

```
DSNEDP01          DCLGEN          SSID: DSN
====>

Enter table name for which declarations are required:
 1 SOURCE TABLE NAME ===>          (Unqualified table name)
 2 TABLE OWNER      ===>          (Optional)
 3 AT LOCATION ..... ===>          (Optional)

Enter destination data set:           (Can be sequential or partitioned)
 4 DATA SET NAME ... ===>
 5 DATA SET PASSWORD ===>          (If password protected)

Enter options as desired:
 6 ACTION ..... ===>          (ADD new or REPLACE old declaration)
 7 COLUMN LABEL .... ===>          (Enter YES for column label)
 8 STRUCTURE NAME .. ===>          (Optional)
 9 FIELD NAME PREFIX ===>          (Optional)
10 DELIMIT DBCS .... ===>          (Enter YES to delimit DBCS identifiers)
11 COLUMN SUFFIX ... ===>          (Enter YES to append column name)
12 INDICATOR VARS .. ===>          (Enter YES for indicator variables)

PRESS: ENTER to process    END to exit    HELP for more information
```

Рисунок 9. Панель DCLGEN

Заполните поля панели DCLGEN:

1 SOURCE TABLE NAME

Неспецифицированное имя таблицы, производной таблицы или временной таблицы, для которой DCLGEN должна сгенерировать объявления данных SQL. Эта таблица может быть расположена на вашей подсистеме DB2 или на другой подсистеме DB2. Чтобы задать имя таблицы из другой подсистемы DB2, введите спецификатор таблицы в поле TABLE OWNER и имя положения в поле AT LOCATION. DCLGEN генерирует трехчастное имя таблицы из полей SOURCE TABLE NAME, TABLE OWNER и AT LOCATION. Вместо имени таблицы можно использовать алиас.

Чтобы задать имя таблицы, содержащее специальные символы или пробелы, заключите его в апострофы. Если само имя содержит апострофы, каждый из них надо удвоить (''). Например, чтобы задать имя таблицы DON'S TABLE, введите:

'DON''S TABLE'

Имена таблиц из символов DBCS не надо заключать в апострофы. Если вы не заключили имя таблицы в апострофы, DB2 переводит символы нижнего регистра в верхний.

DCLGEN не рассматривает подчеркивание как специальный символ. Например, имя таблицы JUNE\_PROFITS не требуется заключать в

апострофы. Поскольку в языке COBOL имена полей не могут содержать символов подчеркивания, DCLGEN заменяет однобайтный символ подчеркивания на дефис (–) в именах полей COBOL, которые строятся по именам таблиц.

## 2 TABLE OWNER

Владелец исходной таблицы. Если вы не задали это значение и таблица – локальная, DB2 предполагает, что спецификатор – ваш ID регистрации TSO. Если таблица находится на удаленной подсистеме, это значение надо указать обязательно.

## 3 AT LOCATION

Положение таблицы или производной таблицы на другой подсистеме DB2. Если вы указываете этот параметр, надо также указать специфицированное имя в поле SOURCE TABLE NAME. Значение поля AT LOCATION добавляется спереди к оператору SQL DECLARE так:

*имя\_положения.id\_владельца.имя\_таблицы*

Например, для положения PLAINS\_GA:

PLAINS\_GA.CARTER.CROP\_YIELD\_89

Если положение не задано, по умолчанию предполагается имя локального положения. Это поле применяется только для распределенных единиц работы (то есть когда названное вами положение должно быть другой DB2 for OS/390).

## 4 DATA SET NAME

Имя размещенного вами набора данных с объявлениями, которые порождает DCLGEN. Это имя надо задать обязательно, умолчания не применяются.

Набор данных должен уже существовать и быть доступным DCLGEN; он может быть последовательным или секционированным. Если не заключать имя набора данных в апострофы, DCLGEN добавит к нему стандартные префикс (идентификатор пользователя) и суффикс (язык) TSO. DCLGEN определяет язык хоста по значению на панели значений DB2I по умолчанию.

Например, для библиотечного имени LIBNAME(MEMBNNAME) будет построено такое имя:

*userid.libname.language(membname)*

а для библиотечного имени LIBNAME –

*userid.libname.language*

Если набор данных защищен паролем, этот пароль надо задать в поле DATA SET PASSWORD.

## 5 DATA SET PASSWORD

Пароль для набора данных, заданного в поле DATA SET NAME, если он защищен паролем. Пароль не выводится на вашем терминале и не распознается, если вы ввели его в предыдущем сеансе.

## 6 ACTION

Сообщает DCLGEN, что надо делать с выводом, если он направляется в секционированный набор данных. (Эта опция игнорируется, если в поле DATA SET NAME задан последовательный набор данных.)

ADD указывает, что старой версии вывода нет, и надо создать новый член с заданным именем набора данных. Эта опция принимается по умолчанию.

REPLACE заменяет старую версию, если она есть. Если старой версии нет, создается новый член.

#### 7 COLUMN LABEL

Сообщает DCLGEN, надо ли включать в объявления данных в качестве комментариев надписи, заданные для столбцов таблицы или производной таблицы. (Оператор SQL LABEL ON создает надписи для столбцов, используемые как дополнения к именам столбцов.) Возможные варианты:

YES – включать надписи для столбцов.

NO – игнорировать надписи для столбцов. Это значение используется по умолчанию.

#### 8 STRUCTURE NAME

Имя генерируемой структуры. Это имя может содержать до 31 символа. Если имя не является строкой DBCS и первый символ не является латинской буквой, заключите имя в апострофы. Если вы используете специальные символы, помните про возможные конфликты имен.

Если оставить это поле пустым, DCLGEN генерирует имя, добавляя к имени таблицы или производной таблицы префикс *DCL*. Для языков COBOL и PL/I, если имя таблицы или производной таблицы – строка DBCS, префикс также будет состоять из символов DBCS.

Для языка С символы, которые вы вводите в это поле, не преобразуются в верхний регистр.

#### 9 FIELD NAME PREFIX

Задает префикс, который DCLGEN использует при формировании имен полей в выводе. Например, если задать ABCDE, будут сгенерированы имена полей ABCDE1, ABCDE2 и так далее.

DCLGEN принимает префиксы длиной до 28 байт; они могут содержать специальные символы и двухбайтные символы. Если вы задаете префикс из однобайтных или смешанных символов и первый символ не является латинской буквой, префикс надо заключить в апострофы. Если вы используете специальные символы, помните про возможные конфликты имен.

Для языков COBOL и PL/I, если имя – строка DBCS, DCLGEN генерирует двухбайтные эквиваленты для добавляемых к префиксу цифр. Для языка С символы, которые вы вводите в это поле, не преобразуются в верхний регистр.

Если оставить это поле пустым, имена полей будут совпадать с именами столбцов в таблице или производной таблице.

#### 10 DELIMIT DBCS

Сообщает DCLGEN, надо ли использовать ограничители для имен таблиц и имен столбцов из символов DBCS в объявлении таблиц. Возможные варианты:

YES – заключать имена таблиц и имен столбцов из символов DBCS в ограничители SQL.

NO – не использовать ограничители SQL для имен таблиц и имен столбцов из символов DBCS.

## 11 COLUMN SUFFIX

Сообщает DCLGEN, надо ли при формировании имен полей добавлять имя столбца к значению, заданному в поле FIELD NAME PREFIX.

Например, если вы зададите YES, префикс имени поля – NEW, а имя столбца – EMPNO, имя поля будет NEWEMPNO.

Если вы задаете YES, надо ввести значение в поле FIELD NAME PREFIX. Если этого не сделать, DCLGEN выдает предупреждение и использует имена столбцов как имена полей.

Значение по умолчанию – NO, и имя поля не используется как суффикс, то есть именами полей управляет значение FIELD NAME PREFIX, если оно задано.

## 12 INDICATOR VARS

Сообщает DCLGEN, надо ли генерировать массив переменных–индикаторов для структуры переменных хоста.

Если указать YES, имя массива будет образовано из имени таблицы добавлением префикса “I” (или символа DBCS “<I>,” если имя таблицы содержит только двухбайтные символы). Форма объявления данных зависит от языка:

Для программы на языке C: short int Iимя-таблицы[n];

Для программы на языке COBOL: 01 Имя-таблицы PIC S9(4) USAGE COMP OCCURS n TIMES.

Для программы на языке PL/I: DCL Имя-таблицы(n) BIN FIXED(15);

где *n* – число столбцов в таблице. Например, если вы определяете таблицу:

```
CREATE TABLE HASNULLS (CHARCOL1 CHAR(1), CHARCOL2 CHAR(1));
```

и хотите объявить массив переменных–индикаторов в программе на языке COBOL, DCLGEN может сгенерировать следующее объявление переменных хоста:

```
01 DCLHASNULLS.  
 10 CHARCOL1          PIC X(1).  
 10 CHARCOL2          PIC X(1).  
01 IHASNULLS PIC S9(4) USAGE COMP OCCURS 2 TIMES.
```

По умолчанию используется NO, и массив переменных–индикаторов не генерируется.

DCLGEN генерирует имя таблицы или имя столбца в операторе DECLARE как идентификатор без ограничителя, за исключением следующих случаев:

- Имя содержит специальные символы, но не является строкой DBCS.
- Имя является строкой DBCS, и вы задали использование ограничителей для имен DBCS.

Если вы используете в качестве идентификатора зарезервированное слово SQL, необходимо будет отредактировать вывод DCLGEN, чтобы добавить соответствующие ограничители SQL.

---

## Включение объявлений данных в вашу программу

Чтобы поместить сгенерированное объявление таблицы и описание записи языка COBOL в вашу исходную программу, используйте следующий оператор SQL INCLUDE:

```
EXEC SQL
INCLUDE имя-члена
END-EXEC.
```

Например, чтобы включить описание для таблицы DSN8610.EMP, используйте такой код:

```
EXEC SQL
INCLUDE DECEMP
END-EXEC.
```

В этом примере DECEMP – имя члена секционированного набора данных, который содержит объявление таблицы и соответствующее описание записи языка COBOL для таблицы DSN8610.EMP. (Описание записи языка COBOL – это двухуровневая структура хоста, которая соответствует столбцам строки таблицы. Сведения о структурах хоста смотрите в разделе “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145.) Чтобы получить текущее описание таблицы, при помощи DCLGEN получите объявление и сохраните его как член DECEMP в библиотеке (обычно это секционированный набор данных) непосредственно перед прекомпиляцией программы.

Вывод DCLGEN должен удовлетворять нуждам большинства пользователей, но в некоторых случаях требуется редактировать его, чтобы он работал в ваших особых условиях. Например, DCLGEN не может определить, определен ли столбец с условием NOT NULL дополнительно с условием DEFAULT, и в этом случае в вывод DCLGEN в соответствующие определения столбцов надо добавить условие DEFAULT.

---

## Поддержка DCLGEN для языков C, COBOL и PL/I

DCLGEN выводит имена переменных из исходных имен в базе данных. В Табл. 7 var означает имена переменных, которые DCLGEN дает при необходимости для уточнения объявления на языке хоста.

Таблица 7 (Стр. 1 из 2). Объявления, генерируемые DCLGEN

Тип данных SQL <sup>6</sup>	C	COBOL	PL/I
SMALLINT	short int	PIC S9(4) USAGE COMP	BIN FIXED(15)
INTEGER	long int	PIC S9(9) USAGE COMP	BIN FIXED(31)
DECIMAL(p,s) или NUMERIC(p,s)	decimal(p,s) <sup>4</sup>	PIC S9(p-s)V9(s) USAGE COMP-3 Если p>18, выводится предупреждение.	DEC FIXED(p,s) Если p>15, выводится предупреждение.
REAL или FLOAT(n) 1 <= n <= 21	float	USAGE COMP-1	BIN FLOAT(n)

Таблица 7 (Стр. 2 из 2). Объявления, генерируемые DCLGEN

Тип данных SQL <sup>6</sup>	C	COBOL	PL/I
DOUBLE PRECISION, DOUBLE или FLOAT(n)	double	USAGE COMP-2	BIN FLOAT(n)
CHAR(1)	char	PIC X(1)	CHAR(1)
CHAR(n)	char var [n+1]	PIC X(n)	CHAR(n)
VARCHAR(n)	struct {short int var_len; char var_data[n]; } var;	10 var. 49 var_LEN PIC 9(4) USAGE COMP. 49 var_TEXT PIC X(n).	CHAR(n) VAR
GRAPHIC(1)	wchar_t	PIC G(1)	GRAPHIC(1)
GRAPHIC(n) n > 1	wchar_t var[n+1];	PIC G(n) USAGE DISPLAY-1. <sup>1</sup> или PIC N(n). <sup>1</sup>	GRAPHIC(n)
VARGRAPHIC(n)	struct VARGRAPH {short len; wchar_t data[n]; } var;	10 var. 49 var_LEN PIC 9(4) USAGE COMP. 49 var_TEXT PIC G(n) USAGE DISPLAY-1. <sup>1</sup> или 10 var. 49 var_LEN PIC 9(4) USAGE COMP. 49 var_TEXT PIC N(n). <sup>1</sup>	GRAPHIC(n) VAR
BLOB(n) <sup>5</sup>	SQL TYPE IS BLOB_LOCATOR	USAGE SQL TYPE IS BLOB-LOCATOR	SQL TYPE IS BLOB_LOCATOR
CLOB(n) <sup>5</sup>	SQL TYPE IS CLOB_LOCATOR	USAGE SQL TYPE IS CLOB-LOCATOR	SQL TYPE IS CLOB_LOCATOR
DBCLOB(n) <sup>5</sup>	SQL TYPE IS DBCLOB_LOCATOR	USAGE SQL TYPE IS DBCLOB-LOCATOR	SQL TYPE IS DBCLOB_LOCATOR
ROWID	SQL TYPE IS ROWID	USAGE SQL TYPE IS ROWID	SQL TYPE IS ROWID
DATE	char var[11] <sup>2</sup>	PIC X(10) <sup>2</sup>	CHAR(10) <sup>2</sup>
TIME	char var[9] <sup>3</sup>	PIC X(8) <sup>3</sup>	CHAR(8) <sup>3</sup>
TIMESTAMP	char var[27]	PIC X(26)	CHAR(26)

#### Примечания к Табл. 7 на стр. 138:

1. DCLGEN выбирает формат в зависимости от символа, заданного в поле DBCS панели умолчаний COBOL.
2. Это объявление используется, если нет особого обработчика установки даты для форматирования дат; иначе берется длина, заданная в опции установки LOCAL DATE LENGTH.
3. Это объявление используется, если нет особого обработчика установки времени для форматирования времени; иначе берется длина, заданная в опции установки LOCAL TIME LENGTH.
4. Если ваш компилятор языка С не поддерживает десятичный тип данных, отредактируйте вывод DCLGEN, заменив объявления десятичных данных на объявления данных типа double.
5. Для типов данных BLOB, CLOB или DBCLOB DCLGEN генерирует локатор большого объекта.

6. Для пользовательского типа данных DCLGEN генерирует объявление на языке хоста для эквивалентного исходного типа данных.

Глава 2 книги *DB2 Command Reference* содержит дополнительные подробности о подкоманде DCLGEN.

## Пример: Добавление объявления таблицы и структуры переменных хоста к библиотеке

В этом примере объявление таблицы SQL и соответствующая структура переменных хоста добавляются к библиотеке. В примере предполагается, что:

- Имя библиотеки – **префикс.TEMP.COBOL**.
- Член библиотеки – новый член под именем VPHONE.
- Таблица – локальная таблица под именем DSN8610.VPHONE.
- Структура переменных хоста генерируется для языка COBOL.
- Эта структура получит имя по умолчанию DCLVPHONE.

Информация, которую вводите вы, выделена жирным шрифтом.

### Шаг 1. Задайте COBOL как язык хоста

Выберите опцию **D** в меню ISPF/PDF, чтобы вывести панель умолчаний DB2I.

Задайте **COBOL** в качестве языка хоста, как показано на рис. 10, и нажмите Enter. Будет выведена панель умолчаний COBOL, как показано на рис. 11 на стр. 141.

Заполните панель умолчаний COBOL, как требуется. Нажмите клавишу Enter, чтобы сохранить новые умолчания, если вы их задали, и вернуться в меню основных опций DB2I.

```
DSNEOP01                               DB2I DEFAULTS
COMMAND ==> _

Change defaults as desired:

1 DB2 NAME ..... ==> DSN          (Subsystem identifier)
2 DB2 CONNECTION RETRIES ==> 0        (How many retries for DB2 connection)
3 APPLICATION LANGUAGE ==> COBOL    (ASM, C, CPP, COBOL, COB2, IBMCOB,
                                         FORTRAN, PLI)
4 LINES/PAGE OF LISTING ==> 80       (A number from 5 to 999)
5 MESSAGE LEVEL ..... ==> I          (Information, Warning, Error, Severe)
6 SQL STRING DELIMITER ==> DEFAULT   (DEFAULT, ' or ")
7 DECIMAL POINT ..... ==> .          (., or ,)
8 STOP IF RETURN CODE >= ==> 8        (Lowest terminating return code)
9 NUMBER OF ROWS ..... ==> 20        (For ISPF Tables)
10 CHANGE HELP BOOK NAMES?==> NO      (YES to change HELP data set names)
11 DB2I JOB STATEMENT: (Optional if your site has a SUBMIT exit)
    ==> //USRTO01A JOB ('ACCOUNT'), 'NAME'
    ==> /**
    ==> /**
    ==> /**

PRESS: ENTER to process      END to cancel      HELP for more information
```

Рисунок 10. Панель умолчаний DB2I – изменение языка программирования

```

DSNEOP02          COBOL DEFAULTS
COMMAND ==> _

Change defaults as desired:

1 COBOL STRING DELIMITER ==>      (DEFAULT, ' or ")
2 DBCS SYMBOL FOR DCLGEN ==>      (G/N - Character in PIC clause)

```

*Рисунок 11. Панель умолчаний COBOL. Выводится только если в поле APPLICATION LANGUAGE панели умолчаний DB2I задано COBOL, COB2 или IBMCOB.*

## Шаг 2. Создание объявления таблицы и структуры хоста

Выберите опцию **2** в меню основных опций DB2I и нажмите клавишу Enter, чтобы вывести панель DCLGEN.

Заполните поля, как показано на рис. 12, и нажмите клавишу Enter.

```

DSNEDP01          DCLGEN          SSID: DSN
==>
Enter table name for which declarations are required:

1 SOURCE TABLE NAME ==> DSN8610.VPHONE
2 TABLE OWNER      ==>
3 AT LOCATION ..... ==>           (Location of table, optional)

Enter destination data set:           (Can be sequential or partitioned)
4 DATA SET NAME ... ==> TEMP(VPHONEC)
5 DATA SET PASSWORD ==>             (If password protected)

Enter options as desired:
6 ACTION ..... ==> ADD            (ADD new or REPLACE old declaration)
7 COLUMN LABEL .... ==> NO           (Enter YES for column label)
8 STRUCTURE NAME .. ==>             (Optional)
9 FIELD NAME PREFIX ==>             (Optional)
10 DELIMIT DBCS    ==> YES          (Enter YES to delimit DBCS identifiers)
11 COLUMN SUFFIX ... ==> NO          (Enter YES to append column name)
12 INDICATOR VARS .. ==> NO          (Enter YES for indicator variables)

PRESS: ENTER to process    END to exit    HELP for more information

```

*Рисунок 12. Панель DCLGEN – выбор исходной таблицы и набора данных назначения*

Если операция будет выполнена успешно, верхняя часть экрана будет выглядеть, как показано на рис. 13.

```

DSNE905I EXECUTION COMPLETE, MEMBER VPHONEC ADDED
***
```

*Рисунок 13. Сообщение об удачном завершении*

Затем DB2 выводит экран, показанный на рис. 14 на стр. 142. Нажмите клавишу Enter, чтобы вернуться в меню основных опций DB2I.

```

DSNEDP01          DCLGEN          SSID: DSN
====>
DSNE294I SYSTEM RETCODE=000      USER OR DSN RETCODE=0
Enter table name for which declarations are required:
 1 SOURCE TABLE NAME ===> DSN8610.VPHONE
 2 TABLE OWNER      ===>
 3 AT LOCATION ..... ===>           (Location of table, optional)

Enter destination data set:      (Can be sequential or partitioned)
 4 DATA SET NAME ... ===> TEMP(VPHONEC)
 5 DATA SET PASSWORD ===>           (If password protected)

Enter options as desired:
 6 ACTION ..... ===> ADD          (ADD new or REPLACE old declaration)
 7 COLUMN LABEL .... ===> NO         (Enter YES for column label)
 8 STRUCTURE NAME .. ===>           (Optional)
 9 FIELD NAME PREFIX ===>           (Optional)
10 DELIMIT DBCS     ===>           (Enter YES to delimit DBCS identifiers)
11 COLUMN SUFFIX ... ===>           (Enter YES to append column name)
12 INDICATOR VARS .. ===>           (Enter YES for indicator variables)

PRESS: ENTER to process    END to exit    HELP for more information

```

*Рисунок 14. Панель DCLGEN – вывод системного и пользовательского кодов возврата*

### Шаг 3. Проверка результатов

Чтобы посмотреть или исправить результаты, надо сначала выйти из DB2I, введя **X** в командной строке меню основных опций DB2I. Появится меню ISPF/PDF, и вы можете выбрать опции просмотра или редактирования результатов.

Пусть, например, надо редактировать набор данных префикс.TEMP.COBOL(VPHONEC), показанный на рис. 15 на стр. 143.

```

***** DCLGEN TABLE(DSN8610.VPHONE)                                ***
*****      LIBRARY(SYSADM TEMP COBOL(VPHONEC))                      ***
*****      QUOTE                                         ***
***** ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS ***
      EXEC SQL DECLARE DSN8610.VPHONE TABLE
      ( LASTNAME          VARCHAR(15) NOT NULL,
        FIRSTNAME         VARCHAR(12) NOT NULL,
        MIDDLEINITIAL    CHAR(1) NOT NULL,
        PHONENUMBER      VARCHAR(4) NOT NULL,
        EMPLOYEENUMBER   CHAR(6) NOT NULL,
        DEPTNUMBER       CHAR(3) NOT NULL,
        DEPTNAME         VARCHAR(36) NOT NULL
      ) END-EXEC.
***** COBOL DECLARATION FOR TABLE DSN8610.VPHONE                 *****
01 DCLVPHONE.
 10 LASTNAME.
    49 LASTNAME-LEN      PIC S9(4) USAGE COMP.
    49 LASTNAME-TEXT    PIC X(15).
 10 FIRSTNAME.
    49 FIRSTNAME-LEN    PIC S9(4) USAGE COMP.
    49 FIRSTNAME-TEXT   PIC X(12).
 10 MIDDLEINITIAL     PIC X(1).
 10 PHONENUMBER.
    49 PHONENUMBER-LEN  PIC S9(4) USAGE COMP.
    49 PHONENUMBER-TEXT PIC X(4).
 10 EMPLOYEENUMBER    PIC X(6).
 10 DEPTNUMBER        PIC X(3).
 10 DEPTNAME.
    49 DEPTNAME-LEN     PIC S9(4) USAGE COMP.
    49 DEPTNAME-TEXT    PIC X(36).
***** THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 7 *****

```

*Рисунок 15. Результаты DCLGEN в режиме редактирования*



---

## Глава 3–4. Встроенные операторы SQL в языках хоста

В этой главе приводится подробная информация о кодировании операторов SQL в следующих языках хоста:

- “Кодирование операторов SQL в программе на языке ассемблер”
- “Кодирование операторов SQL в программах на языках С или С++” на стр. 160
- “Кодирование операторов SQL в программах на языке COBOL” на стр. 181
- “Кодирование операторов SQL в программах на языке FORTRAN” на стр. 204
- “Кодирование операторов SQL в программах на языке PL/I” на стр. 215.

Для каждого из этих языков приводятся собственные инструкции и описания:

- Определения области связи SQL
- Определения областей дескрипторов SQL
- Встроенных операторов SQL
- Использования переменных хоста
- Объявления переменных хоста
- Определений эквивалентных типов данных SQL
- Определения совместимости типов данных SQL и языка хоста
- Использования переменных–индикаторов или структур хоста, в зависимости от языка
- Обработки кодов возврата SQL

Правила чтения синтаксических диаграмм в этой главе смотрите в разделе “Как читать синтаксические диаграммы” на стр. 6.

В этой главе не приводится информация о межъязыковых вызовах и вызовах хранимых процедур. Передача параметров хранимым процедурам, в том числе совместимость типов данных языка и типов данных SQL, обсуждаются в разделе “Написание и подготовка прикладной программы, использующей хранимые процедуры” на стр. 607.

---

### Кодирование операторов SQL в программе на языке ассемблер

В этом разделе рассказывается о способах программирования для кодирования операторов SQL в программе на ассемблере.

#### Задание области связи SQL

В программу на ассемблере, которая содержит операторы SQL, должны быть включены одна или обе следующих переменных хоста:

- Переменная SQLCODE, объявляется как полное целое (длиной в слово)
- Переменная SQLSTATE, объявляется как символьная строка длиной, равной 5 (CL5)

Либо

- SQLCA, которая содержит переменные SQLCODE и SQLSTATE.

После выполнения каждого оператора SQL DB2 устанавливает значения для SQLCODE и SQLSTATE. По значениям этих переменных программа может определить, успешно ли выполнен последний оператор SQL. Все операторы SQL в программе должны находиться в пределах области действия объявления переменных SQLCODE и SQLSTATE.

Переменные SQLCODE или SQLSTATE задаются, если задана опция препомпилиятора STDSQL(YES) (в соответствии со стандартом SQL); SQLCA задается, если задана опция препомпилиятора STDSQL(NO) (в соответствии с правилами DB2).

### Если задано STDSQL(YES)

При использовании параметра препроцессора STDSQL(YES) не следует определять SQLCA. В противном случае DB2 проигнорирует SQLCA, а установки для SQLCA приведут к ошибкам времени компиляции.

Объявляемая переменная SQLSTATE не должна являться элементом структуры. Переменные хоста SQLCODE и SQLSTATE нужно объявить между операторами BEGIN DECLARE SECTION и END DECLARE SECTION в разделе объявлений программы.

### Если задано STDSQL(NO)

При использовании опции препомпилиятора STDSQL(NO) задайте SQLCA явно. В программе на ассемблере SQLCA можно задать либо напрямую, либо с помощью оператора SQL INCLUDE. Оператор SQL INCLUDE запрашивает стандартное объявление SQLCA:

```
EXEC SQL INCLUDE SQLCA
```

Для повторной—входной программы SQLCA должна быть включена в уникальную область данных (DSECT), созданную для вашей задачи.

Например, в начале программы задайте:

```
PROGAREA DSECT  
      EXEC SQL INCLUDE SQLCA
```

Альтернатива – создать для SQLCA отдельную адресуемую область памяти.

Глава 6 книги *DB2 SQL Reference* содержит дополнительную информацию об операторе INCLUDE. Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLCA.

## Определение областей дескрипторов SQL

SQLDA требуется для следующих операторов:

- CALL...USING DESCRIPTOR имя–дескриптора
- DESCRIBE имя–оператора INTO имя–дескриптора
- DESCRIBE CURSOR переменная–хоста INTO имя–дескриптора
- DESCRIBE INPUT имя–оператора INTO имя–дескриптора
- DESCRIBE PROCEDURE переменная–хоста INTO имя–дескриптора
- DESCRIBE TABLE переменная–хоста INTO имя–дескриптора
- EXECUTE...USING DESCRIPTOR имя–дескриптора
- FETCH...USING DESCRIPTOR имя–дескриптора
- OPEN...USING DESCRIPTOR имя–дескриптора
- PREPARE...INTO имя–дескриптора

В отличие от SQLCA, в программе может быть несколько SQLDA, и у SQLDA может быть любое допустимое имя. В программе на ассемблере SQLDA можно задать либо напрямую, либо при помощи оператора SQL INCLUDE. Оператор SQL INCLUDE запрашивает стандартное объявление SQLDA:

```
EXEC SQL INCLUDE SQLDA
```

Объявления SQLDA нужно поместить до первого оператора SQL, в котором есть ссылки на дескриптор данных, если только не используется опция прекомпилятора TWOPASS. Глава 6 книги *DB2 SQL Reference*, содержит дополнительную информацию об операторе INCLUDE, Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLDA.

## Вставка операторов SQL

Операторы SQL в программе на ассемблере можно использовать везде, где можно использовать исполняемые операторы.

Каждый оператор SQL в программе на ассемблере должен начинаться с ключевых слов EXEC SQL. Ключевые слова EXEC и SQL нужно записать в одной строке, но остальная часть оператора может быть написана в последующих строках.

Оператор UPDATE в программе на ассемблере может быть записан так:

EXEC SQL UPDATE DSN8610.DEPT	X
SET MGRNO = :MGRNUM	X
WHERE DEPTNO = :INTDEPT	

**Комментарии:** В операторы SQL комментарии ассемблера включать нельзя. Однако в любой оператор встроенного SQL можно включить комментарий SQL, если задать опцию прекомпилятора STDSQL(YES).

**Продолжения операторов SQL:** Для операторов SQL правила продолжения строк такие же, как и для операторов ассемблера, за исключением того, что EXEC SQL нужно задавать в одной строке. Любую часть оператора, не уместившуюся в одной строке, можно записать в последующих строках, начинающихся с поля продолжения (по умолчанию позиция 16). В каждой строке оператора, кроме последней, сразу же за правым полем в позиции 72 должен стоять символ продолжения (то есть непустой символ).

**Объявление таблиц и производных таблиц:** Программа на ассемблере должна включать операторы DECLARE с описанием каждой таблицы или производной таблицы, к которым она обращается.

**Включение кода:** Чтобы включить операторы SQL или операторы объявления переменных хоста на ассемблере из компонента многораздельного набора данных, вставьте в исходный текст, куда вы хотите включить эти операторы, оператор SQL:

```
EXEC SQL INCLUDE имя-элемента
```

Нельзя использовать вложенные операторы SQL INCLUDE.

**Поля:** Опция прекомпилятора MARGINS позволяет установить левое поле, правое поле и позицию продолжения. По умолчанию для этих полей принимаются, соответственно, значения 1, 71 и 16. Если код EXEC SQL

начинается до заданного для левого поля значения, препомпилиятор DB2 не опознает оператор SQL. При использовании полей по умолчанию оператор SQL можно поместить где угодно между позициями 2 и 71.

**Имена:** Для переменной хоста можно использовать любое допустимое в ассемблере имя. Однако не следует использовать имена внешних записей или имена планов доступа, начинающиеся с 'DSN', и имена переменных хоста, начинающиеся с 'SQL'. Эти имена зарезервированы для DB2.

Имя переменной хоста, используемой во встроенным SQL, не может начинаться с символа подчеркивания. Однако подчеркивание в качестве первого символа можно использовать в именах, которые *не* используются во встроенном SQL.

**Метки операторов:** Оператору SQL может предшествовать метка. Первую строку оператора SQL можно начать с метки в левом поле (позиция 1). Если метка не используется, оставьте позицию 1 пустой.

**Оператор WHENEVER:** Чтобы оператор WHENEVER работал правильно, условие GOTO оператора WHENEVER должно указывать на метку в исходном тексте на ассемблере, которая должна находиться в пределах области действия операторов SQL.

**Особенности программ на ассемблере:** К программам на ассемблере предъявляются следующие требования:

- Для повторно-входимых программ препомпилиятор помещает все генерируемые им переменные и структуры в область DSECT под названием SQLSECT и генерирует ассемблерную переменную SQLDLEN. SQLDLEN содержит длину DSECT.

Программа должна выделить область с размером, который задан в SQLDLEN, инициализировать ее и обеспечить адресуемость к этой области как к DSECT SQLDSECT.

**CICS**

Пример кода для поддержки повторно-входимых программ, запускаемых под CICS:

```

DFHEISTG DSECT
    DFHEISTG
    EXEC SQL INCLUDE SQLCA
*
    DS      0F
SQDWSREG EQU   R7
SQDWSTOR DS     (SQLDLEN)C РЕЗЕРВИРУЕМ ПАМЯТЬ ДЛЯ SQLDSECT
:
&XPROGRM DFHEIENT CODEREG=R12,EIBREG=R11,DATAREG=R13
*
*
* РАБОЧАЯ ПАМЯТЬ SQL
    LA      SQDWSREG,SQDWSTOR      ПОЛУЧАЕМ АДРЕС SQLDSECT
    USING SQLDSECT,SQDWSREG      И СООБЩАЕМ ЕГО АССЕМБЛЕРУ
*
```

**TSO**

В примере программы в префикс.*SDSNSAMP(DSNTIAD)* показано, как выделить память для SQLDSECT в программе, запускаемой в среде TSO.

- DB2 не обрабатывает обозначения set в операторах SQL.
- Комментарий в генерируемом коде может содержать более двух строк продолжения.
- В генерируемом коде используются литеральные константы (например, =F'–84'), поэтому может потребоваться оператор LTORG.
- В генерируемом коде используются регистры 0, 1, 14 и 15. Регистр 13 указывает на область хранения, которую использует вызываемая программа. Регистр 15 не содержит кода возврата после вызова, генерируемого оператором SQL.

## CICS

Прикладная программа CICS при генерации кода точки входа использует макрокоманду DFHEIENT. При использовании этой макрокоманды утите следующее:

- Если в макрокоманде DFHEIENT используется значение по умолчанию DATAREG, регистр 13 указывает именно на эту область хранения.
- Если в DFHEIENT используется значение, отличное от DATAREG, надо обеспечить адресуемость области хранения.

Например, для использования SAVED можно поместить вокруг каждого оператора SQL команды для сохранения, загрузки и восстановления регистра 13, как в следующем примере:

```
ST    13,SAVER13      СОХРАНЯЕМ РЕГИСТР 13
LA    13,SAVED        УКАЗАТЕЛЬ НА ОБЛАСТЬ СОХРАНЕНИЯ
EXEC  SQL . . .
L     13,SAVER13      ВОССТАНАВЛИВАЕМ РЕГИСТР 13
```

- Если в сгенерированном прекомпилятором коде содержится ошибка адресуемости, связанная с переменными хоста ввода или вывода в операторе SQL, убедитесь, что у вас достаточно базовых регистров.
- Не помещайте опции транслятора CICS в исходный текст на ассемблере. Вместо этого передайте эти опции транслятору через поле PARM.

## Использование переменных хоста

Каждую переменную хоста нужно явно объявить перед ее первым использованием в операторе SQL. Если задается опция прекомпилятора TWOPASS, каждую переменную хоста нужно явно объявить перед ее первым использованием в операторе DECLARE CURSOR.

Перед операторами ассемблера, задающими переменные хоста, можно поместить оператор BEGIN DECLARE SECTION, а после них – оператор END DECLARE SECTION. Если используется опция прекомпилятора STDSQL(YES), надо использовать операторы BEGIN DECLARE SECTION и END DECLARE SECTION.

Переменные хоста можно объявить обычными средствами ассемблера (DC или DS) в зависимости от типа данных и ограничений на этот тип данных. Для DC или DS можно задать значение (например, DC H'5'). Препроцессор DB2 проверяет только объявления упакованных десятичных.

Всем переменным хоста в операторе SQL должно предшествовать двоеточие (:).

Оператор SQL, в котором используется переменная хоста, должен находиться в пределах области действия оператора, который объявляет данную переменную.

## Объявление переменных хоста

Для объявления переменных хоста допустимы лишь некоторые из допустимых объявлений ассемблера. Если объявление для переменной хоста недопустимо, любой оператор SQL, содержащий ссылку на эту переменную, может стать причиной сообщения "UNDECLARED HOST VARIABLE".

**Числовые переменные хоста:** На следующем рисунке показан синтаксис допустимых объявлений числовых переменных хоста. Числовое значение задает масштаб упакованной десятичной переменной. Если значение не содержит десятичную точку, масштаб равен 0.

Для типов данных с плавающей точкой (E, EH, EB, D, DH и DB) DB2 использует опцию прекомпилятора FLOAT для определения, используется ли для переменной формат плавающей точки IEEE или же формат плавающей точки System/390®. Если используется опция прекомпилятора FLOAT(S390), переменные хоста с плавающей точкой надо определить как E, EH, D или DH. Если используется опция прекомпилятора FLOAT(IEEE), переменные хоста с плавающей точкой надо определить как EB или DB. DB2 преобразует все входные данные с плавающей точкой в формат System/390 перед тем как сохранить их.

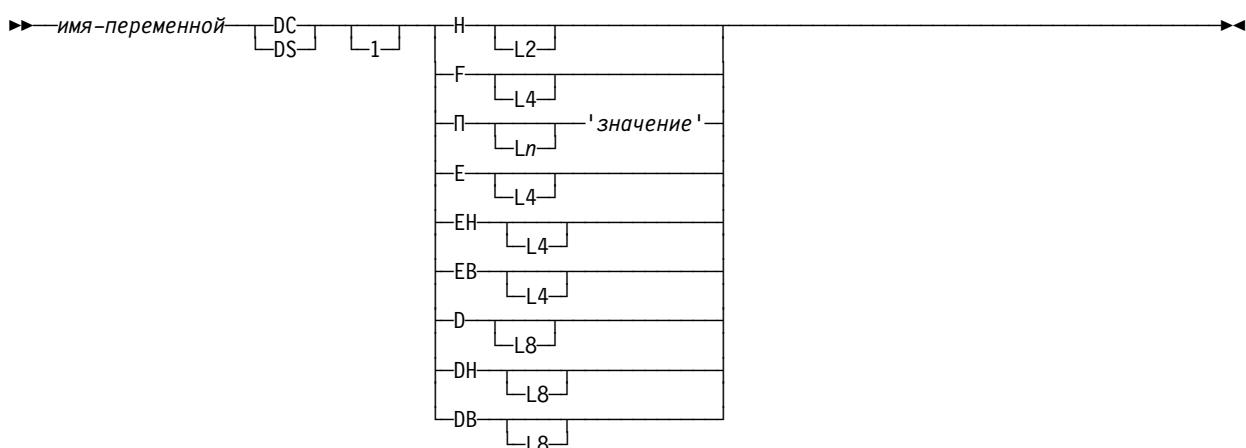


Рисунок 16. Числовые переменные хоста

**Символьные переменные хоста:** Для символьных переменных хоста существует три допустимые формы:

- Строки фиксированной длины
- Строки переменной длины
- CLOB (символьные большие объекты)

На следующих рисунках показан синтаксис для всех этих форм, кроме CLOB. Синтаксис для CLOB смотрите на рис. 23 на стр. 153.



Рисунок 17. Символьные строки фиксированной длины

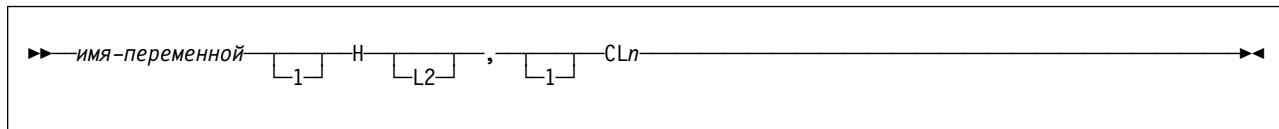


Рисунок 18. Символьные строки переменной длины

**Графические переменные хоста:** Для графических переменных хоста существует три допустимые формы:

- Строки фиксированной длины
- Строки переменной длины
- DBCLOB (двуихбайтные символьные большие объекты)

На следующих рисунках показан синтаксис для всех этих форм, кроме DBCLOB. Синтаксис для DBCLOB смотрите на рис. 23 на стр. 153. На синтаксических диаграммах параметр значение представляет один или несколько символов DBCS, а символы < и > означают символы переключения shift-out и shift-in.

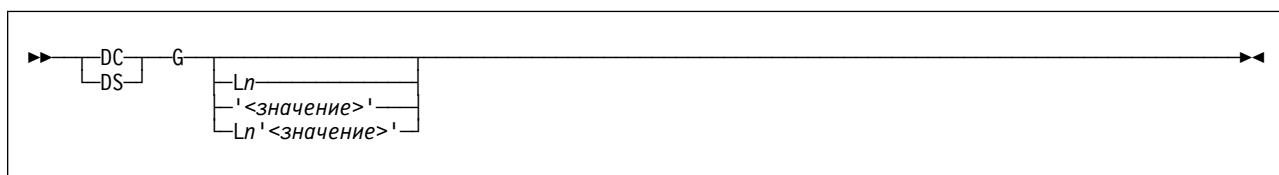


Рисунок 19. Графические строки фиксированной длины



Рисунок 20. Графические строки переменной длины

**Локаторы набора результатов:** На следующем рисунке показан синтаксис объявлений локаторов набора результатов. Использование этих переменных хоста описано в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

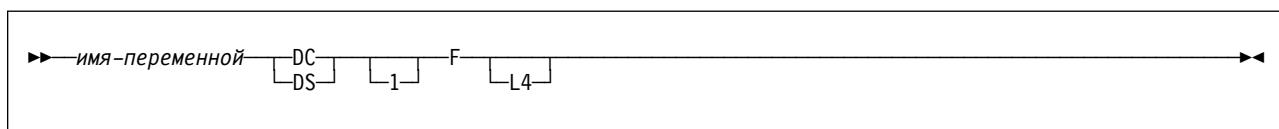


Рисунок 21. Локаторы набора результатов

**Локаторы таблиц:** Ниже показан синтаксис объявлений локаторов таблиц. Использование этих переменных хоста описано в разделе “Доступ к таблицам переходов в пользовательских функциях” на стр. 307.

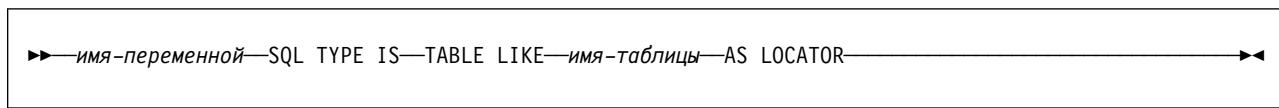


Рисунок 22. Локаторы таблиц

**Переменные и локаторы больших объектов:** На следующем рисунке показан синтаксис объявлений переменных хоста и локаторов BLOB, CLOB и DBCLOB.

Если длина большого объекта задается в Кбайтах, Мбайтах или Гбайтах, между длиной и символами K, M или G не должно оставаться пробелов.

Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

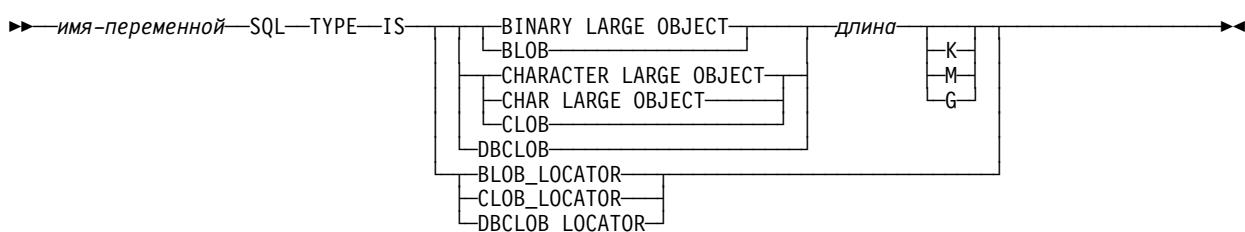


Рисунок 23. Переменные и локаторы LOB

**Переменные ROWID:** На следующем рисунке показан синтаксис объявлений переменных ROWID. Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

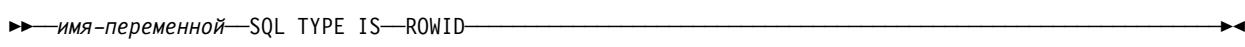


Рисунок 24. Переменные ROWID

## Определение эквивалентных типов данных SQL и ассемблера

В Табл. 8 приводятся типы данных SQL и базовые значения SQLTYPE и SQLLEN, используемые препроцессором для переменных хоста, которые он находит в операторах SQL. Если переменная хоста задается с переменной-индикатором, значению SQLTYPE присваивается базовое значение SQLTYPE плюс 1.

Таблица 8 (Стр. 1 из 2). Типы данных SQL, используемые препроцессором в объявлениях ассемблера

Тип данных ассемблера	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
DS HL2	500	2	SMALLINT
DS FL4	496	4	INTEGER
DS P'значение' DS PLn'значение' или DS PLn 1<=n<=16	484	p в байте 1, s в байте 2	DECIMAL(p,s)  Описание для DECIMAL(p,s) смотрите в Табл. 9 на стр. 155.
DS EL4 DS EHL4 DS EBL4	480	4	REAL или FLOAT(n) 1<=n<=21

## Ассемблер

Таблица 8 (Стр. 2 из 2). Типы данных SQL, используемые прекомпилятором в объявлениях ассемблера

Тип данных ассемблера	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
DS DL8	480	8	DOUBLE PRECISION, или FLOAT ( <i>n</i> ) $22 \leq n \leq 53$
DS DHL8			
DS DBL8			
DS CL <i>n</i> $1 \leq n \leq 255$	452	<i>n</i>	CHAR( <i>n</i> )
DS HL2,CL <i>n</i> $1 \leq n \leq 255$	448	<i>n</i>	VARCHAR( <i>n</i> )
DS HL2,CL <i>n</i> $n > 255$	456	<i>n</i>	VARCHAR( <i>n</i> )
DS GL <i>m</i> $2 \leq m \leq 2541$	468	<i>n</i>	GRAPHIC( <i>n</i> ) <sup>2</sup>
DS HL2,GL <i>m</i> $2 \leq m \leq 2541$	464	<i>n</i>	VARGRAPHIC( <i>n</i> ) <sup>2</sup>
DS HL2,GL <i>m</i> $m > 2541$	472	<i>n</i>	VARGRAPHIC( <i>n</i> ) <sup>2</sup>
DS FL4	972	4	Локатор набора результатов <sup>2</sup>
SQL TYPE IS TABLE LIKE имя-таблицы AS LOCATOR	976	4	Локатор таблиц <sup>2</sup>
SQL TYPE IS BLOB_LOCATOR	960	4	Локатор BLOB <sup>2</sup>
SQL TYPE IS CLOB_LOCATOR	964	4	Локатор CLOB <sup>3</sup>
SQL TYPE IS DBCLOB_LOCATOR	968	4	Локатор DBCLOB <sup>3</sup>
SQL TYPE IS BLOB( <i>n</i> ) $1 \leq n \leq 2147483647$	404	<i>n</i>	BLOB( <i>n</i> )
SQL TYPE IS CLOB( <i>n</i> ) $1 \leq n \leq 2147483647$	408	<i>n</i>	CLOB( <i>n</i> )
SQL TYPE IS DBCLOB( <i>n</i> ) $1 \leq n \leq 10737418232$	412	<i>n</i>	DBCLOB( <i>n</i> ) <sup>2</sup>
SQL TYPE IS ROWID	904	40	ROWID

Примечание к Табл. 8 на стр. 153:

1. *m* – число байтов.
2. *n* – число двухбайтных символов.
3. Этот тип данных нельзя использовать как тип столбца.

С помощью Табл. 9 на стр. 155 можно задать переменные хоста, которые принимают вывод от базы данных. Ее можно использовать для определения типа данных ассемблера, эквивалентного для данного типа данных SQL. Например, если надо обработать данные типа TIMESTAMP, можно по таблице

определить подходящий тип переменной хоста в программе, принимающей значение данных.

Таблица 9 (Стр. 1 из 2). Типы данных SQL, соответствующие объявляемым типам ассемблера

Тип данных SQL	Эквивалент в ассемблере	Примечания
SMALLINT	DS HL2	
INTEGER	DS F	
DECIMAL( <i>p,s</i> ) или NUMERIC( <i>p,s</i> )	DS P'значение' DS PLn'значение' DS PLn	<i>p</i> – точность; <i>s</i> – масштаб. $1 \leq p \leq 31$ и $0 \leq s \leq p$ . $1 \leq n \leq 16$ . значение – литеральное значение, содержащее десятичную точку. Можно использовать <i>Ln</i> , значение либо и то, и другое. Рекомендуется использовать только значение.  Точность: Если используется <i>Ln</i> , точность равна $2n-1$ ; в противном случае она равна числу цифр в параметре значение. Масштаб: Если используется значение, масштаб равен числу цифр справа от десятичной точки; в противном случае масштаб равен 0.
		<b>Чтобы эффективнее использовать индексы:</b> Используйте параметр значение. Если <i>p</i> четное, не используйте <i>Ln</i> и убедитесь, что точность значения равна <i>p</i> , а масштаб значения равен <i>s</i> . Если <i>p</i> нечетное, можно использовать <i>n</i> (хотя это не рекомендуется), но <i>n</i> нужно выбрать так, чтобы $2n-1=p$ , а значение нужно выбрать так, чтобы масштаб был <i>s</i> . Включите в значение десятичную точку, даже если масштаб значения равен 0.
REAL или FLOAT( <i>n</i> )	DS EL4 DS EHL4 DS EBL4 <sup>1</sup>	$1 \leq n \leq 21$
DOUBLE PRECISION, DOUBLE, или FLOAT( <i>n</i> )	DS DL8 DS DHL8 DS DBL8 <sup>1</sup>	$22 \leq n \leq 53$
CHAR( <i>n</i> )	DS CL <i>n</i>	$1 \leq n \leq 255$
VARCHAR( <i>n</i> )	DS HL2,CL <i>n</i>	
GRAPHIC( <i>n</i> )	DS GL <i>m</i>	<i>m</i> выражается в байтах. <i>n</i> – число двухбайтных символов. $1 \leq n \leq 127$
VARGRAPHIC( <i>n</i> )	DS HL2,GL <i>x</i> DS HL2' <i>m</i> ',GL <i>x'</i> <значение>' <sup>1</sup>	<i>x</i> и <i>m</i> выражаются в байтах. <i>n</i> – число двухбайтных символов. Здесь < и > означают символы переключения shift-out и shift-in.
DATE	DS,CL <i>n</i>	Если используется подпрограмма–обработчик даты, <i>n</i> задается этой подпрограммой; в противном случае <i>n</i> должно быть не меньше 10.
TIME	DS,CL <i>n</i>	Если используется подпрограмма–обработчик времени, <i>n</i> задается этой подпрограммой. В противном случае <i>n</i> должно быть не меньше 6. Чтобы включить часть, соответствующую секундам, <i>n</i> должно быть не меньше 8.
TIMESTAMP	DS,CL <i>n</i>	<i>n</i> должно быть не меньше 19. Чтобы подключить часть, соответствующую микросекундам, <i>n</i> должно быть равно 26. Если <i>n</i> меньше 26, микросекунды усекаются.

Таблица 9 (Стр. 2 из 2). Типы данных SQL, соответствующие объявляемым типам ассемблера

Тип данных SQL	Эквивалент в ассемблере	Примечания
Локатор набора результатов	DS F	Этот тип данных используется только для приема наборов результатов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор таблиц	SQL TYPE IS TABLE LIKE имя_таблицы AS LOCATOR	Этот тип данных используется только в пользовательской функции или в хранимой процедуре для приема строк таблицы переходов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор BLOB	SQL TYPE IS BLOB_LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов BLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор CLOB	SQL TYPE IS CLOB_LOCATOR	Этот тип данных используется только для работы с данными в столбцах CLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор DBCLOB	SQL TYPE IS DBCLOB_LOCATOR	Этот тип данных используется только для работы с данными в столбцах DBCLOB. Нельзя использовать этот тип данных в качестве типа столбца.
BLOB( <i>n</i> )	SQL TYPE IS BLOB( <i>n</i> )	1 ≤ <i>n</i> ≤ 2147483647
CLOB( <i>n</i> )	SQL TYPE IS CLOB( <i>n</i> )	1 ≤ <i>n</i> ≤ 2147483647
DBCLOB( <i>n</i> )	SQL TYPE IS DBCLOB( <i>n</i> )	<i>n</i> – число двубайтных символов. 1 ≤ <i>n</i> ≤ 1073741823
ROWID	SQL TYPE IS ROWID	

Примечание к Табл. 9 на стр. 155:

- Переменные хоста с плавающей точкой в формате IEEE не поддерживаются для пользовательских функций и хранимых процедур.

### Примечания по объявлению переменных ассемблера и их использованию

При объявлении переменных ассемблера надо помнить следующее.

**Тип графических данных хоста:** Тип данных ассемблера “графические данные хоста” можно использовать в операторах SQL, когда действует опция препроцессора GRAPHIC. Но даже при действующей опции GRAPHIC нельзя использовать в операторах SQL двухбайтные литералы ассемблера.

**Символьные переменные хоста:** Если переменную хоста объявить как символьную строку, но не задать длину, например, DC C 'ABCD', DB2 интерпретирует ее как строку с длиной 1. Чтобы длина была верной, следует задать атрибут длины (например, DC CL4'ABCD').

**Переменные хоста с плавающей точкой:** Все данные с плавающей точкой в DB2 хранятся в формате System/390. Однако ваши данные переменной хоста могут быть или в формате с плавающей точкой System/390 или в формате с плавающей точкой IEEE. DB2 использует опцию препроцессора FLOAT(S390|IEEE) для формата переменных хоста с плавающей точкой (IEEE

или System/390). DB2 не проверяет, соответствует ли объявление переменной хоста или формат содержания переменной хоста опции препроцессора. Это значит, что вам надо следить за тем, чтобы тип переменных хоста с плавающей точкой соответствовал опции препроцессора.

**Типы данных ассемблера специального назначения:** Типы данных локаторов для ассемблера – такие же типы данных, что и типы данных SQL. Локаторы нельзя использовать в качестве типов столбцов. Информация о том, как использовать эти типы данных, приводится в следующих разделах:

**Локатор таблиц** “Доступ к таблицам переходов в пользовательских функциях” на стр. 307

**Локаторы больших объектов** “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253

**Переполнение:** Следует остерегаться переполнения. Например, при приеме значения столбца INTEGER, которое больше 32767, в переменную хоста DS H вы получите предупреждение о переполнении или ошибку, в зависимости от того, используется ли переменная–индикатор или нет.

**Усечение:** Следует остерегаться усечения. Например, при приеме значения столбца 80–символьного CHAR в переменную хоста, объявленную как DS CL70, в принимаемой строке усекаются десять правых символов. При приеме значения с плавающей точкой или десятичного значения в переменную хоста, объявленную как DS F, будет удалена вся дробная часть этого значения.

## Определение совместимости типов данных SQL и ассемблера

Переменные хоста ассемблера, используемые в операторах SQL, должны быть совместимы по типу со столбцами, с которыми их предполагается использовать.

- Типы числовых данных совместимы друг с другом. Столбцы SMALLINT, INTEGER, DECIMAL или FLOAT совместимы с числовыми переменными хоста ассемблера.
- Типы символьных данных совместимы друг с другом. Столбцы CHAR, VARCHAR или CLOB совместимы с символьными переменными хоста ассемблера фиксированной и переменной длины.
- Типы символьных данных частично совместимы с локаторами CLOB. Столбцу CHAR или VARCHAR можно назначить значение локатора CLOB, но значение из столбца CHAR или VARCHAR нельзя назначать переменной хоста типа локатора CLOB.
- Типы графических данных совместимы друг с другом. Столбцы GRAPHIC, VARGRAPHIC и DBCLOB совместимы с символьными переменными ассемблера графики хоста фиксированной и переменной длины.
- Типы графических данных частично совместимы с локаторами DBCLOB. Значение локатора DBCLOB можно присвоить столбцу GRAPHIC или VARGRAPHIC, но значение столбца GRAPHIC VARGRAPHIC нельзя присвоить переменной хоста локатора DBCLOB.
- Типы данных даты–времени совместимы с символьными переменными хоста. Столбцы DATE, TIME или TIMESTAMP совместимы с символьными переменными хоста ассемблера фиксированной и переменной длины.

- Столбец BLOB совместим только с переменной хоста BLOB.
- Столбцы ROWID совместимы только с переменными хоста ROWID.
- Переменная хоста совместима с особым пользовательским типом, если тип этой переменной хоста совместим с исходным для него типом.  
Информация о назначении и сравнении пользовательских типов приводится в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

При необходимости DB2 автоматически преобразует строку фиксированной длины в строку переменной длины или строку переменной длины в строку фиксированной длины.

### Использование переменных–индикаторов

Переменные–индикатор – это двухбайтное целое (*DS HL2*). Если для переменной X задана переменная–индикатор, то, когда DB2 требуется занести в X пустое значение, в переменную–индикатор заносится отрицательное значение и X не обновляется. Программа перед использованием X должна проверить переменную–индикатор. Если переменная–индикатор отрицательна, это значит, что X имеет пустое значение, а любое реально записанное в нее значение можно игнорировать.

Если программа хочет с помощью X назначить столбцу пустое значение, она должна присвоить переменной–индикатору отрицательное значение. Тогда DB2 присвоит столбцу пустое значение и проигнорирует любое значение, содержащееся в X.

Переменные–индикаторы объявляются так же, как и переменные хоста. Объявления двух типов переменных можно смешивать любым подходящим способом. Дополнительную информацию о переменных–индикаторах смотрите в разделах “Использование индикаторных переменных с переменными хоста” на стр. 113 или в разделе Глава 3 книги *DB2 SQL Reference*.

#### **Пример:**

Задан оператор:

```
EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,          X
                           :DAY :DAYIND,        X
                           :BGN :BGNIND,        X
                           :END :ENDIND
```

Переменные можно объявить так:

CLSCD	DS CL7	
DAY	DS HL2	
BGN	DS CL8	
END	DS CL8	
DAYIND	DS HL2	ПЕРЕМЕННАЯ–ИНДИКАТОР ДЛЯ DAY
BGNIND	DS HL2	ПЕРЕМЕННАЯ–ИНДИКАТОР ДЛЯ BGN
ENDIND	DS HL2	ПЕРЕМЕННАЯ–ИНДИКАТОР ДЛЯ END

На следующем рисунке показан синтаксис допустимой переменной–индикатора.

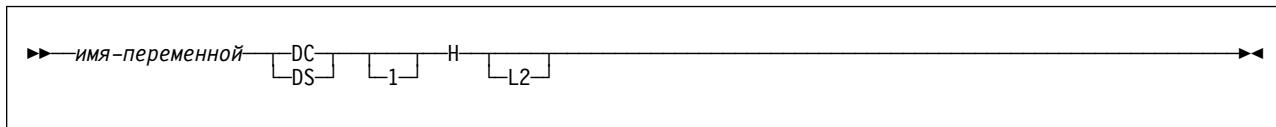


Рисунок 25. Переменная—индикатор

## Обработка кодов возврата ошибок SQL

Для преобразования кода возврата SQL в текстовое сообщение можно пользоваться подпрограммой DSNTIAR. DSNTIAR берет данные, содержащиеся в SQLCA, форматирует из них сообщение и помещает результат в назначеннную прикладной программой область вывода сообщений. Подробная информация о поведении DSNTIAR и связанные с ней понятия приводятся в разделе “Обработка кодов возврата ошибок SQL” на стр. 119.

### Синтаксис DSNTIAR

```
CALL DSNTIAR,(sqlca, сообщение, lrec),MF=(E,PARM)
```

Параметры DSNTIAR означают следующее:

*sqlca*

Область связи SQL.

*сообщение*

Область вывода, заданная как строка переменной длины, в которую DSNTIAR помещает текстовые сообщения. ее минимальное значение – 240.

В эту область помещаются строки вывода текста (каждая строка должна иметь длину, заданную в *lrec*). Например, формат области вывода можно задать так:

```

LINES      EQU      10
LRECL      EQU      132
:
MESSAGE   DS      H,CL(LINES*LRECL)
          ORG      MESSAGE
MESSAGE1  DC      AL2(LINES*LRECL)
MESSAGE1  DS      CL(LRECL)      строка текста 1
MESSAGE2  DS      CL(LRECL)      строка текста 2
:
MESSAGEn DS      CL(LRECL)      строка текста n
:
CALL DSNTIAR,(SQLCA, MESSAGE, LRECL),MF=(E,PARM)
  
```

где MESSAGE – имя области вывода сообщения, LINES – число строк в области вывода сообщения и LRECL – длина каждой строки.

*lrec*

Полное слово, содержащее длину логической записи сообщений вывода (от 72 до 240).

Выражение MF=(E,PARM) – параметр макрокоманды MV, задающий динамическое выполнение. PARM – это имя области данных, содержащей список указателей параметров вызова DSNTIAR.

Пример вызова DSNTIAR из программысмотрите в примере DB2 программы ассемблера DSNTIAR библиотеки префикс. SDSNSAMP. Инструкции о доступе к исходному тексту этого примера программы и о его выводесмотрите в разделе Приложение В, “Примеры прикладных программ” на стр. 883.

### CICS

Если в программе CICS требуется обработка памяти CICS, вместо DSNTIAR нужно использовать подпрограмму DSNTIAC. DSNTIAC имеет следующий синтаксис:

```
CALL DSNTIAC,(eib,commarea,sqlca,msg,lrecl),MF=(E,PARM)
```

В DSNTIAC имеются дополнительные параметры для вызовов подпрограмм, использующих команды CICS.

*eib* блок интерфейса EXEC

*commarea* область связи

Дополнительную информацию об этих новых параметрахсмотрите в соответствующем руководстве по программированию для CICS. Описание остальных параметров аналогично параметрам DSNTIAR. И DSNTIAC, и DSNTIAR форматируют SQLDA одним и тем же способом.

DSNTIA1 необходимо определить в CSD. При загрузке DSNTIAR или DSNTIAC их также нужно задать в CSD. Пример операторов генерации записей CSD для использования с DSNTIACсмотрите в компоненте DSN8FRDO набора данных префикс. SDSNSAMP.

Исходный текст на ассемблере для DSNTIAC и задания DSNTIJ5A для трансляции и компоновки DSNTIAC также находятся в наборе данных префикс. SDSNSAMP.

## Макрокоманды для программ на ассемблере

Все возможные макрокоманды DB2 находятся в наборе данных DSN610.SDSNMACS.

## Кодирование операторов SQL в программах на языках С или С<sup>++</sup>

В этом разделе рассказывается о способах программирования для кодирования операторов SQL в программах на языках С или С<sup>++</sup>. В этой книге под термином С, если не указано иного, понимается С/370 или С<sup>++</sup>.

## Определение области связи SQL

В программе на языке С с операторами SQL должна присутствовать хотя бы одна из двух переменных хоста:

- Переменная SQLCODE, объявленная как длинное целое. Например:  
`long SQLCODE;`
- Переменная SQLSTATE, объявленная как символьный массив длины 6. Например:  
`char SQLSTATE[6];`

Или

- SQLCA, содержащая переменные SQLCODE и SQLSTATE.

DB2 устанавливает значения SQLCODE и SQLSTATE после выполнения каждого оператора SQL. Прикладная программа может проверять значения этих переменных для определения успешности выполнения последнего оператора SQL. Все операторы SQL в программе должны находиться внутри области объявления переменных SQLCODE и SQLSTATE.

Переменные SQLCODE или SQLSTATE задаются, если задана опция препроцессора STDSQL(YES) (в соответствии со стандартом SQL); SQLCA задается, если задана опция препроцессора STDSQL(NO) (в соответствии с правилами DB2).

### Если указывается STDSQL(YES)

При использовании параметра препроцессора STDSQL(YES) не следует определять SQLCA. Если она будет определена, DB2 проигнорирует SQLCA, и определение SQLCA будет выдавать ошибки при компиляции.

Если вы объявляете переменную SQLSTATE, она не должна быть элементом структуры. Переменные хоста SQLCODE и SQLSTATE необходимо объявлять в объявлениях программы между операторами BEGIN DECLARE SECTION и END DECLARE SECTION.

### Если указывается STDSQL(NO)

При использовании опции препроцессора STDSQL(NO) включайте SQLCA явным образом. В программе на языке С можно закодировать SQLCA непосредственно или при помощи оператора SQL INCLUDE. Для SQL INCLUDE требуется стандартное объявление SQLCA:

```
EXEC SQL INCLUDE SQLCA;
```

Стандартное объявление включает в себя как определение структуры, так и область статических данных под названием 'sqlca'. Глава 6 книги *DB2 SQL Reference* содержит дополнительную информацию об операторе INCLUDE. Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLCA.

## Определение областей дескрипторов SQL

SQLDA требуется для следующих операторов:

- CALL...USING DESCRIPTOR *имя дескриптора*
- DESCRIBE *имя\_оператора* INTO *имя\_дескриптора*
- DESCRIBE CURSOR *переменная\_хоста* INTO *имя\_дескриптора*
- DESCRIBE INPUT *имя\_оператора* INTO *имя\_дескриптора*

- DESCRIBE PROCEDURE *переменная\_хоста* INTO *имя\_дескриптора*
- DESCRIBE TABLE *переменная\_хоста* INTO *имя\_дескриптора*
- EXECUTE...USING DESCRIPTOR *имя\_дескриптора*
- FETCH...USING DESCRIPTOR *имя\_дескриптора*
- OPEN...USING DESCRIPTOR *имя\_дескриптора*
- PREPARE...INTO *имя\_дескриптора*

В отличие от SQLCA, в программе может быть несколько SQLDA, причем SQLDA может иметь любое действительное имя. SQLDA в программе на языке С можно кодировать либо непосредственно, либо с помощью оператора SQL INCLUDE. Для оператора SQL INCLUDE требуется стандартное объявление SQLDA:

```
EXEC SQL INCLUDE SQLDA;
```

Стандартное объявление содержит только определение структуры с именем 'sqlda'. Глава 6 книги *DB2 SQL Reference*, содержит дополнительную информацию об операторе INCLUDE. Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLDA.

Если вы не используете опцию прекомпилятора TWOPASS, объявления SQLDA необходимо поместить перед первым оператором SQL, ссылающимся на описатель данных. Объявление SQLDA можно поместить в любое место, где в С допускается определение структуры. Применяются обычные правила области видимости данных в С.

## Встраивание операторов SQL

Операторы SQL можно вставлять в любое место программы на языке С, где допускается использование выполняемых операторов.

Каждый оператор SQL в программе на языке С должен начинаться с EXEC SQL и кончаться точкой с запятой (;). Оба ключевых слова EXEC и SQL должны находиться в одной строке, а остальная часть оператора может находиться в следующих строках.

Поскольку в С регистр символов существен, для ввода любых слов SQL необходимо использовать буквы нижнего регистра. Необходимо также соблюдать регистр имен переменных хоста по всей программе. Если, например, имя переменной хоста в ее объявлении записано в нижнем регистре, оно должно записываться в нижнем регистре во всех операторах SQL.

Оператор UPDATE в программе на языке С можно кодировать так:

```
EXEC SQL
UPDATE DSN8610.DEPT
SET MGRNO = :mgr_num
WHERE DEPTNO = :int_dept;
```

**Комментарии:** Комментарии С /\* ... \*/ можно включать в операторы SQL в любом месте, где допустим пробел, но только не между ключевыми словами EXEC и SQL. Одностроковые комментарии (начинающиеся с //) можно использовать в операторах языка С, но не во встроенном SQL. Недопустимо вкладывать одни комментарии в другие.

Чтобы включить в комментарий символы DBCS, необходимо ограничить эти символы управляющими символами переключения shift-out и shift-in. Первый символ shift-in в последовательности DBCS означает конец последовательности DBCS. Комментарии SQL можно включать в любой встроенный оператор SQL, если указана опция препроцессора STDSQL(YES).

**Продолжение операторов SQL:** Чтобы продолжить строково-символьную константу или разделяющий идентификатор в следующей строке, можно использовать обратную косую черту.

**Объявление таблиц и производных таблиц:** Программа на языке C должна использовать оператор DECLARE TABLE для описания каждой таблицы и производной таблицы, к которой обращается программа. Сгенерировать операторы DECLARE TABLE можно при помощи генератора объявлений DB2 (DCLGEN). Подробностисмотрите в разделе “Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN” на стр. 133.

**Включение кода:** Чтобы включить операторы SQL или операторы объявлений переменных хоста на языке C из компонента секционированного набора данных, вставьте в исходный текст, куда вы хотите включить эти операторы, оператор SQL:

```
EXEC SQL INCLUDE имя_элемента;
```

Нельзя использовать вложенные операторы SQL INCLUDE. Не пользуйтесь операторами C #include для включения операторов SQL или объявлений переменных хоста на языке C.

**Поля:** Код операторов SQL записывайте с 1 по 72 позицию, если для препроцессора DB2 не указаны другие рамки. Если EXEC SQL не попадает в указанные рамки, препроцессор DB2 не сможет распознать оператор SQL.

**Имена:** Для переменной хоста можно использовать любое действительное имя C со следующими ограничениями:

- Не используйте символы DBCS.
- Не используйте имена внешних элементов или имена планов доступа, начинающиеся с 'DSN', а также имена переменных хоста, начинающиеся с 'SQL' (с любой комбинацией букв верхнего и нижнего регистров). Эти имена зарезервированы для DB2.

**Пустые значения и символ NUL:** В C и SQL слово null используется в разных смыслах. В языке C имеются пустой символ (NUL), пустой указатель (NULL), и пустой оператор (просто точка с запятой). NUL в C является одиночным символом, приравниваемым к 0. NULL в C является особым зарезервированным значением указателя, которое не указывает ни на один действительный объект данных. Пустое значение в SQL – это особое значение, отличное от всех непустых значений и обозначающее отсутствие (непустого) значения. В этой главе NUL означает пустой символ в C, а NULL – пустое значение в SQL.

**Порядковые номера:** Генерируемые препроцессором DB2 исходные операторы не включают в себя порядковые номера.

**Метки операторов:** При желании перед операторами SQL можно ставить метки.

**Триграфы:** Некоторые символы из набора символов С недоступны при любых раскладках клавиатуры. Такие символы можно ввести в исходную программу на языке С с помощью последовательности трех символов, которая называется *триграф*. DB2 поддерживает те же самые триграфы, что и компилятор C/370.

**Оператор *WHenever*:** Цель условия GOTO в операторе SQL WHENEVER должна находиться внутри области действия любых операторов SQL, на которые воздействует оператор WHENEVER.

#### **Особые возможности С:**

- Использование многозадачности С/370, когда несколько задач выполняют операторы SQL, ведет к непредсказуемым результатам.
- Перед запуском препроцессора С необходимо задействовать прекомпилятор DB2.
- Прекомпилятор DB2 не поддерживает директивы препроцессора С.
- При использовании условных директив компилятора, содержащих код С, поместите их либо после первой метки С в прикладной программе, либо включите их в программу С при помощи директивы препроцессора #include.

Дополнительную информацию о директивах препроцессора Ссмотрите в соответствующей документации по языку С.

## **Использование переменных хоста**

Каждую переменную хоста необходимо явно объявить до ее первого употребления в операторе SQL. При указании опции прекомпилиатора TWOPASS необходимо объявить каждую переменную хоста до ее первого употребления в операторе DECLARE CURSOR.

Перед операторами С, определяющими переменные хоста, ставьте оператор BEGIN DECLARE SECTION, а после них – оператор END DECLARE SECTION. Программа может содержать несколько разделов объявлений переменных хоста.

Перед всеми переменными хоста в операторе SQL должно стоять двоеточие (:).

Имена переменных хоста должны быть уникальными в пределах программы, даже если переменные хоста находятся в разных блоках, классах или процедурах. Имена переменных хоста можно дополнить именем структуры, сделав их таким образом уникальными.

Оператор SQL, использующий переменную хоста, должен находиться внутри области действия этой переменной.

Переменные хоста должны быть скалярными переменными или структурами хоста; они не могут быть элементами векторов или массивов (индексированными переменными), если только для строк не используются

символьные массивы. Можно использовать массив переменных–индикаторов, если вы связываете его со структурой хоста.

## Объявление переменных хоста

Только некоторые из допустимых объявлений в С допустимы для объявления переменных хоста. Если объявление для некоторой переменной недействительно, любой оператор SQL, ссылающийся на эту переменную, может вызвать появление сообщения "UNDECLARED HOST VARIABLE" (необъявленная переменная хоста).

**Числовые переменные хоста:** Ниже приводится пример синтаксиса допустимых объявлений числовых переменных хоста.

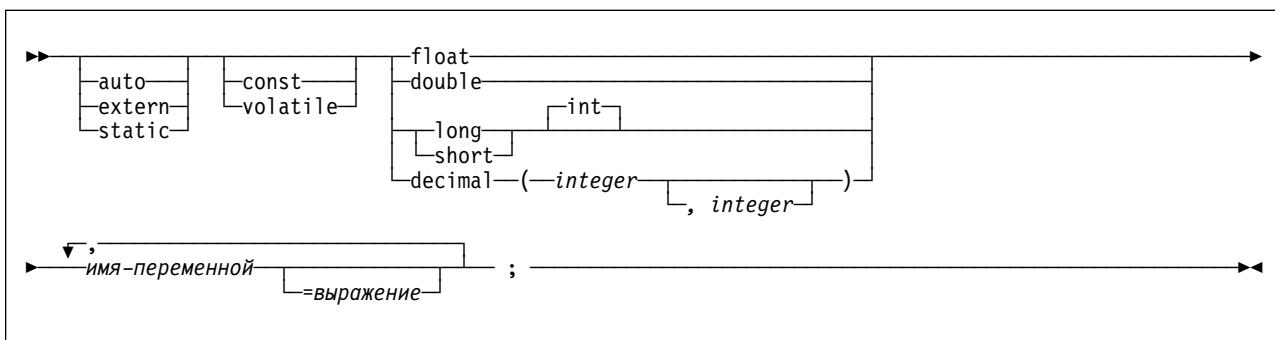


Рисунок 26. Числовые переменные хоста

**Символьные переменные хоста:** Существуют четыре допустимые формы символьных переменных хоста:

- Односимвольная форма
- Форма с символом–ограничителем NUL
- Структурированная форма VARCHAR
- CLOB

На следующих рисунках показан синтаксис этих форм, за исключением CLOB. Синтаксис CLOB смотрите на рис. 35 на стр. 169.

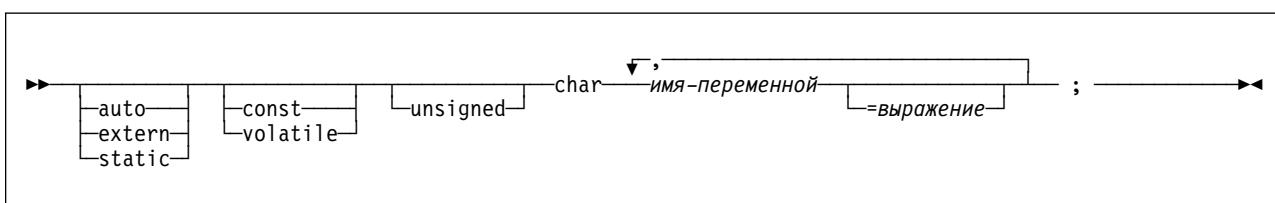


Рисунок 27. Односимвольная форма

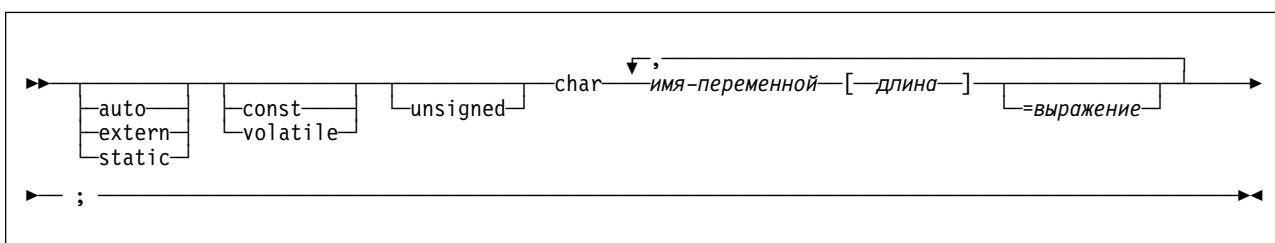


Рисунок 28. Форма с символом–ограничителем NUL

**Примечания:**

1. При вводе содержащаяся в переменной строка должна быть строкой с символом—ограничителем NUL.
2. При выводе строка является строкой с символом—ограничителем NUL.
3. Строке с символом—ограничителем NUL в переменной хоста соответствует символьной строке переменной длины (без NUL).

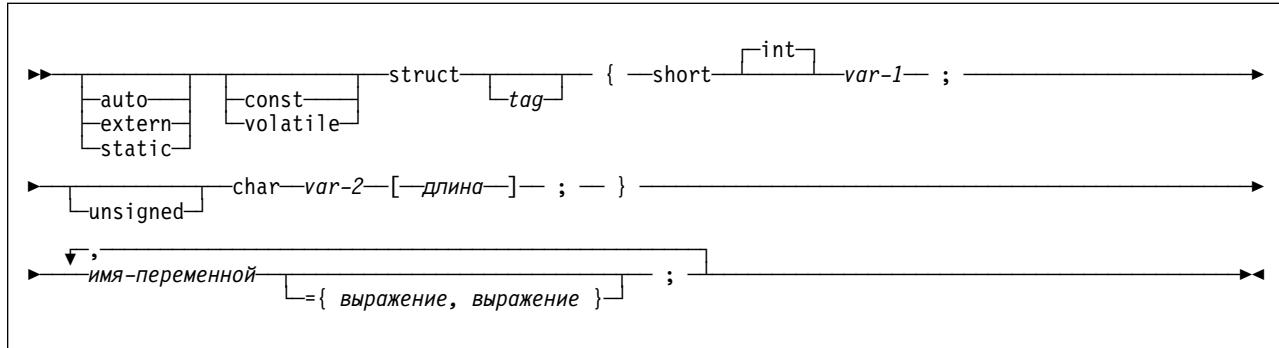


Рисунок 29. Структурированная форма VARCHAR

**Примечания:**

- *var-1* и *var-2* должны быть простыми ссылками на переменные. Их нельзя использовать в качестве переменных хоста.
- Можно использовать тег *struct*, чтобы определить другие области данных, которые нельзя использовать как переменные хоста.

**Пример:**

```

EXEC SQL BEGIN DECLARE SECTION;

/* правильное объявление переменной хоста vstring */

struct VARCHAR {
    short len;
    char s[10];
} vstring;

/* неправильное объявление переменной хоста wstring */

struct VARCHAR wstring;
  
```

**Графические переменные хоста:** Существуют четыре допустимые формы графических переменных хоста:

- Простая графическая форма
- Графическая форма с символом—ограничителем NUL
- Структурированная форма VARGRAPHIC
- DBCLOB

Тип данных C *wchar\_t* можно использовать для определения переменной хоста, которая используется при вставке, изменении, удалении и получении данных столбцов GRAPHIC или VARGRAPHIC.

Ниже показан синтаксис этих форм, за исключением DBCLOB. Синтаксис DBCLOB смотрите на рис. 35 на стр. 169.

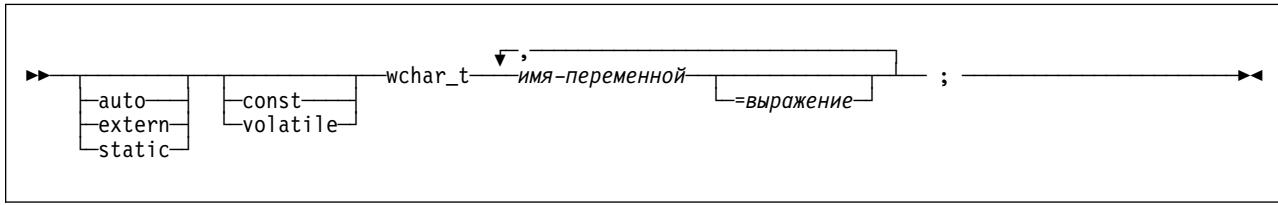


Рисунок 30. Простая графическая форма

Простая графическая форма объявляет графическую последовательность фиксированной длины, равной 1. В параметре *имя\_переменной* нельзя использовать обозначение массива.

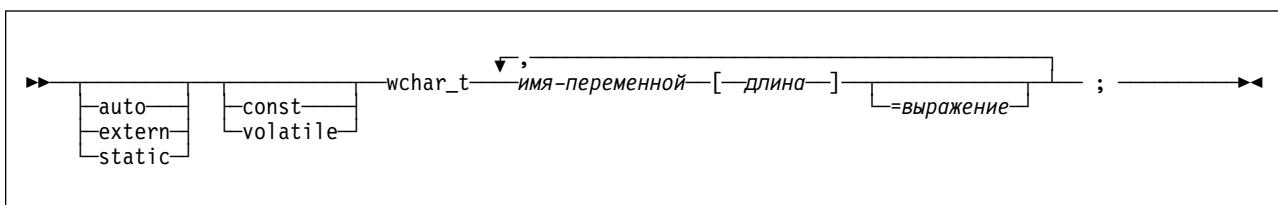


Рисунок 31. Графическая форма с символом-ограничителем NUL

#### Примечания:

1. *длина* должна быть десятичной целой константой от 2 до 16352.
2. При вводе строки *имя\_переменной* должна быть строкой с символом-ограничителем NUL.
3. При выводе эта строка будет строкой с символом-ограничителем NUL.
4. Графическая форма с символом-ограничителем NUL не принимает однобайтных символов в *имя\_переменной*.

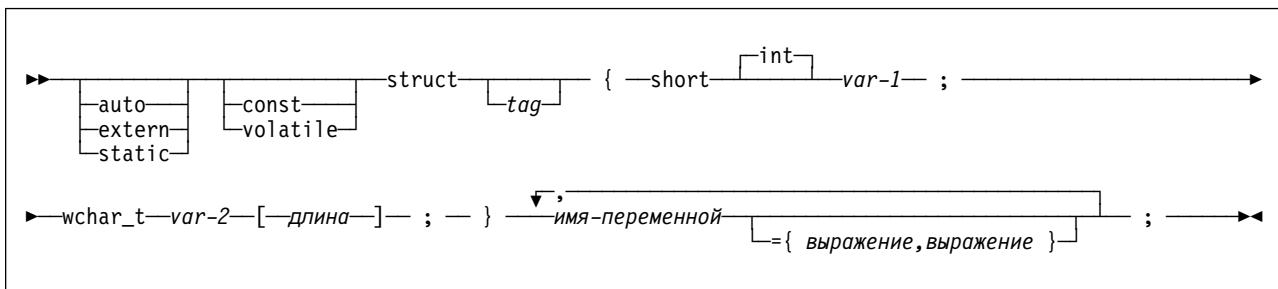


Рисунок 32. Структурированная форма VARGRAPHIC

#### Примечания:

- *длина* должна быть десятичной целой константой от 2 до 16352.
- *var-1* должна быть меньше или равной *длины*.
- *var-1* и *var-2* должны быть простыми ссылками на переменные. Нельзя использовать их в качестве переменных хоста.
- Тег struct можно использовать для определения других областей данных, которые нельзя использовать как переменные хоста.

**Пример:**

```

EXEC SQL BEGIN DECLARE SECTION;

/* правильное объявление переменной хоста vgraph */

struct VARGRAPH {
    short len;
    wchar_t d[10];
} vgraph;

/* неправильное объявление переменной хоста wgraph */

struct VARGRAPH wgraph;

```

**Локаторы набора результатов:** Ниже показан синтаксис объявлений локаторов наборов результатов. Применение этих переменных хоста обсуждается в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

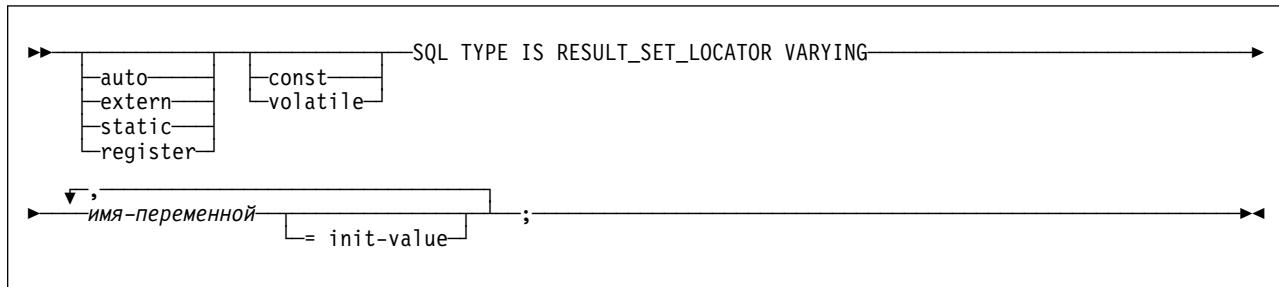


Рисунок 33. Локаторы набора результатов

**Локаторы таблиц:** Ниже показан синтаксис объявлений локаторов таблиц. Применение этих переменных хоста обсуждается в разделе “Доступ к таблицам переходов в пользовательских функциях” на стр. 307.

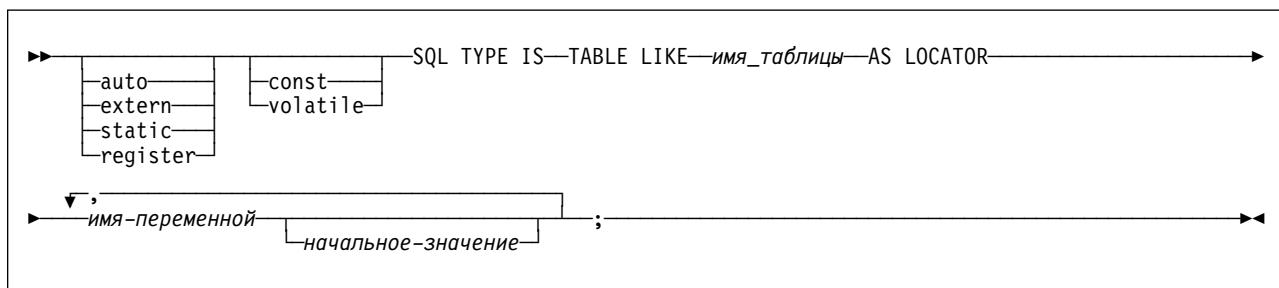


Рисунок 34. Локаторы таблиц

**Переменные и локаторы LOB:** Ниже показан синтаксис объявлений переменных хоста и локаторов BLOB, CLOB и DBCLOB. Применение этих переменных хоста обсуждается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

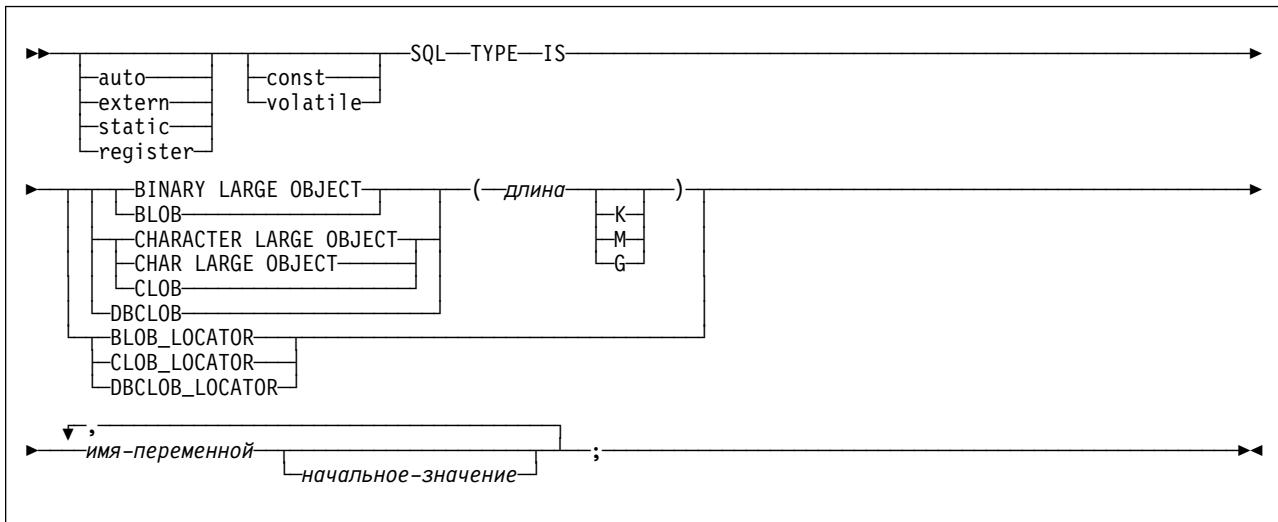


Рисунок 35. Переменные и локаторы LOB

**ROWID:** Ниже показан синтаксис объявлений переменных ROWID.  
Применение этих переменных хоста обсуждается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

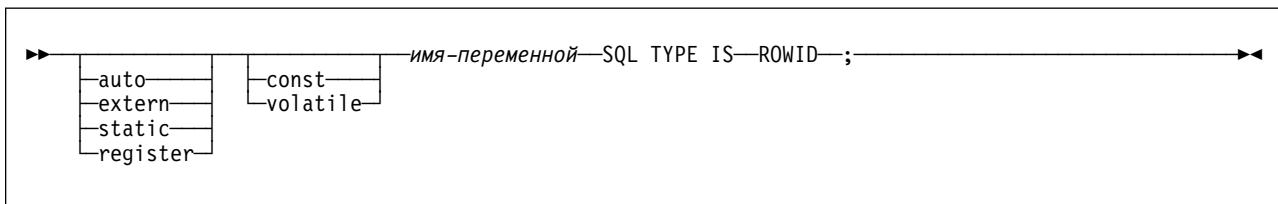


Рисунок 36. Переменные ROWID

## Применение структур хоста

Структура хоста в С содержит упорядоченную группу полей данных.  
Например:

```

struct {char c1[3];
        struct {short len;
                 char data[5];
                 }c2;
        char c3[2];
    }target;
    
```

В этом примере, *target* – имя структуры хоста, состоящей из полей *c1*, *c2* и *c3*. *c1* и *c3* – символьные массивы, а *c2* – переменная хоста, эквивалентная типу данных SQL VARCHAR. Структура хоста *target* может быть частью другой структуры хоста, но при этом должна быть самым глубоким уровнем вложения в структуре.

Следующая иллюстрация демонстрирует синтаксис допустимых структур хоста.

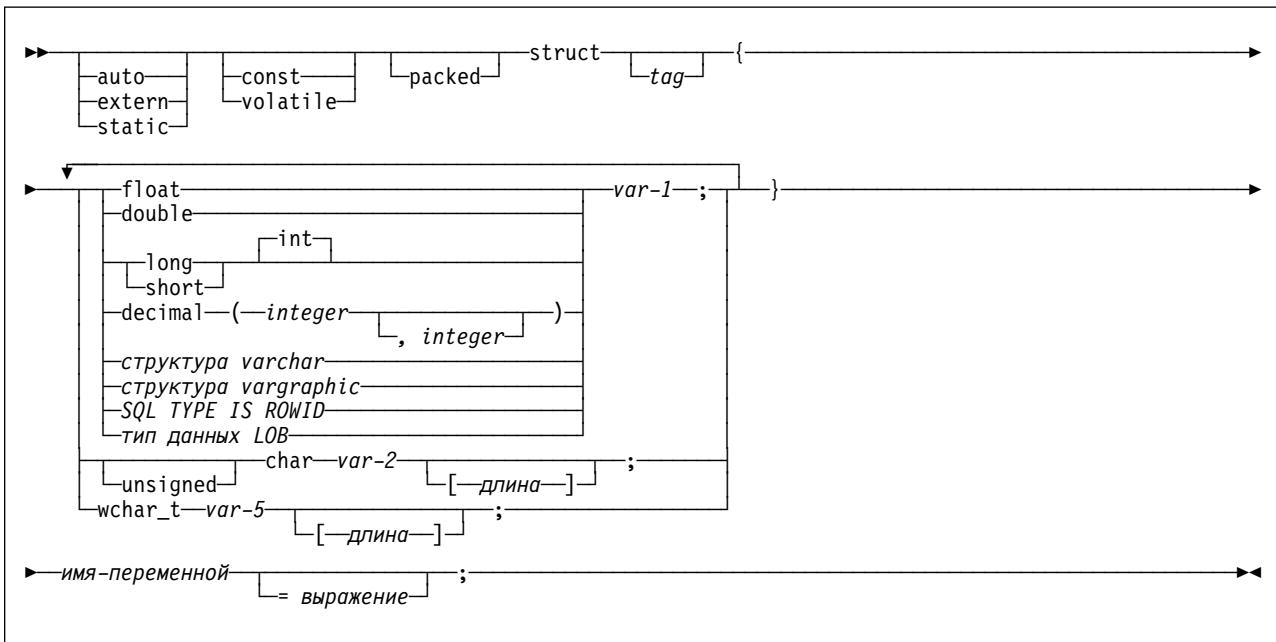


Рисунок 37. Структуры хоста

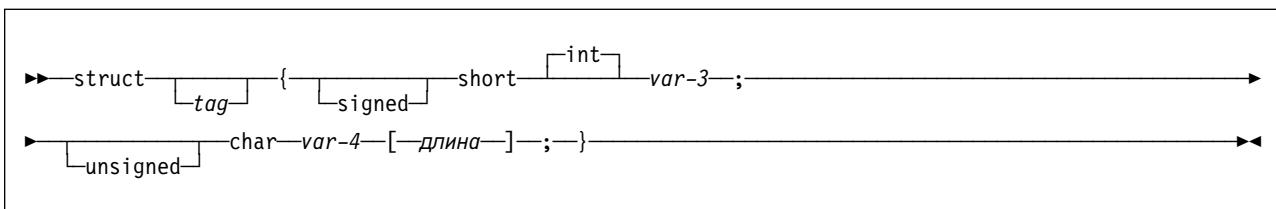


Рисунок 38. Структура varchar

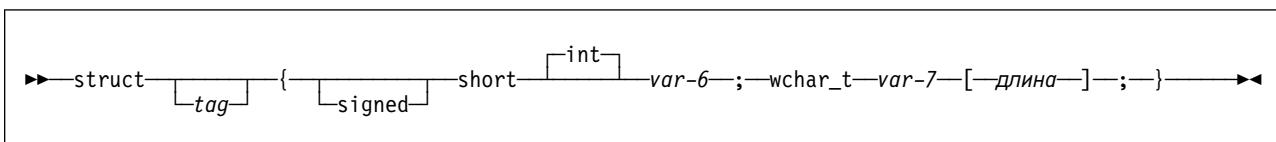


Рисунок 39. Структура VARGRAPHIC

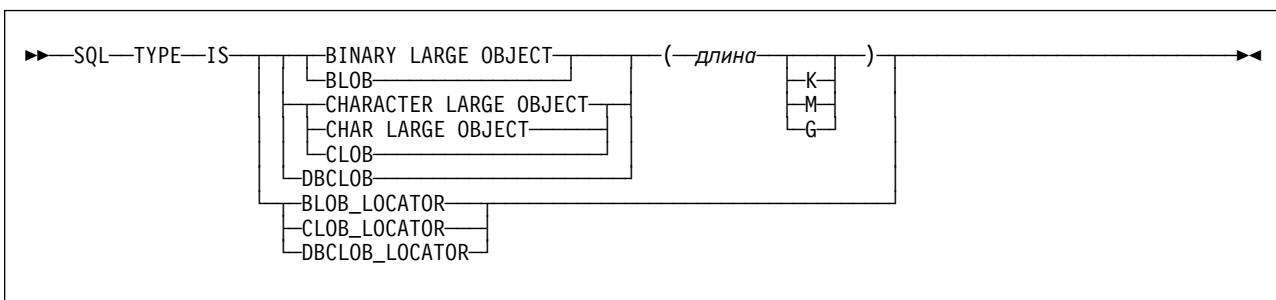


Рисунок 40. Тип данных LOB

## Определение эквивалентных типов данных в SQL и C

Табл. 10 описывает тип данных SQL и базовые значения SQLTYPE и SQLLEN, используемые препроцессором для переменных хоста, которые он находит в операторах SQL. Если переменная хоста появляется вместе с переменной—индикатором, SQLTYPE – это базовый SQLTYPE плюс 1.

Таблица 10 (Стр. 1 из 2). Типы данных SQL, используемые препроцессором для объявлений C

Тип данных C	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
short int	500	2	SMALLINT
long int	496	4	INTEGER
decimal( <i>p,s</i> ) <sup>1</sup>	484	р в байте 1, s в байте 2	DECIMAL( <i>p,s</i> ) <sup>1</sup>
float	480	4	FLOAT (одинарной точности)
double	480	8	FLOAT (двойной точности)
Односимвольная форма	452	1	CHAR(1)
символьная форма с символом—ограничителем NUL	460	<i>n</i>	VARCHAR ( <i>n</i> –1)
строковая форма VARCHAR <i>1&lt;=n&lt;=255</i>	448	<i>n</i>	VARCHAR( <i>n</i> )
строковая форма VARCHAR <i>n&gt;255</i>	456	<i>n</i>	VARCHAR( <i>n</i> )
Простая графическая форма	468	1	GRAPHIC(1)
графическая форма с символом—ограничителем NUL (wchar_t)	400	<i>n</i>	VARGRAPHIC ( <i>n</i> –1)
строковая форма VARGRAPHIC <i>1&lt;=n&lt;128</i>	464	<i>n</i>	VARGRAPHIC( <i>n</i> )
строковая форма VARGRAPHIC <i>n&gt;127</i>	472	<i>n</i>	VARGRAPHIC( <i>n</i> )
SQL TYPE IS RESULT_SET_LOCATOR	972	4	Локатор набора результатов <sup>2</sup>
SQL TYPE IS TABLE LIKE имя_таблицы AS LOCATOR	976	4	Локатор таблиц <sup>2</sup>
SQL TYPE IS BLOB_LOCATOR	960	4	Локатор BLOB <sup>2</sup>
SQL TYPE IS CLOB_LOCATOR	964	4	Локатор CLOB <sup>2</sup>
SQL TYPE IS DBCLOB_LOCATOR	968	4	Локатор DBCLOB <sup>2</sup>

Таблица 10 (Стр. 2 из 2). Типы данных SQL, используемые прекомпилятором для объявлений С

Тип данных С	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
SQL TYPE IS BLOB( <i>n</i> ) $1 \leq n \leq 2147483647$	404	<i>n</i>	BLOB( <i>n</i> )
SQL TYPE IS CLOB( <i>n</i> ) $1 \leq n \leq 2147483647$	408	<i>n</i>	CLOB( <i>n</i> )
SQL TYPE IS DBCLOB( <i>n</i> ) $1 \leq n \leq 1073741823$	412	<i>n</i>	DBCLOB( <i>n</i> ) <sup>3</sup>
SQL TYPE IS ROWID	904	40	ROWID

**Примечания:**

1. *p* – точность по терминологии SQL, определяемая как общее число цифр. В С ее называют *размер*.
2. *s* – масштаб по терминологии SQL, определяемый как число цифр справа от десятичной точки. В С называется *точность*.
3. *n* – число двубайтных символов.

Табл. 11 поможет вам определить переменные хоста, получающие выходные данные от базы данных. Для определения типа данных С, эквивалентного заданному типу данных SQL, можно использовать таблицу. Например, если надо обработать данные типа TIMESTAMP, можно по таблице определить подходящий тип переменной хоста в программе, принимающей значение данных.

Таблица 11 (Стр. 1 из 3). Типы данных SQL, преобразуемые в типовые объявления С

SQL	Тип данных С	Примечания
TINYINT	short int	
INTEGER	long int	
DECIMAL( <i>p,s</i> ) или NUMERIC( <i>p,s</i> )	decimal	Если компилятор С не поддерживает тип данных decimal, можно использовать тип данных double; однако double – не точный эквивалент.
REAL или FLOAT( <i>n</i> )	float	$1 \leq n \leq 21$
DOUBLE PRECISION или FLOAT ( <i>n</i> )	double	$22 \leq n \leq 53$
CHAR(1)	односимвольная форма	
CHAR( <i>n</i> )	точного эквивалента нет	При $n > 1$ используйте форму с символом–ограничителем NUL

Таблица 11 (Стр. 2 из 3). Типы данных SQL, преобразуемые в типовые объявления C

SQL Тип данных	Тип данных C	Примечания
VARCHAR( <i>n</i> )	NUL-терминированная симв. форма	Если данные могут содержать символьные NUL (\0), используйте структурированную форму VARCHAR. Предусмотрите не менее <i>n</i> +1 символов для размещения символа—ограничителя NUL.
	структурная форма VARCHAR	
GRAPHIC(1)	простая графическая форма	
GRAPHIC( <i>n</i> )	точного эквивалента нет	При <i>n</i> >1 используйте графическую форму с символом—ограничителем NUL. <i>n</i> – число двубайтных символов.
VARGRAPHIC( <i>n</i> )	графическая форма с символом—ограничителем NUL	Если данные могут содержать графические значения NUL (\0\0), используйте структурированную форму VARGRAPHIC. Предусмотрите не менее <i>n</i> +1 символов для размещения символа—ограничителя NUL. <i>n</i> – число двубайтных символов.
	структурная форма VARGRAPHIC	<i>n</i> – число двубайтных символов.
DATE	Символьная форма с символом—ограничителем NUL	При использовании процедуры выхода по данным длина определяется этой процедурой. В противном случае надо задать по крайней мере 11 символов, чтобы поместился символ—ограничитель NUL.
	структурная форма VARCHAR	При использовании процедуры выхода по данным длина определяется этой процедурой. В противном случае отведите не менее 10 символов.
TIME	NUL-терминированная симв. форма	При использовании обработчика времени длина определяется этим обработчиком. В противном случае длина должна составлять не менее 7; для включения секунд длина должна составлять не менее 9, считая символ NUL.
	структурная форма VARCHAR	При использовании обработчика времени длина определяется этим обработчиком. В противном случае длина должна составлять не менее 6; для включения секунд длина должна составлять не менее 8.

Таблица 11 (Стр. 3 из 3). Типы данных SQL, преобразуемые в типовые объявления С

SQL	Тип данных С	Примечания
TIMESTAMP	NUL-терминированная симв. форма	Длина должна составлять не менее 20. Для включения микросекунд длина должна составлять 27. Если длина меньше 27, произойдет усечение микросекунд.
	структурированная форма VARCHAR	Длина должна составлять не менее 19. Для включения микросекунд длина должна составлять 26. Если длина меньше 26, произойдет усечение микросекунд.
Локатор набора результатов	SQL TYPE IS RESULT_SET_LOCATOR	Этот тип данных можно использовать только для получения наборов результатов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор таблиц	SQL TYPE IS TABLE LIKE имя-таблицы AS LOCATOR	Этот тип данных используется только в пользовательской функции или в хранимой процедуре для приема строк таблицы переходов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор BLOB	SQL TYPE IS BLOB_LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов BLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор CLOB	SQL TYPE IS CLOB_LOCATOR	Пользуйтесь этим типом данных только для обработки данных столбцов CLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор DBCLOB	SQL TYPE IS DBCLOB_LOCATOR	Пользуйтесь этим типом данных только для обработки данных столбцов DBCLOB. Нельзя использовать этот тип данных в качестве типа столбца.
BLOB( <i>n</i> )	SQL TYPE IS BLOB( <i>n</i> )	$1 \leq n \leq 2147483647$
CLOB( <i>n</i> )	SQL TYPE IS CLOB( <i>n</i> )	$1 \leq n \leq 2147483647$
DBCLOB( <i>n</i> )	SQL TYPE IS DBCLOB( <i>n</i> )	<i>n</i> – число двубайтных символов. $1 \leq n \leq 1073741823$
ROWID	SQL TYPE IS ROWID	

**Замечания об объявлении и использовании переменных С**

При объявлении переменных С нужно учитывать следующее.

**Типы данных С, не имеющие эквивалентов в SQL:** С поддерживает ряд типов данных и классов памяти, не имеющих эквивалентов в SQL, например, классы с хранением в регистрах, `typedef` и указатели.

**Типы данных SQL, не имеющие эквивалентов в С:** Если компилятор С не поддерживает тип данных `decimal`, то для типа данных SQL DECIMAL нет

точного эквивалента. В этом случае для хранения значения такой переменной можно использовать:

- Целочисленную переменную или переменную с плавающей точкой, к которой преобразуется это значение. При выборе типа integer вы потеряете дробную часть числа. Если десятичное число может превзойти допустимый максимум для целого числа, или необходимо сохранить дробное значение, можно воспользоваться числами с плавающей точкой. Числа с плавающей точкой являются приближениями действительных чисел. Следовательно, когда вы присваиваете десятичное значение переменной с плавающей точкой, результат может отличаться от исходного числа.
- Символьные переменные хоста. Воспользуйтесь функцией CHAR для получения строкового представления десятичного числа.
- Функцию DECIMAL для явного преобразования величины в десятичный тип данных, как в следующем примере:

```
long duration=10100; /* 1 год и 1 месяц */
char result_dt[11];
```

```
EXEC SQL SELECT START_DATE + DECIMAL(:duration,8,0)
INTO :result_dt FROM TABLE1;
```

**Переменные хоста с плавающей точкой:** Все данные с плавающей точкой в DB2 хранятся в формате System/390. Однако ваши данные переменной хоста могут быть или в формате с плавающей точкой System/390 или в формате с плавающей точкой IEEE. DB2 использует опцию препроцессора FLOAT(S390|IEEE) для формата переменных хоста с плавающей точкой (IEEE или System/390). DB2 не проверяет, соответствует ли объявление переменной хоста или формат содержания переменной хоста опции препроцессора. Это значит, что вам надо следить за тем, чтобы формат данных с плавающей точкой соответствовал опции препроцессора.

**Типы данных C для особых целей:** Типы данных локаторов поддерживаются как в C, так и в SQL. Локаторы нельзя использовать в качестве типов столбцов. Информация о том, как использовать эти типы данных, приводится в следующих разделах:

**Локатор набора результатов** “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579

**Локатор таблиц** “Доступ к таблицам переходов в пользовательских функциях” на стр. 307

**Локаторы больших объектов** “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253

#### **Строковые переменные хоста:**

При присвоении строки длины *n* с символом—ограничителем NUL, длина которой:

- меньше или равна *n*, DB2 последовательно вставляет в переменную хоста (*n*–1) символов и добавляет NUL в конец строки. DB2 записывает в SQLWARN[1] *W*, а в заданную вами переменную—индикатор — первоначальную длину исходной строки.

- равна  $n+1$ , DB2 вставляет символы в переменную хоста и добавляет NUL в конец строки.
- больше  $n+1$ , правила зависят от того, является ли исходная строка значением столбца строк фиксированной или же переменной длины. Раздел Глава 3 книги *DB2 SQL Reference* содержит дополнительную информацию.

**Операторы *PREPARE* или *DESCRIBE*:** В операторе *PREPARE* или *DESCRIBE* нельзя использовать переменную хоста с символом—ограничителем NUL.

**L–литералы (константы):** DB2 допускает L–литералы в прикладных программах С. DB2 разрешает правильно составленные L–литералы, хотя и не проверяет все ограничения, налагаемые на L–литерал компилятором С. Для работы с L–литералом можно использовать графические строковые константы DB2 в операторах SQL. Не пользуйтесь L–литералами в операторах SQL.

**Переполнение:** Следует остерегаться переполнения. Пусть, например, значение столбца INTEGER считывается в целочисленную переменную хоста типа short, а значение столбца оказалось больше 32767. В зависимости от того, задана ли переменная–индикатор, вы получите предупреждение о переполнении или ошибку.

**Усечение:** Следует остерегаться усечения. Убедитесь, что объявленная вами переменная хоста может вместить данные и, если необходимо, символ—ограничитель NUL. Считывание десятичного или с плавающей точкой значения столбца в целочисленную переменную хоста типа long удалит всю дробную часть этого значения.

### Замечания о различиях в синтаксисе констант

Не забывайте о различиях в синтаксисе.

**Между десятичными константами и константами *REAL* (с плавающей точкой):** В С последовательность цифр с десятичной точкой интерпретируется как действительная константа. Такая же последовательность в операторе SQL интерпретируется как десятичная константа. Чтобы в операторе SQL указать действительную (с плавающей точкой) константу, необходимо воспользоваться экспоненциальной формой записи.

В С действительная (с плавающей точкой) константа может иметь суффикс f или F для указания типа данных *float* или суффикс l или L для указания типа данных *long double*. Константа с плавающей точкой в операторе SQL не должна использовать эти суффиксы.

**Целочисленные константы:** В С можно вводить целочисленные константы в шестнадцатеричном представлении, начинающиеся на 0x или 0X. Этую форму нельзя использовать в операторе SQL.

В С целочисленная константа может иметь суффикс i или U, указывающий, что константа не имеет знака. Целочисленная константа может иметь суффикс l или L, указывающий длинное целое. Нельзя использовать эти суффиксы в операторах SQL.

**Символьные и строковые константы:** В языке С символьные и строковые константы могут использовать escape–последовательности. Нельзя

использовать escape—последовательности в операторах SQL. Апострофы и кавычки имеют различный смысл в C и SQL. В C кавычки можно использовать для ограничения строковых констант, а апострофы – для ограничения символьных констант. Следующие примеры показывают применение кавычек и апострофов в C.

### **Кавычки**

```
printf( "%d lines read. \n", num_lines);
```

### **Апострофы**

```
#define NUL '\0'
```

В SQL кавычки можно использовать для ограничения идентификаторов, а апострофы – для ограничения строковых констант. Следующие примеры показывают применение апострофов и кавычек в SQL.

### **Кавычки**

```
SELECT "COL#1" FROM TBL1;
```

### **Апострофы**

```
SELECT COL1 FROM TBL1 WHERE COL2 = 'BELL';
```

Символьные данные в SQL отличны от целочисленных данных. Символьные данные в C являются подтипов целочисленных данных.

## **Определение совместимости типов данных SQL и C**

Переменные хоста C, используемые в операторах SQL, должны быть совместимы по типу со столбцами, в которых предполагается их использование:

- Числовые типы данных совместимы друг с другом: Столбец SMALLINT, INTEGER, DECIMAL или FLOAT совместим с любой переменной хоста C, определенной как тип short int, long int, decimal, float или double.
- Символьные типы данных совместимы друг с другом: Столбец CHAR, VARCHAR или CLOB совместим с односимвольной формой, формой с символом–ограничителем NUL или структурированной VARCHAR формой символьной переменной хоста C.
- Типы символьных данных частично совместимы с локаторами CLOB. Столбцу CHAR или VARCHAR можно назначить значение локатора CLOB, но значение из столбца CHAR или VARCHAR нельзя назначать переменной хоста типа локатора CLOB.
- Графические типы данных совместимы друг с другом. Столбец GRAPHIC, VARGRAPHIC или DBCLOB совместим с односимвольной формой, формой с символом–ограничителем NUL или структурированной VARGRAPHIC формой графической переменной хоста C.
- Типы графических данных частично совместимы с локаторами DBCLOB. Значение локатора DBCLOB может быть присвоено столбцу GRAPHIC или VARGRAPHIC, однако значение столбца GRAPHIC или VARGRAPHIC нельзя присвоить переменной хоста типа локатор DBCLOB.
- Типы данных число–время совместимы с символьными переменными хоста: Столбец DATE, TIME или TIMESTAMP совместим с односимвольной

формой, формой с символом—ограничителем NUL или структурированной VARCHAR формой символьной переменной хоста С.

- Столбец BLOB совместим только с переменной хоста BLOB.
- Столбцы ROWID совместимы только с переменными хоста ROWID.
- Переменная хоста совместима с пользовательским типом, если тип переменной хоста совместим с исходным типом этого пользовательского типа. Информация о назначении и сравнении пользовательских типов приводится в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

При необходимости DB2 автоматически преобразует строку фиксированной длины в строку переменной длины, и наоборот.

**Строки переменной длины:** Для данных *BIT* переменной длины пользуйтесь структурированной формой VARCHAR. Ряд функций обработки строк С обрабатывает строки с символом—ограничителем NUL, а другие функции обрабатывают другие строки. Функции обработки строк С, обрабатывающие строки с символом—ограничителем NUL, не могут обрабатывать битовые данные; эти функции могут неправильно интерпретировать символ NUL.

## Применение переменных–индикаторов

Переменная–индикатор – это двубайтное целое (short int). Если для переменной X задана переменная–индикатор, то, когда DB2 требуется занести в X пустое значение, в переменную–индикатор заносится отрицательное значение и X не обновляется. Ваша программа перед использованием X должна будет проверить переменную–индикатор. Если переменная–индикатор отрицательна, это значит, что X имеет пустое значение, а любое реально записанное в нее значение можно игнорировать.

Если программа хочет с помощью X назначить столбцу пустое значение, она должна присвоить переменной–индикатору отрицательное значение. Тогда DB2 присвоит столбцу пустое значение и проигнорирует любое значение, содержащееся в X.

Переменные–индикаторы объявляются так же, как и переменные хоста. Объявления обоих типов переменных можно чередовать в произвольном порядке. Дополнительную информацию о переменных–индикаторахсмотрите в разделе “Использование индикаторных переменных с переменными хоста” на стр. 113.

### Пример:

Задан оператор:

```
EXEC SQL FETCH CLS_CURSOR INTO :C1sCd,
                           :Day :DayInd,
                           :Bgn :BgnInd,
                           :End :EndInd;
```

Переменные можно объявить так:

```

EXEC SQL BEGIN DECLARE SECTION;
char ClsCd[8];
char Bgn[9];
char End[9];
short Day, DayInd, BgnInd, EndInd;
EXEC SQL END DECLARE SECTION;

```

Ниже приводится пример синтаксиса переменной–индикатора.

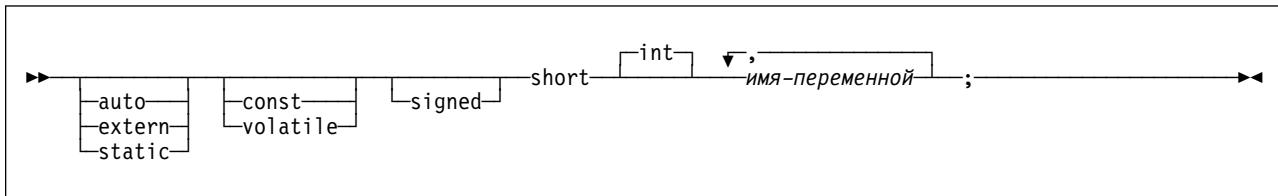


Рисунок 41. Переменная–индикатор

Ниже приводится пример синтаксиса массива индикаторов.

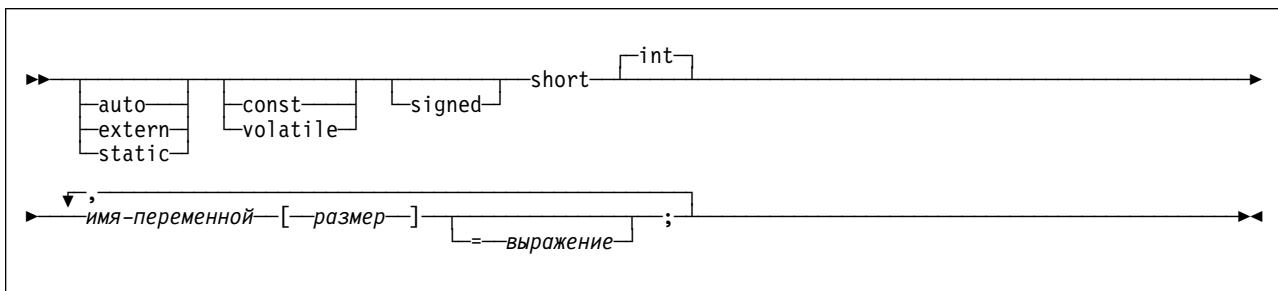


Рисунок 42. Массив индикаторов для структуры хоста

#### **Примечание:**

Размер должен быть целой константой от 1 до 32767.

## **Обработка кодов возврата ошибок SQL**

Для преобразования кода возврата SQL в текстовое сообщение можно воспользоваться подпрограммой DSNTIAR. DSNTIAR берет данные, содержащиеся в SQLCA, форматирует из них сообщение и помещает результат в назначеннную прикладной программой область вывода сообщений. Подробная информация о поведении DSNTIAR и связанные с ней понятия приводятся в разделе “Обработка кодов возврата ошибок SQL” на стр. 119.

#### **Синтаксис DSNTIAR**

```
rc = dsntiar(&sqlca, &сообщение, &lrec);
```

Параметры DSNTIAR означают следующее:

**&sqlca** Область связи SQL.

**&сообщение**

Область вывода, в формате VARCHAR, в которую DSNTIAR помещает текст сообщения. В первом полуслове содержится длина оставшейся области; ее минимальное значение – 240.

Выводимые строки текста. Каждая строка длины, указанной в *&lrec*, помещается в эту область. Например, формат области вывода можно задать так:

```
#define data_len 132
#define data_dim 10
struct error_struct {
    short int error_len;
    char error_text[data_dim][data_len];
} error_message = {data_dim * data_len};
:
rc = dsntiar(&sqlca, &error_message, &data_len);
```

где *error\_message* – имя области вывода сообщений, *data\_dim* – число строк в области вывода сообщений, а *data\_len* – длина каждой строки.

*&lrec* Полное слово, содержащее длину логической записи сообщений вывода (от 72 до 240).

Чтобы указать компьютеру, что DSNTIAR является программой на ассемблере, включите в свою прикладную программу один из следующих операторов.

Для С включите:

```
#pragma linkage (dsntiar,OS)
```

Для C++ включите оператор следующего вида:

```
extern "OS" short int dsntiar(struct sqlca *sqlca,
                               struct error_struct *error_message,
                               int *data_len);
```

В DB2 примеры вызова DSNTIAR из прикладной программы находятся в программе примера на языке С DSN8BD3 и в программе примера на C++ DSN8BE3. Эти программы содержатся в библиотеке DSN8610.SDSNSAMP. Как посмотреть или напечатать исходный текст программы примера, сказано в разделе Приложение В, “Примеры прикладных программ” на стр. 883.

**CICS**

Если в программе CICS требуется обработка памяти CICS, вместо DSNTIAR нужно использовать подпрограмму DSNTIAC. DSNTIAC имеет следующий синтаксис:

```
rc = DSNTIAC(&eib, &commarea, &sqlca, &message, &lrecl);
```

У DSNTIAC есть дополнительные параметры, применяемые при вызовах процедур, использующих команды CICS.

*&eib* блок интерфейса EXEC

*&commarea* область связи

Дополнительную информацию об этих новых параметрах смотрите в соответствующем руководстве по прикладному программированию для CICS. Описания остальных параметров те же, что и для DSNTIAR. И DSNTIAC, и DSNTIAR форматируют SQLDA одним и тем же способом.

DSNTIA1 необходимо определить в CSD. Если загружаются DSNTIAR или DSNTIAC, их также необходимо определить в CSD. В качестве примера операторов генерации ввода CSD для использования с DSNTIAC смотрите задание DSNTEJ5A.

Исходный текст DSNTIAC на ассемблере и задание DSNTEJ5A, которое ассемблирует и связывает DSNTIAC, находятся в наборе данных префикс.SDSNSAMP.

**Особенности C<sup>++</sup>**

При кодировании SQL в программе на C<sup>++</sup> имейте в виду следующее:

**Использование типов данных C<sup>++</sup> в качестве переменных хоста:** Можно использовать элементы класса в качестве переменных хоста. Используемые в качестве переменных хоста элементы класса доступны для любого оператора SQL в пределах данного класса.

Нельзя в качестве переменных хоста использовать объекты класса.

**Кодирование операторов SQL в программах на языке COBOL**

В этом разделе рассказывается о способах программирования для кодирования операторов SQL в программах на языке COBOL.

Эта информация относится ко всем компиляторам языка COBOL, поддерживаемых DB2 for OS/390, если не сказано иного.

**Задание области связи SQL**

В программу на языке COBOL, содержащую операторы SQL, нужно включить одну или обе из следующих переменных хоста:

- Переменную SQLCODE, объявляется как PIC S9(9) BINARY, PIC S9(9) COMP-4 или PICTURE S9(9) COMP
- Переменную SQLSTATE, объявляется как PICTURE X(5)

Либо

- SQLCA, которая содержит и переменную SQLCODE, и переменную SQLSTATE.

DB2 устанавливает значения для SQLCODE и SQLSTATE после выполнения каждого оператора SQL. По этим значениям переменных программа может определить, успешно ли завершился последний оператор SQL. Все операторы SQL в программе должны находиться в пределах области действия объявления переменных SQLCODE и SQLSTATE.

Переменные SQLCODE или SQLSTATE задаются, если задана опция препомпилиятора STDSQL(YES) (в соответствии со стандартом SQL); SQLCA задается, если задана опция препомпилиятора STDSQL(NO) (в соответствии с правилами DB2).

### **Если задано STDSQL(YES)**

При использовании параметра препроцессора STDSQL(YES) не следует определять SQLCA. Иначе DB2 проигнорирует SQLCA, а ваше определение SQLCA приведет к ошибкам времени компиляции.

При использовании опции препомпилиятора STDSQL(YES) нужно объявить переменную SQLCODE. DB2 сама объявляет область SQLCA в разделе программы WORKING-STORAGE SECTION. DB2 сама управляет этой SQLCA, поэтому в программе не должно быть предположений о ее структуре и местоположении.

Объявляемая переменная SQLSTATE не должна быть элементом структуры. Переменные хоста SQLCODE и SQLSTATE нужно объявить между операторами BEGIN DECLARE SECTION и END DECLARE SECTION в разделе объявлений программы.

### **Если задано STDSQL(NO)**

При использовании опции препомпилиятора STDSQL(NO) задайте SQLCA явным образом. В программе на языке COBOL SQLCA можно задать либо напрямую, либо с помощью оператора SQL INCLUDE. Оператор SQL INCLUDE запрашивает стандартное объявление SQLCA:

EXEC SQL INCLUDE SQLCA END-EXEC.

Оператор INCLUDE SQLCA или объявление для SQLCODE можно задать в WORKING-STORAGE SECTION, где в этом разделе можно задать уровень 77 или элемент описания записи. Переменную SQLCODE можно объявить автономно в разделах программы WORKING-STORAGE SECTION или LINKAGE SECTION.

Глава 6 книги *DB2 SQL Reference* содержит дополнительную информацию об операторе INCLUDE. Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLCA.

## Определение областей дескрипторов SQL

SQLDA требуется для следующих операторов:

- CALL...USING DESCRIPTOR *имя–дескриптора*
- DESCRIBE *имя–оператора* INTO *имя–дескриптора*
- DESCRIBE CURSOR *переменная–хоста* INTO *имя–дескриптора*
- DESCRIBE INPUT *имя–оператора* INTO *имя–дескриптора*
- DESCRIBE PROCEDURE *переменная–хоста* INTO *имя–дескриптора*
- DESCRIBE TABLE *переменная–хоста* INTO *имя–дескриптора*
- EXECUTE...USING DESCRIPTOR *имя–дескриптора*
- FETCH...USING DESCRIPTOR *имя–дескриптора*
- OPEN...USING DESCRIPTOR *имя–дескриптора*
- PREPARE...INTO *имя–дескриптора*

В отличие от SQLCA, в программе может быть несколько SQLDA, и SQLDA может иметь любое допустимое имя. Оператор DB2 SQL INCLUDE не обеспечивает отображение SQLDA для языка COBOL. SQLDA можно задать одним из следующих способов:

- SQLDA можно объявить в программах на языке COBOL, компилируемых любыми компиляторами, *за исключением* компилятора OS/VS COBOL. Дополнительную информацию смотрите в разделе “Использование динамического SQL в языке COBOL” на стр. 577. SQLDA нужно объявить в WORKING-STORAGE SECTION или LINKAGE SECTION, где в этих разделах можно задать элемент описания записи.
- Для программ на языке COBOL, компилируемых любыми компиляторами COBOL, можно вызвать подпрограмму (написанную на языке C, PL/I или ассемблере), которая для задания SQLDA использует оператор DB2 INCLUDE SQLDA. Такая подпрограмма также может содержать операторы SQL для любых динамических функций SQL, которые требуются. Этот метод нужно использовать при компиляции программы компилятором OS/VS COBOL. Компилятор OS/VS COBOL не поддерживает тип данных POINTER, используемый при задании SQLDA. Дополнительную информацию об использовании динамического SQL смотрите в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.

Объявление SQLDA нужно поместить перед первым оператором SQL, в котором есть ссылки на конкретный дескриптор данных. Оператор SQL, в котором используется переменная хоста, должен находиться в пределах области действия оператора, объявляющего данную переменную.

## Вставка операторов SQL

Операторы SQL можно кодировать в разделах программы на языке COBOL, перечисленных в Табл. 12.

*Таблица 12 (Стр. 1 из 2). Операторы SQL, допустимые в разделах программы на языке COBOL*

Оператор SQL	Раздел программы
BEGIN DECLARE SECTION END DECLARE SECTION	WORKING-STORAGE SECTION или LINKAGE SECTION
INCLUDE SQLCA	WORKING-STORAGE SECTION или LINKAGE SECTION

Таблица 12 (Стр. 2 из 2). Операторы SQL, допустимые в разделах программы на языке COBOL

Оператор SQL	Раздел программы
INCLUDE имя-текстового-файла	PROCEDURE DIVISION или DATA DIVISION <sup>1</sup>
DECLARE TABLE DECLARE CURSOR	DATA DIVISION или PROCEDURE DIVISION
Другие	PROCEDURE DIVISION

**Примечание:** <sup>1</sup> При объявлении переменных хоста оператор INCLUDE должен находиться в WORKING-STORAGE SECTION или в LINKAGE SECTION.

Операторы SQL нельзя помещать в разделе DECLARATIVES программы на языке COBOL.

Каждый оператор в программе на языке COBOL должен начинаться с EXEC SQL и заканчиваться END-EXEC. Если оператор SQL находится между двумя операторами языка COBOL, точка необязательна и ее можно не использовать. Если оператор SQL задается в конструкции IF...THEN операторов языка COBOL, ставьте конечную точку во избежание непредвиденного завершения оператора IF. Ключевые слова EXEC и SQL должны вводиться в одной строке, но остальную часть оператора можно написать в последующих строках.

Оператор UPDATE в программе на языке COBOL можно кодировать так:

```
EXEC SQL
UPDATE DSN8610.DEPT
SET MGRNO = :MGR-NUM
WHERE DEPTNO = :INT-DEPT
END-EXEC.
```

**Комментарии:** Строки комментария языка COBOL (\* в позиции 7) можно включить в операторы SQL везде, где может использоваться пробел, но не между ключевыми словами EXEC и SQL. Строки отладки языка COBOL и строки перевода страниц (D или / в позиции 7) препроцессор обрабатывает также, как строки комментариев. В операторе SQL INCLUDE DB2 обрабатывает как комментарий любой текст, который следует за точкой после END-EXEC и находится в этой же строке.

Кроме того, если задать опцию препроцессора STDSQL(YES), комментарии SQL можно включить в любой оператор встроенного SQL.

**Продолжения строк операторов SQL:** В операторе SQL, встроенном в программу на языке COBOL, правила продолжения строки при переходе с одной строки на другую для константы типа символьной строки такие же, как и при продолжении строк для нечислового литерала языка COBOL. Однако в качестве первого непустого символа в области В строки продолжения можно использовать кавычку или апостроф. Эти же правила действуют и для продолжения идентификаторов с разделителями, независимо от опции разделителя строк.

В соответствии со стандартом SQL в качестве разделителя для константы типа символьной строки используйте апостроф, а в качестве первого непустого

символа в области В строки продолжения для константы символьной строки используйте кавычку.

**Объявление таблиц и производных таблиц:** Программа на языке COBOL для описания каждой таблицы и производной таблицы, к которым она обращается, должна содержать оператор DECLARE TABLE. Для генерации операторов DECLARE TABLE можно воспользоваться генератором объявлений DB2 (DCLGEN). Компоненты DCLGEN нужно включить в раздел программы DATA DIVISION. Подробную информацию смотрите в разделе “Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN” на стр. 133.

**Динамические операторы SQL в программе на языке COBOL:** Программы на языке COBOL могут обрабатывать динамические операторы SQL. В программах на языке COBOL можно использовать операторы SELECT, если типы данных и число возвращаемых полей фиксированы. При желании использовать операторы SELECT с переменным списком параметров воспользуйтесь SQLDA. Дополнительную информацию о SQLDA смотрите в разделе “Определение областей дескрипторов SQL” на стр. 183.

**Включение кода:** Чтобы включить операторы SQL или объявления переменных хоста языка COBOL из компонента секционированного набора данных, вставьте в исходный текст, куда вы хотите поместить эти операторы, следующий оператор SQL:

EXEC SQL INCLUDE имя-компоненты END-EXEC.

Нельзя использовать вложенные операторы SQL INCLUDE. Для включения операторов SQL или объявлений переменных хоста языка COBOL не пользуйтесь командами языка COBOL, либо воспользуйтесь оператором SQL INCLUDE для включения соответствующего кода препроцессора CICS. Обычно оператор SQL INCLUDE используется только для кодирования, связанного с SQL.

**Поля:** Операторы SQL занимают позиции с 12 по 72. Если начало оператора EXEC SQL находится до позиции 12, препомпилятор DB2 не опознает этот оператор SQL.

Опция препомпилятора MARGINS позволяет установить левое и правое поле с 1 по 80 позицию. Но код оператора EXEC SQL не должен начинаться до позиции 12.

**Имена:** Для переменной хоста можно использовать любое допустимое имя языка COBOL. Не используйте имена внешних записей или имена плана доступа, начинающиеся с 'DSN', и имена переменных хоста, начинающиеся с 'SQL'. Эти имена зарезервированы для DB2.

**Последовательные номера:** В операторы исходного текста, генерируемые препомпилятором DB2, последовательные номера не включаются.

**Метки операторов:** При желании перед выполнимыми операторами SQL в разделе программы PROCEDURE DIVISION можно указать имя параграфа.

**Оператор *WHENEVER*:** В операторе SQL WHENEVER целью для условия GOTO должно являться имя раздела или неспецифицированное имя параграфа в разделе программы PROCEDURE DIVISION.

**Особенности программ на языке COBOL:** При написании программ на языке COBOL надо иметь в виду следующие особенности:

- В программе на языке COBOL, где в качестве имен переменных хоста используются элементы многоуровневой структуры, прекомпилятор DB2 генерирует имена двух самых нижних уровней. Если после этого программа на языке COBOL компилируется OS/VS COBOL, компилятор выдает сообщения IKF3002I и IKF3004I. Если программа компилируется VS COBOL II или более поздними компиляторами, этих сообщений можно избежать.
- Опции DYNAM и NODYNAM языка COBOL используются в зависимости от операционной среды.

#### TSO и IMS

Опцию DYNAM можно задать, если программа на языке COBOL компилируется компиляторами VS COBOL II или COBOL/370™ либо компилятором OS/VS COBOL с библиотеками времени выполнения VS COBOL II или COBOL/370.

IMS и DB2 модуля интерфейса языка совместно используют имя алиаса DSNHLI. При конкатенации библиотек нужно:

- Если с опцией DYNAM языка COBOL используется IMS, убедиться, что первой конкatenируется библиотека IMS.
- Если прикладная программа запускается только под DB2, убедиться, что первой конкatenируется библиотека DB2.

#### CICS и CAF

При компиляции программы на языке COBOL, содержащей операторы SQL нужно задать опцию NODYNAM. Опцию DYNAM использовать нельзя.

Так как в хранимых процедурах используется CAF, хранимые процедуры языка COBOL также нужно компилировать с опцией NODYNAM.

- Чтобы избежать усечения числовых значений, задаются следующие опции компиляторов языка COBOL:
  - TRUNC(OPT) – если известно, что данные, которые будут использоваться программой для каждой двоичной переменной, имеют точность не больше заданной для данной двоичной переменной в условии PICTURE.
  - TRUNC(BIN) – если точность данных, используемых для каждой двоичной переменной, может превысить точность в условии PICTURE.

DB2 назначает значения для двоичных целых переменных хоста языка COBOL также, как при заданной опции компилятора TRUNC(BIN).

- Если программа на языке COBOL содержит несколько точек входа или вызывается несколько раз, условие USING оператора входа, выполняемое перед выполнением первого оператора SQL, должно содержать SQLCA и все записи раздела связей, используемые какими-либо операторами SQL в качестве переменных хоста.
- Оператор REPLACE не влияет на операторы SQL. Он воздействует только на операторы языка COBOL, генерируемые прокомпилятором.
- Не используйте в операторах SQL фигуративные константы языка COBOL (например, ZERO и SPACE), символические знаки, модификацию ссылок и нижние индексы.
- Глава 3 книги *DB2 SQL Reference* содержит правила использования идентификаторов имен SQL.
- Для дефисов применяются следующие правила:
  - Дефисы, используемые в качестве операторов вычитания, обрамляются пробелами. Дефис, не обрамленный пробелами, DB2 обычно интерпретирует как часть имени переменной хоста.
  - Дефис в идентификаторах SQL можно использовать в любой из следующих ситуаций:
    - Прикладная программа – локальная программа, выполняемая на DB2 UDB for OS/390 Версии 6 или более новой.
    - Прикладная программа обращается к удаленным узлам, причем и на локальном, и на удаленных узлах используются DB2 UDB for OS/390 Версии 6 или более новой.
- Если оператор SQL включить в параграф языка COBOL PERFORM ... THRU, а также задать оператор SQL WHENEVER ... GO, компилятор языка COBOL возвращает предупреждающее сообщение IGYOP3094. Это сообщение может указывать на ошибку в зависимости от смысла, который придавался коду. Такой вариант использовать не рекомендуется.
- При использовании VS COBOL II или COBOL/370 с опцией NOCMPR2 действуют следующие дополнительные ограничения:
  - При использовании вложенных программ или пакетной компиляции все операторы SQL и любые переменные хоста, на которые они ссылаются, должны находиться в первой программе.
  - Программы DB2 на языке COBOL должны содержать разделы DATA DIVISION и PROCEDURE DIVISION. Программы, использующие прокомпилятор DB2, должны содержать оба эти раздела, а также раздел WORKING-STORAGE.

Продуктозависимый интерфейс программирования

Если переменные хоста с изменением адресов передаются в программу несколько раз, вызываемая программа должна переустановить SQL-INIT-FLAG. Переустановка этого флага указывает, что оперативная память должна инициализироваться при выполнении следующего оператора SQL. Для переустановки этого флага в раздел программы PROCEDURE DIVISION перед каждым оператором SQL, использующим переменные хоста, вставьте оператор MOVE ZERO TO SQL-INIT-FLAG.

Конец Продуктозависимый интерфейс программирования

## Использование переменных хоста

Все переменные хоста, используемые в операторах SQL, должны быть объявлены явно в разделе программы WORKING-STORAGE SECTION или LINKAGE SECTION. Каждая переменная хоста должна быть объявлена явно перед своим первым использованием в операторе SQL.

Операторам языка COBOL, задающим переменные хоста, должен предшествовать оператор BEGIN DECLARE SECTION, а за ними следовать оператор END DECLARE SECTION. Операторы BEGIN DECLARE SECTION и END DECLARE SECTION необходимо использовать, когда задан параметр препроцессора STDSQL(YES).

В операторе SQL все переменные хоста должны начинаться с двоеточия (:).

В пределах набора данных источника или компонента имена переменных хоста должны быть уникальны, даже если переменные хоста находятся в различных блоках, классах или процедурах. Чтобы имена переменных хоста были уникальными, к ним можно добавить имена структур.

Оператор SQL, в котором используется переменная хоста, должен находиться в пределах области действия оператора, объявляющего эту переменную.

Переменные хоста нельзя задавать как массивы, за исключением переменных-индикаторов. OCCURS можно задавать только при задании структуры индикатора. Для всех других типов переменных хоста задавать OCCURS нельзя.

## Объявление переменных хоста

Для объявления переменных хоста допустимы лишь некоторые объявления языка COBOL. Если объявление переменной недопустимо, результатом выполнения любого оператора SQL со ссылкой на эту переменную может стать сообщение "UNDECLARED HOST VARIABLE" (необъявленная переменная хоста).

**Числовые переменные хоста:** На следующем рисунке показан синтаксис допустимых объявлений числовых переменных хоста.

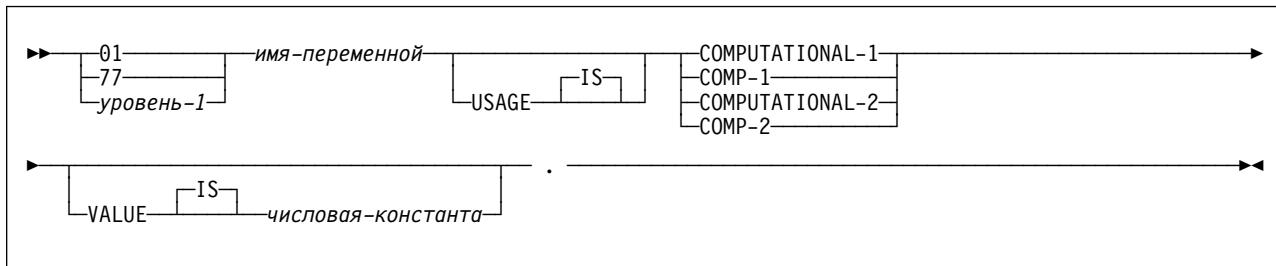


Рисунок 43. Числовые переменные хоста

**Примечания:**

1. уровень-1 указывает уровень языка COBOL от 2 до 48.
2. COMPUTATIONAL-1 и COMP-1 эквивалентны.
3. COMPUTATIONAL-2 и COMP-2 эквивалентны.

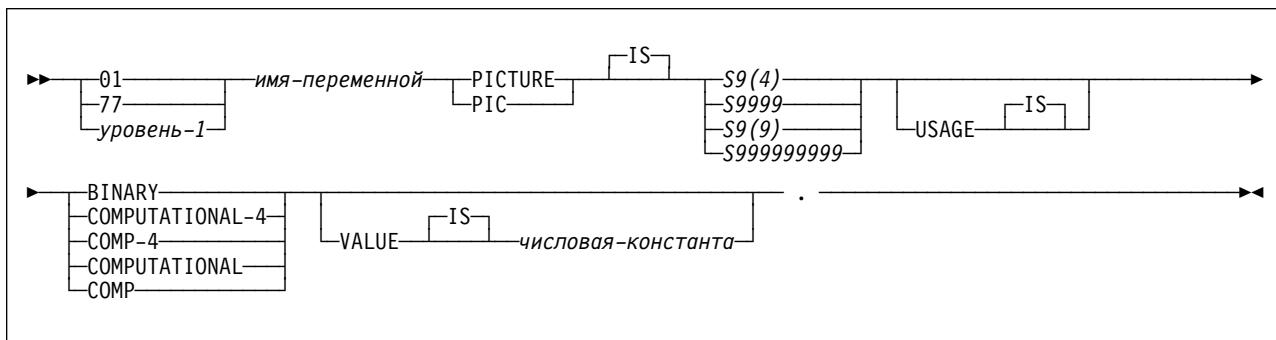


Рисунок 44. Целое и короткое целое

**Примечания:**

1. уровень-1 указывает уровень языка COBOL от 2 до 48.
2. BINARY, COMP, COMPUTATIONAL, COMPUTATIONAL-4 и COMP-4 эквивалентны.
3. Любая спецификация для масштаба игнорируется.

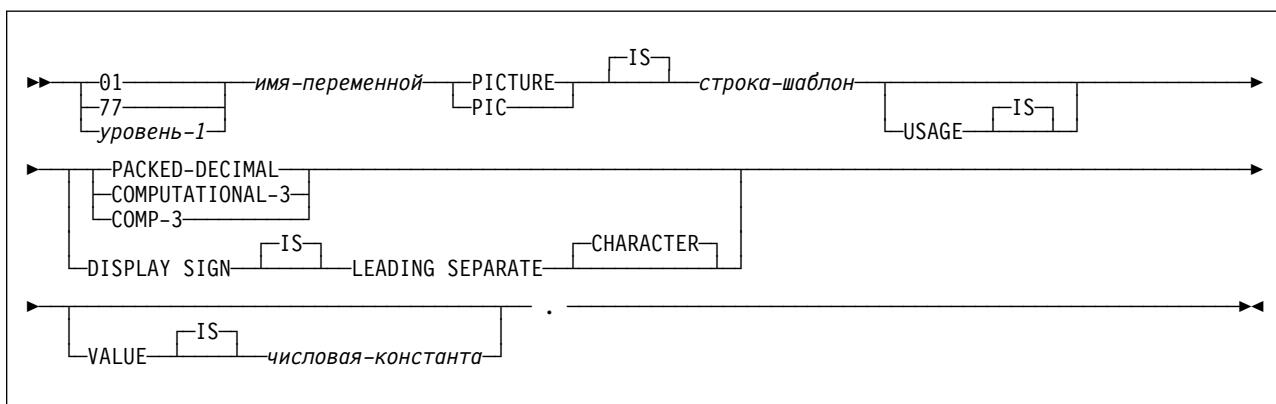


Рисунок 45. Десятичное

**Примечания:**

1. уровень-1 указывает уровень языка COBOL от 2 до 48.

2. PACKED-DECIMAL, COMPUTATIONAL-3 и COMP-3 эквивалентны.

Связанная с этими типами строка-шаблон должна иметь формат S9(i)V9(d) (или S9...9V9...9, с i и d экземплярами 9) или S9(i)V.

3. Связанная с SIGN LEADING SEPARATE строка-шаблон должна иметь формат S9(i)V9(d) (или S9...9V9...9, с i и d экземплярами 9 или S9...9V с i экземплярами 9).

**Символьные переменные хоста:** Для символьных переменных хоста есть три допустимые формы:

- Строки фиксированной длины
- Строки переменной длины
- CLOB (символьные большие объекты)

На следующих рисунках показан синтаксис для всех этих форм, кроме CLOB. Синтаксис для CLOB смотрите на рис. 52 на стр. 193.

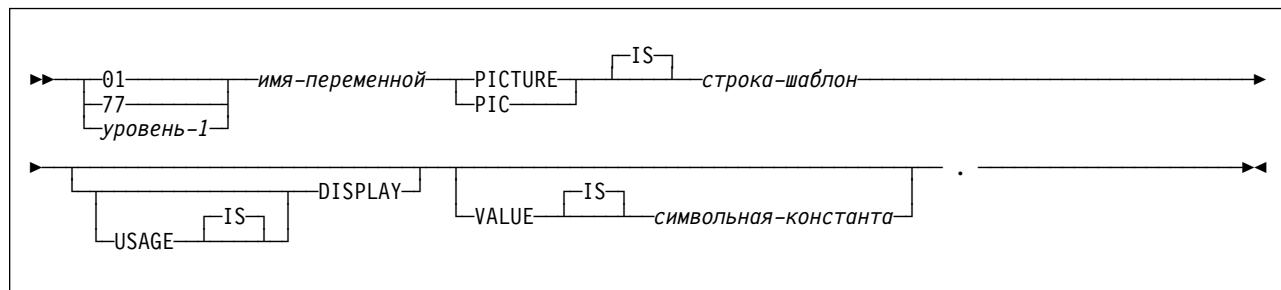


Рисунок 46. Символьные строки фиксированной длины

**Примечание:**

уровень-1 указывает уровень языка COBOL от 2 до 48.

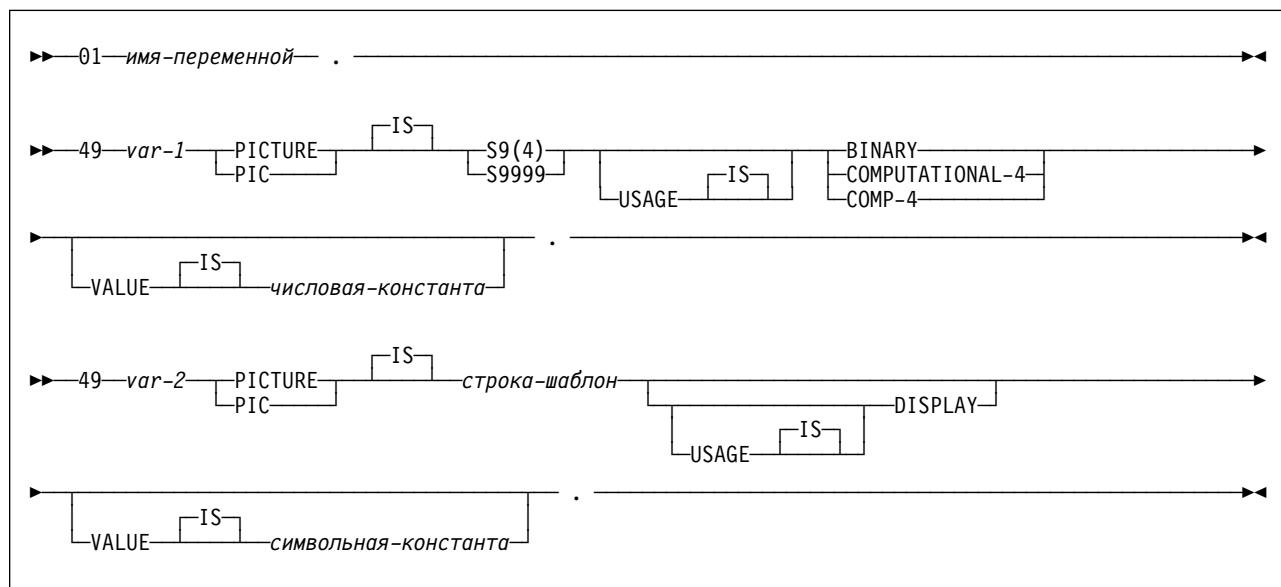


Рисунок 47. Символьные строки переменной длины

**Примечания:**

- Связанная с этими формами строка-шаблон должна быть X(*m*) (или XX...X, с *m* экземплярами X), с  $1 \leq m \leq 255$  для строк фиксированной длины; для других строк *m* не может быть больше максимального размера символьной строки переменной длины.

DB2 использует полную длину переменной S9(4), хотя IBM COBOL для MVS и VM опознает значения только до 9999. Это может привести к ошибкам усечения данных при выполнении операторов языка COBOL и ограничить максимальную длину символьных строк переменной длины до 9999. Чтобы избежать усечения данных, можно использовать опции TRUNC(OPT) или NOTRUNC (что вам больше подходит) компилятора языка COBOL.

- На var-1 и var-2 как на переменные хоста напрямую ссылаться нельзя.

**Символьные переменные графики хоста:** Для символьных переменных графики хоста существует три допустимые формы:

- Строки фиксированной длины
- Строки переменной длины
- DBCLOB (двуухбайтные символьные большие объекты)

На следующих рисунках показан синтаксис для всех этих форм, кроме DBCLOB. Синтаксис для DBCLOB смотрите на рис. 52 на стр. 193.

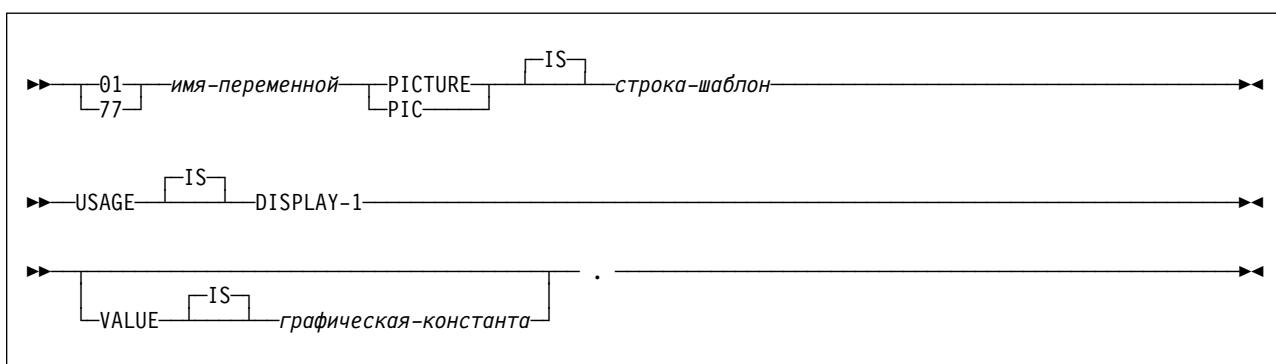


Рисунок 48. Графические строки фиксированной длины

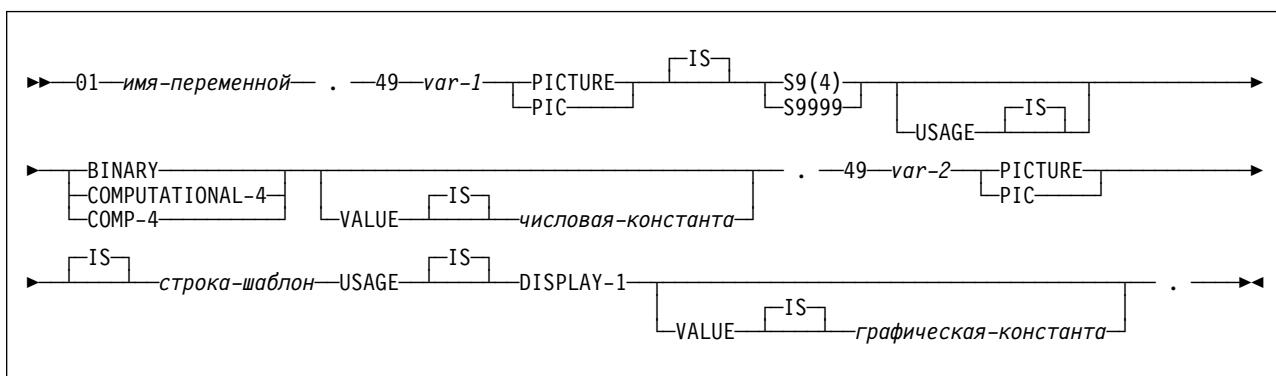


Рисунок 49. Графические строки переменной длины

#### Примечания:

- Связанная с этими формами строка-шаблон должна быть (m) (или GG...G, с *m* экземплярами G), с  $1 \leq m \leq 127$  для строк фиксированной длины.

Для объявлений графических переменных языка COBOL вместо G можно использовать N. Если для объявлений графических переменных используется N, USAGE DISPLAY—1 необязательно. Для всех строк, кроме строк фиксированной длины, t не может быть больше максимального размера графической строки переменной длины.

В DB2 используется полный размер переменной S9(4), хотя некоторые версии языка COBOL и ограничивают максимальную длину графической строки переменной длины до 9999. Это может привести к ошибкам усечения данных при выполнении операторов языка COBOL и ограничить максимальную длину графических строк переменной длины до 9999. Чтобы избежать усечения данных, можно использовать опции TRUNC(OPT) или NOTRUNC компилятора языка COBOL.

2. На var—1 и var—2 как на переменные хоста напрямую ссылаться нельзя.

**Локаторы набора результатов:** На следующем рисунке показан синтаксис объявлений локаторов наборов результатов. Использование этих переменных хоста описано в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

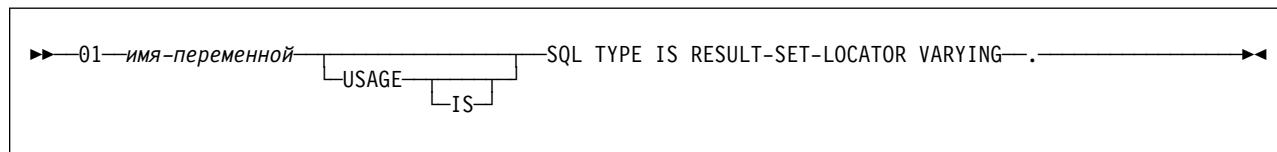


Рисунок 50. Локаторы набора результатов

**Локаторы таблиц:** Ниже показан синтаксис объявлений локаторов таблиц. Использование этих переменных хоста описано в разделе “Доступ к таблицам переходов в пользовательских функциях” на стр. 307.

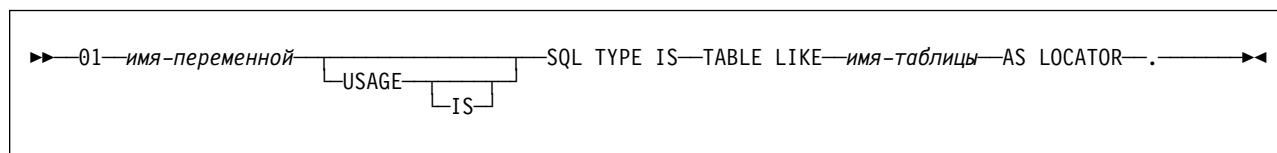


Рисунок 51. Локаторы таблиц

**Переменные и локаторы LOB:** На следующем рисунке показан синтаксис объявлений переменных хоста и локаторов BLOB, CLOB и DBCLOB. Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

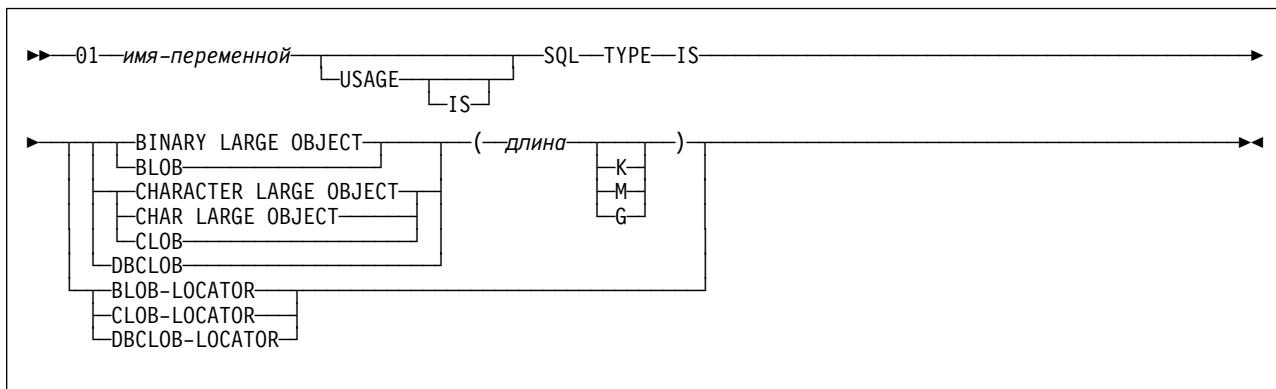


Рисунок 52. Переменные и локаторы LOB

**Переменные ROWID:** На следующем рисунке показан синтаксис объявлений переменных ROWID. Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

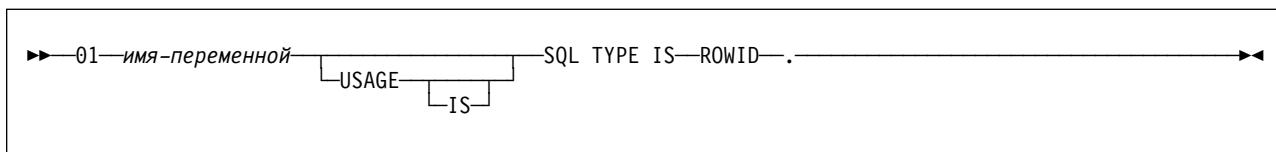


Рисунок 53. Переменные ROWID

## Использование структур хоста

Структура хоста языка COBOL – это именованный набор переменных хоста, заданный в разделе программы WORKING-STORAGE SECTION или LINKAGE SECTION. Структура хоста языка COBOL имеет максимум два уровня, даже если эта структура хоста находится внутри структуры с множеством уровней. Однако можно объявить символьную строку переменной длины, для которой требуется уровень 49.

Имя структуры хоста может быть именем группы, подчиненные уровни которой – элементарные пункты данных. В следующем примере B – имя структуры хоста, которая состоит из элементарных пунктов C1 и C2.

```

01 A
  02 B
    03 C1 PICTURE ...
    03 C2 PICTURE ...
  
```

При использовании в операторе SQL специфицированного имени переменной хоста, например, для идентификации поля внутри структуры, надо указать имя структуры, точку и имя этого поля. Например, вместо C1 OF B или C1 IN B задайте B.C1.

Прекомпилятор не опознает переменные хоста или структуры хоста на любых подчиненных уровнях, которым предшествует один из следующих пунктов:

- Пункт языка COBOL, который должен начинаться в области A
- Любой оператор SQL (кроме SQL INCLUDE)
- Любой оператор SQL в пределах включенного компонента

## COBOL

Когда прекомпилятор встречает в структуре хоста один из этих пунктов, он полагает, что эта структура завершена.

На рис. 54 показан синтаксис допустимых структур хоста.

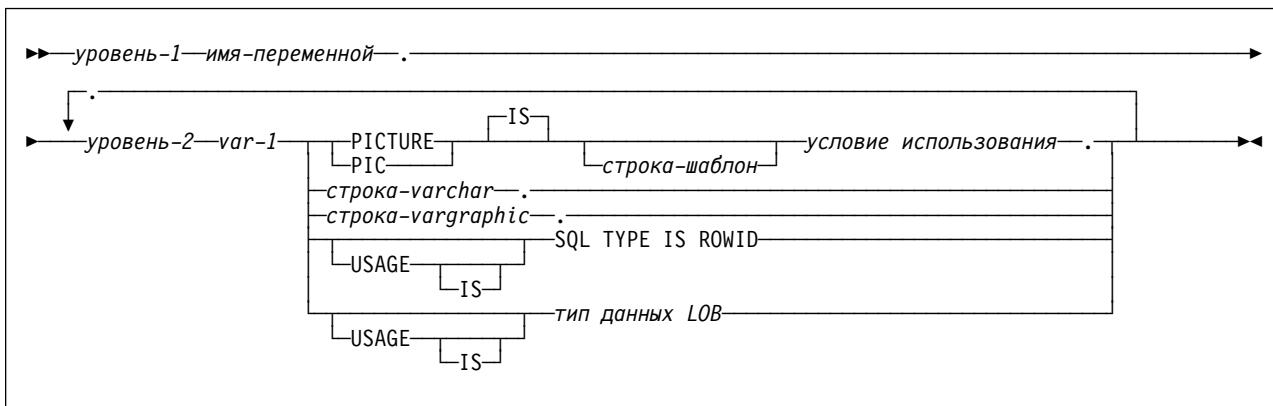


Рисунок 54. Структуры хоста в языке COBOL

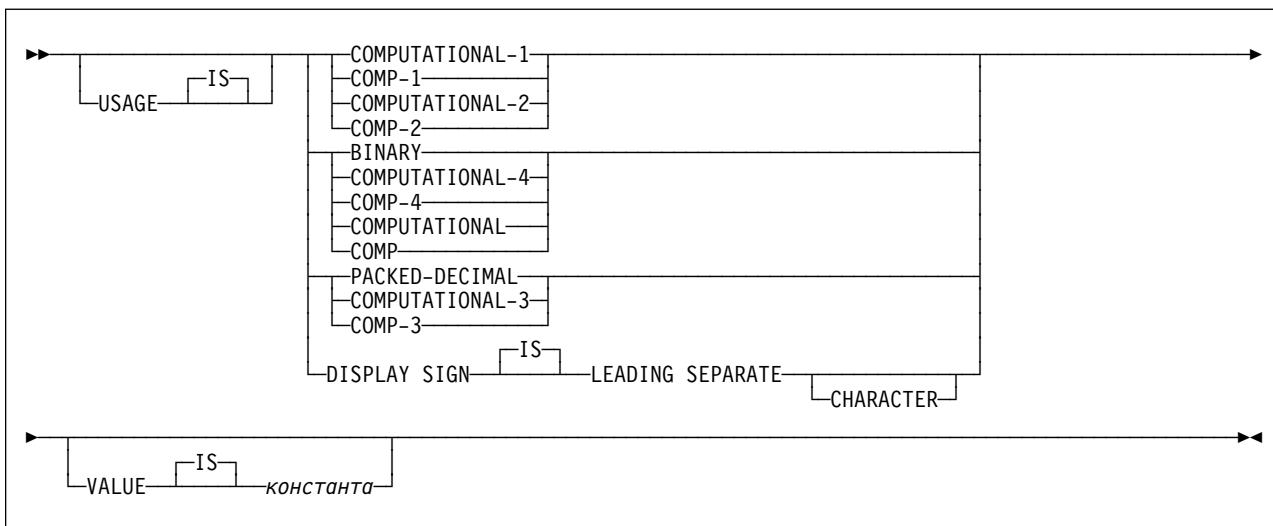


Рисунок 55. Условие использования

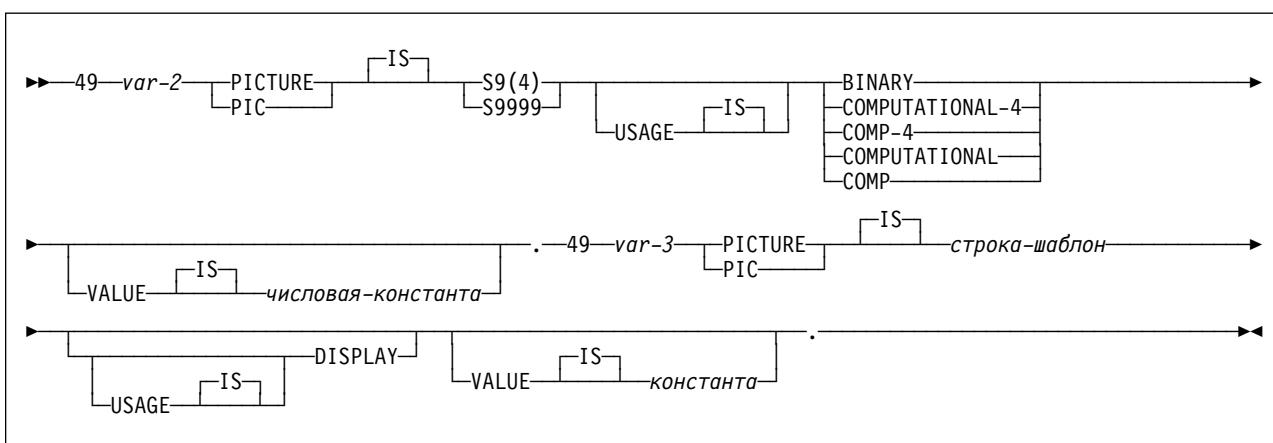


Рисунок 56. Странка-VARCHAR

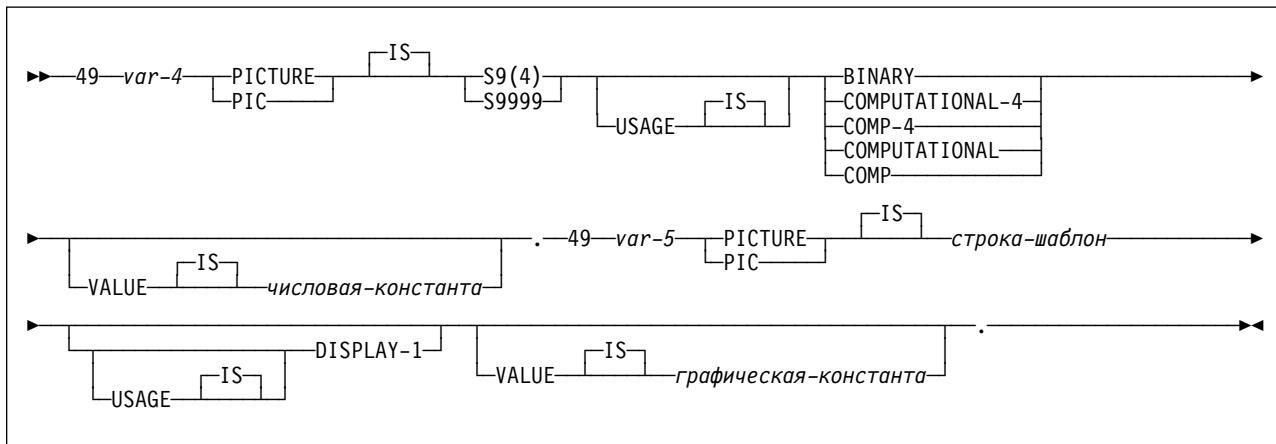


Рисунок 57. Страна—VARGRAPHIC

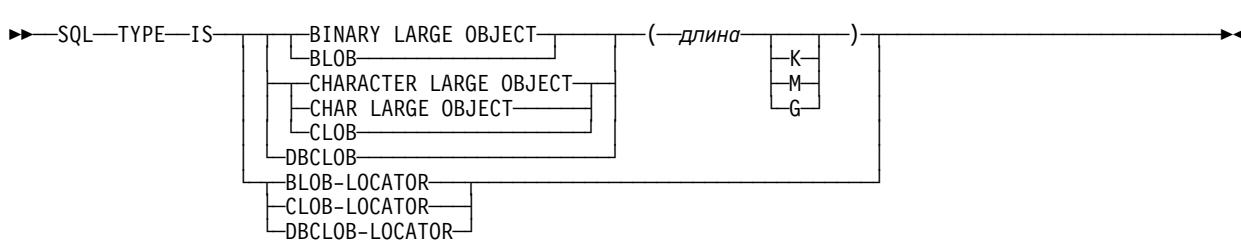


Рисунок 58. Тип данных LOB

**Примечания:**

1. уровень—1 указывает уровень языка COBOL от 1 до 47.
2. уровень—2 указывает уровень языка COBOL от 2 до 48.
3. Для внутренних элементов структуры лучше используйте любой уровень от 02 до 48 (а не 01 или 77), максимально можно задать два уровня.
4. Использование FILLER или необязательного пункта FILLER в пределах объявления структуры хоста может сделать недействительной всю структуру.
5. Строки—шаблоны нельзя использовать для элементов с плавающей точкой, но для других типов данных их использование обязательно.

**Определение эквивалентных типов данных SQL и COBOL**

В Табл. 13 на стр. 196 приводятся типы данных SQL и базовые значения SQLTYPE и SQLLEN, используемые препроцессором для переменных хоста, которые он находит в операторах SQL. Если переменная хоста задается с переменной индикатора, значение SQLTYPE – это базовое значение SQLTYPE плюс 1.

## COBOL

Таблица 13 (Стр. 1 из 2). Типы данных SQL, используемые прекомпилятором в объявлениях языка COBOL

Тип данных языка COBOL	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
COMP–1	480	4	REAL или FLOAT( <i>n</i> ) $1 \leq n \leq 21$
COMP–2	480	8	DOUBLE PRECISION, или FLOAT ( <i>n</i> ) $22 \leq n \leq 53$
S9(⟨i⟩V9(⟨d⟩ COMP–3 или S9(⟨i⟩V9(⟨d⟩ PACKED–DECIMAL	484	<i>i+d</i> в байте 1, <i>d</i> в байте 2	DECIMAL( <i>i+d,d</i> ) или NUMERIC( <i>i+d,d</i> )
S9(⟨i⟩V9(⟨d⟩ DISPLAY SIGN LEADING SEPARATE	504	<i>i+d</i> в байте 1, <i>d</i> в байте 2	Точного эквивалента нет. Используется DECIMAL( <i>i+d,d</i> ) или NUMERIC( <i>i+d,d</i> )
S9(4) COMP–4 или BINARY	500	2	SMALLINT
S9(9) COMP–4 или BINARY	496	4	INTEGER
Символьные данные фиксированной длины	452	<i>m</i>	CHAR( <i>m</i> )
Символьные данные переменной длины $1 \leq m \leq 255$	448	<i>m</i>	VARCHAR( <i>m</i> )
Символьные данные переменной длины <i>m</i> >255	456	<i>m</i>	VARCHAR( <i>m</i> )
Графические данные фиксированной длины	468	<i>m</i>	GRAPHIC( <i>m</i> )
Графические данные переменной длины $1 \leq m \leq 127$	464	<i>m</i>	VARGRAPHIC( <i>m</i> )
Графические данные переменной длины <i>m</i> >127	472	<i>m</i>	VARGRAPHIC( <i>m</i> )
SQL TYPE IS RESULT–SET–LOCATOR	972	4	Локатор набора результатов <sup>1</sup>
SQL TYPE IS TABLE LIKE имя–таблицы AS LOCATOR	976	4	Локатор таблицы <sup>1</sup>
SQL TYPE IS BLOB–LOCATOR	960	4	Локатор BLOB <sup>1</sup>
SQL TYPE IS CLOB–LOCATOR	964	4	Локатор CLOB <sup>1</sup>
USAGE IS SQL TYPE IS DBCLOB–LOCATOR	968	4	Локатор DBCLOB <sup>1</sup>
USAGE IS SQL TYPE IS BLOB( <i>n</i> ) $1 \leq n \leq 2147483647$	404	<i>n</i>	BLOB( <i>n</i> )
USAGE IS SQL TYPE IS CLOB( <i>n</i> ) $1 \leq n \leq 2147483647$	408	<i>n</i>	CLOB( <i>n</i> )
USAGE IS SQL TYPE IS DBCLOB( <i>m</i> ) $1 \leq m \leq 1073741823^2$	412	<i>n</i>	DBCLOB( <i>m</i> ) <sup>2</sup>
SQL TYPE IS ROWID	904	40	ROWID

Таблица 13 (Стр. 2 из 2). Типы данных SQL, используемые прекомпилятором в объявлениях языка COBOL

Тип данных языка COBOL	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
------------------------	--------------------------------	----------------------------	----------------

**Примечания:**

1. Нельзя использовать этот тип данных в качестве типа столбца.
2.  $m$  – число двухбайтных символов.

С помощью Табл. 14 можно задать переменные хоста, которые получают данные из базы данных. Эту таблицу можно использовать для определения типа данных языка COBOL, эквивалентного некоторому типу данных SQL. Например, если надо обработать данные типа TIMESTAMP, можно по таблице определить подходящий тип переменной хоста в программе, принимающей значение данных.

Таблица 14 (Стр. 1 из 2). Типы данных SQL, соответствующие объявляемым типам языка COBOL

Тип данных SQL	Тип данных языка COBOL	Примечания
SMALLINT	S9(4) COMP–4 или BINARY	
INTEGER	S9(9) COMP–4 или BINARY	
DECIMAL( $p,s$ ) или NUMERIC( $p,s$ )	Если $p < 19$ : S9( $p-s$ )V9( $s$ ) COMP–3 или S9( $p-s$ )V9( $s$ ) PACKED–DECIMAL DISPLAY SIGN LEADING SEPARATE	$p$ – точность; $s$ – масштаб. $0 \leq s \leq p \leq 18$ . Если $s=0$ , используйте S9( $p$ )V или S9( $p$ ). Если $s=p$ , используйте SV9( $s$ ). Если $p > 18$ , точного эквивалента нет. Используйте COMP–2.
REAL или FLOAT( $n$ )	COMP–1	$1 \leq n \leq 21$
DOUBLE PRECISION, DOUBLE или FLOAT ( $n$ )	COMP–2	$22 \leq n \leq 53$
CHAR( $n$ )	символьная строка фиксированной длины	$1 \leq n \leq 255$
VARCHAR( $n$ )	символьная строка переменной длины	
GRAPHIC( $n$ )	графическая строка фикс. длины	$n$ указывает число двухбайтных символов, символов, а не число байтов. $1 \leq n \leq 127$
VARGRAPHIC( $n$ )	графическая строка переменной длины	$n$ указывает число двухбайтных символов, символов, а не число байтов.
DATE	символьная строка фикс. длины $n$	Если используется подпрограмма–обработчик даты, $n$ задается этой подпрограммой. Иначе $n$ должно быть не меньше 10.
TIME	символьная строка фикс. длины $n$	Если используется подпрограмма–обработчик времени, $n$ задается этой подпрограммой. Иначе $n$ должно быть не меньше 6; для включения части, соответствующей секундам, $n$ должно быть не меньше 8.

Таблица 14 (Стр. 2 из 2). Типы данных SQL, соответствующие объявляемым типам языка COBOL

Тип данных SQL	Тип данных языка COBOL	Примечания
TIMESTAMP	символьная строка фикс. длины	$n$ должно быть не меньше 19. Для включения части, соответствующей микросекундам, $n$ должно быть равно 26; если $n$ меньше 26, микросекунды усекаются.
Локатор набора результатов	SQL TYPE IS RESULT-SET-LOCATOR	Этот тип данных используется только для приема наборов результатов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор таблиц	SQL TYPE IS TABLE LIKE имя-таблицы AS LOCATOR	Этот тип данных используется только в пользовательской функции или в хранимой процедуре для приема строк таблицы переходов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор BLOB	USAGE IS SQL TYPE IS BLOB-LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов BLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор CLOB	USAGE IS SQL TYPE IS CLOB-LOCATOR	Этот тип данных используется только для обработки данных столбцов CLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор DBCLOB	USAGE IS SQL TYPE IS DBCLOB-LOCATOR	Этот тип данных используется только для обработки данных столбцов DBCLOB. Нельзя использовать этот тип данных в качестве типа столбца.
BLOB( $n$ )	USAGE IS SQL TYPE IS BLOB( $n$ )	$1 \leq n \leq 2147483647$
CLOB( $n$ )	USAGE IS SQL TYPE IS CLOB( $n$ )	$1 \leq n \leq 2147483647$
DBCLOB( $n$ )	USAGE IS SQL TYPE IS DBCLOB( $n$ )	$n$ – число двухбайтных символов. $1 \leq n \leq 1073741823$
ROWID	SQL TYPE IS ROWID	

### Примечания по объявлению и использованию переменных языка COBOL

При объявлении переменных языка COBOL имейте в виду следующее.

**Типы данных SQL, не имеющие эквивалента в языке COBOL:** В языке COBOL нет эквивалента десятичному типу данных точностью больше 18. Для хранения значения такой переменной можно использовать:

- Десятичная переменная с точностью меньше либо равной 18, если это удовлетворяет точности фактических данных. При восстановлении десятичного значения в десятичную переменную с масштабом меньше, чем у столбца источника, мантисса значения может быть усечена.

- Целая переменная или переменная с плавающей точкой, которая преобразует такие значения. При выборе целого теряется мантисса числа. Если десятичное число может превысить максимальное значение целого или при желании сохранить мантиссу, можно воспользоваться числами с плавающей точкой. Число с плавающей точкой является приближением для действительного числа. Поэтому при назначении десятичного числа в переменную с плавающей точкой результат может отличаться от исходного числа.
- Символьные переменные хоста. Для записи в нее десятичных значений используйте функцию CHAR.

**Типы данных специального назначения языка COBOL:** Локаторы поддерживаются как тип данных и в языке COBOL, и в SQL. Локаторы нельзя использовать в качестве типов столбцов. Информация о том, как использовать эти типы данных, приводится в следующих разделах:

**Локатор набора результатов** “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579

**Локатор таблиц** “Доступ к таблицам переходов в пользовательских функциях” на стр. 307

**Локаторы больших объектов** “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253

**Записи описания данных уровня 77:** После записи описания данных уровня 77 может следовать одна или несколько записей REDEFINES. Однако имена из этих записей в операторах SQL использовать нельзя. Записи с именем FILTER игнорируются.

**Типы данных SMALLINT и INTEGER:** В языке COBOL типы данных SMALLINT и INTEGER можно объявлять как число десятичных разрядов. DB2 использует полный размер целых чисел (так же, как при обработке с опциями TRUNC(OPT) или NOTRUNC языка COBOL) и может поместить в переменную хоста значение больше значения, заданного числом разрядов в объявлении языка COBOL. Однако при выполнении операторов языка COBOL это может привести к усечению данных. Убедитесь, что размер чисел в программе не выходит за пределы объявленного числа разрядов.

Для коротких целых, которые могут превышать 9999, используйте S9(5) COMP. Для больших целых, которые могут превышать 999 999 999, используйте S9(10) COMP–3, чтобы получить десятичный тип данных. Если в программе на языке COBOL используются целые, превышающие COBOL PICTURE (шаблон языка COBOL), задайте столбец как десятичное для гарантии соответствия типов данных и их верной обработки.

**Переполнение:** Следует остерегаться переполнения. Например, запись в переменную хоста PICTURE S9(4) значения столбца INTEGER, которое больше 32767 или меньше –32768, приведет к предупреждению о переполнении или к ошибке, в зависимости от того, задана переменная–индикатор или нет.

**Типы данных VARCHAR и VARGRAPHIC:** Если символьные переменные хоста переменной длины принимают значения больше 9999 символов, компилируйте программы, использующие эти переменные, с опцией

TRUNC(BIN). Опция TRUNC(BIN) допускает значения символьной строки длиной до 32767.

**Усечение:** Следует остерегаться усечения. Например, при записи значения столбца 80–символьного CHAR в переменную хоста PICTURE X(70) в восстанавливаемой строке усекаются десять символов справа. При восстановлении значения с плавающей точкой двойной точности или десятичного значения столбца в переменную хоста PIC S9(8) COMP будет удалена вся мантисса этого значения.

Подобным образом при записи числа DB2 DECIMAL в эквивалентную переменную языка COBOL данное число может быть также усечено. Это происходит из–за того, что значение DB2 DECIMAL может иметь 31 разряд, а в десятичном числе языка COBOL может быть только 18 разрядов.

## Определение совместимости типов данных SQL и языка COBOL

Переменные хоста языка COBOL, используемые в операторах SQL, должны быть совместимы по типу со столбцами, с которыми их предполагается использовать:

- Типы числовых данных, совместимые друг с другом: Столбцы SMALLINT, INTEGER, DECIMAL или DOUBLE PRECISION совместимы с переменными хоста языка COBOL PICTURE S9(4), PICTURE S9(9), COMP–3, COMP–1, COMP–4 или COMP–2, BINARY или PACKED–DECIMAL. Столбцы DECIMAL также совместимы с переменными хоста языка COBOL, объявленными как DISPLAY SIGN IS LEADING SEPARATE.
- Типы символьных данных, совместимые друг с другом: Столбцы CHAR, VARCHAR и CLOB совместимы с символьными переменными хоста языка COBOL фиксированной и переменной длины.
- Типы символьных данных частично совместимы с локаторами CLOB. Столбцу CHAR или VARCHAR можно назначить значение локатора CLOB, но значение из столбца CHAR или VARCHAR нельзя назначать переменной хоста типа локатора CLOB.
- Типы графических данных, совместимые друг с другом: Столбцы GRAPHIC, VARGRAPHIC и DBCLOB совместимы с переменными хоста графических строк фиксированной и переменной длины языка COBOL.
- Типы графических данных частично совместимы с локаторами DBCLOB. Значение локатора DBCLOB можно назначить столбцу GRAPHIC или VARGRAPHIC, но значение столбца GRAPHIC VARGRAPHIC нельзя назначить переменной хоста локатора DBCLOB.
- Типы данных даты–времени, совместимые с символьными переменными хоста: Столбцы DATE, TIME и TIMESTAMP совместимы с символьными переменными хоста языка COBOL фиксированной и переменной длины.
- Столбец BLOB совместим только с переменной хоста BLOB.
- Столбцы ROWID совместимы только с переменными хоста ROWID.
- Переменная хоста совместима с особым пользовательским типом, если тип этой переменной хоста совместим с исходным типом этого особого типа. Информация о назначении и сравнении пользовательских типов приводится в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

При необходимости DB2 автоматически преобразует строку фиксированной длины в строку переменной длины или строку переменной длины в строку фиксированной длины.

## Использование переменных–индикаторов

Переменная–индикатор – это двухбайтное целое (PIC S9(4) USAGE BINARY). Если для переменной X задана переменная–индикатор, то, когда DB2 требуется занести в X пустое значение, в переменную–индикатор заносится отрицательное значение и X не обновляется. Программа перед использованием X должна проверить переменную–индикатор. Если переменная–индикатор отрицательна, это значит, что X имеет пустое значение, а любое реально записанное в нее значение можно игнорировать.

Если программа хочет с помощью X назначить столбцу пустое значение, она должна присвоить переменной–индикатору отрицательное значение. Тогда DB2 присвоит столбцу пустое значение и проигнорирует любое значение, содержащееся в X.

Переменные–индикаторы объявляются так же, как и переменные хоста. Объявления двух типов переменных можно смешивать любым подходящим способом. Переменные–индикаторы можно задать как скалярные переменные или как элементы массива, используя условие OCCURS одного уровня. Дополнительную информацию о переменных–индикаторахсмотрите в разделе “Использование индикаторных переменных с переменными хоста” на стр. 113.

### *Пример*

Задан оператор:

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS-CD,
                           :DAY :DAY-IND,
                           :BGN :BGN-IND,
                           :END :END-IND
END-EXEC.
```

Переменные можно объявить так:

```
77 CLS-CD    PIC X(7).
77 DAY       PIC S9(4) BINARY.
77 BGN       PIC X(8).
77 END       PIC X(8).
77 DAY-IND   PIC S9(4) BINARY.
77 BGN-IND   PIC S9(4) BINARY.
77 END-IND   PIC S9(4) BINARY.
```

На следующем рисунке показан синтаксис объявления переменной–индикатора.

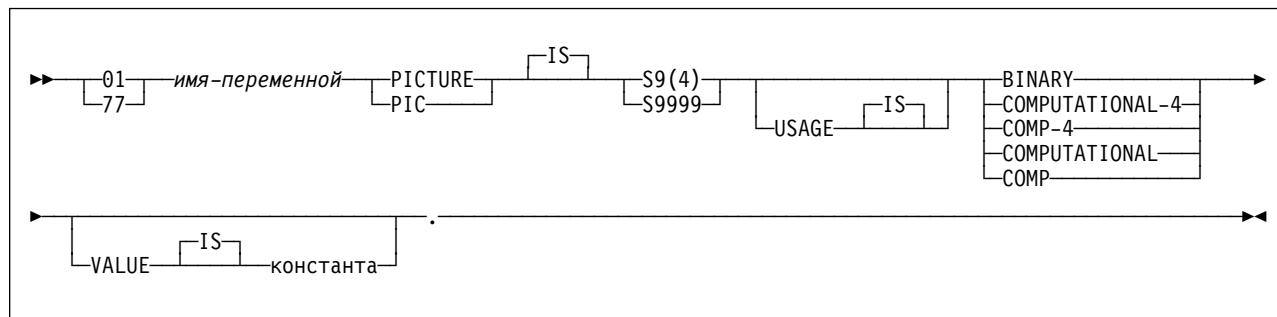


Рисунок 59. Переменная—индикатор

На следующем рисунке показан синтаксис объявления массива индикаторов.

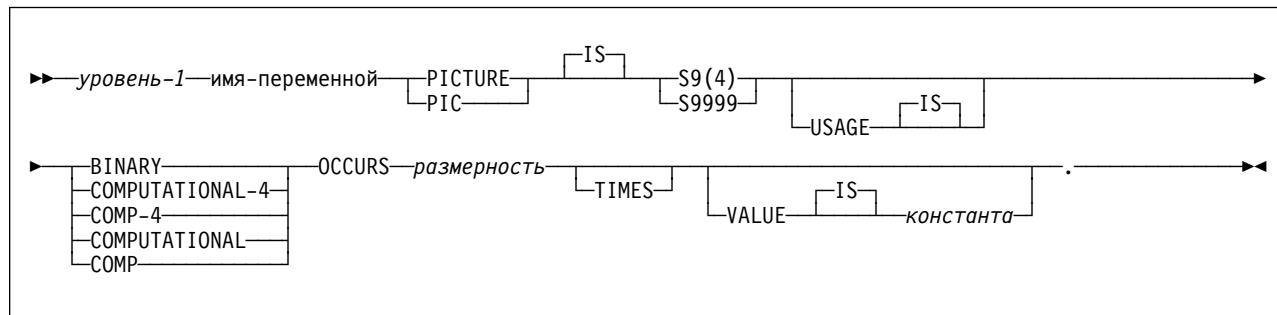


Рисунок 60. Массив индикаторов для структуры хоста

**Примечание:** уровень-1 должен быть целым числом от 2 до 48.

## Обработка кодов возврата ошибок SQL

Для преобразования кода возврата SQL в текстовое сообщение можно использовать подпрограмму DSNTIAR. DSNTIAR берет данные, содержащиеся в SQLCA, форматирует из них сообщение и помещает результат в назначенную прикладной программой область вывода сообщений. Подробная информация о поведении DSNTIAR и связанные с ней понятия приводятся в разделе “Обработка кодов возврата ошибок SQL” на стр. 119.

### Синтаксис DSNTIAR

CALL 'DSNTIAR' USING *sqlca* *сообщение lrec*.

Параметры DSNTIAR означают следующее:

*sqlca*

Область связи SQL.

*сообщение*

Область вывода, в формате VARCHAR, в которую DSNTIAR помещает текст сообщения. В первом полуслове содержится длина оставшейся области; ее минимальное значение – 240.

В эту область помещаются строки вывода текста (каждая строка должна быть длиной, заданной в *lrec*). Например, формат области вывода можно задать так:

```

01  ERROR-MESSAGE.
    02  ERROR-LEN    PIC S9(4)  COMP VALUE +1320.
        02  ERROR-TEXT  PIC X(132) OCCURS 10 TIMES
            INDEXED BY ERROR-INDEX.
    77  ERROR-TEXT-LEN      PIC S9(9)  COMP VALUE +132.
    :
CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.

```

где ERROR-MESSAGE – имя области вывода сообщения с 10 строками длиной 132 каждая, а ERROR-TEXT-LEN – длина каждой строки.

*lrec1*

Полное слово, содержащее длину логической записи сообщений вывода (от 72 до 240).

Пример вызова DSNTIAR из программы находится в примере DB2 программы ассемблера DSN8BC3, в библиотеке DSN8610.SDSNSAMP. Инструкции о доступе к исходному коду этого примера программы и его выводесмотрите в разделе Приложение В, “Примеры прикладных программ” на стр. 883.

### CICS

Если DSNTIAR вызывается динамически из прикладной программы CICS VS COBOL II или CICS COBOL/370, нужно выполнить следующее:

- Скомпилировать программу на языке COBOL с опцией NODYNAM.
- Задать DSNTIAR в CSD.

Если в программе CICS требуется обработка памяти CICS, вместо DSNTIAR нужно использовать подпрограмму DSNTIAC. DSNTIAC имеет следующий синтаксис:

CALL 'DSNTIAC' USING *eib* *commarea* *sqlca* *msg* *lrec1*.

В DSNTIAC есть дополнительные параметры, которые необходимы для вызовов подпрограмм, использующих команды CICS.

*eib* блок интерфейса EXEC

*commarea* область связи

Дополнительную информацию об этих новых параметрахсмотрите в соответствующем руководстве по разработке программ для CICS. Описание остальных параметров аналогично описанию параметров DSNTIAR. И DSNTIAC, и DSNTIAR форматируют SQLDA одним и тем же способом.

DSNTIA1 необходимо определить в CSD. Если загружаются DSNTIAR или DSNTIAC, их также нужно задать в CSD. Пример операторов генерации записей CSD для использования с DSNTIACсмотрите в задании DSNTEJ5A.

Исходный текст DSNTIAC на ассемблере и задание DSNTEJ5A, которое асSEMBЛИРУет и связывает DSNTIAC, находятся в наборе данных префикс.SDSNSAMP.

## Особенности объектно–ориентированных расширений в языке COBOL

При использовании объектно–ориентированных расширений в программе IBM COBOL for MVS & VM нужно знать:

**В каком месте программы разместить операторы SQL:** В IBM COBOL for MVS & VM набор данных источника или компонент могут содержать следующие элементы:

- Несколько программ
- Несколько определений классов, каждый из которых содержит несколько методов

Операторы SQL можно поместить только в первую программу или класс в исходном наборе данных или компоненте. Однако операторы SQL можно поместить в несколько методов в пределах класса. Если прикладная программа содержит несколько наборов данных или компонентов, каждый из них может содержать операторы SQL.

**Где разместить SQLCA, SQLDA и объявления переменных хоста:** SQLCA, SQLDA и объявления переменных хоста SQL можно поместить в раздел WORKING-STORAGE SECTION программы, класса или метода. SQLCA или SQLDA в разделе класса WORKING-STORAGE SECTION являются глобальными для всех методов этого класса. SQLCA или SQLDA в разделе метода WORKING-STORAGE SECTION являются локальными для этого метода.

Если и класс, и метод в пределах этого класса оба содержат SQLCA или SQLDA, метод использует локальную SQLCA или SQLDA.

**Объявление переменных хоста:** Переменные языка COBOL, используемые как переменные хоста, можно объявить в разделах WORKING-STORAGE SECTION или LINKAGE-SECTION программы, класса или метода. Кроме того, переменные хоста можно объявлять в разделе LOCAL-STORAGE SECTION метода. Областью действия переменной хоста является метод, класс или программа, где эта переменная задана.

## Кодирование операторов SQL в программах на языке FORTRAN

В этом разделе рассказывается о способах программирования для кодирования операторов SQL в программах на языке FORTRAN.

### Задание области связи SQL

В программу на языке FORTRAN, содержащую операторы SQL, нужно включить одну или обе из следующих переменных хоста:

- Переменная SQLCOD, объявляется как INTEGER\*4
- Переменная SQLSTA (или SQLSTATE), объявляется как CHARACTER\*5

Либо

- SQLCA, содержащая переменные SQLCOD и SQLSTA.

DB2 задает значения SQLCOD и SQLSTA (или SQLSTATE) после выполнения каждого оператора SQL. Прикладная программа может анализировать

значения этих переменных, чтобы выяснить, успешно ли был выполнен последний оператор SQL. Все операторы SQL в программе должны находиться в области действия объявления переменных SQLCOD и SQLSTA (или SQLSTATE).

Переменные SQLCODE и SQLSTA (или SQLSTATE) задаются, если задана опция препроцессора STDSQL(YES) (в соответствии со стандартом SQL); SQLCA задается, если задана опция препроцессора STDSQL(NO) (в соответствии с правилами DB2).

### **Если задано STDSQL(YES)**

При использовании параметра препроцессора STDSQL(YES) не следует определять SQLCA. Если она будет определена, DB2 проигнорирует SQLCA, и определение SQLCA будет выдавать ошибки при компиляции.

Если объявлена переменная SQLSTA (или SQLSTATE), она не должна быть элементом структуры. Переменные хоста SQLCOD и SQLSTA (или SQLSTATE) должны быть объявлены среди определений программы между операторами BEGIN DECLARE SECTION и END DECLARE SECTION.

### **Если задано STDSQL(NO)**

При использовании параметра препроцессора STDSQL(NO) SQLCA должна быть включена в явном виде. В программе на языке FORTRAN можно закодировать SQLCA либо напрямую, либо с помощью оператора SQL INCLUDE. Для оператора SQL INCLUDE необходимо стандартное объявление SQLCA:

```
EXEC SQL INCLUDE SQLCA
```

Глава 6 книги *DB2 SQL Reference* содержит дополнительную информацию об операторе INCLUDE. Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLCA.

## **Определение областей дескрипторов SQL**

SQLDA необходима для следующих операторов:

- CALL...USING DESCRIPTOR имя–дескриптора
- DESCRIBE имя–оператора INTO имя–дескриптора
- DESCRIBE CURSOR переменная–хоста INTO имя–дескриптора
- DESCRIBE INPUT имя–оператора INTO имя–дескриптора
- DESCRIBE PROCEDURE переменная–хоста INTO имя–дескриптора
- DESCRIBE TABLE переменная–хоста INTO имя–дескриптора
- EXECUTE... USING DESCRIPTOR имя–дескриптора
- FETCH...USING DESCRIPTOR имя–дескриптора
- OPEN...USING DESCRIPTOR имя–дескриптора
- PREPARE...INTO имя–дескриптора

В отличие от SQLCA, в программе может быть несколько SQLDA, и у SQLDA может быть любое допустимое имя. DB2 не поддерживает оператор INCLUDE SQLDA в программах на языке FORTRAN. При его наличии выводится сообщение об ошибке.

Можно заставить программу на языке FORTRAN вызывать подпрограмму (написанную на C, PL/I или языке ассемблера), которая с помощью оператора DB2 INCLUDE SQLDA определяла бы SQLDA, а заодно включала бы

операторы SQL, необходимые для выполнения требуемых функций динамического SQL. Более подробная информация о динамическом SQL приводится в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.

Объявление SQLDA должно находиться до первого оператора SQL, который ссылается на дескриптор данных.

## Вставка операторов SQL

Операторы в исходном тексте на языке FORTRAN должны быть записями фиксированной длины в 80 байтов. Препроцессор DB2 не поддерживает исходных текстов в свободной форме.

Операторы SQL в программе на языке FORTRAN можно вставлять в любое место, где можно поставить выполняемые операторы. Если оператор SQL находится внутри оператора IF, препроцессор сам вставит все необходимые операторы THEN и END IF.

Все операторы SQL в программе на языке FORTRAN должны начинаться с EXEC SQL. Ключевые слова EXEC и SQL должны находиться в одной и той же строке, но остаток оператора может находиться в последующих строках.

Оператор UPDATE можно вставить в программу на языке FORTRAN так:

```
EXEC SQL
C UPDATE DSN8610.DEPT
C SET MGRNO = :MGRNUM
C WHERE DEPTNO = :INTDEPT
```

После оператора SQL в той же строке не может находиться другой оператор SQL или оператор FORTRAN.

Для FORTRAN необязательно, чтобы слова внутри оператора были разделены пробелами, но для языка SQL пробелы необходимы. Правила встроенного SQL аналогичны правилам синтаксиса SQL, для которых необходимо разделять слова одним или несколькими пробелами.

**Комментарии:** В операторах встроенного SQL можно использовать строки комментариев FORTRAN в любом месте, где можно поставить пробел, но не между ключевыми словами EXEC и SQL. Комментарии SQL можно включать в любой оператор встроенного SQL, если задан параметр препроцессора STDSQL(YES).

Препроцессор DB2 не поддерживает в качестве символа начала комментария в программах на языке FORTRAN восклицательный знак (!)

**Перенос оператора SQL на следующую строку:** Правила переноса операторов SQL на следующую строку аналогичны правилам переноса операторов FORTRAN, необходимо только, чтобы EXEC SQL было задано в одной строке. В этом разделе в примерах на SQL как обозначение строк, продолжающих оператор EXEC SQL, используются символы С в шестой позиции.

**Объявление таблиц и производных таблиц:** Для любой таблицы и производной таблицы, к которым обращается программа на языке FORTRAN, в ней должен быть указан оператор DECLARE TABLE.

**Динамический SQL в программе на языке FORTRAN:** В программах на языке FORTRAN можно использовать операторы динамического SQL. Операторы SELECT можно применять, если возвращаемые типы данных и количество полей фиксированы. Если требуется обработка операторов SELECT с произвольным списком параметров, надо воспользоваться SQLDA. Более подробная информация по SQLDA приводится в разделе “Определение областей дескрипторов SQL” на стр. 205.

В операторах PREPARE и EXECUTE IMMEDIATE можно использовать символьные переменные FORTRAN, даже если это переменные фиксированной длины.

**Включение кода:** Чтобы включить операторы SQL или объявления переменных хоста FORTRAN из компонента секционированного набора данных, надо в том месте исходного текста, куда требуется включить операторы, вставить следующий оператор SQL:

EXEC SQL INCLUDE имя-компоненты

Нельзя использовать вложенные операторы SQL INCLUDE. Для включения операторов SQL или объявлений переменных хоста FORTRAN нельзя использовать директиву компилятора FORTRAN INCLUDE.

**Поля:** Операторы SQL могут располагаться между 7 и 72 позиций включительно. Если EXEC SQL начинается до указанной левой границы, препроцессор DB2 не распознает оператор SQL.

**Имена:** Переменной хоста можно дать любое допустимое имя языка FORTRAN. Нельзя использовать имена внешних записей, начинающиеся с 'DSN', и имена переменных хоста, начинающиеся с 'SQL'. Эти имена зарезервированы для DB2.

Нельзя использовать имя DEBUG, кроме как при определении пакета FORTRAN DEBUG. Переменные нельзя называть словами FUNCTION, IMPLICIT, PROGRAM и SUBROUTINE.

**Последовательные номера:** В операторы исходного текста, которые генерирует препроцессор DB2, не включаются последовательные номера.

**Метки операторов:** В колонках от 1 до 5 можно указывать номера для операторов SQL. Но при подготовке программы помеченный оператор SQL генерирует оператор FORTRAN CONTINUE с этой меткой перед выполняемым кодом оператора SQL. Следовательно, помеченный оператор SQL не может быть последним оператором в цикле DO. Кроме того, нельзя ставить метки перед операторами SQL (такими, как INCLUDE и BEGIN DECLARE SECTION), которые стоят до первого выполняемого оператора SQL, так как может произойти ошибка.

**Оператор WHENEVER:** Операнд условия GOTO в операторе SQL WHENEVER должен являться меткой в исходном тексте на языке FORTRAN, находящейся

перед оператором в этой же подпрограмме. Оператор WHENEVER относится только к операторам SQL в этой же подпрограмме.

**Особенности FORTRAN:** При написании программ на языке FORTRAN надо учитывать следующие особенности:

- В исходном тексте нельзя использовать оператор @PROCESS. Вместо этого нужно указывать параметры компилятора в поле PARM.
- Нельзя использовать оператор SQL INCLUDE для включения следующих операторов: PROGRAM, SUBROUTINE, BLOCK, FUNCTION или IMPLICIT.

DB2 поддерживает Версию 3 Выпуск 1 VS FORTRAN со следующими ограничениями:

- Параллелизм не поддерживается. В прикладных программах, содержащих операторы SQL, не должен использоваться параллелизм FORTRAN.
- Во встроенном SQL нельзя использовать байтовый тип данных, так как он не распознается как тип данных хоста.

## Использование переменных хоста

Все переменные хоста, используемые в операторах SQL, должны быть явно объявлены. Неявно объявлять переменные хоста с типизацией по умолчанию или с помощью оператора IMPLICIT нельзя. Любая переменная хоста должна быть явно объявлена до ее первого использования в операторе SQL.

Операторы FORTRAN, определяющие переменные хоста, можно заключать между операторами BEGIN DECLARE SECTION и END DECLARE SECTION. Операторы BEGIN DECLARE SECTION и END DECLARE SECTION необходимо использовать, когда задан параметр препроцессора STDSQL(YES).

В операторе SQL все переменные хоста должны начинаться с двоеточия (:).

Имена переменных хоста должны быть уникальными в пределах программы, даже если переменные хоста находятся в разных блоках, функциях или подпрограммах.

При объявлении символьной переменной хоста нельзя как-либо указывать длину символьной переменной. Можно использовать символьную переменную хоста с неопределенной длиной (например, CHARACTER \*(\*)). Длина всех таких переменных определяется при выполнении соответствующего оператора SQL.

Оператор SQL, использующий какую-либо переменную хоста, должен находиться в области действия оператора, объявившего переменную.

Переменные хоста должны быть скалярными переменными; они не могут быть элементами векторов или массивов (индексированными переменными).

Вызывать подпрограммы, которые могут изменить атрибуты переменной хоста, следует с осторожностью. Такое изменение может создать ошибку при выполнении программы. Приложение С книги *DB2 SQL Reference* содержит более подробную информацию об этом.

## **Объявление переменных хоста**

Не все правильные объявления FORTRAN допустимы как объявления переменных хоста. Если объявление какой-либо переменной неправильно, любой оператор SQL, в котором она упомянута, может вызвать сообщение "UNDECLARED HOST VARIABLE" (Необъявленная переменная хоста).

**Числовые переменные хоста:** Ниже показан синтаксис правильных объявлений числовых переменных хоста.

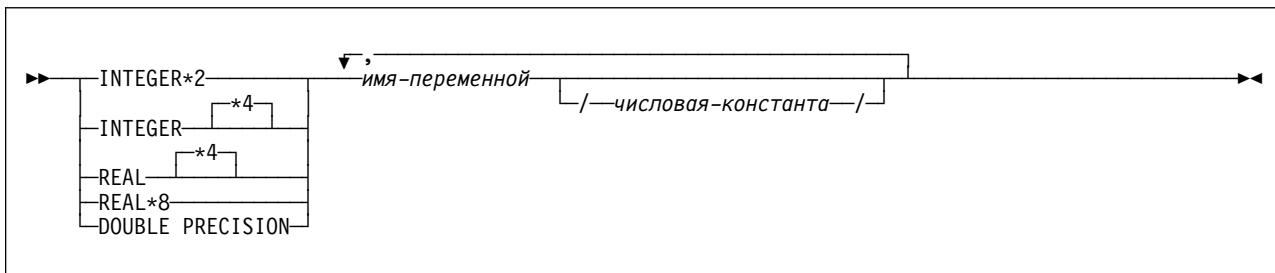
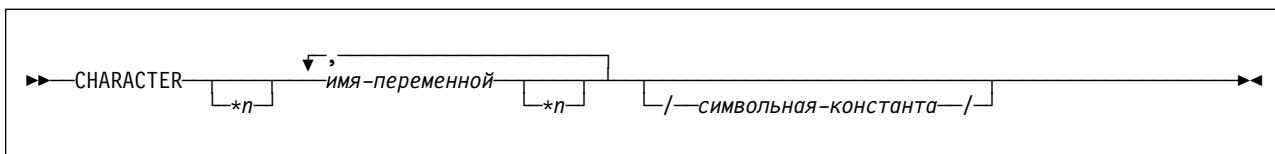


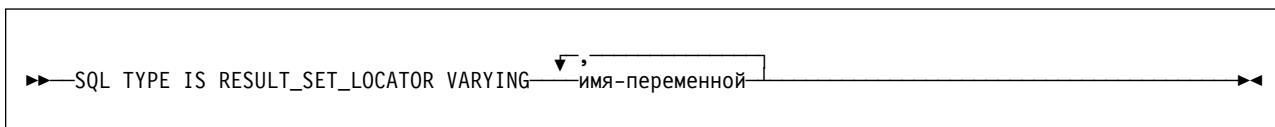
Рисунок 61. Числовые переменные хоста

**Символьные переменные хоста:** Ниже показан синтаксис правильных определений символьных переменных хоста, кроме CLOB. Синтаксис CLOB показан на рис. 64 на стр. 210.



*Рисунок 62. Символьные переменные хоста*

**Локаторы набора результатов:** Ниже показан синтаксис объявлений локаторов набора результатов. Использование этих переменных хоста описано в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.



*Рисунок 63. Локаторы набора результатов*

**Переменные и локаторы больших объектов:** Ниже показан синтаксис объявлений переменных хоста и локаторов BLOB и CLOB. Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

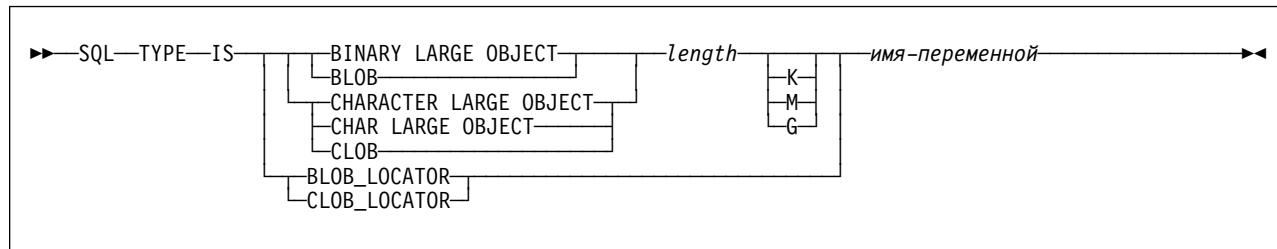


Рисунок 64. Переменные и локаторы LOB

**Переменные ROWID:** Ниже проиллюстрирован синтаксис определений переменных ROWID. Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

```
►►SQL TYPE IS—ROWID—имя-переменной►►
```

Рисунок 65. Переменные ROWID

## Определение эквивалентности типов данных SQL и FORTRAN

В Табл. 15 описан тип данных SQL и базовые значения SQLTYPE и SQLLEN, которые использует препроцессор для переменных хоста, найденных им в операторах SQL. Если переменная хоста задается с переменной-индикатором, SQLTYPE равен базовому SQLTYPE плюс 1.

Таблица 15 (Стр. 1 из 2). Типы данных SQL, которые использует препроцессор в объявлениях FORTRAN

Тип данных FORTRAN	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
INTEGER*2	500	2	SMALLINT
INTEGER*4	496	4	INTEGER
REAL*4	480	4	FLOAT (одинарной точности)
REAL*8	480	8	FLOAT (двойной точности)
CHARACTER*n	452	n	CHAR(n)
SQL TYPE IS RESULT_SET_LOCATOR	972	4	Локатор набора результатов. Этот тип данных нельзя использовать в качестве типа столбца.
SQL TYPE IS BLOB_LOCATOR	960	4	Локатор BLOB. Этот тип данных нельзя использовать в качестве типа столбца.
SQL TYPE IS CLOB_LOCATOR	964	4	Локатор CLOB. Этот тип данных нельзя использовать в качестве типа столбца.
SQL TYPE IS BLOB(n) 1≤n≤2147483647	404	n	BLOB(n)

Таблица 15 (Стр. 2 из 2). Типы данных SQL, которые использует препроцессор в объявлениях FORTRAN

Тип данных FORTRAN	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
SQL TYPE IS CLOB( <i>n</i> ) $1 \leq n \leq 2147483647$	408	<i>n</i>	CLOB( <i>n</i> )
SQL TYPE IS ROWID	904	40	ROWID

В Табл. 16 указано, как определять переменные хоста, которые принимают данные из базы данных. Этой таблицей можно воспользоваться, чтобы найти тип данных языка FORTRAN, эквивалентный какому-либо конкретному типу данных SQL. Например, если надо обработать данные типа TIMESTAMP, можно по таблице определить подходящий тип переменной хоста в программе, принимающей значение данных.

Таблица 16 (Стр. 1 из 2). Соответствия типов данных SQL объявлениям типов FORTRAN

Тип данных SQL	Эквивалент FORTRAN	Примечания
SMALLINT	INTEGER*2	
INTEGER	INTEGER*4	
DECIMAL( <i>p,s</i> ) или NUMERIC( <i>p,s</i> )	точного эквивалента нет	Используйте REAL*8
FLOAT( <i>n</i> ) одинарной точности	REAL*4	$1 \leq n \leq 21$
FLOAT( <i>n</i> ) двойной точности	REAL*8	$22 \leq n \leq 53$
CHAR( <i>n</i> )	CHARACTER*n	$1 \leq n \leq 255$
VARCHAR( <i>n</i> )	точного эквивалента нет	Используйте символьную переменную хоста, достаточно большую, чтобы в нее поместились наибольшее ожидаемое значение VARCHAR.
GRAPHIC( <i>n</i> )	не поддерживается	
VARGRAPHIC( <i>n</i> )	не поддерживается	
DATE	CHARACTER*n	Если используется подпрограмма-обработчик даты, <i>n</i> определяется этой подпрограммой; в противном случае <i>n</i> должно быть не меньше 10.
TIME	CHARACTER*n	Если используется подпрограмма-обработчик времени, <i>n</i> определяется этой подпрограммой; в противном случае <i>n</i> должно быть не меньше 6; чтобы включались секунды, <i>n</i> должно быть не меньше 8.
TIMESTAMP	CHARACTER*n	<i>n</i> должно быть не меньше 19. Чтобы включать микросекунды, <i>n</i> должно быть 26; если <i>n</i> меньше 26, часть, содержащая микросекунды, будет усечена.

Таблица 16 (Стр. 2 из 2). Соответствия типов данных SQL объявлениюм типов FORTRAN

Тип данных SQL	Эквивалент FORTRAN	Примечания
Локатор набора результатов	SQL TYPE IS RESULT_SET_LOCATOR	Этот тип данных можно использовать только для получения наборов результатов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор BLOB	SQL TYPE IS BLOB_LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов BLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор CLOB	SQL TYPE IS CLOB_LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов CLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор DBCLOB	не поддерживается	
BLOB( <i>n</i> )	SQL TYPE IS BLOB( <i>n</i> )	1≤ <i>n</i> ≤2147483647
CLOB( <i>n</i> )	SQL TYPE IS CLOB( <i>n</i> )	1≤ <i>n</i> ≤2147483647
DBCLOB( <i>n</i> )	не поддерживается	
ROWID	SQL TYPE IS ROWID	

### Замечания по объявлению и использованию переменных FORTRAN

При объявлении переменных FORTRAN необходимо иметь в виду следующие особенности.

**Типы данных FORTRAN, не имеющие эквивалента в SQL:** FORTRAN поддерживает некоторые типы данных, не имеющие эквивалента в SQL (например, REAL\*16 и COMPLEX). В большинстве случаев для преобразования данных неподдерживаемых типов в данные типов, поддерживаемых SQL, можно использовать операторы FORTRAN.

**Типы данных SQL, не имеющие эквивалента в FORTRAN:** В FORTRAN не предусмотрено эквивалента для десятичного типа данных (DECIMAL). Для хранения значения такой переменной можно использовать:

- Целочисленные переменные или переменные с плавающей точкой с преобразованием значений. Но при использовании целочисленных переменных будет теряться дробная часть числа. Если десятичное число может превышать максимальное значение целочисленной переменной или если вы хотите сохранить дробное значение, можно воспользоваться числами с плавающей точкой. Числа с плавающей точкой являются приближенными значениями действительных чисел. При присваивании десятичного значения переменной с плавающей точкой результат может отличаться от исходного числа.
- Символьные переменные хоста. Чтобы записать в такую переменную десятичное значение, надо воспользоваться функцией CHAR.

**Типы данных FORTRAN специального назначения:** Локаторы как типы данных поддерживаются и в языке FORTRAN, и в SQL. Локаторы нельзя

использовать в качестве типов столбцов. Информация о том, как использовать эти типы данных, приводится в следующих разделах:

**Локатор набора результатов** “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579

**Локаторы больших объектов** “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253

**Переполнение:** Следует остерегаться переполнения. Например, если при получении значения столбца INTEGER в переменную хоста INTEGER\*2 значение столбца больше, чем 32767 или меньше, чем –32768, будет выдано предупреждение о переполнении или ошибка, в зависимости от того, была ли задана переменная–индикатор.

**Усечение:** Следует остерегаться усечения. Например, при записи 80–символьного значения столбца CHAR в переменную хоста CHARACTER\*70 последние десять символов полученной строки будут усечены.

При получении значения столбца с плавающей точкой или десятичного значения в переменную хоста INTEGER\*4 вся дробная часть будет удалена.

### Замечания по различиям в синтаксисе для констант

Следует иметь в виду следующие различия в синтаксисе для констант.

**Действительные константы:** FORTRAN воспринимает последовательность цифр с десятичной точкой как действительную константу. Оператор SQL воспринимает такую строку как десятичную константу. Таким образом, при указании в операторе SQL действительной константы (то есть константы с плавающей точкой) необходимо использовать экспоненциальную запись.

**Индикаторы экспоненты:** В FORTRAN действительная константа (константа с плавающей точкой) длиной в восемь байтов использует в качестве индикатора экспоненты символ D (например, 3.14159D+04). 8–байтная константа с плавающей точкой в операторе SQL должна использовать символ E (например, 3.14159E+04).

## Определение совместимости типов данных SQL и FORTRAN

Переменные хоста должны быть совместимы по типу со значениями столбцов, для которых они должны использоваться.

- Числовые типы данных совместимы друг с другом. Например, если у значения столбца тип INTEGER, переменная хоста должна быть объявлена как INTEGER\*2, INTEGER\*4, REAL, REAL\*4, REAL\*8 или DOUBLE PRECISION.
- Символьные типы данных совместимы друг с другом. Столбцы CHAR, VARCHAR или CLOB совместимы с символьными переменными хоста FORTRAN.
- Символьные типы данных частично совместимы с локаторами CLOB. Столбцу CHAR или VARCHAR можно назначить значение локатора CLOB, но значение из столбца CHAR или VARCHAR нельзя назначать переменной хоста типа локатора CLOB.

- Типы данных даты и времени совместимы с символьными переменными хоста. Столбцы DATE, TIME или TIMESTAMP совместимы с символьными переменными хоста FORTRAN.
- Столбцы BLOB совместимы только с переменными хоста BLOB.
- Столбцы ROWID совместимы только с переменными хоста ROWID.
- Переменная хоста совместима с особым пользовательским типом, если тип переменной хоста совместим с исходным типом этого особого пользовательского типа. Информация о назначении и сравнении пользовательских типов приводится в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

## Использование переменных–индикаторов

Переменная–индикаторная – это двухбайтное целое число (INTEGER\*2). Если для переменной X задана переменная–индикатор, то, когда DB2 требуется занести в X пустое значение, в переменную–индикатор заносится отрицательное значение и X не обновляется. Программа перед использованием X должна проверить переменную–индикатор. Если переменная–индикатор отрицательна, это значит, что X имеет пустое значение, а любое реально записанное в нее значение можно игнорировать.

Если программа хочет с помощью X назначить столбцу пустое значение, она должна присвоить переменной–индикатору отрицательное значение. Тогда DB2 присвоит столбцу пустое значение и проигнорирует любое значение, содержащееся в X.

Переменные–индикаторы объявляются так же, как и переменные хоста. Объявления этих двух типов переменных можно смешивать любым удобным способом. Более подробная информация о переменных–индикаторах приводится в разделе “Использование индикаторных переменных с переменными хоста” на стр. 113.

**Пример:** Возьмем оператор:

```
EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,
C :DAY :DAYIND,
C :BGN :BGNIND,
C :END :ENDIND
```

Переменные можно объявить следующим образом:

```
CHARACTER*7 CLSCD
INTEGER*2 DAY
CHARACTER*8 BGN, END
INTEGER*2 DAYIND, BGNIND, ENDIND
```

Ниже показан синтаксис переменной–индикатора.

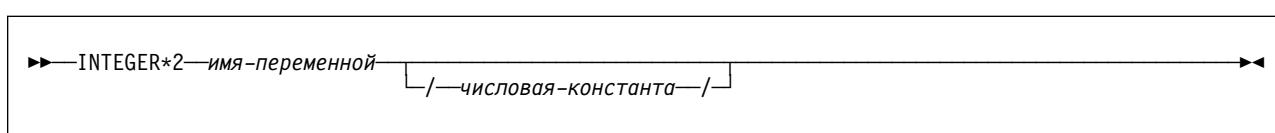


Рисунок 66. Переменная–индикатор

## Обработка кодов возврата ошибок SQL

Чтобы конвертировать код возврата SQL в текстовое сообщение, можно воспользоваться подпрограммой DSNTIR. DSNTIR строит список параметров и вызывает DSNTIAR. DSNTIAR берет данные, содержащиеся в SQLCA, форматирует из них сообщение и помещает результат в назначенную прикладной программой область вывода сообщений. Подробная информация о поведении DSNTIAR и связанные с ней понятия приводятся в разделе “Обработка кодов возврата ошибок SQL” на стр. 119.

### Синтаксис DSNTIR

```
CALL DSNTIR ( длина–ошибки, сообщение, код–возврата )
```

Параметры DSNTIR имеют следующее значение:

*длина–ошибки*

Общая длина области вывода сообщений.

*сообщение*

Область вывода, в формате VARCHAR, в которую DSNTIAR помещает текст сообщения. В первом полуслове содержится длина оставшейся области; ее минимальное значение – 240.

В эту область помещаются выходные строки текста сообщения.

Например, формат области вывода можно задать так:

```
INTEGER ERRLEN /1320/
CHARACTER*132 ERRTXT(10)
INTEGER ICODE
:
CALL DSNTIR ( ERRLEN, ERRTXT, ICODE )
```

где ERRLEN содержит общую длину области вывода сообщений, ERRTXT – имя области вывода сообщений, а ICODE – код возврата.

*код–возврата*

Получает код возврата из DSNTIAR.

Пример вызова DSNTIR (которая затем вызывает DSNTIAR) из прикладной программы приводится в примере программы на ассемблере DB2 DSN8BF3, содержащемся в библиотеке DSN8610.SDSNSAMP. Инструкции по тому, как получить и распечатать исходный текст примера программы, приводятся в разделе Приложение B, “Примеры прикладных программ” на стр. 883.

## Кодирование операторов SQL в программах на языке PL/I

В этом разделе рассказывается о способах программирования для кодирования операторов SQL в программах на языке PL/I.

## Задание области связи SQL

В программе на языке PL/I, содержащей операторы SQL, должны использоваться одна или обе из следующих переменных хоста:

- Переменная SQLCODE, объявляется как BIN FIXED (31)
- Переменная SQLSTATE, объявляется как CHARACTER(5)

Либо

- SQLCA, содержащая переменные SQLCOD и SQLSTATE.

DB2 задает значения SQLCOD и SQLSTATE после выполнения каждого оператора SQL. Прикладная программа может анализировать значения этих переменных, чтобы выяснить, успешно ли был выполнен последний оператор SQL. Все операторы SQL в программе должны находиться в области действия объявления переменных SQLCOD и SQLSTATE.

Переменные SQLCODE и SQLSTA (или SQLSTATE) задаются, если задана опция препроцессора STDSQL(YES) (в соответствии со стандартом SQL); SQLCA задается, если задана опция препроцессора STDSQL(NO) (в соответствии с правилами DB2).

### Если задано STDSQL(YES)

При использовании параметра препроцессора STDSQL(YES) не следует определять SQLCA. Если она будет определена, DB2 проигнорирует SQLCA, и определение SQLCA будет выдавать ошибки при компиляции.

Если объявлена переменная SQLSTATE, она не должна быть элементом структуры. Переменные хоста SQLCOD и SQLSTATE должны быть объявлены среди определений программы между операторами BEGIN DECLARE SECTION и END DECLARE SECTION.

### Если задано STDSQL(NO)

При использовании параметра препроцессора STDSQL(NO) SQLCA должна быть включена в явном виде. В программе на языке PL/I можно закодировать SQLCA либо напрямую, либо с помощью оператора SQL INCLUDE. Для оператора SQL INCLUDE необходимо стандартное объявление SQLCA:

```
EXEC SQL INCLUDE SQLCA;
```

Глава 6 книги *DB2 SQL Reference* содержит дополнительную информацию об операторе INCLUDE. Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLCA.

## Определение областей дескрипторов SQL

SQLDA необходима для следующих операторов:

- CALL...USING DESCRIPTOR имя–дескриптора
- DESCRIBE имя–оператора INTO имя–дескриптора
- DESCRIBE CURSOR переменная–хоста INTO имя–дескриптора
- DESCRIBE INPUT имя–оператора INTO имя–дескриптора
- DESCRIBE PROCEDURE переменная–хоста INTO имя–дескриптора
- DESCRIBE TABLE переменная–хоста INTO имя–дескриптора
- EXECUTE...USING DESCRIPTOR имя–дескриптора

- FETCH...USING DESCRIPTOR имя—дескриптора
- OPEN...USING DESCRIPTOR имя—дескриптора
- PREPARE...INTO имя—дескриптора

В отличие от SQLCA, в программе может быть несколько SQLDA, и у SQLDA может быть любое допустимое имя. В программе на языке PL/I можно закодировать SQLDA либо напрямую, либо с помощью оператора SQL INCLUDE. Для оператора SQL INCLUDE необходимо стандартное объявление SQLCA:

```
EXEC SQL INCLUDE SQLDA;
```

Если не используется параметр препроцессора TWOPASS, объявление SQLDA должно находиться до первого оператора SQL, который ссылается на дескриптор данных. Глава 6 книги *DB2 SQL Reference* содержит дополнительную информацию об операторе INCLUDE, Приложение С книги *DB2 SQL Reference* подробно описывает поля SQLDA.

## Вставка операторов SQL

Первым оператором программы на языке PL/I должен быть оператор PROCEDURE с OPTIONS(MAIN), если только программа не является хранимой процедурой. Хранимую процедуру можно вызывать в качестве подпрограммы. Более подробная информация приводится в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

Операторы SQL в программе на языке PL/I можно вставлять в любое место, где можно поставить выполняемые операторы.

Все операторы SQL в программе на языке PL/I должны начинаться с EXEC SQL и заканчиваться точкой с запятой (;). Ключевые слова EXEC и SQL должны находиться в одной и той же строке, но остальная часть оператора может занимать несколько строк.

Оператор UPDATE можно вставить в программу на языке PL/I так:

```
EXEC SQL UPDATE DSN8610.DEPT
      SET MGRNO = :MGR_NUM
      WHERE DEPTNO = :INT_DEPT ;
```

**Комментарии:** В операторах встроенного SQL можно использовать комментарии PL/I в любом месте, где можно поставить пробел, но не между ключевыми словами EXEC и SQL. Комментарии SQL можно включать в любой оператор встроенного SQL, если задан параметр препроцессора STDSQL(YES).

Чтобы использовать в комментариях символы DBCS, необходимо ограничивать их символами переключения shift-out и shift-in; первый символ shift-in в строке DBCS означает конец строки DBCS.

**Перенос оператора SQL на следующую строку:** Правила переноса операторов SQL на следующую строку аналогичны правилам переноса операторов PL/I, необходимо только, чтобы EXEC SQL было задано в одной строке.

**Объявление таблиц и производных таблиц:** Для любой таблицы или производной таблицы, к которым обращается программа, в ней должен быть задан оператор DECLARE TABLE. Для генерации операторов DECLARE TABLE можно воспользоваться генератором объявлений DB2 (DCLGEN). Подробности приводятся в разделе “Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN” на стр. 133.

**Включение кода:** Чтобы использовать операторы SQL или объявления переменных хоста PL/I из компонента секционированного набора данных, следует в том месте исходного текста, куда надо включить операторы, вставить следующий оператор SQL:

```
EXEC SQL INCLUDE имя-компоненты;
```

Нельзя использовать вложенные операторы SQL INCLUDE. Для включения операторов SQL или операторов DCL переменных хоста нельзя использовать оператор PL/I %INCLUDE. Следует с помощью препроцессора PL/I раскрыть все операторы %INCLUDE до запуска препроцессора DB2. Директивы препроцессора PL/I нельзя использовать внутри операторов SQL.

**Поля:** Операторы SQL могут располагаться между 2 и 72 позициями, если только препроцессор DB2 не заданы другие границы. Если EXEC SQL начинается до указанной левой границы, препроцессор DB2 не распознает оператор SQL.

**Имена:** Переменной хоста можно дать любое правильное имя PL/I. Нельзя использовать имена внешних записей или планов доступа, начинающиеся с 'DSN', и имена переменных хоста, начинающиеся с 'SQL'. Эти имена зарезервированы для DB2.

**Последовательные номера:** В операторы исходного текста, которые генерирует препроцессор DB2, не включаются последовательные номера. Компилятор PL/I выдает для строк без последовательных номеров сообщения IEL0378. На них можно не обращать внимания.

**Метки операторов:** Перед исполняемыми операторами SQL можно ставить метки операторов. Но перед операторами INCLUDE имя-текстового-файла и END DECLARE SECTION метки операторов ставить нельзя.

**Оператор WHENEVER:** Операнд условия GOTO в операторе SQL WHENEVER должен быть меткой в исходном тексте на языке PL/I, находящейся в области действия всех операторов SQL, на которые влияет WHENEVER.

**Использование символов двухбайтного набора (DBCS):** Использование DBCS в программах на языке PL/I с операторами SQL имеет следующие особенности:

- Если в исходном тексте на языке PL/I используется DBCS, то для следующих элементов языка действуют правила DB2:
  - Графические строки
  - Графические строковые константы
  - Идентификаторы хоста
  - Смешанные данные в символьных строках
  - Параметр MIXED DATA

Глава 3 книги *DB2 SQL Reference* содержит подробную информацию об элементах языка.

- Препроцессор PL/I преобразует формат констант DBCS. Чтобы предотвратить это преобразование, следует запускать препроцессор DB2 до препроцессора PL/I.
- При использовании в динамически подготовленных операторах SQL графических строковых констант или смешанных данных, если прикладной программе требуется компилятор PL/I Версии 2, то для динамически подготовленных операторов следует использовать формат смешанных констант PL/I.
  - Если оператор подготавливается из переменной хоста, следует поменять присваивание строкового значения на смешанную строку PL/I.
  - Если оператор подготавливается из смешанной строки PL/I, следует изменить ее на переменную хоста и затем поменять присваивание строкового значения на смешанную строку PL/I.

Например:

```
SQLSTMT = 'SELECT'<dbdb>'' FROM имя-таблицы'M;
EXEC SQL PREPARE FROM :SQLSTMT;
```

Информация о динамической подготовке операторов SQL приводится в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.

- Если нужно, чтобы идентификатор DBCS выглядел как графическая строка PL/I, следует использовать идентификатор с разделителями.
- При использовании символов DBCS в комментариях следует ограничивать их символами переключения shift-out и shift-in. Первый символ shift-in означает конец строки DBCS.
- В прикладных программах на языке PL/I можно объявлять имена переменных хоста, использующие символы DBCS. Правила использования имен переменных DBCS в PL/I те же, что и имеющиеся правила для обычных имен переменных DBCS в SQL, за исключением длины. Максимальная длина переменной хоста в DB2 составляет 64 однобайтных символа. Глава 3 книги *DB2 SQL Reference* содержит правила для обычных имен переменных DBCS в SQL.

Ограничения:

- Имена переменных DBCS должны содержать только символы DBCS. Смешивание символов однобайтного набора символов SBCS с символами DBCS в имени переменной DBCS приводит к непредсказуемым результатам.
- Имя переменной DBCS нельзя переносить на следующую строку.
- Препроцессор PL/I переводит символы DBCS, не входящие в канджи, в однобайтные символы кода EBCDIC. Чтобы избежать этого, следует либо пользоваться для имен переменных символами канджи DBCS, либо запускать компилятор PL/I без препроцессора PL/I.

**Особенности PL/I:** При написании программ на языке PL/I, имейте в виду следующие особенности.

- При компиляции программы на языке PL/I, в которой есть операторы SQL, необходимо использовать параметр компилятора PL/I CHARSET (60 EBCDIC).
  - В некоторых случаях сгенерированные в PL/I комментарии могут содержать точку с запятой. Точка с запятой приводит к сообщению компилятора IEL0239I, на которое можно не обращать внимания.
  - Сгенерированный код в объявлении PL/I может содержать функцию ADDR от поля, определенного как символьная строка произвольной длины (character varying). В этих случаях выдается сообщение IEL0872, на которое можно не обращать внимания.
  - Код, сгенерированный препроцессором DB2 в исходном тексте PL/I, может содержать функцию NULL(). В таких случаях выдается сообщение IEL0533I, на которое можно не обращать внимания, если только NULL не использовалось также в качестве переменной PL/I. Если в прикладной программе DB2 используется NULL в качестве переменной PL/I, необходимо также объявить NULL в качестве встроенной функции (DCL NULL BUILTIN:), чтобы избежать ошибок компилятора PL/I.
  - Если запускать макропроцессор PL/I до препроцессора DB2, макропроцессор может генерировать операторы SQL или операторы DCL переменных хоста.
- При использовании макропроцессора PL/I нельзя ставить в исходном тексте оператор PL/I \*PROCESS, чтобы передавать параметры компилятору PL/I. Необходимые параметры можно задать в параметре COPTION команды DSNH или в параметре PARM.PLI=параметры оператора EXEC в процедуре DSNHPLI.
- Использование многозадачных возможностей PL/I, когда сразу несколько задач выполняют операторы SQL, приводит к непредсказуемым результатам. Глава 2 книги *DB2 Command Reference* содержит описание команды RUN(DSN).

## Использование переменных хоста

Все переменные хоста должны быть явно объявлены до их первого использования в операторе SQL, если не задан параметр препроцессора TWOPASS. Если задан параметр препроцессора TWOPASS, переменные хоста должны быть объявлены до их первого использования в операторе DECLARE CURSOR.

Операторы PL/I, определяющие переменные хоста, можно заключать между операторами BEGIN DECLARE SECTION и END DECLARE SECTION.

Операторы BEGIN DECLARE SECTION и END DECLARE SECTION необходимо использовать, когда задан параметр препроцессора STDSQL(YES).

В операторе SQL все переменные хоста должны начинаться с двоеточия (:).

Имена переменных хоста должны быть уникальными в пределах программы, даже если переменные хоста находятся в разных блоках или процедурах. Чтобы сделать имена переменных хоста уникальными, можно дополнять их названием структуры.

Оператор SQL, использующий какую–либо переменную хоста, должен находиться в области действия оператора, объявившего переменную.

Переменные хоста должны быть скалярными переменными или структурами скалярных переменных. Нельзя определять переменные хоста как массивы, хотя можно использовать массив переменных–индикаторов, если назначить его структуре хоста.

## Объявление переменных хоста

Не все объявления PL/I допустимы как объявления переменных хоста. Препроцессор по умолчанию использует атрибуты данных, заданные в операторе PL/I DEFAULT. Если объявление какой–либо переменной неправильно, любой оператор SQL, в котором она упомянута, может вызвать сообщение "UNDECLARED HOST VARIABLE" (Необъявленная переменная хоста).

Препроцессор DB2 использует только имена и атрибуты данных переменных; атрибуты выравнивания, области действия и хранения игнорируются. Несмотря на то, что препроцессор DB2 игнорирует выравнивание, область действия и хранение, во избежание проблем при компиляции сгенерированного препроцессором DB2 исходного кода PL/I не следует пренебречь наложенным на их использование ограничениями:

- Объявление с атрибутом области действия EXTERNAL и атрибутом хранения STATIC должно иметь еще и атрибут хранения INITIAL.
- При использовании атрибута хранения BASED за ним должно идти выражение локатора элемента (element–locator–expression) PL/I.
- Можно использовать классы памяти переменных хоста STATIC, CONTROLLED, BASED, AUTOMATIC или любое сочетание этих классов. Но в соответствии с требованиями CICS программы должны быть повторно–входящими.

**Числовые переменные хоста:** Ниже показан синтаксис объявлений числовых переменных хоста.

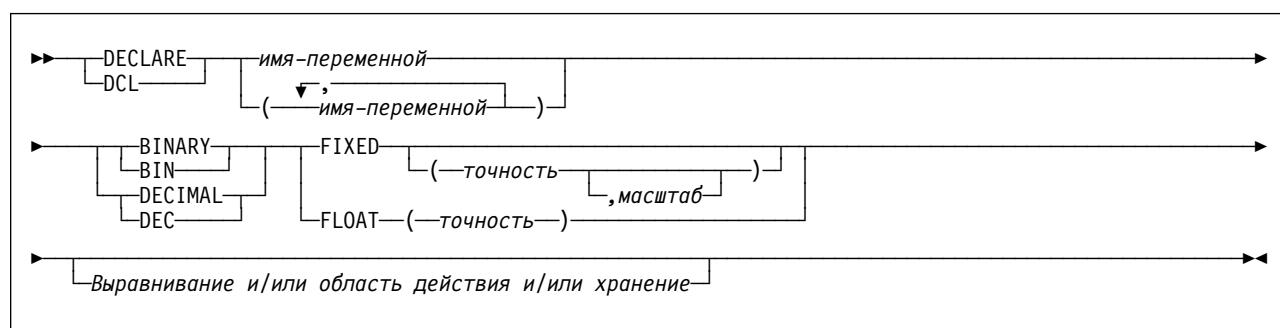


Рисунок 67. Числовые переменные хоста

### Примечания:

1. Можно задавать атрибуты переменных хоста в любом допустимом для PL/I порядке. Например, можно писать и BIN FIXED(31), и BINARY FIXED(31), и BIN(31) FIXED, и FIXED BIN(31).
2. Масштаб можно задавать только для DECIMAL FIXED.

**Символьные переменные хоста:** Ниже показан синтаксис определений символьных переменных хоста, кроме CLOB. Синтаксис CLOB показан на рис. 72 на стр. 223.

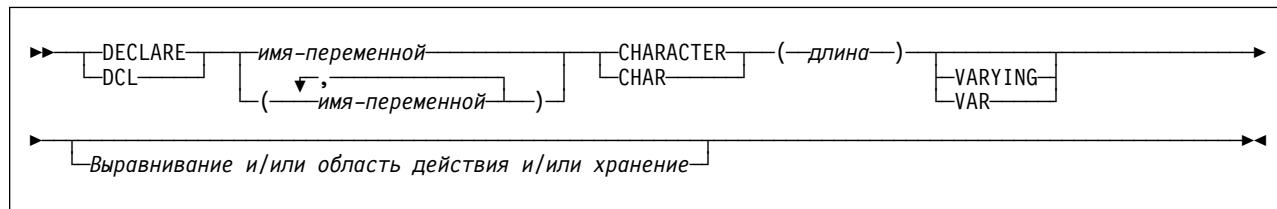


Рисунок 68. Символьные переменные хоста

**Графические переменные хоста:** Ниже показан синтаксис определений графических переменных хоста, кроме DBCLOB. Синтаксис DBCLOB показан на рис. 72 на стр. 223.

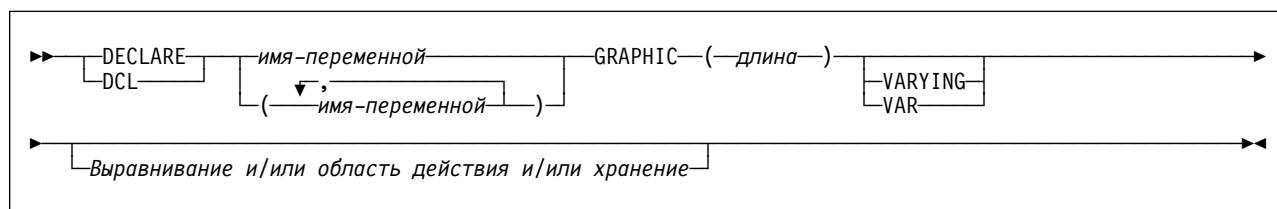


Рисунок 69. Графические переменные хоста

**Локаторы набора результатов:** Ниже показан синтаксис объявлений локаторов набора результатов. Использование этих переменных хоста описывается в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

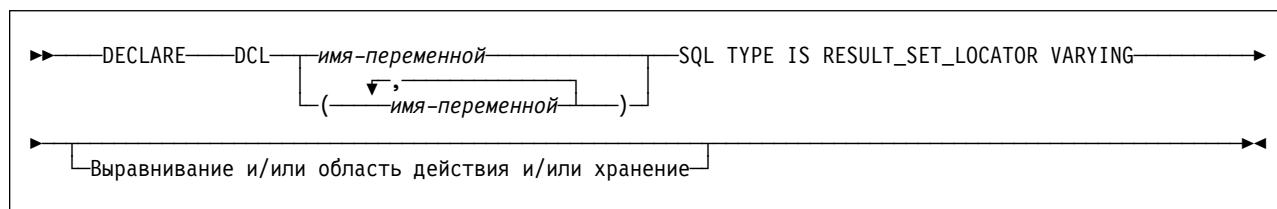


Рисунок 70. Локаторы набора результатов

**Локаторы таблиц:** Ниже показан синтаксис объявлений локаторов таблиц. Использование этих переменных хоста описывается в разделе “Доступ к таблицам переходов в пользовательских функциях” на стр. 307.

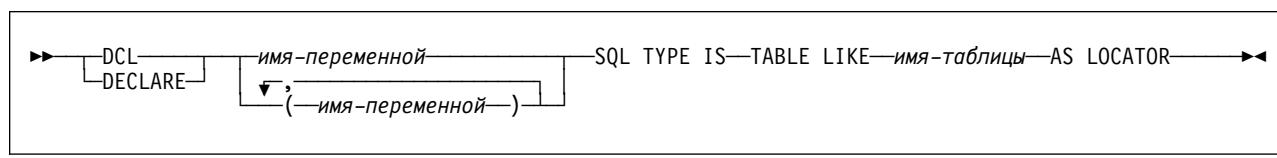


Рисунок 71. Локаторы таблиц

**Переменные и локаторы LOB:** Ниже показан синтаксис объявлений переменных хоста и локаторов BLOB, CLOB и DBCLOB. Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

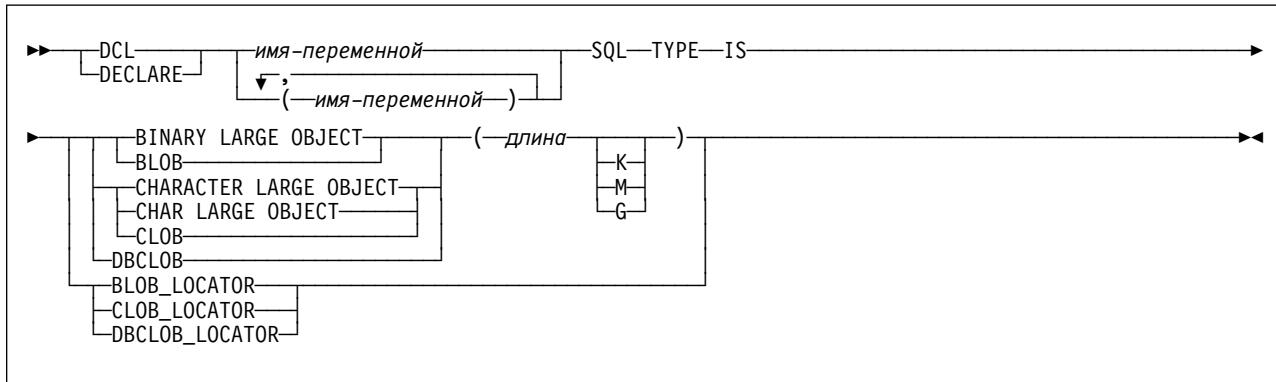


Рисунок 72. Переменные и локаторы LOB

**Переменные ROWID:** Ниже показан синтаксис определений переменных ROWID. Использование этих переменных хоста описывается в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

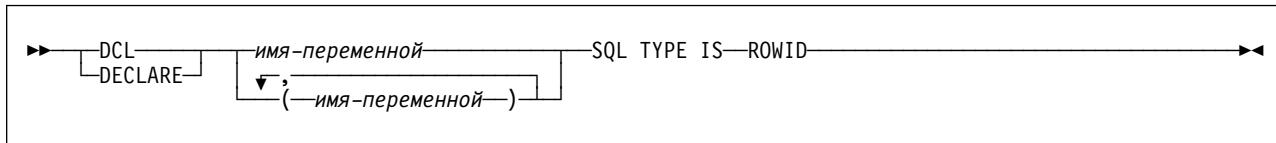


Рисунок 73. Переменные ROWID

## Использование структур хоста

Имя структуры хоста PL/I может быть именем структуры, подуровни которой называются скалярными переменными. Например:

```
DCL 1 A,
  2 B,
    3 C1 CHAR(...),
    3 C2 CHAR(...);
```

В этом примере B – имя структуры хоста, состоящей из скалярных переменных C1 и C2.

Можно использовать имя структуры в качестве сокращенной записи списка скалярных переменных. Имя переменной хоста можно дополнить именем структуры (например, STRUCTURE.FIELD). Структуры хоста ограничены двумя уровнями. Можно считать структуру хоста для данных DB2 именованной группой переменных хоста.

Объявление переменной структуры хоста должно заканчиваться точкой с запятой. Например:

```
DCL 1 A,
  2 B CHAR,
  2 (C, D) CHAR;
DCL (E, F) CHAR;
```

Атрибуты переменной хоста можно задавать в любом допустимом для PL/I порядке. Например, можно писать и BIN FIXED(31), и BIN(31) FIXED, и FIXED BIN(31).

Ниже показан синтаксис определений структур хоста.

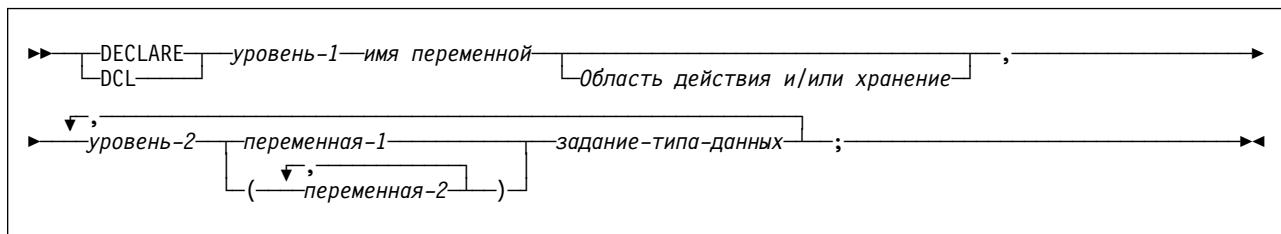


Рисунок 74. Структуры хоста

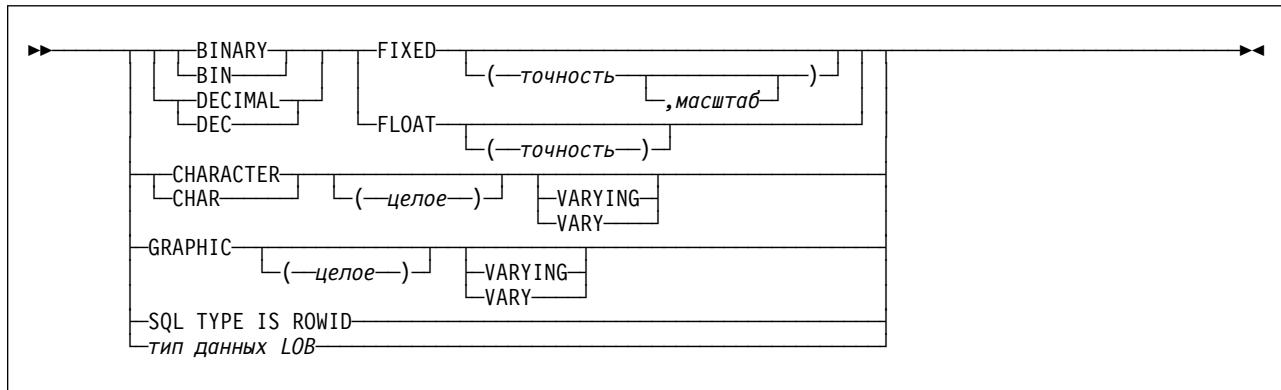


Рисунок 75. Задание типа данных

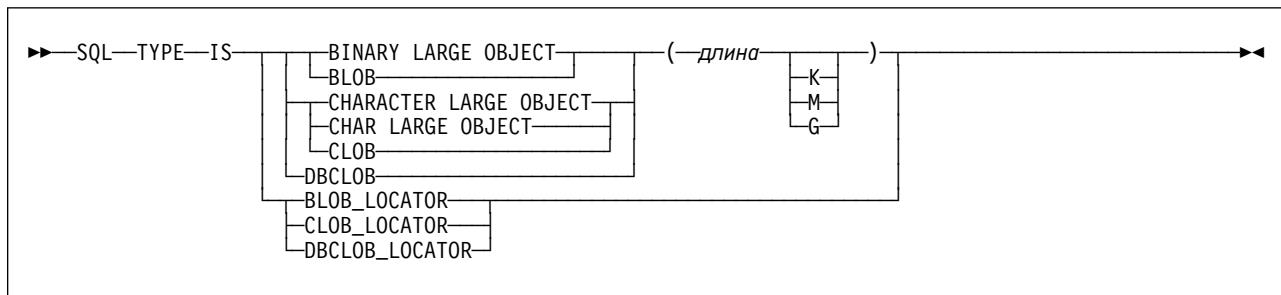


Рисунок 76. Тип данных LOB

**Примечание:** масштаб можно задавать только для DECIMAL FIXED.

## Определение эквивалентности типов данных SQL и PL/I

В Табл. 17 описан тип данных SQL и базовые значения SQLTYPE и SQLLEN, которые использует препроцессор DB2 для переменных хоста, найденных им в операторах SQL. Если переменная хоста задается с переменной-индикатором, SQLTYPE равен базовому SQLTYPE плюс 1.

Таблица 17 (Стр. 1 из 3). Типы данных SQL, которые использует препроцессор DB2 в объявлениях PL/I

Тип данных PL/I	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
BIN FIXED( <i>n</i> ) 1<=n<=15	500	2	SMALLINT
BIN FIXED( <i>n</i> ) 16<=n<=31	496	4	INTEGER

Таблица 17 (Стр. 2 из 3). Типы данных SQL, которые использует препроцессор DB2 в объявлениях PL/I

Тип данных PL/I	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
DEC FIXED( $p,s$ ) $0 \leq p \leq 15$ и $0 \leq s \leq p^1$	484	$p$ в байте 1, $s$ в байте 2	DECIMAL( $p,s$ )
BIN FLOAT( $p$ ) $1 \leq p \leq 21$	480	4	REAL или FLOAT( $n$ ) $1 \leq n \leq 21$
BIN FLOAT( $p$ ) $22 \leq p \leq 53$	480	8	DOUBLE PRECISION или FLOAT( $n$ ) $22 \leq n \leq 53$
DEC FLOAT( $m$ ) $1 \leq m \leq 6$	480	4	FLOAT (одинарной точности)
DEC FLOAT( $m$ ) $7 \leq m \leq 16$	480	8	FLOAT (двойной точности)
CHAR( $n$ )	452	$n$	CHAR( $n$ )
CHAR( $n$ ) VARYING $1 \leq n \leq 255$	448	$n$	VARCHAR( $n$ )
CHAR( $n$ ) VARYING $n > 255$	456	$n$	VARCHAR( $n$ )
GRAPHIC( $n$ )	468	$n$	GRAPHIC( $n$ )
GRAPHIC( $n$ ) VARYING $1 \leq n \leq 127$	464	$n$	VARGRAPHIC( $n$ )
GRAPHIC( $n$ ) VARYING $n > 127$	472	$n$	VARGRAPHIC( $n$ )
SQL TYPE IS RESULT_SET_LOCATOR	972	4	Локатор набора результатов <sup>2</sup>
SQL TYPE IS TABLE LIKE имя_таблицы AS LOCATOR	976	4	Локатор таблицы <sup>2</sup>
SQL TYPE IS BLOB_LOCATOR	960	4	Локатор BLOB <sup>2</sup>
SQL TYPE IS CLOB_LOCATOR	964	4	Локатор CLOB <sup>2</sup>
SQL TYPE IS DBCLOB_LOCATOR	968	4	Локатор DBCLOB <sup>2</sup>
SQL TYPE IS BLOB( $n$ ) $1 \leq n \leq 2147483647$	404	$n$	BLOB( $n$ )
SQL TYPE IS CLOB( $n$ ) $1 \leq n \leq 2147483647$	408	$n$	CLOB( $n$ )
SQL TYPE IS DBCLOB( $n$ ) $1 \leq n \leq 1073741823^3$	412	$n$	DBCLOB( $n$ ) <sup>3</sup>
SQL TYPE IS ROWID	904	40	ROWID

Таблица 17 (Стр. 3 из 3). Типы данных SQL, которые использует препроцессор DB2 в объявлениях PL/I

Тип данных PL/I	SQLTYPE переменной хоста	SQLLEN переменной хоста	Тип данных SQL
-----------------	--------------------------------	----------------------------	----------------

**Примечание:**

1. Если  $p=0$ , DB2 воспринимает это как DECIMAL(15). Например, DB2 воспринимает тип данных PL/I DEC FIXED(0,0) как DECIMAL(15,0), что соответствует типу данных SQL DECIMAL(15,0).
2. Нельзя использовать этот тип данных в качестве типа столбца.
3.  $n$  – количество двухбайтных символов.

В Табл. 18 указано, как определять переменные хоста, в которые записываются данные из базы данных. Этой таблицей можно воспользоваться, чтобы определить тип данных PL/I, эквивалентный какому-либо конкретному типу данных SQL. Например, если надо обработать данные типа TIMESTAMP, можно по таблице определить подходящий тип переменной хоста в программе, принимающей значение данных.

Таблица 18 (Стр. 1 из 2). Соответствия типов данных SQL объявлениям типов PL/I

Тип данных SQL	Эквивалент PL/I	Примечания
SMALLINT	BIN FIXED( $n$ )	$1 \leq n \leq 15$
INTEGER	BIN FIXED( $n$ )	$16 \leq n \leq 31$
DECIMAL( $p,s$ ) or NUMERIC( $p,s$ )	Если $p < 16$ : DEC FIXED( $p$ ) или DEC FIXED( $p,s$ )	$p$ – точность; $s$ – масштаб. $1 \leq p \leq 31$ и $0 \leq s \leq p$  Для $p > 15$ точного соответствия нет. (Более подробная информация приводится на странице 227.)
REAL или FLOAT( $n$ )	BIN FLOAT( $p$ ) или DEC FLOAT( $m$ )	$1 \leq n \leq 21$ , $1 \leq p \leq 21$ и $1 \leq m \leq 6$
DOUBLE PRECISION DOUBLE или FLOAT( $n$ )	BIN FLOAT( $p$ ) или DEC FLOAT( $m$ )	$22 \leq n \leq 53$ , $22 \leq p \leq 53$ и $7 \leq m \leq 16$
CHAR( $n$ )	CHAR( $n$ )	$1 \leq n \leq 255$
VARCHAR( $n$ )	CHAR( $n$ ) VAR	
GRAPHIC( $n$ )	GRAPHIC( $n$ )	$n$ означает количество двухбайтных символов, а не количество байтов. $1 \leq n \leq 127$
VARGRAPHIC( $n$ )	GRAPHIC( $n$ ) VAR	$n$ означает количество двухбайтных символов, а не количество байтов.
DATE	CHAR( $n$ )	Если используется подпрограмма-обработчик даты, $n$ определяется этой подпрограммой; в противном случае $n$ должно быть не меньше 10.

Таблица 18 (Стр. 2 из 2). Соответствия типов данных SQL объявлениям типов PL/I

Тип данных SQL	Эквивалент PL/I	Примечания
TIME	CHAR( <i>n</i> )	Если используется подпрограмма—обработчик времени, <i>n</i> определяется этой подпрограммой; в противном случае, <i>n</i> должно быть не меньше 6; чтобы включались секунды, <i>n</i> должно быть не меньше 8.
TIMESTAMP	CHAR( <i>n</i> )	<i>n</i> должно быть не меньше 19. Чтобы включать микросекунды, <i>n</i> должно быть 26; если <i>n</i> меньше 26, часть, содержащая микросекунды, будет усечена.
Локатор набора результатов	SQL TYPE IS RESULT_SET_LOCATOR	Этот тип данных можно использовать только для получения наборов результатов. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор таблиц	SQL TYPE IS TABLE LIKE имя—таблицы AS LOCATOR	Этот тип данных можно использовать для получения строк таблицы переходов только в пользовательской функции или в хранимой процедуре. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор BLOB	SQL TYPE IS BLOB_LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов BLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор CLOB	SQL TYPE IS CLOB_LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов CLOB. Нельзя использовать этот тип данных в качестве типа столбца.
Локатор DBCLOB	SQL TYPE IS DBCLOB_LOCATOR	Этот тип данных можно использовать только для обработки данных столбцов DBCLOB. Нельзя использовать этот тип данных в качестве типа столбца.
BLOB( <i>n</i> )	SQL TYPE IS BLOB( <i>n</i> )	$1 \leq n \leq 2147483647$
CLOB( <i>n</i> )	SQL TYPE IS CLOB( <i>n</i> )	$1 \leq n \leq 2147483647$
DBCLOB( <i>n</i> )	SQL TYPE IS DBCLOB( <i>n</i> )	<i>n</i> – количество двухбайтных символов. $1 \leq n \leq 1073741823$
ROWID	SQL TYPE IS ROWID	

**Замечания по объявлению и использованию переменных PL/I**

При объявлении переменных PL/I необходимо иметь в виду следующее.

**Типы данных PL/I, не имеющие эквивалента в SQL:** PL/I поддерживает некоторые типы данных, не имеющие эквивалента в SQL (например, переменные COMPLEX и BIT). В большинстве случаев для преобразования данных неподдерживаемых типов в данные типов, допустимых в SQL, можно использовать операторы PL/I.

**Типы данных SQL, не имеющие эквивалента в PL/I:** В PL/I нет эквивалента для десятичного типа данных (DECIMAL) с точностью больше 15. Для хранения значения такой переменной можно использовать:

- Десятичные переменные с точностью, не превышающей 15, если имеющиеся значения помещаются в них. При получении десятичного значения в десятичную переменную с масштабом меньше исходного столбца в базе данных дробная часть числа может быть усечена.
- Целочисленные переменные или переменные с плавающей точкой с преобразованием значения. При использовании целочисленных переменных будет теряться дробная часть числа. Если десятичное число может превышать максимальное значение целочисленной переменной или если вы хотите сохранить дробное значение, можно воспользоваться числами с плавающей точкой. Числа с плавающей точкой являются приближенными значениями действительных чисел. При присваивании десятичного значения переменной с плавающей точкой результат может отличаться от исходного числа.
- Символьные переменные хоста. Чтобы записать в такую переменную десятичное значение, надо воспользоваться функцией CHAR.

**Типы данных PL/I специального назначения:** Локаторы как тип данных поддерживаются и в PL/I, и в SQL. Локаторы нельзя использовать в качестве типов столбцов. Информация о том, как использовать эти типы данных, приводится в следующих разделах:

**Локатор набора результатов** “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579

**Локатор таблиц** “Доступ к таблицам переходов в пользовательских функциях” на стр. 307

**Локаторы больших объектов** “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253

**Правила областей действия PL/I:** Препроцессор DB2 не поддерживает правила областей действия PL/I.

**Переполнение:** Следует остерегаться переполнения. Например, если при записи значения столбца INTEGER в переменную хоста BIN FIXED(15) значение столбца больше, чем 32767 или меньше, чем –32768, будет получено предупреждение о переполнении или ошибка, в зависимости от того, была ли задана переменная–индикатор.

**Усечение:** Следует остерегаться усечения. Например, при получении 80–символьного значения столбца CHAR в переменную хоста CHAR(70), последние десять символов полученной строки будут усечены.

При получении значения столбца с плавающей точкой или десятичного значения в переменную хоста BIN FIXED(31) вся дробная часть будет удалена.

Аналогично при записи числа DB2 типа DECIMAL в соответствующую переменную PL/I значение может быть усечено. Это происходит из–за того, что значение DB2 DECIMAL может содержать до 31 цифры, но десятичное число PL/I может содержать не больше 15 цифр.

## Определение совместимости типов данных SQL и PL/I

При использовании в операторах SQL переменных хоста PL/I переменные должны быть совместимы по типу со значениями столбцов, для которых они должны использоваться.

- Числовые типы данных совместимы друг с другом. Столбцы типа SMALLINT, INTEGER, DECIMAL или FLOAT совместимы с переменной хоста PL/I типа BIN FIXED(15), BIN FIXED(31), DECIMAL(s,p), BIN FLOAT(*n*), где *n* – от 1 до 53, или DEC FLOAT(*m*), где *m* – от 1 до 16.
- Символьные типы данных совместимы друг с другом. Столбцы CHAR, VARCHAR или CLOB совместимы с символьными переменными фиксированной или произвольной длины хоста PL/I.
- Символьные типы данных частично совместимы с локаторами CLOB. Столбцу CHAR или VARCHAR можно назначить значение локатора CLOB, но значение из столбца CHAR или VARCHAR нельзя назначать переменной хоста типа локатора CLOB.
- Графические типы данных совместимы друг с другом. Столбцы GRAPHIC, VARGRAPHIC или DBCLOB совместимы с графическими переменными фиксированной или произвольной длины хоста PL/I.
- Графические типы данных частично совместимы с локаторами DBCLOB. Столбцу GRAPHIC или VARGRAPHIC можно назначить значение локатора DBCLOB, но значение из столбца GRAPHIC или VARGRAPHIC нельзя назначать переменной хоста типа локатора CLOB.
- Типы данных даты и времени совместимы с символьными переменными хоста. Столбцы DATE, TIME или TIMESTAMP совместимы с символьными переменными фиксированной или произвольной длины хоста PL/I.
- Столбцы BLOB совместимы только с переменными хоста BLOB.
- Столбцы ROWID совместимы только с переменными хоста ROWID.
- Переменная хоста совместима с пользовательским особым типом, если тип переменной хоста совместим с исходным типом этого пользовательского типа. Информация о назначении и сравнении пользовательских типов приводится в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

При необходимости DB2 автоматически преобразует строку фиксированной длины в строку произвольной длины или строку произвольной длины в строку фиксированной длины.

## Использование переменных–индикаторов

Переменная–индикатор – это двухбайтное целое число (BIN FIXED(15)). Если для переменной X задана переменная–индикатор, то, когда DB2 требуется занести в X пустое значение, в переменную–индикатор заносится отрицательное значение и X не обновляется. Программа перед использованием X должна проверить переменную–индикатор. Если переменная–индикатор отрицательна, это значит, что X имеет пустое значение, а любое реально записанное в нее значение можно игнорировать.

Если программа хочет с помощью X назначить столбцу пустое значение, она должна присвоить переменной–индикатору отрицательное значение. Тогда

DB2 присвоит столбцу пустое значение и проигнорирует любое значение, содержащееся в X.

Переменные—индикаторы объявляются так же, как и переменные хоста. Объявления этих двух типов переменных можно смешивать любым удобным способом. Более подробная информация о переменных—индикаторах приводится в разделе “Использование индикаторных переменных с переменными хоста” на стр. 113.

### **Пример:**

Возьмем оператор:

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS_CD,
                           :DAY :DAY_IND,
                           :BGN :BGN_IND,
                           :END :END_IND;
```

Переменные могут быть определены так:

```
DCL CLS_CD      CHAR(7);
DCL DAY        BIN FIXED(15);
DCL BGN        CHAR(8);
DCL END        CHAR(8);
DCL (DAY_IND, BGN_IND, END_IND)  BIN FIXED(15);
```

Можно задавать атрибуты переменных хоста в любом допустимом для PL/I порядке. Например, подходят и BIN FIXED(31), и BINARY FIXED(31), и BIN(31) FIXED, и FIXED BIN(31).

Ниже показан синтаксис переменной—индикатора.

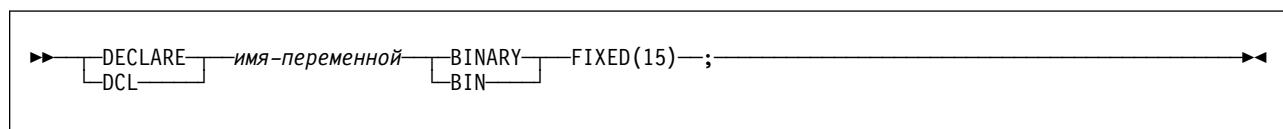


Рисунок 77. Переменная—индикатор

Ниже показан синтаксис массива индикаторов.



Рисунок 78. Массив индикаторов

## **Обработка кодов возврата ошибок SQL**

Чтобы преобразовать код возврата SQL в текстовое сообщение, можно воспользоваться подпрограммой DSNTIAR. DSNTIAR берет данные, содержащиеся в SQLCA, форматирует из них сообщение и помещает результат в назначенную прикладной программой область вывода сообщений.

Подробная информация о поведении DSNTIAR и связанные с ней понятия приводятся в разделе “Обработка кодов возврата ошибок SQL” на стр. 119.

### Синтаксис DSNTIAR

```
CALL DSNTIAR ( sqlca, сообщение, lrec );
```

Параметры DSNTIAR имеют следующее значение:

*sqlca* Область связи с SQL.

*сообщение*

Область вывода, в формате VARCHAR, в которую DSNTIAR помещает текст сообщения. В первом полуслове содержится длина оставшейся области; ее минимальное значение – 240.

В эту область помещаются выходные строки текста сообщения, длина каждой из которых задана параметром *lrec*. Например, формат области вывода можно задать так:

```
DCL DATA_LEN FIXED BIN(31) INIT(132);
DCL DATA_DIM FIXED BIN(31) INIT(10);
DCL 1 ERROR_MESSAGE AUTOMATIC,
      3 ERROR_LEN    FIXED BIN(15) UNAL INIT((DATA_LEN*DATA_DIM)),
      3 ERROR_TEXT(DATA_DIM) CHAR(DATA_LEN);
:
CALL DSNTIAR ( SQLCA, ERROR_MESSAGE, DATA_LEN );
```

где ERROR\_MESSAGE – имя области вывода сообщений, DATA\_DIM – количество строк в области вывода сообщений, а DATA\_LEN – длина каждой строки.

*lrec* Полное слово, содержащее длину логической записи для выходных сообщений, от 72 до 240.

Так как DSNTIAR – программа на языке ассемблера, в прикладную программу на языке PL/I необходимо включить следующие директивы:

```
DCL DSNTIAR ENTRY OPTIONS (ASM,INTER,RETCODE);
```

Пример вызова DSNTIAR из прикладной программы приводится в примере программы на ассемблере DB2 DSN8BP3, содержащемся в библиотеке DSN8610.SDSNSAMP. Инструкции о просмотре и печати исходного текста примера программы приводятся в разделе Приложение В, “Примеры прикладных программ” на стр. 883.

**CICS**

Если в прикладной программе CICS требуется обработка памяти CICS, следует вместо подпрограммы DSNTIAR использовать DSNTIAC.

Синтаксис DSNTIAC:

```
CALL DSNTIAC (eib , область_связи , sqlca , сообщение , lrecl);
```

У DSNTIAC есть дополнительные параметры, которые необходимо использовать для вызова подпрограмм, использующих команды CICS.

*eib* блок интерфейса с EXEC

*область\_связи* область связи

Более подробная информация по этим дополнительным параметрам приводится в соответствующем руководстве по написанию прикладных программ для CICS. Значение остальных параметров то же, что и для DSNTIAR. DSNTIAC и DSNTIAR форматируют SQLCA одинаково.

DSNTIA1 необходимо определить в CSD. Если программа загружает DSNTIAR или DSNTIAC, их также необходимо определить в CSD. Пример операторов создания записей CSD для использования с DSNTIAC, приводится в задании DSNTEJ5A.

Исходный текст DSNTIAC на ассемблере и задание DSNTEJ5A, которое ассемблирует и компонует DSNTIAC, находятся в наборе данных префикс.SDSNSAMP.

---

## Глава 3–5. Использование триггеров для работы с активными данными

Триггеры – это наборы операторов SQL, которые выполняются, когда в таблицах DB2 происходят определенные события. Как и ограничения, триггеры можно использовать для управления изменениями в таблицах DB2. Однако триггеры обладают большими возможностями по сравнению с ограничениями, потому что они реагируют на более широкий набор изменений и выполняют больший набор действий. Например, с помощью ограничения можно запретить изменение столбца оклада в таблице сотрудников, если новое значение превышает заданное число. Триггер же позволяет следить не только за величиной оклада, но и за тем, на сколько изменился оклад. Если разница между старым и новым окладом превышает определенное число, триггер может заменить значение оклада на допустимое и вызвать пользовательскую функцию, чтобы послать администратору уведомление о некорректной операции изменения.

Триггеры также позволяют внести в DB2 логику прикладной программы, что ускоряет разработку программ и упрощает поддержку. Например, требуется написать программы, которые управляют изменением оклада в таблице сотрудников, но при этом каждая программа, которая изменяет столбец оклада, должна включать процедуры проверки этих изменений. Лучший способ – задать триггер, который управляет изменениями столбца оклада. Тогда вместо каждой программы, изменяющей оклад, выполнять необходимые проверки будет DB2.

Эта глава содержит следующие сведения о триггерах:

- “Пример создания и использования триггера”
- “Составные части триггера” на стр. 235
- “Вызов хранимых процедур и пользовательских функций из триггеров” на стр. 240
- “Каскадное срабатывание триггеров” на стр. 242
- “Порядок активации триггеров” на стр. 243
- “Взаимодействие между триггерами и реляционными связями” на стр. 244
- “Создание триггеров, дающих надежные результаты” на стр. 245

---

### Пример создания и использования триггера

Триггеры автоматически выполняют набор операторов SQL, когда происходит определенное событие. Операторы SQL могут выполнять такие задачи, как проверка и редактирование изменений таблицы, чтение и изменение таблиц или вызов функций или хранимых процедур, выполняющих операции как внутри, так и вне DB2.

Триггеры создаются оператором CREATE TRIGGER. На рис. 79 на стр. 234 приведен пример оператора CREATE TRIGGER.

```

1 CREATE TRIGGER REORDER
2   3 AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
4
5   REFERENCING NEW AS N_ROW
6   FOR EACH ROW MODE DB2SQL
7 WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
8 BEGIN ATOMIC
      CALL ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                               N_ROW.ON_HAND,
                               N_ROW.PARTNO);
END

```

*Рисунок 79. Пример триггера*

Триггер включает следующие составные части:

- <sup>1</sup> Имя триггера (REORDER)
- <sup>2</sup> Момент активации триггера (AFTER)
- <sup>3</sup> Событие триггера (UPDATE)
- <sup>4</sup> Имя таблицы триггера (PARTS)
- <sup>5</sup> Внутриоператорное имя новой переменной перехода (N\_ROW)
- <sup>6</sup> Кратность (FOR EACH ROW)
- <sup>7</sup> Условие триггера (WHEN...)
- <sup>8</sup> Тело триггера (BEGIN ATOMIC...END;)

При выполнении этого оператора CREATE TRIGGER DB2 создает пакет под названием REORDER и связывает пакет триггера с таблицей PARTS. При создании DB2 запоминает отметку времени. Если для таблицы PARTS определяются другие триггеры, DB2 использует эту отметку времени, чтобы определить, какой триггер активировать первым. Теперь триггер готов для использования.

После того, как DB2 изменит столбцы ON\_HAND или MAX\_STOCKED в каждой строке таблицы PARTS, активируется триггер REORDER. Триггер вызывает хранимую процедуру с именем ISSUE\_SHIP\_REQUEST, если после изменения строки число имеющихся в наличии деталей меньше 10% от максимального количества хранящихся деталей. В условии триггера N\_ROW представляет собой значение измененной строки после события триггера.

Если триггер REORDER больше не нужен, его можно удалить с помощью оператора:

```
DROP TRIGGER REORDER RESTRICT;
```

Этот оператор удаляет триггер REORDER и связанный с ним пакет с именем REORDER.

Если вы отбрасываете таблицу PARTS, DB2 также отбрасывает триггер REORDER и его пакет.

## Составные части триггера

В этом разделе приводятся сведения для программирования каждого из элементов, составляющих триггер:

- Имени триггера
- Таблицы триггера
- Момента активации триггера
- События триггера
- Кратности
- Переменных перехода
- Таблиц перехода
- Действия триггера, которое состоит из *условия триггера и тела триггера*

**Имя триггера:** Для имени триггера используется обычный короткий идентификатор. Можно указать спецификатор или же предоставить его определение DB2. При создании пакета триггера DB2 использует спецификатор в качестве ID собрания пакета триггера. DB2 использует для определения спецификатора следующие правила:

- Если оператор CREATE TRIGGER выполняется статически, DB2 использует для плана или пакета, содержащего оператор, ID авторизации из опции связывания QUALIFIER. Если команда связывания не содержит опции QUALIFIER, DB2 использует владельца пакета или плана.
- Если оператор CREATE TRIGGER выполняется динамически, DB2 использует ID авторизации из специального регистра CURRENT SQLID.

**Таблица триггера:** При операциях вставки, изменения или удаления с этой таблицей активируется данный триггер. В операторе CREATE TRIGGER должна указываться локальная таблица. Нельзя задавать триггер для таблицы каталога или для производной таблицы.

**Момент активации триггера:** Существует два варианта – NO CASCADE BEFORE и AFTER. NO CASCADE BEFORE означает, что триггер активируется перед тем как DB2 произведет изменения таблицы триггера, и что действие триггера не активирует другие триггеры. AFTER означает, что триггер активируется после того, как DB2 сделает изменения таблицы триггера, причем триггер может активировать другие триггеры. Триггеры с моментом активации NO CASCADE BEFORE называются триггерами BEFORE. Триггеры с моментом активации AFTER называются триггерами AFTER.

**Событие триггера:** Каждый триггер связан с каким-либо событием. Триггер активируется, когда в таблице триггера происходит событие триггера. Событиями триггера могут быть следующие операции SQL:

- INSERT
- UPDATE
- DELETE

Событием триггера может быть также операция изменения или удаления, произведенная в результате действия реляционной связи с условиями ON DELETE SET NULL или ON DELETE CASCADE.

Триггеры не активируются после изменений таблиц, выполняемых утилитами DB2.

Если событие триггера – операция изменения, триггер называется триггером изменения. Триггеры для операций вставки называются триггерами вставки, а триггеры для операции удаления называются триггерами удаления.

Оператор SQL, выполняющий операцию SQL триггера, называется оператором SQL триггера.

Пример определения триггера, где событие – операция вставки:

```
CREATE TRIGGER NEW_HIRED
AFTER INSERT ON EMP
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
END
```

Каждое событие триггера связано с одной таблицей триггера и одной операцией SQL триггера. Если операция SQL триггера – операция изменения, событие может быть связано с определенными столбцами таблицы триггера. В этом случае триггер активируется, только если изменяются какие-либо из указанных столбцов. Например, приведенный ниже триггер PAYROLL1 активируется только при операциях изменения столбцов SALARY или BONUS таблицы PAYROLL:

```
CREATE TRIGGER PAYROLL1
AFTER UPDATE OF SALARY, BONUS ON PAYROLL
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
    VALUES(PAYROLL_LOG(USER, 'UPDATE', CURRENT TIME, CURRENT DATE));
END
```

**Кратность:** Оператор SQL триггера может изменять несколько строк таблицы. Кратность триггера указывает, активируется ли триггер один раз для каждого выполненного оператора SQL триггера или же для каждой строки, которую изменяет оператор. У кратности могут быть следующие значения:

- FOR EACH ROW

Триггер активируется один раз для каждой строки таблицы триггера, которую изменила DB2. Если оператор SQL триггера не изменяет ни одной строки, триггер не активируется. Однако если оператор SQL триггера изменяет значение строки на то же значение, триггер активируется. Например, если определен триггер UPDATE для таблицы COMPANY\_STATS, следующий оператор SQL активирует его.

```
UPDATE COMPANY_STATS SET NBEMP = NBEMP;
```

- FOR EACH STATEMENT

При выполнении оператора SQL триггер активируется один раз. Он активируется, даже если оператор SQL триггера не изменяет ни одной строки.

Триггеры с кратностью FOR EACH ROW называются триггерами строки. Триггеры с кратностью FOR EACH STATEMENT называются триггерами оператора. Триггеры оператора могут быть только триггерами AFTER.

Пример создания триггера строки:

```

CREATE TRIGGER NEW_HIRe
AFTER INSERT ON EMP
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
END

```

Триггер NEW\_HIRe активируется для каждой строки, которая вставляется в таблицу сотрудников.

**Переменные перехода:** При программировании триггера строки может понадобиться доступ к значениям столбцов каждой измененной строки таблицы триггера. Для этого нужно указать переменные перехода в условии REFERENCING оператора CREATE TRIGGER. Переменные перехода бывают двух типов:

- Старые переменные перехода, задаваемые условием OLD *переменная–перехода*, принимают значения столбцов до их изменения оператором SQL триггера. Старые переменные перехода можно определить для триггеров изменения и удаления.
- Новые переменные перехода, задаваемые условием NEW *переменная–перехода*, принимают значения столбцов после их изменения оператором SQL триггера. Новые переменные перехода можно определить для триггеров изменения и вставки.

В следующем примере используются новые переменные перехода для получения значения оклада сотрудника после его изменения:

```

CREATE TRIGGER BIGPAY AFTER UPDATE OF SALARY ON EMP
REFERENCING NEW AS MODIFIED
FOR EACH ROW MODE DB2SQL WHEN (MODIFIED.SALARY > 99999)
BEGIN ATOMIC
    CALL BIGPAY_LIST(MODIFIED.EMPNO, MODIFIED.FIRSTNME,
                      MODIFIED.MIDINIT, MODIFIED.LASTNAME,
                      MODIFIED.SALARY);
END

```

**Таблицы переходов:** Чтобы получить доступ ко всему набору строк, измененных оператором SQL триггера, а не к отдельным строкам, нужно использовать таблицу переходов. Как и переменные перехода, таблицы переходов указываются в условии REFERENCING оператора CREATE TRIGGER. Таблицы переходов могут использоваться как в триггерах строк, так и в триггерах операторов. Таблицы переходов бывают двух типов:

- Старые таблицы переходов, задаваемые условием OLD\_TABLE *таблица–переходов*, принимают значения столбцов до их изменения оператором SQL триггера. Старые таблицы переходов можно определить для триггеров изменения и удаления.
- Новые таблицы переходов, задаваемые условием NEW\_TABLE *таблица–переходов*, принимают значения столбцов после их изменения оператором SQL триггера. Новые таблицы переходов можно определить для триггеров изменения и вставки.

Область действия имен старой и новой таблиц переходов – тело триггера. Если существует другая таблица с тем же именем, что и у таблицы переходов, при неспецифицированной ссылке на это имя из тела триггера будет выбрана

таблица переходов. Для обращения к другой таблице из тела триггера нужно использовать полностью специфицированное имя таблицы.

В приведенном ниже примере новая таблица переходов используется для получения набора строк, которые были вставлены в таблицу INVOICE:

```
CREATE TRIGGER LRG_ORDR
  AFTER INSERT ON INVOICE
  REFERENCING NEW_TABLE AS N_TABLE
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    SELECT LARGE_ORDER_ALERT(CUST_NO, TOTAL_PRICE, DELIVERY_DATE)
      FROM N_TABLE WHERE TOTAL_PRICE > 10000;
  END
```

Оператор SELECT в LRG\_ORDER вызывает пользовательские функции LARGE\_ORDER\_ALERT для каждой строки таблицы переходов, удовлетворяющей условию WHERE (TOTAL\_PRICE > 10000).

**Действие триггера:** Когда активируется триггер, выполняется действие триггера. У каждого триггера есть одно действие триггера, состоящее из условия триггера и тела триггера.

**Условие триггера:** Чтобы при выполнении определенных условий выполнялось действие триггера, запрограммируйте условие триггера. Условие триггера подобно предикату в операторе SELECT, только начинается ключевым словом WHEN, а не WHERE. Если условие триггера не включено в действие триггера, тело триггера выполняется каждый раз, когда активируется триггер.

У триггера строки DB2 проверяет условие триггера один раз для каждой измененной строки таблицы триггера. У триггера оператора DB2 проверяет условие триггера один раз при каждом выполнении оператора SQL триггера.

В приведенном ниже примере условие триггера вызывает выполнение тела триггера, если заказано больше деталей, чем имеется в наличии:

```
CREATE TRIGGER CK_AVAIL
  NO CASCADE BEFORE INSERT ON ORDERS
  REFERENCING NEW AS NEW_ORDER
  FOR EACH ROW MODE DB2SQL
  WHEN (NEW_ORDER.QUANTITY >
        (SELECT ON_HAND FROM PARTS
         WHERE NEW_ORDER.PARTNO=PARTS.PARTNO))
  BEGIN ATOMIC
    VALUES(ORDER_ERROR(NEW_ORDER.PARTNO, NEW_ORDER.QUANTITY));
  END
```

**Тело триггера:** В тело триггера помещаются операторы SQL, выполняющиеся, когда верно условие триггера. Тело триггера начинается с BEGIN ATOMIC и кончается END. В теле триггера не разрешается использовать переменные хоста или маркеры параметров.

Набор операторов, которые можно использовать в теле триггера, зависит от времени активации триггера. Табл. 19 на стр. 239 показывает, какие операторы SQL можно использовать в триггерах разного типа.

Таблица 19. Допустимые операторы SQL для триггеров с различными моментами активации

Оператор SQL	Допустим для момента активации	
	BEFORE	AFTER
SELECT	Да	Да
VALUES	Да	Да
CALL	Да	Да
SIGNAL SQLSTATE	Да	Да
SET <i>переменная—перехода</i>	Да	Нет
INSERT	Нет	Да
UPDATE	Нет	Да
DELETE	Нет	Да

Ниже приводятся более подробные сведения о том, какие операторы SQL допустимо использовать в триггерах:

- SELECT, VALUES и CALL

Операторы SELECT и VALUES используются в теле триггера для условного или безусловного вызова пользовательской функции. Оператор CALL применяется для вызова хранимых процедур. Дополнительные сведения о вызове пользовательских функций и хранимых процедур из триггеровсмотрите в разделе “Вызов хранимых процедур и пользовательских функций из триггеров” на стр. 240.

- SET *переменная—перехода*

Поскольку триггеры BEFORE работают со строками таблицы до их изменения, в теле такого триггера нельзя выполнять операции, непосредственно изменяющие таблицу триггера. Но можно использовать оператор SET *переменная—перехода*, чтобы изменить значения строки перед тем как они будут помещены в таблицу. Например, следующий триггер использует новую переменную перехода, чтобы проставить текущую дату в качестве даты приема на работу нового сотрудника:

```
CREATE TRIGGER HIREDATE
  NO CASCADE BEFORE INSERT ON EMP
  REFERENCING NEW AS NEW_VAR
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    SET NEW_VAR.HIRE_DATE = CURRENT_DATE;
  END
```

- SIGNAL SQLSTATE

Оператор SIGNAL SQLSTATE можно использовать в теле триггера для того, чтобы сообщить об ошибочной ситуации и отменить изменения, произведенные триггером, и действия, вызванные реляционными связями для таблицы триггера. После выполнения оператора SIGNAL SQLSTATE DB2 возвращает прикладной программе SQLCA с кодом SQL -438. SQLCA также содержит следующие значения, задаваемые в операторе SIGNAL SQLSTATE:

- Пятисимвольное значение, которое DB2 использует для SQLSTATE
- Сообщение об ошибке, которое DB2 помещает в поле SQLERRMC

В приведенном ниже примере оператор SIGNAL SQLSTATE вынуждает DB2 возвратить SQLCA с SQLSTATE, равным 75001, и завершить операцию изменения оклада, если повышение оклада сотрудника превышает 20%:

```
CREATE TRIGGER SAL_ADJ
  BEFORE UPDATE OF SALARY ON EMP
  REFERENCING OLD AS OLD_EMP
  NEW AS NEW_EMP
  FOR EACH ROW MODE DB2SQL
  WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY * 1.20))
  BEGIN ATOMIC
    SIGNAL SQLSTATE '75001'
    ('Invalid Salary Increase - Exceeds 20%');
  END
```

- **INSERT, UPDATE и DELETE**

Поскольку в теле триггера могут встречаться операторы INSERT, UPDATE и DELETE, выполнение триггера может вызвать активацию других триггеров. Смотрите более подробные сведения в разделе “Каскадное срабатывание триггеров” на стр. 242.

Если при выполнении триггера какой-либо оператор SQL в теле триггера дает ошибку, DB2 отменяет все изменения, произведенные операцией SQL триггера и операторами SQL триггера. Но если тело триггера выполняет действия, которыми не управляет DB2, или для которых не используется та же координация принятия, что и для подсистемы DB2, в которой выполняется триггер, то DB2 не сможет отменить эти действия. Примеры внешних действий, не управляемых DB2:

- Изменения, принятием которых не управляет RRS
- Отправка сообщений по электронной почте

Если триггер выполняет внешние действия, не имеющие общей координации принятия с подсистемой DB2, в которой он выполняется, возникает ошибка, DB2 переводит процесс прикладной программы, который вызвал оператор, в состояние “необходим откат”. Прикладная программа должна в этом случае выполнить операцию отката, чтобы отменить внешние действия. Вот примеры внешних действий, имеющие общую координацию принятия с операцией SQL триггера:

- Распределенная операция изменения
- Выполнение в пользовательской функции или хранимой процедуре внешнего действия, которое затрагивает менеджер внешних ресурсов, находящийся под управлением принятия RRS.

---

## Вызов хранимых процедур и пользовательских функций из триггеров

Тело триггера может содержать только операторы SQL. Поэтому, чтобы триггер мог выполнить действия или использовать логическую структуру, недоступные в SQL, надо написать пользовательскую функцию или хранимую процедуру и вызвать их из тела триггера. Подробные сведения о написании и подготовке пользовательских функций и хранимых процедур можно найти в разделах “Глава 4–3. Создание и использование пользовательских функций”

на стр. 267 и “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

Поскольку триггер BEFORE не должен изменять таблицы, функции и процедуры, вызываемые из такого триггера, не должны содержать операторы INSERT, UPDATE или DELETE, изменяющие таблицу триггера.

**Чтобы вызвать пользовательскую функцию из триггера**, необходимо использовать операторы SELECT или VALUES. Оператор SELECT используется для условного вызова функции. Число вызовов пользовательской функции зависит от числа строк в наборе результатов оператора SELECT. Например, в приведенном ниже триггере оператор SELECT вызывает пользовательскую функцию LARGE\_ORDER\_ALERT для каждой строки таблицы переходов, где количество заказанных деталей больше 10000:

```
CREATE TRIGGER LRG_ORDR
  AFTER INSERT ON INVOICE
  REFERENCING NEW_TABLE AS N_TABLE
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    SELECT LARGE_ORDER_ALERT(CUST_NO, TOTAL_PRICE, DELIVERY_DATE)
      FROM N_TABLE WHERE TOTAL_PRICE > 10000;
  END
```

Оператор VALUES используется для безусловного выполнения функции, то есть функция выполняется один раз при каждом выполнении оператора в триггере оператора и для каждой строки в триггере строки. В следующем примере пользовательская функция PAYROLL\_LOG выполняется при каждой операции изменения, активирующей триггер PAYROLL1:

```
CREATE TRIGGER PAYROLL1
  AFTER UPDATE ON PAYROLL
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(PAYROLL_LOG(USER, 'UPDATE',
      CURRENT TIME, CURRENT DATE));
  END
```

**Чтобы вызвать из триггера хранимую процедуру**, используйте оператор CALL. Параметры в этом вызове могут быть литералами, переменными переходов, локаторами таблиц или выражениями.

## Передача таблиц переходов пользовательским функциям и хранимым процедурам

При вызове пользовательской функции или хранимой процедуры из триггера им может понадобиться доступ ко всему набору измененных строк. Для этого нужно передать им указатель на старую или новую таблицу переходов. Это делается с помощью локаторов таблиц.

Большая часть программного кода, обслуживающего локаторы таблиц, размещается в функции или хранимой процедуре, которая получает локатор. В разделе “Доступ к таблицам переходов в пользовательских функциях” на стр. 307 описывается, как функция определяет локатор и использует его для получения таблицы переходов. Для передачи таблицы переходов из триггера нужно при вызове функции или хранимой процедуры указать параметр TABLE

таблица—переходов. Например, следующий триггер передает локатор новой таблицы переходов хранимой процедуре CHECKEMP:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW_TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    CALL (CHECKEMP(TABLE NEWEMPS));
  END
```

## Каскадное срабатывание триггеров

Операция SQL, выполняемая триггером, может изменять таблицу триггера или другие таблицы с триггерами, поэтому DB2 активирует также и эти триггеры. Триггер, активируемый в результате работы другого триггера, может быть активирован на том же уровне, что и исходный триггер, или на другом уровне. Два триггера A и B активируются на разных уровнях, если триггер B активируется после того, как активируется триггер A, а завершается до того, как завершится триггер A. Если триггер B активируется после триггера A и завершается после завершения триггера A, тогда триггеры активируются на одном уровне.

Например, в следующих случаях триггеры A и B активируются на одном уровне:

- Для таблицы X определены два триггера A и B. A – триггер BEFORE, а B – триггер AFTER. Изменения в таблице X вызывают активацию обоих триггеров A и B.
- Триггер A изменяет таблицу X, которая связана реляционной связью с таблицей Y, для которой определен триггер B. Реляционная связь вызывает изменение таблицы Y, что активирует триггер B.

Триггеры A и B активируются на разных уровнях в следующих случаях:

- Триггер A определен для таблицы X, а триггер B определен для таблицы Y. B – триггер изменения. Изменение таблицы X активирует триггер A, который содержит в теле триггера оператор UPDATE. Этот оператор UPDATE активирует триггер B.
- Триггер A вызывает хранимую процедуру. Эта процедура содержит оператор INSERT для таблицы X, для которой определен триггер вставки B. Когда выполняется оператор INSERT для таблицы X, активируется триггер B.

Когда триггеры активируются на разных уровнях, это называется *каскадным срабатыванием триггеров*. Каскадное срабатывание может происходить только для триггеров AFTER, потому что для триггеров BEFORE DB2 не поддерживает каскадное срабатывание.

Чтобы не допустить бесконечное каскадное срабатывание триггеров, DB2 поддерживает только 16 уровней триггеров, хранимых процедур и пользовательских функций. Если триггер, хранимая процедура или пользовательская функция активируется на 17-ом уровне, DB2 возвращает код SQL –724 и отменяет все изменения SQL на 16 уровнях каскада. Однако, как и при любой другой ошибке SQL, происходящей во время работы

триггера, если были выполнены действия, не управляемые DB2, эти действия не могут быть отменены.

Вы можете написать программу–монитор, которая посылает требования IFI READS для сбора трассировочной информации об уровнях каскадного срабатывания триггеров, пользовательских функций и хранимых процедур в программах. О том, как написать программу–монитор, можно прочитать в разделе Приложения (Том 2) к *DB2 Administration Guide*.

## Порядок активации триггеров

Можно создать несколько триггеров с одними и теми же таблицей, событием и временем активации. Порядок активации в этом случае совпадает с порядком создания этих триггеров. DB2 запоминает отметки времени для каждого выполненного оператора CREATE TRIGGER. Когда в таблице происходит событие, вызывающее активацию нескольких триггеров, DB2 использует сохраненные отметки времени, чтобы определить, какой триггер активировать первым.

DB2 всегда активирует триггеры BEFORE, определенные для данной таблицы, перед триггерами AFTER, но внутри набора триггеров BEFORE, так же, как и внутри набора триггеров AFTER, порядок активации определяется отметками времени.

В приведенном ниже примере у триггеров NEWHIRE1 и NEWHIRE2 одно и то же событие триггера (INSERT), одна и та же таблица триггера (EMP) и один и тот же момент активации (AFTER). Предположим, что оператор CREATE TRIGGER для NEWHIRE1 был выполнен раньше, чем оператор CREATE TRIGGER для NEWHIRE2:

```
CREATE TRIGGER NEWHIRE1
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEWHIRE2
  AFTER INSERT ON EMP
  REFERENCING NEW AS N_EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE DEPTS SET NBEMP = NBEMP + 1
      WHERE DEPT_ID = N_EMP.DEPT_ID;
  END
```

При операции вставки в таблицу EMP DB2 активирует NEWHIRE1 первым, потому что он был создан раньше. Теперь предположим, что кто–то удалил и создал заново NEWHIRE1. Отметка времени у NEWHIRE1 теперь более поздняя, чем у NEWHIRE2, поэтому при следующей операции вставки в EMP NEWHIRE2 будет активирован до NEWHIRE1.

Если два триггера строк определены для одного и того же действия, сначала для всех затрагиваемых строк активируется тот триггер, который был создан раньше. Затем для всех затрагиваемых строк активируется второй триггер. В

предыдущем примере предположим, что оператор INSERT с подвыбором вставляет 10 строк в таблицу EMP. Для всех 10 строк активируется NEWHIRE1, а затем для всех 10 строк активируется NEWHIRE2.

## Взаимодействие между триггерами и реляционными связями

При создании триггеров необходимо понимать, как взаимодействуют триггеры и реляционные связи, определенные для таблиц, и как может повлиять на результаты порядок обработки этих связей и триггеров.

В общем случае, когда оператор SQL триггера S1 выполняет операцию вставки, изменения или удаления в таблице T1, производятся следующие действия:

1. DB2 определяет, какие строки T1 нужно изменять. Назовем этот набор строк M1. Содержимое M1 зависит от операции SQL:
  - Для операции удаления это все строки, удовлетворяющие критерию поиска для операции удаления с условием поиска, или текущая строка для операции удаления с позиционированием
  - В случае операции вставки это строка, которая задается в операторе VALUES, или строки, которые задаются в условии SELECT
  - В случае операции изменения это все строки, удовлетворяющие критерию поиска для операции изменения с условием поиска, или текущая строка для операции изменения с позиционированием

2. DB2 обрабатывает все триггеры BEFORE, определенные для T1, в порядке их создания.

Для каждого триггера BEFORE действие триггера выполняется один раз для каждой строки в M1. Если набор M1 пуст, действие триггера не выполняется.

Если при выполнении действия триггера происходит ошибка, DB2 отменяет все изменения, произведенные S1.

3. DB2 изменяет таблицу T1, выполняя оператор S1.

Если происходит ошибка, DB2 отменяет все изменения, произведенные S1.

4. Если набор M1 не пуст, DB2 применяет все перечисленные ниже ограничения и проверки, которые определены для таблицы T1:

- Реляционные связи
- Проверочные ограничения
- Проверки, вызванные изменением таблицы через производные таблицы, определенные с условием WITH CHECK OPTION

Реляционные связи с правилами DELETE CASCADE или DELETE SET NULL активируются до триггеров удаления и до триггеров изменения для зависимых таблиц.

Если какое-либо ограничение нарушено, DB2 отменяет все изменения, произведенные ограничениями или оператором S1.

5. DB2 обрабатывает все триггеры AFTER, определенные для T1, и все триггеры AFTER для таблиц, измененных при проверке ограничений, в порядке создания.

Каждый триггер строки AFTER выполняет действие триггера один раз для каждой строки из M1. Если набор M1 пуст, действие триггера не выполняется.

Каждый триггер оператора AFTER выполняет действие триггера один раз для каждого выполнения S1, даже если набор M1 пуст.

Если какое-нибудь действие триггера содержит операции SQL вставки, изменения или удаления, DB2 повторяет шаги 1–5 для каждой операции.

Если при выполнении действия триггера происходит ошибка или достигается 17-й уровень каскада триггеров, DB2 отменяет все изменения, произведенные на пятом шаге и на всех предыдущих шагах.

Например, пусть таблица DEPT – родительская таблица для EMP и:

- Столбец DEPTNO таблицы DEPT – первичный ключ.
- Столбец WORKDEPT таблицы EMP – внешний ключ.
- Реляционная связь задана с условием ON DELETE SET NULL.

Предположим, для EMP определен следующий триггер:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW_TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(CHECKEMP(TABLE NEWEMPS));
  END
```

Предположим также, что оператор SQL удаляет из DEPT строку с номером отдела E21. По реляционной связи DB2 находит в таблице EMP строки со значением WORKDEPT, равным E21, и записывает в WORKDEPT пустое значение (null) для этих строк. Это эквивалентно операции изменения таблицы EMP, для которой существует триггер изменения EMPRAISE. Поэтому, так как EMPRAISE – триггер AFTER, он активируется после того, как действие реляционной связи записывает в WORKDEPT пустые значения.

## Создание триггеров, дающих надежные результаты

При создании триггеров и написании операторов SQL, активирующих эти триггеры, необходимо убедиться, что эти операторы на одном и том же наборе данных всегда будут давать одни и те же результаты. Ненадежные результаты могут быть получены по следующим двум основным причинам:

- Операторы UPDATE и DELETE с позиционированием, использующие несвязанные запросы, вынуждают триггеры работать с большим набором результатов, чем предполагалось.
- DB2 не всегда обрабатывает строки в одном и том же порядке, поэтому триггеры, действующие на много строк таблицы, в разное время могут генерировать разные таблицы результатов.

Вот примеры таких ситуаций.

### Пример: Влияние несвязанного запроса на действие триггера:

Предположим, таблицы T1 и T2 выглядят так:

Таблица T1	Таблица T2
A1	B1
==	==
1	1
2	2

Для T1 определен триггер:

```
CREATE TRIGGER TR1
  AFTER UPDATE OF T1
FOR EACH ROW
  MODE DB2SQL
  BEGIN ATOMIC
    DELETE FROM T2 WHERE B1 = 2;
  END
```

Теперь предположим, что прикладная программа выполняет операцию изменения с позиционированием с помощью следующих операторов:

```
EXEC SQL BEGIN DECLARE SECTION;
long hv1;
EXEC SQL END DECLARE SECTION;
:
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT A1 FROM T1
  WHERE A1 IN (SELECT B1 FROM T2)
  FOR UPDATE OF A1;
:
EXEC SQL OPEN C1;
:
while(SQLCODE>=0 && SQLCODE!=100)
{
  EXEC SQL FETCH C1 INTO :hv1;
  UPDATE T1 SET A1=5 WHERE CURRENT OF C1;
}
```

Когда DB2 первый раз выполняет оператор FETCH, который устанавливает указатель C1, DB2 проверяет подвыбор SELECT B1 FROM T2 и получает таблицу результатов, содержащую две строки и один столбец T2:

```
1
2
```

Когда DB2 первый раз выполняет оператор UPDATE с позиционированием, активируется триггер TR1. При выполнении тела триггера строка со значением 2 удаляется из T2. Однако из-за того, что SELECT B1 FROM T2 выполняется только один раз, при следующем выполнении оператора FETCH DB2 найдет вторую строку T1, хотя эта строка была удалена. Оператор FETCH устанавливает указатель на вторую строку T1, и эта строка изменяется. Операция изменения снова активирует триггер, который пытается удалить вторую строку T2, хотя она уже была удалена.

Чтобы исключить действия со второй строкой после ее удаления, поместите в объявление указателя связанный подзапрос:

```
DCL C1 CURSOR FOR
  SELECT A1 FROM T1 X
  WHERE EXISTS (SELECT B1 FROM T2 WHERE X.A1 = B1)
  FOR UPDATE OF A1;
```

В этом случае подзапрос SELECT B1 FROM T2 WHERE X.A1 = B1 выполняется для каждого оператора FETCH. Когда оператор FETCH выполняется первый раз, он устанавливает указатель на первую строку T1. Операция UPDATE активирует триггер, который удаляет вторую строку T2. Поэтому при следующем выполнении оператора FETCH строка не выбирается и не происходит операции изменения и действия триггера.

**Пример: Влияние порядка обработки строк на действие триггера:**

Следующий пример показывает, как порядок обработки строк может изменить результат работы триггера строки AFTER.

Пусть таблицы T1, T2 и T3 выглядят так:

Таблица T1	Таблица T2	Таблица T3
A1	B1	C1
==	==	==
1	(пусто)	(пусто)
2		

Для T1 определен триггер:

```
CREATE TRIGGER TR1
  AFTER UPDATE ON T1
  REFERENCING NEW AS N
FOR EACH ROW
  MODE DB2SQL
  BEGIN ATOMIC
    INSERT INTO T2 VALUES(N.C1);
    INSERT INTO T3 (SELECT B1 FROM T2);
  END
```

Теперь предположим, что программа выполняет следующий оператор UPDATE:

```
UPDATE T1 SET A1 = A1 + 1;
```

Содержимое таблиц T2 и T3 после выполнения оператора UPDATE зависит от порядка, в котором DB2 изменяет строки T1.

Если DB2 сначала изменяет первую строку T1, после первого выполнения оператора UPDATE и триггера таблицы будут содержать такие значения:

Таблица T1	Таблица T2	Таблица T3
A1	B1	C1
==	==	==
2	2	2
2		

После изменения второй строки T1 таблицы будут содержать:

Таблица T1	Таблица T2	Таблица T3
A1	B1	C1
==	==	==
2	2	2
3	3	2
3		

Но если DB2 сначала изменяет первую строку T1, после первого выполнения оператора UPDATE и триггера в таблицах будут такие значения:

Таблица T1	Таблица T2	Таблица T3
A1	B1	C1
==	==	==
1	3	3
3		

После изменения первой строки T1 таблицы будут содержать:

Таблица T1	Таблица T2	Таблица T3
A1	B1	C1
==	==	==
2	3	3
3	2	3
2		

---

## **Раздел 4. Использование объектно–реляционных расширений DB2**

<b>Глава 4–1. Введение в объектно–реляционные расширения DB2</b>	251
<b>Глава 4–2. Программирование больших объектов (LOB)</b>	253
Введение в LOB	253
Объявление переменных хоста больших объектов и локаторов больших объектов	257
Материализация большого объекта	262
Применение локаторов больших объектов для экономии памяти	262
Отложенная оценка выражений большого объекта для повышения производительности	263
Переменные–индикаторы и локаторы больших объектов	266
Правильные назначения для локаторов больших объектов	266
<b>Глава 4–3. Создание и использование пользовательских функций</b>	267
Определение, реализация и вызов пользовательских функций	267
Пример создания и использования скалярной пользовательской функции	268
Примеры пользовательских функций, поставляемые с DB2	269
Определение пользовательской функции	270
Компоненты определения пользовательской функции	270
Примеры определений пользовательских функций	272
Реализация внешней пользовательской функции	274
Написание пользовательской функции	274
Подготовка пользовательской функции к выполнению	312
Тестирование пользовательской функции	315
Вызов пользовательской функции	317
Синтаксис вызова пользовательской функции	317
Правильный выбор функции для выполнения DB2	318
Преобразование типов аргументов пользовательской функции	324
Что происходит при аварийном завершении работы пользовательской функции	325
Советы по вызову пользовательских функций	325
<b>Глава 4–4. Создание и применение пользовательских типов</b>	327
Введение в пользовательские типы	327
Применение пользовательских типов в прикладных программах	328
Сравнение пользовательских типов	328
Присваивание пользовательских типов	330
Применение пользовательских типов в UNION	332
Вызов функций с пользовательскими типами	332
Объединение пользовательских типов при помощи пользовательских функций и больших объектов	334



---

## Глава 4–1. Введение в объектно–реляционные расширения DB2

С помощью объектных расширений DB2 можно использовать для работы с вашей реляционной базой данных объектно–ориентированные концепции и методы, обогащающие набор типов данных и функций DB2. Эти расширения позволяют хранить экземпляры объектно–ориентированных типов данных в столбцах таблиц и работать с ними с помощью функций в операторах SQL. Кроме того, можно управлять типами операций, которые пользователи могут выполнить с этими типами данных.

В DB2 есть следующие объектные расширения:

- Большие объекты (LOB)

Предельная длина для типов данных VARCHAR и VARGRAPHIC – 32 Кбайт. Хотя этого может быть достаточно для текстовых данных небольшого и среднего объема, прикладные программы часто нуждаются в хранении больших текстовых документов. Может также понадобиться хранить разнообразные дополнительные типы данных, такие как аудио– и видеозаписи, рисунки, смешанные текстово–графические объекты и изображения. В DB2 имеется три типа данных для хранения этих объектов в качестве строк, длина которых ограничена 2 Гбайтами – 1. Эти три типа данных – двоичные большие объекты (BLOB), символьные большие объекты (CLOB) и двухбайтные символьные большие объекты (DBCLOB).

Детальное обсуждение больших объектовсмотрите в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

- Пользовательские типы

Пользовательский тип – это пользовательский тип данных, использующий то же внутреннее представление, что и встроенный тип данных, но считающийся отдельным и несовместимым с ним типом по семантическим причинам. Например, вам может потребоваться определить тип рисунка или тип аудиообъекта, имеющих совершенно различную семантику, но использующих для своего внутреннего представления встроенный тип данных двоичный большой объект.

Подробное обсуждение пользовательских типов данныхсмотрите в разделе “Глава 4–4. Создание и применение пользовательских типов” на стр. 327.

- Пользовательские функции

Встроенные функции, поставляемые вместе с DB2, составляют полезный набор функций, но они могут не удовлетворить все ваши требования. В этих случаях вам могут помочь пользовательские функции. Например, некоторая встроенная функция могла бы выполнить необходимые вычисления, но эта функция не принимает пользовательские типы, которые ей требуется передать. В этом случае можно определить функцию на основе встроенной функции, называемой пользовательской функцией с источником, которая принимает ваши пользовательские типы. Вам может понадобится в операторе SQL некоторое вычисление, для которого встроенная функция отсутствует. В этой ситуации следует определить и написать внешнюю пользовательскую функцию.

|  
| Подробное обсуждение пользовательских функций смотрите в разделе  
| “Глава 4–3. Создание и использование пользовательских функций” на  
| стр. 267.  
|

## Глава 4–2. Программирование больших объектов (LOB)

Термин *большой объект* и аббревиатура *LOB* относятся к объектам DB2, которые можно использовать для хранения больших объемов данных. LOB – это символьная строка переменной длины, которая может содержать до 2 Гбайта – 1 данных.

Есть три типа больших объектов:

- *Бинарный большой объект (binary large object, BLOB)*  
BLOB используется для хранения бинарных данных: изображений, звука, аудио- и видеоданных.
- *Символьный большой объект (character large object, CLOB)*  
CLOB используется для хранения символьных данных (SBCS или смешанных), например, документов.
- *Двухбайтный символьный большой объект (double-byte character large object, DBCLOB)*  
DBCLOB используется для хранения информации, содержащей только символы DBCS.

В этой главе приводятся следующие сведения о LOB:

- “Введение в LOB”
- “Объявление переменных хоста больших объектов и локаторов больших объектов” на стр. 257
- “Материализация большого объекта” на стр. 262
- “Применение локаторов больших объектов для экономии памяти” на стр. 262

### Введение в LOB

Работа с LOB включает определение LOB для DB2, занесение данных LOB в таблицы DB2 и применение операций SQL для обработки данных. Основное внимание в этой главе уделяется обработке данных LOB при помощи операторов SQL. Глава 6 *DB2 SQL Reference* и Раздел 2 *DB2 Administration Guide* содержат информацию об определении LOB для DB2. Раздел 2 *DB2 Utility Guide and Reference* содержит информацию о том, как утилиты DB2 обрабатывают данные больших объектов.

Вот основные этапы определения LOB и занесения данных в DB2:

1. Определите столбец большого объекта соответствующего типа и столбец идентификатора строки (*row identifier – ROWID*) в таблице DB2.  
Определяйте только один столбец ROWID, даже если в таблице есть несколько столбцов больших объектов.

В столбце большого объекта содержится только информация об объекте, но не сами данные объекта. Таблица, содержащая информацию о большом объекте, называется *базовой таблицей*. DB2 использует столбец ROWID для определения местонахождения данных большого объекта. В таблице, содержащей один и более столбцов LOB, необходим лишь один столбец ROWID. Столбец большого объекта и столбец ROWID можно

задать в операторе CREATE TABLE или ALTER TABLE. Если к уже имеющейся таблице добавляются и столбец большого объекта, и столбец ROWID, необходимо использовать два оператора ALTER TABLE. Добавьте ROWID первым оператором ALTER TABLE, а столбец большого объекта – вторым.

2. Создайте табличное пространство и таблицу для хранения данных большого объекта.

Эти табличное пространство и таблица называются табличным пространством большого объекта и вспомогательной таблицей. Если базовая таблица не разбита на разделы, нужно создать одно табличное пространство большого объекта и одну вспомогательную таблицу для каждого столбца большого объекта. Если базовая таблица разбита на разделы, то для каждого столбца большого объекта нужно создать по одному табличному пространству большого объекта и по одной вспомогательной таблице на каждый раздел. Например, если базовая таблица содержит три раздела, для каждого столбца большого объекта нужно создать три табличных пространства большого объекта и три вспомогательных таблицы. Создайте эти объекты с помощью операторов CREATE LOB TABLESPACE и CREATE AUXILIARY TABLE.

3. Создание индекса вспомогательной таблицы.

Каждая вспомогательная таблица должна иметь ровно один индекс. Для этого воспользуйтесь оператором CREATE INDEX.

4. Занесение данных LOB в DB2.

Если общая длина столбца большого объекта и строки базовой таблицы не превышает 32 Кбайт, для занесения данных в DB2 можно воспользоваться утилитой LOAD. В противном случае необходимо воспользоваться оператором INSERT или UPDATE. Несмотря на то, что данные хранятся во вспомогательной таблице, для утилиты LOAD или оператора INSERT надо задать базовую таблицу. Применить INSERT может быть сложно, поскольку в вашей программе потребуется достаточно памяти для хранения полной величины, входящей в столбец большого объекта.

Пусть, например, вы хотите добавить краткие резюме каждого сотрудника в таблицу сотрудников. Резюме занимают не более 5 Мбайт. Резюме сотрудников содержат однобайтные символы, поэтому вы можете определить эти данные в DB2 как символьные большие объекты. Следовательно, нужно добавить столбец типа CLOB длиной 5 Мбайт в таблицу сотрудников. Если столбец ROWID не был определен в таблице, до того, как будет добавлен столбец CLOB, нужно добавить столбец ROWID. Выполните оператор ALTER TABLE для добавления столбца ROWID, а затем выполните еще один оператор ALTER TABLE для добавления столбца CLOB. Это делается так:

```
ALTER TABLE EMP
  ADD ROW_ID ROWID GENERATED ALWAYS;
  COMMIT;
ALTER TABLE EMP
  ADD EMP_RESUME CLOB(1M);
  COMMIT;
```

Далее необходимо определить табличное пространство большого объекта и вспомогательную таблицу для хранения резюме сотрудников. Необходимо также определить индекс вспомогательной таблицы. Табличное пространство

большого объекта можно определить в той же базе данных, как связанное табличное пространство. Это делается так:

```
CREATE LOB TABLESPACE RESUMETS
  IN DSN8D61A
  LOG NO;
COMMIT;
CREATE AUXILIARY TABLE EMP_RESUME_TAB
  IN DSN8D61A.RESUMETS
  STORES DSN8610.EMP
  COLUMN EMP_RESUME;
CREATE UNIQUE INDEX XEMP_RESUME
  ON EMP_RESUME_TAB;
COMMIT;
```

Если значение опции связывания SQLRULES – STD, или в программе задано значение специального регистра CURRENT RULES – STD, при выполнении оператора ALTER для добавления столбца большого объекта DB2 создает табличное пространство большого объекта, вспомогательную таблицу и вспомогательный индекс.

Теперь, когда объекты DB2 для данных большого объекта определены, можно вводить в DB2 резюме сотрудников. Чтобы сделать это в программе SQL, можно определить переменную хоста для хранения резюме, скопировать резюме из файла в переменную хоста, а затем выполнить оператор UPDATE для копирования данных в DB2. Хотя данные вводятся во вспомогательную таблицу, в операторе UPDATE надо задать базовую таблицу. Объявление переменной хоста на языке С выглядит так:

```
SQL TYPE is CLOB (5M) resumedata;
```

Оператор UPDATE выглядит так:

```
UPDATE EMP SET EMP_RESUME=:resumedata
  WHERE EMPNO=:employeeenum;
```

В этом примере employeeenum – переменная хоста, указывающая, к какому сотруднику относится резюме.

Теперь, когда данные большого объекта находятся в DB2, можно писать программы SQL для обработки этих данных. Большинство операторов SQL можно применять для большого объекта. Например, можно применить операторы для извлечения из резюме информации об отделе, где работает сотрудник:

```

EXEC SQL BEGIN DECLARE SECTION;
  long deptInfoBeginLoc;
  long deptInfoEndLoc;
  SQL TYPE IS CLOB_LOCATOR resume;
  SQL TYPE IS CLOB_LOCATOR deptBuffer;
EXEC SQL END DECLARE SECTION;
:
  EXEC SQL DECLARE C1 CURSOR FOR
    SELECT EMPNO, EMP_RESUME FROM EMP;
:
EXEC SQL FETCH C1 INTO :employeenum, :resume;
:
EXEC SQL SET :deptInfoBeginLoc =
  POSSTR(:resumedata, 'Department Information');

EXEC SQL SET :deptInfoEndLoc =
  POSSTR(:resumedata, 'Education');

EXEC SQL SET :deptBuffer =
  SUBSTR(:resume, :deptInfoBeginLoc,
  :deptInfoEndLoc - :deptInfoBeginLoc);

```

Эти операторы используют переменные хоста типа локатор больших объектов (локатор LOB). Локатор большого объекта позволяет обрабатывать данные большого объекта, не извлекая их из таблицы в переменные хоста. При использовании локаторов больших объектов вашим программам требуется гораздо меньше памяти. Локаторы больших объектов описаны в разделе “Применение локаторов больших объектов для экономии памяти” на стр. 262.

**Примеры программ, использующих большие объекты:** В Табл. 20 перечислены программы примеров, поставляемые с DB2, чтобы помочь в написании программ, работающих с большими объектами. Все программы находятся в наборе данных DSN610.SDSNSAMP.

*Таблица 20. Примеры больших объектов, поставляемые с DB2*

Член набора данных, содержащий исходный текст	Язык	Функция
DSN8DLPL		Показывает, как создать таблицу со столбцами больших объектов, вспомогательную таблицу и вспомогательный индекс. Показывает также, как загрузить в табличное пространство большого объекта данные большого объекта, не превышающие 32 Кбайт.
DSN8DLPL	C	Демонстрирует использование локаторов больших объектов и операторов UPDATE для занесения бинарных данных в столбец типа BLOB.
DSN8DLRV	C	Демонстрирует использование локатора для обработки данных типа CLOB.
DSNTEP2	PL/I	Демонстрирует, как разместить SQLDA для строк, включающих данные больших объектов, и применить этот SQLDA для описания оператора занесения и выборки данных из столбцов больших объектов.

Раздел 2 книги *DB2 Installation Guide* содержит инструкции по подготовке и запуску программ примеров больших объектов.

## Объявление переменных хоста больших объектов и локаторов больших объектов

При написании программ для обработки данных больших объектов необходимо объявить переменные хоста для хранения данных больших объектов или переменные локаторов больших объектов для указания на данные больших объектов. В разделе “Применение локаторов больших объектов для экономии памяти” на стр. 262 приводится информация о локаторах больших объектов и рекомендации, в каких случаях следует их использовать вместо переменных хоста.

Переменные хоста больших объектов и локаторы больших объектов можно объявлять на ассемблере, С, С++, COBOL, FORTRAN и PL/I. Для каждой объявляемой переменной хоста или локатора SQL типа BLOB, CLOB или DBCLOB DB2 генерирует эквивалентное объявление, использующее типы данных языка хоста. При ссылке на переменную хоста большого объекта или на локатор в операторе SQL необходимо использовать ту переменную, которую вы указали в объявлении типа SQL. При ссылке на переменную хоста в операторе на языке хоста необходимо использовать переменную, генерируемую DB2. В разделе “Раздел 3. Кодирование SQL в прикладных программах хоста” на стр. 103 показан синтаксис объявлений больших объектов для каждого языка и их эквивалентов на языке хоста для каждого типа больших объектов.

DB2 поддерживает объявления переменных для больших объектов с длиной до 2 Гбайта – 1. Однако размер переменных хоста больших объектов ограничивается правилами языка хоста и объемом доступной программе памяти.

Ниже в примерах показано, как объявлять переменные хоста больших объектов на каждом из поддерживаемых языков. В каждой таблице левый столбец содержит объявление, которое вы кодируете в своей прикладной программе. Правый столбец содержит объявления, генерируемые DB2.

### Объявления переменных хоста больших объектов на ассемблере:

Табл. 21 показывает объявления некоторых типов больших объектов на ассемблере.

Таблица 21 (Стр. 1 из 2). Пример объявлений переменных больших объектов на ассемблере

Вы объявляете данную переменную	DB2 генерирует переменную
blob_var SQL TYPE IS BLOB 1M	blob_var DS 0FL4 blob_var_length DS FL4 blob_var_data DS CL655351 ORG blob_var_data+(1048476–65535)
clob_var SQL TYPE IS CLOB 40000K	clob_var DS 0FL4 clob_var_length DS FL4 clob_var_data DS CL655351 ORG clob_var_data +(40960000–65535)

*Таблица 21 (Стр. 2 из 2). Пример объявлений переменных больших объектов на ассемблере*

<b>Вы объявляете данную переменную</b>	<b>DB2 генерирует переменную</b>
dbclob_var SQL TYPE IS DBCLOB 4000K	dbclob_var DS 0FL4 dbclob_var_length DS FL4 dbclob_var_data DS GL655342 ORG dbclob_var_data+(8192000-65534)
blob_loc SQL TYPE IS BLOB_LOCATOR	blob_loc DS FL4
clob_loc SQL TYPE IS CLOB_LOCATOR	clob_loc DS FL4
dbclob_var SQL TYPE IS DBCLOB_LOCATOR	dbclob_loc DS FL4

Примечания к Табл. 21 на стр. 257:

- Поскольку язык ассемблера допускает символьные объявления не длиннее 65535 байт, DB2 делит объявления на языке хоста для переменных BLOB и CLOB хоста, превышающих 65535 байт, на две части.
- Поскольку язык ассемблера допускает графические объявления не длиннее 65534 байт, DB2 делит объявления на языке хоста для переменных DBCLOB хоста, превышающих 65534 байт, на две части.

**Объявления переменных хоста больших объектов на языке С:** Табл. 22 показывает объявления на языках С и С++ для некоторых типов больших объектов.

*Таблица 22. Примеры объявления переменных на языке С*

<b>Вы объявляете данную переменную</b>	<b>DB2 генерирует переменную</b>
SQL TYPE IS BLOB (1M) blob_var;	struct { unsigned long length; char data[1048576]; } blob_var;
SQL TYPE IS CLOB (40000K) clob_var;	struct { unsigned long length; char data[40960000]; } clob_var;
SQL TYPE IS DBCLOB (4000K) dbclob_var;	struct { unsigned long length; wchar_t data[4096000]; } dbclob_var;
SQL TYPE IS BLOB_LOCATOR blob_loc;	unsigned long blob_loc;
SQL TYPE IS CLOB_LOCATOR clob_loc;	unsigned long clob_loc;
SQL TYPE IS DBCLOB_LOCATOR dbclob_loc;	unsigned long dbclob_loc;

**Объявления переменных хоста больших объектов на языке COBOL:**

Табл. 23 на стр. 259 показывает объявления на языке COBOL для некоторых типов больших объектов.

Таблица 23. Примеры объявлений переменных на языке COBOL

Вы объявляете данную переменную	DB2 генерирует переменную
01 BLOB-VAR USAGE IS SQL TYPE IS BLOB(1M).	01 BLOB-VAR. 02 BLOB-VAR-LENGTH PIC 9(9) COMP. 02 BLOB-VAR-DATA. 49 FILLER PIC X(32767). <sup>1</sup> 49 FILLER PIC X(32767). <i>Repeat 30 times</i> ⋮ 49 FILLER PIC X(1048576–32*32767).
01 CLOB-VAR USAGE IS SQL TYPE IS CLOB(4000K).	01 CLOB-VAR. 02 CLOB-VAR-LENGTH PIC 9(9) COMP. 02 CLOB-VAR-DATA. 49 FILLER PIC X(32767). <sup>1</sup> 49 FILLER PIC X(32767). <i>Repeat 1248 times</i> ⋮ 49 FILLER PIC X(40960000–1250*32767).
01 DBCLOB-VAR USAGE IS SQL TYPE IS DBCLOB(4000K).	01 DBCLOB-VAR. 02 DBCLOB-VAR-LENGTH PIC 9(9) COMP. 02 DBCLOB-VAR-DATA. 49 FILLER PIC G(32767) USAGE DISPLAY–1. <sup>2</sup> 49 FILLER PIC G(32767) USAGE DISPLAY–1. <i>Repeat 1248 times</i> ⋮ 49 FILLER PIC X(20480000–1250*32767) USAGE DISPLAY–1.
01 BLOB-LOC USAGE IS SQL TYPE IS BLOB-LOCATOR.	01 BLOB-LOC PIC S9(9) USAGE IS BINARY.
01 CLOB-LOC USAGE IS SQL TYPE IS CLOB-LOCATOR.	01 CLOB-LOC PIC S9(9) USAGE IS BINARY.
01 DBCLOB-LOC USAGE IS SQL TYPE IS DBCLOB-LOCATOR.	01 DBCLOB-LOC PIC S9(9) USAGE IS BINARY.

Примечания к Табл. 23:

- Поскольку язык COBOL допускает символьные объявления не длиннее 32767 байт, для переменных хоста типа BLOB или CLOB длиннее 32767 байт DB2 создает несколько объявлений на языке хоста, каждое с длиной не более 32767 байт.
- Поскольку язык COBOL допускает графические объявления не длиннее 32767 двубайтных символов, для переменных хоста типа DBCLOB длиннее 32767 двубайтных символов DB2 создает несколько объявлений на языке хоста, каждую с длиной не более 32767 двубайтных символов.

**Объявления переменных хоста больших объектов на языке FORTRAN:**

Табл. 24 на стр. 260 показывает объявления на языке FORTRAN для некоторых типов больших объектов.

Таблица 24. Примеры объявлений переменных на языке FORTRAN

<b>Вы объявляете данную переменную</b>	<b>DB2 генерирует переменную</b>
SQL TYPE IS BLOB(1M) blob_var	CHARACTER blob_var(1048580) INTEGER*4 blob_var_LENGTH CHARACTER blob_var_DATA EQUIVALENCE( blob_var(1), + blob_var_LENGTH ) EQUIVALENCE( blob_var(5), + blob_var_DATA )
SQL TYPE IS CLOB(40000K) clob_var	CHARACTER clob_var(4096004) INTEGER*4 clob_var_length CHARACTER clob_var_data EQUIVALENCE( clob_var(1), + clob_var_length ) EQUIVALENCE( clob_var(5), + clob_var_data )
SQL TYPE IS BLOB_LOCATOR blob_loc	INTEGER*4 blob_loc
SQL TYPE IS CLOB_LOCATOR clob_loc	INTEGER*4 clob_loc

**Объявления переменных хоста больших объектов на языке PL/I:** Табл. 25 на стр. 261 показывает объявления на языке PL/I для некоторых типов больших объектов.

Таблица 25. Примеры объявлений переменных на языке PL/I

Вы объявляете данную переменную	DB2 генерирует переменную
DCL BLOB_VAR SQL TYPE IS BLOB (1M);	DCL 1 BLOB_VAR, 2 BLOB_VAR_LENGTH FIXED BINARY(31), 2 BLOB_VAR_DATA, <sup>1</sup> 3 BLOB_VAR_DATA1(32) CHARACTER(32767), 3 BLOB_VAR_DATA2 CHARACTER(1048576–32*32767);
DCL CLOB_VAR SQL TYPE IS CLOB (40000K);	DCL 1 CLOB_VAR, 2 CLOB_VAR_LENGTH FIXED BINARY(31), 2 CLOB_VAR_DATA, <sup>1</sup> 3 CLOB_VAR_DATA1(1250) CHARACTER(32767), 3 CLOB_VAR_DATA2 CHARACTER(40960000–1250*32767);
DCL DBCLOB_VAR SQL TYPE IS DBCLOB (4000K);	DCL 1 DBCLOB_VAR, 2 DBCLOB_VAR_LENGTH FIXED BINARY(31), 2 DBCLOB_VAR_DATA, <sup>2</sup> 3 DBCLOB_VAR_DATA1(2500) GRAPHIC(16383), 3 DBCLOB_VAR_DATA2 GRAPHIC(40960000–2500*16383);
DCL blob_loc SQL TYPE IS BLOB_LOCATOR;	DCL blob_loc FIXED BINARY(31);
DCL clob_loc SQL TYPE IS CLOB_LOCATOR;	DCL clob_loc FIXED BINARY(31);
DCL dbblob_loc SQL TYPE IS DBCLOB_LOCATOR;	DCL dbblob_loc FIXED BINARY(31);

Примечания к Табл. 25:

- Поскольку язык PL/I допускает символьные объявления не длиннее 32767 байт, для переменных хоста типа BLOB или CLOB длиннее 32767 байт DB2 создает объявления на языке хоста так:
  - Если длина большого объекта больше 32767 байт и делится нацело на 32767, DB2 создает массив 32767–байтных строк. Размерность этого массива – *длина*/32767.
  - Если длина большого объекта больше 32767 байт и не делится нацело на 32767, DB2 создает два объявления. Первое – массив 32767–байтных строк с размерностью *n*, равной *length*/32767. Второе объявление – это символьная строка длины *длина*–*n*\*32767.
- Поскольку язык PL/I допускает графические объявления не длиннее 16383 двухбайтных символов, DB2 создает объявления на языке хоста так:
  - Если длина большого объекта больше 16383 символов и делится нацело на 16383, DB2 создает массив 16383–символьных строк. Размерность этого массива – *длина*/16383.
  - Если длина большого объекта больше 16383 символов и не делится нацело на 16383, DB2 создает два объявления. Первое – массив 16383–символьных строк с размерностью *m*, равной *length*/16383. Второе объявление – это символьная строка длины *длина*–*m*\*16383.

## Материализация большого объекта

*Материализация большого объекта* означает помещение значения большого объекта в непрерывную область пространства данных. Поскольку значения больших объектов могут быть очень большими, DB2 избегает материализации данных больших объектов без крайней необходимости. Однако DB2 обязана материализовать большой объект, если ваша программа:

- Вызывает пользовательскую функцию с большим объектом в качестве аргумента
- Передает большой объект хранимой процедуре или из нее
- Назначает переменную хоста большого объекта переменной хоста локатора большого объекта
- Преобразует большой объект от одного CCSID в другой

**Пространства данных для материализации большого объекта:** Объем памяти в пространствах данных для материализации большого объекта зависит от многих факторов, в том числе от:

- Размера большого объекта
- Количество больших объектов, подлежащих материализации, в операторе

DB2 выделяет определенное число пространств данных для материализации большого объекта. Если в пространстве данных для материализации большого объекта недостаточно места, программа получит код SQLCODE –904.

Хотя и невозможно полностью избежать материализации больших объектов, ее можно свести к минимуму, применяя в прикладных программах локаторы больших объектов вместо переменных хоста. Сведения о применении локаторов больших объектовсмотрите в разделе “Применение локаторов больших объектов для экономии памяти.”

## Применение локаторов больших объектов для экономии памяти

Для извлечения данных большого объекта из таблицы DB2 можно определить переменные хоста, достаточно большие, чтобы вместить все данные большого объекта. Это требует от вашей программы выделения большого объема памяти, а от DB2 – перемещения большого количества данных, что может оказаться неэффективным и непрактичным. Вместо этого можно применить локаторы больших объектов. Локаторы больших объектов позволяют обрабатывать данные большого объекта, не извлекая их из таблицы DB2. Применение локаторов больших объектов для получения данных большого объекта представляется лучшим решением в следующих ситуациях:

- При передаче малой части большого объекта программе клиента
- Если большой объект целиком не помещается в памяти прикладной программы
- Если программе требуется временное значение большого объекта для выражения, но не требуется сохранять результат
- Когда решающее значение имеет производительность

Локатор большого объекта ассоциирован со значением большого объекта или выражением, а не со строкой в таблице DB2 или местонахождением физической памяти в табличном пространстве. Следовательно, после выбора

значения большого объекта при помощи локатора значение в локаторе обычно не меняется, пока текущая единица работы не закончится. Однако само значение большого объекта может измениться.

Если надо удалить связь между локатором и большим объектом до окончания единицы работы, выполните оператор FREE LOCATOR. Чтобы оставить связь локатора с большим объектом после окончания единицы работы, выполните оператор HOLD LOCATOR. После выполнения оператора HOLD LOCATOR локатор сохранит свое значение, пока не будет выполнен оператор FREE LOCATOR, или не будет завершена программа.

Если HOLD LOCATOR или FREE LOCATOR выполняются динамически, нельзя применить EXECUTE IMMEDIATE. Глава 6 *DB2 SQL Reference* содержит дополнительную информацию об операторах HOLD LOCATOR и FREE LOCATOR.

## Отложенная оценка выражений большого объекта для повышения производительности

DB2 не переносит байты в значение большого объекта, пока программа не присвоит выражение большого объекта переменной назначения. Это означает, что когда локатор большого объекта используется со строковыми функциями и операторами, можно создать выражение, которое DB2 не будет вычислять вплоть до момента присваивания. Это называется *отложенным вычислением* выражения большого объекта. Отложенное вычисление может повысить производительность ввода–вывода большого объекта.

Ниже показана программа на языке C с отложенным вычислением выражения большого объекта. Программа выполняется на клиенте и изменяет данные большого объекта на сервере. Программа отыскивает отдельное резюме (EMPNO = '000130') в таблице EMP\_RESUME. Затем она использует локаторы большого объекта для перегруппировки копии резюме (с EMPNO = 'A00130'). В этой копии раздел информации об отделе оказывается в конце резюме. Затем программа вставляет копию в EMP\_RESUME, не изменяя исходного резюме.

Поскольку программа использует локаторы большого объекта, а не занесение данных большого объекта в переменные хоста, данные большого объекта не перемещаются, пока не будет выполнен оператор INSERT. Более того, никакие данные большого объекта не перемещаются между клиентом и сервером.

1

```
EXEC SQL INCLUDE SQLCA;

/*****************/
/* Объявление переменных хоста */
/*****************/
EXEC SQL BEGIN DECLARE SECTION;
    char userid[9];
    char passwd[19];
    long      HV_START_DEPTINFO;
    long      HV_START_EDUC;
    long      HV_RETURN_CODE;
    SQL TYPE IS CLOB_LOCATOR HV_NEW_SECTION_LOCATOR;
    SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR1;
    SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR2;
    SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR3;
EXEC SQL END DECLARE SECTION;
```

*Рисунок 80 (Часть 1 из 3). Пример отложенного вычисления выражений большого объекта*

```

/*****************/
/* Удаление любого экземпляра "000130" из      */
/* предыдущих выполнений этого примера      */
/*****************/
EXEC SQL DELETE FROM EMP_RESUME WHERE EMPNO = 'A00130';

/*****************/
/* Применение выбора единичных строк          */
/* для получения документа                   */
/*****************/
EXEC SQL SELECT RESUME
    INTO :HV_DOC_LOCATOR1
    FROM EMP_RESUME
    WHERE EMPNO = '000130'
        AND RESUME_FORMAT = 'ascii';
/*****************/
/* Применение функции POSSTR для определения начала */
/* разделов информации об отделах и образования      */
/*****************/
EXEC SQL SET :HV_START_DEPTINFO =
    POSSTR(:HV_DOC_LOCATOR1, 'Department Information');

EXEC SQL SET :HV_START_EDUC =
    POSSTR(:HV_DOC_LOCATOR1, 'Education');

/*****************/
/* Замена раздела информации об отделах пустой информацией */
/*****************/
EXEC SQL SET :HV_DOC_LOCATOR2 =
    SUBSTR(:HV_DOC_LOCATOR1, 1, :HV_START_DEPTINFO -1)
    || SUBSTR (:HV_DOC_LOCATOR1, :HV_START_EDUC);
/*****************/
/* Связывание нового локатора с разделом информации      */
/* об отделах                                         */
/*****************/
EXEC SQL SET :HV_NEW_SECTION_LOCATOR =
    SUBSTR(:HV_DOC_LOCATOR1, :HV_START_DEPTINFO,
    :HV_START_EDUC -:HV_START_DEPTINFO);

/*****************/
/* Присоединение информации об отделах к концу           */
/* анкеты                                              */
/*****************/
EXEC SQL SET :HV_DOC_LOCATOR3 =
    :HV_DOC_LOCATOR2 || :HV_NEW_SECTION_LOCATOR;

```

*Рисунок 80 (Часть 2 из 3). Пример отложенного вычисления выражений большого объекта*

```

/*
/* Запись модифицированной анкеты в таблицу. Только здесь */
/* происходит реальное перемещение данных большого объекта. */
*/
EXEC SQL INSERT INTO EMP_RESUME VALUES ('A00130', 'ascii',
                                         :HV_DOC_LOCATOR3, DEFAULT);

/*
/* Освобождение локаторов */
*/
EXEC SQL FREE LOCATOR :HV_DOC_LOCATOR1, :HV_DOC_LOCATOR2, :HV_DOC_LOCATOR3;

```

*Рисунок 80 (Часть 3 из 3). Пример отложенного вычисления выражений большого объекта*

Примечания к рис. 80 на стр. 264:

- 1**      Здесь объявляются локаторы большого объекта.
- 2**      Этот оператор SELECT связывает локатор большого объекта HV\_DOC\_LOCATOR1 со значением столбца RESUME для сотрудника номер 000130.
- 3**      Следующие пять операторов SQL применяют локаторы большого объекта для обработки данных анкеты без перемещения данных.
- 4**      Вычисление выражений большого объекта в предшествующих операторах было отложено до выполнения этого оператора INSERT.
- 5**      Все локаторы большого объекта освобождаются, теряя связь с ассоциированными значениями.

## Переменные–индикаторы и локаторы больших объектов

Для переменных хоста, отличных от локаторов больших объектов, если выбирается пустое значение для переменной хоста, DB2 назначает отрицательное значение связанной с ней переменной–индикатору. Однако для локаторов больших объектов DB2 использует переменные–индикаторы иначе. Локатор большого объекта не может иметь пустое значение. Если значение из столбца большого объекта выбирается при помощи локатора большого объекта, и этот столбец содержит пустое значение, DB2 назначает пустое значение связанной переменной–индикатору. Значение локатора большого объекта не меняется. В среде клиент–сервер эта пустая информация записывается только на клиенте.

Если вы используете локаторы больших объектов для приема данных из столбцов, которые могут содержать пустые значения, определите переменные–индикаторы для локаторов больших объектов и проверяйте их после выборки данных в локаторы. Если после выборки переменная–индикатор имеет пустое значение, значение локатора большого объекта использовать нельзя.

## Правильные назначения для локаторов больших объектов

Хотя обычно локаторы больших объектов используются для занесения данных в столбцы больших объектов и для выборки данных из таких столбцов, локаторы больших объектов можно использовать и для занесения данных в столбцы CHAR, VARCHAR, GRAPHIC или VARGRAPHIC. Однако нельзя записывать данные из столбцов CHAR, VARCHAR, GRAPHIC или VARGRAPHIC в локаторы больших объектов.

## Глава 4–3. Создание и использование пользовательских функций

Пользовательские функции – расширение языка SQL. Пользовательская функция аналогична функции или подпрограмме на языке хоста. Однако часто она лучше подходит для программы SQL, поскольку может быть вызвана в операторе SQL.

В данной главе даются следующие сведения о пользовательских функциях:

- “Определение, реализация и вызов пользовательских функций”
- “Определение пользовательской функции” на стр. 270
- “Реализация внешней пользовательской функции” на стр. 274
- “Вызов пользовательской функции” на стр. 317

### Определение, реализация и вызов пользовательских функций

Существует два типа пользовательских функций:

- Пользовательские функции с источником, которые используют существующие встроенные или пользовательские функции
- Внешние пользовательские функции, создаваемые программистом на языке хоста

Пользовательские функции можно также разделить на *скалярные* и *табличные*:

- Скалярная пользовательская функция при каждом вызове возвращает ответ из одного значения.
- Табличная пользовательская функция возвращает оператору SQL, который к ней обращается, таблицу.

Внешняя пользовательская функция может быть как скалярной, так и табличной. Пользовательская функция с источником не может быть табличной.

Создание и использование пользовательских функций включает следующие этапы:

- Задание среды пользовательских функций

Этот этап, как правило, выполняет системный администратор. Среда пользовательских функций показана на рис. 81 на стр. 268. Действия по заданию и поддержанию среды пользовательских функций те же, что и для хранимых процедур в адресных пространствах, созданных WLM. Эти сведения приводятся в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

- Написание и подготовка пользовательских функций

Этот этап требуется только для внешних пользовательских функций.

Тот, кто выполняет этот этап, называется *разработчиком* пользовательской функции.

- Определение пользовательских функций для DB2

Тот, кто выполняет этот этап, называется *определяющим* пользовательской функции.

- Вызов пользовательских функций из прикладных программ

Тот, кто выполняет этот этап, называется *вызывающим* пользовательской функции.

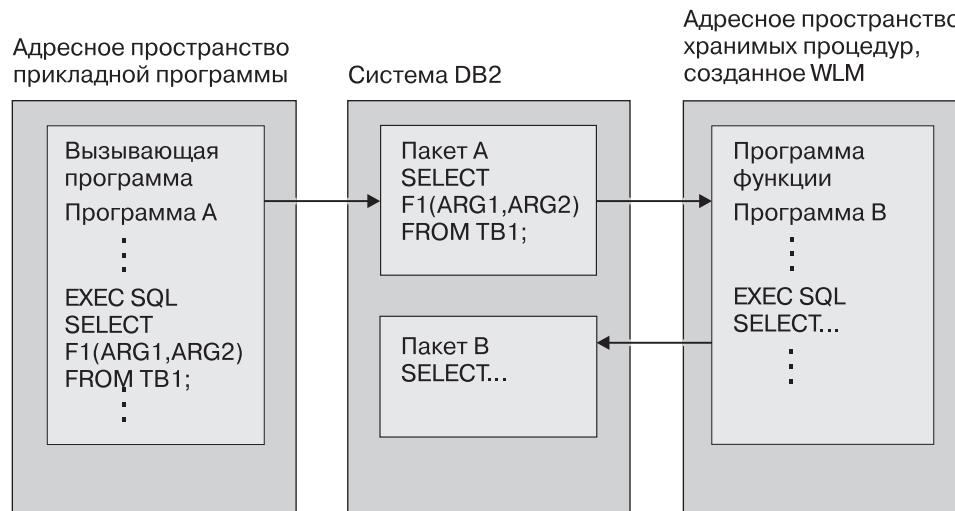


Рисунок 81. Среда пользовательских функций

## Пример создания и использования скалярной пользовательской функции

Предположим, что вашей организации нужна скалярная пользовательская функция, вычисляющая премию для каждого сотрудника. Все данные о сотрудниках, включая оклад, комиссионные и премии, хранятся в таблице сотрудников EMP. Исходные данные для вычисления премии – значения столбцов SALARY (оклад) и COMM (комиссионные). Результат работы функции записывается в столбец BONUS. Поскольку и входные, и выходные данные этой функции хранятся в таблице DB2, лучше использовать пользовательскую функцию.

Определяющий и вызывающий пользовательскую функцию определили, что у нее должны быть следующие характеристики:

- Имя пользовательской функции – CALC\_BONUS.
- Два входных параметра типа DECIMAL(9,2).
- Выходной параметр типа DECIMAL(9,2).
- Программа пользовательской функции написана на языке COBOL и ее загрузочный модуль должен называться CBONUS.

Поскольку нет подходящих встроенных или пользовательских функций, которые можно использовать для построения пользовательской функции с источником, разработчик функции должен написать внешнюю пользовательскую функцию. Разработчик выполняет следующие действия:

- Пишет пользовательскую функцию в виде программы на языке COBOL
- Прекомпилирует, компилирует и компонует программу
- Связывает пакет, если пользовательская функция содержит операторы SQL
- Тщательно тестирует программу
- Дает определяющемуполномочия выполнения пользовательской функции

Определяющий пользовательскую функцию выполняет оператор CREATE FUNCTION, чтобы зарегистрировать CALC\_BONUS на DB2:

```
CREATE FUNCTION CALC_BONUS(DECIMAL(9,2),DECIMAL(9,2))
RETURNS DECIMAL(9,2)
EXTERNAL NAME 'CBONUS'
PARAMETER STYLE DB2SQL
LANGUAGE COBOL;
```

Затем определяющий дает всем вызывающим полномочия выполнения CALC\_BONUS.

Вызывающие пользовательскую функцию пишут и подготавливают прикладные программы, вызывающие CALC\_BONUS. Вызывающий может, например, написать следующий оператор, в котором используется пользовательская функция для изменения поля BONUS в таблице сотрудников:

```
UPDATE EMP
SET BONUS = CALC_BONUS(SALARY,COMM);
```

Этот оператор можно выполнять статически или динамически.

## Примеры пользовательских функций, поставляемые с DB2

Чтобы помочь пользователям определять, писать и вызывать пользовательские функции, с DB2 поставляется большое число примеров таких функций. Все тексты находятся в наборе данных DSN610.SDSNSAMP.

В Табл. 26 приводится сводка характеристик примеров пользовательских функций.

Таблица 26 (Стр. 1 из 2). Примеры пользовательских функций, поставляемые с DB2

Имя пользовательской функции	Язык	Компонент с исходным текстом	Назначение
ALTDATE1	C	DSN8DUAD	Переводит текущую дату в формат, заданный пользователем
ALTDATE2	C	DSN8DUCD	Изменяет формат даты
ALTTIME3	C	DSN8DUAT	Переводит текущее время в формат, заданный пользователем
ALTTIME4	C	DSN8DUCT	Изменяет формат времени
DAYNAME	C <sup>++</sup>	DSN8EUDN	Определяет день недели для введенной даты
MONTHNAME	C <sup>++</sup>	DSN8EUMN	Возвращает название месяца для введенной даты
CURRENCY	C	DSN8DUCY	Переводит плавающее число в формат денежной единицы
TABLE_NAME	C	DSN8DUTI	Возвращает неспецифицированное имя таблицы для таблицы, производной таблицы или алиаса
TABLE_QUALIF	C	DSN8DUTI	Возвращает спецификатор для таблицы, производной таблицы или алиаса

Таблица 26 (Стр. 2 из 2). Примеры пользовательских функций, поставляемые с DB2

Имя пользовательской функции	Язык	Компонент с исходным текстом	Назначение
TABLE_LOCATION	C	DSN8DUTI	Возвращает положение для таблицы, производной таблицы или алиаса
WEATHER	C	DSN8DUWF	Возвращает таблицу сводок погоды из набора данных EBCDIC

Примечания к Табл. 26 на стр. 269:

1. Данный вариант ALTDATETIME имеет один входной параметр типа VARCHAR(13).
2. Данный вариант ALTDATETIME имеет три входных параметра типа VARCHAR(17), VARCHAR(13) и VARCHAR(13).
3. Данный вариант ALTTIME имеет один входной параметр типа VARCHAR(14).
4. Данный вариант ALTTIME имеет три входных параметра типа VARCHAR(11), VARCHAR(14) и VARCHAR(14).

DSN8DUWC содержит клиентскую программу, демонстрирующую вызов табличной пользовательской функции WEATHER.

DSNTEJ2U показывает, как определить и подготовить примеры пользовательских функций и клиентскую программу.

## Определение пользовательской функции

Перед тем как определить пользовательскую функцию для DB2, необходимо установить ее характеристики — имя, схему (спецификатор), а также число и типы данных входных параметров и типы возвращаемых значений. Затем нужно выполнить оператор CREATE FUNCTION, чтобы зарегистрировать информацию в каталоге DB2. Если после определения функции вы обнаружите, что некоторые свойства заданы неправильно, можете изменить их с помощью оператора ALTER FUNCTION. Оператор ALTER FUNCTION не может изменить некоторые характеристики определения пользовательской функции. Какие свойства позволяет изменить оператор ALTER FUNCTION, можно прочитать в разделе Глава 6 *DB2 SQL Reference*.

## Компоненты определения пользовательской функции

Характеристики, которые указываются в операторах CREATE FUNCTION или ALTER FUNCTION, отличаются для внешних пользовательских функций и пользовательских функций с источником. В Табл. 27 на стр. 271 перечислены свойства пользовательских функций, соответствующие параметры операторов CREATE FUNCTION и ALTER FUNCTION и указано, какие параметры действительны для внешних функций и функций с источником.

Таблица 27 (Стр. 1 из 2). Характеристики пользовательских функций

Характеристика	Параметр CREATE FUNCTION или ALTER FUNCTION	Допустим для функции с источником?	Допустим для внешней функции?
Имя пользовательской функции	FUNCTION	Да	Да
Типы входных параметров	FUNCTION	Да	Да
Типы выходных параметров	RETURNS RETURNS TABLE <sup>1</sup>	Да	Да
Особое имя	SPECIFIC	Да	Да
Внешнее имя	EXTERNAL NAME	Нет	Да
Язык	LANGUAGE ASSEMBLE LANGUAGE C LANGUAGE COBOL LANGUAGE PLI	Нет	Да
Детерминированность	NOT DETERMINISTIC DETERMINISTIC	Нет	Да
Типы операторов SQL в функции	NO SQL CONTAINS SQL READS SQL DATA MODIFIES SQL DATA	Нет	Да <sup>2</sup>
Имя исходной функции	SOURCE	Да	Нет
Стиль параметров	PARAMETER STYLE DB2SQL	Нет	Да
Адресное пространство для пользовательских функций	FENCED	Нет	Да
Вызов с пустым вводом	RETURNS NULL ON NULL INPUT CALLED ON NULL INPUT	Нет	Да
Внешние действия	EXTERNAL ACTION NO EXTERNAL ACTION	Нет	Да
Задание временной памяти	NO SCRATCHPAD SCRATCHPAD <i>длина</i>	Нет	Да
Вызов функции после обработки SQL	NO FINAL CALL FINAL CALL	Нет	Да
Возможность параллельной работы функции	ALLOW PARALLEL DISALLOW PARALLEL	Нет	Да <sup>2</sup>
Собрание пакетов	NO COLLID COLLID <i>ID-сборки</i>	Нет	Да
Среда WLM	WLM ENVIRONMENT <i>имя</i> WLM ENVIRONMENT <i>имя,*</i>	Нет	Да
Процессорное время вызова функции	ASUTIME NO LIMIT ASUTIME LIMIT <i>целое</i>	Нет	Да
Модуль загрузки остается в памяти	STAY RESIDENT NO STAY RESIDENT YES	Нет	Да
Тип программы	PROGRAM TYPE MAIN PROGRAM TYPE SUB	Нет	Да
Защита	SECURITY DB2 SECURITY USER SECURITY DEFINER	Нет	Да

Таблица 27 (Стр. 2 из 2). Характеристики пользовательских функций

Характеристика	Параметр CREATE FUNCTION или ALTER FUNCTION	Допустим для функции с источником?	Допустим для внешней функции?
Опции времени выполнения	RUN OPTIONS опции	Нет	Да
Передача информации среды DB2	NO DBINFO DBINFO	Нет	Да
Ожидаемое количество возвращаемых строк	CARDINALITY целое	Нет	Да <sup>1</sup>

Примечания к Табл. 27 на стр. 271:

1. RETURNS TABLE и CARDINALITY допустимы только для табличных пользовательских функций.
2. MODIFIES SQL DATA и ALLOW PARALLEL не допустимы для табличных пользовательских функций.

Глава 6 книги *DB2 SQL Reference* содержит полное описание параметров операторов CREATE FUNCTION и ALTER FUNCTION.

## Примеры определений пользовательских функций

**Пример: Определение внешней скалярной пользовательской функции:**

Программист написал пользовательскую функцию, которая ищет строку с максимальной длиной 200 в символьном большом объекте с максимальной длиной 500 Кбайт. Результат работы функции – плавающее число, но в операторах SQL пользователей требуется целое. Пользовательская функция написана на С и не содержит операторов SQL. Функцию определяет следующий оператор CREATE FUNCTION:

```
CREATE FUNCTION FINDSTRING (CLOB(500K), VARCHAR(200))
RETURNS INTEGER
CAST FROM FLOAT
SPECIFIC FINDSTRNCLOB
EXTERNAL NAME 'FINDSTR'
LANGUAGE C
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION
FENCED;
```

**Пример: Определение внешней скалярной пользовательской функции, переопределяющее оператор:** Программист написал пользовательскую функцию, переопределяющую встроенную в SQL операцию деления (/). Другими словами, эта функция вызывается при выполнении прикладной программой операторов типа:

```
UPDATE TABLE1 SET INTCOL1=INTCOL2/INTCOL3;
UPDATE TABLE1 SET INTCOL1="/"(INTCOL2,INTCOL3);
```

На входе пользовательской функции два целых значения. Результат функции имеет тип integer. Функция находится в схеме MATH, написана на ассемблере и не содержит операторов SQL. Функцию определяет следующий оператор CREATE FUNCTION:

```
CREATE FUNCTION MATH."/\" (INT, INT)
RETURNS INTEGER
SPECIFIC DIVIDE
EXTERNAL NAME 'DIVIDE'
LANGUAGE ASSEMBLE
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION
FENCED;
```

Предположим, что пользовательская функция FINDSTRING должна работать с типами данных BLOB и CLOB. Можно определить другой экземпляр функции, принимающий на вход тип BLOB:

```
CREATE FUNCTION FINDSTRING (BLOB(500K), VARCHAR(200))
RETURNS INTEGER
CAST FROM FLOAT
SPECIFIC FINDSTRINBLOB
EXTERNAL NAME 'FNDBLOB'
LANGUAGE C
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION
FENCED;
```

Каждый экземпляр пользовательской функции FINDSTRING реализуется отдельной прикладной программой.

**Пример: Определение пользовательской функции с источником:**

Предположим, что вам нужна пользовательская функция, которая ищет строку в значении пользовательского типа BOAT. BOAT основан на типе данных BLOB. Пользовательская функция FINDSTRING уже была определена. FINDSTRING выполняет требуемую функцию с данными типа BLOB. Поэтому можно определить пользовательскую функцию с источником на основе FINDSTRING для поиска строки в значениях типа BOAT. Эту функцию с источником определяет следующий оператор CREATE FUNCTION:

```
CREATE FUNCTION FINDSTRING (BOAT, VARCHAR(200))
RETURNS INTEGER
SPECIFIC FINDSTRINBOAT
SOURCE SPECIFIC FINDSTRINBLOB;
```

**Пример: Определение табличной пользовательской функции:** Прикладной программист написал пользовательскую функцию, которая принимает два значения и возвращает таблицу. Входные параметры функции:

- Стока длиной не более 30 символов, описывающая тему
- Стока длиной не больше 255 символов, содержащая текст для поиска

Пользовательская функция просматривает документы по заданной теме, чтобы найти заданную строку и возвращает список документов, удовлетворяющих критерию поиска, с аннотацией для каждого документа. Список имеет форму таблицы с двумя столбцами. Первый столбец – символьный столбец длиной 16, содержащий ID документов. Второй столбец – символьный столбец переменной длины с максимальной длиной 5000, содержащий аннотации к документам.

Пользовательская функция написана на языке COBOL, использует SQL только для запросов, при одинаковых входных данных всегда дает один и тот же результат и не должна выполняться параллельно. Это повторно—входимая программа, использующая информацию, полученную при ее предыдущих вызовах. Предполагается, что результат ее работы должен составлять около 20 строк.

Эту функцию определяет следующий оператор CREATE FUNCTION:

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
RETURNS TABLE (DOC_ID CHAR(16), DOC_ABSTRACT VARCHAR(5000))
EXTERNAL NAME 'DOCMTCH'
LANGUAGE COBOL
PARAMETER STYLE DB2SQL
READS SQL DATA
DETERMINISTIC
NO EXTERNAL ACTION
FENCED
SCRATCHPAD
FINAL CALL
DISALLOW PARALLEL
CARDINALITY 20;
```

## Реализация внешней пользовательской функции

В этом разделе описываются этапы реализации внешней пользовательской функции:

- “Написание пользовательской функции”
- “Подготовка пользовательской функции к выполнению” на стр. 312
- “Тестирование пользовательской функции” на стр. 315

## Написание пользовательской функции

Пользовательские функции аналогичны другим языкам программирования SQL. При написании в них можно включать статические и динамические операторы SQL, вызовы IFI и команды DB2, отдаваемые через вызовы IFI.

Пользовательская функция может также обращаться к удаленным данным с помощью следующих методов:

- по собственному протоколу DB2 с трехчастными именами таблиц или аliasами трехчастных имен
- по протоколу DRDA с трехчастными именами таблиц или аliasами трехчастных имен
- по протоколу DRDA с помощью операторов CONNECT или SET CONNECTION

Пользовательская функция и вызывающая ее программа могут иметь доступ к одному и тому же удаленному положению, если они используют один протокол.

Внешняя пользовательская функция может быть написана на языках ассемблер, C, C++, COBOL или PL/I. Функции, написанные на языке COBOL, могут включать объектно—ориентированные дополнения, как и другие программы DB2 на языке COBOL.

Дополнительные сведения для написания пользовательских функций содержатся в следующих разделах:

- “Ограничения для программ, использующих пользовательские функции”
- “Написание пользовательской функции как главной программы или подпрограммы”
- “Особенности параллелизма” на стр. 276
- “Передача значений параметров в пользовательскую функцию и из пользовательской функции” на стр. 278
- “Примеры передачи параметров пользовательской функции” на стр. 291
- “Использование специальных регистров в пользовательской функции” на стр. 304
- “Использование временной памяти в пользовательских функциях” на стр. 306
- “Доступ к таблицам переходов в пользовательских функциях” на стр. 307

## **Ограничения для программ, использующих пользовательские функции**

При написании пользовательских функций следует соблюдать следующие ограничения:

- DB2 использует соединения RRSAF для связи с пользовательской функцией, поэтому нельзя включать в функцию вызовы RRSAF. DB2 отклоняет любые вызовы RRSAF из пользовательской функции.
- Если пользовательская функция определена без параметров SCRATCHPAD или EXTERNAL ACTION, при разных вызовах она может выполняться в разных заданиях.
- В пользовательской функции нельзя выполнять операторы COMMIT и ROLLBACK.
- В скалярной пользовательской функции необходимо закрыть все открытые указатели. DB2 возвращает ошибку SQL, если после завершения скалярной пользовательской функции остаются открытые указатели.
- При выборе языка для написания пользовательской функции примите во внимание ограничения на число передаваемых подпрограмме параметров в выбранном языке. Особенно много параметров требуется для табличных пользовательских функций. Посмотрите в руководстве по программированию для языка, на котором вы собираетесь писать пользовательскую функцию, сколько параметров ей можно передать.

## **Написание пользовательской функции как главной программы или подпрограммы**

Пользовательская функция может быть написана либо как главная программа, либо как подпрограмма. Это должно согласовываться с заданными при ее определении параметрами PROGRAM TYPE MAIN или PROGRAM TYPE SUB. Главное различие состоит в том, что при начале работы главной программы Language Environment® выделяет память для прикладных программ, которую использует внешняя пользовательская функция. Когда работа программы завершается, языковая среда закрывает файлы и освобождает динамически выделенную память.

Написав пользовательскую функцию как подпрограмму и взяв на себя работу с памятью и файлами, можно повысить производительность. Заканчивая работу, пользовательская функция всегда должна освобождать выделенную

память. Для сохранения данных от вызова к вызову используйте временную память.

Табличную пользовательскую функцию, обращающуюся к внешним ресурсам, следует писать как подпрограмму. Убедитесь также, что определитель задал параметр EXTERNAL ACTION в операторе CREATE FUNCTION или ALTER FUNCTION. Переменные программы, используемые в подпрограмме, сохраняются от вызова к вызову, а использование параметра EXTERNAL ACTION гарантирует, что при каждом вызове функция будет находиться в одном и том же адресном пространстве.

## Особенности параллелизма

Если определяющий указал в определении скалярной пользовательской функции параметр ALLOW PARALLEL, и вызывающий оператор SQL выполняется параллельно, функция может выполняться в параллельных задачах. DB2 выполняет отдельный экземпляр пользовательской функции для каждой параллельной задачи. При написании функции необходимо понимать, как значения перечисленных ниже параметров согласуются с опцией ALLOW PARALLEL, чтобы избежать непредвиденных результатов:

- SCRATCHPAD

При вызове из оператора SQL пользовательской функции, определенной с параметром ALLOW PARALLEL, DB2 выделяет одну область временной памяти каждому экземпляру функции. Это может привести к непредсказуемым или некорректным результатам.

Предположим, например, что пользовательская функция использует временную память для подсчета числа вызовов. Если временная память выделяется для каждой параллельной задачи, подсчитывается число вызовов, произведенное данной параллельной задачей, а не для всего оператора SQL, что неправильно.

- FINAL CALL

Если пользовательская функция выполняет внешние действия, например, посыпает уведомление для каждого окончательного вызова функции, при таком вызове вместо одного уведомления будут посланы уведомления для каждой параллельной задачи.

- EXTERNAL ACTION

Некоторые пользовательские функции с внешними действиями могут получать неверные результаты, если функция выполняется параллельными задачами.

Например, если функция посылает сообщение при каждом начальном вызове, будет отправлено по одному сообщению для каждой параллельной задачи вместо одного на весь вызов функции.

- NOT DETERMINISTIC

Недетерминированная пользовательская функция может давать ошибочные результаты при работе в параллельных задачах.

Например, предположим, что в параллельных задачах выполняется следующий запрос:

```
SELECT * FROM T1 WHERE C1 = COUNTER();
```

COUNTER – пользовательская функция, увеличивающая на единицу переменную во временной памяти при каждом вызове. Это недетерминированная функция, потому что при тех же входных данных могут быть получены разные выходные данные. Таблица T1 содержит один столбец с такими значениями:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

При непараллельном выполнении запроса DB2 вызывает COUNTER по одному разу на каждой строке таблицы T1, и имеется общая временная память, которая инициализируется DB2 при первом вызове функции. COUNTER возвращает 1 при первом вызове, 2 при втором и т.д. Соответственно, таблица результатов запроса имеет вид:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Теперь предположим, что запрос выполняется параллельно, и DB2 создает три параллельных задачи. Для каждой задачи DB2 выполняет предикат WHERE C1 = COUNTER(). Это значит, что каждая параллельная задача вызывает собственный экземпляр пользовательской функции с собственной временной памятью. DB2 присваивает временной памяти нулевое значение при первом вызове функции в каждой из параллельных задач.

Если задача 1 обрабатывает строки с 1 по 3, задача 2 – строки с 4 по 6, а задача 3 – с 7 по 10, будут получены следующие результаты:

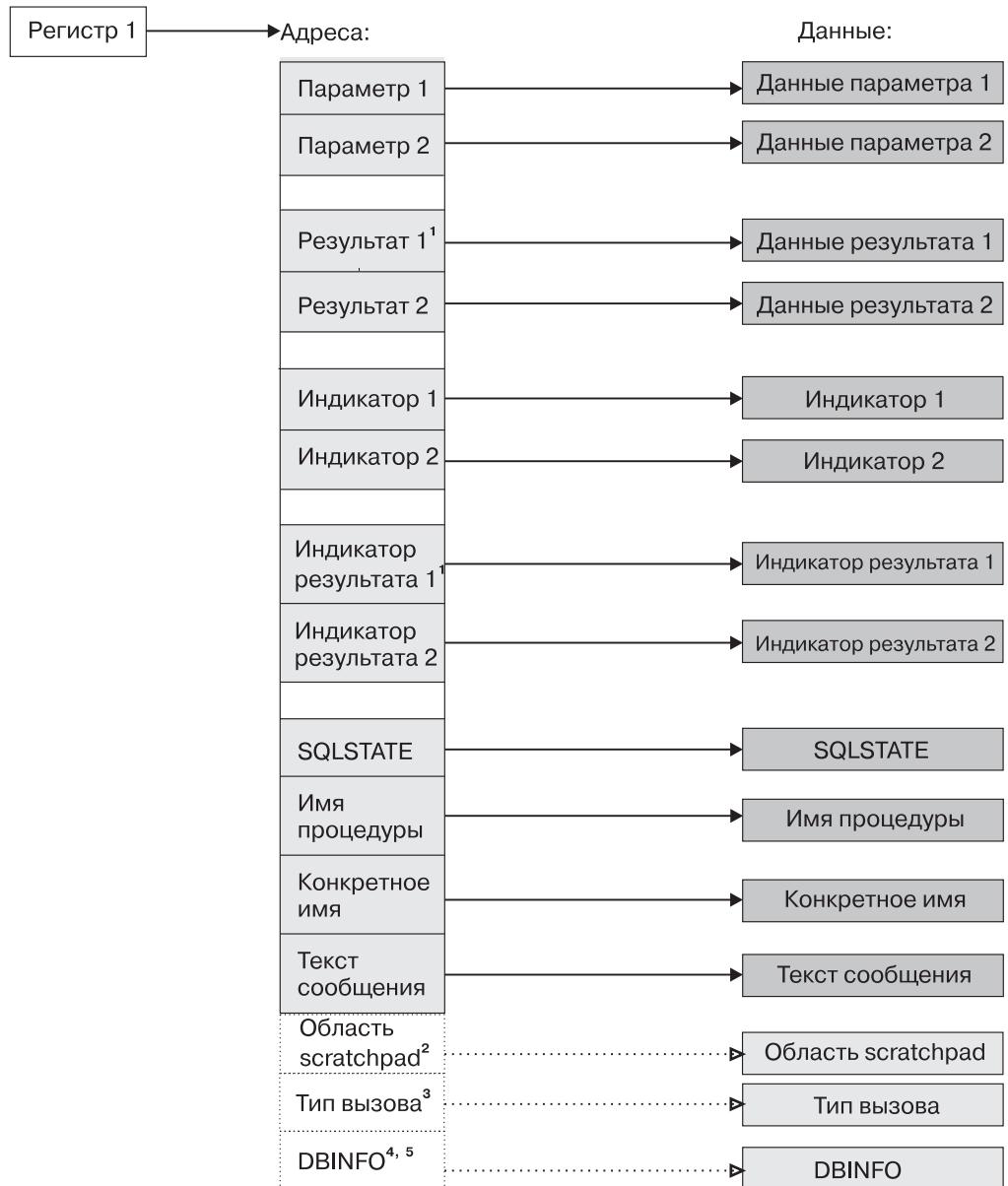
- При выполнении задачи 1 C1 имеет значения 1, 2 и 3, и COUNTER возвращает значения 1, 2 и 3, так что результат запроса – значения 1, 2 и 3.
- При выполнении задачи 2 C1 имеет значения 4, 5 и 6, но COUNTER возвращает значения 1, 2 и 3, поэтому запрос не возвращает ни одной строки.
- При выполнении задачи 3 C1 имеет значения 7, 8, 9 и 10, но COUNTER возвращает значения 1, 2, 3 и 4, поэтому запрос не возвращает ни одной строки.

Таким образом, вместо ожидаемых при этом запросе 10 строк DB2 возвращает только 3.

## **Передача значений параметров в пользовательскую функцию и из пользовательской функции**

Чтобы получать параметры из пользовательской функции и передавать параметры вызывающей программе, необходимо знать структуру списка параметров, смысл каждого параметра, а также задает ли значения параметров DB2 или пользовательская функция. В данном разделе описываются параметры и даются примеры получения списка параметров пользовательской функцией на каждом из языков хоста.

На рис. 82 на стр. 279 показана структура списка параметров, который DB2 передает пользовательской функции. Для каждого параметра дается объяснение.



<sup>1</sup> Для пользовательской скалярной функции передается только один результат и один индикатор результата.

<sup>2</sup> Передается, если в определении пользовательской функции указана опция SCRATCHPAD.

<sup>3</sup> Передается, если в определении пользовательской скалярной функции указана опция FINAL  
CALL;

для табличной пользовательской функции передается всегда.

<sup>4</sup> Для PL/I это значение - адрес указателя на данные DBINFO.

Рисунок 82. Соглашения о параметрах пользовательской функции

**Значения входных параметров:** DB2 получает входные параметры из списка параметров вызывающей программы, а пользовательская функция получает их в соответствии с правилами языка хоста, на котором написана. Количество входных параметров то же, что и количество параметров в вызове пользовательской функции. Если один из параметров в вызове функции

представляет собой выражение, DB2 вычисляет его значение и присваивает параметру результат.

Для всех типов данных, кроме больших объектов, ROWID и локаторов, смотрите Табл. 28, где указана совместимость типов данных хоста с типами данных в определении пользовательской функции. Для больших объектов, ROWID и локаторов смотрите таблицы Табл. 29, Табл. 30 на стр. 281, Табл. 31 на стр. 281 и Табл. 32 на стр. 282.

*Таблица 28. Список таблиц совместимости типов данных*

Язык	Таблица совместимых типов данных
Ассемблер	Табл. 9 на стр. 155
C	Табл. 11 на стр. 172
COBOL	Табл. 14 на стр. 197
PL/I	Табл. 18 на стр. 226

*Таблица 29. Совместимые объявления на ассемблере для больших объектов, ROWID и локаторов*

Тип данных SQL в определении	Объявление ассемблера
TABLE LOCATOR	DS FL4
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	
BLOB( <i>n</i> )	Если <i>n</i> <= 65535: var DS OFL4 var_length DS FL4 var_data DS CL <i>n</i> If <i>n</i> > 65535: var DS OFL4 var_length DS FL4 var_data DS CL65535 ORG var_data+( <i>n</i> -65535)
CLOB( <i>n</i> )	Если <i>n</i> <= 65535: var DS OFL4 var_length DS FL4 var_data DS CL <i>n</i> If <i>n</i> > 65535: var DS OFL4 var_length DS FL4 var_data DS CL65535 ORG var_data+( <i>n</i> -65535)
DBCLOB( <i>n</i> )	Если <i>m</i> (=2* <i>n</i> ) <= 65534: var DS OFL4 var_length DS FL4 var_data DS CL <i>m</i> Если <i>m</i> > 65534: var DS OFL4 var_length DS FL4 var_data DS CL65534 ORG var_data+( <i>m</i> -65534)
ROWID	DS HL2,CL40

*Таблица 30. Совместимые объявления для больших объектов, ROWID и локаторов на языке C*

<b>Тип данных SQL в определении</b>	<b>Объявление на языке C</b>
TABLE LOCATOR	длинное без знака
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	
BLOB( <i>n</i> )	struct {unsigned long length; char data[n]; } var;
CLOB( <i>n</i> )	struct {unsigned long length; char var_data[n]; } var;
DBCLOB( <i>n</i> )	struct {unsigned long length; wchar_t data[n]; } var;
ROWID	struct { short int length; char data[40]; } var;

*Таблица 31 (Стр. 1 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке COBOL*

<b>Тип данных SQL в определении</b>	<b>Объявление на языке COBOL</b>
TABLE LOCATOR	01 var PIC S9(9) USAGE IS BINARY.
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	
BLOB( <i>n</i> )	If n <= 32767: 01 var. 49 var-LENGTH PIC 9(9) USAGE COMP. 49 var-DATA PIC X( <i>n</i> ). If length > 32767: 01 var. 02 var-LENGTH PIC S9(9) USAGE COMP. 02 var-DATA. 49 FILLER PIC X(32767). 49 FILLER PIC X(32767). : 49 FILLER PIC X(mod( <i>n</i> ,32767)).

*Таблица 31 (Стр. 2 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке COBOL*

<b>Тип данных SQL в определении</b>	<b>Объявление на языке COBOL</b>
CLOB( <i>n</i> )	<p>Если <i>n</i> &lt;= 32767:</p> <p>01 var.</p> <p style="padding-left: 20px;">49 var-LENGTH PIC 9(9) USAGE COMP.</p> <p style="padding-left: 20px;">49 var-DATA PIC X(<i>n</i>). Если length &gt; 32767:</p> <p>01 var.</p> <p style="padding-left: 20px;">02 var-LENGTH PIC S9(9) USAGE COMP.</p> <p style="padding-left: 20px;">02 var-DATA. 49 FILLER PIC X(32767).</p> <p style="padding-left: 20px;">49 FILLER PIC X(32767). ⋮ 49 FILLER PIC X(mod(<i>n</i>,32767)).</p>
DBCLOB( <i>n</i> )	<p>Если <i>n</i> &lt;= 32767:</p> <p>01 var.</p> <p style="padding-left: 20px;">49 var-LENGTH PIC 9(9) USAGE COMP.</p> <p style="padding-left: 20px;">49 var-DATA PIC G(<i>n</i>) USAGE DISPLAY-1. Если <i>n</i> &gt; 32767:</p> <p>01 var.</p> <p style="padding-left: 20px;">02 var-LENGTH PIC S9(9) USAGE COMP.</p> <p style="padding-left: 20px;">02 var-DATA. 49 FILLER PIC G(32767) USAGE DISPLAY-1.</p> <p style="padding-left: 20px;">49 FILLER PIC G(32767). USAGE DISPLAY-1. ⋮ 49 FILLER PIC G(mod(<i>n</i>,32767)) USAGE DISPLAY-1.</p>
ROWID	<p>01 var.</p> <p style="padding-left: 20px;">49 var-LEN PIC 9(4) USAGE COMP.</p> <p style="padding-left: 20px;">49 var-DATA PIC X(40).</p>

*Таблица 32 (Стр. 1 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке PL/I*

<b>Тип данных SQL в определении</b>	<b>PL/I</b>
TABLE LOCATOR	BIN FIXED(31)
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	

Таблица 32 (Стр. 2 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке PL/I

Тип данных SQL в определении	PL/I
BLOB( <i>n</i> )	<pre> Если n &lt;= 32767: 01 var,  03 var_LENGTH     BIN FIXED(31),  03 var_DATA     CHAR(<i>n</i>); Если n &gt; 32767: 01 var,  02 var_LENGTH     BIN FIXED(31),  02 var_DATA,  03 var_DATA1(<i>n</i>)     CHAR(32767),  03 var_DATA2     CHAR(mod(<i>n</i>,32767)); </pre>
CLOB( <i>n</i> )	<pre> Если n &lt;= 32767: 01 var,  03 var_LENGTH     BIN FIXED(31),  03 var_DATA     CHAR(<i>n</i>); Если n &gt; 32767: 01 var,  02 var_LENGTH     BIN FIXED(31),  02 var_DATA,  03 var_DATA1(<i>n</i>)     CHAR(32767),  03 var_DATA2     CHAR(mod(<i>n</i>,32767)); </pre>
DBCLOB( <i>n</i> )	<pre> Если n &lt;= 16383: 01 var,  03 var_LENGTH     BIN FIXED(31),  03 var_DATA     GRAPHIC(<i>n</i>); Если n &gt; 16383: 01 var,  02 var_LENGTH     BIN FIXED(31),  02 var_DATA,  03 var_DATA1(<i>n</i>)     GRAPHIC(16383),  03 var_DATA2     GRAPHIC(mod(<i>n</i>,16383)); </pre>
ROWID	CHAR(40) VAR;

*Выходные параметры:* Задайте эти значения перед завершением работы пользовательской функции. Скалярная пользовательская функция возвращает один выходной параметр. Для табличных пользовательских функций число возвращаемых параметров равно числу столбцов в условии RETURNS TABLE оператора CREATE FUNCTION. DB2 отводит буфер для каждого значения параметра результата и передает пользовательской функции его адрес. DB2

выделяет буфер для значения каждого выходного параметра и передает его адрес пользовательской функции. Функция помещает значения выходных параметров в соответствующие буфера. Важно убедиться, что длина помещаемого в буфер значения не превышает длины этого буфера. Проверьте длину буфера по длине и типу данных SQL в операторе CREATE FUNCTION.

Раздел “Передача значений параметров в пользовательскую функцию и из пользовательской функции” на стр. 278 позволит определить, какие типы данных хоста использовать для выходных параметров. Если оператор CREATE FUNCTION содержит условие CAST FROM, используйте тип данных, соответствующий типу данных SQL в этом условии. В прочих случаях используйте тип данных, соответствующий типу данных SQL в условии RETURNS или RETURNS TABLE.

Чтобы повысить производительность табличной пользовательской функции, возвращающей много столбцов, можно передавать вызывающей программе значения только части столбцов. Например, табличная пользовательская функция по определению может возвращать 100 столбцов, но вызывающей программе требуются только два. Параметр DBINFO позволяет указать DB2, значения каких столбцов следует возвращать. После этого можно возвращать значения только этих столбцов. Указание требуемых столбцов с помощью параметра DBINFO описано ниже.

**Индикаторы входных параметров:** Это значения типа SMALLINT, которые DB2 задает перед передачей управления пользовательской функции. Позволяют установить, являются ли соответствующие входные параметры пустыми. Количество и порядок индикаторов соответствуют количеству и порядку входных параметров. Пользовательская функция получает каждый индикатор с одним из следующих значений:

**0** Значение входного параметра не пусто.

**отрицательное** Значение входного параметра пусто.

Предусмотрите в пользовательской функции проверку всех индикаторов, если функция не определена с параметром RETURNS NULL ON NULL INPUT. Функция, определенная с параметром RETURNS NULL ON NULL INPUT выполняется только если ни один входной параметр не имеет пустого значения.

**Индикаторы результата:** Это значения SMALLINT, которые нужно задать перед завершением пользовательской функции, чтобы сообщить вызывающей программе, какие параметры результата содержат пустые значения. У скалярных пользовательских функций один индикатор результата. Скалярная пользовательская функция имеет один индикатор результата. Табличная пользовательская функция имеет индикаторы результатов по числу выходных параметров. Порядок индикаторов результата совпадает с порядком выходных параметров. Присвойте каждому индикатору одно из следующих значений:

**0 или положительное** Соответствующий параметр не пуст.

**отрицательное** Параметр пуст.

**Значение SQLSTATE:** Это значение типа CHAR(5), которое должно быть установлено до завершения работы пользовательской функции. Функция может возвращать одно из следующих значений SQLSTATE:

- |              |   |
|--------------|---|
| <b>00000</b> | Это означает, что функция была выполнена без предупреждений и ошибок.   |
| <b>01Hxx</b> | Это означает, что функция обнаружила ненормальную ситуацию. xx – любые две однобайтных буквы или цифры. xx – любые два однобайтных алфавитно–цифровых символа. DB2 возвращает SQLCODE +462, если пользовательская функция присваивает SQLSTATE значение 01Hxx.  |
| <b>02000</b> | Это означает, что табличная пользовательская функция больше не может возвращать строки.   |
| <b>38ухх</b> | Это означает, что пользовательская функция обнаружила ошибку. у – любой однобайтный алфавитно–цифровой символ, кроме 5. xx – любые два однобайтных алфавитно–цифровых символа. Однако если оператор SQL в пользовательской функции возвращает одно из следующих значений SQLSTATE, рекомендуется передать это значение в вызвавшую программу: |
| <b>38001</b> | Пользовательская функция попыталась выполнить оператор SQL, хотя она определена с параметром NO SQL. При этом значении SQLSTATE DB2 возвращает SQLCODE –487.  |
| <b>38002</b> | Пользовательская функция попыталась выполнить оператор SQL, требующий, чтобы функция была определена с параметром MODIFIES SQL DATA, хотя она определена без этого параметра. При этом значении DB2 возвращает SQLCODE –577.  |
| <b>38003</b> | Пользовательская функция выполнила оператор COMMIT или ROLLBACK, что недопустимо для пользовательских функций. При этом значении DB2 возвращает SQLCODE –751.   |
| <b>38004</b> | Пользовательская функция попыталась выполнить оператор SQL, требующий, чтобы функция была определена с опцией READS SQL DATA или MODIFIES SQL DATA, хотя она определена без этих опций. При этом значении DB2 возвращает SQLCODE –579.  |

Когда пользовательская функция возвращает значение SQLSTATE 38ухх, не совпадающее с перечисленными выше, DB2 возвращает SQLCODE –443.

Если пользовательская функция возвращает значение SQLSTATE, недопустимое для пользовательской функции, DB2 заменяет его на 39001 и возвращает SQLCODE –463.

Если и пользовательская функция, и DB2 присваивают значение SQLSTATE, DB2 возвращает вызывающей программе свое значение SQLSTATE.

**имя пользовательской функции:** DB2 задает значение этого параметра в списке параметров перед началом работы пользовательской функции. Это значение типа VARCHAR(137): 8 байтов на имя схемы, 1 байт на точку и 128 байтов на имя пользовательской функции. Если вы реализуете несколько вариантов пользовательской функции с общим кодом, этот параметр может указывать, какой вариант функции требуется вызывающей программе.

**Особое имя:** DB2 задает значение этого параметра в списке параметров перед началом работы пользовательской функции. Это значение VARCHAR(128), которое представляет собой либо особое имя из оператора CREATE FUNCTION или особое имя, сгенерированное DB2. Если вы реализуете несколько вариантов пользовательской функции с общим кодом, этот параметр может указывать, какой вариант функции требуется вызывающей программе.

**Сообщение об ошибке:** Это значение типа VARCHAR(70), которое устанавливается пользовательской функцией перед окончанием ее работы. Оно используется для передачи вызывающей программе описания ошибки или предупреждения.

DB2 выделяет для этого значения 70-байтный буфер и передает его адрес функции в списке параметров. Убедитесь, что сообщение, записываемое в этот буфер, не длиннее 70 байтов. Не менее первых 17 байтов помещенного в этот буфер сообщения появляются в поле SQLERRMC SQLCA, возвращаемом вызывающей программе. Точное число байт зависит от числа других элементов SQLERRMC. Не используйте в диагностическом сообщении X'FF'. DB2 использует это значение как ограничитель элементов.

**Временная память:** Если определитель указал в операторе CREATE FUNCTION параметр SCRATCHPAD, DB2 выделяет буфер для области временной памяти и передает его адрес пользовательской функции. Перед первым вызовом функции в операторе SQL, DB2 устанавливает длину временной памяти в первых 4 байтах буфера, после чего заполняет область временной памяти X'00'. DB2 не инициализирует временную память между вызовами связанного подзапроса.

Проследите, чтобы пользовательская функция не записывала во временную память больше данных, чем это допускает ее длина.

**Тип вызова:** DB2 передает этот параметр скалярной пользовательской функции, если определитель указал в операторе CREATE FUNCTION параметр FINAL CALL. Табличной пользовательской функции этот параметр передается всегда.

При начале работы *скалярной пользовательской функции* параметр CALL TYPE имеет одно из следующих значений:

- 1** Это *первый вызов* пользовательской функции в операторе SQL. При первом вызове функции передаются все входные параметры. Кроме того, временная память, если она выделяется, принимает значение двоичных нулей.
- 0** Это *обычный вызов*. При обычном вызове функции передаются все входные параметры. Если передается временная память, DB2 не меняет ее значения.

- 1** Это **заключительный вызов**. При заключительном вызове функции не передаются входные параметры. Если передается временная память, DB2 не меняет ее значения.
- Заключительный вызов такого типа происходит, когда вызывающая программа производит явное закрытие указателя. Получив значение 1, функция может выполнять операторы SQL.
- 255** Это **заключительный вызов**. При заключительном вызове функции не передаются входные параметры. Если передается временная память, DB2 не меняет ее значения.
- Заключительный вызов такого типа происходит, когда вызывающая программа выполняет оператор COMMIT или ROLLBACK, или при аварийном завершении ее работы. Получив значение 255, пользовательская функция не может выполнять операторы SQL, за исключением CLOSE CURSOR. Если функция выполняет операторы CLOSE CURSOR при вызове такого типа, она не должна реагировать на SQLCODE –501, поскольку указатели могут быть уже закрыты DB2 до заключительного вызова.
- При первом вызове скалярная пользовательская функция должна получить все необходимые системные ресурсы. При заключительном вызове скалярная функция должна освободить ресурсы, полученные при первом вызове. Возвращать результаты скалярная функция должна только во время обычных вызовов. DB2 не учитывает результатов, возвращаемых при заключительном вызове. Впрочем, при заключительном вызове функция может задать значения SQLSTATE и сообщения об ошибке.
- Если вызывающий оператор SQL содержит несколько скалярных пользовательских функций, и одна из них возвращает значение SQLSTATE, сигнализирующее об ошибке, DB2 осуществляет заключительный вызов всех этих функций, а вызывающий оператор SQL получает значение SQLSTATE функции, обнаружившей ошибку.
- При начале работы *табличной пользовательской функции* параметр CALL TYPE имеет одно из следующих значений:
- 2** Это **первый вызов** пользовательской функции в операторе SQL. Первый вызов осуществляется только если функция определена с ключевым словом FINAL CALL. При первом вызове функции передаются все входные параметры. Кроме того, временная память, если она выделяется, принимает значение двоичных нулей.
  - 1** Это **начальный вызов** пользовательской функции в операторе SQL. Если в определении функции нет ключевого слова FINAL CALL, при начальном вызове функции передаются все входные параметры, а временная память, если она выделяется, принимает значение двоичных нулей. Если задано ключевое слово FINAL CALL, DB2 не изменяет значение временной памяти.
  - 0** Это **вызов выборки** пользовательской функции из оператора SQL. При вызове выборки функции передаются все входные параметры. Если передается временная память, DB2 не меняет ее значения.
  - 1** Это **конечный вызов**. При конечном вызове функции не передаются входные параметры. Если передается временная память, DB2 не меняет ее значения.

**2** Это заключительный вызов. Заключительный вызов такого типа происходит только если в определении пользовательской функции задан параметр FINAL CALL. При заключительном вызове функции не передаются входные параметры. Если передается временная память, DB2 не меняет ее значения.

Заключительный вызов такого типа происходит, когда вызывающая программа выполняет оператор CLOSE CURSOR.

**255** Это заключительный вызов. При заключительном вызове функции не передаются входные параметры. Если передается временная память, DB2 не меняет ее значения.

Заключительный вызов такого типа происходит, когда вызывающая программа выполняет оператор COMMIT или ROLLBACK, или при аварийном завершении ее работы. Получив значение 255, пользовательская функция не может выполнять операторы SQL, за исключением CLOSE CURSOR. Если функция выполняет операторы CLOSE CURSOR при вызове такого типа, она не должна реагировать на SQLCODE –501, поскольку указатели могут быть уже закрыты DB2 до заключительного вызова.

Если табличная пользовательская функция определена с параметром FINAL CALL, она должна выделять все необходимые ресурсы во время первого вызова и освобождать их при заключительном вызове со значением 2.

Если табличная пользовательская функция определена с параметром NO FINAL CALL, она должна выделять все необходимые ресурсы во время начального вызова и освобождать их при конечном вызове.

При вызове выборки табличная пользовательская функция должна возвращать строку. Если функция не может больше возвращать строки, она должна установить SQLSTATE значение 02000.

При конечном вызове табличная пользовательская функция может задавать значение SQLSTATE и сообщение об ошибке.

Если табличная пользовательская функция вызывается в подзапросе, она получает заключительный вызов CLOSE при каждом вызове подзапроса внутри запроса более высокого уровня, и затем начальный вызов OPEN при следующем вызове подзапроса внутри запроса более высокого уровня.

**DBINFO:** Если определитель задал в операторе CREATE FUNCTION параметр DBINFO, DB2 передает структуру DBINFO пользовательской функции. DBINFO содержит информацию о среде вызывающей функцию программы. Она содержит следующие поля, в том порядке, в котором они перечислены:

#### **Длина имени положения**

Поле типа двухбайтное целое без знака. Содержит длину имени положения в следующем поле.

#### **Имя положения**

Поле типа символ, 128 байтов. Содержит имя положения, с которым в настоящее время соединена вызывающая программа.

### **Длина ID авторизации**

Поле типа двухбайтное целое без знака. Содержит длину ID авторизации в следующем поле.

### **ID авторизации**

Поле типа символ, 128 байтов. Содержит ID авторизации программы, вызвавшей пользовательскую функцию. Свободное место справа заполнено пробелами. Если эта функция вложена в другие пользовательские функции, в этом поле содержится ID программы, вызвавшей пользовательскую функцию самого высокого уровня.

### **Кодовая страница подсистемы**

Структура из 48 байтов, состоящая из семи полей типа целое и 20 зарезервированных байтов. Эти поля содержат информацию о CCSID и схеме кодировки подсистемы, из которой вызвана функция. Семь полей типа целое содержат:

- EBCDIC SBCS CCSID
- EBCDIC MIXED CCSID
- EBCDIC DBCS CCSID
- ASCII SBCS CCSID
- ASCII MIXED CCSID
- ASCII DBCS CCSID
- Схему кодирования

### **Длина спецификатора таблицы**

Поле типа двухбайтное целое без знака. Содержит длину спецификатора таблицы в следующем поле. Если имя таблицы не используется, это поле имеет значение 0.

### **Спецификатор таблицы**

Поле типа символ, 128 байтов. Содержит спецификатор таблицы, указанной в поле Имя таблицы.

### **Длина имени таблицы**

Поле типа двухбайтное целое без знака. Содержит длину имени таблицы в следующем поле. Если имя таблицы не используется, это поле имеет значение 0.

### **Имя таблицы**

Поле типа символ, 128 байтов. Содержит имя таблицы, изменяемой операторами UPDATE или INSERT, если пользовательская функция вызывается из оператора SQL в следующих контекстах:

- В правой части условия SET оператора UPDATE
- В списке VALUES оператора INSERT

В прочих случаях это поле пусто.

### **Длина имени столбца**

Поле типа двухбайтное целое без знака. Содержит длину имени столбца в следующем поле. Если имя столбца не передается пользовательской функции, это поле имеет значение 0.

### **Имя столбца**

Поле типа символ, 128 байтов. Содержит имя столбца, изменяемого операторами UPDATE или INSERT, если пользовательская функция вызывается из оператора SQL в следующих контекстах:

- В правой части условия SET оператора UPDATE
- В списке VALUES оператора INSERT

В прочих случаях это поле пусто.

#### Информация о продукте

Поле типа символ, 8 байтов, характеризует продукт, при котором выполняется пользовательская функция. Данное поле имеет вид *pppvvrrm*, где:

- *ppp* – трехбайтный код продукта:

**DSN** DB2 for OS/390

**ARI** DB2 Server for VSE & VM

**QSQ** DB2 for AS/400®

**SQL** DB2 Universal Database

- *vv* идентификатор версии, 2 цифры.
- *rr* идентификатор выпуска, 2 цифры.
- *m* идентификатор уровня модификации, 1 цифра.

#### Операционная система

Поле типа целое, 4 байта. Характеризует операционную систему, в которой выполняется программа, вызвавшая эту пользовательскую функцию.

Может иметь следующие значения:

- |           |                    |
|-----------|--------------------|
| <b>0</b>  | Система неизвестна |
| <b>1</b>  | OS/2               |
| <b>3</b>  | Windows™           |
| <b>4</b>  | AIX                |
| <b>5</b>  | Windows NT™        |
| <b>6</b>  | HP-UX              |
| <b>7</b>  | Solaris            |
| <b>8</b>  | OS/390             |
| <b>13</b> | Siemens Nixdorf    |
| <b>15</b> | Windows 95         |
| <b>16</b> | SCO Unix           |

#### Количество записей в списке столбцов табличной функции

Поле типа двухбайтное целое без знака.

#### Зарезервированная область

24 байта.

#### Указатель на список столбцов табличной функции

Если определена табличная функция, данное поле является указателем на массив, содержащий 1000 двухбайтных целых. DB2 динамически размещает этот массив. Если табличная функция не определена, указатель пуст.

Только первые *n* записей, где *n* – значение поля Количество записей в списке столбцов табличной функции, являются важными. *n* больше или

равно 0 и не превышает количества столбцов результата, определенных для пользовательской функции в условии RETURNS TABLE оператора CREATE FUNCTION. Значения соответствуют номерам столбцов, которые табличная функция должна передать вызывающему оператору. Значение 1 обозначает первый определенный столбец результатов, 2 – второй, и т.д. Порядок значений может быть любым. Если *n* равно 0, первым элементом массива является 0. Это происходит в операторах, подобных следующему, когда не требуется ни одного столбца:

```
SELECT COUNT(*) FROM TABLE(TF(...)) AS QQ
```

Данный массив позволяет оптимизировать работу функции. Вместо того, чтобы возвращать все значения всех столбцов результата, функция может возвращать только столбцы, требуемые в данном контексте, задавая их номера в массиве. Вместо этого она может возвращать только столбцы, которые требуются в каждом конкретном случае и номера которых перечислены в массиве. Однако если такая оптимизация слишком усложняет пользовательскую функцию и не дает выигрыша в производительности, можно возвращать все столбцы.

### **Уникальный идентификатор программы**

Это поле – указатель на строку, служащую уникальным идентификатором соединения программы с DB2. Для каждого соединения с DB2 эта строка порождается заново.

Строка – LUWID, состоящий из полного сетевого имени LU, за которым следует точка и номер экземпляра логической единицы работы. Имя сети LU состоит из ID сети от 1 до 8 символов, точки и сетевого имени LU от 1 до 8 символов. Номер экземпляра LUW состоит из 12 шестнадцатеричных символов, которые уникально идентифицируют единицу работы.

### **Зарезервированная область**

20 байт.

Примеры передачи параметров на каждом языке приводятся в следующем разделе. Если вы пишете пользовательскую функцию на языке C или C++, можете воспользоваться объявлениями в SQLUDF DSN610.SDSNSAMP для многих передаваемых параметров. Чтобы включить SQLUDF, поместите в программу оператор:

```
#include <sqludf.h>
```

### **Примеры передачи параметров пользовательской функции**

Приведенные ниже примеры показывают, как пользовательские функции, написанные на каждом из поддерживаемых на хосте языков, принимают список параметров, передаваемый DB2.

В этих примерах предполагается, что пользовательская функция определена с параметрами SCRATCHPAD, FINAL CALL и DBINFO.

**Ассемблер:** На рис. 83 на стр. 292 показаны соглашения о параметрах для скалярной пользовательской функции, написанной в виде основной программы, которая принимает два параметра и возвращает один результат. Для функций, написанных как подпрограммы, в языке ассемблер передача параметров происходит аналогично. В обоих случаях необходимо включить макросы CEEENTRY и CEEEXIT.

```

MYMAIN    CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
          USING PROGAREA,R13

          L   R7,0(R1)           GET POINTER TO PARM1
          MVC PARM1(4),0(R7)     MOVE VALUE INTO LOCAL COPY OF PARM1
          L   R7,4(R1)           GET POINTER TO PARM2
          MVC PARM1(4),0(R7)     MOVE VALUE INTO LOCAL COPY OF PARM2
          L   R7,12(R1)          GET POINTER TO INDICATOR 1
          MVC F_IND1(2),0(R7)   MOVE PARM1 INDICATOR TO LOCAL STORAGE
          LH  R7,F_IND1         MOVE PARM1 INDICATOR INTO R7
          LTR R7,R7              CHECK IF IT IS NEGATIVE
          BM  NULLIN             ЕСЛИ ДА, PARM1 ПУСТ
          L   R7,16(R1)          GET POINTER TO INDICATOR 2
          MVC F_IND2(2),0(R7)   MOVE PARM2 INDICATOR TO LOCAL STORAGE
          LH  R7,F_IND2         MOVE PARM2 INDICATOR INTO R7
          LTR R7,R7              CHECK IF IT IS NEGATIVE
          BM  NULLIN             ЕСЛИ ДА, PARM2 ПУСТ

          ::

          L   R7,8(R1)           GET ADDRESS OF AREA FOR RESULT
NULLIN  MVC 0(9,R7),RESULT      MOVE A VALUE INTO RESULT AREA
          L   R7,20(R1)          GET ADDRESS OF AREA FOR RESULT IND
          MVC 0(2,R7),=H'0'      MOVE A VALUE INTO INDICATOR AREA

          ::

          CEETERM RC=0

*****
* Объявление переменных и равенства
*****
R1      EQU 1                  REGISTER 1
R7      EQU 7                  REGISTER 7
PPA     CEEPPA ,               CONSTANTS DESCRIBING THE CODE BLOCK
                           LTORG ,
                           PLACE LITERAL POOL HERE
PROGAREA DSECT
          ORG  ++CEEDSASZ      LEAVE SPACE FOR DSA FIXED PART
PARM1   DS   F                 PARAMETER 1
PARM2   DS   F                 PARAMETER 2
RESULT  DS   CL9               RESULT
F_IND1  DS   H                 INDICATOR FOR PARAMETER 1
F_IND2  DS   H                 INDICATOR FOR PARAMETER 2
F_INDR  DS   H                 INDICATOR FOR RESULT

PROGSIZE EQU  *-PROGAREA
          CEEDSA ,             MAPPING OF THE DYNAMIC SAVE AREA
          CEECAA ,             MAPPING OF THE COMMON ANCHOR AREA
          END     MYMAIN

```

*Рисунок 83. Как пользовательская функция на ассемблере принимает параметры*

*C или C++:*

В языках С и С++ передача параметров различается для функций, написанных как главные программы и как подпрограммы.

Подпрограммы получают параметры непосредственно. Для доступа к входным и выходным параметрам в главных программах используются стандартные переменные argc и argv:

- Переменная `argv` содержит набор указателей на параметры, передаваемые пользовательской функции. Каждый строковый параметр, возвращаемый DB2, должны заканчиваться символом—ограничителем `null` (двоичным нулем).
  - `argv[0]` содержит адрес имени модуля загрузки пользовательской функции.
  - `argv[1] – argv[n]` содержат адреса параметров с первого по *n*.
- Переменная `argc` содержит количество параметров, передаваемых внешней пользовательской функции, включая `argv[0]`.

Ниже приводится пример передачи параметров скалярной пользовательской функции, которая написана как главная программа, получает два параметра и возвращает один результат.

```

#include <stdlib.h>
#include <stdlib.h>

main(argc,argv)
    int argc;
    char *argv[];
{
    /*****
    /* Предположим, что пользователь вызвал функцию */
    /* с двумя входными параметрами в списке параметров.*/
    /* также, что в определении указаны опции */
    /* SCRATCHPAD, FINAL CALL и DBINFO, так что передаваемые */
    /* передает параметры временной памяти, типа */
    /* параметр dbinfo. */
    /* Массив argv содержит такие элементы: */
    /*     argv[0]      1   имя модуля загрузки */
    /*     argv[1-2]     2   входные параметры */
    /*     argv[3]      1   выходной параметр */
    /*     argv[4-5]     2   указатели вх.параметров */
    /*     argv[6]      1   null-индикатор результата*/
    /*     argv[7]      1   переменная SQLSTATE */
    /*     argv[8]      1   полное имя функции */
    /*     argv[9]      1   особое имя функции */
    /*     argv[10]     1   сообщение об ошибке */
    /*     argv[11]     1   временная память */
    /*     argv[12]     1   тип вызова */
    /*     argv[13]     + 1 dbinfo */
    /*----- */
    /*           14   в переменной argc */
    *****/
    if argc<>14
    {

    ::

    /*****
    /* Здесь должен содержаться текст, выполняемый при вызове */
    /* пользовательской функции с неправильным числом */
    /* параметров. */
    *****/
}

```

*Рисунок 84 (Часть 1 из 2). Как принимает параметры пользовательская функция на языке C или C++, написанная как основная программа*

```

/*****************/
/* Предположим, что тип первого параметра - целое. */
/* Ниже показано, как скопировать этот параметр */
/* в память программы. */
/*****************/
int parm1;
parm1 = *(int *) argv[1];

/*****************/
/* Проверьте индикатор первого параметра */
/* при вызове пользовательской функции */
/* следующим образом: */
/*****************/
short int ind1;
ind1 = *(short int *) argv[4];

/*****************/
/* Следующее выражение приписывает значение */
/* возвращаемому вызывающей программе для */
/* оператора SQL, из которого вызвана */
/* пользовательская функция */
/*****************/
strcpy(argv[7],"xxxxx/0");

/*****************/
/* Значение полного имени функции получается */
/* с помощью следующего выражения. */
/*****************/
char f_func[28];
strcpy(f_func,argv[8]);

/*****************/
/* Значение особого имени функции получается */
/* с помощью следующего выражения. */
/*****************/
char f_spec[19];
strcpy(f_spec,argv[9]);

/*****************/
/* Следующее выражение приписывает значение */
/* 'ууууууу' строке диагностики, возвращаемой */
/* в SQLCA, связанной с вызванной */
/* пользовательская функция */
/*****************/
strcpy(argv[10],"yyyyyyy/0");

/*****************/
/* Следующее выражение приписывает значение */
/* выходному параметру функции. */
/*****************/
char l_result[11];
strcpy(argv[3],l_result);

:
}

```

*Рисунок 84 (Часть 2 из 2). Как принимает параметры пользовательская функция на языке C или C++, написанная как основная программа*

На рис. 85 на стр. 296 показаны соглашения о параметрах для скалярной пользовательской функции на C, написанной как подпрограмма, которая получает два параметра и возвращает один результат.

```
#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdlib.h>
#include <string.h>

struct sqludf_scratchpad
{
    unsigned long length; /* длина временной памяти */
    char data[SQLUDF_SCRATCHPAD_LEN]; /* данные во временной памяти */
};

struct sqludf_dbinfo
{
    unsigned short dbnamelen; /* длина имени базы данных */
    unsigned char dbname[128]; /* имя базы данных */
    unsigned short authidlen; /* длина id авт. программы */
    unsigned char authid[128]; /* ID авт. программы */
    unsigned long ebcDIC_sbccs; /* EBCDIC SBCS CCSID */
    unsigned long ebcDIC_dbccs; /* EBCDIC MIXED CCSID */
    unsigned long ebcDIC_mixed; /* EBCDIC DBCS CCSID */
    unsigned long ascii_sbccs; /* ASCII SBCS CCSID */
    unsigned long ascii_dbccs; /* ASCII MIXED CCSID */
    unsigned long ascii_mixed; /* ASCII DBCS CCSID */
    unsigned long encode; /* схема кодировки UDF */
    unsigned char reserv0[20]; /* резерв для последующего использования*/
    unsigned short tbqualiflen; /* длина спецификатора таблицы */
    unsigned char tbqualif[128]; /* имя спецификатора таблицы */
    unsigned short tbnamelen; /* длина имени таблицы */
    unsigned char tbname[128]; /* имя таблицы */
    unsigned short colnamelen; /* длина имени столбца */
    unsigned char colname[128]; /* имя столбца */
    unsigned char relver[8]; /* Версия и выпуск базы данных */
    unsigned long platform; /* Платформа базы данных */
    unsigned short numtfcol; /* число столбцов табличной функции */
    unsigned char reserv1[24]; /* зарезервировано */
    unsigned short *tfcolnum; /* список столбцов табличной функции */
    unsigned short *appl_id; /* LUWID для соединения с DB2 */
    unsigned char reserv2[20]; /* зарезервировано */
};

void myfunc(long *parm1, char parm2[11], char result[11],
            short *f_ind1, short *f_ind2, short *f_indr,
            char udf_sqlstate[6], char udf_fname[138],
            char udf_specname[129], char udf_msgtext[71],
            struct sqludf_scratchpad *udf_scratchpad,
            long *udf_call_type,
            struct sql_dbinfo *udf_dbinfo);
```

Рисунок 85 (Часть 1 из 2). Как принимает параметры пользовательская программа на языке C, написанная как подпрограмма

```

{
/* **** */
/* объявление локальных копий параметров */
/* **** */
int l_p1;
char l_p2[11];
short int l_ind1;
short int l_ind2;
char ludf_sqlstate[6];      /* SQLSTATE */
char ludf_fname[138];       /* имя функции */
char ludf_specname[129];    /* особое имя функции */
char ludf_msgtext[71]        /* текст диагностического сообщения */
sqludf_scratchpad *ludf_scratchpad; /* временная память */
long *ludf_call_type;        /* тип вызова */
sqludf_dbinfo *ludf_dbinfo /* dbinfo */
/* **** */
/* Каждый параметр из списка параметров */
/* копируется в локальную переменную, чтобы */
/* показать обращение к этим параметрам */
/* **** */

l_p1 = *parm1;
strcpy(l_p2,parm2);
l_ind1 = *f_ind1;
l_ind1 = *f_ind2;
strcpy(ludf_sqlstate,udf_sqlstate);
strcpy(ludf_fname,udf_fname);
strcpy(ludf_specname,udf_specname);
l_udf_call_type = *udf_call_type;
strcpy(ludf_msgtext,udf_msgtext);
memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));

:
}

```

*Рисунок 85 (Часть 2 из 2). Как принимает параметры пользовательская программа на языке C, написанная как подпрограмма*

Ниже приводится пример передачи параметров скалярной пользовательской функции, написанной как подпрограмма на языке C++, которая получает два параметра и возвращает один результат. В примере указывается модификатор `extern "C"`, чтобы использовались соглашения С о передаче параметров. Модификатор необходим, поскольку интерфейс CEEPIPI `CALL_SUB`, используемый DB2 для вызова пользовательских функций, при передаче параметров использует обозначения связей С.

```

#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdlib.h>
struct sqludf_scratchpad
{
    unsigned long length; /* длина временной памяти */
    char data[SQLUDF_SCRATCHPAD_LEN]; /* данные во временной памяти */
};
struct sqludf_dbinfo
{
    unsigned short dbnameelen; /* длина имени базы данных */
    unsigned char dbname[128]; /* имя базы данных */
    unsigned short authidlen; /* длина id авт. программы */
    unsigned char authid[128]; /* ID авт. программы */
    unsigned long ebcdic_sbcs; /* EBCDIC SBCS CCSID */
    unsigned long ebcdic_dbcs; /* EBCDIC MIXED CCSID */
    unsigned long ebcdic_mixed; /* EBCDIC DBCS CCSID */
    unsigned long ascii_sbcs; /* ASCII SBCS CCSID */
    unsigned long ascii_dbcs; /* ASCII MIXED CCSID */
    unsigned long ascii_mixed; /* ASCII DBCS CCSID */
    unsigned long encode; /* схема кодировки UDF */
    unsigned char reserv0[20]; /* резерв для последующего использования*/
    unsigned short tbqualiflen; /* длина спецификатора таблицы */
    unsigned char tbqualif[128]; /* имя спецификатора таблицы */
    unsigned short tbnamelen; /* длина имени таблицы */
    unsigned char tbname[128]; /* имя таблицы */
    unsigned short colnamelen; /* длина имени столбца */
    unsigned char colname[128]; /* имя столбца */
    unsigned char relver[8]; /* Версия и выпуск базы данных */
    unsigned long platform; /* Платформа базы данных */
    unsigned short numtfcol; /* число столбцов табличной функции */
    unsigned char reserv1[24]; /* зарезервировано */
    unsigned short *tfcolnum; /* список столбцов табличной функции */
    unsigned short *appl_id; /* LUWID для соединения с DB2 */
    unsigned char reserv2[20]; /* зарезервировано */
};

extern "C" void myfunc(long *parm1, char parm2[11],
                      char result[11], short *f_ind1, short *f_ind2, short *f_indr,
                      char udf_sqlstate[6], char udf_fname[138],
                      char udf_specname[129], char udf_msgrtext[71],
                      struct sqludf_scratchpad *udf_scratchpad,
                      long *udf_call_type,
                      struct sql_dbinfo *udf_dbinfo);

```

*Рисунок 86 (Часть 1 из 2). Как принимает параметры пользовательская функция на языке C++, написанная как подпрограмма*

```

{
/* **** */
/* Определение локальных копий параметров. */
/* **** */
int l_p1;
char l_p2[11];
short int l_ind1;
short int l_ind2;
char ludf_sqlstate[6];      /* SQLSTATE           */
char ludf_fname[138];       /* имя функции        */
char ludf_specname[129];    /* особое имя функции */
char ludf_msgtext[71]        /* текст диагностического сообщения */
sqludf_scratchpad *ludf_scratchpad; /* временная память   */
long *ludf_call_type;        /* тип вызова          */
sqludf_dbinfo *ludf_dbinfo /* dbinfo             */
/* **** */
/* Каждый параметр из списка параметров */
/* копируется в локальную переменную, чтобы */
/* показать обращение к этим параметрам */
/* **** */
l_p1 = *parm1;
strcpy(l_p2,parm2);
l_ind1 = *f_ind1;
l_ind1 = *f_ind2;
strcpy(ludf_sqlstate,udf_sqlstate);
strcpy(ludf_fname,udf_fname);
strcpy(ludf_specname,udf_specname);
l_udf_call_type = *udf_call_type;
strcpy(ludf_msgtext,udf_msgtext);
memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));

:
}

```

*Рисунок 86 (Часть 2 из 2). Как принимает параметры пользовательская функция на языке C++, написанная как подпрограмма*

*COBOL*: рис. 87 на стр. 300 показывает соглашения о параметрах для табличной пользовательской функции, оформленной в виде основной программы, которая получает два параметра и возвращает два результата. Для пользовательской функции, написанной на языке COBOL в виде подпрограммы, соглашения те же.

```

CBL APOST,RES,RENT
IDENTIFICATION DIVISION.

:::
DATA DIVISION.

:::
LINKAGE SECTION.

*****
* объявление входных параметров *
*****
01 UDFPARM1 PIC S9(9) USAGE COMP.
01 UDFPARM2 PIC X(10).

:::
*****
* объявление переменных для выходных параметров *
*****
01 UDFRESULT1 PIC X(10).
01 UDFRESULT2 PIC X(10).

:::
*****
* объявление индикатора для входных параметров *
*****
01 UDF-IND1 PIC S9(4) USAGE COMP.
01 UDF-IND2 PIC S9(4) USAGE COMP.

:::
*****
* объявление индикатора для выходных параметров *
*****
01 UDF-RIND1 PIC S9(4) USAGE COMP.
01 UDF-RIND2 PIC S9(4) USAGE COMP.

:::
*****
* Объявить SQLSTATE, которое может быть задано в      *
* пользовательской функции                           *
*****
01 UDF-SQLSTATE PIC X(5).

*****
* объявление полного имени функции                  *
*****
01 UDF-FUNC.
49 UDF-FUNC-LEN PIC 9(4) USAGE BINARY.
49 UDF-FUNC-TEXT PIC X(137).

*****
* объявление особого имени функции                 *
*****
01 UDF-SPEC.
49 UDF-SPEC-LEN PIC 9(4) USAGE BINARY.
49 UDF-SPEC-TEXT PIC X(128).

```

*Рисунок 87 (Часть 1 из 3). Как принимает параметры пользовательская функция на языке COBOL*

```

*****
* Объявить маркер диагностического сообщения SQL      *
*****
01 UDF-DIAG.
  49 UDF-DIAG-LEN PIC 9(4) USAGE BINARY.
  49 UDF-DIAG-TEXT PIC X(70).
*****
* объявление временной памяти      *
*****
01 UDF-SCRATCHPAD.
  49 UDF-SPAD-LEN PIC 9(9) USAGE BINARY.
  49 UDF-SPAD-TEXT PIC X(100).
*****
* объявление типов вызова      *
*****
01 UDF-CALL-TYPE PIC 9(9) USAGE BINARY.
*****
* объявление структуры DBINFO
*****
01 UDF-DBINFO.
*   Имя положения и его длина
  02 UDF-DBINFO-LOCATION.
    49 UDF-DBINFO-LLEN PIC 9(4) USAGE BINARY.
    49 UDF-DBINFO-LOC PIC X(128).
*   ID авторизации и его длина
  02 UDF-DBINFO-AUTHORIZATION.
    49 UDF-DBINFO-ALEN PIC 9(4) USAGE BINARY.
    49 UDF-DBINFO-AUTH PIC X(128).
*   CCSIDs for DB2 for OS/390
  02 UDF-DBINFO-CCSID PIC X(48).
  02 UDF-DBINFO-CCSID-REDEFINE REDEFINES UDF-DBINFO-CCSID.
    03 UDF-DBINFO-ESBCS  PIC 9(9) USAGE BINARY.
    03 UDF-DBINFO-EMIXED PIC 9(9) USAGE BINARY.
    03 UDF-DBINFO-EDBCS  PIC 9(9) USAGE BINARY.
    03 UDF-DBINFO-ASBCS  PIC 9(9) USAGE BINARY.
    03 UDF-DBINFO-AMIXED PIC 9(9) USAGE BINARY.
    03 UDF-DBINFO-ADBCS  PIC 9(9) USAGE BINARY.
    03 UDF-DBINFO-ENCODE PIC 9(9) USAGE BINARY.
    03 UDF-DBINFO-RESERVO PIC X(20).
*   Имя схемы и его длина
  02 UDF-DBINFO-SCHEMA0.
    49 UDF-DBINFO-SLEN PIC 9(4) USAGE BINARY.
    49 UDF-DBINFO-SCHEMA PIC X(128).
*   Имя таблицы и его длина
  02 UDF-DBINFO-TABLE0.
    49 UDF-DBINFO-TLEN PIC 9(4) USAGE BINARY.
    49 UDF-DBINFO-TABLE  PIC X(128).
*   Имя столбца и его длина
  02 UDF-DBINFO-column0.
    49 UDF-DBINFO-CLEN PIC 9(4) USAGE BINARY.
    49 UDF-DBINFO-COLUMN  PIC X(128).
*   Уровень выпуска DB2
  02 UDF-DBINFO-VERREL PIC X(8).

```

*Рисунок 87 (Часть 2 из 3). Как принимает параметры пользовательская функция на языке COBOL*

```

*      Не используется
02 FILLER          PIC X(2).
*
*      Платформа базы данных
02 UDF-DBINFO-PLATFORM PIC 9(9) USAGE BINARY.
*
*      число элементов в списке столбцов табличной функции
02 UDF-DBINFO-NUMTFCOL PIC 9(4) USAGE BINARY.
*
*      зарезервировано
02 UDF-DBINFO-RESERV1 PIC X(24).
*
*      Не используется
02 FILLER          PIC X(2).
*
*      Указатель на список столбцов табличной функции
02 UDF-DBINFO-TFCOLUMN PIC 9(9) USAGE BINARY.
*
*      Указатель на ID программы
02 UDF-DBINFO-APPLID PIC 9(9) USAGE BINARY.
*
*      зарезервировано
02 UDF-DBINFO-RESERV2 PIC X(20).

*
PROCEDURE DIVISION USING UDFPARM1, UDFPARM2, UDFRESULT1,
                     UDFRESULT2, UDF-IND1, UDF-IND2,
                     UDF-RIND1, UDF-RIND2,
                     UDF-SQLSTATE, UDF-FUNC, UDF-SPEC,
                     UDF-DIAG, UDF-SCRATCHPAD,
                     UDF-CALL-TYPE, UDF-DBINFO.

```

*Рисунок 87 (Часть 3 из 3). Как принимает параметры пользовательская функция на языке COBOL*

*PL/I*: рис. 88 на стр. 303 показывает соглашениях о параметрах для скалярной пользовательской функции, написанной как основная программа, которая получает два параметра и возвращает один результат. Для пользовательских функций в виде подпрограммы на языке PL/I соглашения те же.

```

*PROCESS SYSTEM(MVS);
MYMAIN: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
            UDF_IND1, UDF_IND2, UDF_INDR,
            UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
            UDF_DIAG_MSG, UDF_SCRATCHPAD,
            UDF_CALL_TYPE, UDF_DBINFO)
            OPTIONS(MAIN NOEXECOPS REENTRANT);

DCL UDF_PARM1 BIN FIXED(31); /* первый параметр */
DCL UDF_PARM2 CHAR(10); /* второй параметр */
DCL UDF_RESULT CHAR(10); /* выходной параметр */
DCL UDF_IND1 BIN FIXED(15); /* индикатор 1-го пар. */
DCL UDF_IND2 BIN FIXED(15); /* индикатор 2-го пар. */
DCL UDF_INDR BIN FIXED(15); /* индикатор результата */
DCL UDF_SQLSTATE CHAR(5); /* SQLSTATE, возвращаемое DB2 */
DCL UDF_NAME CHAR(137) VARYING; /* Полное имя функции */
DCL UDF_SPEC_NAME CHAR(128) VARYING; /* Особое имя функции */
DCL UDF_DIAG_MSG CHAR(70) VARYING; /* Сообщение об ошибке */
DCL 01 UDF_SCRATCHPAD /* Временная память */
      03 UDF_SPAD_LEN BIN FIXED(31),
      03 UDF_SPAD_TEXT CHAR(100);
DCL UDF_CALL_TYPE BIN FIXED(31); /* Тип вызова */
DCL DBINFO PTR;
DCL 01 UDF_DBINFO BASED(DBINFO),
      03 UDF_DBINFO_LLEN BIN FIXED(15), /* длина положения */
      03 UDF_DBINFO_LOC CHAR(128), /* имя положения */
      03 UDF_DBINFO_ALEN BIN FIXED(15), /* длина ID авт. */
      03 UDF_DBINFO_AUTH CHAR(128), /* ID авторизации */
      03 UDF_DBINFO_CCSID, /* CCSIDs for DB2 for OS/390*/
      05 R1 BIN FIXED(15), /* Зарезервировано */
      05 UDF_DBINFO_ESBCS BIN FIXED(15), /* EBCDIC SBCS CCSID */
      05 R2 BIN FIXED(15), /* Зарезервировано */
      05 UDF_DBINFO_EMIXED BIN FIXED(15), /* EBCDIC MIXED CCSID */
      05 R3 BIN FIXED(15), /* Зарезервировано */
      05 UDF_DBINFO_EDBCS BIN FIXED(15), /* EBCDIC DBCS CCSID */
      05 R4 BIN FIXED(15), /* Зарезервировано */
      05 UDF_DBINFO_ASBCS BIN FIXED(15), /* ASCII SBCS CCSID */
      05 R5 BIN FIXED(15), /* Зарезервировано */
      05 UDF_DBINFO_AMIXED BIN FIXED(15), /* ASCII MIXED CCSID */
      05 R6 BIN FIXED(15), /* Зарезервировано */
      05 UDF_DBINFO_ADBCS BIN FIXED(15), /* ASCII DBCS CCSID */
      05 UDF_DBINFO_ENCODE BIN FIXED(31), /* схема кодировки UDF */
      05 UDF_DBINFO_RESERVO CHAR(20), /* зарезервировано */

```

Рисунок 88 (Часть 1 из 2). Как принимает параметры пользовательская функция на языке PL/I

```

03 UDF_DBINFO_SLEN BIN FIXED(15),      /* длина схемы      */
03 UDF_DBINFO_SCHEMA CHAR(128),        /* имя схемы        */
03 UDF_DBINFO_TLEN BIN FIXED(15),      /* длина таблицы    */
03 UDF_DBINFO_TABLE  CHAR(128),        /* имя таблицы      */
03 UDF_DBINFO_CLEN BIN FIXED(15),      /* длина столбца   */
03 UDF_DBINFO_COLUMN CHAR(128),        /* имя столбца     */
03 UDF_DBINFO_RELVER CHAR(8),          /* версия и выпуск DB2 */
03 UDF_DBINFO_PLATFORM BIN FIXED(31),  /* платформа базы данных */
03 UDF_DBINFO_NUMTFCOL BIN FIXED(15),  /* число столбцов табличной функции */
03 UDF_DBINFO_RESERV1 CHAR(24),        /* зарезервировано */
03 UDF_DBINFO_TFCOLUMN PTR,           /* --> список столбцов табличной функции */
03 UDF_DBINFO_APPLID  PTR,           /* --> id программы */
03 UDF_DBINFO_RESERV2 CHAR(20);       /* Зарезервировано */

```

:

*Рисунок 88 (Часть 2 из 2). Как принимает параметры пользовательская функция на языке PL/I*

## Использование специальных регистров в пользовательской функции

В пользовательской функции можно использовать все специальные регистры. Однако изменять можно только некоторые из них. После завершения работы пользовательской функции DB2 восстанавливает предшествовавшие вызову функции значения всех специальных регистров.

Табл. 33 содержит информацию, нужную для использования в пользовательской функции специальных регистров.

*Таблица 33 (Стр. 1 из 2). Характеристики специальных регистров в пользовательской функции*

Специальный регистр	Исходное значение	Можно ли изменять функцию оператором SET ?
CURRENT DATE	Новое значение для каждого оператора SQL в пакете пользовательской функции <sup>1</sup>	Не применимо <sup>4</sup>
CURRENT DEGREE	Наследуется из вызывающей программы <sup>2</sup>	Да
CURRENT LOCALE LC_TYPE	Наследуется из вызывающей программы	Да
CURRENT OPTIMIZATION HINT	Значение опции связывания OPTHINT пакета пользовательской функции или наследуется из вызывающей программы <sup>5</sup>	Да
CURRENT PACKAGESET	Наследуется из вызывающей программы <sup>3</sup>	Да
CURRENT PATH	Значение опции связывания PATH пакета пользовательской функции или наследуется из вызывающей программы <sup>5</sup>	Да

Таблица 33 (Стр. 2 из 2). Характеристики специальных регистров в пользовательской функции

Специальный регистр	Исходное значение	Можно ли изменять функцию оператором SET ?
CURRENT PRECISION	Наследуется из вызывающей программы	Да
CURRENT RULES	Наследуется из вызывающей программы	Да
CURRENT SERVER	Наследуется из вызывающей программы	Да
CURRENT SQLID	Исходный ID авторизации прикладного процесса или наследуется из вызывающей программы <sup>6</sup>	Да <sup>7</sup>
CURRENT TIME	Новое значение для каждого оператора SQL в пакете пользовательской функции <sup>1</sup>	Не применимо <sup>4</sup>
CURRENT TIMESTAMP	Новое значение для каждого оператора SQL в пакете пользовательской функции <sup>1</sup>	Не применимо <sup>4</sup>
CURRENT TIMEZONE	Наследуется из вызывающей программы	Не применимо <sup>4</sup>
CURRENT USER	Исходный ID авторизации прикладного процесса	Не применимо <sup>4</sup>

Примечания к Табл. 33 на стр. 304:

- Если функция вызывается из области действия триггера, DB2 использует временную отметку оператора SQL триггера в качестве временной отметки для всех операторов SQL в пакете функции.
- DB2 разрешает параллелизм только на одном уровне вложенного оператора SQL. Если специальный регистр CURRENT DEGREE имеет значение ANY, а параллелизм запрещен, DB2 не учитывает значения CURRENT DEGREE.
- Если определитель пользовательской функции указал значение COLLID в операторе CREATE FUNCTION, DB2 присваивает это значение регистру CURRENT PACKAGESET.
- Неприменимо, поскольку для данного специального регистра не существует оператора SET.
- Если программа в области действия вызывающей программы отдает команду SET для специального регистра до вызова пользовательской функции, специальный регистр наследует значение из этого оператора SET. В других случаях регистр содержит значение, присвоенное при связывании пакета пользовательской функции.
- Если программа в области действия вызывающей программы отдает команду SET CURRENT SQLID до вызова пользовательской функции, специальный регистр наследует значение из этого оператора SET. В

других случаях CURRENT SQLID содержит ID авторизации прикладного процесса.

7. Если в пакете пользовательской функции используется значение, отличное от значения RUN параметра связывания DYNAMICRULES, оператор SET CURRENT SQLID может быть выполнен, но не влияет на ID авторизации, используемый для динамических операторов SQL в пакете пользовательской функции. Значение DYNAMICRULES определяет ID авторизации, используемый для динамических операторов SQL. Дополнительные сведения о значениях DYNAMICRULES и ID авторизации смотрите в разделе “Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES” на стр. 457.

## **Использование временной памяти в пользовательских функциях**

Временная память позволяет сохранять информацию, полученную при предыдущих вызовах пользовательской функции. Чтобы указать, что при выполнении функции должна быть выделена временная память, определитель функции указывает в операторе CREATE FUNCTION параметр SCRATCHPAD.

Временная память состоит из четырехбайтного поля длины, за которым следует область временной памяти. Задать длину области временной памяти можно в операторе CREATE FUNCTION. Заданное значение не включает поля длины. Размер по умолчанию – 100 байтов. DB2 инициализирует временную память каждой функции, заполняя ее двоичными нулями в начале выполнения каждого подзапроса оператора SQL, после чего не рассматривает и не изменяет ее содержимое. При каждом вызове пользовательской функции DB2 передает временную память пользовательской функции. Поэтому временная память может использоваться для хранения информации, оставшейся от предыдущих вызовов повторно–входной функции.

пример ввода информации во временную память пользовательской функции, определенной следующим образом:

```
CREATE FUNCTION COUNTER()
  RETURNS INT
  SCRATCHPAD
  FENCED
  NOT DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'UDFCTR';
```

Длина временной памяти не указана, поэтому временная память имеет длину по умолчанию, 100 байтов, плюс 4 байта на поле длины. Пользовательская функция увеличивает значение типа целое и сохраняет во временной памяти при каждом выполнении.

```

#pragma linkage(ctr,fetchable)
#include <stdlib.h>
#include <stdlib.h>
/* Структура scr определяет временную память, передаваемую функции ctr */
struct scr {
    long len;
    long countr;
    char not_used[96];
};
/*****************************************/
/* Функция ctr: Увеличивает счетчик на 1 и возвращает значение */
/* из временной памяти. */
/*
/* Вход:
/* Выход: INTEGER out      значение из временной памяти */
/*****************************************/
void ctr(
    long *out,                      /* Выходной параметр (счетчик)*/
    short *outnull,                 /* Индикатор результата */
    char *sqlstate,                 /* SQLSTATE */
    char *funcname,                 /* имя функции */
    char *specname,                 /* Особое имя функции */
    char *mesgtext,                  /* Сообщение об ошибке */
    struct scr *scratchptr)         /* Временная память */
{
    *out = ++scratchptr->countr;   /* Счетчик увеличивается и */
    /* результат копируется */
    /* в выходную переменную */
    /* Задать пустой выходной индикатор*/
    *outnull = 0;
    return;
}
/* конец пользовательской функции ctr */

```

Рисунок 89. Пример использования временной памяти в пользовательской функции

## **Доступ к таблицам переходов в пользовательских функциях**

При написании пользовательской функции, вызываемой триггером, может понадобиться доступ к таблицам переходов триггера. Таблицы переходов – это наборы строк, которые изменяет оператор SQL триггера. Таблицы переходов описаны в разделе “Составные части триггера” на стр. 235. Доступ к таблицам переходов в пользовательской функции осуществляется с помощью локаторов таблиц, которые являются указателями на таблицы переходов. Локаторы таблиц объявляются в качестве входных параметров в операторе CREATE FUNCTION с помощью условия TABLE Имя–таблицы AS LOCATOR. Глава 6 книги *DB2 SQL Reference* содержит более полную информацию.

Доступ к таблицам переходов в пользовательской функции происходит в пять основных этапов:

1. Получение локаторов таблиц через объявление входных параметров. Каждый параметр, в который будет помещен локатор таблицы, нужно определить как четырехбайтное целое без знака.
2. Объявление локаторов таблиц. Локаторы таблиц можно объявить на языках ассемблер, C, C++, COBOL и PL/I. Синтаксис объявления локаторов

в этих языках описан в разделе “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145.

3. Объявление указателя для доступа к строкам в полученных таблицах переходов.
4. Присвоение значений входных параметров локаторам таблиц.
5. Доступ к строкам таблиц переходов с помощью указателей, объявленных для этих таблиц.

Следующие примеры показывают, как пользовательские функции на каждом из поддерживаемых языков программирования обращаются к таблице переходов для триггера. Таблица переходов, NEWEMP, содержит измененные строки таблицы примера EMP. Триггер определен следующим образом:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW_TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES (CHECKEMP(TABLE NEWEMPS));
  END;
```

Пользовательская функция определена следующим образом:

```
CREATE FUNCTION CHECKEMP(TABLE LIKE EMP AS LOCATOR)
  RETURNS INTEGER
  EXTERNAL NAME 'CHECKEMP'
  PARAMETER STYLE DB2SQL
  LANGUAGE язык;
```

**Ассемблер:** На рис. 90 на стр. 309 – пример обращения программы на ассемблере к строкам таблицы переходов NEWEMPS.

```

CHECKEMP CSECT
    SAVE (14,12)           ЛЮБАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ СОХРАНЕНИЯ
    LR   R12,R15            CODE ADDRESSABILITY
    USING CHECKEMP,R12      СООБЩИТЬ АССЕМБЛЕРУ
    LR   R7,R1               СОХРАНИТЬ УКАЗАТЕЛЬ НА ПАРАМЕТРЫ
    USING PARMAREA,R7       ЗАДАТЬ АДРЕСУЕМОСТЬ ПАРАМЕТРОВ
    USING SQLDSECT,R8       ЗАДАТЬ АДРЕСУЕМОСТЬ SQLDSECT
    L    R6,PROGSIZE         ПОЛУЧИТЬ ПАМЯТЬ ДЛЯ ПРОГР. ПОЛЬЗ.
    GETMAIN R,LV=(6)         ПОЛУЧИТЬ ПАМЯТЬ ДЛЯ ПЕРЕМ. ПРОГР.
    LR   R10,R1              УКАЗАТЬ НА ПОЛУЧЕННУЮ ПАМЯТЬ
    LR   R2,R10             УКАЗАТЬ НА ПОЛЕ
    LR   R3,R6              ПОЛУЧИТЬ ЕГО ДЛИНУ
    SR   R4,R4              ОЧИСТИТЬ ВХОДНОЙ АДРЕС
    SR   R5,R5              ОЧИСТИТЬ ВХОДНУЮ ДЛИНУ
    MVCL R2,R4              ОЧИСТИТЬ ПОЛЕ
    ST    R13,FOUR(R10)     ЦЕПОЧКА УКАЗАТЕЛЕЙ
    ST    R10,EIGHT(R13)    УКАЗАТЕЛЬ SAVEAREA ВПЕРЕД
    LR    R13,R10            УКАЗАТЬ НА SAVEAREA
    USING PROGAREA,R13      ЗАДАТЬ АДРЕСУЕМОСТЬ
    ST    R6,GETLENGTH      СОХРАНИТЬ ДЛИНУ GETMAIN

    :

*****
* объявление локатора таблицы - переменной хоста TRIGTBL *
*****
TRIGTBL SQL TYPE IS TABLE LIKE EMP AS LOCATOR
*****
* объявление указателя, чтобы получить строки из таблицы *
* переходов
*****
EXEC SQL DECLARE C1 CURSOR FOR          X
    SELECT LASTNAME FROM TABLE(:TRIGTBL LIKE EMP)
    WHERE SALARY > 100000
*****
* Копирование локатора таблицы для таблицы переходов триггера *
*****
    L    R2,TABLOC          GET ADDRESS OF LOCATOR
    L    R2,0(0,R2)          GET LOCATOR VALUE
    ST   R2,TRIGTBL
    EXEC SQL OPEN C1
    EXEC SQL FETCH C1 INTO :NAME

    :
    EXEC SQL CLOSE C1
    :

```

*Рисунок 90 (Часть 1 из 2). Доступ пользовательской функции на ассемблере к таблице переходов*

```

PROGAREA DSECT           WORKING STORAGE FOR THE PROGRAM
SAVEAREA DS    18F        THIS ROUTINE'S SAVE AREA
GETLENGTH DS   A         GETMAIN LENGTH FOR THIS AREA
:
NAME    DS    CL24
:
DS    0D
PROGSIZE EQU  *-PROGAREA      DYNAMIC WORKAREA SIZE
PARMAREA DSECT
TABLOC  DS   A            INPUT PARAMETER FOR TABLE LOCATOR
:
END    CHECKEMP

```

*Рисунок 90 (Часть 2 из 2). Доступ пользовательской функции на ассемблере к таблице переходов*

**C или C<sup>++</sup>:** рис. 91 показывает, как программа на языке C или C<sup>++</sup> обращается к таблице переходов NEWEMPS.

```

int CHECK_EMP(int trig_tbl_id)
{
:
/*
 * Объявить переменную хоста локатора таблицы trig_tbl_id */
EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS TABLE LIKE EMP AS LOCATOR trig_tbl_id;
    char name[25];
EXEC SQL END DECLARE SECTION;

:
/*
 * объявление указателя для получения строк из таблицы */
/* переходов */
EXEC SQL DECLARE C1 CURSOR FOR
    SELECT NAME FROM TABLE(:trig_tbl_id LIKE EMPLOYEE)
    WHERE SALARY > 100000;
/*
 * Выборка строки из таблицы переходов */
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 INTO :name;

:
EXEC SQL CLOSE C1;

:
}

```

*Рисунок 91. Как пользовательская функция на языке C или C<sup>++</sup> обращается к таблице переходов*

**COBOL:** рис. 92 на стр. 311 показывает, как программа на языке COBOL обращается к таблице переходов NEWEMPS.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CHECKEMP.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*****
* объявление локатора таблицы - переменной хоста TRIG-TBL-ID *
*****
01 TRIG-TBL-ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR.
01 NAME PIC X(24).

:
LINKAGE SECTION.

:
PROCEDURE DIVISION USING TRIG-TBL-ID.

:
*****
* Объявить указатель для получения строк из таблицы      *
* переходов                                              *
*****
EXEC SQL DECLARE C1 CURSOR FOR
SELECT NAME FROM TABLE(:TRIG-TBL-ID LIKE EMP)
WHERE SALARY > 100000 END-EXEC.
*****
* Выборка строки из таблицы переходов                  *
*****
EXEC SQL OPEN C1 END-EXEC.
EXEC SQL FETCH C1 INTO :NAME END-EXEC.

:
EXEC SQL CLOSE C1 END-EXEC.

:
PROG-END.
GOBACK.

```

*Рисунок 92. Как пользовательская функция на языке COBOL обращается к таблице переходов*

**PL/I:** рис. 93 на стр. 312 показывает, как программа на языке PL/I обращается к таблице переходов NEWEMPS.

```

CHECK_EMP: PROC(TRIG_TBL_ID) RETURNS(BIN FIXED(31))
           OPTIONS(MAIN NOEXECOPS REENTRANT);
/*****
/* объявление локатора таблицы – переменной хоста TRIG_TBL_ID */
/*****
DECLARE TRIG_TBL_ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR;
DECLARE NAME CHAR(24);

:
/*****
/* Объявить указатель для получения строк из */
/* таблицы переходов */
/*****
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT NAME FROM TABLE(:TRIG_TBL_ID LIKE EMP)
  WHERE SALARY > 100000;
/*****
/* Получение строк из таблицы переходов */
/*****
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 INTO :NAME;

:
EXEC SQL CLOSE C1;

:
END CHECK_EMP;

```

*Рисунок 93. Как пользовательская функция на языке PL/I обращается к таблице переходов*

## Подготовка пользовательской функции к выполнению

Чтобы подготовить пользовательскую функцию к выполнению, надо выполнить следующие действия:

1. Прекомпилировать программу пользовательской функции и связать DBRM в пакет.

Этого можно не делать, если пользовательская функция не содержит операторов SQL. Для такой функции не обязательно производить связывание плана.

2. Скомпилировать программу пользовательской функции и скомпоновать ее с помощью языковая среда и RRSAF.

Необходимо скомпилировать программу компилятором, который поддерживает языковая среда, и добавить при построении пользовательской функции нужные компоненты языковая среда. Кроме того, функцию надо скомпоновать с RRSAF.

Минимальные требования к компилятору и языковая среда для пользовательских функций описаны в разделе *DB2 Release Guide*.

Примеры подготовки программ на языке JCL – DSNHASM, DSNHC, DSNHCPP, DSNHICOB и DSNHPLI – иллюстрируют прекомпиляцию, компиляцию и компоновку программ DB2 на языках ассемблер, C, C++, COBOL и PL/I. Если ваша подсистема DB2 сконфигурирована для работы с языковая среда, можно использовать эти примеры при подготовке

пользовательских функций. Для объектно–ориентированных программ на языках C++ или COBOL в примерах JCL DSNHCPP2 и DSNHICB2 приводятся советы по подготовке программ.

3. Если пользовательская функция содержит операторы SQL, дайте определяющему функцию полномочия EXECUTE для пакета функции.

## **Обеспечение повторной входимости пользовательской функции**

Рекомендуется компилировать и строить пользовательскую функцию как повторно–входимую. (В случае программы на ассемблере необходимо также предусмотреть повторную входимость в тексте программы.)

Повторно–входимые пользовательские функции имеют следующие преимущества:

- Операционной системе не нужно загружать функцию в память при каждом новом вызове.
- Несколько задач в адресном пространстве хранимых процедур, заданном WLM, могут использовать одну копию пользовательской функции. Это сокращает объем виртуальной памяти, занимаемый текстом программы в адресном пространстве.

**Подготовка пользовательских функций, содержащих несколько программ:** Если пользовательская функция состоит из нескольких программ, каждая программа, содержащая операторы SQL, должна быть связана в отдельный пакет. У определяющего пользовательскую функцию должны быть полномочия EXECUTE для всех пакетов, входящих в пользовательскую функцию.

Когда основная программа пользовательской функции вызывает другую программу, DB2 использует специальный регистр CURRENT PACKAGESET, чтобы определить, в каком собрании нужно искать пакет вызывающей программы. Основная программа может изменить ID собрания оператором SET CURRENT PACKAGESET. Если значение CURRENT PACKAGESET пусто, DB2 использует для поиска пакета способ, который описан в разделе “Порядок поиска” на стр. 455.

## **Определение ID авторизации для вызова пользовательской функции**

Если пользовательская функция вызывается статически, ID авторизации, с которым она вызывается – это ID владельца пакета, содержащего вызов функции.

Если функция вызывается динамически, ID авторизации, с которым она вызывается, зависит от значения параметра связывания DYNAMICRULES для пакета, содержащего вызов функции.

При выполнении пользовательской функции статические операторы SQL используют в качестве ID авторизации ID владельца пакета этой функции. ID авторизации, с которым выполняются динамические операторы SQL в пакете пользовательской функции, зависит от значения параметра DYNAMICRULES, заданного при связывании.

DYNAMICRULES влияет на многие особенности прикладных программ. Этот параметр описан в разделе “Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES” на стр. 457. Глава 6 *DB2 SQL Reference* и Раздел 3 (Том 1) *DB2 Administration Guide* содержат более подробные сведения об авторизации, требуемой для вызова и выполнения операторов SQL в пользовательской функции.

## Подготовка пользовательских функций для одновременного выполнения

Несколько пользовательских функций и хранимых процедур могут выполняться одновременно, каждая в своей задаче (TCB) OS/390.

Чтобы максимально увеличить число одновременно выполняемых пользовательских функций и хранимых процедур, воспользуйтесь следующими рекомендациями по подготовке:

- Попросите системного администратора задать в процедурах запуска параметр размера региона для адресного пространства хранимых процедур, задаваемого WLM, как REGION=0. Это позволяет получить адресное пространство максимально возможного объема в пределах 16 Мбайт.
- Ограничьте память для прикладных программ в пределах 16 Мбайт путем:
  - Построения программ с атрибутами AMODE(31) и RMODE(ANY)
  - Компиляции программ COBOL с опциями RES и DATA(31)
- Ограничьте память для языковая среда с помощью следующих опций времени выполнения:

<b>HEAP(,,ANY)</b>	Выделяет память для кучи программы выше 16–Мбайтной границы
<b>STACK(,,ANY,,)</b>	Выделяет память для стека программы выше 16–Мбайтной границы
<b>STORAGE(,,4K,,)</b>	Уменьшает резервную область памяти ниже 4–Кбайтной границы
<b>BELOWHEAP(4K,,)</b>	Уменьшает памяти кучи ниже 4–Кбайтной границы
<b>LIBSTACK(4K,,)</b>	Уменьшает стек библиотеки ниже 4–Кбайтной границы
<b>ALL31(ON)</b>	Обеспечивает выполнение всех программ, содержащихся во внешней пользовательской функции, с опциями AMODE(31) и RMODE(ANY)

Эти опции может задать определяющий в параметре RUN OPTIONS оператора CREATE FUNCTION или системный администратор в качестве умолчаний при установке языковая среда.

Например, в параметре RUN OPTIONS можно указать:

H(,,ANY),STAC(,,ANY,,),STO(,,,4K),BE(4K,,),LIBS(4K,,),ALL31(ON)

- Попросите системного администратора задать в параметре NUMTCB для адресных пространств хранимых процедур, задаваемых WLM, значение, большее 1. Это позволит выполнять в одном адресном пространстве

несколько TCB. Учтите, что задание в NUMTCB значения, большего 1, также снижает уровень изоляции прикладной программы. Например, неверный указатель в одной прикладной программе может привести к записи в память, выделенную другой программе.

## Тестирование пользовательской функции

Некоторые распространенные средства отладки, как например, TSO TEST, недоступны в среде выполнения пользовательских функций. В этом разделе описываются некоторые альтернативные способы отладки.

*CODE/370:* Для отладки пользовательских функций DB2 for OS/390, написанных на одном из поддерживаемых языков, можно использовать лицензированную программу CoOperative Development Environment/370, которая работает с языковой средой. С программой CODE/370 можно работать в интерактивном или в пакетном режиме.

*Интерактивная работа с CODE/370:* Для интерактивной отладки пользовательской функции с помощью CODE/370 нужно использовать CODE/370 PWS Debug Tool на рабочей станции. Необходимо также, чтобы в системе, где выполняется пользовательская функция, была установлена CODE/370. Чтобы отладить пользовательскую функцию с помощью отладчика PWS:

1. Скомпилируйте пользовательскую функцию с опцией TEST. При этом в программу будет занесена информация, используемая Debug Tool.
2. Запустите отладчик. Один из способов – указать опцию времени выполнения языковая среда TEST. Опция TEST определяет, когда и как запускается отладчик. Удобнее всего указать опции времени выполнения в параметре RUN OPTIONS операторов CREATE FUNCTION или ALTER FUNCTION. Более подробные сведения о параметре RUN OPTIONS смотрите в разделе “Компоненты определения пользовательской функции” на стр. 270.

Допустим, например, что вы указали опцию:

TEST(ALL,\*,PROMPT,JBJONES%SESSNA:)

Значения параметра имеют следующий смысл:

### ALL

Debug Tool получает управление, когда происходит прерывание внимания, ненормальное завершение или ситуация серьезности не ниже 1 языковая среда.

- \* Команды отладчика будут вводиться с терминала.

### PROMPT

Debug Tool запускается сразу после инициализации языковая среда.

### JBJONES%SESSNA:

CODE/370 запускает на рабочей станции сеанс, определяемый для APPC/MVS как JBJONES с ID сеанса SESSNA.

3. Если нужно сохранять выходную информацию сеанса отладки, введите команду, задающую имя файла журнала. Например, после следующей команды на рабочей станции начинается запись в файл журнала с именем dbgtool.log.

```
SET LOG ON FILE dbgtool.log;
```

Это должна быть первая команда, вводимая с терминала или включенная в командный файл.

*Работа с CODE/370 в пакетном режиме:* Для отладки пользовательской функции в пакетном режиме необходимо, чтобы в системе, где выполняется пользовательская функция, был установлен CODE/370 Mainframe Interface (MFI) Debug Tool. Для отладки пользовательской функции в пакетном режиме с помощью MFI Debug Tool:

1. Если вы собираетесь вызвать CODE/370 с помощью опции времени выполнения языковая среда TEST, скомпилируйте пользовательскую функцию с опцией TEST. В программу будет помещена информация, используемая отладчиком во время сеанса отладки.
2. Выделите набор данных журнала для выходных данных CODE/370. Поместите оператор DD для этого набора журнала данных в процедуру запуска для адресного пространства хранимых процедур.  
3. Введите в набор данных команды для CODE/370. Поместите в процедуру запуска адресного пространства хранимых процедур оператор определения данных для этого набора данных. space. Чтобы определить набор данных, содержащий команды отладчика MFI, для CODE/370, укажите имя этого набора данных или имя определения данных в опции времени выполнения TEST. Например, следующая опция укажет CODE/370, что команды находятся в наборе данных, связанном с именем определения данных TESTDD:

```
TEST(ALL,TESTDD,PROMPT,*)
```

Первой командой в наборе данных должна быть следующая:

```
SET LOG ON FILE имя определения данных;
```

Протокол работы отладчика будет записываться в набор данных журнала, определенный на втором шаге. Например, если в процедуре запуска адресного пространства хранимых процедур набор данных журнала определен с именем определения данных INSPLLOG, первая команда должна иметь вид:

```
SET LOG ON FILE INSPLLOG;
```

4. Вызовите отладчик. Это можно сделать двумя способами:

- Можно задать опцию времени выполнения языковая среда TEST. Удобнее всего сделать это в параметре RUN OPTIONS оператора CREATE FUNCTION или ALTER FUNCTION.
- Можно поместить вызовы CEETEST в исходном тексте пользовательской функции. Если эта функция уже существует, ее придется заново скомпилировать, скомпоновать и связать. После этого необходимо отдать команды STOP FUNCTION SPECIFIC и START FUNCTION SPECIFIC, чтобы заново загрузить пользовательскую функцию.

Можно совместить опцию времени выполнения языковая среда TEST с вызовами CEETEST. Например, можно задать имя набора данных, содержащего команды, с помощью опции TEST, а с помощью вызовов CEETEST определять, когда передать управление отладчику.

Программа CODE/370 описана в книге *CoOperative Development Environment/370: Debug Tool*.

**Направьте сообщения отладки на SYSPRINT:** Можете включить в текст пользовательской функции простые команды печати, которая направляется в поток SYSPRINT. Затем воспользуйтесь System Display and Search Facility (SDSF) для проверки содержимого SYSPRINT, пока выполняется адресное пространство хранимых процедур установленный WLM. Можно сделать ввод–вывод последовательным, запустив адресное пространство хранимых процедур, заданное WLM, с NUMTCB=1.

**Прикладные программы–драйверы:** Можно написать небольшую прикладную программу–драйвер, которая вызывает пользовательскую функцию в качестве подпрограммы и передает ей список параметров. Затем можете протестировать и отладить пользовательскую функцию как обычную прикладную программу в TSO. Это даст возможность воспользоваться TSO TEST и другими распространенными средствами отладки.

**SQL INSERT:** Можно вставить отладочную информацию в таблицу DB2 с помощью операторов SQL. Это позволит другим компьютерам в сети (например, рабочим станциям) легко обращаться к данным в этой таблице, используя DRDA.

DB2 отбрасывает отладочную информацию в прикладной программе, выполняющей оператор ROLLBACK. Чтобы избежать потери отладочных данных, напишите вызывающую программу так, чтобы она получала эти отладочные данные перед выполнением оператора ROLLBACK.

## Вызов пользовательской функции

Вызвать скалярную пользовательскую функцию с источником или внешнюю пользовательскую функцию можно из любого места оператора SQL, где допустимы выражения. Табличные пользовательские функции можно вызывать только из условия FROM оператора SELECT. Вызывающий оператор SQL может находиться в автономной программе, хранимой процедуре, теле триггера или другой пользовательской функции.

Подробные сведения, необходимые для вызова пользовательской функции, приводятся в разделах:

- “Синтаксис вызова пользовательской функции”
- “Правильный выбор функции для выполнения DB2” на стр. 318
- “Преобразование типов аргументов пользовательской функции” на стр. 324
- “Что происходит при аварийном завершении работы пользовательской функции” на стр. 325

## Синтаксис вызова пользовательской функции

Синтаксис вызова скалярной пользовательской функции показан на рис. 94 на стр. 318:

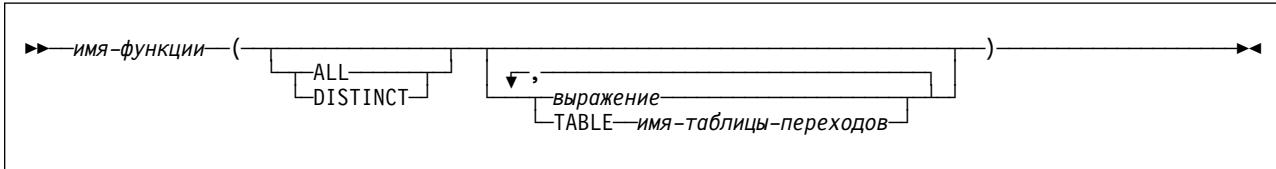


Рисунок 94. Синтаксис вызова скалярной пользовательской функции

Синтаксис вызова табличной функции показан на рис. 95:

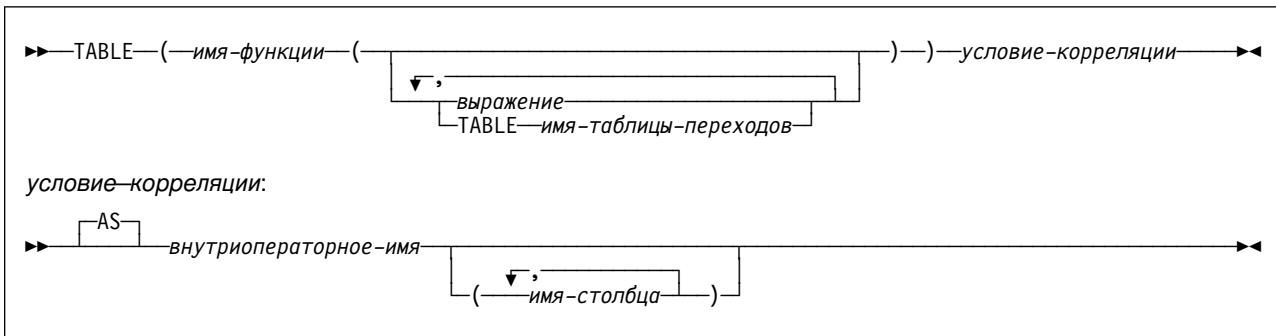


Рисунок 95. Синтаксис вызова табличной функции

Глава 3 *DB2 SQL Reference* содержит дальнейшие сведения о синтаксисе вызова пользовательской функции.

## Правильный выбор функции для выполнения DB2

В подсистеме DB2 может существовать несколько пользовательских функций с одним именем, но с разным числом или типами параметров. Несколько пользовательских функций могут носить одно и то же имя, если типы данных какого-либо из первых 30 параметров различны. Кроме того, имя пользователя функций может совпадать с именем встроенной функции. Когда вызывается функция, DB2 должна определить, какую пользовательскую или встроенную функцию нужно выполнить. Этот процесс называется *разрешением функции*. Понимание процесса разрешения функций DB2 необходимо, чтобы обеспечить вызов именно тех функций, которые требуются.

При разрешении функции DB2 выполняет следующие действия:

1. Определяет, есть ли экземпляры функции, являющиеся кандидатами на выполнение. Если кандидатов не обнаружено, DB2 посыпает сообщение об ошибке SQL.
2. Сравнивает типы данных входных параметров, чтобы определить, какие функции лучше всего подходят для вызова.

При вызове функции с полным именем результатом сравнения типов данных является одна функция с наилучшим совпадением. Она и выбирается для вызова.

При неспецифицированном вызове функции DB2 может найти несколько наиболее подходящих экземпляров, так как в разных схемах могут находиться функции с одинаковыми именами и входными параметрами.

3. Если две или более функций в равной степени подходят для вызова, DB2 выбирает пользовательскую функцию, чье имя схемы раньше встречается на пути выбора.

Например, предположим, что функции SCHEMA1.X и SCHEMA2.X одинаково подходят для вызова. Предположим, что путь выбора функции имеет вид:

"SCHEMA2", "SYSPROC", "SYSIBM", "SCHEMA1", "SYSFUN"

Тогда DB2 выбирает функцию SCHEMA2.X.

В оставшейся части этого раздела говорится о процессе разрешения функции и предлагаются способы обеспечения правильного выбора функции DB2.

## Как DB2 выбирает функции—кандидаты

Экземпляр пользовательской функции будет кандидатом на выполнение, только если он отвечает следующим критериям:

- Если в вызове использовано полное имя функции, схема экземпляра функции совпадает со схемой в вызове.
- Если при вызове указано неспецифицированное имя функции, схема экземпляра функции соответствует схеме в пути выбора вызывающей программы.
- Имя экземпляра функции совпадает с именем в вызове.
  - Количество входных параметров в экземпляре функции соответствует количеству входных параметров в вызове.
  - Автор вызывающей программы имеет право выполнять данный экземпляр функции.
  - Тип каждого из входных параметров в вызове функции совпадает или расширяется до типа соответствующего параметра в экземпляре функции.

Если в вызове функции передается таблица переходов, тип данных, длина, точность и масштаб каждого столбца в таблице переходов должны точно соответствовать типу данных, длине, точности и масштабу столбцов, заданных в определении экземпляра функции. Таблицы переходов описаны в разделе “Глава 3–5. Использование триггеров для работы с активными данными” на стр. 233.

- Отметка времени создания пользовательской функции должна быть более ранней, чем отметка времени связывания пакета или плана, в котором вызывается эта функция.

Если DB2 проверяет права доступа и производит автоматическое повторное связывания плана или пакета, содержащего вызов пользовательской функции, ни одна функция, созданная после выполнения исходного оператора BIND или REBIND для вызывающего плана или пакета не входит в число кандидатов.

Если используется обработчик проверки прав доступа, часть пользовательских функций, не являвшихся кандидатами до выполнения исходного оператора BIND или REBIND вызывающего плана или пакета могут попасть в их число при автоматическом связывании вызывающего плана или пакета. Приложение B (Том 2) *DB2 Administration Guide* описывает разрешение функций с обработчиками авторизации доступа.

Если пользовательская функция вызвана во время автоматического повторного связывания из тела триггера и ей передается таблица переходов, то форма функции, используемая DB2 при выборе функции, включает только те столбцы таблицы переходов, которые существовали во

время выполнения исходного оператора BIND или REBIND пакета или плана вызывающей программы.

Определить, может ли тип данных быть расширен до другого типа, можно по Табл. 34. В первом столбце перечисляются типы данных в вызове функции. Во втором столбце перечисляются типы данных, до которых могут быть расширены типы из первого столбца, по мере убывания предпочтительности. Например, предположим, что в следующем операторе A типа SMALLINT:

```
SELECT USER1.ADDTWO(A) FROM TABLEA;
```

Определены два экземпляра USER1.ADDTWO: в одном входной параметр типа INTEGER, а в другом входной параметр типа DECIMAL. Оба экземпляра функции – кандидаты на выполнение, потому что тип SMALLINT может быть распространен до INTEGER или DECIMAL. Однако экземпляр с параметром типа INTEGER подходит больше, так как INTEGER в списке выше, чем DECIMAL.

Таблица 34. Расширение типов данных

Тип данных при вызове функции	Возможные типы для расширения (в порядке предпочтения)
CHAR или GRAPHIC	CHAR или GRAPHIC VARCHAR или VARGRAPHIC CLOB или DBCLOB
VARCHAR или VARGRAPHIC	VARCHAR или VARGRAPHIC CLOB или DBCLOB
CLOB или DBCLOB <sup>1</sup>	CLOB или DBCLOB
BLOB <sup>1</sup>	BLOB
SMALLINT	SMALLINT INTEGER DECIMAL REAL DOUBLE
INTEGER	INTEGER DECIMAL REAL DOUBLE
DECIMAL	DECIMAL REAL DOUBLE
REAL <sup>2</sup>	REAL DOUBLE
DOUBLE <sup>3</sup>	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
ROWID	ROWID
Пользовательский тип	Пользовательский тип данных с тем же именем

Примечания к Табл. 34:

1. Это расширение также возможно, если тип параметра в вызове – локатор большого объекта с этим типом данных.
2. Тип FLOAT длиной до 22 соответствует типу REAL.
3. Тип FLOAT длиной 22 и выше соответствует типу DOUBLE.

## **Выбор лучшего кандидата**

Кандидатами на выполнение могут оказаться несколько функций. В этом случае DB2 определяет, какие экземпляры функции наиболее подходят для вызова, сравнивая типы данных параметров.

Если типы данных всех параметров в экземпляре функции совпадают с типами данных в вызове, экземпляр признается лучшим кандидатом. Если ни одного экземпляра с полным совпадением нет, DB2 сравнивает типы данных, двигаясь по списку параметров слева направо, следующим способом:

1. DB2 сравнивает тип данных первого параметра в вызове с типом данных первого параметра каждого из кандидатов.
2. Если для какого-нибудь экземпляра тип данных первого параметра подходит лучше, чем для остальных экземпляров, этот экземпляр признается лучшим кандидатом. В Табл. 34 на стр. 320 приведены возможные соответствия для каждого типа, начиная с наилучшего и кончая наихудшим.
3. Если типы данных первого параметра для всех экземпляров функции одинаковы, DB2 повторяет этот процесс для следующего параметра. Затем DB2 повторяет его для всех последующих параметров, пока она не найдет наиболее подходящий экземпляр.

**Пример разрешения функции:** Предположим, что программа содержит следующий оператор:

```
SELECT FUNC(VCHARCOL,SMINTCOL,DECCOL) FROM T1;
```

В вызове пользовательской функции FUNC параметр VCHARCOL имеет тип VARCHAR, SMINTCOL – тип SMALLINT, а DECCOL – тип DECIMAL.

Предположим также, что два экземпляра функции с приведенными ниже определениями удовлетворяют критериям из раздела “Как DB2 выбирает функции–кандидаты” на стр. 319 и поэтому являются кандидатами на выполнение.

1-й кандидат:

```
CREATE FUNCTION FUNC(VARCHAR(20),INTEGER,DOUBLE)
RETURNS DECIMAL(9,2)
EXTERNAL NAME 'FUNC1'
PARAMETER STYLE DB2SQL
LANGUAGE COBOL;
```

2-й кандидат:

```
CREATE FUNCTION FUNC(VARCHAR(20),REAL,DOUBLE)
RETURNS DECIMAL(9,2)
EXTERNAL NAME 'FUNC2'
PARAMETER STYLE DB2SQL
LANGUAGE COBOL;
```

DB2 сравнивает тип данных первого параметра в вызове функции с типами данных первого параметра функций–кандидатов. Поскольку первый параметр в вызове типа VARCHAR, и у обоих кандидатов этот параметр также типа

VARCHAR, DB2 не может определить по первому параметру, какой из экземпляров больше подходит. Поэтому DB2 сравнивает типы данных вторых параметров.

Второй параметр в вызове имеет тип SMALLINT. INTEGER, тип данных первого кандидата, соответствует типу SMALLINT лучше, чем REAL, тип второго кандидата. Поэтому DB2 выбирает для выполнения первый кандидат.

## Как можно упростить процесс разрешения функции

Упростить разрешение функции можно с помощью следующих приемов:

- Использовать при вызове функции специфицированное имя. Это вынуждает DB2 искать функции только в указанной вами схеме. Такой вариант дает два преимущества:
  - DB2 с меньшей вероятностью выберет не ту функцию, которую вы хотите использовать. Для вызова могут подходить в равной степени несколько функций. DB2 выбирает функцию, чья схема раньше встречается на пути выбора, а это может быть не та функция.
  - Меньше функций—кандидатов, поэтому DB2 требуется меньше времени на разрешение функции.
- Преобразуйте параметры в вызове пользовательской функции в типы, указанные в ее определении. Например, если входной параметр пользовательской функции FUNC определен как DECIMAL(13,2), а значение, которое надо передать функции, имеет тип целое, преобразуйте целое в DECIMAL(13,2):

```
SELECT FUNC(CAST (INCOL AS DECIMAL(13,2))) FROM T1;
```

- Не определяйте параметры пользовательской функции как CHAR, GRAPHIC, SMALLINT или REAL. Используйте вместо них типы VARCHAR, VARGRAPHIC, INTEGER или DOUBLE. При вызове функции, параметры которой определены как CHAR, GRAPHIC, SMALLINT или REAL, необходимо использовать те же типы данных, что и в определении. Например, если функция FUNC определена с параметром типа SMALLINT, правильный выбор даст только вызов с параметром типа SMALLINT. При следующем вызове не будет выбрана функция FUNC, поскольку константа 123 имеет тип INTEGER, а не SMALLINT:

```
SELECT FUNC(123) FROM T1;
```

## Контроль за разрешением функции с помощью таблицы DSN\_FUNCTION\_TABLE

Утилита DB2 EXPLAIN позволяет получать информацию о разрешении функций. DB2 помещает информацию в создаваемую вами таблицу с именем DSN\_FUNCTION\_TABLE. DB2 помещает строку в DSN\_FUNCTION\_TABLE для каждой упомянутой в операторах SQL функции в следующих ситуациях:

- Выполняется оператор SQL EXPLAIN для оператора SQL, содержащего вызов пользовательской функции.
- Выполняется программа, план которой связан с опцией EXPLAIN(YES), и эта программа выполняет оператор SQL, содержащий вызов пользовательской функции.

Прежде чем использовать EXPLAIN для наблюдения за разрешением функций, создайте таблицу DSN\_FUNCTION\_TABLE. Ее определение имеет следующий вид:

```
CREATE TABLE DSN_FUNCTION_TABLE
  (QUERYNO          INTEGER      NOT NULL WITH DEFAULT,
   QBLOCKNO         INTEGER      NOT NULL WITH DEFAULT,
   APPLNAME        CHAR(8)      NOT NULL WITH DEFAULT,
   PROGNAME        CHAR(8)      NOT NULL WITH DEFAULT,
   COLLID           CHAR(18)     NOT NULL WITH DEFAULT,
   GROUP_MEMBER    CHAR(8)      NOT NULL WITH DEFAULT,
   EXPLAIN_TIME    TIMESTAMP    NOT NULL WITH DEFAULT,
   SCHEMA_NAME     CHAR(8)      NOT NULL WITH DEFAULT,
   FUNCTION_NAME   CHAR(18)     NOT NULL WITH DEFAULT,
   SPEC_FUNC_NAME  CHAR(18)     NOT NULL WITH DEFAULT,
   FUNCTION_TYPE   CHAR(2)      NOT NULL WITH DEFAULT,
   VIEW_CREATOR   CHAR(8)      NOT NULL WITH DEFAULT,
   VIEW_NAME       CHAR(18)     NOT NULL WITH DEFAULT,
   PATH            VARCHAR(254) NOT NULL WITH DEFAULT,
   FUNCTION_TEXT   VARCHAR(254) NOT NULL WITH DEFAULT);
```

Столбцы QUERYNO, QBLOCKNO, APPLNAME, PROGNAME, COLLID и GROUP\_MEMBER имеют те же значения, что в таблице PLAN\_TABLE. Эти столбцы описаны в разделе “Глава 7–4. Использование EXPLAIN для повышения производительности SQL” на стр. 701. Остальные столбцы имеют следующие значения:

#### **EXPLAIN\_TIME**

Отметка времени выполнения оператора EXPLAIN.

#### **SCHEMA\_NAME**

Имя схемы функции, вызываемой в наблюдаемом операторе.

#### **FUNCTION\_NAME**

Имя функции, вызываемой в наблюдаемом операторе

#### **SPEC\_FUNC\_NAME**

Особое имя функции, вызываемой в наблюдаемом операторе

#### **FUNCTION\_TYPE**

Тип функции, вызываемой в наблюдаемом операторе. Возможные значения:

**SU** Скалярная функция

**TU** Табличная функция

#### **VIEW\_CREATOR**

Создатель производной таблицы, если функция, указанная в столбце FUNCTION\_NAME, упоминается в определении производной таблицы. В прочих случаях это поле пусто.

#### **VIEW\_NAME**

Имя производной таблицы, если функция, указанная в столбце FUNCTION\_NAME, упоминается в определении производной таблицы. В прочих случаях это поле пусто.

#### **PATH**

Значение пути выбора функции при разрешении данного вызова функции.

#### **FUNCTION\_TEXT**

Текст вызова функции (имя функции и параметры). Если его длина превышает 100 байтов, этот столбец содержит первые 100 байтов.

Если функция вызывается в инфиксной записи, столбец FUNCTION\_TEXT содержит только имя функции. Предположим, например, что пользовательская функция под названием / записывается в виде A/B. Тогда FUNCTION\_TEXT будет содержать только /, а не A/B.

### **Преобразование типов аргументов пользовательской функции**

При каждом вызове пользовательской функции DB2 присваивает значения ее входных параметров параметрам с типом данных и длиной, указанными в определении функции. Глава 3 книги *DB2 SQL Reference* описывает, как DB2 присваивает значения, если типы данных источника и назначения отличаются.

При вызове пользовательской функции, основанной на другой функции, DB2 преобразует тип данных и длину ее параметров в тип и длину параметров исходной функции.

Следующий пример показывает, что происходит, когда определение параметров функции с источником не совпадает с параметрами исходной функции.

Предположим, что внешняя пользовательская функция TAXFN1 определена следующим образом:

```
CREATE FUNCTION TAXFN1(DEC(6,0))
  RETURNS DEC(5,2)
  PARAMETER STYLE DB2SQL
  LANGUAGE C
  EXTERNAL NAME TAXPROG;
```

Пользовательская функция TAXFN2, основанная на TAXFN1, определена следующим образом:

```
CREATE FUNCTION TAXFN2(DEC(8,2))
  RETURNS DEC(5,0)
  SOURCE TAXFN1;
```

TAXFN2 вызывается следующим оператором SQL:

```
UPDATE TB1
  SET SALESTAX2 = TAXFN2(PRICE2);
```

TB1 имеет следующее определение:

```
CREATE TABLE TB1
  (PRICE1    DEC(6,0),
   SALESTAX1 DEC(5,2),
   PRICE2    DEC(9,2),
   SALESTAX2 DEC(7,2));
```

Предположим теперь, что PRICE2 имеет значение типа DECIMAL(9,2), равное 0001234.56. Сначала DB2 преобразует это значение в тип данных входного параметра, определенный для TAXFN2, то есть DECIMAL(8,2). Параметр получает значение 001234.56. Затем DB2 преобразует это значение в

параметр исходной функции, DECIMAL(6,0). В результате значение принимает вид 001234. (При преобразовании величина усекается, а не округляется.)

Далее, если функция TAXFN1 вернет значение типа DECIMAL(5,2), равное 123.45, DB2 преобразует его в тип DECIMAL(5,0), определенный как тип результата TAXFN2, и оно примет вид 00123. Это значение будет помещено DB2 в столбец SALESTAX2 в операторе UPDATE.

*Преобразование маркеров параметров:* Если при вызове функции используются маркеры параметров, следует правильно преобразовывать типы параметров. Например, если для функции FX определен один параметр типа INTEGER, вызов FX с маркером параметра имеет следующий вид:

```
SELECT FX(CAST(? AS INTEGER)) FROM T1;
```

## Что происходит при аварийном завершении работы пользовательской функции

При ненормальном завершении пользовательской функции программа получает код ошибки SQL –430 для вызывающего оператора, и DB2 переводит единицу работы, содержащую этот оператор, в состояние "необходим откат". Включите в свою программу проверку на ненормальное завершение пользовательской функции и откат единицы работы, содержащей вызов пользовательской функции.

## Советы по вызову пользовательских функций

*Вызывайте пользовательские функции с внешними действиями из списков SELECT:* Функции с внешними действиями лучше вызывать не из предикатов, а из списков SELECT. Будет ли выполнена пользовательская функция в предикате, зависит от пути доступа, который DB2 выбирает для предиката. Чтобы обеспечить выполнение внешнего действия для каждой строки набора результатов, поместите вызов функции в список SELECT.

*Вызов пользовательских функций, определенных как NOT DETERMINISTIC из списков SELECT:* Недетерминированные пользовательские функции лучше вызывать не из предикатов, а из списков SELECT. Следующий пример показывает, что вызов недетерминированной функции из предиката может привести к нежелательным результатам.

Предположим, что выполняется следующий запрос:

```
SELECT COUNTER(), C1, C2 FROM T1 WHERE COUNTER() = 2;
```

Таблица T1 имеет следующий вид:

C1	C2
--	--
1	b
2	c
3	a

COUNTER – пользовательская функция, которая увеличивает значение во временной памяти при каждом вызове.

DB2 вызывает экземпляр функции COUNTER из предиката 3 раза. Предположим, функция COUNTER была сначала вызвана для строки 1, затем для строки 2, затем для строки 3. COUNTER возвращает 1 для строки 1, 2 для

строки 2 и 3 для строки 3. При этом строка 2 удовлетворяет условию предиката WHERE COUNTER()=2, так что DB2 приписывает значение списку SELECT для строки 2. В списке SELECT используется другой экземпляр функции COUNTER, чем в предикате. Поскольку экземпляр в списке SELECT вызван только один раз, он возвращает значение 1. Таким образом, результат запроса имеет следующий вид:

COUNTER()	C1	C2
-----	--	--
1	2	c

Такой результат может не совпадать с ожидаемым.

Результат может отличаться от ожидаемого еще сильнее, в зависимости от порядка извлечения строк из таблицы. Предположим, что на столбце C2 определен индекс по возрастанию. Тогда DB2 сначала извлечет строку 3, затем строку 1, затем строку 2. Это значит, что строка 1 удовлетворяет условию предиката WHERE COUNTER()=2. Значение COUNTER в списке SELECT снова 1, так что результат запроса выглядит следующим образом:

COUNTER()	C1	C2
-----	--	--
1	1	b

---

## Глава 4–4. Создание и применение пользовательских типов

Пользовательский тип – это тип данных, определяемый при помощи оператора CREATE DISTINCT TYPE. Каждый пользовательский тип имеет то же внутреннее представление, что и встроенный тип данных. Пользовательские типы можно использовать так же, как и встроенные типы данных, в любом типе прикладной программы SQL, кроме программ с собственным протоколом DB2.

В этой главе приводятся следующие сведения о пользовательских типах:

- “Введение в пользовательские типы”
- “Применение пользовательских типов в прикладных программах” на стр. 328
- “Объединение пользовательских типов при помощи пользовательских функций и больших объектов” на стр. 334

---

### Введение в пользовательские типы

Допустим, вам нужно определить аудио– и видеоданные в таблице DB2. Можно, конечно, определить столбцы для обоих типов данных как BLOB, однако вам наверняка захочется воспользоваться типом, более адекватно описывающим ваши данные. Для этого определите пользовательские типы. Их затем можно будет использовать при определении столбцов таблицы или обработке данных в этих столбцах. Например, можно определить пользовательские типы для аудио– и видеоданных следующим образом:

```
CREATE DISTINCT TYPE AUDIO AS BLOB (1M);
CREATE DISTINCT TYPE VIDEO AS BLOB (1M);
```

Тогда оператор CREATE TABLE может выглядеть так:

```
CREATE TABLE VIDEO_CATALOG;
(VIDEO_NUMBER CHAR(6) NOT NULL,
 VIDEO_SOUND AUDIO,
 VIDEO_PICS VIDEO,
 ROW_ID ROWID GENERATED ALWAYS);
```

В таблице нужно определить столбец типа ROWID, поскольку таблицы с любым типом столбцов больших объектов требуют наличия столбца ROWID, а если учитывать внутреннее представление, таблица VIDEO\_CATALOG содержит два столбца больших объектов. Дополнительную информацию о данных больших объектовсмотрите в разделе “Глава 4–2. Программирование больших объектов (LOB)” на стр. 253.

Определив пользовательские типы и столбцы с этими типами, можно теперь использовать эти типы данных, как и встроенные типы. Можно использовать эти типы данных в присваиваниях, сравнениях и вызовах функций и хранимых процедур. Однако при присваивании значения одного столбца другому или при сравнении значений двух столбцов эти значения должны быть одинакового пользовательского типа. Например, необходимо присваивать значение столбца типа VIDEO столбцу типа VIDEO, равно как сравнивать значение столбца типа AUDIO можно только со столбцом типа AUDIO. Присваивая значение переменной хоста столбцу с пользовательским типом,

вы можете использовать любой тип данных хоста, совместимый с типом данных, исходным для пользовательского типа. Например, для приема значения AUDIO или VIDEO можно определить переменную хоста следующим образом:

```
SQL TYPE IS BLOB (1M) HVAV;
```

При использовании пользовательского типа в качестве аргумента функции должна существовать версия данной функции, принимающая такой пользовательский тип. Например, если функция SIZE берет на входе тип BLOB, нельзя для ввода автоматически использовать значение типа AUDIO. Однако можно создать пользовательскую функцию с источником, принимающую тип AUDIO в качестве ввода. Например:

```
CREATE FUNCTION SIZE(AUDIO)
RETURNS INTEGER
SOURCE SIZE(BLOB(1M));
```

## Применение пользовательских типов в прикладных программах

Основной причиной использования пользовательских типов является то, что DB2 навязывает *строгую типизацию* для пользовательских типов. Строгая типизация гарантирует, что могут использоваться только функции, процедуры, сравнения и присваивания, определенные для типа данных.

Например, определив пользовательскую функцию для пересчета курса доллара США в евро, вы хотите перестраховаться, чтобы кто-то другой не использовал эту же функцию для пересчета йены в евро, поскольку функция пересчета доллара в евро возвратит в этом случае неправильное значение. Допустим, вы определили три пользовательских типа:

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL(9,2) WITH COMPARISONS;
CREATE DISTINCT TYPE EURO AS DECIMAL(9,2) WITH COMPARISONS;
CREATE DISTINCT TYPE JAPANESE_YEN AS DECIMAL(9,2) WITH COMPARISONS;
```

Если функция преобразования определена так, что в качестве исходного значения берет параметр ввода типа US\_DOLLAR, то DB2 возвратит ошибку при попытке выполнить функцию со входным параметром типа JAPANESE\_YEN.

## Сравнение пользовательских типов

Основное правило сравнения состоит в том, что типы данных для операндов должны быть совместимы. Правило совместимости определяет, например, что все числовые типы (SMALLINT, INTEGER, FLOAT и DECIMAL) являются совместимыми. Таким образом, можно сравнивать значение типа INTEGER со значением типа FLOAT. Однако нельзя сравнивать объект пользовательского типа с объектом другого типа. Объект пользовательского типа можно сравнивать только с объектом точно такого же пользовательского типа.

DB2 не позволяет сравнивать данные пользовательского типа непосредственно с данными типа его источника. Можно, однако, сравнить пользовательский тип с типом его источника, воспользовавшись функцией преобразования типов.

Пусть, например, требуется узнать, какие товары дороже 100 000 долларов проданы в США в июле 1992 (7/92). Поскольку нельзя непосредственно

сравнивать данные типа US\_DOLLAR с элементами данных исходного для US\_DOLLAR типа (DECIMAL), необходимо воспользоваться функцией преобразования типов, чтобы перевести данные из DECIMAL в US\_DOLLAR или из US\_DOLLAR в DECIMAL. Всякий раз, когда вы создаете пользовательский тип, DB2 создает две функции преобразования типов, одну для преобразования исходного типа в пользовательский тип и другую для преобразования пользовательского типа в исходный тип. Для пользователя типа US\_DOLLAR DB2 создает функцию преобразования типов с названием DECIMAL и функцию преобразования типов с названием US\_DOLLAR. При сравнении объекта типа US\_DOLLAR с объектом типа DECIMAL можно воспользоваться одной из этих функций преобразования типов, чтобы типы данных стали идентичными. Пусть таблица US\_SALES определяется следующим образом:

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         US_DOLLAR);
```

Тогда преобразовать данные DECIMAL в данные US\_DOLLAR можно так:

```
SELECT PRODUCT_ITEM
  FROM  US_SALES
 WHERE TOTAL > US_DOLLAR(100000.00)
   AND MONTH = 7
   AND YEAR  = 1992;
```

Преобразование типов обеспечивает выполнение требования идентичности сравниваемых данных.

Нельзя использовать переменные хоста в операторах, подготовленных для динамического выполнения. Как объясняется в разделе “Использование маркеров параметров” на стр. 557, при подготовке оператора можно подставить маркеры параметров для переменных хоста, а затем использовать переменные хоста при выполнении оператора.

Если маркер параметра используется в предикате запроса, а столбец, с которым сравнивается представленное маркером параметра значение, является столбцом пользовательского типа, необходимо преобразовать маркер параметра в пользовательский тип или преобразовать столбец в его исходный тип.

Пусть, например, определен пользовательский тип CNUM:

```
CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISONS;
```

Таблица CUSTOMER определяется следующим образом:

```
CREATE TABLE CUSTOMER
  (CUST_NUM    CNUM NOT NULL,
   FIRST_NAME  CHAR(30) NOT NULL,
   LAST_NAME   CHAR(30) NOT NULL,
   PHONE_NUM   CHAR(20) WITH DEFAULT,
   PRIMARY KEY (CUST_NUM));
```

В прикладной программе вы подготавливаете оператор SELECT, сравнивающий столбец CUST\_NUM с маркером параметра. Поскольку

CUST\_NUM является пользовательским типом, следует преобразовать пользовательский тип в его исходный тип:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER  
  WHERE CAST(CUST_NUM AS INTEGER) = ?
```

Можно, наоборот, преобразовать маркер параметра в пользовательский тип:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER  
  WHERE CUST_NUM = CAST (? AS CNUM)
```

## Присваивание пользовательских типов

Для присваивания столбцов столбцам или констант столбцам пользовательских типов применяются те же правила, что и для сравнения: тип присваиваемого значения должен соответствовать типу объекта, которому присваивается данное значение, или должна существовать возможность преобразования одного типа в другой.

Если необходимо присвоить значение одного пользовательского типа столбцу другого пользовательского типа, должна существовать функция преобразования этого значения из одного типа в другой. Поскольку DB2 предоставляет только функции преобразования типов только между пользовательскими типами и их исходными типами, необходимо написать отдельную функцию для преобразования одного пользовательского типа в другой.

## Присваивание значений столбцов столбцам других пользовательских типов

Пусть таблицы JAPAN\_SALES и JAPAN\_SALES\_98 определены так:

```
CREATE TABLE JAPAN_SALES  
  (PRODUCT_ITEM  INTEGER,  
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),  
   YEAR          INTEGER CHECK (YEAR > 1985),  
   TOTAL         JAPANESE_YEN);  
  
CREATE TABLE JAPAN_SALES_98  
  (PRODUCT_ITEM  INTEGER,  
   TOTAL         US_DOLLAR);
```

Вам требуется вставить значения из столбца TOTAL таблицы JAPAN\_SALES в столбец TOTAL таблицы JAPAN\_SALES\_98. Поскольку операторы INSERT подчиняются правилам назначения, DB2 не позволяет непосредственно вставлять значения из одного столбца в другой, потому что это столбцы являются столбцами разных пользовательских типов. Предположим, что написана пользовательская функция с названием US\_DOLLAR, которая принимает значения ввода типа JAPANESE\_YEN и возвращает значения типа US\_DOLLAR. Тогда эту функцию можно использовать для вставки значений в таблицу JAPAN\_SALES\_98:

```
INSERT INTO JAPAN_SALES_98  
  SELECT PRODUCT_ITEM, US_DOLLAR(TOTAL)  
    FROM JAPAN_SALES  
   WHERE YEAR = 1998;
```

## **Присваивание значений столбцов пользовательских типов переменным хоста**

Правила присваивания пользовательских типов переменным хоста или переменных хоста – столбцам пользовательских типов отличаются от правил присваивания для констант и столбцов.

Присваивание значения столбца пользовательского типа переменной хоста допустимо, если допустимо присваивание значения столбца исходного типа этого пользовательского типа переменной хоста. В следующем примере можно присвоить SIZECOL1 и SIZECOL2, имеющие пользовательский тип SIZE, переменным хоста типа double и short, поскольку исходный тип SIZE, то есть INTEGER, можно присвоить переменным хоста типа double или short.

```
EXEC SQL BEGIN DECLARE SECTION;
    double hv1;
    short hv2;
EXEC SQL END DECLARE SECTION;
CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
:
SELECT SIZECOL1, SIZECOL2
    INTO :hv1, :hv2
    FROM TABLE1;
```

## **Присваивание значений переменных хоста столбцам пользовательских типов**

Если вы присваиваете значение переменной хоста столбцу пользовательского типа, тип переменной хоста должен быть преобразуемым в пользовательский тип. Таблица базовых типов данных и тех базовых типов данных, в которые их можно преобразовать, приведена в разделе Табл. 34 на стр. 320.

В этом примере значения переменной хоста hv2 могут быть присвоены столбцам SIZECOL1 и SIZECOL2, поскольку тип данных short в С эквивалентен типу данных SMALLINT в DB2, а SMALLINT преобразуем в тип данных INTEGER. Однако значения hv1 нельзя присвоить SIZECOL1 и SIZECOL2, поскольку тип данных double в С, эквивалентный типу данных DOUBLE в DB2, не преобразуем в тип данных INTEGER.

```
EXEC SQL BEGIN DECLARE SECTION;
    double hv1;
    short hv2;
EXEC SQL END DECLARE SECTION;
CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
:
INSERT INTO TABLE1
    VALUES (:hv1,:hv1);      /* Неправильный оператор */
INSERT INTO TABLE1
    VALUES (:hv2,:hv2);      /* Правильный оператор */
```

## Применение пользовательских типов в UNION

Как и для сравнений, DB2 навязывает строгую типизацию пользовательских типов в UNION. При использовании UNION для объединения значений столбцов из нескольких таблиц объединяемые столбцы должны быть одинаковых типов. Предположим, например, что создается производная таблица, объединяющая значения таблиц US\_SALES, EUROPEAN\_SALES и JAPAN\_SALES. Столбцы TOTAL этих трех таблиц имеют разные пользовательские типы, поэтому перед объединением табличных значений необходимо преобразовать типы двух столбцов TOTAL в тип третьего столбца TOTAL. Предположим, что в качестве общего пользовательского типа выбран тип US\_DOLLAR. Поскольку DB2 не генерирует функции преобразования типов для преобразования одного пользовательского типа в другой, надо определить две пользовательские функции:

- Функция, преобразующая значения типа EURO в US\_DOLLAR
- Функция, преобразующая значения типа JAPANESE\_YEN в US\_DOLLAR

Предположим, эти функции существуют и обе называются US\_DOLLAR. Тогда запрос показать таблицу объединенных продаж можно выполнить так:

```
SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL  
FROM US_SALES  
UNION  
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR(TOTAL)  
FROM EUROPEAN_SALES  
UNION  
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR(TOTAL)  
FROM JAPAN_SALES;
```

Поскольку итоговым типом обеих функций US\_DOLLAR является US\_DOLLAR, удовлетворено требование совпадения пользовательских типов для объединяемых столбцов.

## Вызов функций с пользовательскими типами

DB2 навязывает строгую типизацию при передаче функции ее аргументов. Это означает что:

- Вы можете передавать функции аргументы, имеющие пользовательские типы, если истинно одно из следующих условий:
  - Определена версия функции, принимающей эти пользовательские типы.Это также приложимо к инфиксным операциям. Для использования одной из пяти встроенных инфиксных операций (||, /, \*, +, -) с пользовательскими типами необходимо определить версию операции, принимающую эти пользовательские типы.
- Можно преобразовать пользовательские типы в типы аргументов функции.
- Если аргументы передаются функции, принимающей только пользовательские типы, передаваемые аргументы должны иметь те же пользовательские типы, что и в определении функции. Если эти типы различны, необходимо преобразовать аргументы в пользовательские типы из определения функции.

Если константы или переменные хоста передаются функции, принимающей только пользовательские типы, необходимо преобразовать константы или переменные хоста в пользовательские типы, принимаемые функцией.

Следующие примеры показывают, как использовать пользовательские типы в качестве аргументов при вызовах функций.

**Пример: Определение функции с пользовательскими типами в качестве аргументов:** Допустим, нужно вызватьстроенную функцию HOUR с пользовательским типом, определенным следующим образом:

```
CREATE DISTINCT TYPE FLIGHT_TIME AS TIME WITH COMPARISONS;
```

Функция HOUR берет в качестве аргумента только данные типа TIME или TIMESTAMP, поэтому нужна функция с источником на основе функции HOUR, принимающая данные типа FLIGHT\_TIME. Эту функцию можно объявить так:

```
CREATE FUNCTION HOUR(FLIGHT_TIME)
RETURNS INTEGER
SOURCE SYSIBM.HOUR(TIME);
```

**Пример: Преобразование аргументов функций в приемлемые типы:**

Другой способ вызвать функцию HOUR – преобразовать аргумент типа FLIGHT\_TIME в тип данных TIME перед вызовом функции HOUR. Допустим, что таблица FLIGHT\_INFO имеет столбец DEPARTURE\_TIME, содержащий данные типа FLIGHT\_TIME, а вы хотите воспользоваться функцией HOUR, чтобы извлечь из времени вылета значение часа. Можно преобразовать DEPARTURE\_TIME в тип данных TIME, а затем вызвать функцию HOUR:

```
SELECT HOUR(CAST(DEPARTURE_TIME AS TIME)) FROM FLIGHT_INFO;
```

**Пример: Использование инфиксной операции с аргументами пользовательского типа:** Допустим, надо сложить два значения типа US\_DOLLAR. Чтобы это сделать, сначала надо определить версию функции +, принимающую в качестве исходных данных данные типа US\_DOLLAR:

```
CREATE FUNCTION "+"(US_DOLLAR,US_DOLLAR)
RETURNS US_DOLLAR
SOURCE SYSIBM."+"(DECIMAL(9,2),DECIMAL(9,2));
```

Поскольку тип US\_DOLLAR основан на типе DECIMAL(9,2), исходная функция должна быть версией функции + с аргументами типа DECIMAL(9,2).

**Пример: Преобразование констант и переменных хоста в пользовательские типы для вызова пользовательской функции:** Допустим, определена функция CDN\_TO\_US:

```
CREATE FUNCTION EURO_TO_US(EURO)
RETURNS US_DOLLAR
EXTERNAL NAME 'CDNCVT'
PARAMETER STYLE DB2SQL
LANGUAGE C;
```

Это означает, что EURO\_TO\_US принимает в качестве исходных только данные типа EURO. Следовательно, если надо вызвать CDN\_TO\_US с аргументом в виде константы или переменной хоста, необходимо преобразовать этот аргумент в пользовательский тип EURO:

```
SELECT * FROM US_SALES
WHERE TOTAL = EURO_TO_US(EURO(:H1));
```

```
SELECT * FROM US_SALES  
WHERE TOTAL = EURO_TO_US(EURO(10000));
```

## Объединение пользовательских типов при помощи пользовательских функций и больших объектов

Пример из этого раздела демонстрирует следующие понятия:

- Создание пользовательского типа на основе типа данных большого объекта
- Определение пользовательских функций с пользовательским типом в качестве аргумента
- Создание таблицы со столбцом пользовательского типа, основанного на типе большого объекта
- Определение табличного пространства большого объекта, вспомогательной таблицы и вспомогательного индекса
- Вставка данных из переменной хоста в столбец пользовательского типа, основанный на столбце большого объекта
- Выполнение запроса, содержащего вызов пользовательской функции
- Преобразование локатора большого объекта во входной тип данных пользовательской функции

Допустим, вы держите документы электронной почты, присланные в адрес вашей компании, в таблице DB2. Тип данных DB2 для документа электронной почты – CLOB, но вы определяете его как пользовательский тип, чтобы иметь возможность контролировать типы операций, выполняемых над электронной почтой. Пользовательский тип определяется следующим образом:

```
CREATE DISTINCT TYPE E_MAIL AS CLOB(5M);
```

Вы также определили и написали пользовательские функции для поиска и возврата следующей информации о документе электронной почты:

- Тема
- Отправитель
- Дата отправки
- Содержание сообщения
- Индикатор наличия в документе указанной пользователем последовательности

Определения пользовательских функций выглядят следующим образом:

```
CREATE FUNCTION SUBJECT(E_MAIL)  
RETURNS VARCHAR(200)  
EXTERNAL NAME 'SUBJECT'  
LANGUAGE C  
PARAMETER STYLE DB2SQL  
NO SQL  
DETERMINISTIC  
NO EXTERNAL ACTION;
```

```

CREATE FUNCTION SENDER(E_MAIL)
RETURNS VARCHAR(200)
EXTERNAL NAME 'SENDER'
LANGUAGE C
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;

CREATE FUNCTION SENDING_DATE(E_MAIL)
RETURNS DATE
EXTERNAL NAME 'SENDDATE'
LANGUAGE C
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;

CREATE FUNCTION CONTENTS(E_MAIL)
RETURNS CLOB(1M)
EXTERNAL NAME 'CONTENTS'
LANGUAGE C
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;

CREATE FUNCTION CONTAINS(E_MAIL, VARCHAR (200))
RETURNS INTEGER
EXTERNAL NAME 'CONTAINS'
LANGUAGE C
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;

```

Таблица, содержащая документы электронной почты, определяется так:

```

CREATE TABLE DOCUMENTS
(LAST_UPDATE_TIME TIMESTAMP,
DOC_ROWID ROWID GENERATED ALWAYS,
A_DOCUMENT E_MAIL);

```

Поскольку эта таблица содержит столбец исходных данных типа CLOB, для нее требуется столбец ROWID и ассоциированное пространство вспомогательной таблицы, вспомогательная таблица и индекс для вспомогательной таблицы. Для определения табличного пространства большого объекта, вспомогательной таблицы и индекса воспользуйтесь операторами, показанными ниже:

```

CREATE LOB TABLESPACE DOCTSLOB
LOG YES
GBPCACHE SYSTEM;

CREATE AUX TABLE DOCAUX_TABLE
IN DOCTSLOB
STORES DOCUMENTS COLUMN A_DOCUMENT;

CREATE INDEX A_IX_DOC ON DOCAUX_TABLE;

```

Для заполнения таблицы документов вы пишете код, который выполняет оператор INSERT, чтобы поместить в таблицу первую часть документа, а затем выполняет серию операторов UPDATE, чтобы присоединить остальные части документа. Например:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hv_current_time[26];
  SQL TYPE IS CLOB (1M) hv_doc;
EXEC SQL END DECLARE SECTION;
/* Определяет текущее время и помещает это значение */
/* в переменную хоста hv_current_time.          */
/* Считывает из файла до 1 Мбайта документальных */
/* данных в переменную хоста hv_doc.           */
:
/* Вставляет значение времени и первые 1 Мбайт      */
/* документальных данных в таблицу.                */
EXEC SQL INSERT INTO DOCUMENTS
  VALUES(:hv_current_time, DEFAULT, E_MAIL(:hv_doc));

/* Если в файле находится больше документальных      */
/* данных, считывает дополнительные 1 Мбайт данных, */
/* а затем применяет, как здесь, оператор UPDATE    */
/* для присоединения данных переменной хоста к      */
/* существующим данным в таблице.                  */
EXEC SQL UPDATE DOCUMENTS
  SET A_DOCUMENT = A_DOCUMENT || E_MAIL(:hv_doc)
  WHERE LAST_UPDATE_TIME = :hv_current_time;
```

Теперь, когда данные находятся в таблице, можно выполнять запросы для получения дополнительных сведений о документах. Например, можно выполнить запрос для определения, какие документы содержат слово 'performance':

```
SELECT SENDER(A_DOCUMENT), SENDING_DATE(A_DOCUMENT), SUBJECT(A_DOCUMENT)
  FROM DOCUMENTS
 WHERE CONTAINS(A_DOCUMENT,'performance') = 1;
```

Поскольку документы электронной почты могут быть очень большими, вам стоит воспользоваться локаторами больших объектов для обработки документальных данных вместо выборки целого документа в переменную хоста. Можно использовать локатор большого объекта для любого пользовательского типа, который определен для одного из типов больших объектов. В следующих примерах показано, как можно преобразовать локатор большого объекта в пользовательский тип, а затем использовать результат в пользовательской функции, принимающей пользовательский тип в качестве аргумента:

```
EXEC SQL BEGIN DECLARE SECTION
    long hv_len;
    char hv_subject[200];
    SQL TYPE IS CLOB_LOCATOR hv_email_locator;
EXEC SQL END DECLARE SECTION
:
/* Выбирает документ в локатор CLOB. */
EXEC SQL SELECT A_DOCUMENT, SUBJECT(A_DOCUMENT)
    INTO :hv_email_locator, :hv_subject
    FROM DOCUMENTS
    WHERE LAST_UPDATE_TIME = :hv_current_time;
:
/* Извлекает тему из документа. Функция */
/* SUBJECT берет аргумент типа E_MAIL, */
/* представляя локатор CLOB как E_MAIL. */
EXEC SQL SET :hv_subject =
    SUBJECT(CAST(:hv_email_locator AS E_MAIL));
:
```



---

## Раздел 5. Разработка прикладных программ баз данных DB2

<b>Глава 5–1. Планирование прекомпиляции и связывания</b> . . . . .	341
Планирование прекомпиляции . . . . .	342
Планирование связывания . . . . .	342
Как связывать модули DBRM . . . . .	343
Планирование будущих изменений в программе . . . . .	344
<b>Глава 5–2. Планирование одновременности</b> . . . . .	351
Что такое одновременность? Что такое блокировки? . . . . .	351
Эффекты блокировок DB2 . . . . .	353
Приостановка . . . . .	353
Истечение срока ожидания . . . . .	353
Тупиковая ситуация . . . . .	354
Основные рекомендации по обеспечению одновременности . . . . .	357
Рекомендации по разработке базы данных . . . . .	357
Рекомендации по разработке прикладных программ . . . . .	358
Атрибуты блокировок транзакций . . . . .	360
Размер блокировки . . . . .	361
Длительность блокировки . . . . .	363
Режим блокировки . . . . .	364
Объект блокировки . . . . .	367
Настройка использования блокировок . . . . .	368
Опции связывания . . . . .	368
Задание уровня изоляции в операторе SQL . . . . .	382
Оператор LOCK TABLE . . . . .	383
Пути доступа . . . . .	385
Блокировка больших объектов . . . . .	387
Связь между блокировками транзакций и блокировками больших объектов . . . . .	387
Иерархия блокировок больших объектов . . . . .	390
Режимы блокировки больших объектов и табличных пространств больших объектов . . . . .	390
Длительность блокировок . . . . .	391
Когда блокировка табличного пространства большого объекта не производится . . . . .	392
Команда LOCK TABLE . . . . .	392
<b>Глава 5–3. Планирование восстановления</b> . . . . .	393
Единица работы TSO (пакетной и диалоговой) . . . . .	393
Единицы работы в CICS . . . . .	394
Единица работы в IMS (диалоговой) . . . . .	395
Планирование программного восстановления: контрольная точка и перезапуск . . . . .	397
Когда необходимы контрольные точки? . . . . .	398
Контрольные точки в программах MPP и транзакционно-ориентированных BMP . . . . .	399
Контрольные точки в пакетно-ориентированных BMP . . . . .	400
Задание частоты контрольных точек . . . . .	401
Единица работы в DL/I batch и в IMS batch . . . . .	401
Координация принятия и отката . . . . .	401

Перезапуск и восстановление в IMS (batch) . . . . .	403
<b>Глава 5–4. Планирование доступа к распределенным данным . . . . .</b>	<b>405</b>
Введение в доступ к распределенным данным . . . . .	405
Два метода программирования распределенных данных . . . . .	408
Использование трехчастных имен таблиц . . . . .	408
Использование явных операторов CONNECT . . . . .	410
Особенности программирования методов доступа . . . . .	412
Подготовка программ для доступа DRDA . . . . .	413
Опции препроцессора . . . . .	413
Опции BIND PACKAGE . . . . .	413
Опции BIND PLAN . . . . .	414
Проверка опций BIND PACKAGE . . . . .	415
Координация изменений нескольких источников данных . . . . .	416
Как координировать изменения . . . . .	416
Что можно делать без двухфазного принятия . . . . .	417
Различные вопросы, связанные с распределенными данными . . . . .	418
Повышение производительности систем с удаленным доступом . . . . .	418
Повышение производительности для больших объектов в распределенной среде . . . . .	419
Используйте опции связывания, повышающие производительность . . . . .	421
Использование блочной выборки . . . . .	424
Задание опции OPTIMIZE FOR n ROWS . . . . .	425
Поддержание согласованности данных . . . . .	429
Копирование таблицы из удаленного положения . . . . .	429
Передача смешанных данных . . . . .	429
Получение данных из таблиц в формате ASCII . . . . .	429
Особенности перехода от доступа по собственному протоколу DB2 к доступу по протоколу DRDA . . . . .	430

## Глава 5–1. Планирование прекомпиляции и связывания

Прикладные программы DB2 включают операторы SQL. Чтобы компилировать эти программы, надо сначала перевести операторы SQL на язык, который распознается компилятором или ассемблером. Поэтому необходимо воспользоваться прекомпилятором DB2, чтобы:

- Заменить операторы SQL в исходных программах на компилируемый код
- Создать модуль требований базы данных (DBRM), который передает запросы SQL DB2 в процессе связывания

На рис. 96 показан весь процесс подготовки программы. В разделе “Глава 6–1. Подготовка прикладной программы к запуску” на стр. 435 даются дополнительные подробности об этих этапах.

После прекомпиляции исходной программы следует создать модуль загрузки, план программы и, возможно, один или несколько пакетов. Это можно делать в любом порядке. Создание модуля загрузки проходит так же, как компиляция и компоновка программы, не содержащей операторов SQL. Создание пакета или плана программы – процесс, свойственный только DB2 – включает связывание одного или нескольких модулей DBRM.

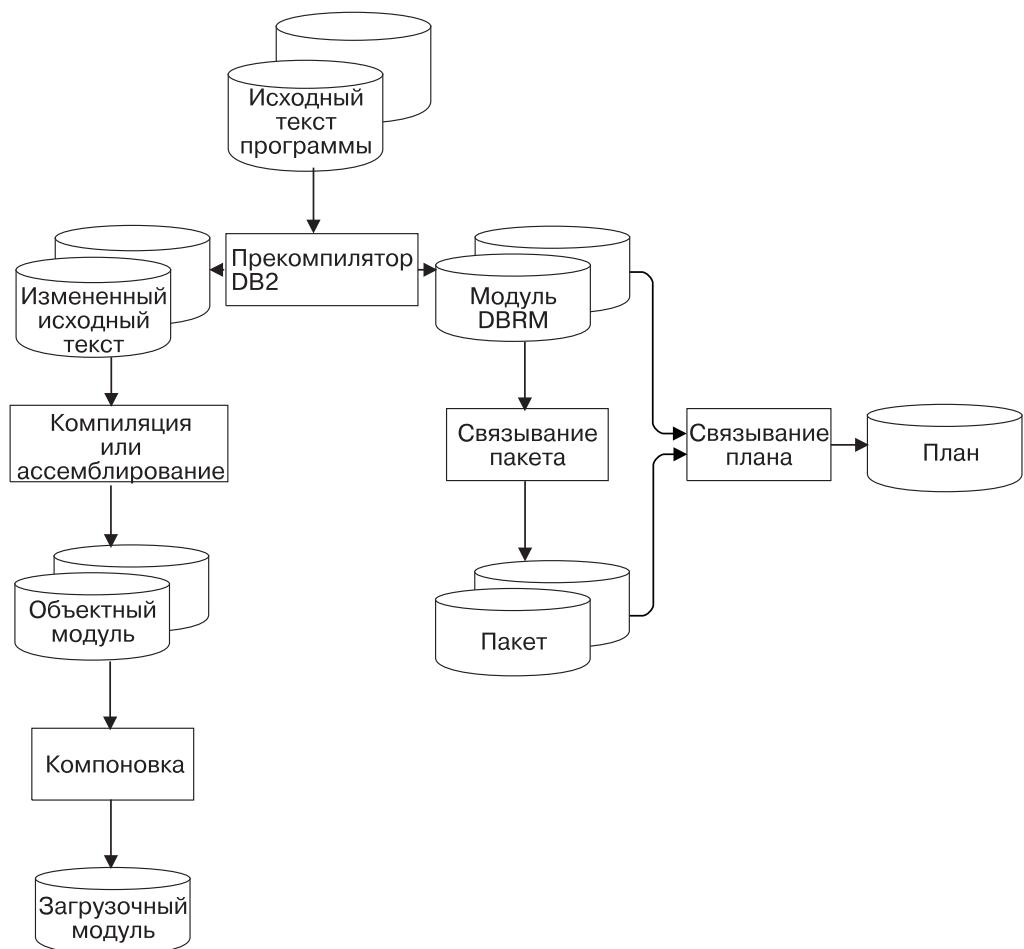


Рисунок 96. Подготовка программы. Необходимо провести два процесса: (1) Компиляцию и компоновку и (2) связывание.

---

## Планирование прекомпиляции

Прекомпилятор DB2 предоставляет много возможностей. Большинство не влияют на разработку и написание программы. Одни из них позволяют сообщить прекомпилятору, что уже сделано — например, какой язык хоста используется или от какой величины зависит максимальная точность десятичных чисел. Другие сообщают, что надо сделать — сколько строк помещать на страницу выходного текста или создавать ли отчет с перекрестными ссылками. Часто бывает удобно придерживаться значений по умолчанию.

Однако несколько возможностей затрагивают структуру программы. Например, надо заранее знать, используются ли параметры NOFOR или STDSQL(YES).

Прежде чем писать программу, просмотрите, пожалуйста, список опций в Табл. 44 на стр. 441.

---

## Планирование связывания

Создавая программу DB2, вы можете связать все модули DBRM в одну операцию и создать единый план программы, или связать все или некоторые модули DBRM в отдельные пакеты в нескольких операциях. После этого программу все равно следует связать как единый план, задать списки включенных в нее пакетов или сортировок и связать все еще не связанные модули DBRM. Независимо от содержания плана, *программа не может работать, пока план не связан*.

### **Связывание или повторное связывание используемого пакета или плана:**

Пакеты и планы блокируются при связывании или запуске. Пакеты, выполняемые в плане, не блокируются, пока план к ним не обращается. Если при выполнении плана некоторые пакеты из списка ни разу не выполняются, они не блокируются.

Нельзя связать или повторно связать работающий план или пакет. Тем не менее, можно связать другую версию работающего пакета.

**Параметры связывания и повторного связывания:** Некоторые параметры команд BIND PACKAGE и BIND PLAN затрагивают структуру программы. Например, существует параметр связывания, который позволяет пакету или плану работать только на определенном соединении CICS или в определенной области IMS и избавляет от необходимости указывать это в программе. В следующих главах подробно разобрано несколько других параметров, в частности, влияющие на использование блокировок в программе, такие как ISOLATION. Читая настоящую главу, вы можете просмотреть эти параметры в разделе Глава 2 книги *DB2 Command Reference*.

### **Предварительные шаги:** До связывания обратите внимание на следующее:

- Выберите, как связывать модули DBRM. Их можно связать в пакеты, непосредственно в планы или использовать комбинированную стратегию.
- Разработайте требования к именам, чтобы эффективно и рационально использовать планы и пакеты.

- Решите, когда программа будет устанавливать блокировку на используемых объектах: на всех объектах при первом размещении плана или по мере использования объектов. Последствия этих решений описаны в разделе “Опции ACQUIRE и RELEASE” на стр. 369.

## Как связывать модули DBRM

Структура программы с самого начала зависит от того, используются ли пакеты. Например, можно поместить некоторые операторы SQL рядом, чтобы при прекомпиляции они попали в один модуль DBRM и были связаны в один пакет.

При связывании плана на вход можно подать только модули DBRM, только список пакетов или и то, и другое. Выбирая между этими альтернативами, учтите эффект повторного связывания (смотрите раздел “Планирование будущих изменений в программе” на стр. 344).

### Связывание только со списком пакетов

Крайнее решение состоит в том, чтобы связать каждый модуль DBRM в отдельный пакет. При связывании пакета на вход поступит один модуль DBRM. Взаимно-однозначное соответствие между программами и пакетами облегчает работу с ними, пока количество пакетов в программе не слишком велико.

Связывание плана, включающего только список пакетов, целесообразно, если предполагается, что программа в дальнейшем будет сильно изменяться.

### Связывание всех модулей DBRM в один план

Другая крайность состоит в том, чтобы связать все модули DBRM в один план. Недостаток этого подхода в том, что, изменив хотя бы один модуль DBRM, необходимо будет повторно связывать весь план, хотя большинство модулей DBRM не изменились.

Связывать все модули DBRM в один план удобно, если программа невелика и не будет изменяться, или если важно получить все ресурсы при размещении плана, а не по мере их использования.

### Связывание с комбинацией модулей DBRM и списка пакетов

Связывать DBRM непосредственно в план и задавать список пакетов удобно при работе с уже существующими программами. Список пакетов можно добавить при повторном связывании существующего плана. Чтобы постепенно перейти к использованию пакетов, при внесении изменений связывайте модули DBRM как пакеты.

### Преимущества пакетов

Использование пакетов в программе определяется ее структурой и задачами. Помните следующее:

**Простота в работе:** Использование пакетов избавляет от необходимости заново связывать весь план после замены одного оператора SQL. Достаточно будет связать пакет, связанный с измененным оператором SQL.

**Возможность наращивания программы:** Связывание пакетов в собрания позволяет добавлять пакеты к существующему плану программы, не связывая

этот план заново. *Собрание* представляет собой группу связанных пакетов. Если указать имя собрания в списке пакетов при связывании плана, план получает доступ ко всем пакетам в этом собрании. Можно первый раз связать план даже с пустым собранием, постепенно пополнять его, отбрасывать и заменять существующие пакеты, не связывая план.

**Использование версий:** Чтобы работать с несколькими версиями плана, не используя пакеты, надо иметь для каждой версии отдельный план, а значит, отдельное имя плана и команду RUN. Если выделить разные версии программы в пакеты, достаточно одного плана, что упрощает миграции и возвраты. Например, можно иметь в программе отдельные уровни — предварительный, отладочный и рабочий, связав каждый уровень как отдельную версию пакета в рамках единого плана.

**Гибкое использование параметров связывания:** Действие параметров BIND PLAN распространяется на все модули DBRM, связанные непосредственно с планом. Действие параметров BIND PACKAGE распространяется только на одиночный модуль DBRM, связанный в этот пакет. Значения параметров для пакета и плана могут не совпадать друг с другом и со значениями для остальных пакетов, включенных в тот же план.

**Гибкое использование спецификаторов имен:** Существует параметр связывания, который присваивает спецификаторы неспецифицированным именам объектов в операторах SQL в плане или пакете. Пакеты позволяют использовать разные спецификаторы для операторов SQL в разных частях программы. При повторном связывании можно будет перенаправить операторы SQL, например, с отладочной таблицы на рабочую.

#### CICS

При работе с пакетами скорее всего не возникнет необходимости в выборе динамического плана и в его обработчике. Доступ к пакету, указанному в списке в плане, не осуществляется до его выполнения. Однако можно использовать выбор динамического плана совместно с пакетами. Это может сократить количество планов в программе и тем самым упростить работу с обработчиком динамических планов. Раздел “Использование пакетов с динамическим выбором плана” на стр. 462 описывает использование пакетов совместно с выбором динамического плана.

## Планирование будущих изменений в программе

Разрабатывая программу, следует предусмотреть, что случится с планами и пакетами, если она будет изменена.

Изменение в программе может сделать неверными один или несколько пакетов, а возможно и весь план. При некоторых изменениях необходимо связывать новый объект; при других достаточно повторного связывания.

- Чтобы связать новый план или пакет, за исключением пакета триггера, используйте подкоманды BIND PLAN или BIND PACKAGE с опцией ACTION(REPLACE).

Чтобы связать новый пакет триггера, заново создайте триггер, связанный с этим пакетом.

- Чтобы повторно связать существующий план или пакет, за исключением пакета триггера, используйте подкоманду REBIND.

Чтобы повторно связать пакет триггера, используйте подкоманду REBIND TRIGGER PACKAGE.

В Табл. 35 описаны меры, которых требуют те или иные изменения. Подробности о пакетах триггеров смотрите в разделе “Работа с пакетами триггеров” на стр. 348.

Если вы хотите изменить параметры связывания при работающем плане или пакете, посмотрите описание этих параметров в разделе Глава 2 *DB2 Command Reference*. Некоторые параметры BIND отсутствуют у команды REBIND.

План или пакет может также стать дефектным по причинам, независимым от работы программы: например, если отбрасывается индекс, используемый в одном из ваших запросов. В таких случаях DB2 может осуществить автоматическое повторное связывание плана или пакета при его следующем использовании. (Подробнее эта операция описана в разделе “Автоматическое связывание” на стр. 348.)

*Таблица 35. Изменения, требующие BIND или REBIND*

<b>Изменение:</b>	<b>Минимальные необходимые меры:</b>
Отбрасывание и пересоздание таблицы, индекса или другого объекта	Если отброшена таблица с триггером, создайте триггер заново при повторном создании таблицы. В остальных случаях изменений не требуется; при следующем запуске будет предпринята попытка автоматического связывания.
Отзыв права использования объекта	Никаких действий не требуется; при следующем запуске будет предпринята попытка автоматического связывания. Если право будет еще недоступно, попытка завершится неудачей; тогда для пакета или плана потребуется команда REBIND.
Обновление статистики каталогов командой RUNSTATS	Используйте команду REBIND для пакета или плана чтобы, возможно, изменить выбранный путь доступа.
Добавление индекса к таблице	Используйте команду REBIND для пакета или плана, чтобы воспользоваться новым индексом.
Изменение параметров связывания	Используйте команду REBIND для пакета или плана или команду BIND с параметром ACTION(REPLACE), если требуемый параметр отсутствует у команды REBIND.
Изменение операторов языка хоста и операторов SQL	Прекомпилируйте, скомпилируйте и скомпонуйте программу. Используйте команду BIND с параметром ACTION(REPLACE) для пакета или плана.

## Отбрасывание объектов

При отбрасывании объекта, от которого зависит пакет, происходит следующее:

- Если пакет не принадлежит работающему плану, этот пакет становится неверным.
- Если пакет принадлежит работающему плану и объект отброшен вне этого плана, объект не отбрасывается и пакет не становится неверным.
- Если пакет принадлежит работающему плану и объект отброшен внутри этого плана, пакет становится неверным.

В любом случае план не становится неверным, если он не содержит модулей DBRM, обращающихся к отброшенному объекту. Если пакет или план становится неверным, при его следующем размещении происходит автоматическое связывание.

## Повторное связывание пакета

Табл. 36 поясняет, какие пакеты связываются в зависимости от того, как указаны *id собрания* (*id собр.*) *id пакета* (*id пак.*) и *id версии* (*id вер.*) в подкоманде REBIND PACKAGE. Глава 2 *DB2 Command Reference* содержит описание и правила использования этой подкоманды.

REBIND PACKAGE не действует на пакеты, для которых у вас нет права связывания. Звездочка (\*) как идентификатор собраний, пакетов или версий не действует для удаленных пакетов.

Таблица 36. Сфера действия REBIND PACKAGE. “Все” означает все собрания, пакеты или версии на локальном сервере DB2, для которых ID авторизации отдавшего команду имеет право связывания. Символ ‘.’ означает, что требуется точка. ‘\*’ обозначает звездочку.

На входе	Собрания	Пакеты	Версии
*	все	все	все
*•••(*)	все	все	все
*•*	все	все	все
*•••(id вер.)	все	все	id вер.
*•••()	все	все	пустая строка
id собр.*•	id собр.	все	все
id собр.*••(*)	id собр.	все	все
id собр.*••(id вер.)	id собр.	все	id вер.
id собр.*••()	id собр.	все	пустая строка
id собр.*id пак.*(*)	id собр.	id пак.	все
id собр.*id пак.	id собр.	id пак.	пустая строка
id собр.*id пак.*()	id собр.	id пак.	пустая строка
id собр.*id пак.*(id вер.)	id собр.	id пак.	id вер.
*•id пак.*(*)	все	id пак.	все
*•id пак.	все	id пак.	пустая строка
*•id пак.*()	все	id пак.	пустая строка
*•id пак.*(id вер.)	все	id пак.	id вер.

Ниже показаны опции повторного связывания удаленного пакета SNTERSA. Собрание называется GROUP1, ID пакета – PROGA, ID версии – V1. Типы соединений, показанные в подкоманде REBIND, заменят типы соединений, заданные в исходной подкоманде BIND. Параметры подкоманды REBIND описаны в книге *DB2 Command Reference*.

```
REBIND PACKAGE(SNTERSA.GROUP1.PROGA.(V1)) ENABLE(CICS,REMOTE)
```

Звездочку в подкоманде REBIND можно использовать для локальных пакетов, но не для удаленных. Все следующие команды повторно связывают все версии всех пакетов во всех собраниях на локальной системе DB2, для которых у вас есть право связывания.

```
REBIND PACKAGE (*)
REBIND PACKAGE (*.*)
REBIND PACKAGE (*.*.*)
```

Обе следующие команды повторно связывают все версии всех пакетов в локальном собрании LEDGER, для которых у вас есть право связывания.

```
REBIND PACKAGE (LEDGER.*)
REBIND PACKAGE (LEDGER.*.*)
```

Обе следующие команды повторно связывают пустую версию пакета DEBIT во всех собраниях на локальной системе DB2, для которых у вас есть право связывания.

```
REBIND PACKAGE (*.DEBIT)
REBIND PACKAGE (*.DEBIT.)
```

### **Повторное связывание плана**

Ключевое слово PKLIST заменяет ранее заданный список пакетов. Если опустить ключевое слово PKLIST, можно провести повторное связывание с предыдущим списком пакетов. Ключевое слово NOPKLIST удаляет список пакетов, заданный при предыдущем связывании плана.

Следующий пример повторно связывает PLANA и заменяет список пакетов.

```
REBIND PLAN(PLANA) PKLIST(GROUP1.*) MEMBER(ABC)
```

Следующий пример повторно связывает план и отбрасывает весь список пакетов.

```
REBIND PLAN(PLANA) NOPKLIST
```

### **Повторное связывание списков планов и пакетов**

Для набора планов или пакетов, для описания которых недостаточно звездочек, можно создать список подкоманд REBIND, используя информацию в каталоге DB2. После этого список подкоманд можно выполнить через DSN.

Данный прием особенно удобен при завершении операции повторного связывания, прерванной из-за нехватки ресурсов. Повторное связывание для

большого числа объектов, например, REBIND PACKAGE (\*) для ID с правами SYSADM, прерывается, если необходимый ресурс становится недоступным. В результате часть объектов оказывается повторно связанной, часть – нет. Если повторить подкоманду, DB2 попытается заново связать все объекты. Но если создать подкоманды повторного связывания для каждого несвязанного объекта и выполнить только их, DB2 не будет повторять уже сделанную работу, так что ресурсов должно хватить.

Описание этого приема и несколько примеровсмотрите в разделе Приложение Е, “Подкоманды REBIND для списков планов или пакетов” на стр. 957.

## Работа с пакетами триггеров

Пакет триггера – особый тип пакета, создаваемый только когда выполняется оператор CREATE TRIGGER. Пакет триггера выполняется только при активации триггера, с которым он связан.

Подобно остальным пакетам, DB2 помечает пакет триггера как неверный, если отбрасывается таблица, индекс или производная таблица, от которой зависит пакет триггера. При следующей активации триггера DB2 выполняет автоматическое связывание. Но если оно заканчивается неудачно, DB2 не помечает пакет триггера как неработоспособный.

В отличие от других пакетов, пакет триггера освобождается, если отбрасывается таблица, на которой определен этот триггер, поэтому заново создать пакет триггера можно, только создав таблицу и триггер.

С помощью подкоманды REBIND TRIGGER PACKAGE можно повторно связать пакет триггера, помеченный DB2 как неработоспособный. REBIND TRIGGER PACKAGE также позволяет изменять значения параметров, с которыми был первоначально связан пакет триггера. Значения по умолчанию для параметров, которые можно изменить:

- CURRENTDATA(YES)
- EXPLAIN(YES)
- FLAG(I)
- ISOLATION(RR)
- IMMEDWRITE(NO)
- RELEASE(COMMIT)

При запуске подкоманды REBIND TRIGGER PACKAGE можно изменить только значения параметров CURRENTDATA, EXPLAIN, FLAG, IMMEDWRITE, ISOLATION и RELEASE.

## Автоматическое связывание

Автоматическое связывание может производиться, если авторизованный пользователь активировал план или пакет, когда изменяются атрибуты данных, от которых зависит этот план или пакет, или среда, в которой выполняется пакет. Происходит ли автоматическое связывание, зависит от значения поля AUTO BIND на панели установки DSNTIPO. При автоматическом связывании используются те же параметры, что и при предыдущем связывании.

Обычно DB2 помечает план или пакет, для которого требуется автоматическое связывание, как *неверный*. Вот несколько частых ситуаций, в которых DB2 помечает план или пакет как неверный:

- Когда отбрасывается таблица, индекс или производная таблица, от которой зависел план или пакет
- Когда отзывается право владельца на доступ к этим объектам
- Когда отзывается право владельца плана или пакета на выполнение хранимой процедуры, а план или пакет вызывает эту процедуру с помощью формы *литерала CALL*
- Когда в таблицу, от которой зависит план или пакет, добавляется столбец TIME, TIMESTAMP или DATE
- Когда во временную таблицу, от которой зависит план или пакет, добавляется столбец
- Когда изменяется пользовательская функция, от которой зависит план или пакет

Неверность плана или пакета отмечается в столбце VALID в таблицах каталога SYSPLAN и SYSPACKAGE.

В следующих случаях DB2 может автоматически связать план или пакет, не помеченный как неверный:

- План или пакет связан в другой версии DB2, чем версия, в которой он был впервые использован.
- План или пакет, зависимый от положения, работает не на том положении, где был связан. Это случается, когда участники группы совместного использования данных определены по именам положений, и пакет запущен не на том участнике, на котором был связан.

DB2 отмечает план или пакет как *неработоспособный*, если автоматическое связывание завершилось неудачей. Работоспособность плана или пакета отмечается в столбце OPERATIVE в таблицах SYSPLAN и SYSPACKAGE.

Выполняется ли EXPLAIN при автоматическом связывании, зависит от значения поля EXPLAIN PROCESSING на панели установки DSNTIPO и от того, задан ли параметр EXPLAIN(YES). Автоматическое связывание заканчивается неудачей при всех ошибках EXPLAIN кроме “PLAN\_TABLE not found.”

SQLCA недоступна при автоматическом связывании. Поэтому если при автоматическом связывании возникнет конфликт блокировок, сообщения DSNT376I не могут сопровождаться сообщениями DSNT501I. Чтобы увидеть соответствующие сообщения DSNT501I, надо воспользоваться подкомандами REBIND PLAN или REBIND PACKAGE.



---

## Глава 5–2. Планирование одновременности

В начале главы описывается:

- “Что такое одновременность? Что такое блокировки?,”
- “Эффекты блокировок DB2” на стр. 353 и
- “Основные рекомендации по обеспечению одновременности” на стр. 357.

После общих рекомендаций в этой главе описано использование двух главных методов, используемых DB2 для управления одновременностью.

- **Блокировки транзакций** в основном управляют доступом к ресурсам для операторов SQL. Эти блокировки позволяют получить максимальный контроль.
  - В разделе “Атрибуты блокировок транзакций” на стр. 360 описываются различные типы используемых DB2 блокировок транзакций и их взаимодействия.
  - В разделе “Настройка использования блокировок” на стр. 368 описывается, что можно изменить для управления блокировками. Можно использовать:
    - “Опции связывания” на стр. 368
    - “Задание уровня изоляции в операторе SQL” на стр. 382
    - “Оператор LOCK TABLE” на стр. 383

В этих разделах термин **блокировка** (без уточнения) используется для обозначения **блокировки транзакции**.

В некоторых ситуациях для управления одновременностью также используются два других метода.

- **Заявки и бронирование** контролируют доступ к ресурсам для утилит и команд DB2. Информацию о них смотрите в разделе 5 (том 2) руководства *DB2 Administration Guide*.
- **Физические блокировки** используются, только если применяется совместное использование данных DB2. Информацию об этом смотрите в руководстве *DB2 Data Sharing: Planning and Administration*.

В заключительном разделе этой главы описаны блокировки больших объектов. Смотрите раздел “Блокировка больших объектов” на стр. 387.

---

### Что такое одновременность? Что такое блокировки?

**Определение:** Одновременность — это возможность нескольких процессов прикладных программ практически одновременно обращаться к одним и тем же данным.

**Пример:** Прикладная программа для ввода заказа одновременно используется несколькими транзакциями. Каждая транзакция вставляет данные в таблицы информации о накладных и деталях накладных, читает таблицу данных о покупателях и читает и обновляет информацию об имеющихся в наличии деталях. Выполняемые двумя транзакциями две операции с одними и теми же данными могут быть разделены только

микросекундами. С точки зрения пользователей эти операции выполняются одновременно.

**Базовые понятия:** Необходимо управлять одновременностью, чтобы предотвратить потерю изменений данных и такие нежелательные ситуации, как невозможность многократного чтения и обращение к непринятым данным.

**Потеря изменений данных.** Без управления одновременностью два процесса, А и В, могут прочесть из базы данных одну и ту же строку и на основе ее содержимого вычислить новые значения для одного из столбцов этой строки. Если процесс А записывает в строку новое значение, а затем эта же строка обновляется процессом В, обновление данных, выполненное процессом А, будет утеряно.

**Обращение к непринятым данным.** Также без управления одновременностью процесс А может обновить значение в базе данных, а процесс В может прочитать это значение до того, как оно будет принято. Если впоследствии для этого значения, записанного процессом А, будет выполнено не принятие, а откат, вычисления процесса В будут основаны на непринятых (и, возможно, неверных) данных.

**Невозможность многократного чтения.** Некоторые процессы требуют следующей последовательности событий: Из базы данных считывается строка и затем выполняется обработка других требований SQL. Впоследствии вновь считывается эта же строка и она должна содержать те же значения, что и в первый раз. Без управления одновременностью процесс В может изменить эту строку в промежутке времени между этими двумя операциями чтения.

Чтобы предотвратить возникновение таких ситуаций, если они не разрешены специально, DB2 может использовать *блокировки* для управления одновременностью.

**Что делают блокировки?** Блокировка связывает ресурс DB2 с процессом прикладной программы, влияя на возможности других процессов обращаться к этому ресурсу. Про такой процесс, связанный с ресурсом, говорят, что он “сохраняет” блокировку или “владеет” блокировкой. DB2 использует блокировки, чтобы гарантировать, что к данным, измененным процессом, но еще не принятым, не обращаются другие процессы.

**Что нужно делать для использования блокировок?** Для защиты целостности данных процесс прикладной программы использует блокировки неявно (этим управляет DB2). Нет необходимости явно требовать блокировку данных, чтобы сделать недоступными непринятые данные. Поэтому иногда для использования блокировок DB2 не нужно ничего делать. Вне зависимости от того, запрашивает процесс блокировки или нет, для блокировок применяются некоторые главные параметры. Можно добиться лучшего использования ресурсов и улучшить одновременность, если понимать влияние этих параметров.

## Эффекты блокировок DB2

Эффекты блокировок, которые нужно минимизировать – это *приостановка, истечение срока ожидания и тупиковая ситуация*.

### Приостановка

**Определение:** Процесс прикладной программы *приостановлен* если он запрашивает блокировку ресурса, который уже заблокирован другим процессом прикладной программы и не может использоваться совместно. Приостановленный процесс временно прекращает выполняться.

**Порядок преимуществ при запросах блокировок:** Входящие запросы блокировок вносятся в очередь. Запросы на повышение блокировки и запросы на блокировку от процесса прикладной программы, уже сохраняющего блокировку на этот объект, имеют преимущество перед запросами блокировок от новых прикладных программ. Внутри этих групп запросы выполняются в порядке поступления.

**Пример:** Используя прикладную программу для работы с инвентарной описью, два пользователя пытаются одновременно уменьшить количество имеющихся деталей для одного и того же типа детали. В очередь вносятся два запроса блокировок. Второй запрос в очереди приостанавливается и ожидает, пока первый запрос не освободит блокировку.

**Эффекты:** Приостановленный процесс возобновляет работу, если:

- Все процессы, сохраняющие конфликтующие блокировки, освободят их.
- Запрашивающий блокировку процесс заканчивается истечением срока ожидания или тупиковой ситуацией и этот процесс продолжает работу, получив сообщение об ошибке.

### Истечение срока ожидания

**Определение:** Говорят, что для процесса прикладной программы *истек срок ожидания*, если его выполнение прекращено, поскольку он был приостановлен на время, большее чем заранее заданный интервал ожидания.

**Пример:** Процесс прикладной программы пытается обновить данные в большом табличном пространстве, которое в настоящее время реорганизуется утилитой REORG TABLESPACE с опцией SHRLEVEL NONE. Вероятно, что задание этой утилиты не разрешит обращение к этому табличному пространству до истечения срока ожидания процесса прикладной программы.

**Эффекты:** DB2 прекращает процесс, выдает на консоль два сообщения и возвращает процессу SQLCODE -911 или -913. (SQLSTATE '40001' или '57033'). В поле SQLERRD(3) в SQLCA возвращается код причины 00C9008E. Если активна трассировка класса 3 статистики, DB2 записывает информацию трассировки с IFCID 0196.

## IMS

Если используется IMS и истек срок ожидания, выполняются следующие действия:

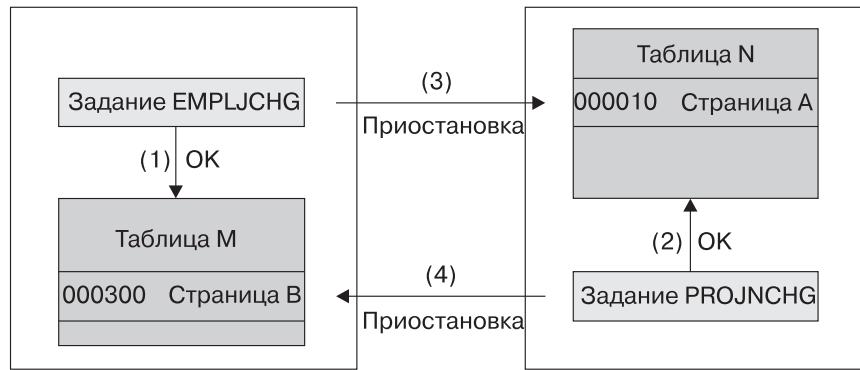
- Процесс пакетной прикладной программы DL/I завершается ненормально с кодом выполнения 04E и кодом причины 00D44033 или 00D44050.
- В любой среде IMS, кроме пакетной среды DL/I:
  - DB2 выполняет откат операций для процесса прикладной программы, чтобы отменить все обновления DB2, выполненные в текущей единице работы.
  - Для BMP, не управляемой сообщениями, IMS выполняет операцию отката для прикладной программы. Если эта операция успешна, IMS возвращает управление прикладной программе и прикладная программа получает SQLCODE –911. Если эта операция неудачна, IMS выдает пользовательский код аварийного завершения 0777 и прикладная программа не получает SQLCODE.
  - Для MPP, IFP или управляемой сообщениями BMP, IMS выдает пользовательский код аварийного завершения 0777, выполняет откат всех непринятых изменений и заново планирует транзакцию. Прикладная программа не получает SQLCODE.

Для операций COMMIT и ROLLBACK срок ожидания не истекает. Но команда STOP DATABASE может обнаруживать истечение срока ожидания и выдавать сообщения на консоль, хотя может повторять это до 15 раз.

## Тупиковая ситуация

**Определение:** Тупиковая ситуация возникает, когда каждый из двух или нескольких процессов прикладных программ сохраняет блокировки для ресурсов, которые необходимы другим процессам и без которых они не могут продолжать работу.

**Пример:** На рис. 97 на стр. 355 показана тупиковая ситуация между двумя транзакциями.



#### Примечания:

1. Две транзакции – это задания EMPLJCHG и PROJNCHG. Задание EMPLJCHG обращается к таблице М и получает монопольную блокировку для страницы В, содержащей запись 000300.
2. Задание PROJNCHG обращается к таблице N и получает монопольную блокировку для страницы А, содержащей запись 000010.
3. Задание EMPLJCHG запрашивает блокировку для страницы А таблицы N, продолжая сохранять блокировку для страницы В таблицы М. Оно приостанавливается, так как монопольная блокировка для страницы А сохраняется заданием PROJNCHG.
4. Задание PROJNCHG запрашивает блокировку для страницы В таблицы М, продолжая сохранять блокировку для страницы А таблицы N. Оно приостанавливается, так как монопольная блокировка для страницы В сохраняется заданием EMPLJCHG. Это тупиковая ситуация.

Рисунок 97. Пример тупиковой ситуации

**Эффекты:** По истечении заранее заданного интервала ожидания (значение параметра DEADLOCK TIME), DB2 может выполнить откат текущей рабочей единицы для одного из процессов или потребовать завершения процесса. Это освободит блокировки и позволит оставшимся процессам продолжать работу. Если активна трассировка класса 3 статистики, DB2 записывает информацию трассировки с IFCID 0172. В поле SQLERRD(3) в SQLCA возвращается код причины 00C90088.

Возможна ситуация, когда два процесса выполняются в распределенных подсистемах DB2 и каждый обращается к ресурсу на другой системе. В таком случае ни одна из подсистем не сможет обнаружить тупиковую ситуацию для этих двух процессов и эта ситуация может быть разрешена, только когда истечет срок ожидания для одного из процессов.

**Диагностика тупиковых ситуаций:** В некоторых случаях тупиковые ситуации могут возникать, если два процесса прикладных программ пытаются обновить данные в одной странице или табличном пространстве.

### TSO, Batch и CAF

При возникновении в этих средах тупиковой ситуации или превышения срока ожидания DB2 пытается выполнить откат SQL для одного из процессов прикладных программ. Если операция ROLLBACK выполняется успешно, эта прикладная программа получает SQLCODE –911. Если операция ROLLBACK оказывается неудачной, а прикладная программа не завершается аварийно, эта прикладная программа получает SQLCODE –913.

### IMS

Если используется IMS и возникла тупиковая ситуация, выполняются следующие действия:

- Процесс пакетной прикладной программы DL/I завершается ненормально с кодом выполнения 04E и кодом причины 00D44033 или 00D44050.
- В любой среде IMS, кроме пакетной среды DL/I:
  - DB2 выполняет откат операций для процесса прикладной программы, чтобы отменить все обновления DB2, выполненные в текущей единице работы.
  - Для BMP, не управляемой сообщениями, IMS выполняет операцию отката для прикладной программы. Если эта операция успешна, IMS возвращает управление прикладной программе и прикладная программа получает SQLCODE –911. Если эта операция неудачна, IMS выдает пользовательский код аварийного завершения 0777 и прикладная программа не получает SQLCODE.
  - Для MPP, IFP или управляемой сообщениями BMP, IMS выдает пользовательский код аварийного завершения 0777, выполняет откат всех непринятых изменений и заново планирует транзакцию. Прикладная программа не получает SQLCODE.

## CICS

Если используется CICS и возникла тупиковая ситуация, средство подключения CICS решает, нужно ли выполнить откат одного из процессов прикладных программ, исходя из значения параметра ROLBE или ROLBI. Если выполняется откат процесса прикладной программы, он получает в SQLCA один из следующих двух SQLCODE:

- 911      Для процесса прикладной программы выдана команда SYNCPOINT с опцией ROLLBACK. Отменяются все обновления (команды CICS, вызовы DL/I, операторы SQL), выполненные во время текущей единицы работы. (SQLSTATE '40001')
- 913      Не была выдана команда SYNCPOINT с опцией ROLLBACK. DB2 выполняет откат только незавершенного оператора SQL, в котором возникла тупиковая ситуация или превышение срока ожидания. CICS не выполняет откат для каких-либо ресурсов. Процесс прикладной программы должен или сам выдать команду SYNCPOINT с опцией ROLLBACK, или завершить работу. (SQLSTATE '57033')

Для проверки SQLCODE и вывода информации SQLCA можно использовать процедуру DSNTIAC. Прикладная программа должна перед возобновлением работы выполнить соответствующие действия.

## Основные рекомендации по обеспечению одновременности

Рекомендации сгруппированы по примерной области действия:

- “Рекомендации по разработке базы данных”
- “Рекомендации по разработке прикладных программ” на стр. 358

## Рекомендации по разработке базы данных

**Храните однотипные объекты вместе:** Поместите таблицы для одной прикладной программы в одну базу данных и создайте для каждого процесса прикладной программы, создающего собственные таблицы, собственную базу данных для этих таблиц. Лучше всего, когда каждый процесс прикладной программы использует как можно меньше баз данных.

**Храните разнородные объекты отдельно:** Дайте пользователям разные ID авторизации для работы с разными базами данных; например, один ID для работы с совместно используемой базой данных и другой ID для работы с собственной базой данных. Это увеличивает число возможных (но не одновременных) прикладных программ, уменьшая число баз данных, к которым может обращаться каждый процесс прикладной программы.

**Планируйте пакетные вставки:** Если прикладная программа выполняет последовательные пакетные вставки, могут возникать избыточные конфликты из-за страниц карты отображения табличного пространства. Эта проблема особенно важна при совместном использовании данных, когда конфликт из-за карты отображения вызывает добавочные затраты на согласование Р-блокировок для страницы. Для таких типов прикладных программ можно использовать опцию MEMBER CLUSTER оператора CREATE TABLESPACE.

При использовании этой опции DB2 игнорирует индекс кластеризации (или неявный индекс кластеризации) при выделении пространства для оператора SQL INSERT. Дополнительную информацию об использовании этого опции при совместном использовании данных смотрите в главе 7 руководства *DB2 Data Sharing: Planning and Administration*. Синтаксис смотрите в главе 6 руководства *DB2 SQL Reference*.

**Используйте *LOCKSIZE ANY* всегда, когда нет необходимости использовать другие опции:**

*LOCKSIZE ANY* – это значение по умолчанию для оператора CREATE TABLESPACE. Это значение позволяет DB2 выбирать размер блокировки; DB2 обычно выбирает *LOCKSIZE PAGE* и *LOCKMAX SYSTEM*, если табличное пространство не является табличным пространством больших объектов. Для табличных пространств больших объектов выбирается *LOCKSIZE LOB* и *LOCKMAX SYSTEM*. Прежде чем использовать *LOCKSIZE TABLESPACE* или *LOCKSIZE TABLE*, необходимо убедиться, что не требуется одновременный доступ к этому объекту. Прежде чем выбрать *LOCKSIZE ROW*, нужно оценить, не приведет ли это к росту затрат на блокировки, и сопоставить эти затраты с получаемым улучшением одновременности.

**Проверьте небольшие таблицы:** Для небольших таблиц, для которых требуется высокая одновременность, оцените число страниц для данных и для индекса. Если индекс содержит короткие записи или много дублирующихся записей, весь индекс может состоять из одной корневой страницы и нескольких конечных страниц. В этом случае для улучшения одновременности разместите данные в большем числе страниц или рассмотрите возможность использования блокировок строк.

**Используйте разбиение данных:** Диалоговые запросы обычно выполняют мало изменений данных, но выполняются часто. Пакетные задания, напротив, выполняются долго и изменяют много строк, но выполняются редко. Эти два типа процессов лучше выполнять отдельно. Можно отделить диалоговые программы от пакетных или два пакетных задания друг от друга. Чтобы отделить диалоговые прикладные программы от пакетных, создайте отдельные разделы. Использование разделов может также эффективно отделять пакетные задания друг от друга.

**Используйте меньшее число строк данных на странице:** Используя условие MAXROWS оператора CREATE или ALTER TABLESPACE, можно задать максимальное число строк на одной странице. Например, если используется MAXROWS 1, каждая строка займет целую страницу и блокировка страницы будет относиться только к одной строке. Рассмотрите эту возможность, если есть причины избегать использования блокировок строк, например, в средах с совместным использованием данных, в которых затраты на блокировку строк могут быть слишком велики.

## Рекомендации по разработке прикладных программ

**Обращайтесь к данным последовательно:** Если разные прикладные программы обращаются к одним и тем же данным, постарайтесь, чтобы они делали это в одном и том же порядке. Например, обе программы обращались к строкам в порядке 1,2,3,5. В таком случае одна прикладная программа будет получать доступ к данным с небольшой задержкой, но две прикладные программы не смогут попасть в тупиковую ситуацию. По этой же причине старайтесь, чтобы разные прикладные программы обращались к одним и тем же таблицам в одном и том же порядке.

**По возможности чаще выполняйте принятие:** Чтобы избежать лишних конфликтов блокировок, выдавайте оператор COMMIT как можно скорее после достижения точки согласованности данных (даже в прикладных программах, только читающих данные). Чтобы неудачные операторы SQL (например, PREPARE) не сохраняли блокировки, после ошибки выдайте оператор ROLLBACK. Операторы, выданные через SPUFI, могут быть немедленно приняты средством автоматического принятия SPUFI.

**Повторяйте выполнение прикладной программы после возникновения тупиковой ситуации или истечения срока ожидания:** Включите в пакетную программу команды для повтора операции после тупиковой ситуации или истечения срока ожидания. Это помогает автоматически исправить такую ситуацию. В поле SQLERRD(3) в SQLCA возвращается код причины, указывающий, возникла ли тупиковая ситуация или истечение срока ожидания.

**Закрывайте указатели:** Если указатель определен с опцией WITH HOLD, блокировки для него могут сохраняться после точки принятия. Как можно раньше закройте этот указатель (используя оператор CLOSE CURSOR), чтобы освободить блокировки и сделать доступными заблокированные ими ресурсы. Будут ли блокировки страниц или строк сохраняться для указателей WITH HOLD, управляет параметром RELEASE LOCKS на панели DSNTIP4. Сохранение локаторов для больших объектов сохраняет блокировки больших объектов после точек принятия. Для снятия этих блокировок используйте оператор FREE LOCATOR.

**Используйте при связывании плана опцию ACQUIRE(USE):** Это лучший вариант для одновременности. Пакеты всегда по умолчанию связываются с опцией ACQUIRE(USE). Опция ACQUIRE(ALLOCATE) дает лучшую защиту от тупиковых ситуаций для высокоприоритетного задания; если нужна эта опция, можно связать все DBRM напрямую с планом.

**В большинстве случаев используйте при связывании сочетание опций ISOLATION(CS) и CURRENTDATA(NO):** Опция ISOLATION(CS) позволяет DB2 освобождать блокировки при первой возможности. Опция CURRENTDATA(NO) позволяет DB2 по возможности избегать блокировок. Кроме этого сочетания, можно использовать следующие опции связывания (в порядке ухудшения одновременности):

1. ISOLATION(CS) с CURRENTDATA(YES), если данные, возвращенные программе, не должны меняться до следующей операции FETCH.
2. ISOLATION(RS), если данные, возвращенные программе, не должны меняться до выполнения прикладной программой операции принятия или отката. Но другие прикладные программы могут вставлять дополнительные строки.
3. ISOLATION(RR), если данные, соответствующие результату запроса, не должны меняться до выполнения прикладной программой операции принятия или отката. Новые строки не могут вставляться в набор ответов.

**С осторожностью используйте опцию ISOLATION(UR):** При изоляции UR почти не используются блокировки. Это быстрый режим и он вызывает мало конфликтов, но при нем могут быть прочтены непринятые данные. Не используйте этот тип изоляции, если не уверены, что прикладные программы и

конечные пользователи готовы получать логически несовместимые данные, которые могут возникать в таких случаях.

**Используйте глобальные транзакции:** Утилита подключения служб управления восстановимыми ресурсами (RRSAF) использует компонент OS/390 под названием службы управления транзакциями OS/390 и менеджера восстановимых ресурсов (OS/390 RRS). OS/390 RRS поддерживает общесистемные службы, координирующие операции двухфазного принятия между продуктами MVS. Для программ RRSAF и транзакций IMS, выполняемых под OS/390 RRS, можно сгруппировать несколько агентов DB2 в единую глобальную транзакцию. Глобальная транзакция позволяет нескольким агентам DB2, участвуя в единой глобальной транзакции, совместно использовать одни и те же блокировки и обращаться к одним и тем же данным. Когда два агента в одной глобальной транзакции обращаются в единице работы к одному и тому же объекту DB2, между ними не возникает тупиковой ситуации. Применяются следующие ограничения:

- Глобальная транзакция ограничена пределами одной подсистемы DB2, выполняемой на СЕС без поддержки параллелизма sysplex.
- Поскольку каждая из "ветвей" глобальной транзакции использует блокировки совместно, непринятые исправления, выполненные одной из ветвей такой транзакции, будут видны другим ветвям транзакции.
- Обработка заявок/бронирования между ветвями глобальной транзакции не поддерживается, что означает, что попытки выполнить CREATE, DROP, ALTER, GRANT, REVOKE могут привести к тупиковой ситуации или к истечению срока ожидания, если они производятся из различных ветвей одной глобальной транзакции.
- Попытки изменить ключ разделения могут привести к тупиковой ситуации или к истечению срока ожидания из-за тех же ограничений, что и обработка заявок/бронирования.
- LOCK TABLE может привести к тупиковой ситуации или к истечению срока ожидания между ветвями глобальной транзакции.

Сведения о том, как включить агент в качестве части глобальной транзакции для программ RRSAF, смотрите в разделе "Глава 7–8. Программирование для утилиты подключения служб управления восстановимыми ресурсами (RRSAF)" на стр. 807.

---

## Атрибуты блокировок транзакций

**Четыре основных атрибута блокировки:**

- "Размер блокировки" на стр. 361
- "Длительность блокировки" на стр. 363
- "Режим блокировки" на стр. 364
- "Объект блокировки" на стр. 367

Знание этих атрибутов помогает понять, почему для процесса возникают приостановки или истечения срока и почему возникают тупиковые ситуации для двух процессов.

# Размер блокировки

## Определение

Размер (иногда также называемый *областью действия* или *уровнем*) блокировки данных в таблице описывает количество контролируемых данных. Возможные размеры блокировок: табличное пространство, таблица, раздел, страница, строка. В этом разделе описаны блокировки данных, кроме данных больших объектов. Информацию о блокировках больших объектовсмотрите в разделе “Блокировка больших объектов” на стр. 387.

## Иерархия размеров блокировок

Одна и та же часть данных может контролироваться блокировками разных размеров. Блокировка табличного пространства (наибольший размер блокировки) контролирует наибольшее количество данных – все данные в табличном пространстве. Блокировка страницы или строки контролирует только данные в одной странице или строке.

Как показано на рис. 98, блокировки строк и блокировки страниц занимают одинаковое положение в иерархии размеров блокировки.

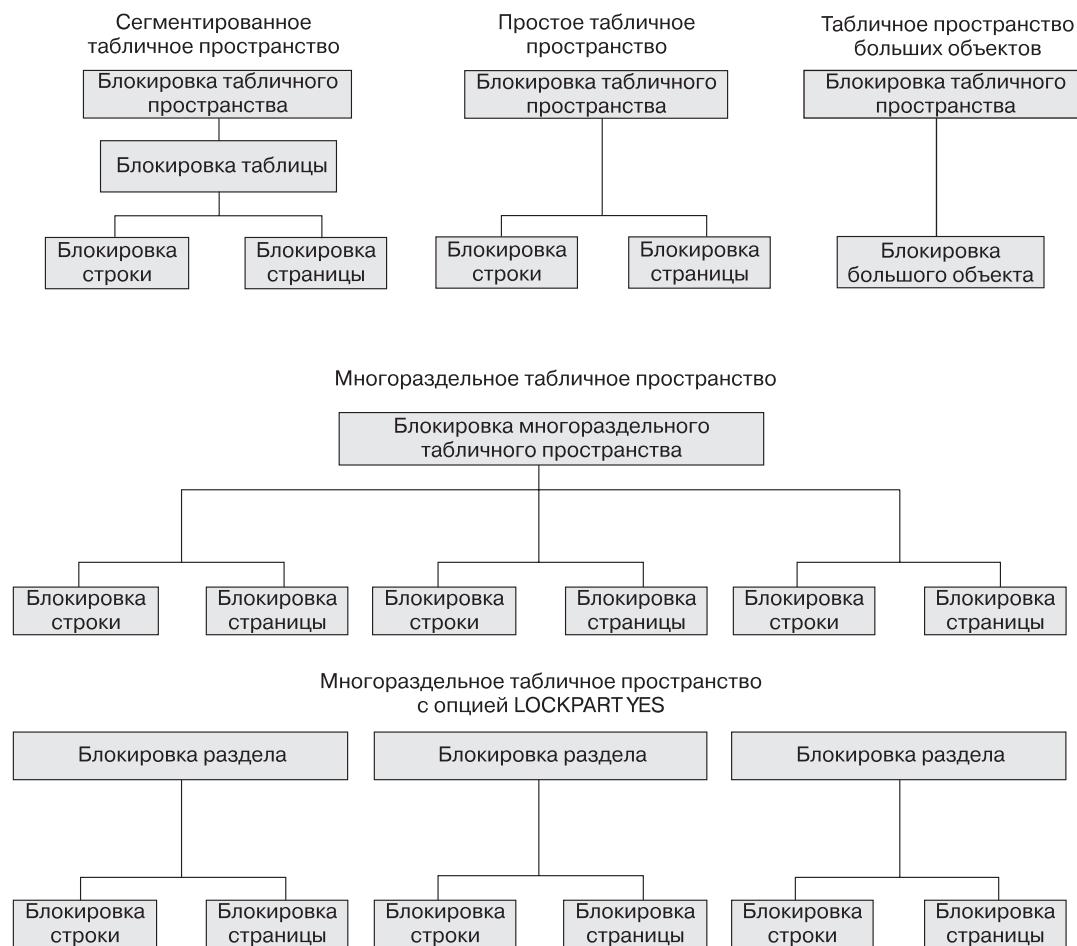


Рисунок 98. Размеры блокируемых объектов

## **Общее влияние размера блокировок**

Блокируя большее или меньшее количество данных, вы выбираете между большей одновременностью и большей производительностью. Если использовать блокировки страниц или строк вместо блокировок таблиц или табличных пространств:

- Одновременность обычно улучшается, то есть уменьшается время отклика и улучшается производительность для многих пользователей.
- Увеличиваются времена обработки и использование памяти. Это особенно заметно при пакетной обработке, просматривающей или обновляющей много строк.

Если используются только блокировки таблиц или табличных пространств:

- Уменьшаются времена обработки и использование памяти.
- Может ухудшаться одновременность, то есть увеличиваться время отклика, но улучшается производительность для одного пользователя.

## **Влияние разных типов табличных пространств**

- Условие LOCKPART операторов CREATE и ALTER TABLESPACE позволяет управлять тем, как DB2 блокирует **распределенные табличные пространства**. По умолчанию LOCKPART имеет значение NO, что означает, что при обращении к любому разделу используется одна блокировка, блокирующая все это распределенное табличное пространство. В большинстве случаев удобно использовать значение LOCKPART NO.

Если задано LOCKPART YES, отдельные разделы блокируются только при обращении к ним.

Один из случаев, когда нужно использовать LOCKPART YES – это совместное использование прикладными программами одних и тех же данных, как это описано в главе 7 руководства *DB2 Data Sharing: Planning and Administration*. Это также может быть выгодно при использовании прикладных программ, не использующих совместных данных, но работающих с распределенными табличными пространствами. Для этих программ может быть желательно использовать большие блокировки (S, U или X) разделов, чтобы избежать большого числа блокировок более низких уровней, но сохранить одновременность. Если табличное пространство определено с LOCKPART YES, расширение блокировок не приводит к конфликтам прикладных программ, обращающихся к разным разделам одного и того же табличного пространства.

**Ограничения:** При любом из следующих условий DB2 блокирует *все* разделы даже при использовании LOCKPART YES:

- План связан с опцией ACQUIRE(ALLOCATE)
- Табличное пространство определено с LOCKSIZE TABLESPACE
- Используется LOCK TABLE IN EXCLUSIVE MODE или LOCK TABLE IN SHARE MODE (без опции PART)

Независимо от того, как определено LOCKPART, задания утилит могут контролировать отдельные разделы табличного пространства или индексного пространства и могут выполняться одновременно, работая с другими разделами.

- **Простое табличное пространство** может содержать несколько таблиц. Блокировка этого табличного пространства блокирует все данные во всех этих таблицах. Страница табличного пространства может содержать строки из каждой таблицы. Блокировка страницы блокирует все строки в этой странице, независимо от того, к какой таблице они относятся. Таким образом, блокировка, необходимая для обращения к данным из одной таблицы, может сделать временно недоступными данные из других таблиц. Этого эффекта можно частично избежать, используя блокировки строк вместо блокировок страниц. Но такой способ не помогает избежать эффекта недоступности данных при блокировке табличного пространства.
- В **сегментированном табличном пространстве**, строки разных таблиц хранятся в разных страницах. Блокировка страницы блокирует данные только одной таблицы. DB2 может также использовать блокировку таблицы, блокирующую данные только для одной конкретной таблицы. Одна строка, естественно, содержит данные только из одной таблицы, так что блокировка строки одинаково влияет на доступ к данным как для простого, так и для распределенного табличного табличного пространства: она блокирует одну строку данных таблицы.
- В **табличном пространстве большого объекта** не используются блокировки страниц. В табличном пространстве большого объекта нет понятия строки, поэтому нет блокировки строк. Вместо этого используется блокировка большого объекта. Дополнительную информацию смотрите в разделе “Блокировка больших объектов” на стр. 387.

## Длительность блокировки

### Определение

Длительность блокировки – это длительность интервала времени, в течение которого сохраняется блокировка. Она меняется в зависимости от того, когда блокировка была установлена и когда была освобождена.

### Действие

Для максимальной одновременности лучше использовать блокировки небольших количеств данных, сохраняемые на короткое время, чем использовать блокировки больших количеств данных, сохраняемые на долгое время. Однако выполнение операций блокировки занимает время процессора, а удержание блокировок требует памяти, поэтому использование одной блокировки табличного пространства более экономно, чем использование множества блокировок страниц. При выборе типа блокировок учитывайте эти соображения в соответствии с вашими требованиями к производительности и одновременности.

### **Длительность блокировок разделов, таблиц и табличных пространств:**

Блокировки разделов, таблиц и табличных пространств можно установить при первом выделении плана или же отложить их установление до первого использования блокируемого ресурса. Их можно освободить в следующей точке принятия или удержать до завершения программы.

С другой стороны, блокировки табличного пространства большого объекта всегда устанавливаются при обращении к ресурсу и освобождаются при принятии или же сохраняются до окончания программы. Информацию о

блокировках большого объекта и табличных пространствах больших объектов смотрите в разделе “Блокировка больших объектов” на стр. 387.

#### **Длительность блокировок страниц, строк и больших объектов:**

Блокировки страниц или строк устанавливаются DB2, только когда они необходимы. Момент освобождения блокировки зависит от многих факторов, но она редко сохраняется после следующей точки принятия.

Информацию об управлении длительностью блокировок смотрите в разделе “Опции связывания” на стр. 368.

## **Режим блокировки**

### **Определение**

Режим (иногда также называемый состоянием) блокировки определяет, какие типы доступа к заблокированному объекту разрешены для владельца блокировки и для других одновременных процессов.

Ниже перечислены возможные режимы для блокировок страниц и строк и режимы для блокировок разделов, таблиц и табличных пространств.

Дополнительную информацию о режимах для блокировок больших объектов и блокировках табличных пространств больших объектов смотрите в разделе “Блокировка больших объектов” на стр. 387.

Если заблокирована страница или строка, содержащая ее таблица, раздел или табличное пространство также блокируется. В этом случае эта блокировка таблицы, раздела или табличного пространства имеет один из *неявных* режимов: IS, IX или SIX. Режимы S, U и X блокировок таблиц, разделов или табличных пространств иногда называют *большими* режимами. При чтении режим блокировки SIX является большим режимом, так как он не использует блокировки страниц или строк; в этом смысле он аналогичен режиму блокировки S.

**Пример:** Оператор SQL находит покупателя John Smith в таблице данных о покупателях и изменяет его адрес. Этот оператор блокирует все табличное пространство в режиме IX и конкретную изменяемую строку в режиме X.

### **Режимы блокировок страниц и строк**

Режимы блокировок и их влияния перечислены в порядке возрастания контроля за ресурсами.

**S (SHARE)** Владелец блокировки и любой одновременный процесс могут читать, но не изменять заблокированную страницу или строку. Одновременные процессы могут установить для этой страницы или строки блокировку S или U, или же читать данные, не используя блокировку страницы или строки.

**U (UPDATE)** Владелец блокировки может читать, но не изменять заблокированную страницу или строку. Процесс, одновременный с S-блокировкой, может получить S-блокировку или же читать данные, не используя блокировку страницы или строки, но одновременно выполняемый процесс не может получить U-блокировку.

Использование блокировок U уменьшает вероятность тупиковых ситуаций в случаях, когда владелец блокировки читает страницу или строку, чтобы определить, нужно ли ее изменять – он может начать с блокировки U и затем повысить ее в блокировку X для изменения страницы или строки.

**X (EXCLUSIVE)** Владелец блокировки может читать или изменять заблокированную страницу или строку. Одновременный процесс может обращаться к данным в них, если он выполняется с уровнем изоляции UR. (Одновременный процесс, связанный с изоляцией "стабильность на уровне указателя" (CS) и CURRENTDATA(NO), также может читать данные, заблокированные с режимом X, если DB2 может определить, что эти данные были приняты.)

## Режимы блокировок разделов, таблиц и табличных пространств

Режимы блокировок и их влияния перечислены в порядке возрастания контроля за ресурсами.

### IS (INTENT SHARE)

Владелец блокировки может читать заблокированные таблицу, раздел или табличное пространство, но не может их изменять. Одновременные процессы могут как читать, так и изменять данные в них. Владелец блокировки может установить блокировку страницы или строки для любых данных, которые он читает.

### IX (INTENT EXCLUSIVE)

Владелец блокировки и одновременные процессы могут читать и изменять данные в таблице, разделе или табличном пространстве. Владелец блокировки может установить блокировку страницы или строки для любых данных, которые он читает; он должен установить такую блокировку для всех данных, которые он изменяет.

### S (SHARE)

Владелец блокировки и одновременные процессы могут читать, но не могут изменять данные в таблице, разделе или табличном пространстве. Владельцу данных не требуются блокировки страниц или строк для данных, которые он читает.

### U (UPDATE)

Владелец блокировки может читать, но не может изменять заблокированные данные; однако он может повысить эту блокировку в блокировку X и тогда сможет изменять данные. Процессы, одновременные с U-блокировкой, могут получать S-блокировки и читать эти данные, но не могут получать U-блокировку. Владельцу данных не требуются блокировки страниц или строк для данных.

Использование U-блокировок уменьшает вероятность тупиковых ситуаций в случаях, когда владелец блокировки читает данные,

чтобы определить, нужно ли их изменить. U-блокировки устанавливаются для табличного пространства, если параметр locksize – TABLESPACE и оператор – SELECT FOR UPDATE OF. Аналогично U-блокировки устанавливаются для таблицы, если параметр lock – TABLE и оператор – SELECT FOR UPDATE OF.

**SIX (SHARE с INTENT EXCLUSIVE)** Владелец блокировки может читать и изменять данные в таблице, разделе или табличном пространстве. Одновременные процессы могут читать данные в таблице, разделе или табличном пространстве, но не могут изменять их. Владелец блокировки устанавливает блокировки страниц или строк, только когда изменяет их данные.

#### X (EXCLUSIVE)

Владелец блокировки может читать и изменять данные в таблице, разделе или табличном пространстве. Одновременный процесс может обращаться к данным в них, если он выполняется с уровнем изоляции UR или если данные в табличном пространстве с LOCKPART(YES) обрабатываются с ISO(CS) CD(NO). Владельцу данных не требуется блокировки страниц или строк для данных.

### Совместимость режимов блокировки

Режим блокировки прежде всего влияет на совместимость этой блокировки с другими блокировками.

**Определение:** Блокировки некоторых типов не мешают работе других пользователей. Предположим, процесс прикладной программы A сохраняет блокировку для табличного пространства, к которому также хочет обратиться процесс B. DB2 запрашивает для B тот же конкретный режим блокировки. Если режим блокировки, используемый процессом A, разрешает запросы процесса B, эти две блокировки (или эти два режима) называются **совместимыми**.

**Результаты несовместимости:** Если две блокировки несовместимы, процесс B не может продолжать работу. Он должен ждать, пока процесс A не освободит свою блокировку. (Фактически он должен ждать, пока не будут освобождены все существующие несовместимые блокировки.)

**Какие режимы блокировки совместимы?** Совместимость для блокировок страниц или строк легко определить: В Табл. 37 на стр. 367 показано, совместимы (Да) или нет (Нет) какие-либо два режима блокировки страниц или блокировок строк. Вопросы о совместимости блокировки страницы с блокировкой строки не возникают, так как табличное пространство не может одновременно использовать блокировки страниц и блокировки строк.

*Таблица 37. Совместимость режимов блокировки страниц и блокировки строк*

Режим блокировки	S	U	X
S	Да	Да	Нет
U	Да	Нет	Нет
X	Нет	Нет	Нет

Несколько сложнее определяется совместимость для блокировок табличных пространств. В Табл. 38 показано, совместимы или нет какие-либо два режима блокировки табличных пространств.

*Таблица 38. Совместимость режимов блокировки таблиц и табличных пространств (или разделов)*

Режим блокировки	IS	IX	S	U	SIX	X
IS	Да	Да	Да	Да	Да	Нет
IX	Да	Да	Нет	Нет	Нет	Нет
S	Да	Нет	Да	Да	Нет	Нет
U	Да	Нет	Да	Нет	Нет	Нет
SIX	Да	Нет	Нет	Нет	Нет	Нет
X	Нет	Нет	Нет	Нет	Нет	Нет

## Объект блокировки

### Определение и примеры

Объект блокировки – это блокируемый ресурс.

Может понадобиться учитывать блокировки для каких-либо из этих объектов:

- **Пользовательские данные в таблицах назначения.** Таблица назначения – это таблица, для обращения к которой используется оператор SQL, в котором она задана или своим именем, или через производную таблицу. Блокировки таких таблиц – это наиболее обычный случай и возможности управления ими наибольшие.
- **Пользовательские данные в связанных таблицах.** Для операций, подчиняющихся реляционным ограничениям, могут потребоваться блокировки для связанных таблиц. Например, при удалении данных из родительской таблицы DB2 может также удалять строки из зависимой таблицы. В этом случае DB2 блокирует данные и в зависимой таблице, и в родительской таблице.

Аналогично операции со строками, содержащими значения больших объектов, могут требовать блокировок табличного пространства большого объекта и, возможно, значений больших объектов внутри этого табличного пространства. Дополнительную информацию смотрите в разделе “Блокировка больших объектов” на стр. 387.

Если прикладная программа использует триггеры, триггерные операторы SQL могут вызывать установление дополнительных блокировок.

- **Внутренние объекты DB2.** О большинстве из этих объектов ничего не нужно знать. Некоторые блокировки, которые можно учитывать:
  - Части каталога DB2
  - скелетная таблица указателей (SKCT) представляет план прикладной программы

- скелетная таблица пакетов (SKPT) представляет пакет
- Дескриптор базы данных (DBD) представляет базу данных DB2

Информацию об этих объектах смотрите в разделе 5 (том 2) руководства *DB2 Administration Guide*.

## Индексы и блокировка только для данных

При обработке блокировки страниц индекса не устанавливаются. Вместо этого DB2 использует для упорядочения изменений метод *блокировки только для данных*. Для упорядочения изменений внутри страницы устанавливаются задвижки страниц индексов, которые гарантируют физическую непротиворечивость этой страницы. Использование задвижек страниц индексов гарантирует, что транзакции, одновременно обращающиеся к одной странице индекса, не видят эту страницу в частично измененном состоянии.

Для упорядочения операций чтения и обновления записей индексов используются блокировки страниц или строк базовых данных, гарантирующие логическую непротиворечивость данных, то есть свидетельствующие, что данные приняты и для них не будет выполнен откат или отмена. Эти блокировки данных могут сохраняться длительное время, вплоть до принятия. Однако задвижки страниц сохраняются только на короткое время, пока транзакция обращается к этой странице. Поскольку страницы индекса не блокируются, сценарии вставки в активную область (когда несколько транзакций пытаются одновременно вставить различные записи в одну страницу индекса) не приводят к конфликтам для индекса.

Требования, использующие доступ только через индексы, могут блокировать страницу или строку данных, и такие блокировки могут конфликтовать с другими процессами, блокирующими данные. Однако можно уменьшить число конфликтов, используя методы избежания блокировок. Дополнительную информацию об избежании блокировок смотрите в разделе “Избежание блокировок” на стр. 379.

## Настройка использования блокировок

В этом разделе описывается, как воздействовать на использование блокировок транзакций в конкретной прикладной программе:

- “Опции связывания”
- “Задание уровня изоляции в операторе SQL” на стр. 382
- “Оператор LOCK TABLE” на стр. 383

## ОПЦИИ СВЯЗЫВАНИЯ

Эти опции определяют, когда процесс прикладной программы устанавливает и освобождает свои блокировки и насколько его действия изолированы от возможных влияний других выполняющихся одновременно процессов.

Эти опции операций связывания относятся к блокировкам транзакций:

- “Опции ACQUIRE и RELEASE” на стр. 369
- “Опция ISOLATION” на стр. 373
- “Опция CURRENTDATA” на стр. 377

## Опции ACQUIRE и RELEASE

**Действие:** Опции связывания ACQUIRE и RELEASE определяют, когда DB2 блокирует объект (таблицу, раздел или табличное пространство), используемый прикладной программой, и когда освобождает эту блокировку. (Опции ACQUIRE и RELEASE не влияют на блокировки страниц, строк или больших объектов.) Эти опции применяются к статическим операторам SQL, которые связаны перед выполнением программы. Если программа выполняет динамические операторы SQL, блокируемые ими объекты блокируются при первом обращении к ним и освобождаются в следующей точке принятия хотя некоторые блокировки для динамического SQL могут сохраняться и после точки принятия. Смотрите раздел “Опция RELEASE и кэширование динамических операторов” на стр. 370.

Опция	Действие
<b>ACQUIRE(ALLOCATE)</b>	Блокировка устанавливается при выделении объекта. Эта опция не разрешена для BIND или REBIND PACKAGE.
<b>ACQUIRE(USE)</b>	Блокировка устанавливается при первом обращении к объекту.
<b>RELEASE(DEALLOCATE)</b>	Блокировка освобождается при освобождении объекта (при завершении программы). Это значение не влияет на динамические операторы SQL, для которых всегда используется RELEASE(COMMIT), за исключением случаев, когда используется кэширование динамических операторов. Информацию об опции RELEASE при кэшировании динамических операторов смотрите в разделе “Опция RELEASE и кэширование динамических операторов” на стр. 370.
<b>RELEASE(COMMIT)</b>	Блокировка освобождается в следующей точке принятия, за исключением случаев использования сохраняемых указателей или сохраняемых локаторов . Если прикладная программа вновь обращается к объекту, она должна опять установить блокировку.

**Пример:** Прикладная программа по разным критериям выбирает из таблицы имена и номера телефонов сотрудников. Сотрудники могут изменять свои собственные номера телефонов. Они могут выполнять несколько поисков подряд. При связывании этой прикладной программы использовались опции ACQUIRE(USE) и RELEASE(DEALLOCATE), так как:

- Альтернативная к ACQUIRE(USE) опция ACQUIRE(ALLOCATE) приводит к тому, что сразу после запуска прикладной программы табличное пространство блокируется в режиме IX, так как он потребуется, если будут выполняться обновления. Но в большинстве случаев использования этой прикладной программы обновления таблицы не производятся и поэтому требуется только менее ограничивающая блокировка IS. Опция ACQUIRE(USE) приводит к тому, что при первом обращении к таблице устанавливается блокировка IS, а потом в случае необходимости DB2 повышает эту блокировку до режима IX.

- В большинстве случаев использования этой программы операции обновления и принятия не производятся. Для этих случаев разница между RELEASE(COMMIT) и RELEASE(DEALLOCATE) невелика. Но администраторы могут обновлять несколько номеров телефонов за один сеанс работы с программой, и программа будет выполнять принятие после каждого обновления. В таком случае при использовании RELEASE(COMMIT) блокировка будет освобождаться и затем DB2 должна будет сразу вновь ее установить. При использовании RELEASE(DEALLOCATE) блокировка будет сохраняться до завершения прикладной программы и не нужно будет освобождать и устанавливать блокировку несколько раз.

**Действие LOCKPART YES:** При блокировке разделов применяются те же правила, что и при блокировке табличных пространств, и блокировки *всех* разделов сохраняются в течение того же времени. Таким образом, если один пакет использует RELEASE(COMMIT), а другой RELEASE(DEALLOCATE), для всех разделов используется RELEASE(DEALLOCATE).

**Опция RELEASE и кэширование динамических операторов:** Обычно опция RELEASE не влияет на динамические операторы SQL. Но есть одно исключение: если используются опции связывания RELEASE(DEALLOCATE) и KEEPDYNAMIC(YES) и подсистема установлена со значением YES для поля CACHE DYNAMIC SQL панели DSNTIP4, DB2 сохраняет в памяти подготовленные операторы SELECT, INSERT, UPDATE и DELETE после точек принятия. Поэтому DB2 может использовать для этих динамических операций опцию RELEASE(DEALLOCATE). В следующих случаях блокировки сохраняются до освобождения объекта или до выполнения принятия после того, как подготовленный оператор удален из памяти:

- Прикладная программа выдает оператор PREPARE с тем же идентификатором оператора.
- Оператор удален из памяти, так как он более не будет использоваться.
- Объект, от которого зависит этот оператор, был удален или изменен или были отменены необходимые для выполнения этого оператора привилегии.
- Выполнена утилита RUNSTATS для объекта, от которого зависит этот оператор.

Если блокировка S, SIX или X табличного пространства или таблицы в сегментированном табличном пространстве сохраняется после принятия, DB2 иногда во время принятия понижает режим такой блокировки до неявной блокировки (IX или IS). DB2 понижает режим больших блокировок, если они были установлены по одной из следующих причин:

- DB2 установила большую блокировку из-за расширения блокировок
- Прикладная программа выдала оператор LOCK TABLE
- Прикладная программа выполняет оператор массового удаления (DELETE FROM условия WHERE)

Для табличных пространств, определенных с LOCKPART YES, понижение режима блокировки производится так же, как и для других табличных пространств, то есть блокировка понижается на уровне табличного пространства, но не на уровне раздела.

**Рекомендация:** Выберите комбинацию значений для ACQUIRE и RELEASE, исходя из характеристик конкретной прикладной программы.

### Преимущества и недостатки этих комбинаций

**ACQUIRE(ALLOCATE) / RELEASE(DEALLOCATE):** В некоторых случаях можно избежать тупиковых ситуаций, заблокировав все необходимые ресурсы сразу после запуска программы. Эта комбинация наиболее удобна для долго работающих прикладных программ, которые выполняются часами и обращаются к различным таблицам, так как она предохраняет эту обработку от возможных тупиковых ситуаций.

- Все таблицы или табличные пространства, используемые в DBRM, связанными с этим планом, блокируются при выделении плана.
- Все таблицы или табличные пространства разблокируются только при завершении плана.
- Используются наиболее ограничивающие блокировки, которые могут потребоваться для выполнения всех операторов SQL в плане, независимо от того, выполняются ли в действительности эти операторы.
- Ограничительные условия не проверяются до обращения к данному набору страниц. Установление блокировок при выделении плана гарантирует совместимость задания с другими заданиями SQL. Откладывание проверки ограничительных условий до момента первого доступа улучшает характеристики, однако возможно, что транзакция SQL может:
  - Сохранять блокировку остановленного табличного пространства или раздела
  - Установить блокировку табличного пространства или раздела, запущенных только для обращений утилит DB2 (ACCESS(UT))
  - Установить монопольную блокировку (IX, X) табличного пространства или раздела, запущенных для доступа только для чтения (ACCESS(RO)), запрещая тем самым другим процессам доступ для чтения

**Недостатки:** Эта комбинация ухудшает одновременность. Она может сильно блокировать ресурсы на большее время, чем это необходимо. Кроме того, опция ACQUIRE(ALLOCATE) отключает выборочную блокировку разделов – при обращении к табличному пространству, определенному с LOCKPART YES, блокируются все разделы.

**Ограничения:** Эта комбинация не разрешена для BIND PACKAGE.

Используйте эту комбинацию, если эффективность обработки важнее, чем одновременность. Ее хорошо использовать для пакетных заданий, которые освобождают таблицы или табличные пространства только для того, чтобы почти сразу вновь их заблокировать. Это может даже улучшить одновременность, так как позволит пакетным заданиям быстрее завершать свою работу. Обычно не нужно использовать эту комбинацию, если прикладная программа содержит много операторов SQL, которые выполняются редко.

**ACQUIRE(USE) / RELEASE(DEALLOCATE):** В большинстве случаев эта комбинация дает наиболее эффективное использование времени обработки.

- Таблицы, разделы или табличные пространства, используемые планом или пакетом, блокируются только, когда это требуется во время выполнения.
- Все таблицы или табличные пространства разблокируются только при завершении плана.
- Используется наименее ограничивающая блокировка, требуемая для выполнения каждого оператора SQL, за исключением случаев, когда более ограничивающая блокировка использовалась предыдущим оператором, в этих случаях блокировка остается без изменений.

**Недостатки:** Эта комбинация может увеличить число тупиковых ситуаций. Поскольку порядок, в котором устанавливаются блокировки, определяется только во время выполнения, может возникнуть больше задержек одновременного доступа.

**ACQUIRE(USE) / RELEASE(COMMIT):** Это комбинация по умолчанию, она обеспечивает наилучшую одновременность, но требует больше времени обработки, если прикладная программа часто выполняет принятие.

- Таблица или табличное пространство блокируется только при необходимости. Это важно в случаях, когда процесс содержит много редко выполняемых операторов SQL или операторов, которые обращаются к данным только при определенных обстоятельствах.
- Все таблицы и табличные пространства разблокируются, когда:

— TSO, Batch и CAF —

Выполняется оператор SQL COMMIT или ROLLBACK или завершается процесс прикладной программы

— IMS —

Выполняется вызов CHKP или SYNC (для транзакций одиночного режима), вызов GU для PCB ввода–вывода или вызов ROLL или ROLB

— CICS —

Выполняется команда SYNCPOINT.

**Исключение:** Если указатель определен с условием WITH HOLD, блокировки таблиц или табличных пространств необходимы для сохранения позиции указателя после точки принятия. (Дополнительную информациюсмотрите в разделе “Действие условия WITH HOLD для указателя” на стр. 381.)

- Используется наименее ограничивающая блокировка, требуемая для выполнения каждого оператора SQL, за исключением случаев, когда более ограничивающая блокировка использовалась предыдущим оператором. В этих случаях блокировка остается без изменений.

**Недостатки:** Эта комбинация может увеличить число тупиковых ситуаций. Поскольку порядок, в котором устанавливаются блокировки, определяются только во время выполнения, может возникнуть больше задержек одновременного доступа.

**ACQUIRE(ALLOCATE) / RELEASE(COMMIT):** Эта комбинация не разрешена, она вызывает сообщение об ошибке процесса BIND.

### Опция ISOLATION

**Действие:** Задает степень изоляции операций от возможного влияния других выполняющихся одновременно процессов. На основе этого значения DB2 выбирает блокировки таблиц и табличных пространств с наименьшей возможной степенью ограничений и при первой возможности освобождает S- и U-блокировки строк или страниц.

**Рекомендации:** Выберите значение опции ISOLATION, исходя из характеристик конкретной прикладной программы.

### Преимущества и недостатки типов изоляции

Различные уровни изоляции обеспечивают меньшую или большую степень одновременности при большей или меньшей степени защиты от влияния других процессов прикладных программ. Выбирая значения уровня изоляции, следует прежде всего исходить из потребностей прикладной программы. В этом разделе представлены уровни изоляции в порядке от уровня, обеспечивающего наименьшую одновременность (RR), до уровня, обеспечивающего наибольшую одновременность (UR).

**ISOLATION (RR)** Этот тип изоляции позволяет прикладной программе более одного раза читать одни и те же строки и строки, не позволяя другим процессам выполнять для них операции UPDATE, INSERT, или DELETE. Блокируются все строки или страницы, к которым производится обращение, даже если они не удовлетворяют критерию поиска.

На рис. 99 показано, что все блокировки сохраняются до выполнения программой принятия. На этом примере строки, защищенные блокировками L2 и L4, удовлетворяют предикату.

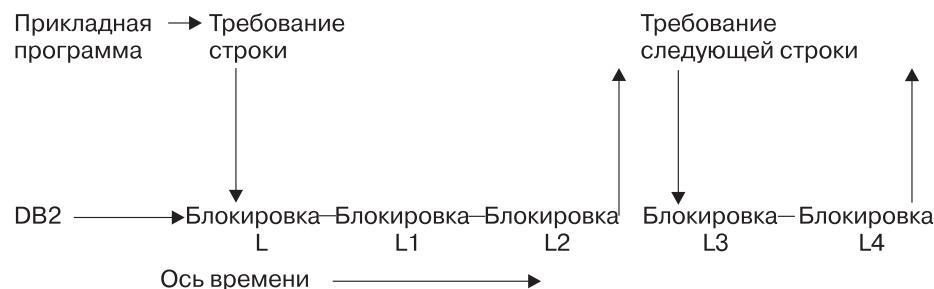


Рисунок 99. Как устанавливает блокировки прикладная программа с уровнем изоляции RR. Все блокировки сохраняются до выполнения прикладной программой операции принятия.

Прикладные программы, использующие изоляцию "многократное чтение" (RR), могут оставлять строки или страницы заблокированными на большее время, особенно в распределенной среде, и могут выдавать заявки для большего числа логических разделов, чем

аналогичная прикладная программа, использующая изоляцию "стабильность на уровне указателя" (CS).

Кроме того, на их работу чаще плохо влияют операции утилит.

Поскольку такой режим может использовать очень много блокировок, может возникнуть расширение блокировок. Частое выполнение принятия освобождает блокировки и помогает избежать расширения блокировок.

При использовании изоляции "многократное чтение" для сканирования табличного пространства выполняется повышение блокировки, чтобы предотвратить вставку строк, которые могут удовлетворять критерию поиска. При обращении через индекс DB2 блокирует диапазон ключей. При обращении к данным с использованием просмотра табличного пространства DB2 блокирует таблицу, раздел, или табличное пространство.

**ISOLATION (RS)** Этот тип изоляции позволяет прикладной программе несколько раз читать одни и те же страницы и строки, не позволяя другим процессам обновлять или удалять выбранные строки. Он дает возможность получить лучшую одновременность, чем изоляция "многократное чтение" (RR), так как хотя другие прикладные программы не могут изменять строки, возвращенные данной прикладной программе, они могут вставлять новые строки или обновлять строки, не удовлетворяющие критерию поиска данной программы. Только те строки или таблицы, которые выбраны как удовлетворяющие предикату первого шага, блокируются до того момента, когда программа выполнит принятие. Это показано на рис. 100. В этом примере строки, защищенные блокировками L2 и L4, удовлетворяют предикату.

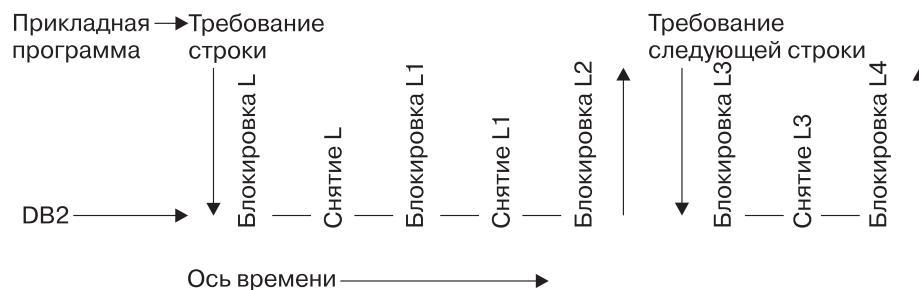


Рисунок 100. Как устанавливает блокировки программа, использующая изоляцию RS.  
Блокировки L2 и L4 сохраняются до выполнения программой операции принятия.  
Прочие блокировки не сохраняются.

Прикладные программы, использующие изоляцию "стабильность чтения" (RS), могут оставлять строки или страницы заблокированными на большое время, особенно в распределенной среде.

Используя этот тип изоляции, предусмотрите частое выполнение принятия.

**ISOLATION (CS)** Этот тип изоляции позволяет получить максимальную одновременность при сохранении целостности данных. Однако после того, как процесс перестает использовать строку или страницу, другой процесс может изменять эти данные. При CURRENTDATA(NO) процесс не должен оставлять строку или страницу, позволяя другому

процессу изменять данные. Если первый процесс возвращается к чтению той же строки или страницы, данные могут оказаться измененными. Учтите следующие возможные последствия:

- Для табличных пространств, созданных с LOCKSIZE ROW, PAGE или ANY, данные могут оказаться изменены даже во время выполнения одного оператора SQL, если этот оператор читает одну строку несколько раз. В следующем примере:

```
SELECT * FROM T1  
WHERE COL1 = (SELECT MAX(COL1) FROM T1);
```

данные, прочитанные внутренним SELECT, могут быть изменены другой транзакцией перед тем как они будут прочитаны внешним SELECT. Следовательно, этот запрос может вернуть строку, которая уже не содержит максимальное значение COL1.

- В другом случае, если процесс считывает строку и впоследствии возвращается к этой строке, чтобы обновить ее, эта строка может уже не существовать или может существовать в состоянии, не совпадающем с тем, в котором она была, когда ее содержимое было прочитано процессом прикладной программы. То есть другая прикладная программа могла удалить или обновить эту строку. **Если прикладная программа выполняет операции без указателя для строки, на которой находится указатель, убедитесь, что эта прикладная программа допускает возникновение условий “не найден”.**

Аналогично предположим, что другая прикладная программа обновляет строку после того, как данная программа прочитает ее. Если данный процесс впоследствии возвращается к этой строке, чтобы обновить ее, используя ранее считанное из нее значение, он в результате сотрет данные, записанные другим процессом. **Если используется изоляция CS и выполняются обновления, процессу может потребоваться блокировка одновременных обновлений.** Один из способов сделать это – объявить указатель с условием FOR UPDATE OF.

**ISOLATION (UR)** Этот тип изоляции позволяет прикладной программе читать данные, устанавливая малое число блокировок, но с риском чтения непринятых данных. Изоляция UR применяется только к операциям в режиме только для чтения: SELECT, SELECT INTO или FETCH из таблицы результатов только для чтения.

**При чтении непринятых данных в них нельзя быть полностью уверенными.**

**Пример:** Прикладная программа отслеживает перемещение деталей от одного рабочего места к другому вдоль сборочной линии. Когда деталь перемещается с одного места на другое, прикладная программа уменьшает счетчик деталей для первого места и увеличивает счетчик деталей для второго. Теперь вы хотите запросить число деталей на всех рабочих местах при одновременном выполнении этой прикладной программы.

Что может случиться, если запрос читает данные, которые были изменены этой прикладной программой, но не были приняты?

Если прикладная программа сначала уменьшает число в первой записи, а затем увеличивает число во второй записи, *ваш запрос может получить меньшее значение, чем нужно*.

Если прикладная программа сначала увеличивает число во второй записи, а затем уменьшает число в первой записи, *ваш запрос может учесть добавленную величину дважды*.

Если такая ситуация может возникнуть и она недопустима, не используйте изоляцию UR.

**Ограничения:** Нельзя использовать изоляцию UR для перечисленных ниже типов операторов. Если при связывании задано ISOLATION(UR), а в операторе не задано условие WITH RR или WITH RS, DB2 использует изоляцию CS для:

- Операторов INSERT, UPDATE и DELETE
- Всех указателей, объявленных с FOR UPDATE OF

**Когда можно использовать изоляцию "чтение непринятого" (UR)?**

Возможно в таких случаях:

- **Когда ошибки не могут возникнуть.**

**Пример:** Имеется таблица ссылок, например, таблица описаний деталей по номерам деталей. Она обновляется редко и чтение непринятых обновленных данных, скорее всего, не даст информацию, менее правильную, чем данные, которые можно было прочитать из этой таблицы на 5 секунд раньше. Используйте в этом случае чтение с изоляцией ISOLATION(UR).

**Пример:** Имеется таблица сотрудников фирмы Spiffy Computer. В целях защиты обновлять информацию в этой таблице разрешено только членам одного подразделения фирмы. И только они могут обращаться с запросами ко всей таблице. Этим пользователям несложно организовать свою работу так, чтобы обращаться с запросами только в те моменты времени, когда не производятся обновления данных, и, значит, они могут выполняться с изоляцией UR.

- **Когда ошибки допустимы.**

**Пример:** В фирме Spiffy Computer нужно проводить некоторый статистический анализ информации о сотрудниках. Типичный запрос: "Какое среднее значение оклада в зависимости от пола сотрудника внутри группы по уровню образования?" Было решено, что возможное чтение непринятых данных не должно сильно повлиять на искомое среднее значение, поэтому используется изоляция UR.

- **Когда данные уже содержат несовместимую информацию.**

**Пример:** Фирма Spiffy получает информацию о перспективных каналах сбыта продукции из различных источников. Эти данные часто противоречивы или неправильны, но конечные пользователи умеют использовать такие данные. Обращение к несовместимым данным к таблице данных о сбыте продукции не ухудшает ситуации.

**НЕ используйте изоляцию "чтение непринятого" (UR):**

**Когда вычисления должны быть согласованы.**  
**Когда ответ должен быть точным.**  
**Когда нет уверенности, что это не может нанести вред.**

**Ограничения на одновременный доступ:** Прикладная программа, использующая изоляцию UR, не может выполняться одновременно с утилитами, бронирующими все заявочные классы. Кроме этого, такая прикладная программа должна устанавливать следующие блокировки:

- Специальную блокировку для массового удаления, устанавливаемую в режиме S для таблицы или табличного пространства назначения. “Массовое удаление” – это оператор DELETE без условия WHERE; такая операция должна устанавливать блокировку в режиме X и, следовательно, не может выполняться одновременно с другими процессами.
- Блокировку IX для любого табличного пространства, используемого в базе данных рабочих файлов. Эта блокировка предотвращает удаление табличного пространства во время выполнения этой прикладной программы.
- Блокировки больших объектов и блокировку табличного пространства большого объекта, если читаются значения большого объекта. Если блокировка большого объекта невозможна из-за того, что для этого большого объекта другой прикладной программой сохраняется несовместимая блокировка, процесс, читающий с изоляцией UR, пропускает этот большой объект и переходит к следующему большому объекту, соответствующему критерию запроса.

## Опция CURRENTDATA

Влияние опции CURRENTDATA различается в зависимости от того, используется локальный или удаленный доступ:

- Для **локального** доступа эта опция задает, должны ли данные, на которые указывает указатель, оставаться одинаковыми (или “согласованными с”) с данными в локальной базовой таблице. На указатели на данные в рабочем файле опция CURRENTDATA не влияет. Действие этой опции распространяется только на указатели только для чтения или **неоднозначные** указатели в планах или пакетах, связанных с изоляцией CS.

Указатель является “неоднозначным,” если DB2 не может определить, используется ли он для изменения или только для чтения. Если указатель, видимо, используется только для чтения, но динамический SQL мог бы изменять данные через этот указатель, он считается неоднозначным. Если вы при помощи CURRENTDATA укажете, что неоднозначный указатель является указателем только для чтения, а в действительности он используется динамическим SQL для изменения данных, вы получите ошибку. Дополнительную информацию о неоднозначных указателях смите в разделе “Проблемы с неоднозначными указателями” на стр. 380.

- Для запросов к **удаленной** системе опция CURRENTDATA влияет на неоднозначные указатели, использующие уровни изоляции RR, RS, или CS. Для неоднозначных указателей она включает или выключает блочную выборку. (Указатели только для чтения при изоляции UR всегда используют блочную выборку.) Включение блочной выборки улучшает

производительность, но это означает, что указатель не будет согласован с базовой таблицей на удаленной системе.

**Локальный доступ:** При локальном доступе опция **CURRENTDATA(YES)** означает, что пока указатель указывает на какие-то данные, эти данные не могут изменяться. Если указатель указывает на данные в локальных базовой таблице или индексе, данные, возвращаемые с использованием этого указателя, согласованы с содержимым этих таблицы или индекса. Если указатель указывает на данные в рабочем файле, данные, возвращаемые с использованием этого указателя, согласованы только с содержимым этого рабочего файла и не обязательно согласованы с содержимым базовых таблицы или индекса.

На рис. 101 показана блокировка при использовании CURRENTDATA(YES).

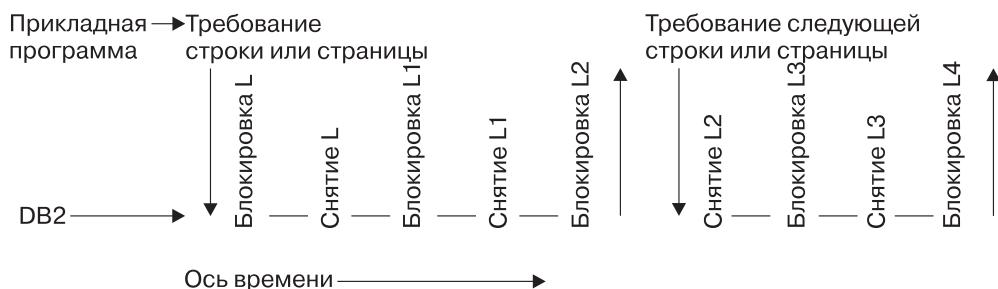


Рисунок 101. Как устанавливает блокировки прикладная программа, использующая изоляцию CS с CURRENTDATA(YES). На этом рисунке показано обращение к базовой таблице. Блокировки L2 и L4 освобождаются после того, как DB2 переходит к следующей строке или странице. Когда прикладная программа выполняет принятие, освобождается последняя блокировка.

Как и при использовании рабочих файлов, если указатель использует параллелизм запросов, данные не обязательно будут согласованы с содержимым таблицы или индекса, независимо от того, используется ли рабочий файл. Следовательно, опция CURRENTDATA не влияет на обращения к рабочим файлам или на запросы только для чтения в режиме параллелизма.

Если используется параллелизм, но необходимо обеспечить согласованность данных, можно:

- Запретить параллелизм (использовать SET DEGREE = '1' или задать при связывании DEGREE(1))
- Использовать изоляцию RR или RS (при этом можно использовать параллелизм)
- Использовать оператор LOCK TABLE (при этом можно использовать параллелизм)

Для локального доступа опция **CURRENTDATA(NO)** сходна с опцией CURRENTDATA(YES), за исключением случаев, когда указатель используется для доступа к основной таблице (а не таблице результата в рабочем файле). В этих случаях опция CURRENTDATA(NO) не обеспечивает согласованности указателя и базовой таблицы, в то время как опция CURRENTDATA(YES) обеспечивает такую согласованность.

**Удаленный доступ:** Для доступа к удаленной таблице или индексу опция CURRENTDATA(YES) отключает блочную выборку для неоднозначных указателей. Возвращаемые при использовании неоднозначного указателя данные согласованы с содержимым удаленной таблицы или индекса. Дополнительную информацию о влиянии опции CURRENTDATA на блочную выборкусмотрите в разделе “Использование блочной выборки” на стр. 424.

**Избежание блокировок:** Использование опции CURRENTDATA(NO) дает гораздо большие возможности избегать блокировок. DB2 может определить, были ли принятые данные в строке или странице. Если это так, DB2 вообще не нужно блокировать данные. Прикладной программе возвращаются незаблокированные данные, которые могут быть изменены, пока указатель указывает на эту строку. (Для операторов SELECT, не использующих указатели, например, возвращающих одну строку, блокировка строки не используется, если в операторе не задано условие WITH RS или WITH RR.)

Чтобы наилучшим образом использовать преимущество этого метода избежания блокировок, убедитесь, что все одновременно обращающиеся к данным прикладные программы часто выдают операторы COMMIT.

На рис. 102 показано, как DB2 может избежать блокировок, а в Табл. 39 перечислены факторы, влияющие на избежание блокировок.

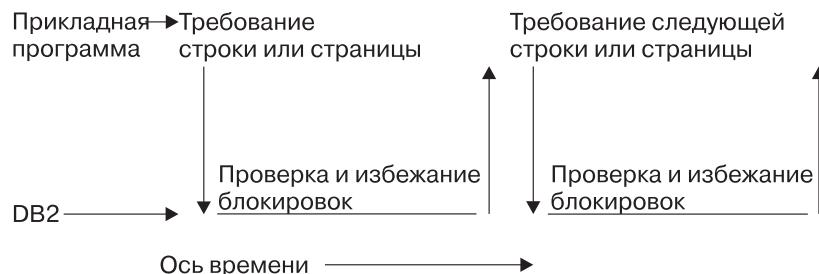


Рисунок 102. Наилучший случай избежания блокировок при использовании изоляции CS с опцией CURRENTDATA(NO). На этом рисунке показано обращение к базовой таблице. Если DB2 приходится использовать блокировки, эти блокировки освобождаются при переходе к следующей строке или странице или при выполнении прикладной программой принятия (так же, как для CURRENTDATA(YES)).

Таблица 39 (Стр. 1 из 2). Факторы избежания блокировок. “Возвращенные данные” – это данные, удовлетворяющие критерию поиска. “Отвергнутые данные” – это данные, не удовлетворяющие этому критерию.

Изоляция	CURRENTDATA	Тип указателя	Избежать блокировок возвращ. данных?	Избежать блокировок отвергнутых данных?
UR	Неприменим	Только для чтения	Неприменим	Неприменим
CS	Да	Только для чтения	Нет	Да
		Обновляемый		
		Неоднозначный		
	Нет	Только для чтения	Да	
		Обновляемый	Нет	
		Неоднозначный	Да	

Таблица 39 (Стр. 2 из 2). Факторы избежания блокировок. “Возвращенные данные” – это данные, удовлетворяющие критерию поиска. “Отвергнутые данные” – это данные, не удовлетворяющие этому критерию.

Изоляция	CURRENTDATA	Тип указателя	Избежать блокировок возвращ. данных?	Избежать блокировок отвергнутых данных?
RS	Неприменим	Только для чтения	Нет	Да
		Обновляемый		
		Неоднозначный		
RR	Неприменим	Только для чтения	Нет	Нет
		Обновляемый		
		Неоднозначный		

**Проблемы с неоднозначными указателями:** Как показано в Табл. 39 на стр. 379, неоднозначные указатели иногда могут мешать DB2 использовать методы избежания блокировок. Однако при неправильном использовании неоднозначного указателя ваша программа может получить SQLCODE –510:

- При связывании плана или пакета использована опция CURRENTDATA(NO)
- Оператор OPEN CURSOR выполнен *до того*, как подготовлен динамический оператор DELETE WHERE CURRENT OF для этого указателя
- Для открытого указателя выполняется одно из следующих условий:
  - Для оператора успешно используется избежание блокировки.
  - Используется параллелизм запросов.
  - Это распределенный указатель и используется блочная выборка.

Во всех случаях хороший стиль программирования – устранять неоднозначность указателя, объявляя его с условием FOR FETCH ONLY или с условием FOR UPDATE OF.

### Разные опции для плана и для пакета

План, при связывании которого использовался один набор опций, может содержать с своем списке пакетов пакеты, при связывании которых использовались другие наборы опций. В целом операторы в DBRM, связанном как пакет, используют опции с которыми связан этот пакет, а операторы в DBRM, связанном с планом, используют опции с которыми связан этот план.

Например, значение опции CURRENTDATA для плана не влияет на пакеты, выполняемые под этим планом. Если опция CURRENTDATA не задана явно при связывании пакета, используется значение по умолчанию CURRENTDATA(YES).

Немного другие правила используются для опций связывания RELEASE и ISOLATION. Значения этих двух опций задаются при установлении блокировки ресурса и действуют обычно пока блокировка не будет освобождена. Но может возникнуть конфликт, если оператор, связанный с одной парой

значений этих опций, запрашивает блокировку ресурса, который уже заблокирован оператором, связанным с другой парой этих значений. DB2 разрешает конфликт, меняя значения опций на допустимые значения, вызывающие наибольшую длительность удержания блокировки.

При конфликте между опциями RELEASE(COMMIT) и RELEASE(DEALLOCATE) используется значение RELEASE(DEALLOCATE).

В Табл. 40 показано, как разрешаются конфликты между уровнями изоляции. В первом столбце представлен существующий уровень изоляции, а в остальных столбцах показано, что происходит, если другой процесс прикладной программы запрашивает новый уровень изоляции.

*Таблица 40. Разрешение конфликтов между уровнями изоляции*

	<b>UR</b>	<b>CS</b>	<b>RS</b>	<b>RR</b>
<b>UR</b>	Не прим.	CS	RS	RR
<b>CS</b>	CS	Не прим.	RS	RR
<b>RS</b>	RS	RS	Не прим.	RR
<b>RR</b>	RR	RR	RR	Не прим.

### **Действие условия WITH HOLD для указателя**

Позиция указателя, определенного как указатель WITH HOLD, сохраняется после точки принятия. Поэтому блокировки и заявки, необходимые для сохранения этой позиции указателя, не освобождаются немедленно, даже если они установлены с ISOLATION(CS) или RELEASE(COMMIT).

Для блокировок и заявок, необходимых для сохранения положения указателя , описанные выше правила отличаются следующим образом:

**Блокировки страниц и строк:** Если в поле RELEASE LOCKS панели установки DSNTIP4 задано значение NO (как описано в разделе 5 тома 2 руководства *DB2 Administration Guide*), блокировка страницы или строки, если ее не удалось избежать, используя избежание блокировок, сохраняется после точки принятия. Такая блокировка не требуется для сохранения позиции указателя, но опция NO обеспечивает совместимость с прикладными программами, которые могут быть рассчитаны на такие блокировки. Однако при этом X или U-блокировка понижается до S-блокировки. (Поскольку изменения приняты, монопольное управление больше не требуется.) После точки принятия блокировка освобождается в следующей точке принятия, и в результате никакой указатель более не указывает на данную страницу или строку.

Значение YES поля RELEASE LOCKS означает, что блокировки страниц или строк не сохраняются после точки принятия.

**Блокировки таблиц, табличных пространств и DBD:** Все необходимые блокировки сохраняются после точки принятия. После этого они освобождаются в соответствии с опцией RELEASE, с которой они были установлены: для значения COMMIT – в следующей точке принятия после

закрытия указателя; для значения DEALLOCATE – при завершении прикладной программы.

**Заявки:** Все заявки всех заявочных классов сохраняются после точки принятия. Они освобождаются в последующей точке принятия, в которой все фиксированные указатели перемещены с этого объекта или закрыты.

## Задание уровня изоляции в операторе SQL

**Условие WITH:** Используя условие WITH в некоторых операторах SQL, можно переопределить уровень изоляции, с которым связан план или пакет.

**Пример:** Оператор:

```
SELECT MAX(BONUS), MIN(BONUS), AVG(BONUS)
  INTO :MAX, :MIN, :AVG
    FROM DSN8610.EMP
      WITH UR;
```

находит максимальное, минимальное и среднее значение премии в таблице примера, содержащей информацию о сотрудниках. Этот оператор выполняется с изоляцией "чтение непринятого" (UR), независимо от значения ISOLATION, с которым связан план или пакет, содержащий этот оператор.

**Правила использования условия WITH:** Условие WITH:

- Может использоваться в следующих операторах:
  - операторы выборки
  - SELECT INTO
  - поиск с удалением
  - INSERT из подвыборки
  - поиск с обновлением
- Не может использоваться в подзапросах.
- Может задавать уровни изоляции, допустимые для данного оператора. (Например, поскольку WITH UR применимо только к операциям только для чтения, такое значение нельзя использовать в операторе INSERT).
- Переопределяет уровень изоляции плана или пакета только для оператора, в котором оно задано.

**Использование KEEP UPDATE LOCKS в условии WITH:** В операторе SELECT с FOR UPDATE OF можно использовать условие KEEP UPDATE LOCKS. Его можно использовать только с условием WITH RR или WITH RS. При использовании этого условия DB2 будет устанавливать для всех выбранных страниц или строк блокировку X вместо блокировки U или S.

Пример:

```
SELECT ...
  FOR UPDATE OF WITH RS KEEP UPDATE LOCKS;
```

При использовании изоляции "стабильность чтения" (RS) строка или страница, отвергнутая на втором шаге обработки, будет заблокирована в режиме X, хотя она и не возвращена прикладной программе.

При использовании изоляции "многократное чтение" (RR) DB2 устанавливает блокировки X для всех строк или страниц, удовлетворяющих условию выбора.

Все блокировки X сохраняются до выполнения прикладной программой принятия. Хотя эта опция может ухудшить одновременность, она предотвращает некоторые типы тупиковых ситуаций и может обеспечить более упорядоченное обращение к данным.

## Оператор LOCK TABLE

Информацию об использовании оператора LOCK TABLE для дополнительной таблицы смотрите в разделе “Команда LOCK TABLE” на стр. 392.

### Назначение оператора LOCK TABLE

Этот оператор переопределяет правила DB2 для выбора исходных атрибутов блокировки. Вот два примера:

```
LOCK TABLE имя-таблицы IN SHARE MODE;  
LOCK TABLE имя-таблицы PART n IN EXCLUSIVE MODE;
```

Как описано ниже, при выполнении такого оператора блокировка запрашивается немедленно, если подходящая блокировка еще не существует. Опция связывания RELEASE определяет, когда будут освобождены блокировки, установленные оператором LOCK TABLE или LOCK TABLE с PART.

Оператор LOCK TABLE можно использовать для любых таблиц, включая дополнительные таблицы табличных пространств LOB. Информацию о блокировке дополнительных таблиц смотрите в разделе “Команда LOCK TABLE” на стр. 392.

Оператор LOCK TABLE не влияет на блокировки, установленные на удаленном сервере.

### Действие оператора LOCK TABLE

В Табл. 41 показаны режимы блокировок, устанавливаемых в сегментированных и несегментированных табличных пространствах для режимов SHARE и EXCLUSIVE оператора LOCK TABLE. Дополнительные таблицы табличных пространств LOB считаются несегментированными табличными пространствами и для них применяются те же правила установления блокировок.

Таблица 41. Режимы блокировок, устанавливаемых оператором LOCK TABLE. LOCK TABLE для разделов выполняется так же, как для несегментированных табличных пространств.

LOCK TABLE IN	Несегментированное табличное пространство	Сегментированное табличное пространство	
	Таблица	Табличное пространство	
EXCLUSIVE MODE	X	X	IX
SHARE MODE	S или SIX	S или SIX	IS

**Примечание:** Блокировка SIX устанавливается, если процесс уже сохраняет блокировку IX. SHARE MODE не влияет на процесс, уже имеющий блокировку с режимом SIX, U или X.

## **Рекомендации по использованию LOCK TABLE**

Используйте оператор LOCK TABLE, чтобы запретить другим процессам прикладных программ изменять какие-либо строки в таблице или разделе, к которым обращается данный процесс. Предположим, например, что процесс обращается к нескольким таблицам. Допустимы одновременные обновления во всех таблицах кроме одной — для нее требуется изоляция RR или RS. В такой ситуации есть несколько способов поведения:

- Связать план прикладной программы с изоляцией RR или RS. Но это повлияет на все используемые программой таблицы и может ухудшить одновременность.
- Спроектировать прикладную программу так, чтобы она использовала пакеты и обращалась к этой конкретной таблице только из нескольких пакетов. Связать эти пакеты с изоляцией RR или RS, а план с изоляцией CS. Только для таблиц, к которым обращаются эти пакеты, будет использовать изоляция RR или RS.
- Добавить условие WITH RR или WITH RS в операторы, которые должны выполняться с изоляцией RR или RS. Изоляция для операторов, в которых не используется условие WITH, определяется опцией связывания ISOLATION.
- Связать план прикладной программы с изоляцией CS и выполнить оператор LOCK TABLE для этой конкретной таблицы. (Если в этом же табличном пространстве есть другие таблицы, учтите приведенное ниже предостережение.) Оператор LOCK TABLE запрещает другим процессам производить в таблице любые изменения, обеспечивая для конкретной таблицы степень изоляции, даже большую, чем изоляция "многократное чтение". Все таблицы в других табличных пространствах будут совместно использоваться в режиме одновременных обновлений.

**Предостережение об использовании оператора LOCK TABLE с простыми табличными пространствами:** Этот оператор блокирует все таблицы в простом табличном пространстве, даже если задано имя только одной таблицы. Ни один другой процесс не сможет обновлять это табличное пространство во время действия этой блокировки. Если это блокировка в монопольном режиме, ни один из других процессов не сможет читать это табличное пространство, за исключением процессов, выполняющихся с изоляцией UR.

**Дополнительные примеры использования оператора LOCK TABLE:** заблокировать таблицу или раздел, которые обычно используются совместно, может потребоваться по одной из следующих причин:

**Получение “снимка”** Если в единице работы нужно обращаться к целой таблице, какой она была в некоторый определенный момент времени, необходимо заблокировать одновременные изменения. Если другие процессы могут обращаться к таблице, используйте LOCK TABLE IN SHARE MODE. (Изоляции RR недостаточно — она блокирует только изменения строк и страниц, к которым уже было произведено обращение.)

**Избежание накладных расходов** Если нужно обновить большую часть таблицы, может оказаться эффективнее запретить одновременный доступ, чем блокировать каждую страницу при

обновлении и разблокировать ее при принятии. Используйте LOCK TABLE IN EXCLUSIVE MODE.

**Предотвращение истечения срока ожидания** Прикладная программа имеет высокий приоритет и не должна подвергаться риску истечения срока ожидания из-за конфликтов с другими процессами прикладных программ. В зависимости от того, выполняет ли эта прикладная обновления или нет, используйте LOCK IN EXCLUSIVE MODE или LOCK TABLE IN SHARE MODE.

## Пути доступа

Используемый путь доступа может влиять на режим, на размер и даже на объект блокировки. Например, оператор UPDATE, использующий сканирование табличного пространства, может потребовать X-блокировки для всего табличного пространства. Если обновляемые строки выбираются по индексу, тот же оператор может потребовать только блокировки IX для табличного пространства и X-блокировок для отдельных страниц или строк.

Используя оператор EXPLAIN, чтобы узнать выбранный для оператора SQL путь доступа, проверьте режим блокировки в столбце TSLOCKMODE полученной таблицы PLAN\_TABLE. Если таблица находится в несегментированном табличном пространстве или определена с LOCKSIZE TABLESPACE, там представлен режим блокировки табличного пространства. В противном случае там представлен режим блокировки таблицы.

### **Главные сведения о блокировках DB2:**

- Обычно не нужно явно выполнять в программе блокировку данных.
- DB2 гарантирует, что программа не получит непринятых данных, если только вы специально не разрешите это.
- Все обновляемые, вставляемые или удаляемые программой страницы или строки остаются заблокированными по крайней мере до конца единицы работы, независимо от уровня изоляции. Ни один другой процесс не сможет до этого каким-либо образом обратиться к этому объекту, если вы специально не разрешите это обращение для этого процесса.
- Для улучшения одновременности чаще выполняйте принятие. Определите точки в программе, в которых измененные данные находятся в непротиворечивом состоянии. В этих точках выполните:

#### **TSO, Batch и CAF**

Оператор SQL COMMIT

#### **IMS**

Вызов CHKP или SYNC или (для транзакций одиночного режима) вызов GU для PCB ввода–вывода

#### **CICS**

Команду SYNCPOINT.

- Для улучшения одновременности используйте при связывании опцию ACQUIRE(USE).
- При связывании плана или пакета задайте ISOLATION (обычно RR, RS или CS).
  - При использовании изоляции RR (многократное чтение) все страницы или строки, к которым производится обращение, блокируются до следующей точки принятия. (Информацию о блокировках позиции указателя для указателей, определенных с WITH HOLD, смотрите в разделе “Действие условия WITH HOLD для указателя” на стр. 381.)
  - При использовании изоляции RS (стабильность чтения) все выбранные страницы или строки блокируются до следующей точки принятия. (Информацию о блокировках позиции указателя для указателей, определенных с WITH HOLD, смотрите в разделе “Действие условия WITH HOLD для указателя” на стр. 381.)
  - При использовании изоляции CS (стабильность на уровне указателя) могут блокироваться только страницы или строки, к которым производится обращение в настоящий момент, и может использоваться избежание этих блокировок. (Можно обращаться к одной странице или строке для каждого открытого указателя.)

- Можно также задать изоляцию для конкретных операторов SQL, используя условие WITH.
- Тупиковая ситуация может возникнуть, если каждый из двух процессов блокирует ресурс, требуемый второму процессу. Один из процессов выбирается в качестве “жертвы” и для его единицы работы выполняется откат и выдается код ошибки SQL.
- Можно заблокировать все несегментированное табличное пространство или всю таблицу в сегментированном табличном пространстве, используя оператор LOCK TABLE:
  - Чтобы разрешить другим пользователям получать, но не обновлять, удалять или вставлять данные, используйте оператор:  
`LOCK TABLE имя-таблицы IN SHARE MODE`
  - Чтобы запретить другим пользователям любым образом обращаться к строкам (за исключением случаев, когда они используют изоляцию UR), используйте оператор:  
`LOCK TABLE имя-таблицы IN EXCLUSIVE MODE`

Рисунок 103. Сводная информация о блокировках DB2

## Блокировка больших объектов

Процесс блокировки больших объектов описывается отдельно от блокировок транзакций, потому что предназначение блокировок больших объектов отличается от предназначения обычных блокировок транзакций.

**Термин:** Блокировка, производимая над значением большого объекта в табличном пространстве большого объекта называется **блокировкой большого объекта**.

В этом разделе описаны следующие темы:

- “Связь между блокировками транзакций и блокировками больших объектов”
- “Иерархия блокировок больших объектов” на стр. 390
- “Режимы блокировки больших объектов и табличных пространств больших объектов” на стр. 390
- “Длительность блокировок” на стр. 391
- “Когда блокировка табличного пространства большого объекта не производится” на стр. 392
- “Команда LOCK TABLE” на стр. 392

## Связь между блокировками транзакций и блокировками больших объектов

Как описано в разделе Раздел 5 (Том 2) книги *DB2 Administration Guide*, значения столбцов большого объекта хранятся в другом табличном пространстве (табличном пространстве большого объекта) по отношению к значениям основной таблицы. Программа, которая читает или изменяет строку в таблице, содержащей столбцы большого объекта, получает свои обычные

блокировки транзакций основной таблицы. Блокировки основной таблицы также управляют одновременными обращениями к табличному пространству большого объекта. Если блокировки базовой таблицы не устанавливаются, как в случае с ISO(UR), DB2 поддерживает согласованность данных, используя блокировки табличного пространства большого объекта. Даже когда блокировки запрашиваются для базовой таблицы, DB2 все равно получает блокировки для табличного пространства больших объектов.

DB2 получает также блокировки табличного пространства большого объекта и значений большого объекта, хранимых в этом табличном пространстве, но основными задачами этих блокировок являются:

- Определить, можно ли снова использовать пространство удаленного большого объекта для вставленного или обновленного большого объекта.

Место удаленного большого объекта не используется снова до тех пор, пока остается хотя бы одна возможность (включая сохраняемые локаторы) прочесть большой объект и для операции удаления не выполнено принятие.

- Предотвратить освобождение пространства большого объекта, который читается в данный момент.

Большой объект может быть удален с точки зрения одной из программ, в то время как другая программа читает его. Эта программа читает большой объект, поскольку все читающие объект программы, включая и те, которые используют уровень изоляции "чтение непринятого", запрашивают S-блокировки на большие объекты, чтобы предотвратить освобождение памяти читаемого ими большого объекта. Эта блокировка не снимается до принятия. Установленный локатор также удерживает блокировки большого объекта и его табличного пространства до принятия.

В целом, основное назначение блокировок больших объектов – управлять пространством, используемым для больших объектов, и гарантировать, что читающие большой объект не читают на самом деле частично обновленный объект. Программы должны освобождать установленные локаторы, чтобы можно было снова использовать пространство.

В Табл. 42 описана связь между действиями над значением большого объекта и ассоциированным табличным пространством большого объекта и полученными блокировками большого объекта.

*Таблица 42 (Стр. 1 из 2). Блокировки для действий над большими объектами. Эта таблица не учитывает большие блокировки, которые могут быть установлены из-за LOCKSIZE TABLESPACE, оператора LOCK TABLE или из-за расширения блокировки.*

Действие над значением большого объекта	Блокировка табличного пространства большого объекта	Блокировка большого объекта	Комментарий
Чтение (включая "чтение непринятого")	IS	S	Предотвращает повторное использование памяти, пока большой объект читается или пока локаторы ссылаются на большой объект

*Таблица 42 (Стр. 2 из 2). Блокировки для действий над большими объектами. Эта таблица не учитывает большие блокировки, которые могут быть установлены из-за LOCKSIZE TABLESPACE, оператора LOCK TABLE или из-за расширения блокировки.*

<b>Действие над значением большого объекта</b>	<b>Блокировка табличного пространства большого объекта</b>	<b>Блокировка большого объекта</b>	<b>Комментарий</b>
Вставка	IX	X	Запрещает другим процессам просмотр неполного большого объекта
Удаление	IS	S	Чтобы удерживать пространство на случай отката удаления. (X устанавливается для строки или страницы таблицы). Память нельзя использовать снова, пока удаление не будет подтверждено и не останется ни одного процесса, читающего большой объект.
Обновление	IS→IX	Две блокировки больших объектов: S–блокировка для удаления и X–блокировка для вставки.	Действие заключается в удалении с последующей вставкой.
Изменение значения на пустое или значение нулевой длины	IS	S	Нет вставки, только удаление.
Изменение пустого большого объекта или большого объекта нулевой длины до значащего	IX	X	Нет удаления, только вставка.

**программы, использующие "чтение непринятого":** Когда программа читает строки, используя чтение непринятого или отмену блокировки, никаких блокировок страницы или строки базовой таблицы не производится. Следовательно эта программа должна выполнить S–блокировку большого объекта, чтобы гарантировать, что они не читают неполный большой объект или несоответствующее основной строке значение большого объекта.

## Иерархия блокировок больших объектов

Как существует иерархическая взаимосвязь между блокировками страниц (или строк) и блокировками табличного пространства, так существует и иерархическая взаимосвязь между блокировками больших объектов и блокировками табличных пространств больших объектов. Если табличное пространство большого объекта блокировано большой блокировкой, блокировки больших объектов не выполняются. В среде совместного использования данных блокировка табличного пространства большого объекта используется для определения, должна ли блокировка большого объекта быть распространена за пределы локальной IRLM.

## Режимы блокировки больших объектов и табличных пространств больших объектов

### Режимы блокировок больших объектов

Возможны следующие режимы блокировок больших объектов:

**S (SHARE)** Владелец блокировки и любой действующий в то же время процесс могут читать, обновлять или удалять блокированный большой объект. Параллельные процессы могут получить S-блокировку на большой объект. S-блокировка предназначена для резервирования пространства, занятого большим объектом.

**X (EXCLUSIVE)** Владелец блокировки может читать или изменять блокированный большой объект. Параллельные процессы не могут обращаться к большому объекту.

### Режимы блокировок табличного пространства большого объекта

Возможны следующие режимы блокировок табличного пространства больших объектов:

**IS (INTENT SHARE)** Владелец блокировки может заменять значение на пустое значение или на значение нулевой длины или читать или удалять большие объекты в табличном пространстве большого объекта. Параллельные процессы могут читать, и изменять большие объекты в том же табличном пространстве. Владелец блокировки получает блокировку большого объекта для любого чтения или удаления данных.

**IX (INTENT EXCLUSIVE)** Владелец блокировки и параллельные процессы могут читать и изменять данные в табличном пространстве большого объекта. Владелец блокировки получает блокировку большого объекта для любых данных, к которым получает доступ.

**S (SHARE)** Владелец блокировки и любые другие параллельные процессы могут читать и удалять большие объекты в табличном пространстве большого объекта. Владелец блокировки не нуждается в блокировках больших объектов.

**SIX (SHARE вместе с INTENT EXCLUSIVE)** Владелец блокировки может читать и изменять данные в табличном пространстве большого объекта. Если владелец блокировки вставляет данные

(оператором INSERT или UPDATE), ему дается блокировка большого объекта. Параллельные процессы могут читать или удалять данные в табличном пространстве большого объекта (или заменять значение на пустое или большой объект нулевой длины).

**X (EXCLUSIVE)** Владелец блокировки может читать или изменять большие объекты в табличном пространстве большого объекта. Владелец блокировки не нуждается в блокировках больших объектов.

## Длительность блокировок

### Длительность блокировок табличных пространств больших объектов

Блокировка табличных пространств больших объектов устанавливается тогда, когда она нужна; то есть опция ACQUIRE команды BIND не действует для блокировки табличного пространства большого объекта. Блокировка табличного пространства большого объекта снимается в соответствии со значением опции RELEASE команды BIND (за исключением случая, когда указатель определен с условием WITH HOLD или используется локатор большого объекта).

### Длительность блокировок больших объектов

Блокировки больших объектов устанавливаются, когда они нужны, и обычно снимаются при принятии. Однако если значение большого объекта назначено локатору большого объекта, S–блокировка не снимается, пока программа не выполнит принятие.

Если программа использует HOLD LOCATOR, то локатор (и блокировка большого объекта) не освобождаются, пока не будет выполнен первый оператор принятия после оператора FREE LOCATOR, или пока поток не будет освобожден.

**Замечание о сохраняемых указателях:** Если указатель определен с условием WITH HOLD, блокировки больших объектов не снимаются при операциях принятия.

**Замечание об операторе INSERT с подвыбором:** Поскольку блокировки больших объектов не снимаются до принятия и поскольку блокировки защищают каждый столбец большого объекта в исходной таблице и в таблице назначения, возможно, что оператор, например, INSERT с подвыбором, работающий со столбцами больших объектов, может вызвать гораздо больше блокировок, чем подобный оператор, где не участвуют столбцы больших объектов. Чтобы предотвратить системные проблемы, вызванные слишком большим количеством блокировок, можно сделать следующее:

- Убедитесь, что у вас включено расширение блокировок для табличных пространств больших объектов, затрагиваемых оператором INSERT. Другими словами, задайте ненулевое значение LOCKMAX для этих табличных пространств больших объектов.
- Задайте для табличное пространство большого объекта значение TABLESPACE для параметра LOCKSIZE до выполнения INSERT с подвыбором.

- Используйте оператор LOCK TABLE для блокировки табличного пространства большого объекта.
- Увеличьте значение LOCKMAX для затронутых табличных пространств и проверьте достаточность предела пользовательских блокировок.
- Используйте операторы LOCK TABLE для участвующих вспомогательных таблиц.

## Когда блокировка табличного пространства большого объекта не производится

Блокировка табличного пространства большого объекта может вообще не производиться. Например, если строка удаляется из таблицы, а значение столбца большого объекта – пустое, табличное пространство большого объекта, ассоциированное с этим столбцом большого объекта, не блокируется. DB2 не обращается к табличному пространству большого объекта, если программа:

- Выбирает пустой большой объект или большой объект нулевой длины
- Удаляет строку, в которой большой объект пуст или имеет нулевую длину
- Вставляет пустой большой объект или большой объект нулевой длины
- Заменяет пустое значение или большой объект нулевой длины на пустой большой объект или большой объект нулевой длины

## Команда LOCK TABLE

В разделе “Оператор LOCK TABLE” на стр. 383 описывается, как и зачем использовать оператор LOCK TABLE для таблицы. Причины использования LOCK TABLE для вспомогательных таблиц могут несколько отличаться от тех же причин для обычных таблиц.

- Можно использовать LOCK TABLE для управления числом блокировок для вспомогательной таблицы.
- Можно использовать LOCK TABLE IN SHARE MODE, чтобы запретить другим программам вставлять большие объекты.

Для вспомогательных таблиц LOCK TABLE IN SHARE MODE не запрещает изменений во вспомогательной таблице. Этот оператор предотвращает вставку больших объектов во вспомогательную таблицу, но не запрещает удаление. Обновления тоже запрещены, за исключением случая записи пустого значения или строки нулевой длины.

- Можно использовать LOCK TABLE IN EXCLUSIVE MODE, чтобы запретить другим программам обращаться к большим объектам.

Для вспомогательных таблиц LOCK TABLE IN EXCLUSIVE MODE также не дает доступа для чтения непринятого.

- Любой из этих операторов устраняет необходимость низкоуровневых блокировок больших объектов.

---

## Глава 5–3. Планирование восстановления

Для восстановления, то есть возврата базы данных DB2 к последнему согласованному состоянию, нужно отменить все не принятые изменения данных, выполненные перед аварийным завершением программы или ошибкой системы. При этом нужно не помешать другим действиям системы.

Если ваша программа перехватит условия аварийного завершения, DB2 не узнает о случившемся и выполнит принятие. Так что если вы хотите, чтобы DB2 автоматически выполняла откат при аварийном завершении программы, не позволяйте программе и среде, в которой она запущена, перехватывать условие аварийного завершения. Например, если вы запускаете программу в Language Environment и хотите, чтобы DB2 автоматически выполняла откат при аварийном завершении программы, задайте опции запуска ABTERMENC(ABEND) и TRAP(ON).

*Единица работы* – это логически единая процедура, в которой есть шаги, изменяющие данные. Если все шаги завершаются успешно, изменения данных желательно сделать необратимыми. Но если на каком–то шаге произошел сбой, желательно вернуть все модифицированные данные к начальным значениям, которые они имели до начала процедуры.

Пусть, например, два сотрудника в таблице DSN8610.EMP обменялись кабинетами. Вам нужно поменять местами номера их рабочих телефонов в столбце PHONENO. Чтобы обновить номера телефонов, вы используете два оператора UPDATE. Эти два оператора, взятые вместе, составляют единицу работы. Важно, чтобы они оба завершились успешно. Если же успешно будет выполнен только один оператор, нужно выполнить откат для всех телефонных номеров, вернув их первоначальные значения, а уж потом повторять попытку обновления.

Когда единица работы завершается, снимаются все блокировки, которые она устанавливает по умолчанию, и становится возможным выполнение другой единицы работы.

Время обработки единицы работы в вашей программе определяет время, в течение которого DB2 запрещает другим пользователям доступ к заблокированным данным. Если несколько программ пытаются одновременно обратиться к одним и тем же данным, желательно, чтобы единицы работы этих программ были как можно короче, чтобы программы меньше мешали друг другу. В оставшейся части этой главы описывается функционирование единицы работы в различных средах. Дополнительную информацию о единице работы смотрите в Главе 2 руководства *DB2 SQL Reference* и Разделе 4 (Том 1) руководства *DB2 Administration Guide*.

---

### Единица работы TSO (пакетной и диалоговой)

Началом единицы работы считается первое обновление объекта DB2.

Единица работы заканчивается при наступлении одной из следующих ситуаций:

- Программа выполняет оператор COMMIT. К этому моменту ваша программа уже знает, что данные приведены в допустимое состояние; все изменения данных после предыдущего принятия выполнены правильно.
- Программа выполняет оператор ROLLBACK. К этому моменту ваша программа обнаружила, что изменения данных выполнены некорректно, и, следовательно, нежелательно делать эти изменения необратимыми.
- Программа завершает работу и возвращает управление процессору DSN, который возвращает управление программе TSO Terminal Monitor Program (TMP).

*Точка принятия* – это тот момент, когда вы выполняете оператор COMMIT или ваша программа успешно завершает работу. Оператор COMMIT желательно выполнять только тогда, когда вы уверены, что данные приведены в согласованное состояние. Пусть, например, банковская транзакция должна перевести средства со счета А на счет В. Эта транзакция вначале вычтет переводимую сумму из счета А, а затем добавит ее к счету В. Эти два события, взятые вместе, составляют единицу работы. Только после завершения обоих событий (но не раньше!) данные счетов окажутся в согласованном состоянии. После этого можно выполнить оператор COMMIT. Оператор ROLLBACK отменяет все изменения данных, произведенные со времени последнего принятия.

Вы не сможете соединиться с другой СУБД, пока не будет выполнен оператор COMMIT. Если в этот момент произойдет сбой системы, DB2 не узнает о завершении транзакции. В этом случае, как и в случае сбоя во время операции однофазного принятия для одной подсистемы, вы должны сами принять меры к поддержанию целостности данных.

При аварийном завершении вашей программы или при системном сбое DB2 отменяет непринятые изменения данных. Измененным данным будет возвращен их первоначальный вид, причем этот процесс не будет мешать другим действиям системы.

## Единицы работы в CICS

В среде CICS все действия, производимые вашей программой между двумя точками принятия, называются логической единицей работы (LUW) или просто единицей работы. Вообще говоря, единица работы есть *последовательность действий, до завершения которой ни одно из входящих в нее элементарных действий нельзя считать завершенным*. Например, такие действия, как уменьшение значения запасов и увеличение числа заказанных деталей на одну и ту же величину могут составлять единицу работы: пока не завершатся *оба* шага, ни один из шагов не должен считаться завершенным. (Если одно из действий будет выполнено, а другое нет, база данных утратит целостность или согласованность.)

Единица работы считается завершенной после обработки точки *принятия* или точки *синхронизации*, которая задается:

- По умолчанию в конце транзакции, сигналом которого служит команда CICS RETURN на высшем логическом уровне.
- В явном виде командами CICS SYNCPOINT, выдаваемыми программой в логически оправданные моменты транзакций.

- По умолчанию по вызову или команде завершения DL/I PSB (TERM).
- По умолчанию, когда пакетная программа DL/I вызывает контрольную точку DL/I. Это может произойти, когда пакетная программа DL/I пользуется общей базой данных с программами CICS при помощи средств совместного использования баз данных.

Рассмотрим пример учета запасов, в котором количество проданного товара вычитается из файла запасов и прибавляется к файлу заказов. Когда завершатся обе транзакции (и не ранее), данные в этих двух файлах окажутся согласованными, и программа сможет вызвать DL/I TERM или команду SYNCPOINT. Если один из шагов завершится неудачно, желательно вернуть данным значения, которые они имели до начала единицы работы. Другими словами, желательно откатить их до предыдущей точки совместимости. Для этого можно использовать команду SYNCPOINT с опцией ROLLBACK.

При помощи команды SYNCPOINT с опцией ROLLBACK вы можете отменить непринятые изменения данных. Например, программа, которая обновляет набор связанных строк, может иногда обнаружить ошибку после того, как уже обновит некоторые из них. При помощи команды SYNCPOINT с опцией ROLLBACK программа может отменить все изменения, не теряя управления.

В среде CICS операторы SQL COMMIT и ROLLBACK не допускаются. Вы можете координировать DB2 с функциями CICS, используемыми в программах, так что данные DB2 и не—DB2 будут согласованными.

При аварии системы DB2 отменяет не принятые изменения данных. Измененным данным будет возвращено их первоначальное состояние, причем этот процесс не будет мешать другим действиям системы. Иногда возврат DB2 к согласованному состоянию происходит не сразу. DB2 не начнет обрабатывать *неоднозначные данные* (данные, которые не являются ни принятыми, ни не принятыми), пока не будет перезапущена утилита подключения CICS. Чтобы гарантировать синхронизацию DB2 и CICS, перезапустите и DB2, и утилиту подключения CICS.

## **Единица работы в IMS (диалоговой)**

В среде IMS единица работы начинается:

- Когда запускается программа
- После завершения вызовов CHKP, SYNC, ROLL и ROLB
- При транзакциях в групповом режиме, когда вызов GU передается PCB ввода–вывода.

Единица работы заканчивается, когда:

- Программа выдает вызов CHKP или SYNC или (при транзакциях во множественном режиме) передает вызов GU PCB ввода–вывода. К этому моменту в работе программы данные будут согласованными. Все изменения в данных после предыдущей точки принятия прошли корректно.
- Программа вызывает ROLB или ROLL. К этому моменту в работе программы обнаружилось, что изменения в данных некорректны и, следовательно, не должны стать необратимыми.
- Программа завершает работу.

*Точка принятия* возникает при выполнении программы в результате одного из следующих четырех событий:

- Программа завершается нормально. Нормальное завершение программы всегда является точкой принятия.
- Программа выполняет *вызов контрольной точки*. Вызовы контрольной точки – это способ, при помощи которого программа явно указывает IMS, что достигла в своей работе точки принятия.
- Программа выполняет *вызов SYNC*. Вызов SYNC – это вызов системной службы Fast Path, который запрашивает обработку точки принятия. Использовать вызов SYNC можно только в программе Fast Path, которая не управляется сообщениями.
- Если программа обрабатывает сообщения как входные данные, точкой принятия может оказаться момент, когда программа получает новое сообщение. IMS считает новое сообщение началом новой единицы работы программы. Но если вы не зададите одиночный или множественный режим транзакций в операторе TRANSACT в макрокоманде APPLCTN в вашей программе, получение нового сообщения не будет восприниматься как точка принятия. Дополнительную информацию о макрокоманде APPLCTN смотрите в руководстве *IMS/ESA Installation Volume 2: System Definition and Tailoring*.
  - Если вы зададите *одиночный режим*, каждый выполняемый программой вызов на получение нового сообщения будет точкой принятия DB2. Задав одиночный режим, вы упрощаете восстановление; при аварийных завершениях вы сможете перезапускать программу начиная с последнего вызова нового сообщения. Когда IMS перезапустит программу, она начнется с обработки следующего сообщения.
  - Если вы зададите *множественный режим*, точками принятия будут моменты, когда программа выдает вызов контрольной точки или завершается нормально. Только в эти моменты работы программы IMS будет посыпать выходные сообщения программы в места назначения. Поскольку в программах множественного режима точек принятия меньше, чем в программах одиночного режима, их быстродействие несколько выше, чем у программ одиночного режима. Когда программа множественного режима завершается аварийно, IMS может перезапустить ее только начиная с вызова контрольной точки. Программе, возможно, придется повторно обработать несколько сообщений. Конкретное число зависит от того, насколько давно программа в последний раз выполняла вызов контрольной точки.

DB2 при выполнении программ одиночного и множественного режима производит некоторые действия, которые не производит IMS. Когда программа множественного режима выдает вызов на получение нового сообщения, DB2 производит проверку полномочий и закрывает все открытые указатели в программе.

В точке принятия:

- IMS и DB2 могут снять блокировки, которые программа поставила на данные после последней точки принятия. Это открывает другим программам и пользователям доступ к данным. (Однако, когда вы задаете

указатель WITH HOLD в программе BMP, DB2 сохраняет соответствующие блокировки, пока не закроется указатель или не завершится программа.)

- DB2 закрывает все открытые указатели, использовавшиеся программой. Ваша программа должна выдавать в очередь сообщений операторы закрытия указателей CLOSE CURSOR до вызова контрольной точки или GU, а не после.
- IMS и DB2 делают необратимыми произведенные программой изменения данных базы.

Если программа завершится аварийно до достижения точки принятия:

- И IMS, и DB2 отменят все изменения в базе данных, произведенные программой после последней точки принятия.
- IMS удалит все выходные сообщения, порожденные программой после последней точки принятия (для неэкспрессных PCB).

Если программа обрабатывает сообщения, порождаемые ей выходные сообщения к их конечным местам назначения посыпает IMS. Пока программа не достигнет точки принятия, IMS удерживает выходные сообщения программы во временном месте назначения. Если программа завершится аварийно, содержащаяся в ее сообщениях неточная информация не попадет к адресатам, сидящим за терминалами, и другим программам.

В среде IMS не допускаются операторы SQL COMMIT и ROLLBACK.

При сбое системы DB2 отменяет не принятые изменения данных. Измененным данным будет возвращен их первоначальный вид, причем этот процесс не будет мешать другим действиям системы. Иногда возврат DB2 к согласованному состоянию происходит не сразу. DB2 не начнет обрабатывать спорные данные, пока вы не перезагрузите IMS. Чтобы гарантировать синхронизацию DB2 и IMS, перезагрузите и DB2, и IMS.

## **Планирование программного восстановления: контрольная точка и перезапуск**

И IMS, и DB2 обрабатывают восстановление в программе IMS, обращающейся к данным DB2. IMS координирует этот процесс; роль DB2 заключается в восстановлении данных DB2.

Есть два вызова, при помощи которых IMS—программы могут упростить программное восстановление: вызов *символической контрольной точки* и вызов *перезапуска*.

### **Какую роль играет символическая контрольная точка**

Вызовы символической контрольной точки указывают IMS, что программа достигла точки синхронизации. Такие вызовы также отмечают те места в программе, с которых ее можно перезапускать.

Вызов СНКР заставляет IMS:

- Сообщить DB2, что произведенные вашей программой изменения в базе данных можно сделать необратимыми. DB2 делает необратимыми изменения данных DB2, а IMS делает необратимыми изменения данных IMS.

- Послать сообщение, содержащее заданное в вызове имя контрольной точки, оператору системной консоли и оператору главного терминала IMS.
- Вернуться к следующему входному сообщению, направляемому в область ввода–вывода программы, если программа обрабатывает входные сообщения. В программах MPP и транзакционно–ориентированных программах BMP вызов контрольной точки действует как вызов нового сообщения.
- Вновь зарегистрироваться в DB2, в результате чего будут переустановлены специальные регистры:
  - в CURRENT PACKAGESET будут записаны пробелы
  - в CURRENT SERVER будут записаны пробелы
  - в CURRENT SQLID будут записаны пробелы
  - в CURRENT DEGREE будет записано число 1

*Вашей программе надо восстановить значения этих регистров, если они будут нужны после контрольной точки.*

Программы, выдающие вызовы символической контрольной точки, могут задавать до семи областей данных программы, подлежащих восстановлению при перезапуске. Вызовы символической контрольной точки не поддерживают файлов OS/VS; если ваша программа обращается к файлам OS/VS, вы можете преобразовать их в GSAM и тогда пользоваться символическими контрольными точками. DB2 всегда восстанавливается до последней контрольной точки. Перезапускать программу надо с этой же точки.

### **Какую роль играет перезапуск**

Один из способов обеспечить перезапуск программы после аварийного завершения — использовать вместе с символическими контрольными точками вызов перезапуска (XRST). Он сохраняет области данных программы в том виде, в котором они были на момент ненормального завершения программы, и перезапускает программу с последнего вызова контрольной точки, выполненного программой до ненормального завершения.

## **Когда необходимы контрольные точки?**

Выполняя вызовы контрольных точек, программа высвобождает заблокированные ресурсы. Нужно ли это делать (и если нужно, то как часто), зависит от конкретной программы.

Вот основные типы программ, которые должны выполнять вызовы контрольных точек:

- Программы множественного режима
- Пакетно–ориентированные программы BMP
- Программы Fast Path, не управляемые сообщениями (для этих программ есть специальный вызов Fast Path, но они могут использовать вызовы символических контрольных точек)
- Большинство пакетных программ
- Программы, запускаемые в среде с совместным использованием данных. (В таких средах диалоговые и пакетные программы в различных системах IMS с общим или с разными процессорами могут одновременно обращаться к базам данных. В среде с совместным использованием

данных программы должны достаточно часто выполнять вызовы контрольной точки, чтобы не мешать доступу к базе данных программам из разных систем IMS.)

Не следует выполнять вызовы контрольных точек в:

- Программах одиночного режима
- Программах загрузки баз данных
- Программах, обращающихся к базе данных в режиме только для чтения (задаваемом при помощи опции обработки GO во время PSBGEN) и не настолько длинных, чтобы их было неудобно перезапустить с начала
- Программах, которые по своей сути должны пользоваться базой данных монопольно.

## Контрольные точки в программах MPP и транзакционно-ориентированных BMP

**Программы одиночного режима:** Для программ одиночного режима и вызовы контрольных точек, и вызовы приема сообщений (так называемые вызовы get-unique) создают точки принятия. Вызов контрольной точки позволяет получить входные сообщения; он заменяет вызовы get-unique. Программы BMP, обращающиеся к базам данных не DL/I, и программы MPP могут создавать точки принятия как при помощи вызовов get unique, так и при помощи вызовов контрольных точек.

Напротив, управляемые сообщениями программы BMP для создания точек принятия должны выполнять вызовы контрольных точек, а не вызовы get-unique, поскольку они могут перезапускаться только с контрольной точки. Если программа завершится аварийно после вызова get-unique, программа IMS отменит все изменения в базе данных до последней точки принятия, то есть до вызова get-unique.

**Программы множественного режима:** В BMP и MPP множественного режима точками принятия являются только вызовы контрольной точки, выдаваемые программой, и нормальное завершение программы. Если программа завершится аварийно, не выполнив ни одного вызова контрольной точки, IMS отменит произведенные программой изменения базы данных и созданные программой сообщения с момента начала работы программы. Если программа выполняла вызовы контрольных точек, IMS отменит произведенные программой изменения и созданные ею выходные сообщения, начиная с последнего вызова контрольной точки.

Выполняя вызовы контрольных точек в программах множественного режима, следует принимать во внимание три обстоятельства:

- Много ли времени займет отмена и восстановление обрабатываемой единицы.

Программа должна достаточно часто выполнять контрольные точки, чтобы облегчить отмену и восстановление.

- Надолго ли оказываются заблокированы ресурсы базы данных в DB2 и IMS.
- На какие группы вы хотите разбить выходные сообщения.

Вызовы контрольной точки разбивают выходные сообщения программы множественного режима. Программы должны достаточно часто выдавать контрольные точки, чтобы не создавать слишком громоздких выходных сообщений.

## Контрольные точки в пакетно–ориентированных BMP

Выполнять контрольные точки в пакетно–ориентированной BMP бывает важно по нескольким причинам:

- Чтобы принять изменения в базе данных
- Чтобы задать места, с которых программу можно перезапускать
- Чтобы высвобождать заблокированные данные DB2 и IMS, которые IMS включила в очередь для программы.

Кроме того, контрольные точки закрывают все открытые указатели, откуда следует, что вам придется повторно открывать и позиционировать нужные указатели.

Если пакетно–ориентированная BMP не будет достаточно часто выполнять вызовы контрольных точек, IMS может аварийно завершить эту BMP или другую программу по одной из следующих причин:

- Если программа BMP прочитает и обновит большое число записей между вызовами контрольных точек, она может монополизировать большую долю соответствующих баз данных и затянуть ожидание для других программ, нуждающихся в тех же сегментах. (Это не относится к BMP, имеющей опцию обработки GO. IMS не ставит в очередь сегменты для программы с этой опцией обработки.) Выполнение вызовов контрольных точек высвобождает поставленные в очередь для программы BMP сегменты и открывает к ним доступ другим программам.
- Когда IMS использует очередь для изоляции программы, значительное пространство расходуется на организацию очереди информации о сегментах, которые программа прочитала и обновила, и важно, чтобы требуемое пространство не превысило заданный в системе IMS размер. Если программа BMP поставит в очередь слишком много сегментов, требуемый для них объем памяти может превысить доступный объем памяти. Если это случится, IMS завершает программу с кодом аварийного завершения U0775. В этом случае надо увеличить частоту контрольных точек программы, и только после этого перезапускать ее. Объем доступной памяти задается при определении системы IMS.  
Дополнительную информацию смотрите в руководстве *IMS/ESA Installation Volume 2: System Definition and Tailoring*.

Когда вы выполняете вызов DL/I CHKP из программы, использующей базы данных DB2, IMS обрабатывает вызов CHKP для всех баз данных DL/I, а DB2 выполняет принятие на всех задействованных участках баз данных DB2. Никакой информации о контрольной точке для баз данных DB2 в файлы регистрации IMS и DB2 не записывается. При необходимости программа должна сама записать соответствующую информацию о базах данных DB2 при данной контрольной точке.

Для этого можно поместить информацию в области данных, включаемых в вызов DL/I CHKP. При таком подходе может понизиться быстродействие в процессе восстановления позиции в пределах базы данных DB2, поскольку

при вызове DL/I СНКР будет производиться принятие. Самым быстрым способом восстановления позиции в базе данных DB2 является использование индекса по таблице назначения, с ключом, который полностью соответствует всем столбцам предиката SQL.

Другое ограничение в обработке баз данных DB2 программой BMP – перезапускать программу можно только с последней контрольной точки, а не с любой контрольной точки, как в IMS.

## Задание частоты контрольных точек

Частоту контрольных точек в программе желательно задавать так, чтобы ее можно было легко изменить, если первоначально заданная частота не подойдет. Этого можно добиться, в частности, одним из следующих способов:

- Задайте в программе счетчик для учета затраченного времени и выполняйте вызов контрольной точки через определенный интервал времени.
- Задайте в программе счетчик для учета числа корневых сегментов, к которым обращается программа. Выполняйте вызов контрольной точки через определенное число корневых сегментов.
- Задайте в программе счетчик для учета числа обновлений, производимых вашей программой. Выполните вызов контрольной точки через определенное число обновлений.

---

## Единица работы в DL/I batch и в IMS batch

В этом разделе описывается, как скоординировать операции принятия и отката в среде DL/I batch, и как выполнять перезапуск и восстановление в среде IMS batch.

## Координация принятия и отката

DB2 координирует принятие и откатку для пакета DL/I, учитывая следующие обстоятельства:

- Изменения DB2 и DL/I принимаются в результате вызовов IMS СНКР. Однако позиции программы в базе данных в DL/I теряются. Кроме того, и в DB2 позиции программы в базе данных могут не сохраняться в следующих случаях:
  - Если вы не зададите для указателя опцию WITH HOLD, вы потеряете его позицию.
  - Если вы зададите для указателя опцию WITH HOLD, но программа управляет сообщениями, вы потеряете позицию этого указателя.
  - Если же вы зададите для указателя опцию WITH HOLD, но программа работает в пакете DL/I или в DL/I BMP, вы сохраните позицию этого указателя.
- DB2 автоматически отменяет все изменения, если программа завершается аварийно. Чтобы отменить изменения в DL/I, вы должны использовать пакетную утилиту резервирования DL/I.
- Нельзя использовать операторы SQL COMMIT и ROLLBACK DB2 в пакетной среде DL/I, поскольку рабочую единицу координирует IMS.

Попытка выполнить COMMIT приведет к коду ошибки SQLCODE –925 (SQLSTATE '2D521'); попытка выполнить ROLLBACK приведет к коду ошибки SQLCODE –926 (SQLSTATE '2D521').

- При сбое системы единица работы будет автоматически повторена после восстановления соединения DB2 и пакетной программы IMS. Если встречается неоднозначная единица работы, проблема решается во время соединения.
- Вы можете использовать вызовы отката IMS, ROLL и ROLB для отмены изменений DB2 и DL/I до последней точки принятия. Когда выполняется вызов ROLL, DL/I завершает вашу программу аварийно. Когда выполняется вызов ROLB, DL/I после вызова возвращает управление вашей программе.

Повлияют ли ROLL и ROLB на изменения DL/I в пакетной среде, зависит от используемого IMS журнала регистрации системы и заданных опций вывода, как указано в следующей сводке:

- Если задан вызов ROLL с регистрацией на ленте (заданная опция BKO не имеет значения) или с регистрацией на диске и опцией BKO=NO, DL/I не отменяет изменений и происходит аварийное завершение U0778. DB2 отменяет изменения до предыдущей контрольной точки.
- Если задан вызов ROLB с регистрацией на ленте (заданная опция BKO не имеет значения) или с регистрацией на диске и опцией BKO=NO, DL/I не отменяет изменений и в PCB возвращается код состояния AL. DB2 отменяет изменения до предыдущей контрольной точки.  
Поддержка DB2 DL/I вызывает аварийное завершение программы при сбое ROLB.
- Если задан вызов ROLL с регистрацией на диске и опцией BKO=YES, DL/I отменит изменения и произойдет аварийное завершение U0778. DB2 отменит изменения до предыдущей контрольной точки.
- Если задан вызов ROLB с регистрацией на диске и опцией BKO=YES, DL/I отменит изменения баз данных и вернет управление программе. DB2 отменит изменения до предыдущей контрольной точки.

## Использование ROLL

Вызова ROLL заставляет IMS завершить программу с кодом аварийного завершения пользователем U0778. При этом программа завершается без сохранения дампа памяти.

Выполняя вызов ROLL, вы задаете только одну опцию – саму функцию ROLL.

## Использование ROLB

Преимущества ROLB в том, что после выполнения ROLB IMS возвращает управление программе, так что она может продолжить работу. Опции ROLB:

- Вызываемая функция, ROLB
- Имя PCB ввода–вывода.

## **В пакетных программах**

Если журнал регистрации вашей системы IMS хранится на устройстве прямого доступа, и если опция запуска ВКО – Y (динамическая отмена), можно использовать вызов ROLB в пакетной программе. Вызов ROLB отменит изменения баз данных с последней точки принятия и вернет управление вашей программе. Вы не можете задавать адрес области ввода–вывода как одну из опций при вызове; если вы это сделаете, программа получит код состояния AD. Однако в программе бывают нужны PCB ввода–вывода. Задайте CMPAT=YES в параметре CMPAT оператора PSBGEN для PSB вашей программы. Дополнительную информацию по использованию параметра CMPATсмотрите в руководстве *IMS/ESA Utilities Reference: System*.

## **Перезапуск и восстановление в IMS (batch)**

В диалоговой системе IMS восстановление и перезапуск входят в саму систему IMS. Для пакетного региона восстановлением и перезапуском управляют операционные процедуры вашего каталога. Дополнительную информацию смотрите в руководстве *IMS/ESA Application Programming: Design Guide*.



---

## Глава 5–4. Планирование доступа к распределенным данным

Экземпляр DB2 for OS/390 может связываться с другими экземплярами этого же продукта, а также с некоторыми другими продуктами. В этой главе:

1. В разделе “Введение в доступ к распределенным данным” вводятся некоторые основные понятия. Ключевой момент – это существование двух методов доступа, которые мы будем рассматривать.
2. В разделе “Два метода программирования распределенных данных” на стр. 408 описывается методика разработки программ распределенного доступа, в качестве иллюстрации используется задание примера.
3. В разделе “Особенности программирования методов доступа” на стр. 412 обсуждается выбор метода доступа.
4. В разделе “Подготовка программ для доступа DRDA” на стр. 413. описана подготовка программ, использующих метод, который требует особых операций подготовки.
5. В разделе “Координация изменений нескольких источников данных” на стр. 416. приводятся особые соображения о возможных сложных ситуациях.
6. Главу завершает раздел “Различные вопросы, связанные с распределенными данными” на стр. 418.

---

### Введение в доступ к распределенным данным

**Определения:** *Распределенные данные* – это данные, находящиеся в какой-либо системе управления базами данных (СУБД), отличной от вашей локальной системы. *Локальная СУБД* – это та, на которой был связан ваш план программы. Все остальные СУБД называются *удаленными*.

В настоящей главе предполагается, что вам требуются услуги удаленной СУБД. Эта СУБД в данной ситуации работает как *сервер*, а локальная система – как *реквестер* или *клиент*.

Программа может быть одновременно соединена с многими СУБД; та из них, которая в текущий момент выполняет работу, называется *текущим сервером*. Когда работу выполняет локальная система, она также называется *текущим сервером*.

Удаленный сервер может быть действительно физически удален на тысячи километров. Однако это не обязательно; реально им может оказаться даже другая подсистема на том же самом компьютере, где выполняется локальная СУБД. Мы будем предполагать, что локальная СУБД – это экземпляр DB2 for OS/390. Удаленный сервер также может быть экземпляром DB2 for OS/390 или экземпляром одного из многих других продуктов.

СУБД, локальная или удаленная, известна системе DB2 по своему *имени положения*. Имя положения удаленной СУБД записывается в базе данных связи. (Дополнительную информацию об именах положения и о базе данных связи смотрите в разделе Раздел 3 книги *DB2 Installation Guide*.)

**Пример 1:** Чтобы получить доступ к данным на удаленном сервере, можно ввести, например, такой запрос:

```
SELECT * FROM CHICAGO.DSN8610.EMP  
WHERE EMPNO = '0001000';
```

Режим доступа зависит от того, связаны ли модули требования базы данных в пакеты, и от значения поля DATABASE PROTOCOL на панели установки DSNTIP5 или же от значения опции связывания DBPROTOCOL. Опция связывания DBPROTOCOL переопределяет значение установки.

**Пример 2:** Для выполнения того же самого задания можно использовать следующие операторы:

```
EXEC SQL  
CONNECT TO CHICAGO;  
SELECT * FROM DSN8610.EMP  
WHERE EMPNO = '0001000';
```

Перед выполнением запроса к базе данных в положении CHICAGO надо связать пакет на сервере CHICAGO.

**Пример 3:** Можно вызвать хранимую процедуру – подпрограмму, которая может содержать несколько операторов SQL. Программа выполняет следующие операторы:

```
EXEC SQL  
CONNECT TO ATLANTA;  
  
EXEC SQL  
CALL имя_процедуры (список_параметров);
```

список\_параметров – это список переменных хоста, который передается хранимой процедуре и через который она возвращает результаты своего выполнения. К моменту выполнения хранимая процедура должна уже существовать в положении ATLANTA.

**Два метода доступа:** Приведенные примеры иллюстрируют два различных метода доступа к распределенным данным.

- В примере 1 приведен оператор, который можно выполнить с помощью **доступа по собственному протоколу DB2 или доступа DRDA**.

Если модуль требования базы данных (DBRM), содержащий оператор, связывается в план в локальной DB2 и при этом задана опция связывания DBPROTOCOL(PRIVATE), доступ к серверу осуществляется по собственному протоколу DB2.

Если связывается DBRM с оператором, использующим один из указанных методов, доступ к серверу осуществляется по протоколу DRDA.

*Метод 1:*

- Свяжите модуль требования базы данных в пакет в локальной DB2, используя опцию связывания DBPROTOCOL(DRDA).
- Свяжите модуль требования базы данных в пакет в удаленном положении (CHICAGO).
- Свяжите пакеты в план, используя опцию связывания DBPROTOCOL(DRDA).

*Метод 2:*

- Свяжите модуль требования базы данных в пакет в удаленном положении.
- Свяжите удаленный пакет и в план, используя опцию связывания DBPROTOCOL(DRDA).
- В примерах 2 и 3 операторы при выполнении используют только **доступ DRDA**. Если указанные методы используют для доступа DRDA, программа должна содержать явный оператор CONNECT для переключения соединения от одной системы к другой.

**Выбор метода доступа при планировании:** Доступ по собственному протоколу DB2 и доступ DRDA отличаются некоторыми особенностями. Чтобы выбрать один из них, следует знать:

- **Тип сервера, к которому обращается запрос.**

Доступ по собственному протоколу DB2 может использоваться только в тех версиях DB2 for OS/390, где он поддерживается.

Доступ DRDA можно использовать во всех СУБД, для которых реализована архитектура распределенных реляционных баз данных (DRDA®). К ним относятся поддерживающие версии DB2 for OS/390, другие члены семейства DB2 продуктов IBM, а также многие продукты иных компаний.

- **Какие операции должен выполнять сервер.**

Доступ по собственному протоколу DB2 поддерживает только операторы обработки данных: INSERT, UPDATE, DELETE, SELECT, OPEN, FETCH и CLOSE. Нельзя вызывать пользовательские функции и хранимые процедуры или использовать большие объекты или пользовательские типы в программах, использующих доступ по собственному протоколу DB2.

Доступ DRDA допускает использование всех операторов, которые сервер может выполнить.

- **Какая производительность вам требуется.**

Доступ DRDA имеет несколько значительных преимуществ по сравнению с доступом по собственному протоколу DB2:

- Доступ DRDA использует более компактный формат для передачи данных по сети, который повышает производительность в медленных сетях.
- Запросы, посылаемые с помощью доступа по собственному протоколу DB2, связываются на сервере в момент первого выполнения единицы работы. Повторные связывания могут снизить производительность часто выполняемого запроса.

Модуль требования базы данных для операторов, которые выполняются с помощью доступа по протоколу DRDA, связывается в пакет на сервере только один раз. Эти операторы могут содержать ключевые слова PREPARE и EXECUTE, что позволит программе принимать динамические операторы для их выполнения на сервере. Однако связывание пакета — это дополнительный шаг в процессе подготовки программы.

- Доступ DRDA позволяет использовать хранимые процедуры.

Во время выполнения хранимой процедуры трафика сообщений в сети не требуется. Тем самым уменьшает влияние главного препятствия к повышению производительности при работе с распределенными данными.

**Рекомендация:** По мере возможности используйте доступ DRDA.

**Другие особенности планирования:** Авторизация связи с удаленным сервером и использование его ресурсов должна быть предоставлена на сервере соответствующему ID авторизации. Дополнительную информацию для сервера DB2 for OS/390смотрите в разделе Раздел 3 (Том 1) *DB2 Administration Guide*. Информацию о других серверахсмотрите в документации по соответствующему продукту.

Если изменяются две или более СУБД, надо предусмотреть координацию изменений, чтобы для всех единиц работы на различных СУБД либо было выполнено принятие, либо откат. Обязательно прочтите раздел “Координация изменений нескольких источников данных” на стр. 416.

Для управления распределенными операторами SQL на сервере можно использовать утилиту ограничения ресурсов. Ограничение в случае доступа по собственному протоколу DB2 производится с помощью плана, а для доступа по протоколу DRDA – с помощью пакета. Дополнительную информацию о необходимых изменениях в таблицах ограничения ресурсов при переходе от доступа по собственному протоколу DB2 к доступу по протоколу DRDA смотрите в разделе “Особенности перехода от доступа по собственному протоколу DB2 к доступу по протоколу DRDA” на стр. 430.

---

## Два метода программирования распределенных данных

В данном разделе для иллюстрации двух методов кодирования программ, работающих с распределенными данными, используется следующая гипотетическая программа:

Предположим, в компании Spiffy Computer есть главная таблица проекта, которая содержит информацию обо всех текущих проектах компании. Spiffy Computer имеет несколько отделений в различных точках мира, каждое положение DB2 использует копию таблицы проекта с именем DSN8610.PROJ. Главное отделение время от времени добавляет данные во все копии таблицы. Программа, которая делает эти добавления, использует таблицу имен положений. Для каждой добавляемой строки программа для каждого положения выполняет оператор INSERT для таблицы DSN8610.PROJ.

## Использование трехчастных имен таблиц

Трехчастные имена таблиц можно употреблять для доступа к данным в удаленном положении при использовании доступа DRDA или доступа по собственному протоколу DB2. При использовании трехчастных имен таблиц способ кодирования программы будет одним и тем же, независимо от выбранного метода доступа. Метод доступа определяется при связывании операторов SQL в пакет или план. Если используется доступ DRDA, все модули требования базы данных для каждого оператора SQL, выполняющегося на сервере, надо связать в пакеты, которые будут находиться на этом сервере.

Поскольку на платформах, отличных от DB2 for OS/390, синтаксис трехчастных имен таблиц может не поддерживаться, при кодировании программ не следует использовать трехчастные имена таблиц, если планируется перенос этих программ на другие платформы.

В трехчастных именах таблиц первая часть указывает положение. Локальная DB2 по необходимости устанавливает и прерывает неявное соединение с удаленным сервером.

Программа компании Spiffy Computer использует имя положения для создания трехчастного имени таблицы в операторе INSERT. Затем она подготавливает оператор и динамически его выполняет. (Этот метод описан в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.) Вставляемые значения передаются в удаленное положение и заменяют маркеры параметров в операторе INSERT.

Следующая схема показывает, как программа использует трехчастные имена:

```
Прочесть входные величины  
Выполнить для всех положений  
    Прочесть имя положения  
    Задать оператор для подготовки  
    Подготовить оператор  
    Выполнить оператор  
Конец цикла  
Принять
```

Получив имени положения, например, 'SAN\_JOSE', программа создает следующую символьную строку:

```
INSERT INTO SAN_JOSE.DSN8610.PROJ VALUES (?,?,?,?,?,?,?,?,?,?)
```

Программа присваивает переменной INSERTX эту символьную строку и затем выполняет следующие операторы:

```
EXEC SQL  
PREPARE STMT1 FROM :INSERTX;  
  
EXEC SQL  
EXECUTE STMT1 USING :PROJNO, :PROJNAME, :DEPTNO, :RESPEMP,  
                  :PRSTAFF, :PRSTDATE, :PRENDATE, :MAJPROJ;
```

Переменные хоста для таблицы проекта компании Spiffy Computer соответствуют объявлению таблицы проекта примера в разделе “Таблица проектов (DSN8610.PROJ)” на стр. 871.

Чтобы данные во всех положениях были непротиворечивыми, программа принимает работу, только если цикл выполнен для всех положений. Либо для оператора INSERT в каждом положении будет выполнено принятие, либо, если сбой помешает вставке в некотором положении, во всех остальных положениях будет выполнен откат оператора INSERT. (Если сбой произойдет при принятии, вся единица работы может оказаться неоднозначной.)

**Совет по программированию:** Может оказаться удобным использовать при создании символьных строк для подготовленных операторов алиас, а не полные трехчастные имена типа SAN\_JOSE.DSN8610.PROJ. Дополнительную информацию об алиасахсмотрите в разделе об операторе CREATE ALIAS в справочнике *DB2 SQL Reference*.

## Использование явных операторов CONNECT

С помощью этого метода программа явно соединяется с каждым новым сервером. Надо связать все модули требования базы данных для каждого оператора SQL, выполняющегося на сервере, в пакеты, которые будут находиться на этом сервере.

В нашем примере программа компании Spiffy Computer выполняет оператор CONNECT по очереди для каждого сервера, а сервер выполняет оператор INSERT. В этом случае все обновляемые таблицы называются одинаково, хотя они определены на различных серверах. Программа выполняет операторы в цикле, по одной итерации для каждого сервера.

Программа соединяется с каждым новым сервером при помощи переменной хоста в операторе CONNECT. Чтобы сообщить положение нового сервера, оператор CONNECT изменяет специальный регистр CURRENT SERVER. Величины, добавляемые в таблицу, передаются в положение как входные переменные хоста.

Следующая схема показывает, как программа использует явные операторы CONNECT:

```
Прочесть входные величины
Выполнить для всех положений
    Прочесть имя положения
    Соединиться с положением
    Выполнить оператор добавления
Конец цикла
Принять
Все освободить
```

Программа заносит новое имя положения в переменную LOCATION\_NAME и выполняет следующие операторы:

```
EXEC SQL
CONNECT TO :LOCATION_NAME;

EXEC SQL
INSERT INTO DSN8610.PROJ VALUES (:PROJNO, :PROJNAME, :DEPTNO, :RESPEMP,
:PRSTAFF, :PRSTDATE, :PRENDATE, :MAJPROJ);
```

Чтобы данные во всех положениях были непротиворечивыми, программа принимает работу, только если цикл выполнен для всех положений. Либо для оператора INSERT в каждом положении будет выполнено принятие, либо, если сбой помешает вставке в некотором положении, во всех остальных положениях будет выполнен откат оператора INSERT. (Если сбой произойдет при принятии, вся единица работы может оказаться неоднозначной.)

Переменные хоста для таблицы проекта компании Spiffy Computer соответствуют объявлению таблицы проекта примера в разделе “Таблица проектов (DSN8610.PROJ)” на стр. 871. LOCATION\_NAME – символьная переменная длиной 16 байт.

## **Освобождение соединений**

При явном соединении с удаленными положениями разрывать эти соединения тоже надо явно. При определении продолжительности соединения у вас больше возможностей, поэтому оператор RELEASE значительно отличается от оператора CONNECT.

### ***Отличия между операторами CONNECT и RELEASE:***

- Оператор CONNECT устанавливает немедленное соединение ровно с одной удаленной системой. Оператор CONNECT (Типа 2) не освобождает ни одного текущего соединения.
- Оператор RELEASE
  - Не выполняет немедленный разрыв соединения. Оператор *RELEASE* помечает соединения для их освобождения в момент следующего принятия. Помеченное таким образом соединение находится в состоянии отложенного освобождения и может использоваться до момента следующего принятия.
  - Может задавать для освобождения в момент следующего принятия как отдельное соединение, так и набор соединений. Приведенные ниже примеры показывают некоторые возможности данного оператора.

**Примеры:** Используя оператор RELEASE, можно перевести следующие соединения в состоянии отложенного освобождения.

- Определенное соединение, которое следующая единица работы не использует:  
`EXEC SQL RELEASE SPIFFY1;`
- Текущее соединение SQL, независимо от его имени положения:  
`EXEC SQL RELEASE CURRENT;`
- Все соединения, кроме локального:  
`EXEC SQL RELEASE ALL;`
- Все соединения, использующие собственный протокол DB2. Если первая фаза прикладной программы использует доступ по собственному протоколу DB2, а вторая – доступ DRDA, то открытые в первой фазе соединения, использующие собственный протокол DB2, могут во второй фазе вызывать сбой оператора CONNECT. Для предотвращения этой ошибки выполните следующий оператор перед операцией принятия, разделяющей эти фазы:  
`EXEC SQL RELEASE ALL PRIVATE;`

Предложение PRIVATE относится к соединениям, использующим собственный протокол DB2, которые существуют только между экземплярами программы DB2 for OS/390.

---

## Особенности программирования методов доступа

**Хранимые процедуры:** Если используется доступ DRDA, программа может вызывать хранимые процедуры в других системах, которые поддерживают их. Хранимые процедуры ведут себя как подпрограммы, в которых могут стоять операторы SQL, а также другие операторы. О них написано в разделе “Глава 7–2. Использование хранимых процедур в системах клиент–сервер” на стр. 579.

**Ограничения SQL для разнородных серверов:** Как правило, программа, использующая доступ DRDA, может выполнять операторы и условия SQL, поддерживаемые удаленным сервером, даже если они не поддерживаются локальным сервером. В справочнике *DB2 SQL Reference* перечисляются возможности, поддерживаемые DB2 for OS/390; аналогичная документация обычно имеется и для других продуктов. В следующих примерах показана ситуация с разнородными серверами:

- Всегда поддерживаются операторы SELECT, INSERT, UPDATE, DELETE, DECLARE CURSOR и FETCH, но детали могут различаться.

*Пример:* SQL/DS™ не поддерживает условие WITH HOLD в операторе DECLARE CURSOR.

- Операторы определения данных различаются сильнее.

*Пример:* SQL/DS не поддерживает оператор CREATE DATABASE. Вместо него используется оператор ACQUIRE DBSPACE.

- Операторы могут иметь различные ограничения.

*Пример:* В запросе DB2 for OS/390 можно использовать 750 столбцов; в других системах максимальным значением может быть 255. Но запрос, использующий 255 или менее столбцов, может выполняться во всех этих системах.

- Некоторые операторы не посыпаются на сервер, а целиком обрабатываются реквестером. Эти операторы нельзя использовать в удаленном пакете, даже если сервер их поддерживает. Список этих операторов приведен в разделе Приложение G, “Характеристики операторов SQL в DB2 for OS/390” на стр. 967.
- Вообще, если оператор, который должен быть выполнен на удаленном сервере, содержит переменные хоста, то реквестер DB2, если не поддерживает синтаксис оператора, предполагает, что они являются входными переменными хоста. Если предположение неверно, оператор отвергается сервером.

**Трехчастные имена для нескольких серверов:** Если удаленный сервер использует доступ DRDA для выполнения оператора, содержащего трехчастное имя или алиас, эквивалентный этому имени, а также если имя положения не совпадает с именем сервера, то метод, с помощью которого удаленный сервер получает доступ к данным, зависит от значения DBPROTOCOL. Если пакет на первом удаленном сервере связан с помощью опции DBPROTOCOL(PRIVATE), DB2 использует доступ по собственному протоколу DB2 для доступа ко второму удаленному серверу. Если пакет на первом удаленном сервере связан с помощью опции DBPROTOCOL(DRDA), DB2 использует доступ DRDA для доступа ко второму удаленному серверу.

Чтобы для доступа ко второму удаленному серверу был использован доступ DRDA, рекомендуется выполнить следующие действия:

- Заново свяжите пакет на первом удаленном сервере с помощью опции DBPROTOCOL(DRDA).
- На втором сервере свяжите пакет, содержащий трехчастное имя.

## Подготовка программ для доступа DRDA

Как правило, связывание пакета для выполнения в удаленном положении аналогично связыванию пакета для выполнения на локальной DB2.

Связывание плана для выполнения пакета аналогично связыванию любого другого плана. Общие инструкции приведены в разделе “Глава 6–1.

Подготовка прикладной программы к запуску” на стр. 435. В данном раздел описаны некоторые различия.

## Опции прекомпилятора

Если пакет при выполнении использует доступ по собственному протоколу DB2, для его подготовки допустимы следующие опции прекомпилятора:

### CONNECT

Следует использовать **CONNECT(2)**, явно или по умолчанию.

**CONNECT(1)** предоставляет операторам CONNECT только ограниченные возможности (возможности “удаленной единицы работы”). В особенности избегайте использовать **CONNECT(1)**, если программа изменяет несколько СУБД в одной единице работы.

### SQL

Для пакета, который выполняется на сервере, который *не является* DB2 for OS/390, следует явно задать опцию **SQL(ALL)**. В результате прекомпилятор будет считать, что все операторы подчиняются правилам протокола DRDA.

Если используется только сервер DB2 for OS/390, задайте **SQL(DB2)**, явно или по умолчанию. В результате прекомпилятор отвергнет все операторы, которые не подчиняются правилам DB2 for OS/390.

## Опции BIND PACKAGE

Если пакет при выполнении использует доступ DRDA, для его связывания допустимы следующие опции BIND PACKAGE:

### имя–положения

Укажите имя положения сервера, на котором выполняется пакет.

Необходимые для выполнения пакета привилегии должны быть предоставлены владельцу пакета на сервере. Если вы – не владелец пакета, вам необходимо дополнительно иметь права доступа SYSCTRL или привилегии BINDAGENT, предоставленные локально.

### SQLERROR

Используйте опцию **SQLERROR(CONTINUE)**, если при прекомпиляции была использована опция SQL(ALL). В этом случае пакет будет создан, даже если во время связывания будут найдены ошибки SQL, такие, например, как операторы, допустимые на удаленном сервере, но которые

прекомпилятор не смог распознать. Другой вариант – использовать опцию SQLERROR(NOPACKAGE), явно или по умолчанию.

#### CURRENTDATA

Используйте опцию **CURRENTDATA(NO)** для принудительной блочной выборки для неоднозначных указателей. Дополнительную информацию смотрите в разделе “Использование блочной выборки” на стр. 424.

#### OPTIONS

При создании удаленной копии пакета с помощью команды BIND PACKAGE с опцией COPY используйте эту опцию для управления опциями связывания, которые DB2 использует по умолчанию. Задайте:

**COMPOSITE**, чтобы DB2 использовала все опции, которые были заданы в команде BIND PACKAGE. Для всех остальных опций DB2 будет использовать опции из скопированного пакета. Эта опция принимается по умолчанию.

**COMMAND**, чтобы DB2 использовала опции, заданные в команде BIND PACKAGE. Для всех других опций DB2 использует значения по умолчанию того сервера, на котором пакет связан. Это дает уверенность в том, что сервер поддерживает опции, использованными при связывании пакета.

#### DBPROTOCOL

Чтобы DB2 использовала доступ по собственному протоколу DB2 для доступа к удаленным данным, заданным трехчастными именами, укажите **DBPROTOCOL(PRIVATE)**.

Чтобы DB2 использовала доступ DRDA для доступа к удаленным данным, заданным трехчастными именами, укажите **DBPROTOCOL(DRDA)**. Пакет должен быть связан во всех положениях, чьи имена заданы трехчастными именами.

Указанные значения замещают значение DATABASE PROTOCOL на панели установки DSNTIP5. Поэтому если установки DATABASE PROTOCOL на узле реквестера задают тот же тип удаленного доступа, который надо использовать для трехчастных имен, задавать опцию связывания DBPROTOCOL не требуется.

### Опции BIND PLAN

Следующие опции BIND PLAN особенно важны для связывания плана, использующего доступ DRDA:

#### DISCONNECT

Для большей гибкости используйте DISCONNECT(EXPLICIT), явно или по умолчанию. В этом случае в программе для явного завершения соединений надо использовать операторы RELEASE.

Однако полезны и другие значения этой опции.

DISCONNECT(AUTOMATIC) завершает все удаленные соединения во время операции принятия, наличия операторов RELEASE в программе в этом случае не требуется.

DISCONNECT(CONDITIONAL) завершает удаленные соединения во время операции принятия, за исключением случая, когда с соединением связан открытый указатель, определенный с условием WITH HOLD.

## **SQLRULES**

Используйте **SQLRULES(DB2)**, явно или по умолчанию.

**SQLRULES(STD)** применяет стандартные правила SQL к операторам CONNECT, поэтому оператор CONNECT TO x вызовет ошибку, если уже имеется соединение с x. Используйте STD, только если требуется получить код ошибки.

Если ваша программа берет с удаленного положения данные большого объекта, и вы связываете план для этой программы с **SQLRULES(DB2)**, формат, в котором вы получите данные большого объекта с указателем, ограничен. Открыв указатель для получения данных большого объекта, надо получить все данные либо при помощи переменной большого объекта, либо при помощи переменной локатора большого объекта. Если значение **SQLRULES – STD**, такого ограничения нет.

Если вы намереваетесь при получении данных указателя переходить от использования переменных большого объекта к использованию локаторов большого объекта или наоборот, перед соединением с удаленным положением выполните оператор SET **SQLRULES=STD**.

## **CURRENTDATA**

Используйте опцию **CURRENTDATA(NO)** для принудительной блочной выборки для неоднозначных указателей. Дополнительную информацию смотрите в разделе “Использование блочной выборки” на стр. 424.

## **DBPROTOCOL**

Чтобы DB2 использовала доступ по собственному протоколу DB2 для доступа к удаленным данным, заданным трехчастными именами, укажите **DBPROTOCOL(PRIVATE)**.

Чтобы DB2 использовала доступ DRDA для доступа к удаленным данным, заданным трехчастными именами, укажите **DBPROTOCOL(DRDA)**. Пакет должен быть связан во всех положениях, чьи имена заданы трехчастными именами.

Значение опции DBPROTOCOL пакета переопределяет опцию плана. Например, если для удаленного пакета задано DBPROTOCOL(DRDA), а для плана задано DBPROTOCOL(PRIVATE), для доступа к данным в положении, заданном трехчастным именем, DB2 будет использовать доступ DRDA. Если в опции DBPROTOCOL не указано ни одно из значений, DB2 использует значение DATABASE PROTOCOL на панели установки DSNTIP5.

## **Проверка опций BIND PACKAGE**

В требованиях могут быть только те опции BIND PACKAGE, которые поддерживаются сервером. Но эти опции надо задать в реквестере с использованием синтаксиса реквестера для BIND PACKAGE. Чтобы определить, какие опции поддерживает конкретный сервер СУБД, смотрите документацию, поставляемую с этим сервером. Если сервер распознает опцию по другому имени, помочь ее идентифицировать может таблица общих описаний, приведенная в Приложение Н, “Опции подготовки программ для удаленных пакетов” на стр. 973.

- Разъяснения об использовании опций связывания DB2 и процессе связывания приведены в данной книге, в частности, в разделе “Глава 6–1. Подготовка прикладной программы к запуску” на стр. 435.

- Синтаксис подкоманд DB2 BIND и REBIND описан в разделе Глава 2 книги *DB2 Command Reference*.
- Общий список опций связывания DB2, включая те опции, которых нельзя задавать для DB2, но можно использовать для других серверов (не DB2), приводится в разделе Приложение Н, “Опции подготовки программ для удаленных пакетов” на стр. 973.

---

## Координация изменений нескольких источников данных

**Определение:** Несколько изменений считаются *координированными*, если все они должны быть приняты или для всех произведен откат в одной и той же единице работы.

Изменение нескольких СУБД можно координировать автоматически если во всех системах реализован метод *двухфазного принятия*.

**Пример:** Ситуация, обычная для банка: некоторая сумма снимается с одного счета и начисляется на другой. Для этих действий надо в конце единицы работы либо одновременно выполнить принятие, либо откат.

В DB2 и IMS, а также DB2 и CICS совместно реализован процесс двухфазного принятия. В одной единице работы можно изменять и базу данных IMS, и таблицу DB2. Если в самой системе или системе связи между принятием работы в IMS и в DB2 происходит сбой, после восстановления активности две программы приводят две системы в непротиворечивое состояние.

Детали процесса двухфазного принятия для последующего описания несущественны. Их можно прочесть в разделе Раздел 4 (Том 1) книги *DB2 Administration Guide*.

## Как координировать изменения

Старайтесь работать только в тех системах, где реализовано двухфазное принятие.

В Версии 3 DB2 for OS/390 и более поздних двухфазное принятие реализовано. Для СУБД других типов проверьте спецификацию продукта.

**Пример:** В примерах, приведенных в разделах “Использование трехчастных имен таблиц” на стр. 408 и “Использование явных операторов CONNECT” на стр. 410, предполагается, что во всех используемых системах реализовано двухфазное принятие. В обеих примерах предлагается в цикле изменить несколько систем и окончить единицу работы принятием, только если цикл будет завершен. В обеих случаях изменения координируются во всех системах.

**Ограничения изменений на серверах, не поддерживающих двухфазное принятие:** Нельзя производить сконфигурированные изменения, если на СУБД не реализовано двухфазное принятие. Далее такая СУБД будет называться *ограниченной системой*. DB2 не позволяет в одной единице работы одновременно изменять ограниченную и любую другую систему. В данном контексте под *изменением* подразумевается выполнение операторов INSERT, DELETE, UPDATE, CREATE, ALTER, DROP, GRANT, REVOKE и RENAME.

Чтобы скоординировать изменения для ограниченной системы, сначала надо изменить первую систему и принять работу, а затем изменить следующую систему и принять эту работу. Если сбой происходит после принятия первого изменения, но до принятия второго, нет автоматического средства приведения двух систем в непротиворечивое состояние. Эти действия должны быть предусмотрены в вашей программе.

#### CICS и IMS

Нельзя производить изменения на серверах, которые не поддерживают двухфазное принятие.

#### среда TSO и пакетный файл

Изменения возможны, если и только если:

- Нет других соединений или
- Все существующие соединения выполняются с серверами, операции с которыми ограничены только чтением.

Если эти условия не выполнены, то операции ограничены только чтением.

Если первое соединение в логической единице работы производится с сервером, поддерживающим двухфазное принятие, и больше соединений нет, или есть только соединения, ограниченные чтением, то на этом сервере и всех серверах, поддерживающих двухфазное принятие, можно производить изменения. Однако, если первое соединение осуществлено с сервером, не поддерживающим двухфазное принятие, разрешено изменять только этот сервер.

**Рекомендация:** Следует использовать DB2 для предотвращения изменений в двух системах в одной единице работы, если одна из них – ограниченная система.

## Что можно делать без двухфазного принятия

При обращении к системам смешанного типа, некоторые из которых – ограниченные, можно:

- Читать в любой системе в любое время.
- Изменять любую систему любое число раз в одной единице работы.
- Изменять несколько систем, включая CICS или IMS, в одной единице работы, если ни одна из них не является ограниченной системой. Если первая изменяемая в единице работы система не является ограниченной, любая попытка изменить ограниченную систему в данной единице работы завершится ошибкой.
- Изменять одну ограниченную систему в единице работы, если только не пытаться изменить любую другую систему в той же единице работы. Если первая изменяемая в единице работы система – ограниченная, попытка изменить любую другую систему в этой же единице работы завершится ошибкой.

**Ограничения для CONNECT (типа 1):** Используя правила типа 1 для оператора CONNECT, можно ограничить программу правилами для ограниченных систем. Эти правила совместимы с пакетами, связанными в DB2 для MVS Версия 2 Выпуск 3 и которые не были связаны заново в более поздних версиях. Чтобы эти правила применялись для пакета, надо использовать опцию препроцессора CONNECT(1). Следует быть внимательным, чтобы не использовать пакеты, скомпилированные с опцией CONNECT(1), и пакеты, скомпилированные с опцией CONNECT(2), в одном и том же списке пакетов. Первый оператор CONNECT, выполняемый программой, определяет, какие правила, типа 1 или типа 2, действуют в течение всего выполнения. Попытка выполнить следующий оператор CONNECT, скомпилированный с другим типом правил, завершается ошибкой.

Дополнительную информацию об операторе CONNECT (типа 1) и об управлении соединениями с другими системамисмотрите в разделе Глава 2 *DB2 SQL Reference*.

---

## Различные вопросы, связанные с распределенными данными

Выбор метода доступа и управление соединениями с другими системами — ключевые моменты в разработке программ, использующих распределенные данные. Этот раздел содержит советы о других темах:

- “Повышение производительности систем с удаленным доступом”
- “Повышение производительности для больших объектов в распределенной среде” на стр. 419
- “Задание опции OPTIMIZE FOR n ROWS” на стр. 425
- “Поддержание согласованности данных” на стр. 429
- “Копирование таблицы из удаленного положения” на стр. 429
- “Передача смешанных данных” на стр. 429
- “Особенности перехода от доступа по собственному протоколу DB2 к доступу по протоколу DRDA” на стр. 430

## Повышение производительности систем с удаленным доступом

Запрос к удаленной подсистеме почти всегда выполняется дольше этого же запроса, обращающегося к таблицам равного размера в локальной подсистеме. Основные причины этого:

- Дополнительные вычислительные затраты, включая запуск, согласование ограничений сеанса, и, для доступа по собственному протоколу DB2, связывание, необходимое для удаленного положения
- Время на пересылку сообщений по сети.

## Эффективность кода запроса

Чтобы повысить эффективность доступа к удаленным подсистемам по сравнению с локальной подсистемой при работе со сходными таблицами, надо создавать запросы, посылающие по сети мало сообщений. Для этого надо:

- Уменьшить число столбцов и строк в таблице результатов, возвращаемой программе. Делать списки SELECT как можно короче. Условия WHERE, GROUP BY и HAVING использовать творчески, чтобы уменьшить объем ненужных данных от удаленного сервера.

- Использовать условия FOR FETCH ONLY или FOR READ ONLY.  
Например, разумно получать тысячи строк в виде последовательного потока. Посылка отдельного сообщения для каждой из них может оказаться значительно более медленной.
- По возможности не следует связывать программные планы и пакеты с опцией ISOLATION(RR), даже если она используется по умолчанию. Если программа не должна снова обращаться к уже прочтенным строкам, другой уровень изоляции может ослабить конфликт блокировок и затраты на пересылки сообщений во время выполнения оператора COMMIT.
- Минимизируйте использование маркеров параметров.

Если программа использует доступ DRDA, DB2 может ускорять выполнение динамических запросов, где нет маркеров параметров.

Если в таком запросе встречается оператор PREPARE, реквестер DB2 предполагает, что программа собирается открыть указатель. Поэтому реквестер посыпает на сервер, одно сообщение, содержащее комбинированное требование для операторов PREPARE, DESCRIBE и OPEN. Сервер DB2, получив это сообщение, возвращает одно ответное сообщение, содержащее вывод операций PREPARE, DESCRIBE и OPEN. Таким образом, число сообщений, посланных и принятых для этих операций, снижается с 2 до 1.

DB2 комбинирует сообщения для этих запросов, независимо от того, задана ли опция связывания DEFER(PREPARE).

## **Повышение производительности для больших объектов в распределенной среде**

Если используется доступ DRDA, можно обращаться к столбцам больших объектов в удаленной таблице. Поскольку значения больших объектов обычно довольно велики, надо использовать такие приемы для получения данных, которые минимизируют количество байтов, передаваемых между клиентом и сервером.

**Используйте локаторы больших объектов вместо переменных хоста:** Если надо сохранить только часть значения большого объекта на клиенте или если программа клиента работает с данными большого объекта, но не требует их копирования, хорошим решением будет использование локаторов больших объектов. Когда программа клиента получает от сервера столбцы большого объекта и записывает его адрес в локатор, DB2 передает клиенту только значение 4-байтного локатора, но не всю величину типа большой объект. Дополнительную информацию об использовании в программе локаторов больших объектов смотрите в разделе “Применение локаторов больших объектов для экономии памяти” на стр. 262.

**Используйте наборы результатов хранимых процедур:** При возвращении данных большого объекта клиентской программе от хранимой процедуры используйте наборы результатов вместо передачи значений большого объекта через параметры. Использование для передачи данных сокращает число материализаций большого объекта и передачу данных между адресными пространствами. Сведения о создании хранимых процедур, возвращающих наборы результатов, смотрите в разделе “Написание хранимой процедуры, возвращающей наборы результатов клиенту DRDA” на стр. 603. Сведения о написании клиентских программ, принимающих наборы результатов, смотрите

в разделе “Написание клиентской программы DB2 for OS/390, получающей наборы результатов” на стр. 637.

**Установите для специального регистра CURRENT RULES значение DB2:**

Если сервер DB2 for OS/390 получает для указателя требование OPEN, он использует значение специального регистра CURRENT RULES для определения типа переменных хоста, которые соответствующий оператор использует для получения величин типа большой объект. Если для CURRENT RULES установлено значение DB2, а первый оператор FETCH курсора использует локатор большого объекта для получения значений столбцов типа большой объект, DB2 позволяет использовать только локаторы большого объекта для всех последующих операторов FETCH для этого столбца, пока указатель не будет закрыт. Если первый оператор FETCH использует переменную хоста, DB2 позволяет использовать только переменные хоста для всех последующих операторов FETCH для этого столбца, пока указатель не будет закрыт. Однако если для CURRENT RULES установлено значение STD, DB2 позволяет использовать один и тот же открытый указатель для выборки столбца типа большой объект либо в локатор большого объекта, либо в переменную хоста.

Хотя значение STD для CURRENT RULES дает большую программную гибкость при получение данных типа большой объект, производительность много выше для значения DB2. При использовании опции STD сервер должен посыпать и получать сетевые сообщения для каждого оператора FETCH, чтобы указать, являются ли переданные данные локатором большого объекта или величиной типа большой объект. Для опции DB2 серверу известен размер данных типа большой объект после выполнения первого оператора FETCH, поэтому дополнительных сообщений о размере данных типа большой объект не требуется. Сервер одновременно может посыпать реквестеру несколько блоков данных, что снижает общее время передачи данных.

Например, конечный пользователь хочет пролистать большой набор записей о служащих, но посмотреть фотографии только некоторых из них. Для этого на сервере для специального регистра CURRENT RULES устанавливается значение DB2. В программе объявляется и открывается указатель для выбора записей о служащих. Программа затем делает выборку всех фотографий в 4–байтные локаторы больших объектов. DB2 знает, что 4 байта данных типа большой объект возвращаются для каждого оператора FETCH, поэтому DB2 может заполнить сетевые буферы локаторами многих фотографий. Если пользователь захочет увидеть фотографию определенного человека, программа может получить фотографию с сервера, присвоив значение, на которое ссылается локатор большого объекта, переменной хоста типа большой объект:

```
SQL TYPE IS BLOB my_blob[1M];
SQL TYPE IS BLOB AS LOCATOR my_loc;
:
FETCH C1 INTO :my_loc; /* Выбрать BLOB в локатор большого объекта */
:
SET :my_blob = :my_loc; /* Присвоить BLOB переменной хоста */
```

## Используйте опции связывания, повышающие производительность

Выбор опций связывания может сильно влиять на производительность распределенных программ:

- DEFER(PREPARE) или NODEFER(PREPARE)
- REOPT(VARS) или NOROPT(VARS)
- CURRENTDATA(YES) или CURRENTDATA(NO)
- KEEPDYNAMIC(YES) или KEEPDYNAMIC(NO)
- DBPROTOCOL(PRIVATE) или DBPROTOCOL(DRDA)

### DEFER(PREPARE)

Чтобы повысить производительность как для статических, так и динамических операторов, использующих доступ по собственному протоколу DB2, а также динамических операторов SQL, использующих доступ DRDA, задайте опцию DEFER(PREPARE) при связывании или пересвязывании планов или пакетов.

Помните, что статически связанные операторы SQL при доступе по собственному протоколу DB2 обрабатываются динамически. Когда динамические операторы SQL получают доступ к удаленным данным, операторы PREPARE и EXECUTE могут быть переданы по сети вместе и быть обработаны в удаленном положении, и ответы на оба оператора также могут быть посланы назад в локальную подсистему вместе, что сокращает трафик в сети. DB2 не подготавливает динамический оператор SQL до начала его выполнения. (Иключение – динамический оператор SELECT, в котором присутствует комбинация PREPARE и DESCRIBE, независимо от наличия опции DEFER(PREPARE).)

Все сообщения PREPARE для динамических операторов SQL, относящиеся к удаленному объекту, откладываются, пока:

- Оператор не начнет выполняться
- Программа не потребует описания результатов работы оператора.

Обычно, если PREPARE откладывается, DB2 возвращает для операторов PREPARE SQLCODE 0. Поэтому необходимо, чтобы программа обрабатывала все коды SQL, которые могут быть возвращены для оператора PREPARE после соответствующего оператора EXECUTE или DESCRIBE.

Если используется прогностическое ограничение ресурсов, код SQL возвращается реквестеру, если сервер превышает порог предупреждения прогностического ограничения, зависящий от уровня DRDA на реквестере. Дополнительную информациюсмотрите в разделе “Как задать в программе обработку прогностического ограничения” на стр. 553.

При доступе по собственному протоколу DB2, если статический оператор SQL обращается к удаленному объекту, прозрачный оператор PREPARE и операторы EXECUTE автоматически комбинируются и совместно передаются по сети. Оператор PREPARE будет отложен, только если была задана опция связывания DEFER(PREPARE).

Операторы PREPARE, содержащие условия INTO, не откладываются.

## PKLIST

Порядок, в котором заданы собрания пакетов в списке пакета, может влиять на производительность программы. Если локальный экземпляр программы DB2 пытается выполнить оператор SQL на удаленном сервере, локальная подсистема DB2 должна определить, в каком собрании пакетов находится оператор SQL. DB2 должна послать сообщение на сервер с требованием, чтобы сервер искал этот оператор SQL во всех ID собраний, пока искомый оператор не будет найден или пока не будут проверены все ID собраний в списке пакета. Можно добиться снижения сетевого трафика и, тем самым, повысить производительность, уменьшив количество собраний пакетов, которые каждый сервер должен искать. Примеры показывают пути уменьшения объема поиска собраний пакетов:

- Уменьшите число тех пакетов в собрании, которые надо искать. В следующем примере в каждом собрании задан только один пакет:  
`PKLIST(S1.COLLA.PGM1, S1.COLLB.PGM2)`
- Уменьшите число тех собраний пакетов в каждом положении, в которых идет поиск. В следующем примере в каждом положении задано только одно собрание пакета:  
`PKLIST(S1.COLLA.* , S2.COLLB.* )`
- Уменьшите число собраний, используемых в каждой программе. В следующем примере задан поиск только одного собрания:  
`PKLIST(*.COLLA.* )`

Можно также задать собрание пакета, связанное с оператором SQL в прикладной программе. Перед выполнением оператора SQL выполните оператор SET CURRENT PACKAGESET, чтобы сообщить DB2, какое собрание пакета для оператора надо искать.

Если опция DEFER(PREPARE) используется при доступе по протоколу DRDA, пакет, содержащий операторы, подготовку которых надо отложить, для DB2 должен быть *первым* в последовательности поиска. (Дополнительную информацию смотрите в разделе “Задание пакетов при запуске” на стр. 454.) Например, предположим, что список пакета для плана содержит две записи:  
`PKLIST(LOCB.COLLA.* , LOCB.COLLB.* )`

Если необходимый пакет – это собрание COLLB, убедитесь, что DB2 прежде всего ищет именно это собрание. Это можно сделать, выполнив оператор SQL  
`SET CURRENT PACKAGESET = 'COLLB';`

или поставив COLLB первым в параметре PKLIST оператора BIND PLAN:  
`PKLIST(LOCB.COLLB.* , LOCB.COLLA.* )`

Если используется опция NODEFER(PREPARE), собрания в списке пакета могут следовать в любом порядке, но если пакет не найден в первой заданной записи PKLIST, увеличиваются затраты в сети на поиски в списке.

## **REOPT(VARS)**

Если задана опция REOPT(VARS), DB2 определяет пути доступа времени связывания и времени выполнения для операторов, которые содержат одну или несколько следующих переменных:

- Переменные хоста
- Маркеры параметра
- Специальные регистры

Во время выполнения DB2 использует значения этих переменных для определения путей доступа.

Если задана опция связывания REOPT(VARS), DB2 устанавливает опцию связывания DEFER(PREPARE) автоматически.

При реоптимизации пути доступа во время выполнения производительность DB2 снижается, поэтому рекомендуется следующее:

- Использовать опцию связывания REOPT(VARS) только для пакетов или планов с операторами, выполняющимися неэффективно из-за плохого пути доступа.
- Использовать опцию NOREOPT(VARS) при связывании плана или пакета, в котором есть операторы с доступом по собственному протоколу DB2.

Если задана опция REOPT(VARS) при связывании плана, в котором для доступа к удаленным данным операторы используют доступ по собственному протоколу DB2, DB2 подготавливает эти операторы дважды. Дополнительную информацию об опции REOPT(VARS) смотрите в разделе “Как опция связывания REOPT(VARS) влияет на динамический SQL” на стр. 576.

## **CURRENTDATA(NO)**

Используйте эту опцию связывания для принудительной блочной выборки для неоднозначных запросов. Дополнительную информацию о блочной выборке смотрите в разделе “Использование блочной выборки” на стр. 424.

## **KEEPDYNAMIC(YES)**

Используйте эту опцию связывания для повышения производительности запросов, использующих указатели, определенные с условием WITH HOLD.

Если использована KEEPDYNAMIC(YES), DB2 автоматически закрывает указатель, когда не остается получаемых данных. Клиенту не требуется посылать DB2 сетевое сообщение закрыть указатель. Дополнительную информацию об опции KEEPDYNAMIC(YES) смотрите в разделе “Сохранение подготовленных операторов после точек принятия” на стр. 549.

## **DBPROTOCOL(DRDA)**

Если значение поля DATABASE PROTOCOL на панели установки – не DRDA, используйте эту опцию связывания, чтобы DB2 для выполнения операторов SQL с трехчастными именами использовала доступ DRDA. Операторы, использующие доступ DRDA, при выполнении имеют лучшую производительность, потому что:

- Связывание происходит при связывании пакета, а не во время выполнении программы.

- DB2 не удаляет статическую информацию во время выполнения оператора COMMIT (в отличие от доступа по собственному протоколу DB2). Это означает, что при использовании доступа по протоколу DRDA, если COMMIT происходит между двумя выполнениями оператора, DB2 не подготавливает оператор дважды.

## Использование блочной выборки

В DB2 есть два различных метода для уменьшения числа сообщений, посылаемых по сети при выборке данных с использованием указателя:

- Доступ DRDA может использовать *ограниченную блочную выборку*. Она оптимизирует передачу данных, гарантируя передачу минимального объема данных в ответ на каждое требование реквестера.
- Доступ по собственному протоколу DB2 может использовать *непрерывную блочную выборку*, которая посылает одно требование от реквестера на сервер. Сервер заполняет буфер получаемыми данными и посыпает его на реквестер. Обработка на реквестере не синхронизирована с работой сервера; сервер продолжает посыпать блоки данных на реквестер без дополнительного предупреждения.

**Как обеспечить блочную выборку:** Чтобы использовать один из типов блочной выборки, DB2 должна установить, что указатель не используется для изменения или удаления. Это надо задать в программе, указав в операторе DECLARE CURSOR запроса условие FOR FETCH ONLY или FOR READ ONLY. Если FOR FETCH ONLY или FOR READ ONLY не указаны, DB2 продолжает использовать блочную выборку для запроса, если:

- Таблица результатов указателя предназначена только для чтения. (Описание таблиц, предназначенных только для чтения, смотрите в разделе Глава 6 *DB2 SQL Reference*.)
- Таблица результатов указателя не предназначена только для чтения, но указатель неоднозначен, и опция CURRENTDATA оператора BIND имеет значение NO. Указатель неоднозначен, если:
  - Он определен без использования условий FOR FETCH ONLY, FOR READ ONLY или FOR UPDATE OF.
  - Он не определен на таблице результатов, предназначеннной только для чтения.
  - Он не является назначением условия WHERE CURRENT операторов SQL UPDATE или DELETE.
  - Он находится в плане или пакете, содержащем операторы SQL PREPARE или EXECUTE IMMEDIATE.

Табл. 43 содержит сводку условий использования DB2 блочной выборки.

Таблица 43. Влияние CURRENTDATA и уровня изоляции на блочную выборку

Изоляция	CURRENTDATA	Тип указателя	Блочная выборка
CS или RR	Да	Только для чтения	Да
		Обновляемый	Нет
		Неоднозначный	Нет
	Нет	Только для чтения	Да
		Обновляемый	Нет
		Неоднозначный	Да
UR	Да	Только для чтения	Да
	Нет	Только для чтения	Да

DB2 не использует непрерывную блочную выборку, если:

- На указатель где–то в программе есть ссылка из оператора DELETE WHERE CURRENT OF
- Оказывается, что оператор указателя может быть изменен запрашиваемой системой. (DB2 не проверяет, ссылается ли указатель на производную таблицу сервера, который не допускает изменений.)

## Задание опции OPTIMIZE FOR n ROWS

В операторах SELECT можно задавать условие OPTIMIZE FOR *n* ROWS, чтобы ограничить число строк, которые сервер возвращает при каждой сетевой передаче по протоколу DRDA. Условие OPTIMIZE FOR *n* ROWS можно также использовать для возвращения набора результатов запроса из хранимой процедуры.

Число строк, которое DB2 передает в каждой сетевой передаче, зависит от следующих факторов:

- Если *n* строк набора результатов SQL помещается в один блок запроса DRDA, сервер DB2 может послать *n* строк любому клиенту DRDA. В этом случае DB2 посыпает по *n* строк в каждой сетевой передаче, пока весь набор результатов запроса не будет исчерпан.
- Если *n* строк набора результатов SQL превышает один блок запроса DRDA, число строк в каждой сетевой передаче, зависит от уровня программы DRDA клиента и конфигурации:
  - Если клиент не поддерживает уровень 3 DRDA, сервер DB2 автоматически уменьшает значение *n*, чтобы оно соответствовало числу строк, которое помещается в блок запроса DRDA.
  - Если клиент поддерживает уровень 3 DRDA, клиент DRDA может принимать несколько блоков запроса DRDA в одной передаче данных. DRDA позволяет клиенту устанавливать верхний предел числа блоков запроса DRDA в каждой сетевой передаче.

Число строк, которое посыпает сервер DB2 – это наименьшее из *n* и числа, не превышающего меньшее из двух чисел:

- Величины EXTRA BLOCKS SRV на панели установки DSNTIP5 сервера DB2

Это максимальное число дополнительных блоков запроса DRDA, которое сервер DB2 возвращает клиенту в одной сетевой передаче.

- Предельного числа дополнительных блоков запроса клиента, которое берется из параметра DDM MAXBLKEXT, полученного от клиента

Когда DB2 работает в качестве клиента DRDA, для параметра DDM MAXBLKEXT установлено значение, заданное в опции установки EXTRA BLOCKS REQ панели установки DSNTIP5.

Условие OPTIMIZE FOR *n* ROWS полезно в двух случаях:

- Если *n* меньше числа строк, помещающихся в блоке запроса DRDA, OPTIMIZE FOR *n* ROWS может повысить производительность, предотвращая выборку сервером DB2 строк, которые, возможно, никогда не будут использованы клиентской программой DRDA.
- Если *n* больше числа строк, помещающихся в блок запроса DRDA, OPTIMIZE FOR *n* ROWS позволяют клиенту DRDA в каждой сетевой передаче требовать несколько блоков с данными запроса. Такое использование OPTIMIZE FOR *n* ROWS может значительно уменьшить время выполнения для запросов с объемными операциями загрузки.

Задание большого значения *n* в OPTIMIZE FOR *n* ROWS может увеличить число блоков запроса DRDA, возвращаемых сервером DB2 в каждой сетевой передаче. Эта функция может значительно повысить производительность программ, которые используют доступ DRDA для загрузки данных большого объема. Однако эта же функция при неправильном использовании может снизить производительность. Следующие примеры демонстрируют снижение производительности, которое может произойти, если использовать OPTIMIZE FOR *n* ROWS неразумно.

На рис. 104 на стр. 427 клиент DRDA открывает указатель и выбирает строки из указателя. В некоторый момент до возвращения всех строк в наборе результатов запроса, программа выполняет оператор SQL INSERT. DB2 использует нормальную блокировку DRDA, которая дает два преимущества по сравнению с блокировкой с помощью OPTIMIZE FOR *n* ROWS:

- Если программа выполняет оператор SQL, отличный от FETCH, (в примере приведен оператор INSERT), клиент DRDA может передать оператор SQL немедленно, потому что соединение DRDA не используется после выполнения оператора SQL OPEN.
- Если программа SQL закрывает указатель перед выборкой всех строк в наборе результатов запроса, сервер выбирает только столько строк, сколько помещается в один блок запроса (100 строк набора результатов). По существу размер блока запроса DRDA ограничивает число строк, которые могут быть выбраны сверх необходимости.

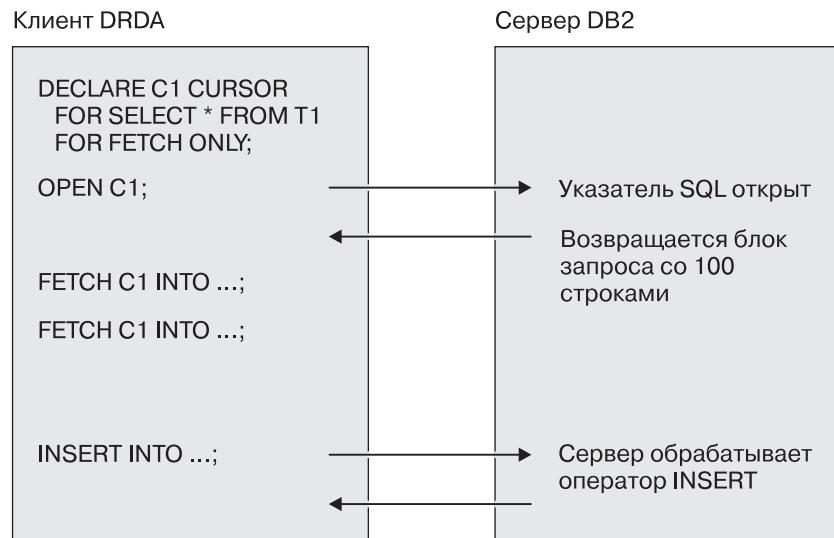


Рисунок 104. Потоки сообщений без условия *OPTIMIZE FOR 1000 ROWS*

На рис. 105 на стр. 428 клиент DRDA открывает указатель и выбирает строки из него с помощью *OPTIMIZE FOR n ROWS*. Как клиент DRDA, так и сервер DB2 сконфигурированы для поддержки нескольких блоков запроса DRDA. В некоторый момент до окончания создания набора результатов запроса программа выполняет оператор SQL *INSERT*. Поскольку используется *OPTIMIZE FOR n ROWS*, соединение DRDA недоступно для оператора SQL *INSERT*, потому что соединение продолжает использоваться для приема блоков запроса DRDA для 1000 строк данных. Это вызывает две проблемы снижения производительности:

- Время выполнения программы может возрасти, если клиент DRDA ожидает передачи большого набора результатов запроса; при этом соединение DRDA не может быть использовано другими операторами SQL. На рис. 105 на стр. 428 показана задержка в операторе SQL *INSERT* из-за большого набора результатов запроса.
- Если программа закрывает указатель перед выборкой всех строк набора результатов SQL, сервер может выбрать много ненужных строк.

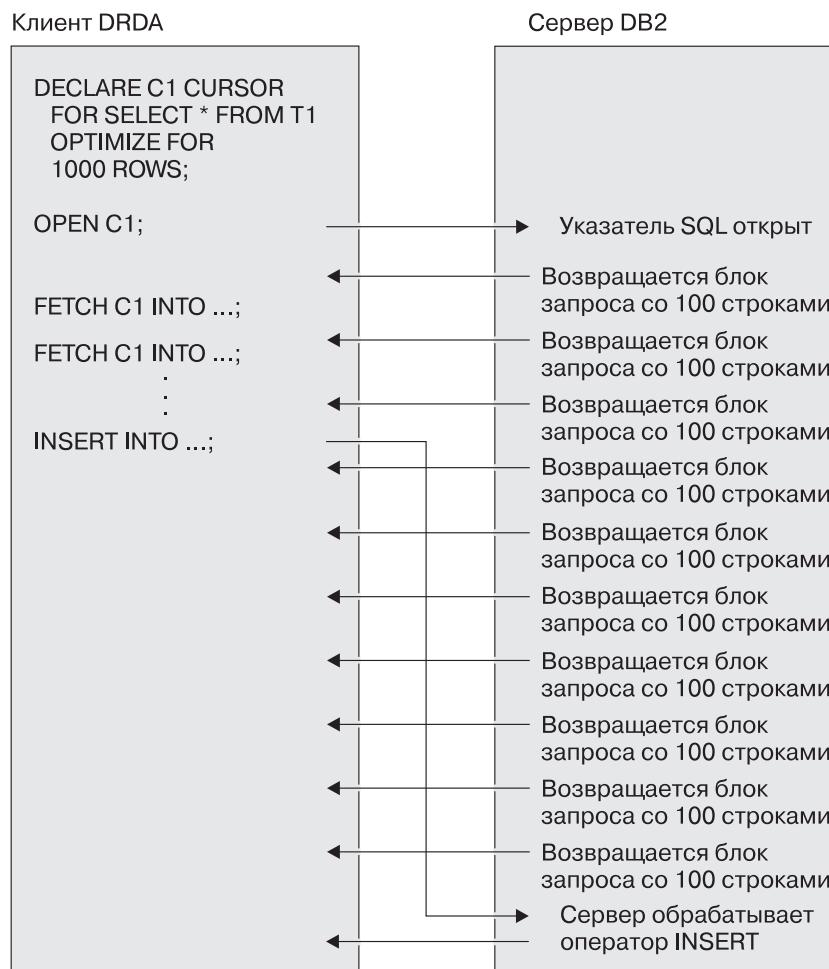


Рисунок 105. Потоки сообщений с условием *OPTIMIZE FOR 1000 ROWS*

**Рекомендация:** *OPTIMIZE FOR n ROWS* следует использовать для увеличения числа блоков запроса DRDA только в программах, которые имеют все следующие атрибуты:

- Программа выбирает большое число строк в запросе только для чтения.
- Программа редко закрывает указатель SQL перед выборкой всего набора результатов запроса.
- Программа не посыпает иных, кроме *FETCH*, операторов серверу DB2, пока открыт указатель SQL.
- Программа не выполняет операторов *FETCH* для нескольких одновременно открытых указателей, определенных с условием *OPTIMIZE FOR n ROWS*.

Дополнительную информацию об условии *OPTIMIZE FOR n ROWS* смотрите в разделе “Минимизация затрат при получении небольшого числа строк: *OPTIMIZE FOR n ROWS*” на стр. 694.

## **Поддержание согласованности данных**

Указатели, используемые в операциях блочной выборки, связанные с уровнем изоляции "устойчивость на уровне указателя", особенно уязвимы к чтению измененных данных. При блочной выборке выполняются опережающий доступ к базе данных для предварительной выборки строк. До завершения процесса, перед тем как программа получит данные, указатель может быть закрыт, а блокировки освобождены. Поэтому программа может выбрать строку или величины, которых больше не существует, или пропустить только что добавленную строку. Во многих случаях это допустимо; если же это не допустимо, говорят, что требуется *согласованность данных*.

**Как предотвратить блочную выборку:** Если программа требует согласованность данных для указателя, это означает, что надо предотвратить блочную выборку для данных, на которые он указывает. Чтобы предотвратить блочную выборку для распределенного указателя, объягите указатель условием FOR UPDATE OF, указав какой-либо столбец списка SELECT.

## **Копирование таблицы из удаленного положения**

Для копирования таблицы из одного положения в другое можно либо написать собственную программу, или использовать продукт DataPropagator Relational.

## **Передача смешанных данных**

Если смешанные данные передаются между локальной и удаленной системами, поместите данные в символьные строки не фиксированной, а переменной длины.

Если данные типа ASCII MIXED конвертируются в формат EBCDIC MIXED, конвертированная строка длиннее исходной. Если это преобразование производится в переменную хоста фиксированной длины, происходит ошибка. Правильным решением будет использование символьной переменной переменную длину, причем максимальная длина должна быть достаточна для преобразования.

## **Идентификация сервера во время выполнения**

Специальный регистр CURRENT SERVER содержит имя положения системы, с которой установлено соединение. Это имя можно присвоить переменной хоста, использовав оператор:

```
EXEC SQL SET :CS = CURRENT SERVER;
```

## **Получение данных из таблиц в формате ASCII**

Когда выполняется распределенный запрос, схему кодировки таблицы результатов определяет сервер. Если распределенный запрос к таблице ASCII поступает на сервер DB2 for OS/390, сервер в ответном сообщении указывает, что столбцы таблицы результатов содержат данные типа ASCII, а не EBCDIC. Ответное сообщение также включает CCSID возвращаемых данных. Этот CCSID – величина, заданная при установке в поле ASCII CODED CHARACTER SET на панели DSNTIPF.

Схема кодировки для вывода результатов зависит от двух факторов:

- Используется ли на реквестере ASCII или EBCDIC

Если формат реквестера ASCII, возвращенные данные выводятся в формате ASCII. Если формат реквестера EBCDIC, возвращенные данные выводятся в формате EBCDIC, даже если они хранились на сервере в формате ASCII. Однако если использованный для получения данных оператор SELECT содержит условие ORDER BY, данные выводятся в формате ASCII.

- Переопределяет ли прикладная программа CCSID для возвращенных данных

Программа, выполняющая динамические операторы SELECT с использованием SQLDA, может для возвращенных данных задать в SQLDA переопределение CCSID.

Если сервер DB2 for OS/390 получает оператор FETCH, он преобразует возвращаемые данные из формата CCSID хранимой таблицы в формат указанного в SQLDA CCSID. Информацию о том, как задавать переопределение CCSID,смотрите в разделе “Изменение CCSID для прочитанных данных” на стр. 570.

## Особенности перехода от доступа по собственному протоколу DB2 к доступу по протоколу DRDA

**Рекомендация:** Где это возможно, переходите от доступа по собственному протоколу DB2 к доступу по протоколу DRDA. Поскольку DB2 поддерживает трехчастные имена, переход к доступу по протоколу DRDA не требует изменения программ. Чтобы программа, которая использует доступ по собственному протоколу DB2, стала использовать доступ DRDA, сделайте следующее:

1. Определите, к каким положениям программа получает доступ.

Для программ со статическими операторами SQL для этого надо найти все операторы SQL, содержащие трехчастные имена и алиасы для трехчастных имен. В трехчастных именах квалификатор высшего уровня – это имя положения. Для потенциальных алиасов запросите таблицу каталога SYSTABLES, чтобы определить, является ли объект алиасом; если это так, надо определить имя положения таблицы, которую алиас представляет. Например:

```
SELECT NAME, CREATOR, LOCATION, TBCREATOR, TBNAME  
      FROM SYSIBM.SYSTABLES  
     WHERE NAME='имя'  
       AND TYPE='A';
```

здесь *имя* – потенциальный алиас.

Для программ с динамическими операторами SQL свяжите пакеты во всех удаленных положениях, к которым пользователи могут иметь доступ с помощью трехчастных имен.

2. Свяжите программу в пакет в каждом положении, упомянутом в программе. Если надо, свяжите также пакет локально.

Для программ, использующих явные операторы CONNECT для связи со другим узлом и затем получающих доступ к третьему узлу с помощью трехчастных имен, свяжите пакет на другом узле с опцией DBPROTOCOL(DRDA), а затем свяжите другой пакет на третьем узле.

3. Свяжите все удаленные пакеты в план с локальным пакетом или модулем DBRM. Свяжите этот план с опцией DBPROTOCOL(DRDA).
4. Проверьте правильность соответствия алиасов.

Для доступа по собственному протоколу DB2 устанавливает соответствие алиасов на узле реквестера. Для доступа по протоколу DRDA, однако, DB2 устанавливает соответствие алиасов на узле, где пакет выполняется. Поэтому может понадобиться определить алиасы для трехчастных имен в удаленных положениях.

Предположим, например, что доступ по протоколу DRDA используется для выполнения программы, содержащей следующий оператор:

```
SELECT * FROM MYALIAS;
```

где MYALIAS – аlias для LOC2.MYID.MYTABLE. DB2 устанавливает соответствие MYALIAS на локальном узле, чтобы определить, что этот оператор должен выполняться на LOC2, но не посылает соответствующее имя на LOC2. Когда оператор выполняется на LOC2, DB2 устанавливает соответствие MYALIAS, используя каталог на LOC2. Если каталог не содержит алиаса MYID.MYTABLE для MYALIAS, оператор SELECT выполняется с ошибкой.

Ситуация может еще сложнее, если трехчастные имена используются для доступа к объектам DB2 с удаленных узлов. Предположим, например, что вы явно установили соединение с LOC2, и используете доступ DRDA для выполнения оператора:

```
SELECT * FROM YRALIAS;
```

где YRALIAS – аlias для LOC3.MYID.MYTABLE. Когда этот оператор SELECT выполняется на LOC3, и LOC2, и LOC3 должны иметь алиас YRALIAS, который соответствует MYID.MYTABLE на LOC3.

5. Если в удаленных положениях, заданных трехчастными именами, для управления временем выполнения распределенных динамических операторов SQL используется ограничение ресурсов, в этих положениях надо изменить таблицы спецификации ограничения ресурсов.

При использовании доступ по собственному протоколу DB2 для ограничения операторов SQL, созданных в удаленном положении, надо задать имена планов. При использовании доступ DRDA для этих целей надо задать имена пакетов. Поэтому в таблицы спецификации ограничения ресурсов в удаленных положениях для пакетов, связанных для доступа по протоколу DRDA, надо добавить строки с трехчастными именами. Кроме того, надо удалить строки, задающие имена планов для доступа по собственному протоколу DB2.

Дополнительную информацию об ограничении ресурсовсмотрите в разделе Раздел 5 (Том 2) *DB2 Administration Guide*.



---

## Раздел 6. Разработка прикладных программ

<b>Глава 6–1. Подготовка прикладной программы к запуску</b>	435
Действия при подготовке программы	435
Шаг 1: Прекомпиляция программы	438
Методы прекомпиляции	438
Программа на входе прекомпилятора	439
Вывод прекомпилятора	439
Опции прекомпилятора	441
Трансляция операторов уровня команд в программе CICS	449
Шаг 2: Связывание прикладной программы	450
Связывание модуля DBRM с пакетом	451
Связывание плана программы	453
Задание пакетов при запуске	454
Использование опций BIND и REBIND для пакетов и планов	457
Использование пакетов с динамическим выбором плана	462
Шаг 3: Компиляция (или ассемблирование) и компоновка прикладной программы	463
Шаг 4: Запуск прикладной программы	465
Командный процессор DSN	465
Запуск программы в среде TSO в основном режиме	466
Запуск пакетной программы DB2 в среде TSO	467
Вызов прикладных программ в командной процедуре (CLIST)	468
Использование процедур JCL для подготовки прикладных программ	469
Доступные процедуры JCL	469
Включение текстов программ из наборов данных SYSLIB	470
Динамический запуск прекомпилятора	471
Альтернативный метод подготовки программ в среде CICS	472
Использование задания JCL для подготовки программ с объектно-ориентированными расширениями	475
Использование ISPF и DB2 Interactive (DB2I)	475
Справка по DB2I	475
Меню основных опций DB2I	476
Панель подготовки программ DB2	477
Панель параметров DB2I по умолчанию 1	482
Панель параметров DB2I по умолчанию 2	485
Панель прекомпиляции	486
Панель BIND PACKAGE	489
Панель BIND PLAN	492
Панели параметров по умолчанию для связывания или повторного связывания плана или пакета	496
Панель типов соединений системы	500
Панели для ввода списков значений	502
Панель подготовки программ: Compile, Link и Run	503
<b>Глава 6–2. Тестирование прикладной программы</b>	507
Создание среды тестирования	507
Создание тестовой структуры данных	507
Заполнение таблиц тестовыми данными	510
Тестирование операторов SQL при помощи SPUFI	511
Отладка программы	511
Отладка программ в TSO	511

Отладка программ в IMS . . . . .	512
Отладка программ в CICS . . . . .	513
Поиск источника ошибки . . . . .	518
Анализ сообщений об ошибках и предупреждений, выданных прекомпилятором . . . . .	519
Выходной поток SYSTEMR прекомпилятора . . . . .	520
Выходной поток SYSPRINT прекомпилятора . . . . .	521
<b>Глава 6–3. Обработка программ среды DL/I batch . . . . .</b>	<b>525</b>
Планирование работы с DL/I batch . . . . .	525
Свойства и функции поддержки DL/I batch в DB2 . . . . .	525
Требования к DB2 в задании DL/I batch . . . . .	526
Авторизация . . . . .	526
Особенности разработки программы . . . . .	526
Адресные пространства . . . . .	527
Принятие . . . . .	527
Операторы SQL и вызовы IMS . . . . .	527
Вызовы контрольных точек . . . . .	527
Синхронизация прикладной программы . . . . .	527
Контрольная точка и использование XRST . . . . .	528
Аварийные завершения вызова синхронизации . . . . .	528
Наборы входных и выходных данных . . . . .	528
Ввод DL/I batch в DB2 . . . . .	528
Вывод DL/I batch в DB2 . . . . .	530
Особенности подготовки программы . . . . .	530
Прекомпиляция . . . . .	530
Связывание . . . . .	530
Компоновка . . . . .	531
Загрузка и запуск . . . . .	531
Перезапуск и восстановление . . . . .	533
Пример JCL пакетного возврата . . . . .	533
Пример JCL перезапуска задания DL/I batch . . . . .	534
Поиск ID контрольной точки DL/I batch . . . . .	535

---

## Глава 6–1. Подготовка прикладной программы к запуску

В прикладных программах DB2 операторы SQL встроены в программы языка хоста. Для пользования этими программами необходимы не только обычные шаги по их подготовке (компилярование, компоновка и запуск), но также шаги по прекомпиляции и связыванию DB2.

**Совет по производительности:** Чтобы избежать переделок, проверьте сначала ваши операторы SQL с помощью утилиты SPUFI, затем скомпилируйте вашу программу без операторов SQL и устранит все ошибки. Затем выполните подготовку и шаги по прекомпилярованию и связыванию DB2.

Поскольку большинство компиляторов не распознают операторы SQL, для предотвращения ошибок компилятора необходимо использовать прекомпилятор DB2 до компиляции программы. Прекомпилятор просматривает программу и возвращает модифицированный исходный код, который можно затем скомпилировать и скомпоновать. Прекомпилятор создает также модуль требований базы данных (DBRM – database request module). Этот DBRM надо связать в пакет или в план с помощью подкоманды BIND. (Сведения о пакетах и планах смотрите в разделе “Глава 5–1. Планирование прекомпиляции и связывания” на стр. 341.) После выполнения этих шагов можно запустить вашу прикладную программу DB2.

В этой главе детально рассматриваются шаги по подготовке вашей прикладной программы к запуску. Она содержит инструкции по основным шагам создания прикладной программы, дополнительным шагам, которые могут понадобиться, и шагам повторного связывания.

---

### Действия при подготовке программы

Чтобы запустить программу со встроенными операторами SQL, необходимо выполнить ряд действий. Они подробно описаны в следующих разделах:

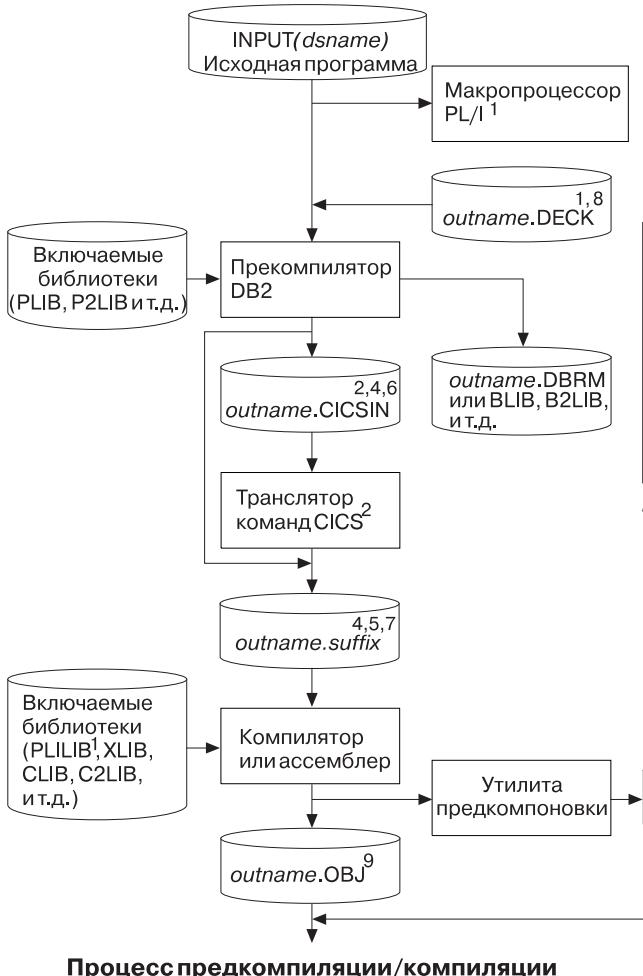
- “Шаг 1: Прекомпиляция программы” на стр. 438
- “Шаг 2: Связывание прикладной программы” на стр. 450
- “Шаг 3: Компиляция (или ассемблирование) и компоновка прикладной программы” на стр. 463
- “Шаг 4: Запуск прикладной программы” на стр. 465.

Как сказано в разделе “Глава 5–1. Планирование прекомпиляции и связывания” на стр. 341, связывать пакет требуется не во всех случаях. Однако в этом руководстве мы будем считать, что вы связываете некоторые из модулей DBRM с пакетами и включаете в план список пакетов.

В среде CICS могут потребоваться дополнительные действия; смотрите разделы:

- Трансляция операторов уровня команд на стр. 449
- Определение программы в среде CICS и RCT на стр. 450
- Сделайте новую копию программы на стр. 469

Есть разные способы управлять ходом подготовки программы. Мы описываем их в разделе “Использование процедур JCL для подготовки прикладных программ” на стр. 469.



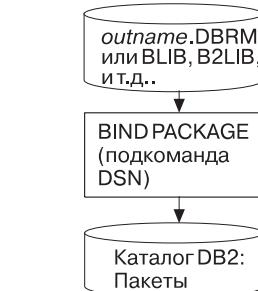
**Обзор процесса подготовки программы DB2, выполняемого DSNH CLIST. В зависимости от программы может потребоваться до четырех стадий.**

На этой схеме обработчики, вызываемые DSNH CLIST, показаны как прямоугольники, а наборы данных, используемые DSNH - как цилинды. Обратите внимание на то, что DSNH CLIST выполняет работу для панелей подготовки программы DB2I. Большинство полей на этих панелях становятся параметрами DSNH.

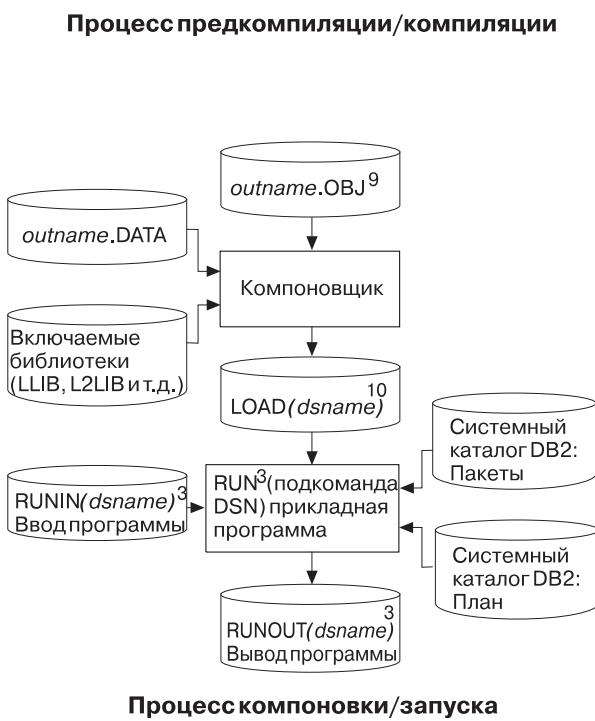
#### Источники файлов листинга

Имя файла	Источник
print.LIST	Макропроцессор PL/I <sup>1</sup> . Этот набор данных заменяется выводом компилятора или ассемблера.
print.PCLIST	Прекомпилятор
print.CXLIST	Транслятор команд CICS <sup>2</sup>
print.LIST	Компилятор или ассемблер <sup>3</sup>
printLINKLIST	Редактор связей
print.RUNLIST	Подкоманда RUN
print.SYSCPRT.LIST	Компилятор C
print.SYSOUT.PRELLIST	Утилита предкомпоновки

print - текущее значение параметра DSNH print



**Процесс связывания пакета**  
(Повторяется для каждого модуля DBRM, связываемого как пакет)



- 1 Необязательный, только для программ PL/I
- 2 Только для программ CICS
- 3 Только для программ TSO и пакетных программ
- 4 'outname' - текущее значение параметра DSNH outname
- 5 'suffix' - ASM, C, CPP, COBOL, COB2, IBMCOB, FORTRAN или PL/I
- 6 outname.CICSIN - исходная программа с включенными командами CICS
- 7 outname.suffix - исходная программа с удаленными операторами SQL и CICS
- 8 DECK - исходная программа после макрообработки PL/I
- 9 OBJ - объектный модуль
- 10 LOAD - загрузочный модуль

Рисунок 106. Процесс подготовки программы. DB2 необходима на этапе связывания и при запуске программы, но не для прекомпиляции.

## Шаг 1: Прекомпиляция программы

Перед компиляцией или созданием объектного файла для программы на языке хоста, необходимо подготовить встроенные в программу операторы SQL. В программах на языках ассемблер, C, COBOL, FORTRAN и PL/I операторы SQL подготавливаются прекомпилятором DB2.

### Внимание

Размер исходной программы, которую может прекомпилировать DB2, ограничен размером региона и доступной прекомпилятору виртуальной памятью. Обычно максимальный размер региона и доступная прекомпилятору виртуальная память – около 8 Мбайт, но это зависит от конкретной установки системы.

### среда CICS

Если программа содержит команды среды CICS, перед компиляцией программу необходимо транслировать. (Смотрите раздел “Трансляция операторов уровня команд в программе CICS” на стр. 449.)

## Методы прекомпиляции

Чтобы запустить процесс прекомпиляции, используйте один из следующих методов:

- Панели DB2I. Используйте панель прекомпиляции или панели подготовки программ DB2.
- Командная процедура DSNH (задание на языке CLIST в среде TSO). Описание соответствующих заданий CLIST смотрите в Главе 2 руководства *DB2 Command Reference*.
- Процедуры JCL, поставляемые с DB2. Дополнительную информацию об этом методе смотрите на странице 469.

Размеры программы на входе и на выходе прекомпилятора не зависят от выбранного метода.

Обработать программу со встроенными операторами SQL с помощью прекомпилятора можно в любое время. При этом DB2 может не быть активна, поскольку прекомпилятор не обращается к табличным каталогам DB2. По этой причине DB2 не проверяет, соответствуют ли имена таблиц и столбцов, используемые в операторах SQL, текущим базам данных DB2, хотя прекомпилятор сверяет их со всеми операторами SQL DECLARE TABLE, входящими в программу. Таким образом, для получения корректных операторов SQL DECLARE TABLE следует использовать генератор объявлений DCLGEN.

Можно несколько раз прекомпилировать и компилировать операторы исходной программы, пока не будут устранены все ошибки, и можно будет проводить компоновку. При этом прекомпилятор может выводить полную диагностику, если задать опции прекомпилятора SOURCE и XREF.

## Программа на входе препроцессора

Исходная программа, впервые обрабатываемая препроцессором DB2, состоит из операторов на языке программирования хоста и встроенных операторов SQL. При составлении операторов на языке хостов и операторов SQL нужно использовать те же размеры полей, что были заданы опцией препроцессора MARGINS.

Входной набор данных SYSIN должен иметь атрибуты RECFM F или FB, LRECL 80.

При помощи оператора SQL INCLUDE можно включать в программу фрагменты из SYSLIB (библиотеки включения). Оператор SQL INCLUDE считывает входные данные из заданного компонента библиотеки SYSLIB, пока не достигает его конца. Ввод из библиотеки INCLUDE может содержать операторы как на языке хоста, так и операторы SQL, но он не должен содержать других операторов препроцессора INCLUDE. SYSLIB должна быть секционированным набором данных с атрибутами RECFM F или FB, LRECL 80.

Исходные операторы для препроцессора DB2 может готовить другой препроцессор, например, макропрограммный PL/I. Все препроцессоры, запускаемые до препроцессора DB2, должны быть в состоянии передавать операторы SQL.

Аналогичным образом другие препроцессоры могут использоваться для обработки программы после препроцессинга, но до компиляции или создания объектного файла. Есть ограничения для форм операторов исходной программы, которые могут быть переданы через препроцессор. Так, константы, комментарии и другие элементы синтаксиса исходной программы, если они неприемлемы для компилятора (например, отсутствие правой скобки для языка C), могут помешать препроцессору сканировать исходную программу и привести к ошибкам. Может оказаться полезно запустить компилятор хоста до препроцессора и отыскать исходные операторы, неприемлемые для компилятора хоста. На этом этапе можно игнорировать сообщения об ошибках компилятора, вызванные операторами SQL. Очистив операторы исходной программы от неприемлемых для компилятора ошибок, можно затем обычным образом подготовить программу DB2 для этого языка хоста.

## Вывод препроцессора

В следующих разделах описаны различные виды вывода препроцессора.

**Вывод программы:** Выходной набор данных SYSPRINT, используемый для листинга программы на выходе препроцессора DB2, имеет атрибуты LRECL 133 и RECFM FBA. Номера операторов на выходе препроцессора всегда соответствуют последовательности их вывода. Но номера операторов, превышающие 32 767, DB2 сохраняет в модуле DBRM, как 0.

Препроцессор выводит в SYSPRINT следующие данные:

- Исходный текст программы на входе препроцессора

Исходные операторы с номерами строк, назначенными препроцессором, если действует опция SOURCE

- Диагностику прекомпилятора DB2

Диагностические сообщения, содержащие назначенные прекомпилятором номера строк с ошибочными операторами

- Список перекрестных ссылок прекомпилятора

Список перекрестных ссылок (если действует опция XREF), содержащий назначенные прекомпилятором номера операторов SQL со ссылками на имена хоста и столбцы.

**Диагностика терминала:** Если есть выходной файл терминала SYSTERM, прекомпилятор выводит в него диагностические сообщения. Там, где это возможно, в зависимости от сути ошибки, сообщения в этом файле сопровождаются копией оператора из исходного текста программы. Благодаря этому часто удается исправить ошибки, не выводя весь файл.

**Модифицированные операторы исходной программы:** Прекомпилятор выводит обрабатываемые им исходные операторы в SYSCIN, входной набор данных для компилятора или ассемблера. Этот набор данных должен иметь атрибуты RECFM F или FB, LRECL 80. Модифицированный прекомпилятором текст исходной программы содержит операторы SQL в виде комментариев и вызовы интерфейса языка DB2.

**Модули требований базы данных:** Главный результат работы прекомпилятора – модуль требований базы данных (DBRM). Этот набор данных содержит операторы SQL и информацию о переменных хоста, извлеченную из исходной программы, а также информацию, которая идентифицирует программу и связывает модуль DBRM с транслированными операторами исходной программы. Он играет затем роль исходного файла в процессе связывания.

Набор данных требует пространства, вмещающего все операторы SQL, плюс пространство для всех имен переменных хоста и некоторой информации заголовка. Для информации заголовка требуется примерно две записи на каждый модуль DBRM, по 20 байтов расходуется на каждую запись SQL и по 6 байтов – на каждую переменную хоста. Точное описание формата DBRM смотрите в макрокоманде отображения DBRM DSNXDBRM в библиотеке *prefix*.SDSNMACS. Атрибуты набора данных DCB – RECFM FB, LRECL 80. Характеристики задает прекомпилятор. Работать с этими наборами данных можно при помощи команд IEBCOPY, IEHPROGM, команд среды TSO COPY и DELETE и других средств работы с секционированными наборами данных.

Процедуры подготовки языка в задании DSNTIJMV (это задание установки, определяющее DB2 для системы MVS) используют параметр DISP=OLD, что обеспечивает целостность данных. Однако при выполнении установки CLIST параметр DISP=OLD для библиотечного набора данных модуля DBRM преобразуется в DISP=SHR, что может привести к нарушениям целостности данных при параллельном запуске нескольких заданий прекомпилятора. Если вы планируете запускать параллельно несколько заданий прекомпилятора и не используете расширенные секционированные наборы данных DBSMSdfp (PDSE), надо изменить процедуры подготовки языка (DSNHCOB, DSNHCOB2, DSNHFOR, DSNHC, DSNHPLI, DSNHASM), задав параметр DISP=OLD вместо DISP=SHR.

**Связывание на другой системе:** Для прекомпиляции программы не обязательно использовать ту же систему DB2, на которой будет проходить связывание модуля DBRM и запуск программы. В частности, можно связать модуль DBRM на уровне текущего выпуска и запустить ее на подсистеме DB2 на уровне предыдущего выпуска, если исходная программа не использует свойств DB2, которые появились только в текущем выпуске. Заведомо можно запускать программы на системе текущего выпуска после связывания на системах уровня предыдущего выпуска.

## Опции прекомпилятора

Можно управлять работой прекомпилятора, задавая при его использовании опции. Эти опции регулируют интерпретацию и обработку текста программы на входе и формат на выходе прекомпилятора.

Опции прекомпилятора DB2 можно задать с помощью операнда DSNH или с помощью опции PARM.PC оператора EXEC JCL. Можно также задавать их из соответствующих панелей DB2I.

Можно прекомпилировать программу, не задавая ничего, кроме имени набора данных, содержащего операторы исходного текста программы. Всем не заданным явно опциям прекомпилятора система DB2 присвоит значения по умолчанию. Это будут те значения, которые были заданы при установке или при поставке DB2.

**Таблица опций прекомпилятора:** Табл. 44 показывает, какие опции можно задать при использовании прекомпилятора, и сокращенные имена команды для этих опций, если они есть. В таблице вертикальная черта (|) используется как разделитель взаимоисключающих опций, а квадратные скобки ([ ]) – как указание на то, что заключенную в них опцию можно опустить.

Таблица 44 (Стр. 1 из 6). Опции прекомпилятора DB2

Ключевое слово опции	Смысл
APOST	Считать апостроф ('') ограничителем строк в операторах языка хоста. Опция доступна не для всех языков; смите Табл. 46 на стр. 448. Опции APOST и QUOTE исключают друг друга. По умолчанию принимается значение поля STRING DELIMITER на панели Application Programming Defaults Panel 1 при установке DB2. Если STRING DELIMITER – апостроф (''), по умолчанию принимается опция прекомпилятора APOST.
APOSTSQL	Считать апостроф ('') ограничителем строк, а кавычки ("") – символом выделения SQL внутри операторов SQL. Для программы на языке COBOL, задавая SQLFLAG, необходимо также задать APOSTSQL. Опции APOSTSQL и QUOTESQL исключают друг друга. По умолчанию берется значение из поля SQL STRING DELIMITER на панели Application Programming Defaults Panel 1 при установке DB2. Если SQL STRING DELIMITER – апостроф (''), в качестве опции прекомпилятора по умолчанию используется APOSTSQL.
ATTACH(TSO CAF RRSAF)	Задает утилиту подключения, при помощи которой программа будет обращаться к DB2. Программы TSO, CAF и RRSAF, которые загружают утилиту подключения, могут при помощи этой опции задавать корректную утилиту подключения, не вставляя в текст программы фиктивную входную точку DSNHLI. Эта опция недоступна для программ на языке FORTRAN. По умолчанию принимается ATTACH(TSO).

Таблица 44 (Стр. 2 из 6). Опции прекомпилятора DB2

Ключевое слово опции	Смысл
COMMA	<p>Считать запятую (,) отделителем дробной части в литералах десятичных дробей и чисел с плавающей запятой. Эта опция применима только к языку COBOL.</p> <p>Если задать эту опцию, нужно будет ставить пробел после каждой запятой, используемой как разделитель списка.</p> <p>Опции COMMA и PERIOD исключают друг друга. По умолчанию запятая или точка выбирается с соответствии со значением DECIMAL POINT IS на панели Application Programming Defaults Panel 1 при установке DB2.</p>
CONNECT(2 1) CT(2 1)	<p>Определяет, какие правила оператора CONNECT применять – тип 1 или тип 2.</p> <p>CONNECT(2) По умолчанию: Применять правила для оператора CONNECT (Type 2).</p> <p>CONNECT(1) Применять правила для оператора CONNECT (Type 1)</p> <p>Если не задать опцию CONNECT во время прекомпиляции программы, будут применяться правила оператора CONNECT (Type 2). Дополнительную информацию об этой опции смотрите в разделе “Опции прекомпилятора” на стр. 413, а сравнение CONNECT (Type 1) и CONNECT (Type 2) приводится в Главе 6 руководства <i>DB2 SQL Reference</i>.</p>
DATE(ISO USA EUR JIS LOCAL)	<p>Назначает формат для всех дат на выходе, независимо от действующего в системе формата по умолчанию. Описание форматов дат смотрите в Главе 3 руководства <i>DB2 SQL Reference</i>.</p> <p>По умолчанию принимается формат даты из поля DATE FORMAT на панели Application Programming Defaults Panel 2 при установке DB2.</p> <p>Нельзя использовать подопцию LOCAL, если у вас нет обработчика дат.</p>
DEC(15 31)	<p>Задает максимальную точность десятичных арифметических операций.</p> <p>Смотрите раздел “Выбор 15– или 31–значной точности для десятичных чисел” на стр. 35.</p> <p>По умолчанию принимается точность, заданная в поле DECIMAL ARITHMETIC на панели Application Programming Defaults Panel 1 при установке DB2.</p>
FLAG(I W E S)	<p>Подавляет диагностические сообщения ниже заданного уровня важности (I – информационные сообщения, W – предупреждения, E – ошибки и S – серьезные ошибки при кодах важности 0, 4, 8 и 12 соответственно).</p> <p>По умолчанию принимается FLAG(I).</p>
FLOAT(S390 IEEE)	<p>Определяет формат переменных хоста с плавающей точкой в программах на языках C, C++ или ассемблере: IEEE или System/390. DB2 игнорирует эту опцию, если значение HOST отличается от ASM, C и CPP.</p> <p>Значение по умолчанию – FLOAT(S390).</p>
GRAPHIC	<p>Указывает, что в исходном тексте программы могут встретиться смешанные данные, и что X'0E' и X'0F' являются специальными управляющими символами shift-out (переход к двухбайтной кодировке DBCS) и shift-in (переход к однобайтной кодировке SBCS) для данных в коде EBCDIC.</p> <p>Опции GRAPHIC и NOGRAPHIC исключают друг друга. По умолчанию выбором между GRAPHIC и NOGRAPHIC управляет поле MIXED DATA на панели Application Programming Defaults Panel 1 при установке DB2.</p>

Таблица 44 (Стр. 3 из 6). Опции прекомпилятора DB2

Ключевое слово опции	Смысл
HOST(ASM C (FOLD)  CPP (FOLD)  COBOL COB2 IBMCOB PLI FORTRAN)	<p>Задает язык хоста, содержащий операторы SQL. Значение COBOL используйте только для OS/VS COBOL. Для VS COBOL II используйте значение COB2. Для IBM SAA AD/Cycle COBOL/370 и IBM COBOL for MVS &amp; VM используйте IBMCOB.</p> <p>Для языка C задайте:</p> <ul style="list-style-type: none"> <li>• C, если не хотите, чтобы DB2 переводила символы нижнего регистра в составе обычных идентификаторов SQL с однобайтным набором символов в верхний регистр</li> <li>• C(FOLD), если нужно, чтобы DB2 переводила символы нижнего регистра в составе обычных идентификаторов SQL с однобайтным набором символов в верхний регистр</li> </ul> <p>Для языка C++ задайте:</p> <ul style="list-style-type: none"> <li>• CPP, если не хотите, чтобы DB2 переводила символы нижнего регистра в обычных идентификаторах SQL с однобайтным набором символов в верхний регистр</li> <li>• CPP(FOLD), если нужно, чтобы DB2 переводила символы нижнего регистра в составе обычных идентификаторов SQL с однобайтным набором символов в верхний регистр</li> </ul> <p>Если опустить опцию HOST, прекомпилятор выдаст диагностическое сообщение четвертого уровня и будет использовать для этой опции значение по умолчанию. Это значение берется из поля LANGUAGE DEFAULT на панели Application Programming Defaults Panel 1 при установке DB2.</p> <p>Эта опция также задает значения по умолчанию, специфичные для языка; смотрите Табл. 46 на стр. 448.</p>
LEVEL[(aaaa)] L	<p>Задает уровень модуля, где aaaa – произвольная алфавитно–цифровая строка длиной до семи символов. Эта опция не рекомендуется для обычного использования, и ее не поддерживают панели DSNH CLIST и DB2I. Дополнительные сведения смотрите в разделе “Задание уровня программы” на стр. 457.</p> <p>Для ассемблера, C, C++, FORTRAN, и PL/I можно опустить подопцию (aaaa). При этом маркер согласованности будет пустым. Для языка COBOL прекомпилятор проигнорирует опцию LEVEL без подопции.</p>
LINECOUNT( <i>n</i> ) LC	Определяет, что на странице вывода прекомпилятора будет <i>n</i> строк. При этом учитываются и вставляемые прекомпилятором строки заголовка. По умолчанию принимается LINECOUNT(60).
MARGINS( <i>m,n[,c]</i> ) MAR	<p>Задает, какую часть в каждой исходной записи занимают операторы языка хоста или операторы SQL; для ассемблера эта опция указывает, где начинаются продолжения столбцов. Первая подопция (<i>m</i>) – начало столбца для операторов. Вторая подопция (<i>n</i>) – конец столбца для операторов. Третья подопция (<i>c</i>) указывает ассемблеру, где начинаются продолжения. Если она не указана, прекомпилятор помещает указатель продолжения в столбце сразу после конца столбца. Значения параметров должны находиться в диапазоне от 1 до 80.</p> <p>Значения по умолчанию зависят от задаваемой вами опции HOST; смотрите Табл. 46 на стр. 448.</p> <p>Эту опцию не поддерживают панели DSNH CLIST и DB2I. Если исходная программа на ассемблере содержит инструкцию ICTL, опция MARGINS должна быть согласована с ней.</p>

Таблица 44 (Стр. 4 из 6). Опции прекомпилятора DB2

Ключевое слово опции	Смысл
NOFOR	<p>Устраняет необходимость в условии FOR UPDATE OF в статическом операторе SQL. Опция NOFOR разрешает вашей программе выполнять позиционированные изменения во всех столбцах таблиц, в которые DB2 позволяет программе вносить изменения. Если вы затем используете условие FOR UPDATE OF, оно ограничит изменения столбцами, перечисленными в этом условии, и затребует блокировки для изменения.</p> <p>Без опции NOFOR (значение по умолчанию) все запросы, появляющиеся в операторе DECLARE CURSOR, должны содержать условие FOR UPDATE OF, если вы используете указатель для позиционированных изменений. В условии нужно назвать все столбцы, которые смогут изменять указатель.</p> <p>Эту опцию применяют при использовании опции STDSQL(YES).</p> <p>Если полученный модуль DBRM оказывается очень большим, для этой опции потребуется дополнительная память.</p>
NOGRAPHIC	<p>Указывает, что X'0E' и X'0F' можно использовать в строке, и они не являются управляющими символами.</p> <p>Опции GRAPHIC и NOGRAPHIC исключают друг друга. По умолчанию выбором между GRAPHIC и NOGRAPHIC управляет поле MIXED DATA на панели Application Programming Defaults Panel 1 при установке DB2.</p>
NOOPTIONS NOOPTN	Подавляет вывод опций прекомпилятора.
NOSOURCE NOS	Подавляет вывод исходного текста прекомпилятора. Эта опция принимается по умолчанию.
NOXREF NOX	Подавляет вывод списка перекрестных ссылок прекомпилятора. Эта опция принимается по умолчанию.
ONEPASS ON	<p>Проводит обработку в один проход, сокращая затраты времени. Чтобы не возникало необходимости во втором проходе, объявления надо делать до ссылок SQL.</p> <p>По умолчанию выбор одного или двух проходов зависит от заданной опции HOST;смотрите Табл. 46 на стр. 448.</p> <p>Опции ONEPASS и TWOPASS исключают друг друга.</p>
OPTIONS OPTN	Выводит опции прекомпилятора. Эта опция принимается по умолчанию.
PERIOD	<p>Считать отделителем дробной части в литералах десятичных дробей точку (.).Эта опция применима не для всех языков; смотрите Табл. 46 на стр. 448.</p> <p>Опции COMMA и PERIOD исключают друг друга. По умолчанию запятая или точка выбирается по значению в поле DECIMAL POINT IS на панели Application Programming Defaults Panel 1 при установке DB2.</p>
QUOTE Q	<p>Считать кавычки ("") ограничителем строк <i>внутри операторов на языке хоста</i>. Эта опция применима только для языка COBOL.</p> <p>По умолчанию берется значение из поля STRING DELIMITER на панели Application Programming Defaults Panel 1 при установке DB2. Если STRING DELIMITER – кавычки ("") или DEFAULT, QUOTE – опция прекомпилятора по умолчанию.</p> <p>Опции APOST и QUOTE исключают друг друга.</p>

Таблица 44 (Стр. 5 из 6). Опции прекомпилятора DB2

Ключевое слово опции	Смысл
QUOTESQL	<p>Считать кавычки ("") ограничителем строк, а апостроф ('') – символом выделения SQL <i>внутри операторов SQL</i>. Эта опция применима только для языка COBOL.</p> <p>По умолчанию принимается значение в поле SQL STRING DELIMITER на панели Application Programming Defaults Panel 1 при установке DB2. Если SQL STRING DELIMITER – кавычки ("") или DEFAULT, QUOTESQL – опция прекомпилятора по умолчанию.</p> <p>APOSTSQL и QUOTESQL исключают друг друга.</p>
SOURCE S	Выводить исходную программу и диагностику прекомпилятора.
SQL(ALL DB2)	<p>Сообщает, содержит ли исходный текст программы такие операторы SQL, которые не опознаются системой DB2 for OS/390.</p> <p>Рекомендуется задавать SQL(ALL) для программ, в которых операторы SQL должны выполняться на сервере, отличном от DB2 for OS/390, при помощи доступа по протоколу DRDA. В этом случае прекомпилятор примет все операторы, включая не согласующиеся с синтаксическими правилами DB2. Прекомпилятор будет интерпретировать и обрабатывать операторы SQL согласно правилам протокола DRDA. Прекомпилятор выдаст также информационное сообщение, если программа попытается использовать зарезервированные слова IBM SQL в качестве обычных идентификаторов. SQL(ALL) не влияет на ограничения прекомпилятора.</p> <p>Опция SQL(DB2), которая используется по умолчанию, поручает прекомпилятору интерпретировать операторы SQL и проверять их синтаксис для использования DB2 for OS/390. SQL(DB2) рекомендуется, когда в качестве сервера программ будет использоваться DB2 for OS/390.</p>
SQLFLAG(IBM STD [(ssname [,qualifier]])]	<p>Задает стандарт, по которому будет проверяться синтаксис операторов SQL. Когда операторы будут отклоняться от стандарта, на выходе прекомпилятора будут появляться информационные сообщения (флаги). Опция SQLFLAG не зависит от других опций прекомпилятора, включая SQL и STDSQL. Но если ваша программа написана на языке COBOL, и вы зададите SQLFLAG, следует также задать APOSTSQL.</p> <p>Подопция IBM означает проверку операторов SQL по синтаксису IBM SQL Версии 1. Можно также использовать SAA, как в выпусках ранее Версии 6.</p> <p>Подопция STD означает проверку операторов SQL по синтаксису введенного уровня стандарта SQL. Можно также использовать значение 86, как в выпусках ранее Версии 6.</p> <p>Параметр <i>ssname</i> означает требование семантической проверки, причем для доступа к каталогу будет использоваться указанная подсистема DB2. Если не задать параметр <i>ssname</i>, прекомпилятор будет проверять только синтаксис.</p> <p>Параметр <i>qualifier</i> задает спецификатор при вставке флагов. Чтобы задать <i>qualifier</i>, нужно также задать перед ним <i>ssname</i>. Если параметр <i>qualifier</i> не задан, по умолчанию принимается ID авторизации процесса, запустившего прекомпилятор.</p>

Таблица 44 (Стр. 6 из 6). Опции прекомпилятора DB2

Ключевое слово опции	Смысл
STDSQL(NO YES)	<p>Выбирает правила, которым должны соответствовать операторы на выходе.</p> <p>При STDSQL(YES)<sup>4</sup> после прекомпиляции операторы SQL из исходной программы должны будут соответствовать определенным правилам стандарта SQL.<sup>4</sup> При STDSQL(NO) будут действовать правила DB2.</p> <p>По умолчанию принимается значение из поля STD SQL LANGUAGE на панели Application Programming Defaults Panel 2 при установке DB2.</p> <p>Опция STDSQL(YES) автоматически влечет опцию NOFOR.</p>
TIME(ISO USA  EUR JIS LOCAL)	<p>Назначает определенный формат для вывода времени, независимо от действующего в подсистеме формата по умолчанию. Описание форматов времени смотрите в Главе 3 руководства <i>DB2 SQL Reference</i>.</p> <p>По умолчанию принимается формат из поля TIME FORMAT на панели Application Programming Defaults Panel 2 при установке DB2.</p> <p>Подопцию LOCAL можно использовать только при наличии программы—обработчика времени.</p>
TWOPASS TW	<p>Задает обработку в два прохода, благодаря чему можно использовать объявления после ссылок на них. По умолчанию выбор одного или двух проходов зависит от заданной опции HOST; смотрите Табл. 46 на стр. 448.</p> <p>Опции ONEPASS и TWOPASS исключают друг друга.</p>
VERSION(aaaa AUTO)	<p>Определяет идентификатор версии пакета, программы и получаемого модуля DBRM. Если опция VERSION задана, прекомпилятор создаст идентификатор версии в программе и в DBRM. Это повлияет на размер модуля загрузки и модуля DBRM. DB2 использует идентификатор версии, если вы свяжете модуль DBRM с планом или пакетом.</p> <p>Если не задать версию на этапе прекомпиляции, по умолчанию идентификатор версии будет пустой строкой. Если задать AUTO, прекомпилятор создаст идентификатор версии на основе маркера согласованности. Если маркер согласованности представляет собой отметку времени, отметка времени будет преобразована в символьный формат ISO и в таком виде использована как идентификатор версии. Используемая отметка времени основана на значении часов System/370™ Store Clock. Сведения об использовании опции VERSION смотрите в разделе “Идентификация версии пакета” на стр. 456.</p>
XREF	Выводить отсортированный список перекрестных ссылок, использованных в операторах SQL.

**Значения опций прекомпилятора DB2 по умолчанию:** У некоторых опций прекомпилятора значения по умолчанию основываются на значениях, заданных на панелях Application Programming Defaults. В Табл. 45 на стр. 447 приведены эти опции прекомпилятора и значения по умолчанию:

<sup>4</sup> Как и в прежних версиях DB2, можно использовать STDSQL(86). Прекомпилятор обрабатывает это значение так же, как STDSQL(YES).

Таблица 45. Опции прекомпилятора по умолчанию при стандартной установке IBM. Эти значения по умолчанию можно изменить в процессе установки.

Опции установки (DSNTIPF)	Значения по умолчанию при установке	Эквивалентная опция прекомпилятора	Доступные опции прекомпилятора
STRING DELIMITER	кавычки ("")	QUOTE	APOST QUOTE
SQL STRING DELIMITER	кавычки ("")	QUOTESQL	APOSTSQL QUOTESQL
DECIMAL POINT IS	PERIOD	PERIOD	COMMA PERIOD
DATE FORMAT	ISO	DATE(ISO)	DATE(ISO USA  EUR JIS LOCAL)
DECIMAL ARITHMETIC	DEC15	DEC(15)	DEC(15 31)
MIXED DATA	NO	NOGRAPHIC	GRAPHIC NOGRAPHIC
LANGUAGE DEFAULT	COBOL	HOST(COBOL)	HOST(ASM C[(FOLD)]  CPP[(FOLD)]  COBOL COB2 IBMCOB  FORTRAN PLI)
STD SQL LANGUAGE	NO	STDSQL(NO)	STDSQL(YES NO 86)
TIME FORMAT	ISO	TIME(ISO)	TIME(IS USA EUR  JIS LOCAL)

Примечание к Табл. 45:

Для динамических операторов SQL еще одно программное значение по умолчанию, USE FOR DYNAMICRULES, определяет, какие значения по умолчанию – программы или прекомпилятора – будет использовать система DB2 для следующих опций установки:

- STRING DELIMITER
- SQL STRING DELIMITER
- DECIMAL POINT IS
- DECIMAL ARITHMETIC
- MIXED DATA

Если значение опции USE FOR DYNAMICRULES – YES, динамические операторы SQL будут использовать значения программы. Если значение опции USE FOR DYNAMICRULES – NO, динамические операторы SQL в пакетах и планах при связывании, определении и вызове будут использовать опции прекомпилятора. Описание использования опций при связывании, определении и вызовах смотрите в разделе “Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES” на стр. 457.

У некоторых опций прекомпилятора значения по умолчанию зависят от языка хоста. Некоторые опции применимы не для всех языков. В Табл. 46 на стр. 448 приведены опции, зависящие от языка, и значения по умолчанию.

Таблица 46. Опции прекомпилятора, зависящие от языка, и их значения по умолчанию

язык	Значения по умолчанию
Assembler	APOST1, APOSTSQL1, PERIOD1, TWOPASS, MARGINS(1,71,16)
C или CPP	APOST1, APOSTSQL1, PERIOD1, ONEPASS, MARGINS(1,72)
COBOL, COB2 или IBMCOB	QUOTE <sup>2</sup> , QUOTESQL <sup>2</sup> , PERIOD, ONEPASS <sup>1</sup> , MARGINS(8,72) <sup>1</sup>
FORTRAN	APOST1, APOSTSQL1, PERIOD1, ONEPASS <sup>1</sup> , MARGINS(1,72) <sup>1</sup>
PL/I	APOST1, APOSTSQL1, PERIOD1, ONEPASS, MARGINS(2,72)

Примечания к Табл. 46

1. Единственно возможное значение для этого языка.
2. Значение по умолчанию выбирается на панели Application Programming Defaults Panel 1 при установке DB2. При стандартной установке IBM значения по умолчанию для ограничителей строк – QUOTE (ограничителем языка хоста служат кавычки) и QUOTESQL (символом выделения SQL служат кавычки). Во время установки можно вместо стандартных значений по умолчанию IBM задать другие значения по умолчанию. При задании опций прекомпилятора действовавшие значения по умолчанию будут проигнорированы.

**Значения по умолчанию прекомпилятора для динамических операторов:**

В основном динамические операторы используют значения по умолчанию, заданные на панели установки DSNTIPF. Но если у опции DSNHDECP параметр DYNRULS равен NO, можно использовать эти опции прекомпилятора для динамических операторов SQL в пакетах и планах, в которых выполняется связывание, определение и вызовы:

- COMMA или PERIOD
- APOST или QUOTE
- APOSTSQL или QUOTESQL
- GRAPHIC или NOGRAPHIC
- DEC(15) или DEC(31)

## Трансляция операторов уровня команд в программе CICS

### Среда CICS

**Трансляция операторов уровня команд:** программы CICS можно транслировать с помощью транслятора языка команд CICS на этапе подготовки программы. (Трансляторы языка команд CICS доступны только для ассемблера, языка C, COBOL и PL/I; для языка FORTRAN транслятор в настоящее время отсутствует.) Для подготовки программ CICS возможны следующие последовательности операций:

**Использовать сначала прекомпилятор DB2,** а затем транслятор языка команд CICS. Эта предпочтительная последовательность для подготовки программы, и ее поддерживают панели DB2I Program Preparation. Если вы используете для подготовки программы панели DB2I, вы можете задать опции транслятора автоматически, не создавая отдельной строки опций. Дальнейшее описание утилиты DB2I и ее использования в подготовке программы смотрите в разделе "Использование ISPF и DB2 Interactive (DB2I)" на стр. 475.

**Использовать сначала транслятор языка команд CICS,** а затем прекомпилятор DB2. При такой последовательности транслятор CICS выдает предупреждение для каждого встреченного им оператора EXEC SQL. Эти сообщения не влияют на результат трансляции. Если вы используете набор двухбайтных символов (DBCS), мы рекомендуем проводить прекомпиляцию до трансляции, как описано выше.

### Требования к программе и к процессу:

Используйте опцию прекомпилятора NOGRAPHIC, чтобы прекомпилятор не принял ошибочно символы на выходе транслятора CICS за графические данные.

Если исходная программа написана на языке COBOL, нужно задать один и тот же ограничитель строк для прекомпилятора DB2, компилятора COBOL и транслятора CICS. Значения по умолчанию для прекомпилятора DB2 и компилятора COBOL не совместимы со значениями по умолчанию для транслятора CICS.

Если операторы SQL в вашей исходной программе содержат ссылки на переменные хоста, к которым адресуется указатель, записанный в CICS TWA, нужно сделать переменные хоста адресуемыми для TWA, прежде чем выполнять такие операторы. Например, программа на языке COBOL может обеспечить адресуемость для TWA, выполнив следующий оператор:

```
EXEC CICS ADDRESS  
    TWA (адрес-области-twa)  
END-EXEC
```

### Среда CICS (продолжение)

Программы CICS можно запускать только из адресных пространств CICS. Это ограничение распространяется на опцию RUN во втором командном процессоре программы DSN. Все эти возможности встречаются в TSO.

Для подготовки программы можно присоединить задания JCL, созданные панелями DB2 Program Preparation, к заданиям JCL транслятора CICS. Для запуска подготовленной программы в среде CICS может понадобиться обновить таблицу RCT и определить программы и транзакции в среде CICS. Ваш системный программист должен создать соответствующую таблицу управления ресурсами (RCT) и ресурс или табличные записи CICS. Сведения о необходимых ресурсных записяхсмотрите в Разделе 2 руководства *DB2 Installation Guide* и *CICS for MVS/ESA Resource Definition Guide*.

В *prefix.SDSNSAMP* приводятся примеры заданий JCL, используемых для подготовки и запуска программы CICS, содержащей операторы SQL. Список имен программ CICS и членов JCL смотрите в Табл. 123 на стр. 888. В задание JCL входят:

- обработка макропроцессором PL/I
- прекомпиляция DB2
- трансляция командного языка CICS
- компиляция операторов исходной программы на исходном языке хоста
- компоновка программы, полученной после компиляции
- связывание модуля DBRM
- запуск подготовленной программы.

## Шаг 2: Связывание прикладной программы

Прежде чем запускать прикладную программу DB2, необходимо связать модуль DBRM, созданный прекомпилятором, с планом или пакетом, План может состоять из модулей DBRM, списка пакетов, задающего пакеты и собрания пакетов или комбинации модулей DBRM со списком пакетов. План должен содержать по меньшей мере один пакет или один непосредственно связанный модуль DBRM. Каждый связываемый пакет может содержать только один модуль DBRM.

### Исключение

Нет необходимости связывать модуль DBRM, в котором все операторы SQL выбраны из следующего списка:

```
CONNECT  
COMMIT  
ROLLBACK  
DESCRIBE TABLE  
RELEASE  
SET CONNECTION  
SET CURRENT PACKAGESET  
SET переменная-хоста = CURRENT PACKAGESET  
SET переменная-хоста = CURRENT SERVER
```

Планы нужно связывать локально, даже если в них есть пакеты ссылок, запускаемые удаленно. Но пакеты, запускаемые на удаленных подсистемах, нужно связывать на этих подсистемах.

Из реквестера DB2 можно запустить план, назвав его в подкоманде RUN, но нельзя запустить непосредственно пакет. Пакет нужно включить в план, а затем запустить этот план.

## Связывание модуля DBRM с пакетом

При связывании пакета вы задаете собрание, которому принадлежит пакет. Собрание не является физической сущностью, и вы его не создаете; имя собрания – это просто удобный способ сослаться на группу пакетов.

Чтобы связать пакет, нужно располагать соответствующими правами.

## Связывание пакетов на удаленной подсистеме

Если прикладная программа обращается к данным по протоколу DRDA, нужно связать пакеты на тех системах, на которых они будут запускаться. На своей локальной системе нужно связать план, у которого в списке пакетов будут все пакеты, локальные и удаленные.

Чтобы связать пакет на удаленной системе DB2, нужно иметь на этой системе те же права, которые нужны для связывания пакетов на локальной системе. Чтобы связать пакет на системе другого типа, например, на SQL/DS, необходимы все права, которые требуются на этой системе для выполнения ее операторов SQL и использования ее объектов данных.

Процесс связывания для удаленного пакета не отличается от связывания локального пакета, за исключением того, что локальная база данных связей должна быть в состоянии опознавать имя подсистемы, которым вы пользуетесь для определения удаленной подсистемы. Например, чтобы связать DBRM PROGA на подсистеме PARIS в собрании GROUP1, введите:

```
BIND PACKAGE(PARIS.GROUP1)
    MEMBER(PROGA)
```

Затем включите удаленный пакет в список пакетов локального плана, например, плана PLANB:

```
BIND PLAN (PLANB)
    PKLIST(PARIS.GROUP1.PROGA)
```

Во время связывания и повторного связывания DB2 проверяет права доступа, читает и обновляет каталог, создает пакет в каталоге на удаленном узле. DB2 не читает и не обновляет каталоги и не проверяет права доступа на локальном узле.

Если задается опция EXPLAIN(YES) и не задается опция SQLERROR(CONTINUE), в подсистеме, заданной в подкоманде BIND или REBIND, должна существовать таблица PLAN\_TABLE. Эта подсистема может также быть подсистемой по умолчанию.

Если при связывании задается опция COPY, должны существовать локальные права COPY. DB2 выполняет проверку авторизации, читает и обновляет каталог, создает пакет в каталоге на удаленном узле. DB2 читает записи

каталога, относящиеся к скопированному пакету на локальном узле. Если локальный узел установлен с форматом времени или даты LOCAL, и пакет создается на удаленном узле с помощью COPY, опция COPY заставляет DB2 на удаленном узле преобразовать значения, возвращаемые удаленным узлом в формате ISO, если оператор SQL не задает другой формат.

Связав пакет, можно его связать повторно, освободить или связать с опцией REPLACE при помощи локального или удаленного связывания.

## **Превращение существующего плана в пакеты, запускаемые удаленно**

Если вы уже использовали DB2, у вас могли остаться прикладные программы, которые нужно запускать на удаленной подсистеме с доступом по протоколу DRDA. Для этого вам нужно повторно связать модули DBRM в текущем плане как пакеты на удаленной подсистеме. Кроме того, понадобится новый план, у которого в списке пакетов будут эти удаленные пакеты.

Для каждой удаленной подсистемы выполните следующие действия:

1. Выберите имя для собрания, в которое войдут все пакеты плана, например, REMOTE1. (Можно при желании использовать несколько собраний, но достаточно и одного.)
2. Если сервер – система DB2, выполните на удаленной подсистеме:
  - a. GRANT CREATE IN COLLECTION REMOTE1 TO *имя–авторизации*;
  - b. GRANT BINDADD TO *имя–авторизации*;где *имя–авторизации* – имя владельца пакета.
3. Свяжите каждый модуль DBRM как пакет на удаленной подсистеме, используя инструкции в разделе “Связывание пакетов на удаленной подсистеме” на стр. 451. Ко времени запуска владелец пакета должен располагать всеми правами доступа к данным, требуемыми на удаленной подсистеме. Если владелец не имеет этих прав на этапе связывания, используйте опцию VALIDATE(RUN). Опция позволит вам создать пакет, даже если проверка авторизации не пройдет. DB2 вновь проверит права доступа во время запуска.
4. Свяжите новый план программы в локальной системе DB2, используя такие опции:

PKLIST (*имя–положения.REMOTE1.\**)  
CURRENTSERVER (*имя–положения*)

где *имя–положения* – значение опции LOCATION в SYSIBM.SYSLOCATIONS в локальной DB2, которая обозначает удаленную подсистему, где будет происходить запуск. Нет необходимости связывать модули DBRM непосредственно с этим планом: достаточно списка пакетов.

Теперь, когда вы запустите старую программу на локальной DB2, используя новый план программы, произойдет следующее:

- Вы немедленно соединитесь с удаленной подсистемой, названной в опции CURRENTSERVER.
- Перед запуском пакета DB2 проведет его поиск в собрании REMOTE1 на удаленной подсистеме.

- Все операторы UPDATE, DELETE и INSERT в вашей программе будут действовать на таблицы в удаленной подсистеме.
- Все результаты операторов SELECT будут возвращены вашей старой программе, которая обработает их так же, как если бы они поступили с вашей локальной DB2.

## **Связывание плана программы**

Чтобы связать модули DBRM и списки пакетов с планом, используйте подкоманду BIND PLAN. Синтаксис и полные описания BIND PLAN смотрите в Главе 3 руководства *DB2 Command Reference*.

### **Связывание модулей DBRM непосредственно с планом**

План может содержать модули DBRM, связанные непосредственно с ним. Чтобы связать три модуля DBRM – PROGA, PROGB и PROGC – непосредственно с планом PLANW, введите:

```
BIND PLAN(PLANW)
    MEMBER(PROGA,PROGB,PROGC)
```

В план можно включить любое число модулей DBRM. Но если вы используете много модулей DBRM в одном плане (например, больше 500), у вас могут возникнуть трудности с обслуживанием плана. Чтобы облегчить работу с планом, можно связать каждый модуль DBRM с отдельным пакетом, задав одно собрание для всех связываемых пакетов, и затем связать план, задав это собрание в списке пакетов плана. Если структура программы не позволяет использовать этот метод, узнайте, не может ли ваш системный администратор увеличить размер пула EDM, чтобы он стал хотя бы в 10 раз больше самого большого дескриптора базы данных (DBD) и плана.

### **Включение пакетов в список пакетов**

Чтобы включить пакеты в список пакетов плана, перечислите их после ключевого слова PKLIST в BIND PLAN. Чтобы включить в список целое собрание пакетов, поставьте звездочку после имени собрания. Например,

```
PKLIST(GROUP1.*)
```

Чтобы связать модули DBRM непосредственно с планом, а также включить пакеты в список пакетов, используйте и MEMBER, и PKLIST. В приведенных ниже примерах используются:

- модули DBRM PROG1 и PROG2
- все пакеты в собрании, названном TEST2
- пакеты PROGA и PROGC в собрании GROUP1

```
MEMBER(PROG1,PROG2)
PKLIST(TEST2.* ,GROUP1.PROGA,GROUP1.PROGC)
```

Необходимо задать опции MEMBER, PKLIST или обе. Получаемый план может соответственно иметь следующий состав:

- только программы, связанные с модулями DBRM в списке MEMBER
- только программы, связанные с пакетами и собраниями, обозначенными в PKLIST
- сочетание указанного в MEMBER и PKLIST

## **Задание пакетов при запуске**

На этапе препроцессинга программы, содержащей операторы SQL, препроцессор помечает все обращения к DB2 *маркером согласованности*. Тот же маркер отмечает модуль DBRM, создаваемый препроцессором, и план или пакет, с которым вы связываете DBRM. При запуске программы система DB2 использует маркер согласованности, чтобы соотнести обращение к DB2 с нужным модулем DBRM.

(Обычно маркер согласованности имеет внутренний формат DB2. Этот маркер можно переписать по собственному усмотрению; смотрите раздел “Задание уровня программы” на стр. 457.)

Нужны также и другие идентификаторы. Хотя маркер согласованности – уникальная метка модуля DBRM, связанного непосредственно с планом, он не гарантирует уникальную идентификацию пакета. Когда вы связываете модули DBRM непосредственно с конкретным планом, вы связываете каждый модуль только один раз. Но можно связать DBRM со многими пакетами, в разных подсистемах и в разных собраниях; тогда можно включить все пакеты в список пакетов одного и того же плана. Все эти пакеты будут иметь общий маркер согласованности. Естественно, есть способы указать конкретные подсистемы и конкретное собрание в момент запуска.

### **Идентификация подсистемы**

Когда программа исполняет оператор SQL, DB2 для определения подсистемы нужного пакета или модуля DBRM использует значение специального регистра CURRENT SERVER. Если текущий сервер – ваша локальная система DB2, и у нее нет имени подсистемы, это значение будет пусто.

Значение CURRENT SERVER можно изменить из программы с помощью оператора SQL CONNECT. Если вы не используете CONNECT, значение CURRENT SERVER – это имя локальной подсистемы DB2 (или пусто, если имя подсистемы DB2 не задано).

### **Идентификация собрания**

Когда ваша программа исполняет оператор SQL, DB2 использует в качестве имени собрания для необходимого пакета значение специального регистра CURRENT PACKAGESET. Чтобы задать или изменить это значение из программы, используйте оператор SQL SET CURRENT PACKAGESET.

Если вы не используете SET CURRENT PACKAGESET, значение в регистре при запуске программы будет пустым. (Таким оно и останется.) В этом случае может оказаться важным порядок, в котором DB2 ищет доступные собрания.

При вызове хранимой процедуры специальный регистр CURRENT PACKAGESET содержит значение столбца COLLID таблицы каталога SYSPROCEDURES. Когда управление возвращается от хранимой процедуры к вызвавшей ее программе, DB2 восстанавливает значение, хранившееся в CURRENT PACKAGESET перед вызовом.

## **Порядок поиска**

Порядок, в котором вы зададите пакеты в списке пакетов, может повлиять на производительность при запуске программы. Поиск конкретного пакета подразумевает поиск в каталоге DB2, что может быть связано с большими затратами. Если вы используете id—собрания.\* с ключевым словом PKLIST, следует вначале задать собрания, в которых шансы найти пакет максимальны.

Например, если вы выполните следующее связывание: BIND PLAN (PLAN1) PKLIST (COL1.\*., COL2.\*., COL3.\*., COL4.\*.) и затем запустите программу PROG1, DB2 будет действовать так:

1. проверит, нет ли программы PROG1, связанной как часть плана
2. проведет поиск пакета COL1.PROG1.*отметка–времени*
3. если пакет COL1.PROG1.*отметка–времени* не будет найден, проведет поиск пакета COL2.PROG1.*отметка–времени*
4. если пакет COL2.PROG1.*отметка–времени* не будет найден, проведет поиск пакета COL3.PROG1.*отметка–времени*
5. если пакет COL3.PROG1.*отметка–времени* не будет найден, проведет поиск COL4.PROG1.*отметка–времени*.

**Если специальный регистр CURRENT PACKAGESET пуст, DB2** проведет поиск модуля DBRM или пакета в одной из следующих последовательностей:

- *На локальной подсистеме* (если CURRENT SERVER пуст или называет эту подсистему явно) порядок такой:
  1. Все модули DBRM, связанные непосредственно с планом.
  2. Все пакеты, уже размещенные в плане, пока план выполняется.
  3. Все не размещенные пакеты, названные явно в списке пакетов плана, и все собрания, полностью включенные в список пакетов плана. DB2 проведет поиск пакетов в том порядке, в котором они перечислены в списке пакетов.
- *На удаленной подсистеме* порядок такой:
  1. Все пакеты, уже размещенные в плане на данной подсистеме, пока план выполняется.
  2. Все не размещенные пакеты, названные явным образом в списке пакетов плана, и все собрания, полностью включенные в список пакетов плана, для которых подсистемы отвечают значению CURRENT SERVER. DB2 проведет поиск пакетов в том порядке, в котором они перечислены в списке пакетов.

Если вы используете BIND PLAN с опцией DEFER(PREPARE), DB2 не проведет поиска по всем собраниям в списке пакетов. Дополнительную информацию смотрите в разделе “Используйте опции связывания, повышающие производительность” на стр. 421.

**Если вы зададите специальный регистр CURRENT PACKAGESET, DB2** пропустит проверку программ, являющихся частью плана, и использует в качестве собрания значение CURRENT PACKAGESET. Например, если CURRENT PACKAGESET содержит COL5, DB2 использует при поиске COL5.PROG1.*отметка–времени*.

Задавая явным образом нужное собрание с помощью регистра CURRENT PACKAGESET, можно предотвратить затраты на поиск по списку пакетов; это может оказаться существенно, если есть много специфицирующих записей.

**Когда порядок поиска не важен:** Во многих случаях порядок поиска DB2 не важен для вас и не влияет на производительность. Для программы, исполняемой только на локальной системе DB2, можно назвать каждый пакет своим именем и включить все пакеты в одно собрание. Список пакетов в подкоманде BIND PLAN может выглядеть так:

`PKLIST (собрание.*)`

Можно добавлять пакеты в собрание и после связывания плана. DB2 позволяет вам связывать пакеты с одинаковыми именами в одно собрание, только если у них отличаются ID версий.

Если ваша программа использует доступ по протоколу DRDA, надо связать некоторые пакеты на удаленных подсистемах. Используйте на всех подсистемах одно и то же имя собрания, и обозначьте список пакетов так:

`PKLIST (*.собрание.*)`

Если указать звездочку вместо части имени в списке пакетов, DB2 будет проверять авторизацию для пакета, выбранного для данного имени во время запуска. Чтобы избежать проверки во время запуска в вышеприведенном примере, можно предоставить права EXECUTE на все собрание владельцу плана до того, как связать план .

### Идентификация версии пакета

Но иногда требуется, чтобы плану были доступны несколько пакетов с совпадающими именами. Это становится возможным благодаря опции препроцессора VERSION. При использовании VERSION ваша программа помечается особой версией пакета. Если вы связываете план при помощи PKLIST (COLLECT.\*), будет возможно следующее:

номер шага	для версии 1	для версии 2
1	прекомпилировать программу 1 с помощью VERSION(1).	прекомпилировать программу 2 с помощью VERSION(2).
2	Связать модуль DBRM с именем собрания COLLECT и именем выбранного вами пакета (например, PACKA).	Связать модуль DBRM с именем собрания COLLECT и именем пакета PACKA.
3	Скомпоновать программу 1.	Скомпоновать программу 2.
4	Запустить скомпонованную программу; она будет использовать программу 1 и PACKA, VERSION 1.	Запустить скомпонованную программу; она будет использовать программу 2 и PACKA, VERSION 2.

Можно проделать это со многими версиями программы, устранив необходимость повторного связывания плана программы. Не придется также переименовывать план и изменять подкоманды RUN, использующие его.

### **Задание уровня программы**

Чтобы не позволить системе DB2 самостоятельно строить маркер согласованности, используйте опцию препроцессора LEVEL (aaaa). При порождении маркера согласованности DB2 будет использовать выбранное вами значение aaaa. Хотя этот метод не рекомендуется для всеобщего пользования, и его не поддерживают ни DSNH CLIST, ни панели DB2 Program Preparation, он позволяет сделать следующее:

1. Изменить текст исходной программы (кроме операторов SQL) на выходе препроцессора для связанной программы.
2. Скомпилировать и скомпоновать измененную программу.
3. Запустить скомпонованную программу без повторного связывания плана или пакета.

### **Использование опций BIND и REBIND для пакетов и планов**

В этом разделе обсуждаются несколько более сложных опций связывания и повторного связывания. Синтаксис и полные описания всех опций связывания и повторного связывания смотрите в Главе 2 руководства *DB2 Command Reference*.

### **Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES**

Опции BIND и REBIND DYNAMICRULES определяют, какие значения будут действовать во время запуска для следующих динамических атрибутов SQL:

- ID авторизации, используемый при проверки авторизации
- Спецификатор, используемый для неспецифицированных объектов
- Источник опций прикладного программирования, используемый системой DB2 для анализа и семантической проверки динамических операторов SQL
- Могут ли динамические операторы SQL содержать операторы GRANT, REVOKE, ALTER, CREATE, DROP и RENAME

Помимо значения DYNAMICRULES, поведением динамических операторов SQL при запуске программы управляет среда запуска пакета. Возможны две среды запуска:

- Пакет запускается как часть самостоятельной программы.
- Пакет запускается как пакет хранимой процедуры или пользовательской функции, или из-под хранимой процедуры или пользовательской функции.

Пакет считается запускаемым из-под хранимой процедуры или пользовательской функции, если ассоциированная с ним программа удовлетворяет одному из следующих условий:

- Программа вызывается хранимой процедурой или пользовательской функцией.
- Программа входит в последовательность вложенных вызовов, начинающуюся с хранимой процедуры или пользовательской функции.

Комбинация значения DYNAMICRULES и среды запуска определяет значения динамических атрибутов SQL. Этот набор значений атрибутов называется *стратегией динамических операторов SQL*. Таких стратегий четыре:

- Стратегия запуска

- Стратегия связывания
- Стратегия определения
- Стратегия вызова

В Табл. 47 приводятся комбинации значения DYNAMICRULES и среды запуска, при которых достигаются все эти динамические стратегии SQL. В Табл. 48 приводятся значения динамических атрибутов SQL для каждого типа динамической стратегии SQL.

*Таблица 47. Каким образом DYNAMICRULES и среда запуска определяют стратегию динамических операторов SQL*

значение DYNAMICRULES	Стратегия динамических операторов SQL	
	среда самостоятельной программы	среда пользовательской функции или хранимой процедуры
BIND	Стратегия связывания	Стратегия связывания
RUN	Стратегия запуска	Стратегия запуска
DEFINEBIND	Стратегия связывания	Стратегия определения
DEFINERUN	Стратегия запуска	Стратегия определения
INVOKEBIND	Стратегия связывания	Стратегия вызова
INVOKERUN	Стратегия запуска	Стратегия вызова

Примечание к Табл. 47:

Значения BIND и RUN можно задавать для пакетов и планов. Остальные значения можно задавать только для пакетов.

*Таблица 48. Определения стратегий динамических операторов SQL*

динамический атрибут SQL	задание динамического атрибута SQL			
	Стратегия связывания	Стратегия запуска	Стратегия определения	Стратегия вызова
ID авторизации	владелец плана или пакета	текущий SQLID	владелец пользовательской функции или хранимой процедуры	ID авторизации для вызывающей программы <sup>1</sup>
Спецификатор для неспецифицированных объектов по умолчанию	значение Bind OWNER или QUALIFIER	текущий SQLID	владелец пользовательской функции или хранимой процедуры	ID авторизации для вызывающей программы
CURRENT SQLID <sup>2</sup>	не применимо	применимо	не применимо	не применимо
Источник опций прикладного программирования	Определяется параметром DSNHDECP DYNRULS <sup>3</sup>	панель установки DSNTIPF	Определяется параметром DSNHDECP DYNRULS <sup>3</sup>	Определяется параметром DSNHDECP DYNRULS <sup>3</sup>
может выполнять GRANT, REVOKE, CREATE, ALTER, DROP, RENAME?	Нет	Да	Нет	Нет

Примечание к Табл. 48:

1. Еслизывающая программа имеет первичный ID авторизации процесса или значение CURRENT SQLID, для вторичных ID авторизации также будет проверяться, нужны ли они для требуемой авторизации. В противном случае для требуемой авторизации будет проверяться только один ID, а именно ID вызывающей программы.
2. DB2 использует значение CURRENT SQLID как ID авторизации для динамических операторов SQL только для планов и пакетов, имеющих стратегию запуска. Для остальных динамических стратегий SQL DB2 использует ID авторизации, связанный с каждой динамической стратегией SQL, как показано в этой таблице.

Начальное значение, присваиваемое каждому CURRENT SQLID, не зависит от динамической стратегии SQL. Для самостоятельных программ начальное значение CURRENT SQLID равно первичному ID авторизации. Информацию о начальных значениях CURRENT SQLID для пользовательских функций и хранимых процедур смотрите в Табл. 33 на стр. 304 и Табл. 56 на стр. 600.

Изменить значение CURRENT SQLID можно для пакетов с любой динамической стратегией SQL, выполнив оператор SET CURRENT SQLID, но DB2 будет использовать значение CURRENT SQLID только для планов и пакетов со стратегией запуска.
3. Значение параметра DSNHDECP DYNRULS, которое задается в поле USE FOR DYNAMICRULES на панели установки DSNTIPF, определяет, будет ли система DB2 использовать значения по умолчанию опций препроцессора и опций прикладного программирования для динамических операторов SQL. Дополнительную информацию смотрите в разделе “Опции препроцессора” на стр. 441.

### **Определение оптимального размера кэша авторизации**

Когда DB2 определяет, есть ли у вас права EXECUTE на план, собрание пакетов, хранимую процедуру и пользовательскую функцию, DB2 может кэшировать ваш ID авторизации. Тогда при запуске плана, пакета, хранимой процедуры и пользовательской функции DB2 может быстрее проводить проверку ваших прав.

**Определение размера кэша авторизации для планов:** Опция CACHESIZE (необязательная) позволяет задать размер кэша, выделяемого плану. DB2 будет использовать этот кэш для размещения ID авторизации тех пользователей, которые запускают план. DB2 будет использовать значение CACHESIZE для определения объема памяти, выделяемого на кэш авторизации. Память берется из пула EDM. По умолчанию значение CACHESIZE равно 1024 или размеру, заданному при установке.

Разумный размер кэша зависит от числа индивидуальных ID авторизации, активно использующих план. Необходимый заголовок занимает 32 байта, и на каждый ID авторизации расходуется 8 байтов памяти. Минимальный размер кэша равен 256 байтам (то есть 28 записей плюс заголовок), а максимальный – 4096 байтам (то есть 508 записей плюс заголовок). Размер следует задавать кратным 256 байтам; в противном случае заданное значение будет округлено в большую сторону до кратного 256.

Если план запускается не часто, или права для запуска плана принадлежат группе PUBLIC, может оказаться желательным отключить кэширование плана,

тем самым избежав излишнего расхода памяти системой DB2. Для этого задайте опцию CACHESIZE равной 0.

Все планы, запускаемые часто, разумно попытаться настроить при помощи опции CACHESIZE. Кроме того, если план запускают одновременно много пользователей, может оказаться желательно использовать больший CACHESIZE.

**Определение размера кэша авторизации для пакетов:** DB2 поддерживает один кэш авторизации пакетов на всю подсистему DB2. При установке DB2 размер кэша авторизации пакетов задается в поле PACKAGE AUTH CACHE на панели установки DB2 DSNTIPP. Кэш авторизации 32 Кбайта достаточно для размещения информации приблизительно о 375 сбояниях пакетов.

Дополнительную информацию о задании размера кэша авторизации пакетовсмотрите в руководстве *DB2 Installation Guide*.

**Определение размера кэша авторизации для хранимых процедур и пользовательских функций:** DB2 поддерживает один кэш авторизации подпрограмм на всю подсистему DB2. Кэш авторизации подпрограмм хранит список ID авторизации, имеющих права EXECUTE на пользовательские функции и хранимые процедуры. При установке DB2 размер кэша авторизации подпрограмм задается в поле ROUTINE AUTH CACHE на панели установки DB2 DSNTIPP. Кэша авторизации размером 32 Кбайта достаточно для информации об авторизации около 380 хранимых процедур и пользовательских функций.

Дополнительную информацию о задании размера кэша авторизации подпрограммсмотрите в руководстве *DB2 Installation Guide*.

## Задание правил SQL

Опция SQLRULES задает не только правила, по которым будет исполняться оператор CONNECT второго типа, но и начальное значение специального регистра CURRENT RULES, когда сервером прикладных программ является локальная система DB2. Когда сервер прикладных программ отличен от локальной DB2, начальное значение CURRENT RULES равно DB2. После связывания плана изменить значение CURRENT RULES можно в прикладной программе при помощи оператора SET CURRENT RULES.

CURRENT RULES определяет правила SQL (DB2 или стандарт SQL), применяемые к стратегии SQL при запуске. Например, значение в CURRENT RULES влияет на стратегию определения проверочных ограничений при применении оператора ALTER TABLE к непустой таблице:

- **Если CURRENT RULES имеет значение STD** и ни одна из существующих строк в таблице не нарушает проверочное ограничение, DB2 внесет ограничение в определение таблицы. В противном случае возникнет ошибка и DB2 не станет вносить проверочное ограничение в определение таблицы.

Если таблица содержит данные и уже находится в состоянии отложенной проверки, оператор ALTER TABLE не выполняется.

- **Если CURRENT RULES имеет значение DB2**, DB2 включит ограничение в определение таблицы, отложит применение проверочных ограничений и

переведет табличное пространство или раздел в состояние отложенной проверки.

При помощи оператора SET CURRENT RULES можно управлять действиями, выполняемыми оператором ALTER TABLE. В следующем примере считается, что значение CURRENT RULES вначале равно STD. Приведенные операторы SQL изменят правила с SQL на DB2, добавят проверочное ограничение, откладывая проверку этого ограничения, и переведут таблицу в состояние отложенной проверки, после чего восстановят правила STD.

```
EXEC SQL
SET CURRENT RULES = 'DB2';
EXEC SQL
ALTER TABLE DSN8610.EMP
ADD CONSTRAINT C1 CHECK (BONUS <= 1000.0);
EXEC SQL
SET CURRENT RULES = 'STD';
```

добавят проверочное ограничение непосредственно к непустой таблице, отложив проверку ограничения, переведут таблицу в состояние отложенной проверки и восстановят стандартную опцию. Информацию о проверочных ограниченияхсмотрите в разделе “Изменение таблиц с проверочными ограничениями” на стр. 61.

Можно также использовать CURRENT RULES в операторе присваивания переменных хоста, например:

```
SET :XRULE = CURRENT RULES;
```

и как аргумент критерия поиска, например:

```
SELECT * FROM SAMPTBL WHERE COL1 = CURRENT RULES;
```

## Использование пакетов с динамическим выбором плана

### Среда CICS

Можно совместно использовать пакеты и динамический выбор плана, но при динамическом переключении планов должны быть выполнены следующие условия:

- Все специальные регистры, включая CURRENT PACKAGESET, должны содержать свои начальные значения.
- Значение в специальном регистре CURRENT DEGREE не должны были измениться в течение текущей транзакции.

Преимущество совместного использования динамического выбора плана и пакетов в том, что отдельные программы можно по одной преобразовывать в прикладную программу, содержащую много программ и планов, а потом использовать комбинацию планов и пакетов. Так уменьшается число планов, приходящихся на одну прикладную программу, а чем меньше число планов, тем меньше усилий тратится на поддержание динамической обработки плана.

При помощи следующих примеров программ и модулей DBRM:

имя программы	имя модуля DBRM
MAIN	MAIN
PROGA	PLANA
PROGB	PKGB
PROGC	PLANC

вы можете создавать пакеты и планы с помощью следующих операторов связывания:

```
BIND PACKAGE(PKGB) MEMBER(PKGB)
BIND PLAN(MAIN) MEMBER(MAIN,PLANA) PKLIST(*.PKGB.*)
BIND PLAN(PLANC) MEMBER(PLANC)
```

Следующий сценарий иллюстрирует связывание потоков для задания, запустившего программу MAIN:

#### последовательность операторов SQL события

1. EXEC CICS START TRANSID(MAIN) TRANSID(MAIN) выполняет программу MAIN.
2. EXEC SQL SELECT... Программа MAIN выдает оператор SQL SELECT. По умолчанию динамический обработчик плана выбирает план MAIN.
3. EXEC CICS LINK PROGRAM(PROGA)
4. EXEC SQL SELECT... DB2 не вызывает динамический обработчик плана по умолчанию, поскольку программа не выполнила точку синхронизации. Используется план MAIN.

## Среда CICS (продолжение)

### последовательность операторов SQL события

5. EXEC CICS LINK PROGRAM(PROGB)	
6. EXEC SQL SELECT...	DB2 не вызывает динамический обработчик плана по умолчанию, поскольку программа не выполнила точку синхронизации. Планом является MAIN, и программа использует пакет PKGB.
7. EXEC CICS SYNCPOINT	DB2 вызывает динамический обработчик плана, когда выполняется следующий оператор SQL.
8. EXEC CICS LINK PROGRAM(PROGC)	
9. EXEC SQL SELECT...	DB2 вызывает динамический обработчик плана по умолчанию и выбирает PLANC.
10. EXEC SQL SET CURRENT SQLID = 'ABC'	
11. EXEC CICS SYNCPOINT	DB2 не вызывает динамический обработчик плана, когда выполняется следующий оператор SQL, поскольку предыдущий оператор изменил специальный регистр CURRENT SQLID.
12. EXEC CICS RETURN	Управление возвращается программе PROGB.
13. EXEC SQL SELECT...	Возникает состояние SQLCODE -815, поскольку текущим планом является PLANC, а программой является PROGB.

## Шаг 3: Компиляция (или асSEMBЛИрование) и компоновка прикладной программы

Следующий шаг в процессе подготовке программы – скомпилировать и скомпоновать программу. Как и на шаге прекомпиляции, можно выбрать один из нескольких методов:

- панели DB2I
- Командная процедура DSNH (задание TSO на языке CLIST)
- процедуры на языке JCL, поставляемые с системой DB2.

Целью компоновки является создание исполняемого загрузочного модуля. Чтобы ваша программа могла взаимодействовать с подсистемой DB2, необходимо использовать компоновочную процедуру, строящую загрузочный модуль, который должен:

### среда TSO и пакетный файл

Включать модуль языкового интерфейса утилиты подключения TSO DB2 (DSNELI) или модуль языкового интерфейса утилиты подключения вызова DB2 (DSNALI).

При создании объектного файла программы, запускаемой в среде MVS/ESA с 31–битовой адресацией:

- Использовать для создания объектного файла программы Assembler H, версии 2.
- Использовать для компоновки программы компоновщик MVS/ESA. Задать AMODE=31 и RMODE=ANY как опции для компоновщика.

Дополнительные подробности смотрите в соответствующих изданиях по MVS/ESA.

### IMS

Включать модуль языкового интерфейса (DFSLI000) системы DB2 IMS (Версия 1 Выпуск 3 или более поздние). Кроме того, библиотека IMS RESLIB должна предшествовать библиотеке SDSNLOAD в списке связей или конкатенациях JOBLIB и STEPLIB.

### Среда CICS

Включать модуль языкового интерфейса DB2 CICS (DSNCLI).

Можно связать DSNCLI с вашей программой в режиме 24–битной или 31–битной адресации (AMODE=31). Если ваша прикладная программа запускается в режиме 31–битной адресации, следует скомпоновать с вашей программой заглушку для DSNCLI, используя атрибуты AMODE=31 и RMODE=ANY, чтобы ваша программа могла запускаться выше границы 16 Мбайт. Дополнительную информацию о компиляции и компоновке прикладных программ CICS смотрите в соответствующем справочнике по CICS.

Понадобится также модуль интерфейса CICS EXEC, соответствующий языку программирования. Среда CICS требует, чтобы этот модуль был первой контрольной секцией (CSECT) в итоговом загрузочном модуле.

Размер исполняемого загрузочного модуля, создаваемого на этапе компоновки, может меняться в зависимости от значений, вставленных препрессором DB2 в исходный текст программы.

Дополнительную информацию о компиляции и компоновке смотрите в разделе “Использование процедур JCL для подготовки прикладных программ” на стр. 469.

Дополнительную информацию об атрибутах компоновки смотрите в соответствующих справочниках MVS. Подробности о DSNH смотрите в Главе 2 руководства *DB2 Command Reference*.

---

## Шаг 4: Запуск прикладной программы

После завершения всех предыдущих шагов вашу программу можно запустить. К этому времени DB2 проверит, что информация в плане программы и ассоциированных пакетах отвечает соответствующей информации в системном каталоге DB2. Если произойдут какие-либо деструктивные изменения, например, будут выполнены операторы DROP или REVOKE (затрагивающие либо структуры данных, к которым обращается ваша программа, либо права связывателя на доступ к этим структурам данных), DB2 автоматически проведет необходимое повторное связывание пакетов или плана.

### Командный процессор DSN

Командный процессор DSN – это командный процессор в среде TSO, который запускается в основном режиме в среде TSO или в пакете TSO, инициированном подсистемой JES. Для доступа к DB2 он использует утилиту подключения TSO. Командный процессор DSN обеспечивает альтернативный метод запуска программ, обращающихся к DB2 в среде TSO.

Командный процессор DSN можно использовать неявно, по ходу программы, для таких целей, как:

- Использование генератора объявлений (DCLGEN)
- Запуск подкоманд BIND, REBIND и FREE для планов и пакетов DB2 для вашей программы
- Использование SPUFI (процессор SQL, использующий файловый ввод) для тестирования некоторых функций SQL в программе

Командный процессор DSN запускается программой монитора терминала TSO (TMP). Поскольку программа TMP может запускаться либо в основном режиме, либо в фоновом, программы DSN запускаются интерактивно или как пакетные задания.

Командный процессор DSN может обеспечивать следующие службы для запущенной из-под него программы:

- Автоматическое соединение с DB2
- Поддержка клавиши Attention ("Внимание")
- Трансляция кодов возврата в сообщения об ошибках

### Ограничения командного процессора DSN

При использовании служб DSN ваша программа работает под управлением DSN. Поскольку для запуска DSN среда TSO выполняет макрокоманду ATTACH, и DSN выполняет макрокоманду ATTACH для запуска части самого себя, ваша программа получает управление на два уровня заданий ниже, чем у TSO.

Поскольку ваша программа в отношении обслуживания связей с DB2 полагается на DSN:

- Если DB2 не работает, ваша программа не запустится.
- Если DB2 завершит работу, ваша программа также завершит работу.
- Программа может использовать только один план.

Если эти ограничения чрезмерны, рассмотрите возможность использования вашей программой утилиты подключения вызовов или утилиты Recoverable Resource Manager Services. Дополнительную информацию об этих утилитах подключения смотрите в разделах “Глава 7–7. Программирование для утилиты подключения по вызову (CAF)” на стр. 769 и “Глава 7–8. Программирование для утилиты подключения служб управления восстановимыми ресурсами (RRSAF)” на стр. 807.

### **Обработка кода возврата DSN**

В конце сеанса DSN регистр 15 содержит наибольшее значение, помещенное в него всеми подкомандами DSN в сеансе, и всеми программами, запущенными подкомандой RUN. Ваша среда запуска может отформатировать это значение как код возврата. Но это значение порождено *не* в DSN.

## **Запуск программы в среде TSO в основном режиме**

*При помощи панели DB2I RUN можно запустить программу в основном режиме TSO.* Другой способ – выдача команды DSN и затем подкоманды RUN системы DSN. Перед запуском программы убедитесь, что выделено место для всех наборов данных, нужных вашей программе.

В следующем примере показывается, как запустить программу в основном режиме TSO. Программа здесь называется SAMPPGM, а ID системы – *ssid*:

```
приглашение TSO:      READY
введите:          DSN SYSTEM(ssid)
приглашение DSN:      DSN
введите:          RUN PROGRAM(SAMPPGM) -
                  PLAN (SAMPLAN) -
                  LIB (SAMPPROJ.SAMPLIB) -
                  PARMS ('/D01 D02 D03')
:
(Zдесь программа запустится и, возможно, выдаст приглашение для ввода)
приглашение DSN:      DSN
введите:          END
приглашение TSO:      READY
```

Эта последовательность сработает и в опции 6 ISPF. Можно превратить эту последовательность в пакет на языке CLIST. DB2 не поддерживает доступ к нескольким подсистемам DB2 из одного адресного пространства.

Параметр PARMS подкоманды RUN позволяет передавать параметры процессору времени выполнения и вашей прикладной программе:

```
PARMS ('/D01, D02, D03')
```

Дробная черта (/) указывает, куда нужно передать параметры.  
Дополнительные подробности посмотрите в изданиях о вашем языке хоста.

## Запуск пакетной программы DB2 в среде TSO

Большинство прикладных программ, написанных для пакетной среды, запускаются под Terminal Monitor Program – программой монитора терминала (TMP) в фоновом режиме среды TSO. В разделе рис. 107 приводятся операторы JCL, необходимые для запуска такого задания. Далее все эти операторы будут объяснены.

```
//имя-задания JOB USER=MY DB2ID
//GO EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=префикс.SDSNEXIT,DISP=SHR
//          DD DSN=префикс.SDSNLOAD,DISP=SHR
::
//SYSTSPPRT DD SYSOUT=A
//SYSTSIN DD *
  DSN SYSTEM (ssid)
  RUN PROG (SAMPPGM) -
    PLAN (SAMPLAN) -
    LIB (SAMPPROJ.SAMPLIB) -
    PARM ('/D01 D02 D03')
  END
/*
```

Рисунок 107. Задание на языке JCL для запуска прикладной программы DB2 под программой монитора терминала TSO

- Оператор JOB начинает задание. Опция USER задает ID авторизации пользователя DB2.
- Оператор EXEC вызывает программу монитора терминала TSO (TMP).
- Оператор STEPLIB задает библиотеку, в которой находятся загружаемый модуль командного процессора DSN и модуль значений прикладного программирования по умолчанию, DSNHDECP. Он может также содержать ссылки на библиотеки, где находятся пользовательские программы, обработчики и настроенный пользователем модуль DSNHDECP. Настроенный пользователем модуль DSNHDECP создается при установке. Если не задать библиотеку, содержащую настроенный пользователем DSNHDECP, DB2 будет использовать DSNHDECP по умолчанию.
- Идущие далее операторы DD определяют дополнительные файлы, необходимые для вашей программы.
- Команда DSN соединяет прикладную программу с конкретной подсистемой DB2.
- Подкоманда RUN задает имя запускаемой прикладной программы.
- Ключевое слово PLAN задает имя плана.
- Ключевое слово LIB задает библиотеку, к которой должна иметь доступ прикладная программа.
- Ключевое слово PARM передает параметры процессору запуска и прикладной программе.
- END завершает командный процессор DSN.

### Примечания по использованию:

- Желательно, чтобы шаги задания DSN были короткими.

- Не рекомендуется использовать DSN для вызова командного процессора EXEC, чтобы запустить задания на языке CLIST, содержащие операторы ISPEXEC; результаты будут непредсказуемыми.
- Если ваша программа завершится аварийно или выдаст ненулевой код возврата, DSN завершит работу.
- Для соединения с членом группы совместного использования данных можно вместо конкретного *ssid* использовать групповое имя подключения. Дополнительную информациюсмотрите в руководстве *DB2 Data Sharing: Planning and Administration*.

Дополнительную информацию об использовании TSO TMP в пакетном режиме смотрите в руководстве *OS/390 TSO/E User's Guide*.

## Вызов прикладных программ в командной процедуре (CLIST)

Альтернативой для ранее описанных вызовов прикладной программы в основном и в пакетном режимах является запуск прикладной программы TSO или пакетной программы при помощи командной процедуры (CLIST).

Следующая процедура CLIST вызывает прикладную программу DB2 под именем MYPROG. Вместо *ssid* следует подставить имя подсистемы DB2 или групповое имя подключения.

```
PROC 0                                /* ВЫЗОВ DSN ИЗ CLIST          */
  DSN SYSTEM(ssid)    /* ВЫЗОВ ПОДСИСТЕМЫ DB2 ssid      */
  IF &LASTCC = 0 THEN      /* ПРОВЕРКА УСПЕШНОСТИ КОМАНДЫ DSN */
    DO                      /* ЕСЛИ ДА, ВЫПОЛНЯЕМ ПОДКОМАНДУ DSN RUN */
      DATA                  /* ИНАЧЕ ПРОПУСКАЕМ СЛЕДУЮЩЕЕ: */
      RUN PROGRAM(MYPROG)
    END
    ENDDATA                 /* RUN И END ДЛЯ DSN           */
  END
  EXIT
```

### Система IMS

#### Чтобы запустить программу, управляемую сообщениями

Вначале убедитесь, что сможете ответить на интерактивные требования программы о вводе данных, и что сможете оценить ожидаемые результаты. Затем введите код транзакции, ассоциированной с программой. У пользователей кода транзакции должны быть права на запуск программы.

#### Чтобы запустить программу, не управляемую сообщениями

Введите операторы управления заданием JCL, необходимые для запуска программы.

## Среда CICS

### Чтобы запустить программу

Во-первых, убедитесь, что соответствующие записи в контрольных областях RCT, SNT, и RACF\* предоставляют права запуска для вашей программы. За эти функции отвечает администратор системы; дополнительную информацию смотрите в разделе Раздел 3 (Том 1) *DB2 Administration Guide*.

Кроме того, убедитесь, что в среде CICS определен код транзакции, назначенный вашей программе, и сама программа.

### Сделайте новую копию программы

Ведите команду NEWCOPY, если среда CICS не была повторно инициализирована после последнего связывания и компиляции программы.

## Использование процедур JCL для подготовки прикладных программ

Для подготовки прикладной программы к запуску есть несколько методов. Можно:

- Использовать интерактивные панели DB2 (DB2I), которые помогут вам провести процесс подготовки шаг за шагом. Смотрите раздел “Использование ISPF и DB2 Interactive (DB2I)” на стр. 475.
- Запустить фоновое задание, используя процедуру на языке JCL. Ее для вас могут создать панели подготовки программ.
- Запустить задание DSNH CLIST в среде TSO в основном или фоновом режиме.
- Использовать приглашения TSO и командный процессор DSN.
- Использовать процедуры JCL, внесенные в вашу библиотеку SYS1.PROCLIB (или эквивалентную) во время установки DB2.

В этом разделе описывается, как подготовить программу при помощи процедур JCL. Информацию об использовании заданий DSNH CLIST, командного процессора TSO DSN и процедур JCL, внесенных в вашу библиотеку SYS1.PROCLIB, смотрите в Главе 2 руководства *DB2 Command Reference*. Общий обзор процесса подготовки программы DB2, выполняемого заданиями DSNH CLIST, смотрите на рис. 106 на стр. 437.

## Доступные процедуры JCL

Для прекомпиляции и подготовки прикладной программы можно использовать процедуру, поставляемую с системой DB2. В DB2 включены отдельные процедуры для каждого из поддерживаемых языков, с соответствующими значениями по умолчанию для запуска прекомпилиатора DB2 и компилятора языка хоста или ассемблера. Процедуры находятся в компоненте *prefix.SDSNSAMP* библиотеки DSNTIJMV, которая устанавливает процедуры.

Таблица 49. Процедуры для прекомпиляции программ

язык	процедура	вызов, включенный в...
ассемблер высокого уровня	DSNHASM	DSNTEJ2A
C	DSNHCO	DSNTEJ2D
C <sup>++</sup>	DSNHCPP DSNHCPP2 <sup>22</sup>	DSNTEJ2E не применимо
OS/VS COBOL	DSNHCOB	DSNTEJ2C
COBOL/370	DSNHICOB	DSNTEJ2C1
COBOL для MVS & VM	DSNHICOB DSNHICB2 <sup>22</sup>	DSNTEJ2C1 не применимо
VS COBOL II	DSNHCOB2	DSNTEJ2C1
FORTRAN	DSNHFOR	DSNTEJ2F
PL/I	DSNHPLI	DSNTEJ2P

Примечания к Табл. 49

1. Необходимо настроить эти программы для вызова процедур, перечисленных в этой таблице. О том, как это сделать, написано в Разделе 2 руководства *DB2 Installation Guide*.
2. Эта процедура демонстрирует, как можно подготовить объектно-ориентированную программу, состоящую из двух наборов данных или компонентов, каждый из которых содержит SQL.

Если вы используете макропроцессор PL/I, для передачи опций компилятору PL/I нельзя использовать в исходном тексте программы оператор \*PROCESS языка PL/I. Можно задать необходимые опции как подопцию PARM.PLI=параметр оператора EXEC в процедуре DSNHPLI.

## Включение текстов программ из наборов данных SYSLIB

Чтобы включить нужный текст программы интерфейса при запуске процедур JCL, используйте в своей процедуре JCL один из наборов операторов, приведенных ниже, или, если вы используете вызов утилиты подключения, следуйте указаниям в разделе “Обращение к языковому интерфейсу CAF” на стр. 776.

Среда TSO, TSO batch и CAF

```
//LKED.SYSIN DD *
INCLUDE SYSLIB(компонент)
/*
```

Компонент во всех случаях DSNELI, кроме языка FORTRAN, для которого компонент называется DSNHFT.

## IMS

```
//LKED.SYSIN DD *
INCLUDE SYSLIB(DFSLI000)
ENTRY (спецификация)
/*
```

DFSLI000 – это модуль для подключения пакета DL/I.

ENTRY спецификация зависит от языка хоста. Это может быть:

DLITCBL – для программ на языке COBOL  
PLICALLA – для программ на языке PL/I  
Имя вашей программы – для программ на ассемблере.

## Среда CICS

```
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNCLI)
/*
```

Дополнительную информацию о требуемых модулях CICS смотрите в разделе “Шаг 3: Компиляция (или ассемблирование) и компоновка прикладной программы” на стр. 463.

## Динамический запуск прекомпилятора

Прекомпилятор можно вызвать из ассемблерной программы с помощью макрокоманды ATTACH, CALL, LINK или XCTL. Ниже приводится информация, дополняющая описание этих макроинструкций, данное в руководстве *OS/390 MVS Programming: Assembler Services Reference*.

Чтобы вызвать прекомпилятор, задайте DSNHPC в качестве имени входной точки. Прекомпилятору можно передать три опции–адреса; в следующих разделах описаны их форматы. Опции – это адреса:

- Списка опций прекомпилятора
- Списка альтернативных ddnames для наборов данных, которые использует прекомпилятор
- Номера страницы, используемой как первая страница листинга компилятора, выводимого в поток SYSPRINT.

## Формат списка опций прекомпилятора

Список опций должен начинаться на границе между парами байтов. Первые 2 байта задают в двоичном виде число байтов в списке (исключая сам счетчик). Остальная часть списка может содержать в коде EBCDIC ключевые слова опций прекомпилятора, разделенных одним или несколькими пробелами, запятыми или и тем, и другим.

## **Формат списка имен DDNAME**

Список имен ddname должен начинаться на границе между парами байтов. Первые 2 байта задают в двоичном виде число байтов в списке (исключая сам счетчик). Каждая запись в списке представляет собой 8-байтное поле, выравненное по левому краю и при необходимости дополненное пробелами.

В следующей таблице приводится последовательность записей:

Таблица 50. Записи в списке имен DDNAME

запись	стандартное имя ddname	применение
1	не применимо	
2	не применимо	
3	не применимо	
4	SYSLIB	входная библиотека
5	SYSIN	исходный текст
6	SYSPRINT	вывод диагностики
7	не применимо	
8	SYSUT1	рабочие данные
SYSUT2	рабочие данные	
SYSUT3	рабочие данные	
11	не применимо	
12	SYSTEM	вывод диагностики
13	не применимо	
14	SYSCIN	выходной обработанный текст программы
15	не применимо	
16	DBRMLIB	модуль DBRM на выходе

## **формат номера страницы**

6-байтное поле, начинающееся на границе между парами байтов, которое содержит номер страницы. Первые два байта должны содержать в двоичном формате число 4 (это длина остальной части поля). Последние 4 байта содержат номер страницы в символах или в зонном десятичном формате.

Прекомпилятор прибавляет 1 к последнему номеру страницы, использованному в выводе прекомпилиатора, и помещает это значение в поле номера страницы, прежде чем вернуть управление вызвавшей его программе. Таким образом, если вы снова вызовите прекомпилиатор, нумерация страниц будет продолжена.

## **Альтернативный метод подготовки программ в среде CICS**

## Среда CICS

Вместо использования панелей DB2 Program Preparation для подготовки вашей программы в среде CICS можно настроить для этого процедуры JCL, поставляемые вместе с CICS. Настройка процедуры CICS будет состоять в добавлении нескольких шагов и редактировании нескольких операторов DD. Внесите необходимые изменения и выполните следующие действия:

- Обработайте программу препроцессором DB2.
- Свяжите план программы. Это можно сделать в любой момент после препроцессии программы. Можно связать программу либо в диалоговом режиме при помощи панелей DB2I, или как шаг в пакете в том или ином задании MVS.
- Вставьте оператор DD в шаг компоновки, чтобы получить доступ к библиотеке загрузки DB2.
- Убедитесь, что среди управляющих операторов компоновщика есть оператор INCLUDE для модуля языкового интерфейса DB2.

На следующем примере показаны необходимые изменения. В этом примере предполагается, что программа написана на языке VS COBOL II или COBOL/370. Для других языков программирования измените имя процедуры CICS и опции препроцессора DB2.

```
//TESTC01 JOB
/*
//***** ПРЕКОМПИЛЯЦИЯ DB2 ПРОГРАММЫ НА ЯЗЫКЕ COBOL
//*****
(1) //PC      EXEC PGM=DSNHPC,
(1) //          PARM='HOST(COB2),XREF,SOURCE,FLAG(I),APOST'
(1) //STEPLIB  DD  DISP=SHR,DSN=prefix.SDSNEXIT
(1) //          DD  DISP=SHR,DSN=prefix.SDSNLOAD
(1) //DBRMLIB  DD  DISP=OLD,DSN=USER.DBRLIB.DATA(TESTC01)
(1) //SYSCIN   DD  DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
(1) //          SPACE=(800,(500,500))
(1) //SYSLIB   DD  DISP=SHR,DSN=USER.SRCLIB.DATA
(1) //SYSPRINT DD  SYSOUT=*
(1) //SYSTEMR  DD  SYSOUT=*
(1) //SYSUDUMP DD  SYSOUT=*
(1) //SYSUT1   DD  SPACE=(800,(500,500),,ROUND),UNIT=SYSDA
(1) //SYSUT2   DD  SPACE=(800,(500,500),,ROUND),UNIT=SYSDA
(1) //SYSIN    DD  DISP=SHR,DSN=USER.SRCLIB.DATA(TESTC01)
(1) /*
```

## Среда CICS (продолжение)

```
*****  
//*** СВЯЗЫВАЕМ ЭТУ ПРОГРАММУ.  
*****  
(2) //BIND EXEC PGM=IKJEFT01,  
(2) // COND=((4,LT,PC))  
(2) //STEPLIB DD DISP=SHR,DSN=prefix.SDSNEXIT  
(2) // DD DISP=SHR,DSN=prefix.SDSNLOAD  
(2) //DBRMLIB DD DISP=OLD,DSN=USER.DBRMLIB.DATA(TESTC01)  
(2) //SYSPRINT DD SYSOUT=**  
(2) //SYSTSPRT DD SYSOUT=**  
(2) //SYSUDUMP DD SYSOUT=**  
(2) //SYSTSIN DD *  
(2) DSN S(DSN)  
(2) BIND PLAN(TESTC01) MEMBER(TESTC01) ACTION(REP) RETAIN ISOLATION(CS)  
(2) END  
*****  
/* КОМПИЛИРУЕМ ПРОГРАММУ НА ЯЗЫКЕ COBOL  
*****  
(3) //CICS EXEC DFHEITVL  
(4) //TRN.SYSIN DD DSN=&DSNHOUT,DISP=(OLD,DELETE)  
(5) //LKED.SYSLMOD DD DSN=USER.RUNLIB.LOAD  
(6) //LKED.DB2LOAD DD DISP=SHR,DSN=prefix.SDSNLOAD  
//LKED.SYSIN DD *  
(7) INCLUDE DB2LOAD(DSNCLI)  
NAME TESTC01(R)  
*****
```

Процедура выполняет следующие шаги:

**Шаг 1.** Прекомпиляция программы.

**Шаг 2.** Связывание плана программы.

**Шаг 3.** Вызов процедуры CICS для трансляции, компиляции и компоновки программы на языке COBOL. Эта процедура имеет несколько опций, которые необходимо рассмотреть.

**Шаг 4.** Выход прекомпилиатора DB2 становится входом для транслятора с языка команд CICS.

**Шаг 5.** Учесть библиотеку загрузки прикладной программы в имени набора данных оператора SYSLMOD DD. Необходимо включить имя этой библиотеки загрузки в оператор DFHRPL DD запускающего задания JCL среды CICS.

**Шаг 6.** Назвать библиотеку загрузки DB2, содержащую модуль DSNCLI.

**Шаг 7.** Указать компоновщику включить модуль языкового интерфейса DB2 для CICS (DSNCLI). В этом примере порядок различных управляющих разделов (разделов CSECT) не важен, поскольку структура процедуры автоматически удовлетворяет любому порядку требований.

Дополнительную информацию о процедуре DFHEITVL, других процедурах CICS и требованиях CICS для прикладных программ смотрите в соответствующем справочнике по среде CICS.

Если вы готовите особо большую или сложную прикладную программу, можно использовать один из двух последних методов. Например, если ваша программа требует включения четырех ваших собственных компоновочных библиотек, вы не сможете подготовить эту программу при помощи DB2I, поскольку DB2I ограничивает число библиотек включения тремя (не считая языковую библиотеку IMS или CICS и библиотеки DB2). Таким образом, вам

потребуется другой метод подготовки. Программы, использующие вызов утилиты подключения, могут использовать любую из двух последних вышенназванных методик. *Обратите внимание на использование правильного языкового интерфейса.*

## Использование задания JCL для подготовки программ с объектно–ориентированными расширениями

Для подготовки программы на языке C++ или IBM COBOL for MVS & VM потребуется особое задание JCL, если:

- Программа состоит из нескольких наборов данных или компонентов.
- Несколько наборов данных или компонентов содержат операторы SQL.

Необходимо препомпилировать содержимое каждого набора данных или компонента отдельно, но препомпоновщик должен получить весь выход компилятора одновременно.

Процедуры JCL DSNHICB2 и DSNHCPP2, которые записаны в компоненте DSNTIJMV набора данных DSN610.SDSNSAMP, дают пример того, как можно это сделать. DSNHICB2 – процедура для языка COBOL, а DSNHCPP2 – процедура для языка C++.

---

## Использование ISPF и DB2 Interactive (DB2I)

При разработке программ, использующих TSO и ISPF, подготовку к работе можно выполнять при помощи панелей подготовки программ DB2. Эти панели позволяют готовить прикладные программы к запуску шаг за шагом. Это не единственный способ подготовки программ к запуску, но использование DB2I – простейший путь, который будет вести вас от одного шага к другому.

В этом разделе описаны опции, которые можно задать на панелях подготовки программ. При описании процесса в примерах подготовки программ предполагается, что используются программы на языке COBOL, запускаемые под TSO.

**Внимание:** Если для программы на C++ или IBM COBOL for MVS & VM выполнены оба следующих условия, для ее подготовки необходимо использовать процедуру JCL:

- Программа состоит из нескольких наборов данных или компонентов.
- Операторы SQL содержатся в нескольких наборах данных или компонентах.

Дополнительную информацию смотрите в разделе “Использование задания JCL для подготовки программ с объектно–ориентированными расширениями.”

## Справка по DB2I

Функция оперативной справки обеспечивает поиск информации в электронной книге о DB2 с панели DB2I.

Указания по настройке оперативной справки о DB2 смотрите в обсуждении настройки оперативной справки по DB2 в Разделе 2 руководства *DB2 Installation Guide*.

Если вы используете обновления с компакт–диска, обновленные книги можно сделать доступными из DB2I. Выберите на панели установок по умолчанию DB2I опцию 10 и введите имена новых наборов данных с книгами. Чтобы воспользоваться этой функцией, у вас должен быть доступ для записи к префикс.*SDSNCLST*.

Чтобы вызвать справку DB2I, нажмите клавишу PF1 (HELP)<sup>5</sup>.

## Меню основных опций DB2I

На рис. 108 показан пример меню основных опций DB2I. Отсюда можно непосредственно попасть на любую панель DB2I, не проходя те из них, которые вам не нужны. Чтобы связать программу, введите цифру, соответствующую пункту BIND/REBIND/FREE, и вы попадете на панель BIND PLAN, минуя все предшествующие.

Чтобы подготовить новую прикладную программу, начав с прекомпиляции и далее проходя через все последовательные этапы подготовки, введите 3, что соответствует панели подготовки программ (опция 3), как показано на рис. 108.

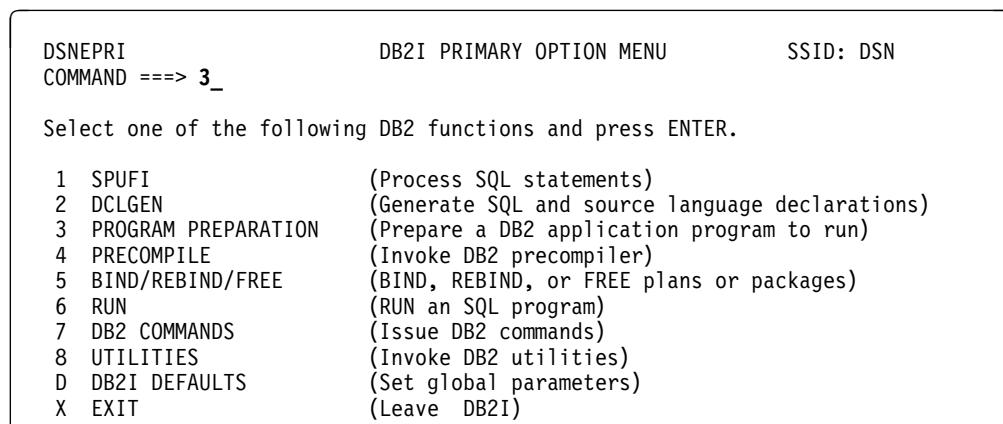


Рисунок 108. Запуск подготовки программ через DB2I. В меню основных опций DB2I выберите подготовку программ.

Далее приводится описание функций в меню основных опций DB2I.

### 1 SPUFI (ОБРАБОТКА ОПЕРАТОРОВ SQL)

Позволяет интерактивно выполнить один или несколько операторов SQL. Подробности смотрите в разделе “Глава 2–5. Исполнение SQL с вашего терминала при помощи SPUFI” на стр. 93.

### 2 DCLGEN (ГЕНЕРАЦИЯ ОПИСАНИЙ SQL И ИСХОДНОГО ЯЗЫКА)

Позволяет генерировать объявления данных таблиц C, COBOL и PL/I. Подробности смотрите в разделе “Глава 3–3. Генерация объявлений для таблиц при помощи DCLGEN” на стр. 133.

<sup>5</sup> Возможно, у вас для справки назначена другая клавиша PF.

- 3 PROGRAM PREPARATION (ПОДГОТОВКА ПРОГРАММЫ DB2 К ЗАПУСКУ)  
Позволяет подготавливать к запуску и запускать прикладные программы. Подробности смотрите в разделе “Панель подготовки программ DB2” на стр. 477.
- 4 PRECOMPILE (ВЫЗОВ ПРЕКОМПИЛЯТОРА DB2)  
Позволяет преобразовывать встроенные операторы SQL в операторы, пригодные для обработки языком хоста. Подробности смотрите в разделе “Панель прекомпиляции” на стр. 486.
- 5 BIND/REBIND/FREE (СВЯЗЫВАНИЕ, ПОВТОРНОЕ СВЯЗЫВАНИЕ И ОСВОБОЖДЕНИЕ ПЛАНОВ ИЛИ ПАКЕТОВ)  
Позволяет связывать, повторно связывать и освобождать план пакета или прикладной программы.
- 6 RUN (ЗАПУСК ПРОГРАММЫ SQL)  
Позволяет запускать прикладную программу в среде TSO или в пакетной среде.
- 7 DB2 COMMANDS (ВВОД КОМАНД DB2)  
Позволяет выполнять команды DB2. Подробности о командах DB2 смотрите в Главе 2 руководства *DB2 Command Reference*.
- 8 UTILITIES (ВЫЗОВ УТИЛИТ DB2)  
Позволяет вызывать утилиты DB2. Подробности смотрите в руководстве *DB2 Utility Guide and Reference*.
- D DB2I DEFAULTS (ЗАДАНИЕ ГЛОБАЛЬНЫХ ПАРАМЕТРОВ)  
Позволяет устанавливать параметры DB2I по умолчанию. Смотрите раздел “Панель параметров DB2I по умолчанию 1” на стр. 482.
- X EXIT (ВЫХОД ИЗ DB2I)  
Выход из DB2I.

## Панель подготовки программ DB2

Панель подготовки программ позволяет выбирать для выполнения конкретные функции подготовки программ. Для выбранных функций можно также задать, выводить ли панели для ввода опций выполнения этих функций. Приводим некоторые из функций, которые можно выбрать:

- Прекомпиляция. Панель для этой функции позволяет управлять прекомпилятором DB2. Смотрите страницу 486.
- Связывание пакета. Панель этой функции позволяет связывать DBRM программы в пакет (смотрите страницу 489), и изменять настройки по умолчанию для связывания этих пакетов (смотрите страницу 496).
- Связывание плана. Панель этой функции позволяет создавать планы применения программ (смотрите страницу 492), и изменять настройки по умолчанию для связывания этих планов (смотрите страницу 496).
- Компиляция, компоновка и запуск. Панель этих функций позволяет управлять компилятором или ассемблером и редактором связей. Смотрите страницу разделе 503.

### TSO и TSO batch

Для программ TSO программы подготовки программ можно использовать для управления процессором времени выполнения базового языка, а также самой программой.

Панель подготовки программ также дает возможность изменять параметры по умолчанию DB2I (смотрите страницу 482) и выполнять другие функции прекомпиляции и прокомпоновки.

На панели подготовки программ DB2, показанной на рис. 109, введите имя набора данных исходного текста программы (в этом примере используется SAMPLEPG.COBOL) и укажите другие опции, которые вы хотите использовать. Закончив, нажмите ENTER, чтобы перейти к следующей панели.

DSNEPP01	DB2 PROGRAM PREPARATION	SSID: DSN
COMMAND ==> _		
Enter the following:		
1 INPUT DATA SET NAME ....	==> SAMPLEPG.COBOL	
2 DATA SET NAME QUALIFIER	==> TEMP	(For building data set names)
3 PREPARATION ENVIRONMENT	==> FOREGROUND	(BACKGROUND, FOREGROUND, EDITJCL)
4 RUN TIME ENVIRONMENT ...	==> TSO	(TSO, CAF, CICS, IMS, RRSF)
5 OTHER DSNH OPTIONS .....	==>	(Optional DSNH keywords)
Select functions:      Display panel?      Perform function?		
6 CHANGE DEFAULTS .....	==> Y (Y/N)	==> N (Y/N)
7 PL/I MACRO PHASE .....	==> N (Y/N)	==> Y (Y/N)
8 PRECOMPILE .....	==> Y (Y/N)	==> N (Y/N)
9 CICS COMMAND TRANSLATION		
10 BIND PACKAGE .....	==> Y (Y/N)	==> Y (Y/N)
11 BIND PLAN.....	==> Y (Y/N)	==> Y (Y/N)
12 COMPILE OR ASSEMBLE ....	==> Y (Y/N)	==> Y (Y/N)
13 PRELINK.....	==> N (Y/N)	==> N (Y/N)
14 LINK.....	==> N (Y/N)	==> Y (Y/N)
15 RUN.....	==> N (Y/N)	==> Y (Y/N)

Рисунок 109. Панель подготовки программ DB2. Введите имя набора данных исходной программы и другие опции.

Ниже приводятся описание функций на панели подготовки программ DB2 и указания по заполнению полей для подготовки программы.

1 INPUT DATA SET NAME (ИМЯ ВХОДНОГО НАБОРА ДАННЫХ)

Позволяет задать имя входного набора данных. Входной набор данных может быть секционированным или последовательным набором данных, в имя может входить также имя компонента. Если не заключать имя набора данных в апострофы, к имени будет добавлен стандартный префикс TSO (идентификатор пользователя).

Указанное вами имя входного набора данных используется для прекомпиляции, связывания, компоновки—редактирования и для запуска программы.

2 DATA SET NAME QUALIFIER (СПЕЦИФИКАТОР ИМЕНИ НАБОРА ДАННЫХ)

Позволяет специфицировать имена временных наборов данных, используемых в процессе подготовки программы. Введите строку длиной от 1 до 8 символов, соответствующую обычным соглашениям по именам TSO. (По умолчанию задается TEMP.)

Для программ, подготавливаемых в фоновом режиме или использующих EDITJCL для опции PREPARATION ENVIRONMENT (среда подготовки), DB2 создает набор данных с именем *префикс\_tso.спецификатор.CNTL* с операторами JCL подготовки программы. Здесь *префикс\_tso* – это назначаемый TSO префикс, а *спецификатор* – значение, введенное в

поле DATA SET NAME QUALIFIER. Если набор данных с таким именем уже существует, DB2 удаляет его.

### 3 PREPARATION ENVIRONMENT (СРЕДА ПОДГОТОВКИ)

Позволяет задать, происходит подготовка программы в приоритетном или в фоновом режиме. Можно также задать EDITJCL, что даст возможность сначала отредактировать задание, а затем передать его на выполнение. Возможные варианты:

FOREGROUND (приоритетный режим) – использовать значения, заданные на панели подготовки программ, и немедленно запустить программу.

BACKGROUND (фоновый режим) – создать и передать на выполнение файл с DSNH CLIST, который немедленно запускается с использованием управляющего оператора JOB либо с панели параметров DB2I по умолчанию, либо вашим обработчиком SUBMIT. Этот файл будет сохранен.

EDITJCL – создать файл с DSNH CLIST и открыть его в режиме редактирования. Затем можно выполнить CLIST или сохранить его.

### 4 RUN TIME ENVIRONMENT (СРЕДА ВРЕМЕНИ ВЫПОЛНЕНИЯ)

Позволяет задать среду (TSO, CAF, CICS, IMS, RRSAF) выполнения вашей программы.

Все программы подготавливаются в TSO, но могут работать в любой из перечисленных сред. Если вы задаете CICS, IMS или RRSAF, в поле RUN (запуск) надо задать NO, поскольку такие программы нельзя запустить с панели подготовки программ. Если в поле RUN установить YES, можно задать только TSO или CAF.

(Пакетные программы также запускаются из–под TSO Terminal Monitor Program. Поэтому для пакетных программ в этом поле следует указывать TSO.)

### 5 OTHER DSNH OPTIONS (ДРУГИЕ ОПЦИИ DSNH)

Позволяет задать список опций DSNH, влияющих на процесс подготовки программы и переопределяющих опции на других панелях. При использовании CICS сюда могут входить опции, которые вы хотите передать транслятору команд CICS.

Если вы задаете в этом поле несколько опций, разделяйте их запятыми. Перечень опций можно продолжить на следующей строке, но общая длина списка должна быть не более 70 байт.

Подробнее об этих опциях смотрите описание DSNH в Главе 2 справочника *DB2 Command Reference*.

Описываемые далее поля с 7 по 15 позволяют выбрать функцию для выполнения и определить, показывать ли панели DB2I для выбранных функций. Ответ "да" вводите как Y, а "нет" – N.

Если вы хотите для всех этапов принять значения по умолчанию, в столбце Display panel (показать панель) введите N для всех панелей подготовки.

Для внесения изменений в устанавливаемые по умолчанию значения введите Y в столбце Display panel для всех панелей, которые вы хотите увидеть. После этого DB2I покажет каждую из запрошенных вами панелей. После показа всех панелей DB2 перейдет к отдельным этапам подготовки программы к запуску.

Переменные для всех функций, использованных во время подготовки программы, поддерживаются отдельно от переменных, введенных из меню основных опций DB2I. Например, переменные связывания плана, введенные на панели подготовки программ, сохраняются отдельно от переменных на любой из панелей связывания планов, на которые можно попасть из меню основных опций.

6 CHANGE DEFAULTS (ИЗМЕНИТЬ ОПЦИИ ПО УМОЛЧАНИЮ)

Позволяет задать необходимость изменения параметров по умолчанию DB2I. Для этого введите Y в поле *Display panel* у этой опции; в противном случае введите N. На панели параметров по умолчанию обязательно надо указать идентификатор вашей подсистемы и язык программирования. Подробности смотрите в разделе “Панель параметров DB2I по умолчанию 1” на стр. 482.

7 PL/I MACRO PHASE (СТАДИЯ МАКРОПРОЦЕССОРА PL/I)

Позволяет указать, выводить ли панель “Program Preparation: Compile, Link, and Run” (Подготовка программ: Компиляция, компоновка и запуск) для управления стадией макропроцессора PL/I путем ввода опций PL/I в поле OPTIONS (опции) на этой панели. Эта панель также выводится для опций COMPILE OR ASSEMBLE (компиляция или ассемблирование), LINK (компоновка) и RUN (запуск).

Это поле применяется только для программ на PL/I. Если ваша программа написана на другом языке PL/I или не использует макропроцессор PL/I, укажите в поле *Perform function* для этой опции N, что установит в поле *Display panel* по умолчанию значение N.

Информацию об опциях PL/I смотрите в разделе “Панель подготовки программ: Compile, Link и Run” на стр. 503.

8 PRECOMPILE (ПРЕКОМПИЛЯЦИЯ)

Позволяет задать вывод панели прекомпиляции. Чтобы увидеть эту панель, введите Y в поле *Display panel* возле этой опции, в противном случае введите N. Информацию о панели прекомпиляции смотрите в разделе “Панель прекомпиляции” на стр. 486.

9 CICS COMMAND TRANSLATION (ТРАНСЛЯЦИЯ КОМАНД CICS)

Позволяет задать необходимость использования транслятора команд CICS. Это поле применяется только для программ CICS.

**IMS и TSO**

Если вы работаете в TSO или IMS, проигнорируйте этот этап, что по умолчанию установит N в поле *Perform function*.

## CICS

Если вы используете CICS и уже прекомпилировали программу, ее необходимо отранслировать с использованием транслятора команд CICS.

В DB2I нет отдельной панели для транслятора команд. Опции трансляции можно задать либо в поле *Other options* панели подготовки программ DB2, либо в исходной программе, если она написана не на ассемблере.

Так как указана среда выполнения CICS, в столбце *Perform function* по умолчанию ставится Y. Трансляция команд происходит автоматически после прекомпиляции программы.

### 10 BIND PACKAGE (СВЯЗЫВАНИЕ ПАКЕТА)

Позволяет задать необходимость вывода панели BIND PACKAGE. Чтобы ее увидеть, введите Y в столбце *Display panel* у этой опции; в противном случае введите N. Информацию об этой панели смотрите в разделе “Панель BIND PACKAGE” на стр. 489.

### 11 BIND PLAN (СВЯЗАТЬ ПЛАН)

Позволяет задать необходимость вывода панели BIND PLAN. Чтобы увидеть эту панель, введите Y в столбце *Display panel* у этой опции; в противном случае введите N. Информацию об этой панели смотрите в разделе “Панель BIND PLAN” на стр. 492.

### 12 COMPILE OR ASSEMBLE (КОМПИЛЯЦИЯ ИЛИ АССЕМБЛИРОВАНИЕ)

Позволяет задать необходимость вывода панели “Program Preparation: Compile, Link, and Run” (Подготовка программ: компиляция, компоновка и запуск). Чтобы увидеть эту панель, введите Y в столбце *Display panel* у этой опции; в противном случае введите N.

Информацию об этой панели смотрите в разделе “Панель подготовки программ: Compile, Link и Run” на стр. 503.

### 13 PRELINK (ПРЕКОМПОНОВКА)

Позволяет использовать утилиту прекомпоновки, чтобы сделать программу на C, C++ или IBM COBOL for MVS & VM повторновходной. Эта утилита объединяет информацию инициализации во время компиляции из одного или нескольких текстовых подзаголовков в один модуль инициализации. Чтобы воспользоваться этой утилитой, введите Y в столбце *Display panel* у этой опции; в противном случае введите N. Если вы запросили этот этап, вам надо также запросить этапы компиляции и компоновки–редактирования.

Дополнительную информацию об утилите прекомпоновки смотрите в руководстве *OS/390 Language Environment for OS/390 & VM Programming Guide*.

### 14 LINK (КОМПОНОВКА)

Позволяет задать необходимость вывода панели “Program Preparation: Compile, Link, and Run” (Подготовка программ: компиляция, компоновка и запуск). Чтобы ее увидеть, введите Y в столбце *Display panel* у этой опции; в противном случае введите N. Если вы задаете Y в поле *Display panel* для опции COMPILE OR ASSEMBLE (компиляция или ассемблирование), не вносите изменений в это поле; для пункта COMPILE OR ASSEMBLE выводится та же самая панель, что и для

пункта LINK. Изменения, влияющие на этап компоновки—редактирования, можно делать одновременно с изменениями на этапе компиляции.

Информацию об этой панели смотрите в разделе “Панель подготовки программ: Compile, Link и Run” на стр. 503.

#### 15 RUN (ЗАПУСК)

Позволяет указать необходимость запуска программы. RUN можно указывать только, если в качестве среды выполнения задана TSO или CAF.

Если в поле *Display panel* для опций COMPILE OR ASSEMBLE или LINK задано Y, это поле можно не задавать, поскольку и для пункта COMPILE OR ASSEMBLE, и для пункта LINK выводится та же самая панель, что и для пункта RUN.

#### IMS и CICS

Программы IMS и CICS нельзя запускать из DB2I. При использовании IMS или CICS в этих полях поставьте N.

#### TSO и TSO batch

Если вы хотите запустить программу при использовании TSO, введите Y в столбце *Perform function* у этой опции. Можно также указать, что вы хотите задать опции и значения, влияющие на выполнение программы, введя Y в столбце *Display panel*.

Информацию об этой панели смотрите в разделе “Панель подготовки программ: Compile, Link и Run” на стр. 503.

Нажав ENTER, вы попадаете на первую из указанных вами панелей. В приведенном примере это будет панель параметров DB2I по умолчанию. Если в процессе перехода от панели к панели нажать кнопку END, вы возвратитесь к этой первой панели, на которой можно изменить указания по обработке. Звездочка (\*) в строках 7 –14 столбца *Display panel* укажут, какие из панелей вы уже посетили. Если вместо звездочки ввести Y, можно посетить панель еще раз.

## Панель параметров DB2I по умолчанию 1

Панель параметров DB2I по умолчанию 1 позволяет изменять многие из системных настроек, задающих значения по умолчанию и выбираемых при установке DB2. На рис. 110 показаны поля, влияющие на обработку других панелей DB2I.

```

DSNEOP01          DB2I DEFAULTS PANEL 1
COMMAND ==>_

Change defaults as desired:

1 DB2 NAME ..... ==> DSN      (Subsystem identifier)
2 DB2 CONNECTION RETRIES ==> 0   (How many retries for DB2 connection)
3 APPLICATION LANGUAGE ==> COBOL (ASM, C, CPP, COBOL, COB2, IBMCOB,
                                     FORTRAN, PLI)
4 LINES/PAGE OF LISTING ==> 60   (A number from 5 to 999)
5 MESSAGE LEVEL ..... ==> I     (Information, Warning, Error, Severe)
6 SQL STRING DELIMITER ==> DEFAULT (DEFAULT, ' or ")
7 DECIMAL POINT ..... ==> .     (. or ,)
8 STOP IF RETURN CODE >= ==> 8   (Lowest terminating return code)
9 NUMBER OF ROWS ..... ==> 20   (For ISPF Tables)
10 CHANGE HELP BOOK NAMES?==> NO  (YES to change HELP data set names)

```

Рисунок 110. Панель параметров DB2I по умолчанию 1

Ниже приводится описание полей на панели параметров DB2I по умолчанию 1.

1 DB2 NAME (ИДЕНТИФИКАТОР ПОДСИСТЕМЫ)

Позволяет задать подсистему DB2, обрабатывающую ваши запросы DB2I. Если вы задаете другую подсистему DB2, ее идентификатор появляется в поле SSID (идентификатор подсистемы) в правом верхнем углу экрана. По умолчанию устанавливается DSN.

2 DB2 CONNECTION RETRIES (СКОЛЬКО ПОПЫТОК СОЕДИНЕНИЯ С DB2 ДЕЛАТЬ)

Позволяет задать число повторных попыток соединения с DB2, если DB2 не подключилась по команде DSN от программы. В процессе подготовки программ эта опция не используется.

Возможны значения от 0 до 120. По умолчанию устанавливается 0.

Попытки соединения повторяются через 30—секундные интервалы.

3 APPLICATION LANGUAGE (ЯЗЫК ПРИКЛАДНОЙ ПРОГРАММЫ)

Позволяет задать для прикладной программы используемый по умолчанию язык программирования. Можно задать любой из следующих языков:

<b>ASM</b>	Для ассемблера высокого уровня/MVS
<b>C</b>	Для C/370
<b>CPP</b>	Для C++
<b>COBOL</b>	Для OS/VS COBOL (по умолчанию)
<b>COB2</b>	Для VS COBOL II
<b>IBMCOB</b>	Для IBM SAA AD/Cycle COBOL/370 или IBM COBOL for MVS & VM
<b>FORTRAN</b>	Для VS FORTRAN
<b>PLI</b>	Для PL/I

Если задать COBOL, COB2 или IBMCOB, DB2 запросит дополнительные параметры COBOL по умолчанию на панели DSNEOP02. Смотрите раздел “Панель параметров DB2I по умолчанию 2” на стр. 485.

Для программ IMS или CICS нельзя указать FORTRAN.

**4 LINES/PAGE OF LISTING (ЧИСЛО СТРОК НА СТРАНИЦЕ)**

Позволяет задать число строк распечатки на каждой из страниц печати или вывода SPUFI. Допустимы значения от 5 до 999, по умолчанию устанавливается 60.

**5 MESSAGE LEVEL (УРОВЕНЬ СООБЩЕНИЙ)**

Позволяет задать наименший уровень сообщения, которое на этапе связывания возвратит вас к процессу подготовки. Возможные значения:

- I** Любые информационные сообщения, предупреждения и сообщения о простых и серьезных ошибках
- W** Предупреждения и сообщения о простых и серьезных ошибках
- E** Сообщения о простых и серьезных ошибках
- S** Только сообщения о серьезных ошибках

**6 SQL STRING DELIMITER (ОГРАНИЧИТЕЛЬ СТРОК SQL)**

Позволяет задать символ—ограничитель строк в операторах SQL в программах на языке COBOL. Эта опция действительна только в том случае, если язык прикладной программы – COBOL, COB2 или IBMCOB.  
Использование:

- DEFAULT** Использовать значение по умолчанию, заданное при установке
  - ' Апостроф
  - " Кавычки

**7 DECIMAL POINT (ОТДЕЛИТЕЛЬ ДРОБНОЙ ЧАСТИ)**

Позволяет задать представление в исходном тексте программы отделятеля дробной части. Используется запятая (,) или точка (.). По умолчанию устанавливается точка (.).

**8 STOP IF RETURN CODE >= (ОСТАНОВКА ПРИ КОДЕ ВОЗВРАТА >=)**

Позволяет задать минимальное значение кода возврата (от препомпилиятора, компилятора, компоновщика–редактора связей и утилиты связывания), которое блокирует последующие этапы.  
Возможные варианты:

- 4** Остановка при предупреждениях и более серьезных ошибках.
- 8** Остановка при простых и более серьезных ошибках. По умолчанию устанавливается 8.

**9 NUMBER OF ROWS (ЧИСЛО СТРОК)**

Позволяет задать используемое по умолчанию число строк входных записей для генерации при первоначальном отображении панелей ISPF. Число строк с непустыми записями определяет число строк, показываемых на следующих экранах.

**10 CHANGE HELP BOOK NAMES? (ИЗМЕНИТЬ НАЗВАНИЯ КНИГ СПРАВКИ?)**

Позволяет изменить название книги BookManager®, на которую дается ссылка в оперативной справке. По умолчанию NO.

Предположим, что по умолчанию установлен язык программирования PL/I и число строк листинга программы на страницу равно 60. Ваша программа написана на языке COBOL, поэтому вы хотите изменить поле 3 APPLICATION LANGUAGE. Вы также хотите печатать 80 строк на странице, поэтому необходимо также изменить поле 4, LINES/PAGE OF LISTING. На рис. 110 на стр. 483 показано, что надо ввести на панели параметров DB2I по умолчанию 1, чтобы выполнить эти изменения. В этом случае после нажатия кнопки ENTER вы попадаете на панель параметров DB2 по умолчанию 2.

## Панель параметров DB2I по умолчанию 2

После нажатия кнопки Enter на панели параметров DB2I по умолчанию 1 появляется панель параметров DB2I по умолчанию 2. Если на панели параметров DB2I по умолчанию 1 в качестве языка выбран COBOL, COB2 или IBMCOB, показываются три поля. В противном случае показывается только первое поле. На рис. 111 показана панель параметров DB2I по умолчанию 2 при выборе COBOL, COB2 или IBMCOB.

```
DSNEOP02          DB2I DEFAULTS PANEL 2
COMMAND ==>_

Change defaults as desired:

1 DB2I JOB STATEMENT: (Optional if your site has a SUBMIT exit)
  ==> //USRTO01A JOB (ACCOUNT),'NAME'
  ==> //*
  ==> //*
  ==> //*

COBOL DEFAULTS:           (For COBOL, COB2, or IBMCOB)
2 COBOL STRING DELIMITER ==> DEFAULT (DEFAULT, ' or ")
3 DBCS SYMBOL FOR DCLGEN ==> G      (G/N - Character in PIC clause)
```

Рисунок 111. Панель параметров DB2I по умолчанию 2

### 1 DB2I JOB STATEMENT (ОПЕРАТОР JOB DB2I)

Позволяет изменять используемый по умолчанию оператор задания. Укажите оператор JCL JOB и (не обязательно) оператор JOBLIB для использования либо в фоновом режиме, либо в среде подготовки программ EDITJCL. Используйте оператор JOBLIB для задания необходимой прикладной программе библиотеки времени выполнения. Если у программы есть подпрограмма—обработчик SUBMIT, DB2 использует эту подпрограмму. Если эта процедура строит оператор управления заданием, поле можно оставить пустым.

### 2 COBOL STRING DELIMITER (ОГРАНИЧИТЕЛЬ СТРОКИ COBOL)

Позволяет задать символ—ограничитель строки в операторах COBOL прикладной программы на языке COBOL. Возможные варианты:

<b>DEFAULT</b>	Использовать значение по умолчанию, заданное при установке
'	Апостроф
"	Кавычки

Пробел задает использование значения по умолчанию.

### 3 DBCS SYMBOL FOR DCLGEN (СИМВОЛ DBCS ДЛЯ DCLGEN)

Позволяет вводить либо G (по умолчанию), либо N, чтобы указать, генерирует ли DCLGEN условие изображения вида PIC G(*n*) DISPLAY-1 или PIC N(*n*).

Пробел задает использование значения по умолчанию.

Нажав ENTER, вы попадаете на следующую панель из указанных вами на панели подготовки программ DB2, в данном случае – на панель прокомпиляции.

## Панель прекомпиляции

Следующий этап процесса – прекомпиляция. На рис. 108 на стр. 476 (Меню основных опций DB2I) показано, что на панель прекомпиляции можно попасть двумя способами: либо указав прекомпиляцию в качестве составной части процесса подготовки программы на панели подготовки программ DB2, либо непосредственно из меню основных опций DB2I. От того, каким способом вы попадаете на эту панель, зависит, какие значения устанавливаются по умолчанию в ее полях. На рис. 112 показана панель прекомпиляции.

```
DSNETP01          PRECOMPILE          SSID: DSN
COMMAND ==>_

Enter precompiler data sets:
1 INPUT DATA SET .... ==> SAMPLEPG.COBOL
2 INCLUDE LIBRARY ... ==> SRCLIB.DATA

3 DSNAME QUALIFIER .. ==> TEMP      (For building data set names)
4 DBRM DATA SET ..... ==>

Enter processing options as desired:
5 WHERE TO PRECOMPILE ==> FOREGROUND (FOREGROUND, BACKGROUND, or EDITJCL)
6 VERSION ..... ==>                  (Blank, VERSION, or AUTO)
7 OTHER OPTIONS ..... ==>
```

Рисунок 112. Панель прекомпиляции. Укажите вызываемую библиотеку (если надо), которую должна использовать программа, а также прочие необходимые опции.

Ниже описываются функции на панели прекомпиляции и заполнение полей для подготовки к прекомпиляции.

### 1 INPUT DATA SET (ВХОДНОЙ НАБОР ДАННЫХ)

Позволяет задать имя набора данных с исходным текстом программы и операторы SQL для прекомпиляции.

Если вы попали на эту панель через панели подготовки программ DB2, в этом поле будет стоять указанное там имя набора данных. При желании на этой панели его можно изменить.

Если вы попали на эту панель непосредственно из меню основных опций DB2I, надо ввести имя набора данных программы, которую вы собираетесь прекомпилировать. Имя набора данных может включать в себя имя компонента. Если вы не заключили имя набора данных в апострофы, к нему будет добавлен стандартный префикс TSO (идентификатор пользователя).

### 2 INCLUDE LIBRARY (ВЫЗЫВАЕМАЯ БИБЛИОТЕКА)

Позволяет ввести имя библиотеки, содержащей компоненты, которые должен использовать прекомпилятор. Эти компоненты могут содержать вывод DCLGEN. Если вы не заключили это имя в апострофы, к нему будет добавлен стандартный префикс TSO (идентификатор пользователя).

Можно запросить дополнительные библиотеки, вызываемые по INCLUDE, введя параметры DSNH CLIST в виде PnLIB(dsname), где  $n = 2, 3$  или  $4$  в поле OTHER OPTIONS на этой панели или в поле OTHER DSNH OPTIONS на панели подготовки программ.

### 3 DSNAMES QUALIFIER (СПЕЦИФИКАТОР DSNAMES)

Позволяет задать строку символов, задающую имена временных наборов данных для прекомпиляции. Можно ввести строку длиной от 1 до 8 символов, удовлетворяющую обычным соглашениям по именам TSO.

Если вы попали на эту панель через панели подготовки программ DB2, в этом поле содержится заданный там спецификатор имени набора данных. При желании на этой панели его можно изменить.

Если вы попали на эту панель непосредственно из меню основных опций DB2I, можно либо задать DSNAMES QUALIFIER, либо оставить в поле значение по умолчанию TEMP.

#### IMS и TSO

Для программ IMS и TSO DB2 сохраняет прекомпилированные исходные операторы (для передачи на этап компиляции или ассемблирования) в наборе данных с именем *префикс\_tso.спецификатор.суффикс*. В набор данных с именем *префикс\_tso.спецификатор.PCLIST* записывается вывод прекомпиляции.

Для программ, подготовленных в фоновом режиме или использующих опцию EDITJCL PREPARATION ENVIRONMENT (на панели подготовки программ DB2), в наборе данных с именем *префикс\_tso.спецификатор.CNTL* содержится JCL подготовки программы.

В этих примерах *префикс\_tso* – это назначаемый TSO префикс, часто совпадающий с идентификатором авторизации. *спецификатор* – это значение, введенное в поле DSNAMES QUALIFIER, а *суффикс* – выходное расширение из следующих: COBOL, FORTRAN, C, PLI, ASM, DECK, CICSIN, OBJ или DATA. В примере на рис. 112 на стр. 486 в наборе данных *префикс\_tso.TEMP.COBOL* содержатся прекомпилированные исходные операторы, а в наборе *префикс\_tso.TEMP.PCLIST* – вывод прекомпиляции. Если наборы данных с такими именами уже существуют, DB2 их удалит.

#### CICS

Для программ CICS в набор данных *префикс\_tso.спецификатор.suffix* в процессе подготовки команд CICS к трансляции заносятся прекомпилированные исходные операторы.

Если вы не собираетесь проводить трансляцию команд CICS, исходные операторы в наборе данных *префикс\_tso.спецификатор.suffix*, готовы к компиляции. В набор данных *префикс\_tso.спецификатор.PCLIST* помещается вывод прекомпиляции.

После завершения работы прекомпилятора управление переходит к транслятору команд CICS. Поскольку у транслятора нет своей панели, трансляция происходит автоматически. В набор данных *префикс\_tso.спецификатор.CXLIST* помещается вывод транслятора команд.

#### 4 DBRM DATA SET (НАБОР ДАННЫХ DBRM)

Позволяет присваивать имена наборам данных библиотек DBRM для вывода прекомпилиатора. В имя набора данных может входить имя компонента.

Когда вы попадаете на панель, это поле пусто. Однако если нажать ENTER, будет использован набор данных с именем, получаемым добавлением к значению поля DSNAME QUALIFIER этой панели расширения *DBRM: спецификатор.DBRM*.

Другое имя набора данных можно ввести в это поле только в том случае, если перед этим вы разместите и закатологизируете этот набор данных. Это справедливо даже в том случае, если введенное вами имя набора данных совпадает с тем, которое будет использовано для этого поля по умолчанию.

Прекомпилиатор посыпает модифицированный исходный код в набор данных *спецификатор.язык*, где *язык* – язык, указанный в поле APPLICATION LANGUAGE панели параметров DB2I по умолчанию 1.

#### 5 WHERE TO PRECOMPILE (КАК ПРОВОДИТЬ ПРЕКОМПИЛЯЦИЮ)

Позволяет указать, проводить прекомпилиацию в приоритетном или в фоновом режиме. Можно также задать EDITJCL, что даст возможность сначала отредактировать задание, а затем передать его на выполнение.

Если вы попали на эту панель с панели подготовки программ DB2, в этом поле содержится указанная там среда подготовки. При желании это значение можно изменить.

Если вы попали на эту панель непосредственно из меню основных опций DB2I, можно либо задать среду обработки, либо оставить в этом поле значение по умолчанию. Возможные варианты:

FOREGROUND – немедленно прекомпилировать программу с заданными на этих панелях значениями.

BACKGROUND – создать и передать на выполнение файл с DSNH CLIST с использованием управляющего оператора JOB либо с панели параметров DB2I по умолчанию 2, либо при помощи вашего обработчика SUBMIT. Этот файл будет сохранен.

EDITJCL – создать файл с DSNH CLIST и открыть его в режиме редактирования. Затем можно выполнить CLIST или сохранить его.

#### 6 VERSION (ВЕРСИЯ)

Позволяет задать версию программы и ее DBRM. Если имя версии содержит максимально допустимое число символов (64), все символы необходимо вводить без пробелов между строками. Это необязательный параметр.

Более подробно об этой опции смотрите в разделе “Преимущества пакетов” на стр. 343.

#### 7 OTHER OPTIONS (ДРУГИЕ ОПЦИИ)

Позволяет вводить любую опцию, принимаемую DSNH CLIST, что дает дополнительные возможности управления программой. Заданные в этом поле опции DSNH переопределяют опции, заданные на других панелях. Список опций может продолжаться на следующей строке, но общая длина списка не должна превышать 70 байтов.

Дополнительную информацию об опциях DSNH смотрите в Главе 2 справочника *DB2 Command Reference*.

## Панель BIND PACKAGE

Если выбрана эта опция, появляется первая из двух панелей BIND PACKAGE. На панель BIND PACKAGE можно попасть либо непосредственно из меню основных опций DB2I, либо в процессе подготовки программы. Если войти на панель BIND PACKAGE с панели подготовки программ, во многих полях панели BIND PACKAGE будут находиться значения из основной панели и панели прекомпиляции. На рис. 113 показана панель BIND PACKAGE.

DSNEBP07	BIND PACKAGE	SSID: DSN
COMMAND ==>_		
Specify output location and collection names:		
1 LOCATION NAME .....	==>	(Defaults to local)
2 COLLECTION-ID .....	==>	(Required)
Specify package source (DBRM or COPY):		
3 DBRM: COPY: .....	==> DBRM	(Specify DBRM or COPY)
4 MEMBER or COLLECTION-ID .....	==>	
5 PASSWORD or PACKAGE-ID ..	==>	
6 LIBRARY or VERSION .....	==>	
7 ..... -- OPTIONS .....	==>	(Blank, or COPY version-id) (COMPOSITE or COMMAND)
Enter options as desired:		
8 CHANGE CURRENT DEFAULTS? .....	==> NO	(NO or YES)
9 ENABLE/DISABLE CONNECTIONS? .....	==> NO	(NO or YES)
10 OWNER OF PACKAGE (AUTHID)..	==>	(Leave blank for primary ID)
11 QUALIFIER .....	==>	(Leave blank for OWNER)
12 ACTION ON PACKAGE .....	==> REPLACE	(ADD or REPLACE)
13 INCLUDE PATH? .....	==> NO	(NO or YES)
14 REPLACE VERSION .....	==>	(Replacement version-id)

Рисунок 113. Панель BIND PACKAGE

Далее описываются функции на панели BIND PACKAGE и заполнение полей, необходимых для связывания программы. Подробности смотрите в описании команды BIND PACKAGE в Главе 2 *DB2 Command Reference*.

### 1 LOCATION NAME (ИМЯ ПОЛОЖЕНИЯ)

Позволяет задать систему, в которой будет связан пакет. Имя положения может иметь длину от 1 до 16 символов. Имя положения должно быть определено в таблице каталога SYSLOCATIONS. По умолчанию устанавливается локальная DBMS.

### 2 COLLECTION-ID (ИДЕНТИФИКАТОР СОБРАНИЯ)

Позволяет задать собрание, в котором расположен пакет. Идентификатора может содержать от 1 до 18 символов, первый из которых должен быть буквой.

### 3 DBRM: COPY: (DBRM ИЛИ КОПИЯ)

Позволяет указать, создается ли новый пакет (DBRM), или делается копия уже существующего пакета (COPY). Возможные варианты:

#### DBRM

Создать новый пакет. Необходимо задать значения в полях LIBRARY, PASSWORD и MEMBER.

## COPY

Копировать существующий пакет. Необходимо задать значения в полях COLLECTION-ID и PACKAGE-ID. (Можно также задать значение в поле VERSION.)

### 4 MEMBER OR COLLECTION-ID (КОМПОНЕНТ ИЛИ ИДЕНТИФИКАТОР СОБРАНИЯ)

**MEMBER (для новых пакетов):** Если вы создаете новый пакет, эта опция позволяет задать DBRM для связывания. Имя компонента может содержать от 1 до 8 символов. Используемое по умолчанию имя зависит от имени входного набора данных.

- Для секционированного входного набора данных по умолчанию используется имя компонента входного набора данных, указанное в поле INPUT DATA SET NAME на панели подготовки программ DB2.
- Для последовательного входного набора данных в качестве имени по умолчанию используется второй спецификатор этого входного набора данных.

### COLLECTION-ID (идентификатор собрания) (для копирования пакета):

Если вы копируете пакет, эта опция задает идентификатор собрания, в котором содержится исходный пакет. Идентификатор собрания может содержать от 1 до 18 символов и отличаться от идентификатора собрания, указанного в поле PACKAGE ID.

### 5 PASSWORD OR PACKAGE-ID (ПАРОЛЬ ИЛИ ИДЕНТИФИКАТОР ПАКЕТА)

**PASSWORD (для новых пакетов):** При создании нового пакета здесь можно ввести пароль для библиотеки, упомянутой в поле LIBRARY. Это поле можно использовать только, если вы попали на панель BIND PACKAGE непосредственно из меню основных опций DB2.

**PACKAGE-ID (для копирования пакетов):** При копировании пакета эта опция позволяет задать имя исходного пакета. Идентификатор пакета может содержать от 1 до 8 символов.

### 6 LIBRARY OR VERSION (БИБЛИОТЕКА ИЛИ ВЕРСИЯ)

**LIBRARY (для новых пакетов):** При создании нового пакета здесь можно ввести имена библиотек, содержащих DBRM, указанные для процесса связывания в поле MEMBER. Библиотеки ищутся в указанном порядке и должны быть заданы в таблицах каталогов.

**VERSION (для копирования пакетов):** При копировании пакета эта опция позволяет задавать версию исходного пакета. Можно задавать идентификатор версии длиной от 1 до 64 символов. Более подробно об этой опции смотрите в разделе “Преимущества пакетов” на стр. 343.

### 7 OPTIONS (ОПЦИИ)

Позволяет задать, какие опции связывания DB2 использует с командой BIND PACKAGE с опцией COPY. Возможные варианты:

**COMPOSITE (по умолчанию)** DB2 будет использовать все опции, указанные вами в команде BIND PACKAGE. Для прочих опций DB2 использует опции копируемого пакета.

**COMMAND** DB2 будет использовать опции, указанные вами в команде BIND PACKAGE. Для прочих опций DB2 использует следующие значения:

- Для локальной копии пакета DB2 использует установки по умолчанию для локальной подсистемы DB2.

- Для удаленной копии пакета DB2 использует установки по умолчанию для сервера, на котором связан этот пакет.

#### 8 CHANGE CURRENT DEFAULTS? (ИЗМЕНИТЬ ТЕКУЩИЕ ПАРАМЕТРЫ ПО УМОЛЧАНИЮ?)

Позволяет задать необходимость изменения текущих параметров по умолчанию для связывания пакетов. Если ввести в это поле YES, на следующем шаге появится панель параметров по умолчанию BIND PACKAGE. На ней можно ввести новые значения; указания смотрите в разделе “Панели параметров по умолчанию для связывания или повторного связывания плана или пакета” на стр. 496.

#### 9 ENABLE/DISABLE CONNECTIONS? (РАЗРЕШИТЬ/ЗАПРЕТИТЬ СОЕДИНЕНИЯ?)

Позволяет задать, хотите ли вы для этого пакета разрешить или запретить различные виды соединений системы. Действует только в том случае, если в поле LOCATION NAME указано имя вашей локальной системы DB2.

Если в это поле ввести YES, появится панель (показанная на рис. 117 на стр. 501), которая позволяет указать, разрешены ли для этой прикладной программы те или иные соединения системы. Можно задать имена соединений для дальнейшей идентификации разрешенных соединений в пределах одного вида соединений. Имя соединения действительно только в том случае, если одновременно задан соответствующий ему вид соединения.

По умолчанию разрешены все виды соединений.

#### 10 OWNER OF PACKAGE (AUTHID) (ВЛАДЕЛЕЦ ПАКЕТА)

Позволяет задать первичный идентификатор авторизации владельца нового пакета. Этот идентификатор – имя владельца пакета, которое указывается во всех учетных записях и записях трассировки этого пакета.

У владельца должны быть привилегии, необходимые для запуска содержащихся в этом пакете операторов SQL.

По умолчанию устанавливается первичный идентификатор авторизации процесса связывания.

#### 11 QUALIFIER (СПЕЦИФИКАТОР)

Позволяет задать в явном виде спецификатор для таблиц, производных таблиц, индексов и алиасов, у которых он не указан. Длина спецификатора – от 1 до 8 символов. По умолчанию устанавливается идентификатор авторизации владельца пакета.

#### 12 ACTION ON PACKAGE (ДЕЙСТВИЕ НАД ПАКЕТОМ)

Позволяет задать замену существующего пакета или создание нового. Возможные варианты:

**REPLACE (по умолчанию)** – заменить пакет, имя которого введено в поле PACKAGE-ID, если этот пакет уже существует, и добавить его, если такого пакета не существует. (Эта опция используется, если вы изменяете пакет, поскольку в программе изменяются операторы SQL. Если изменяется только среда SQL, но не операторы SQL, можно использовать REBIND PACKAGE.)

**ADD** – добавить пакет, имя которого введено в поле PACKAGE-ID, если такого пакета еще не существует.

#### 13 INCLUDE PATH? (ВКЛЮЧИТЬ ПУТЬ?)

Указывает, предоставите ли вы список имен схем, где DB2 будет проводить поиск при разрешении в операторах SQL неспецифицированных имен пользовательских типов, пользовательских функций и хранимых процедур. По умолчанию задается NO. Если задать YES, DB2 выведет панель задания имен схем для поиска DB2.

#### 14 REPLACE VERSION (ЗАМЕНИТЬ ВЕРСИЮ)

Позволяет указать, заменять ли конкретную версию существующего пакета или создавать новую. Если и пакет, и версия, указанные в полях PACKAGE-ID и VERSION, уже существуют, можно задать REPLACE. Можно задавать идентификатор версии длиной от 1 до 64 символов. По умолчанию используется идентификатор версии, указанный в поле VERSION.

## Панель BIND PLAN

Данная панель BIND PLAN – первая из двух панелей BIND PLAN. На ней задаются опции для процесса связывания плана прикладной программы. Как и на панель прекомпиляции, на панель BIND PLAN можно попасть либо непосредственно из меню первичных опций DB2I, либо в процессе подготовки программы. План прикладной программы должен быть у вас даже в том случае, если вы связываете ее с пакетами; эта панель также появляется после панелей BIND PACKAGE.

При входе на панель BIND PLAN с панели подготовки программ во многих полях панели BIND PLAN будут находиться значения с основной панели и панели прекомпиляции. Смотрите рис. 114.

```
DSNEBP02          BIND PLAN          SSID: DSN
COMMAND ==>_

Enter DBRM data set name(s):
1 MEMBER ..... ==> SAMPLEPG
2 PASSWORD ..... ==>
3 LIBRARY ..... ==> TEMP.DBRM
4 ADDITIONAL DBRMS? ..... ==> NO      (YES to list more DBRMs)

Enter options as desired:
5 PLAN NAME ..... ==> SAMPLEPG      (Required to create a plan)
6 CHANGE CURRENT DEFAULTS? ==> NO    (NO or YES)
7 ENABLE/DISABLE CONNECTIONS? ==> NO   (NO or YES)
8 INCLUDE PACKAGE LIST? .... ==> NO   (NO or YES)
9 OWNER OF PLAN (AUTHID) ... ==>      (Leave blank for your primary ID)
10 QUALIFIER ..... ==>                (For tables, views, and aliases)
11 CACHESIZE ..... ==> 0             (Blank, or value 0-4096)
12 ACTION ON PLAN ..... ==> REPLACE  (REPLACE or ADD)
13 RETAIN EXECUTION AUTHORITY ==> YES  (YES to retain user list)
14 CURRENT SERVER ..... ==>           (Location name)
15 INCLUDE PATH? ..... ==>            (NO or YES)
```

Рисунок 114. Панель BIND PLAN

Ниже приводится описание функций на панели BIND PLAN и правила заполнения полей для связывания программы. Подробности смотрите в описании команды BIND PLAN в Главе 2 справочника *DB2 Command Reference*.

#### 1 MEMBER (КОМПОНЕНТ)

Позволяет задать DBRM для включения в план. Имена могут содержать от 1 до 8 символов. Необходимо задать MEMBER, INCLUDE PACKAGE

LIST или и то, и другое. Если не задавать MEMBER, поля 2, 3 и 4 игнорируются.

Используемое по умолчанию имя компонента зависит от входного набора данных.

- Для секционированного входного набора данных по умолчанию используется имя компонента входного набора данных, указанное в поле 1 панели подготовки программ DB2.
- Для последовательного входного набора данных в качестве имени по умолчанию используется второй спецификатор этого входного набора данных.

Если вы попали на эту панель непосредственно из меню первичных опций DB2I, необходимо ввести значения в поля MEMBER и LIBRARY.

Если планируется использование нескольких DBRM, можно ввести имя библиотеки и имя компонента каждого из DBRM в поля MEMBER и LIBRARY, разделяя записи запятыми. Можно также задать дополнительные DBRM в поле ADDITIONAL DBRMS? на этой панели.

## 2 PASSWORD (ПАРОЛЬ)

Позволяет вводить пароли для библиотек, перечисленных в поле LIBRARY. Это поле доступно только в том случае, если вы попали на панель BIND PLAN непосредственно из меню первичных опций DB2.

## 3 LIBRARY (БИБЛИОТЕКА)

Позволяет задать имена библиотек с DBRM, используемыми в процессе связывания. Имя может содержать до 44 символов.

## 4 ADDITIONAL DBRMS? (ДОПОЛНИТЕЛЬНЫЕ DBRM?)

Позволяет задать дополнительные DBRM в случае недостатка места. Если вы попали на эту панель в процессе подготовки программы, можно включить дополнительные DBRM, указав в этом поле YES. После этого появится отдельная панель, где можно ввести имена дополнительных библиотек и компонентов DBRM;смотрите “Панели для ввода списков значений” на стр. 502.

## 5 PLAN NAME (ИМЯ ПЛАНА)

Позволяет задать имя создаваемого плана прикладной программы. Имя может содержать от 1 до 8 символов, первый символ должен быть буквой. При отсутствии ошибок процесс связывания подготавливает план и вводит его описание в таблицу EXPLAIN.

Если вы попали на эту панель через панели подготовки программ DB2, устанавливаемое по умолчанию для этого поля значение зависит от значения, введенного там в поле INPUT DATA SET NAME.

Если вы попали на эту панель непосредственно из меню первичных опций DB2 и хотите создать план прикладной программы, здесь необходимо ввести имя этого плана. Устанавливаемое по умолчанию в этом поле имя зависит от входного набора данных:

- Для секционированного входного набора данных по умолчанию используется имя компонента.
- Для последовательного входного набора данных в качестве имени по умолчанию используется второй спецификатор имени набора данных.

## 6 CHANGE CURRENT DEFAULTS? (ИЗМЕНИТЬ ТЕКУЩИЕ ПАРАМЕТРЫ ПО УМОЛЧАНИЮ?)

Позволяет указать, надо ли изменять текущие параметры по умолчанию для связывания планов. Если в это поле ввести YES, на следующем этапе появится панель параметров по умолчанию BIND PLAN. На ней можно ввести новые значения; указания смотрите в разделе “Панели параметров по умолчанию для связывания или повторного связывания плана или пакета” на стр. 496.

## 7 ENABLE/DISABLE CONNECTIONS? (РАЗРЕШИТЬ/ЗАПРЕТИТЬ СОЕДИНЕНИЯ?)

Позволяет задать, хотите ли вы для этого пакета разрешить или запретить различные виды соединений системы. Действует только в том случае, если в поле LOCATION NAME указано имя вашей локальной системы DB2.

Если в это поле ввести YES, появится панель (показанная на рис. 117 на стр. 501), которая позволяет указать, разрешены ли для этой прикладной программы те или иные соединения системы. Можно задать имена соединений для дальнейшей идентификации разрешенных соединений в пределах одного вида соединений. Имя соединения действительно только в том случае, если одновременно задан соответствующий ему вид соединения.

По умолчанию разрешены все виды соединений.

## 8 INCLUDE PACKAGE LIST? (ВКЛЮЧИТЬ СПИСОК ПАКЕТОВ?)

Позволяет включить в план список пакетов. Если задать YES, появится отдельная панель, на которой надо ввести положение пакета, имя собрания и имя пакета для каждого из включаемых в план пакетов (смотрите раздел “Панели для ввода списков значений” на стр. 502). При использовании поля MEMBER этот список задавать не обязательно.

Можно задать имя положения длиной от 1 до 16 символов, идентификатор собрания длиной от 1 до 18 символов и идентификатор пакета длиной от 1 до 8 символов. Если вы задаете дополнительное имя положения, оно должно быть записано в таблице каталога SYSLOCATIONS; по умолчанию положение – локальная DBMS.

При вводе для связывания плана необходимо задать либо INCLUDE PACKAGE LIST?, либо MEMBER, либо и то и другое.

## 9 OWNER OF PLAN (AUTHID) (ВЛАДЕЛЕЦ ПЛАНА)

Позволяет задать первичный идентификатор авторизации владельца нового плана. Этот идентификатор – имя владельца пакета, которое указывается во всех учетных записях и записях трассировки этого пакета.

У владельца должны быть привилегии, необходимые для запуска содержащихся в этом пакете операторов SQL.

## 10 QUALIFIER (СПЕЦИФИКАТОР)

Позволяет задать в явном виде спецификатор для таблиц, производных таблиц, индексов и алиасов, у которых он не указан. Длина спецификатора – от 1 до 8 символов, он должен соответствовать правилам именования коротких идентификаторов SQL. Если оставить это поле пустым, по умолчанию устанавливается идентификатор авторизации владельца пакета.

#### 11 CACHESIZE (РАЗМЕР КЭША)

Позволяет задать размер (в байтах) кэша авторизации. Допускаются значения от 0 до 4096. Значения, не кратные 256, округляются до следующего за ним значения, кратного 256. 0 означает, что DB2 не использует кэш авторизации. По умолчанию задается 1024.

Для каждого из одновременных пользователей плана требуется 8 байт памяти + дополнительно 32 байта. Дополнительную информацию об этой опции смотрите в разделе “Определение оптимального размера кэша авторизации” на стр. 459.

#### 12 ACTION ON PLAN (ДЕЙСТВИЕ НАД ПЛАНОМ)

Позволяет указать, является план прикладной программы новым или измененным. Возможные варианты:

**REPLACE (по умолчанию)** – заменить план, указанный в поле PLAN NAME, если он уже существует, и добавить этот план, если он еще не существует.

**ADD** – добавить план, указанный в поле PLAN NAME, если он еще не существует.

#### 13 RETAIN EXECUTION AUTHORITY (СОХРАНИТЬ АВТОРИЗАЦИЮ ВЫПОЛНЕНИЯ)

Позволяет выбрать, будут ли сохранять свои полномочия в отношении измененного плана пользователи, имевшие полномочия на связывание или запуск исходного плана. Это поле используется только при замене существующего плана.

Если при изменении права собственности на план ввести YES, новый пользователь предоставляет полномочия на связывание и на выполнение предыдущему владельцу плана.

Если при изменении права собственности на план не указать YES, все, кроме нового владельца плана, потеряют полномочия на выполнение (но не полномочия на связывание), и новый владелец плана предоставляет полномочия на связывание предыдущему владельцу плана.

#### 14 CURRENT SERVER (ТЕКУЩИЙ СЕРВЕР)

Позволяет задать начальный сервер для приема и обработки операторов SQL в этом плане. Можно задать имя длиной от 1 до 16 символов; оно должно быть предварительно задано в таблице каталога SYSLOCATIONS.

Если задать удаленный сервер, DB2 свяжется с ним при выполнении первого оператора SQL. По умолчанию устанавливается имя локальной подсистемы DB2. Подробнее об этой опции смотрите в описании опции связывания CURRENTSERVER в главе 2 справочника *DB2 Command Reference*.

#### 15 INCLUDE PATH? (ВКЛЮЧИТЬ ПУТЬ?)

Указывает, предоставите ли вы список имен схем, где DB2 будет проводить поиск при разрешении в операторах SQL неспецифицированных имен пользовательских типов, пользовательских функций и хранимых процедур. По умолчанию задается NO. Если задать YES, DB2 выведет панель задания имен схем для поиска DB2.

После внесения необходимых изменений на этой панели нажмите ENTER для перехода на вторую из панелей подготовки программ, Program Prep: Compile, Link, and Run.

## Панели параметров по умолчанию для связывания или повторного связывания плана или пакета

На этой панели вводятся новые параметры по умолчанию для связывания пакетов.

```
DSNEBP10          DEFAULTS FOR BIND PACKAGE      SSID: DSN
COMMAND ==>_

Change default options as necessary:

 1 ISOLATION LEVEL ..... ==>      (RR, RS, CS, UR, or NC)
 2 VALIDATION TIME ..... ==>      (RUN or BIND)
 3 RESOURCE RELEASE TIME ... ==>    (COMMIT or DEALLOCATE)
 4 EXPLAIN PATH SELECTION .. ==>   (NO or YES)
 5 DATA CURRENCY ..... ==>      (NO or YES)
 6 PARALLEL DEGREE ..... ==>     (1 or ANY)
 7 SQLERROR PROCESSING .... ==>   (NOPACKAGE or CONTINUE)
 8 REOPTIMIZE FOR INPUT VARS ==>  (NO OR YES)
 9 DEFER PREPARE ..... ==>       (NO OR YES)
10 KEEP DYN SQL PAST COMMIT ==>   (NO or YES)
11 DBPROTOCOL ..... ==>        (DRDA OR PRIVATE)
12 OPTIMIZATION HINT ..... ==>   (Blank or 'hint-id')
13 DYNAMIC RULES ..... ==>      (RUN, BIND,
                                  DEFINERUN, DEFINEBIND,
                                  INVOKERUN, INVOKEBIND)
```

Рисунок 115. Панель параметров по умолчанию для связывания пакетов

Эта панель позволяет изменять параметры по умолчанию для опций BIND PACKAGE. Почти все опции на этой панели — те же, что и опции для параметров по умолчанию при повторном связывании пакета. Однако параметры по умолчанию для REBIND PACKAGE отличаются от тех, которые приведены на рисунке выше, кроме того, и в любом поле можно указать SAME, чтобы задать те же значения, что и при последнем связывании этого пакета. При повторном связывании для всех полей по умолчанию устанавливается значение SAME.

На этой панели вводятся новые параметры по умолчанию для связывания плана.

DSNEBP10	DEFAULTS FOR BIND PLAN	SSID: DSN
COMMAND ==>		
Change default options as necessary:		
1 ISOLATION LEVEL .....	==> RR	(RR, RS, CS, or UR)
2 VALIDATION TIME .....	==> RUN	(RUN or BIND)
3 RESOURCE RELEASE TIME ..	==> COMMIT	(COMMIT or DEALLOCATE)
4 EXPLAIN PATH SELECTION ..	==> NO	(NO or YES)
5 DATA CURRENCY .....	==> NO	(NO or YES)
6 PARALLEL DEGREE .....	==> 1	(1 or ANY)
7 RESOURCE ACQUISITION TIME ==>	USE	(USE or ALLOCATE)
8 REOPTIMIZE FOR INPUT VARS ==>	NO	(NO OR YES)
9 DEFER PREPARE .....	==> NO	(NO or YES)
10 KEEP DYN SQL PAST COMMIT. ==>	NO	(NO or YES)
11 DBPROTOCOL .....	==>	(Blank, DRDA, OR PRIVATE)
12 OPTIMIZATION HINT .....	==>	(Blank or 'hint-id')
13 DYNAMIC RULES .....	==> RUN	(RUN or BIND)
14 SQLRULES.....	==> DB2	(DB2 or STD)
15 DISCONNECT .....	==> EXPLICIT	(EXPLICIT, AUTOMATIC, or CONDITIONAL)

Рисунок 116. Панель параметров по умолчанию для связывания планов

Эта панель позволяет изменять параметры по умолчанию для опций BIND PLAN. Почти все опции на этой панели те же, что и опции для параметров по умолчанию при повторном связывании пакета. Однако для параметров по умолчанию для REBIND PLAN в любом поле можно ввести SAME, чтобы задать те же значения, что и при последнем связывании этого плана. При повторном связывании для всех полей по умолчанию устанавливается значение SAME.

**Описание полей на панели:** На панелях параметров по умолчанию для BIND PACKAGE BIND PLAN есть следующие поля:

#### 1 ISOLATION LEVEL (УРОВЕНЬ ИЗОЛЯЦИИ)

Позволяет задать, насколько изолировать прикладную программу от влияния других запущенных прикладных программ. По умолчанию, если заменяется существующий план или пакет, устанавливается то же значение, что и для старого плана или пакета.

Можно ввести RR, RS, CS или UR. Описание влияния этих величин смотрите в разделе “Опция ISOLATION” на стр. 373.

#### 2 VALIDATION TIME (ВРЕМЯ ПРОВЕРКИ)

Можно задать RUN (при выполнении) или BIND (при связывании), чтобы указать, проверять авторизацию во время выполнения или во время связывания. По умолчанию, если заменяется существующий план или пакет, устанавливается то же значение, что и для старого плана или пакета. Подробнее об этой опции смотрите в описание опции связывания VALIDATE в главе 2 справочника *DB2 Command Reference*.

#### 3 RESOURCE RELEASE TIME (ВРЕМЯ ОСВОБОЖДЕНИЯ РЕСУРСА)

Можно задать COMMIT (при принятии) или DEALLOCATE (при освобождении) для указания, когда снимать блокировку ресурсов. По умолчанию, если заменяется существующий план или пакет, устанавливается то же значение, что и для старого плана или пакета. Описание влияния этих значений смотрите в разделе “Опции ACQUIRE и RELEASE” на стр. 369.

#### 4 EXPLAIN PATH SELECTION (ВЫБОР ПУТИ ДЛЯ ОБЪЯСНЕНИЯ)

Можно задать YES или NO для получения информации объяснения о выполнении операторов SQL в пакете. По умолчанию задается NO.

Процесс связывания вставляет информацию в таблицу *владелец.PLAN\_TABLE*, где *владелец* – идентификатор авторизации владельца плана или пакета. Если вы задали *владелец.DSN\_STATEMENT\_TABLE*, в эту таблицу DB2 также вставит информацию о стоимости выполнения операторов. Если вы задали в этом поле YES, а в поле VALIDATION TIME – BIND, и если вы неправильно определили PLAN\_TABLE, связывание закончится неудачно.

Информацию об объяснении и создании PLAN\_TABLE смотрите в разделе “Получение информации PLAN\_TABLE с помощью EXPLAIN” на стр. 702.

#### 5 DATA CURRENCY (ТЕКУЩИЕ ДАННЫЕ)

Позволяет задать YES или NO, чтобы указать, требуется ли согласованность данных для неоднозначных указателей, открытых в удаленных положениях.

Данные считаются согласованными, если данные в хост–структуре идентичны данным в базовой таблице. При локальной обработке данные всегда считаются согласованными. Дополнительную информацию о согласованности данных смотрите в разделе “Поддержание согласованности данных” на стр. 429.

#### 6 PARALLEL DEGREE (СТЕПЕНЬ ПАРАЛЛЕЛИЗМА)

Можно задать либо ANY для выполнения запросов с использованием параллельной обработки (когда это возможно), либо 1, чтобы DB2 не выполняла запросы параллельно. Дополнительную информацию об этой опции смотрите в разделе “Глава 7–5. Параллельные операции и производительность запросов” на стр. 753.

#### 8 REOPTIMIZE FOR INPUT VARS (ПЕРЕОПТИМИЗАЦИЯ ДЛЯ ВХОДНЫХ ПЕРЕМЕННЫХ)

Указывает, определяет ли DB2 пути доступа во время связывания и еще раз во время выполнения для операторов, содержащих:

- Входные переменные хоста
- Маркеры параметров
- Специальные регистры

Если задать YES, DB2 будет снова определять пути доступа во время выполнения. Если для этой опции задано YES, необходимо также задать YES для DEFER PREPARE, чтобы не получать ошибки связывания.

#### 9 DEFER PREPARE (ОТЛОЖИТЬ ПОДГОТОВКУ)

Можно отложить подготовку динамических операторов SQL до тех пор, пока DB2 не встретит первый оператор OPEN, DESCRIBE или EXECUTE, который ссылается на эти операторы. Укажите YES, чтобы отложить подготовку оператора. Информацию об использовании этой опции смотрите в разделе “Используйте опции связывания, повышающие производительность” на стр. 421.

#### 10 KEEP DYN SQL PAST COMMIT (СОХРАНЯТЬ ДИНАМИЧЕСКИЕ SQL ПОСЛЕ ПРИНЯТИЯ)

Указывает, сохраняет ли DB2 динамические операторы SQL после точек принятия. YES заставляет DB2 сохранять динамические операторы SQL после точек принятия. Программа получает возможность,

выполнив оператор PREPARE для динамического оператора SQL, выполнить этот оператор еще раз после точки принятия без повторного выполнения PREPARE. Подробности смотрите в разделе “Производительность статического и динамического SQL” на стр. 545.

11 DBPROTOCOL (ПРОТОКОЛ DB2)

Указывает, будет ли DB2 использовать для выполнения операторов с трехсоставными именами, протокол DRDA или собственный протокол DB2. Подробности смотрите в разделе “Глава 5–4. Планирование доступа к распределенным данным” на стр. 405.

12 OPTIMIZATION HINT (СОВЕТЫ ПО ОПТИМИЗАЦИИ)

Указывает, хотите ли вы использовать советы по оптимизации для определения пути доступа. Задайте 'id-совета' если хотите, чтобы DB2 использовала советы по оптимизации в *владелец.PLAN\_TABLE*, где *владелец* – идентификатор авторизации владельца плана или пакета. 'id-совета' – строка длиной до 8 символов, заключенная в ограничители, которую DB2 сравнивает со значением OPTHINT в *владелец.PLAN\_TABLE* для определения строк с советами по оптимизации. Если вы задаете для 'id-совета' значение, отличное от пробела, DB2 использует советы по оптимизации только в том случае, если в поле OPTIMIZATION HINTS панели установки DSNTIP4 введено YES.

Пустое поле означает, что вы не хотите, чтобы DB2 использовала советы по оптимизации. Эта опция принимается по умолчанию. Подробности смотрите в Разделе 5 (Том 2) руководства *DB2 Administration Guide*.

13 DYNAMIC RULES (ДИНАМИЧЕСКИЕ ПРАВИЛА)

Для планов позволяет задавать, применяются ли к динамическим операторам SQL во время выполнения правила времени выполнения (RUN) или времени связывания (BIND).

Для пакетов позволяет задавать, применяются ли к динамическим операторам SQL во время выполнения правила времени выполнения (RUN) или времени связывания (BIND). Для пакета, выполняемого в среде активной пользовательской функции или хранимой процедуры, опции INVOKEBIND, INVOKERUN, DEFINEBIND и DEFINERUN указывают, у кого должны быть полномочия на выполнение динамических операторов SQL в этом пакете.

В случае пакетов устанавливаемые по умолчанию правила для пакета на локальном сервере – такие же, что и правила для плана, к которому во время выполнения присоединяется этот пакет. Для пакета на удаленном сервере по умолчанию устанавливается RUN.

Если задать для пакета правила, отличные от правил для плана, операторы SQL для этого пакета используют указанные для него правила. Если пакет, связанный с опцией DEFINEBIND или INVOKEBIND, выполняется не в среде активной пользовательской функции или хранимой процедуры, операторы SQL для этого пакета используют правила BIND. Если пакет, связанный с опцией DEFINERUN или INVOKERUN, выполняется не в среде активной пользовательской функции или хранимой процедуры, операторы SQL для этого пакета используют правила RUN.

Подробности смотрите в разделе “Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES” на стр. 457.

#### **Для пакетов:**

##### 7 SQLERROR PROCESSING (ОБРАБОТКА ОШИБОК SQL)

Можно задать CONTINUE, чтобы продолжить создание пакета после обнаружения ошибок SQL, или NOPACKAGE, чтобы прекратить создание пакета.

#### **Для планов:**

##### 7 RESOURCE ACQUISITION TIME (ВРЕМЯ БЛОКИРОВКИ РЕСУРСА)

Позволяет задать, когда начинать блокировку ресурсов. Возможные варианты:

**USE (по умолчанию)** открывать табличные пространства и устанавливать блокировки только при первом их использовании связанной с планом программой.

**ALLOCATE** открывать табличные пространства и устанавливать все блокировки при размещении плана. Это значение не оказывает влияния на динамический SQL.

Описание влияния этих значений смотрите в разделе “Опции ACQUIRE и RELEASE” на стр. 369.

##### 14 SQLRULES (ПРАВИЛА SQL)

Можно задать, выполняется ли оператор CONNECT (тип 2) в соответствии с правилами DB2 (DB2) или стандартного SQL (STD). Подробности смотрите в разделе “Задание правил SQL” на стр. 460.

##### 15 DISCONNECT (ОТСОЕДИНЕНИЕ)

Можно задать, какие из удаленных соединений заканчивать при принятии или откате. Вне зависимости от того, что задано, все соединения в состоянии отложенного освобождения завершаются при принятии.

Возможные варианты:

**EXPLICIT** – при принятии или откате разрывать только соединения в состоянии отложенного освобождения

**AUTOMATIC** – разрывать все удаленные соединения

**CONDITIONAL** – при принятии или откате разрывать удаленные соединений, у которых нет открытых указателей с условием HOLD.

Подробности об этих значениях смотрите в описании опции DISCONNECT подкоманды BIND PLAN в главе 2 справочника *DB2 Command Reference*.

## **Панель типов соединений системы**

Эта панель появляется, если ввести YES в поле ENABLE/DISABLE CONNECTIONS? на панелях BIND PACKAGE, REBIND PACKAGE, BIND PLAN или REBIND PLAN. На панелях BIND PACKAGE и REBIND PACKAGE показанная ниже опция REMOTE не появляется.

```

DSNEBP13      SYSTEM CONNECTION TYPES FOR BIND ...
SSID: DSN
COMMAND ===>

Select system connection types to be Enabled/Disabled:

1  ENABLE ALL CONNECTION TYPES? ===>    (* to enable all types)
or
2  ENABLE/DISABLE SPECIFIC CONNECTION TYPES ===>  (E/D)

BATCH ..... ==>   (Y/N)          SPECIFY CONNECTION NAMES?
DB2CALL ..... ==>  (Y/N)
RRSAF ..... ==>  (Y/N)
CICS ..... ==>  (Y/N)          ===> N  (Y/N)
IMS ..... ==>  (Y/N)
DLIBATCH ..... ==>  (Y/N)          ===> N  (Y/N)
IMSBMP ..... ==>  (Y/N)          ===> N  (Y/N)
IMSMPP ..... ==>  (Y/N)          ===> N  (Y/N)
REMOTE ..... ==>  (Y/N)          ===> N  (Y/N)

```

Рисунок 117. Панель типов соединений системы

Чтобы разрешить или запретить отдельные виды соединений (т. е. разрешить или запретить соединению запускать конкретный пакет или план), введите приводящуюся ниже информацию.

**1 ENABLE ALL CONNECTION TYPES? (РАЗРЕШИТЬ ВСЕ ВИДЫ СОЕДИНЕНИЙ?)**

Можно ввести звездочку (\*), чтобы разрешить все соединения. После этого остальная часть панели игнорируется.

**2 ENABLE/DISABLE SPECIFIC CONNECTION TYPES (РАЗРЕШИТЬ/ЗАПРЕТИТЬ ОТДЕЛЬНЫЕ ВИДЫ СОЕДИНЕНИЙ)**

Позволяет задать список видов разрешенных или запрещенных соединений; в одной и той же операции нельзя разрешать одни виды и запрещать другие. Если вы перечисляете виды разрешенных соединений, введите E; это запретит все другие виды соединений. Если вы перечисляете виды запрещенных соединений, введите D; это разрешит все другие виды соединений. Подробнее об этой опции смотрите в описании опций связывания ENABLE и DISABLE в главе 2 справочника *DB2 Command Reference*.

Для каждого из перечисленных ниже видов соединений введите Y (да), если оно есть вашем списке, и N (нет), если его там нет. Виды соединений:

- **BATCH** для соединения TSO
- **DB2CALL** для соединения CAF
- **RRSAF** для соединения RRSAF
- **CICS** для соединения CICS
- **IMS** для всех соединений IMS: DLIBATCH, IMSBMP и IMSMPP
- **DLIBATCH** для соединения DL/I Batch Support Facility
- **IMSBMP** для соединения IMS с регионом BMP
- **IMSMPP** для соединения IMS с регионом MPP или IFP
- **REMOTE** для удаленных имен положений и имен LU

Для каждого из видов соединений, у которого показана вторая стрелка, можно ввести Y под пунктом SPECIFY CONNECTION NAMES? (задать имена соединений), чтобы перечислить имена конкретных соединений данного вида. Если это не требуется, оставьте по умолчанию N. При вводе в любое из этих полей Y появится другая панель, где можно

ввести имена соединений. Подробности смотрите в разделе “Панели для ввода списков значений” на стр. 502.

Если в TSO на этой панели воспользоваться командой DISPLAY, можно посмотреть “разрешенные” и “запрещенные” соединения в библиотеке ISPF DSNSPFT (компонент DSNCONNS). Эта информация не отражает текущее состояние каталога DB2.

Если в командной строке ввести DISPLAY ENABLED, вы получите имена соединений, разрешенных в настоящее время для ваших соединений TSO. Например:

```
Display OF ALL connection name(s) to be ENABLED
```

CONNECTION	SUBSYSTEM
CICS1	ENABLED
CICS2	ENABLED
CICS3	ENABLED
CICS4	ENABLED
DLI1	ENABLED
DLI2	ENABLED
DLI3	ENABLED
DLI4	ENABLED
DLI5	ENABLED

## Панели для ввода списков значений

Некоторые поля на панелях DB2I связаны с ключевыми словами команд, принимающими несколько значений. Эти поля приводят к панелям списков, позволяющим вводить или изменять неограниченное количество значений. Панель списка выглядит, как сеанс редактирования ISPF и допускает прокрутку и использование некоторого ограниченного набора команд.

У всех панелей списков разные форматы, зависящие от содержания и назначения конкретной панели. На рис. 118 приведен обобщенный пример панели списка:

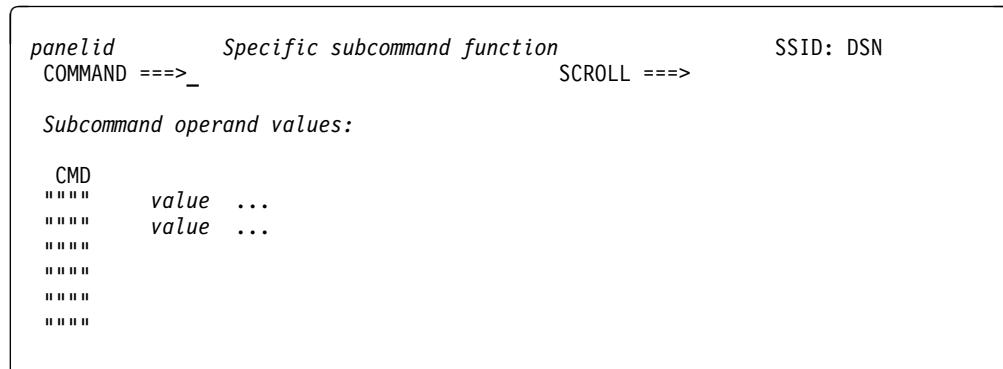


Рисунок 118. Общий пример панели списка DB2I

Синтаксис для задания имен на панели списка смотрите в главе 2 справочника *DB2 Command Reference*, где описаны требуемые типы имен.

Все панели списков позволяют вводить ограниченное число команд в двух местах:

- В системной командной строке – с префиксом ====>
- В специальной командной зоне – с идентификатором "===="

В системной командной строке можно использовать следующие команды:

**END** Сохранить введенных переменных, выйти из таблицы и продолжить обработку.

**CANCEL** Отбросить всех введенные переменные, прекратить обработку и вернуться к предыдущей панели.

**SAVE** Сохранить все введенные переменные без выхода из таблицы.

В специальной командной зоне можно следующие команды:

**I<sub>nn</sub>** Вставить *пп* строк после текущей.

**D<sub>nn</sub>** Удалить *пп* строк, начиная с текущей.

**R<sub>пп</sub>** Повторить текущую строку *пп* раз.

*пп* по умолчанию устанавливается равным 1.

После окончания работы с панелью списка введите END, чтобы сохранить введенные значения и продолжить обработку.

## Панель подготовки программ: Compile, Link и Run

Вторая из панелей подготовки программ (рис. 119) позволяет выполнить последние два этапа процесса подготовки программ (компиляцию и компоновку–редактирование), а также макрообработку PL/I для программ, где это необходимо. Для программ TSO эта панель также позволяет запускать программы.

```
DSNEPP02      PROGRAM PREP: COMPILE, PRELINK, LINK, AND RUN      SSID: DSN
COMMAND ====_

Enter compiler or assembler options:
 1 INCLUDE LIBRARY ===> SRCLIB.DATA
 2 INCLUDE LIBRARY ===>
 3 OPTIONS ..... ===> NUM, OPTIMIZE, ADV

Enter linkage editor options:
 4 INCLUDE LIBRARY ===> SAMPLIB.COBOL
 5 INCLUDE LIBRARY ===>
 6 INCLUDE LIBRARY ===>
 7 LOAD LIBRARY .. ===> RUNLIB.LOAD
 8 PRELINK OPTIONS ===>
 9 LINK OPTIONS... ===>

Enter run options:
10 PARAMETERS .... ===> D01, D02, D03/
11 SYSIN DATA SET ===> TERM
12 SYSPRINT DS ... ===> TERM
```

Рисунок 119. Панель подготовки программы: COMPILE, LINK, AND RUN

### 1.2 INCLUDE LIBRARY (ВЫЗЫВАЕМАЯ БИБЛИОТЕКА)

Позволяет задать до двух библиотек, содержащих компоненты для включения компилятором по команде INCLUDE. Эти компоненты могут также быть выводом DCLGEN. При желании эти поля можно оставить пустыми. Значение по умолчанию отсутствует.

### 3 OPTIONS (ОПЦИИ)

Позволяет задавать опции компилятора, ассемблера и макропроцессора PL/I. Можно также ввести список опций компилятора или ассемблера, разделяя записи запятыми, пробелами или и тем, и другим. При желании эти поля можно оставить пустыми. Значение по умолчанию отсутствует.

### 4,5,6 INCLUDE LIBRARY (ВЫЗЫВАЕМАЯ БИБЛИОТЕКА)

Позволяет вводить до трех имен библиотек, содержащих компоненты для включения редактором связей. При желании эти поля можно оставить пустыми. Значение по умолчанию отсутствует.

### 7 LOAD LIBRARY (ЗАГРУЗОЧНАЯ БИБЛИОТЕКА)

Позволяет задать имя библиотеки, где будет находиться загрузочный модуль. По умолчанию устанавливается RUNLIB.LOAD.

Если указанная загрузочная библиотека — секционированный набор данных и входной набор данных тоже секционированный, имя компонента, указанное в поле INPUT DATA SET NAME на панели подготовки программ, будет именем загрузочного модуля. Если входной набор данных последовательный, именем загрузочного модуля будет второй спецификатор входного набора данных.

Эти поля необходимо заполнить, если на панели подготовки программ вы задали LINK или RUN.

### 8 PRELINK OPTIONS (ОПЦИИ ПРЕКОМПОНОВКИ)

Позволяет вводить список опций прекомпоновщика. Элементы списков разделяются запятыми, пробелами или и тем, и другим. При желании эти поля можно оставить пустыми. Значение по умолчанию отсутствует.

Утилита прекомпоновки применяется только к программам на C, C++ и IBM COBOL for MVS & VM. Дополнительную информацию об опциях прекомпоновки смотрите в руководстве *OS/390 Language Environment for OS/390 & VM Programming Guide*.

### 9 LINK OPTIONS (ОПЦИИ КОМПОНОВКИ)

Позволяет вводить список опций компоновки—редактирования. Элементы списков разделяются запятыми, пробелами или и тем, и другим.

Для подготовки программы, использующей 31-битную адресацию и работающей над 16-мегабайтной линией, укажите следующие опции компоновки—редактирования: AMODE=31, RMODE=ANY.

### 10 PARAMETERS (ПАРАМЕТРЫ)

Позволяет задать список параметров, которые вы хотите передать либо процессору времени выполнения базового языка, либо прикладной программе. Элементы списков разделяются запятыми, пробелами или и тем, и другим. Можно оставить это поле пустым.

Если подготавливается программа IMS или CICS, это поле следует оставить пустым; DB2I нельзя использовать для запуска программ IMS и CICS.

Чтобы отделить опции для процессора времени выполнения от опций для программы, используйте косую черту (/).

- Для языков PL/I и FORTRAN параметры для процессора времени выполнения должны быть слева от косой черты, а параметры прикладной программы — справа.  
парам. процессора времени выполнения / парам. прикл. программы

- Для языка COBOL используется обратный порядок. Параметры процессора времени выполнения должны быть справа от косой черты, а параметры прикладной программы – слева.
- Для ассемблера и С среда выполнения не поддерживается, и необходимость использования косой черты для передачи параметров прикладной программе отсутствует.

#### 11 SYSIN DATA SET (НАБОР ДАННЫХ SYSIN)

Позволяет задать для прикладной программы имя набора данных SYSIN (или FT05F001 для языка FORTRAN), если в этом есть необходимость.

Если не заключать имя набора данных в апострофы, к нему будет добавлен стандартные префикс (идентификатор пользователя) и суффикс TSO. По умолчанию для этого поля устанавливается TERM.

Если подготавливается программа IMS или CICS, это поле следует оставить пустым; DB2I нельзя использовать для запуска программ IMS и CICS.

#### 12 SYSPRINT DS

Позволяет задать для прикладной программы имена набора данных SYSPRINT (или FT06F001 для языка FORTRAN), если в этом есть необходимость. Если не заключать имя набора данных в апострофы, к нему будет добавлен стандартные префикс (идентификатор пользователя) и суффикс TSO. По умолчанию для этого поля устанавливается TERM.

Если подготавливается программа IMS или CICS, это поле следует оставить пустым; DB2I нельзя использовать для запуска программ IMS и CICS.

Возможно, вашей прикладной программе потребуются другие наборы данных, кроме SYSIN и SYSPRINT. В таком случае не забудьте внести их в каталог и разместить их до запуска программы.

После ввода значений на этой панели и нажатия ENTER DB2 проводит компиляцию и компоновку–редактирование прикладной программы. Если на панели DB2 PROGRAM PREPARATION вы указали, что хотите запустить прикладную программу, DB2 выполнит и саму программу.



## Глава 6–2. Тестирование прикладной программы

В этом разделе обсуждаются: настройка среды тестирования, тестирование операторов SQL, отладка программ и чтение выходных данных прокомпилятора.

### Создание среды тестирования

В этом разделе описывается создание тестовой структуры данных и заполнение таблиц тестовыми данными.

#### CICS

Перед тем как выполнять прикладную программу, убедитесь, что ее выполнение разрешено в соответствующих записях в управляемых областях RCT, SNT и RACF. Эти функции находятся в ведении системного администратора; дополнительную информацию о них смотрите в Разделе 3 (Том 1) руководства *DB2 Administration Guide*.

Кроме того, убедитесь, что заданный для программы код транзакций определен для CICS CSD.

### Создание тестовой структуры данных

При тестировании прикладной программы, обращающейся к данным DB2, необходимо, чтобы данные DB2 были доступны для тестирования. Для этого можно создать тестовые таблицы и производные таблицы.

**Производные таблицы для существующих таблиц.** Если прикладная программа не изменяет набор данных DB2 и одна или несколько таблиц для этого продукта уже содержат данные, можно использовать производные таблицы, созданные на основе этих существующих таблиц.

**Тестовые таблицы.** Чтобы создать тестовую таблицу, требуется база данных и табличное пространство. Обратитесь к администратору базы данных и убедитесь, что база данных и табличные пространства доступны для вас.

Если данные, которые нужно изменять, уже существуют в некоторой таблице, можно использовать условие LIKE оператора CREATE TABLE. Если нужно, чтобы другие пользователи также обладали правами на использование таблицы для тестирования, можно задать вторичный ID в качестве владельца таблицы. Чтобы сделать это, можно использовать оператор SET CURRENT SQLID; подробное описание смотрите в Главе 6 руководства *DB2 SQL Reference*. Дополнительную информацию об ID авторизации смотрите в Разделе 3 (Том 1) руководства *DB2 Administration Guide*.

Если для тестирования используется отдельная система DB2, можно создать тестовые таблицы и производные таблицы в этой тестовой системе, а затем тщательно протестировать свою программу в этой системе. В этой главе подразумевается, что все операции тестирования проводятся в отдельной системе и что пользователь, создавший тестовые таблицы и производные

таблицы, обладает ID авторизации для операции TEST. Эти таблицы называются TEST.EMP, TEST.PROJ и TEST.DEPT.

## **Анализ данных, используемых прикладной программой**

Чтобы создать тестовые таблицы и производные таблицы, сначала проанализируйте данные, используемые прикладной программой.

- Перечислите данные, к которым обращается прикладная программа, и опишите, как она обращается к каждому элементу данных. Например, предположим, что тестируется прикладная программа, которая обращается к таблицам DSN8610.EMP, DSN8610.DEPT и DSN8610.PROJ. Информацию об этих данных можно записать, как показано в Табл. 51.

*Таблица 51. Описание данных, используемых прикладной программой*

Имя таблицы или производной таблицы	Вставляет строки?	Удаляет строки?	Имя столбца	Тип данных	Изменяет данные?
DSN8610.EMP	Нет	Нет	EMPNO LASTNAME WORKDEPT PHONENO JOB	CHAR(6) VARCHAR(15) CHAR(3) CHAR(4) DECIMAL(3)	Да Да Да
DSN8610.DEPT	Нет	Нет	DEPTNO MGRNO	CHAR(3) CHAR (6)	
DSN8610.PROJ	Да	Да	PROJNO DEPTNO RESPEMP PRSTAFF PRSTDAT PRENDATE	CHAR(6) CHAR(3) CHAR(6) DECIMAL(5,2) DECIMAL(6) DECIMAL(6)	Да Да Да Да Да

- Определите тестовые таблицы и производные таблицы, которые нужны для тестирования прикладной программы.

Создайте тестовую таблицу, если:

- Прикладная программа изменяет данные в этой таблице
- Нужно создать производную таблицу на основе тестовой таблицы, так как прикладная программа изменяет данные в этой производной таблице.

Для нашего примера создайте следующие тестовые таблицы:

- TEST.EMP со следующим форматом:

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOB
:	:	:	:	:

- TEST.PROJ. с теми же столбцами и тем же форматом, что и таблица DSN8610.PROJ, поскольку прикладная программа вставляет строки в таблицу DSN8610.PROJ.

Для нашего примера создайте тестовую производную таблицу для таблицы DSN8610.DEPT. Поскольку прикладная программа не изменяет данные в таблице DSN8610.DEPT, можно создать производную таблицу на основе самой этой таблицы (а не на основе отдельной тестовой таблицы). Однако безопаснее создать полный набор тестовых таблиц и использовать для

тестирования программы только тестовые данные. Производная таблица TEST.DEPT имеет следующий формат:

DEPTNO	MGRNO
:	:

### Получение полномочий

Перед тем как можно будет создавать таблицы, нужно получить полномочия на создание таблиц и использование табличного пространства, в котором должны располагаться эти таблицы. Необходимо также обладать полномочиями на связывание и выполнение тестируемых программ. Необходимые полномочия на создание таблиц и обращение к таблицам, а также на связывание и выполнение программ предоставляются администратором базы данных.

Если предполагается использование существующих таблиц и производных таблиц (или прямо, или в качестве основы для производных таблиц), необходимо иметь права на доступ к этим таблицам и производным таблицам. Эти права предоставляет администратор базы данных.

Чтобы создать производную таблицу, необходимо обладать полномочиями для всех таблиц и производных таблиц, на основе которых создается данная производная таблица. После создания производной таблицы вы будете обладать для нее теми же полномочиями, что и для таблиц и производных таблиц, на основе которых она создана. Перед тем как пытаться выполнить эти примеры, получите от администратора базы данных права на создание новых таблиц и производных таблиц и права на доступ к существующим таблицам. Узнайте у администратора базы данных имена таблиц и производных таблиц, для которых у вас есть права на доступ (а также конкретные права, которыми вы обладаете для каждой таблицы). Дополнительную информацию о создании таблиц и производных таблиц смотрите в разделе “Глава 2–2. Работа с таблицами и изменение данных” на стр. 57.

### Создание полной тестовой структуры

Ниже показаны операторы SQL, которые можно использовать для создания полной тестовой структуры, содержащей небольшую таблицу с именем SPUFINUM. Тестовая структура состоит из:

- Группы хранения с именем SPUFISG
- Базы данных с именем SPUFIDB
- Табличного пространства SPUFITS в SPUFIDB и использующего SPUFISG
- Таблицы с именем SPUFINUM в табличном пространстве SPUFITS

```

CREATE STOGROUP SPUFISG
  VOLUMES (номер-тота-пользователя)
  VCAT DSNCAT ;

CREATE DATABASE SPUFIDB ;

CREATE TABLESPACE SPUFITS
  IN SPUFIDB
  USING STOGROUP SPUFISG ;

CREATE TABLE SPUFINUM
  ( XVAL CHAR(12) NOT NULL,
    ISFLOAT FLOAT,
    DEC30 DECIMAL(3,0),
    DEC31 DECIMAL(3,1),
    DEC32 DECIMAL(3,2),
    DEC33 DECIMAL(3,3),
    DEC10 DECIMAL(1,0),
    DEC11 DECIMAL(1,1),
    DEC150 DECIMAL(15,0),
    DEC151 DECIMAL(15,1),
    DEC1515 DECIMAL(15,15) )
  IN SPUFIDB.SPUFITS ;

```

Подробное описание каждого оператора CREATE смотрите в руководстве *DB2 SQL Reference* или в Разделе 2 (Том 1) руководства *DB2 Administration Guide*.

## **Заполнение таблиц тестовыми данными**

Существует несколько способов, которые можно использовать для внесения в таблицу тестовых данных:

- **INSERT ... VALUES** (оператор SQL) – при каждом выполнении этого оператора в таблицу помещается одна строка. Информацию об операторе **INSERT** смотрите в разделе “Вставка строки: **INSERT**” на стр. 66.
- **INSERT ... SELECT** (оператор SQL) – такой оператор получает данные из существующей таблицы (на основе условия **SELECT**) и помещает их в таблицу, заданную в операторе **INSERT**. Информацию об этом методе смотрите в разделе “Заполнение таблицы значениями из другой таблицы: **INSERT** для нескольких строк” на стр. 69.
- Утилита **LOAD** получает данные из последовательного файла (не из файла DB2), форматирует эти данные и помещает их в таблицу. Дополнительную информацию об утилите **LOAD** смотрите в руководстве *DB2 Utility Guide and Reference*.
- Пример программы **DB2 UNLOAD** (DSNTIAUL) может выгрузить данные из таблицы или производной таблицы и создать операторы управления загрузкой, помогающие произвести эту операцию. Дополнительную информацию о примере программе **UNLOAD** смотрите в разделе 2 руководства *DB2 Installation Guide*.

---

## Тестирование операторов SQL при помощи SPUFI

Для тестирования операторов SQL в среде TSO/ISPF можно использовать SPUFI (интерфейс между ISPF и DB2). Используя панели SPUFI, можно ввести в набор данных операторы SQL, которые DB2 затем выполнит. Главная панель SPUFI содержит несколько функций, позволяющих:

- Задать имя входного набора данных, содержащего операторы SQL, которые передаются для выполнения системе DB2
- Задать имя выходного набора данных, содержащего результаты выполнения операторов SQL
- Задать опции операций SPUFI.

Операторы SQL, выполняющиеся под SPUFI, работают с реальными таблицами (в данном случае с таблицами, созданными для тестирования). Следовательно, перед обращением к данным DB2:

- Убедитесь, что существуют все таблицы и производные таблицы, используемые в операторах SQL
- Если таблицы или производные таблицы не существуют, создайте их (или пусть их создаст администратор базы данных). Можно использовать SPUFI для ввода операторов CREATE, которые создают таблицы и производные таблицы, необходимые для тестирования.

Более подробную информацию об использовании SPUFI смотрите в разделе “Глава 2–5. Исполнение SQL с вашего терминала при помощи SPUFI” на стр. 93.

---

## Отладка программы

Для многих систем существуют особые рекомендации, относящиеся к случаям аварийного завершения программы. Ниже представлены некоторые общие рекомендации такого рода.

### Отладка программ в TSO

Полученная при тестировании информация об ошибках помогает обнаружить и исправить ошибки в программе. Может быть полезна следующая информация:

- Имя плана для этой прикладной программы
- Обрабатываемые входные данные
- Ошибочный оператор SQL и его функция
- Содержимое SQLCA (область связи SQL) или, если программа использует динамические операторы SQL, SQLDA (область дескрипторов SQL)
- Текущие дата и время
- Код аварийного завершения и сообщения об ошибках.

Если программа обнаруживает ошибку, которая не приводит к аварийному завершению программы, она может передать всю необходимую информацию об ошибке стандартной процедуре обработки ошибок. Программы, работающие в диалоговом режиме, могут также послать сообщение об ошибке на терминал.

## **Языковые средства тестирования**

Информацию о средствах тестирования компилятора или ассемблера смотрите в руководствах для компилятора или CODE/370. Руководства для компиляторов содержат информацию о подходящих для используемого языка отладчиках.

## **Команда TSO TEST**

Команда TSO TEST особенно удобна для отладки программ на ассемблере.

Следующий пример – командная процедура (CLIST), которая выполняет под TSO TEST прикладную программу DB2 с именем MYPROG и задает точку остановки на входе в эту программу. Имя подсистемы DB2 в этом примере – DB4.

```
PROC 0
TEST 'prefix.SDSNLOAD(DSN)' CP
DSN SYSTEM(DB4)
AT MYPROG.MYPROG.+0 DEFER
GO
RUN PROGRAM(MYPROG) LIBRARY('L186331.RUNLIB.LOAD(MYPROG)')
```

Дополнительную информацию о команде TEST смотрите в руководстве *OS/390 TSO/E Command Reference*.

Другое средство для отладки задач – диалоговое средство тестирования ISPF.

## **Отладка программ в IMS**

Полученная при тестировании информация об ошибках помогает обнаружить и исправить ошибки в программе. Может быть полезна следующая информация:

- Имя плана для этой прикладной программы
- Обрабатываемые входные данные
- Имя исходного логического терминала
- Ошибочный оператор SQL и его функция
- Содержимое SQLCA (область связи SQL) или, если программа использует динамические операторы SQL, SQLDA (область дескрипторов SQL)
- Текущие дата и время
- Имя PSB программы
- Код транзакции, выполнявшейся программой
- Функция вызова (т.е. имя функции DL/I)
- Содержимое PCB, на который ссылается вызов этой программы
- Если выполнялся вызов базы данных DL/I, SSA, если они использовались при вызове.
- Код аварийного завершения, код причины аварийного завершения и сообщения об ошибках.

Если программа обнаруживает ошибку, она может передать всю необходимую информацию об ошибке стандартной процедуре обработки ошибок. Программы, работающие в диалоговом режиме, могут также послать сообщение об ошибке на исходный логический терминал.

Интерактивные программы могут также послать сообщение оператору главного терминала, сообщив ему информацию о завершении программы. Чтобы сделать это, программа помещает имя логического терминала для главного терминала в специальный PCB и выполняет один или несколько вызовов ISRT.

В некоторых системах в конце дня выполняется программа BMP, создающая список всех ошибок, возникших в течение дня. Если используемая система выполняет такие действия, можно послать сообщение, используя специальный PCB, в котором в качестве назначения задана эта программа BMP.

**Пакетный симулятор терминала (BTS):** Пакетный симулятор терминала (BTS) позволяет тестировать прикладные программы IMS. BTS производит трассировку вызовов прикладных программ DL/I и операторов SQL и симулирует функции передачи данных. Для оператора терминала такой терминал TSO может выглядеть как терминал IMS, позволяя конечному пользователю взаимодействовать с прикладной программой, как если бы она работала в диалоговом режиме. Пользователь может использовать любые прикладные программы, которые ему разрешено выполнять, для обращения к любым базам данных (как DL/I, так и DB2), доступ к которым ему разрешен. Для обращения к базам данных DB2 необходимо, чтобы BTS выполняются в режиме пакетной BMP или BMP TSO. Дополнительную информацию о пакетном симуляторе терминаласмотрите в руководстве *IMS Batch Terminal Simulator General Information*.

## Отладка программ в CICS

Полученная при тестировании информация об ошибках помогает обнаружить и исправить ошибки в программе. Может быть полезна следующая информация:

- Имя плана для этой прикладной программы
- Обрабатываемые входные данные
- ID исходного логического терминала
- Ошибочный оператор SQL и его функция
- Содержимое SQLCA (область связи SQL) или, если программа использует динамические операторы SQL, SQLDA (область дескрипторов SQL)
- Текущие дата и время
- Особые данные для CICS, которые следует записать
- Код аварийного завершения и сообщения об ошибках
- Дамп транзакций, если таковой создается.

Используя средства CICS, можно вывести на печать информацию об ошибках; можно также напечатать содержимое SQLCA (и SQLDA).

## Средства отладки для CICS

В CICS есть следующие средства для тестирования, отслеживания событий и отладки прикладной программы:

**Средство диагностики выполнения (уровень команд) (EDF).** EDF показывает команды CICS для всех выпусков CICS. Дополнительную информацию смотрите в разделе “Утилита диагностики выполнения CICS” на стр. 514. Если используется более ранняя версия CICS, на экране

CALL TO RESOURCE MANAGER DSNCSQL (Вызов менеджера ресурсов DSNCSQL) выводится информация о состоянии "ABOUT TO EXECUTE" или "COMMAND EXECUTION COMPLETE" ("Информация о выполнении" или "Выполнение команды завершено").

**Восстановление при аварийном завершении.** Команду HANDLE ABEND можно использовать для обработки условий аварийного завершения, а команду ABEND – чтобы вызвать аварийное завершение задачи.

**Утилита трассировки.** Таблица трассировки может содержать записи с информацией о выполнении различных команд CICS и операторов SQL, а также записи, сгенерированные прикладными программами; они могут быть записаны в основную память, а также на дополнительное устройство хранения.

**Дамп.** Можно задать области в основной памяти для сохранения дампа в последовательный набор данных, на ленту или диск, для последующего форматирования и печати при помощи утилиты CICS.

**Журналы.** Для сбора статистической информации или отслеживания работы программ средства системы могут создавать записи в специальных наборах данных, называемых журналами. Один из них – системный журнал.

**Восстановление.** В случае аварийного завершения программы система CICS восстанавливает некоторые ресурсы, возвращая их в исходное состояние, после чего оператор может легко вызвать повторный запуск транзакции. Команду SYNCPOINT можно использовать для выделения в программе частей, чтобы повторно выполнять только невыполненную часть транзакции.

Более подробную информацию по этим темамсмотрите в руководстве *CICS for MVS/ESA Application Programming Reference*.

### **Утилита диагностики выполнения CICS**

Утилита диагностики выполнения (EDF) системы CICS выполняет трассировку операторов SQL в режиме интерактивной отладки, позволяя разработчикам прикладных программ тестировать и отлаживать программы в диалоговом режиме, не изменяя программу и процедуру подготовки программы.

EDF прерывает выполнение прикладной программы в различных точках и выводит полезную информацию о типе оператора, входных и выходных переменных и всех условиях ошибок после выполнения оператора. Она также выводит все экраны, посылаемые прикладной программой, позволяя в процессе тестирования использовать тот же диалог с программой, что и при последующем использовании готовой программы.

При выполнении задачи в режиме EDF эта утилита выводит существенную информацию перед выполнением и после выполнения оператора SQL. Это важное вспомогательное средство при отладке транзакций CICS для программ, содержащих операторы SQL. Выводимая EDF информация SQL полезна при отладке программ и для анализа ошибок после возникновения ошибок или предупреждений SQL. При использовании этой утилиты уменьшается объем работы, необходимый для написания специальных обработчиков ошибок.

```

TRANSACTION: XC05 PROGRAM: TESTC05 TASK NUMBER: 0000668 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL INSERT
DBRM=TESTC05, STMT=00368, SECT=00004
IVAR 001: TYPE=CHAR, LEN=00007, IND=000 AT X'03C92810'
  DATA=X'F0F0F9F4F3F4F2'
IVAR 002: TYPE=CHAR, LEN=00007, IND=000 AT X'03C92817'
  DATA=X'F0F1F3F3F7F5F1'
IVAR 003: TYPE=CHAR, LEN=00004, IND=000 AT X'03C9281E'
  DATA=X'E7C3F0F5'
IVAR 004: TYPE=CHAR, LEN=00040, IND=000 AT X'03C92822'
  DATA=X'E3C5E2E3C3F0F540E2C9D47D3C540C4C2F240C9D5E2C5D9E3404040'...
IVAR 005: TYPE=SMALLINT, LEN=00002, IND=000 AT X'03C9284A'
  DATA=X'0001'

OFFSET:X'001ECE' LINE:UNKNOWN EIBFN=X'1002'

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

```

Рисунок 120. Экран EDF перед выполнением оператора SQL DB2

**EDF перед выполнением оператора:** На рис. 120 на стр. 515 показан пример экрана EDF перед выполнением оператора SQL. Имена ключевых полей в этой панели выделены **жирным шрифтом**.

Информация SQL DB2 на этом экране:

- **EXEC SQL тип оператора**

Это тип оператора SQL, который должен быть выполнен. Это может быть любой правильный оператор SQL, например, COMMIT, DROP TABLE, EXPLAIN, FETCH или OPEN.

- **DBRM=имя dbrm**

Имя модуля требований базы данных (DBRM), обрабатываемого в настоящее время. DBRM, созданный прекомпилятором DB2, содержит информацию об операторе SQL.

- **STMT=номер оператора**

Это номер оператора, генерируемый прекомпилятором DB2. Номер оператора указывается в тексте исходной программы и в сообщениях об ошибках прекомпилятора; его можно использовать, чтобы определить, какой именно оператор выполняется. Это номер строки исходной программы, содержащей операторы на языке хоста. Если номер оператора больше 32767, вместо него выводится 0.

- **SECT=номер раздела**

Номер раздела плана, используемого оператором SQL.

**Операторы SQL, содержащие входные переменные хоста:** Раздел IVAR (входные переменные хоста) и соответствующие ему поля появляются, только если выполняемый оператор содержит входные переменные хоста.

Этот раздел переменных хоста включает в себя переменные из предикатов, значения, используемые для вставки или обновления, и текст подготавливаемых динамических операторов SQL. Адрес входной переменной – AT '*nnnnnnnn*'.

Дополнительная информация о переменных хоста:

- **TYPE=тип данных**

Задает тип данных для этой переменной хоста. Основные типы данных: символьная строка, графическая строка, двоичное целое, число с плавающей точкой, десятичный, дата, время и отметка времени.

Дополнительную информациюсмотрите в разделе “Типы данных” на стр. 20.

- **LEN=длина**

Длина переменной хоста.

- **IND=номер переменной–индикатора**

Представляет переменную–индикатор, связанную с данной переменной хоста. Значение 0 означает, что переменная–индикатор отсутствует. Если выбранный столбец имеет пустое значение, DB2 помещает в переменную–индикатор для этой переменной хоста отрицательное значение. Дополнительную информациюсмотрите в разделе “Использование индикаторных переменных с переменными хоста” на стр. 113.

- **DATA=данные переменной хоста**

Данные, связанные с этой переменной хоста и выводимые в шестнадцатеричном формате. Если данные имеют слишком большую длину и не могут быть выведены в одной строке, в правой части строки выводятся три точки (...), которые указывают на наличие дополнительных данных.

**EDF после выполнения оператора:** На рис. 121 на стр. 517 показан пример первого экрана EDF, выводимого после выполнения оператора SQL. Имена ключевых полей в этой панели выделены **жирным шрифтом**.

```

TRANSACTION: XC05 PROGRAM: TESTC05      TASK NUMBER: 0000698   DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL FETCH          P.AUTH=SYSADM , S.AUTH=
PLAN=TESTC05, DBRM=TESTC05, STMT=00346, SECT=00001
SQL COMMUNICATION AREA:
SQLCABC    = 136           AT X'03C92789'
SQLCODE     = 000           AT X'03C9278D'
SQLERRML    = 000           AT X'03C92791'
SQLERRMC    = ''            AT X'03C92793'
SQLERRP     = 'DSN'          AT X'03C927D9'
SQLERRD(1-6) = 000, 000, 00000, -1, 00000, 000  AT X'03C927E1'
SQLWARN(0-A) = ''            AT X'03C927F9'
SQLSTATE    = 00000          AT X'03C92804'
+ OVAR 001: TYPE=INTEGER,      LEN=00004,     IND=000  AT X'03C920A0'
          DATA=X'00000001'
OFFSET:X'001D14'   LINE:UNKNOWN        EIBFN=X'1802'

ENTER:  CONTINUE
PF1 : UNDEFINED      PF2 : UNDEFINED      PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK     PF8 : SCROLL FORWARD  PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED     PF12: ABEND USER TASK

```

Рисунок 121. Экран EDF после выполнения оператора SQL DB2

Информация SQL DB2 в этом экране:

- P.AUTH=*первичный ID авторизации*  
Первичный ID авторизации DB2.
- S.AUTH=*вторичный ID авторизации*

Если список RACF опций группы не активен, DB2 использует в качестве вторичного ID авторизации имя группы соединения, используемое утилитой подключений CICS. Если список RACF опций группы активен, DB2 игнорирует имя группы соединения, используемое утилитой подключений CICS, и использует вместо него значение из списка вторичных ID авторизации DB2.

- PLAN=*имя плана*

Имя выполняемого в настоящее время плана. План представляет собой управляющую структуру, созданную в процессе связывания и используемую системой DB2 для обработки операторов SQL при выполнении прикладной программы.

- Область связи SQL (SQLCA)

В случае возникновения ошибок SQLCA содержит информацию о них. Эта информация доступна после возврата из DB2. DB2 использует SQLCA, чтобы передать прикладной программе информацию о выполнении операторов SQL.

Знак плюс (+) в левой части экрана указывает, что можно увидеть дополнительную выходную информацию EDF, используя клавиши PF для прокрутки экрана вперед и назад.

Раздел OVAR (выходные переменные хоста) и соответствующие ему поля появляются, только если выполняемый оператор возвращает выходные переменные хоста.

На рис. 122 показана оставшаяся часть выходных данных EDF для данного примера.

Рисунок 122. Экран EDF после выполнения оператора SQL DB2, продолжение

Средство подключения автоматически выводит информацию SQL при работе в режиме EDF. (Можно запустить EDF, как описано в соответствующем руководстве разработчика прикладных программ CICS.) Если это не происходит, обратитесь к тому, кто устанавливал систему, и посмотрите Раздел 2 руководства *DB2 Installation Guide*.

## Поиск источника ошибки

Если программа не работает правильно, необходимо определить источник ошибки. Если DB2 не отменила план этой прикладной программы, следует проверить следующее:

- Выходную информацию препроцессора, которая содержит информацию об ошибках и предупреждениях. Убедитесь, что исправлены все ошибки и предупреждения.
  - Выходную информацию компилятора или ассемблера. Убедитесь, что исправлены все ошибки.
  - Выходную информацию компоновщика.
    - Были ли определены все внешние ссылки?
    - Были ли включены все необходимые модули в правильном порядке?

- Был ли включен правильный модуль языкового интерфейса?  
Правильный модуль языкового интерфейса:
  - DSNELI для TSO
  - DFSLI000 для IMS
  - DSNCLI для CICS
  - DSNALI для средства подключений вызовов.
- Задана ли правильная точка входа в программу?
- Выходную информацию процесса связывания.
  - Были ли исправлены все ошибки?
  - Задано ли имя плана? Если нет, в процессе связывания подразумевается, что для диагностики должен использоваться процесс DBRM, но не должен создаваться план прикладной программы.
  - Заданы ли в одном плане прикладной программе все DBRM и пакеты, связанные с программами, входящими в данную прикладную программу, а также имена их распределенных наборов данных (PDS)?
- Правильное ли задание JCL вы использовали?

#### **IMS**

- Если используется IMS, включен ли оператор опций DL/I в правильном формате?

- Включен ли в оператор EXEC параметр размера региона? Задан ли достаточный размер региона памяти для интерфейса DB2, системы TSO, IMS или CICS и вашей программы?
- Включены ли имена всех необходимых для программы наборов данных (DB2 и других)?
- Вашу программу.

Кроме того, для поиска ошибки в программе можно использовать дампы. Например, одна из самых общих ситуаций ошибки возникает, когда при выполнении программы выдается сообщение о ее аварийном завершении. В таком случае тестовая процедура может сохранять дамп TSO. Чтобы сделать это, необходимо перед вызовом DB2 разместить набор данных дампа SYSUDUMP или SYSABEND. Когда вы нажимаете клавишу ENTER (после сообщения об ошибке и сообщения READY), система запрашивает дамп. Затем необходимо освободить (FREE) этот набор данных дампа.

## **Анализ сообщений об ошибках и предупреждений, выданных препомпилятором**

В некоторых случаях оператор, сгенерированный препомпилятором DB2, может вызывать сообщения об ошибках компилятора или ассемблера. Необходимо знать, почему возникают ошибки при компиляции исходных операторов, созданных системой DB2. Дополнительную информацию о предупрежденияхсмотрите в следующих разделах, посвященных языкам хоста:

- “Кодирование операторов SQL в программе на языке ассемблер” на стр. 145
- “Кодирование операторов SQL в программах на языках С или С++” на стр. 160

- “Кодирование операторов SQL в программах на языке COBOL” на стр. 181
- “Кодирование операторов SQL в программах на языке FORTRAN” на стр. 204
- “Кодирование операторов SQL в программах на языке PL/I” на стр. 215.

## Выходной поток SYSTEM прекомпилиатора

Прекомпилиатор DB2 создает выходные данные SYSTEM, если размещен набор данных SYSTEM. Если для подготовки и выполнения программы используются панели подготовки программы, DB2I размещает SYSTEM в соответствии с заданной опцией TERM.

В выходной поток SYSTEM направляется краткая информация о результатах работы прекомпилиатора, все сгенерированные прекомпилиатором сообщения об ошибках и ошибочные операторы, если удалось их обнаружить. Иногда одних только сообщений об ошибках оказывается недостаточно. В таких случаях для обнаружения в исходном тексте программы ошибочного оператора можно использовать номер строки, содержащийся в каждом сообщении об ошибках.

На рис. 123 показан формат выходных данных SYSTEM.

```
DB2 SQL PRECOMPILER      MESSAGES
DSNH104I E    DSNHPARS LINE 32 COL 26  ILLEGAL SYMBOL "X"  VALID SYMBOLS ARE:, FROM1
SELECT VALUE INTO HIPPO X;2

DB2 SQL PRECOMPILER      STATISTICS
SOURCE STATISTICS3
  SOURCE LINES READ: 36
  NUMBER OF SYMBOLS: 15
  SYMBOL TABLE BYTES EXCLUDING ATTRIBUTES: 1848
  THERE WERE 1 MESSAGES FOR THIS PROGRAM.4
THERE WERE 0 MESSAGES SUPPRESSED BY THE FLAG OPTION.5
111664 BYTES OF STORAGE WERE USED BY THE PRECOMPILER.6
RETURN CODE IS 87
```

*Рисунок 123. Вывод SYSTEM прекомпилиатора DB2*

### Примечания к рис. 123:

1. Сообщение об ошибке.
2. Исходный оператор SQL.
3. Значения статистики для исходной программы.
4. Общее число обнаруженных ошибок.
5. Общее число обнаруженных ошибок, для которых не выведены сообщения. Такие ошибки могут появляться, если для опции FLAG задано значение, отличное от I.
6. Число байт рабочей памяти, использованных прекомпилиатором DB2 при обработке исходных операторов. Это значение помогает определить требования к выделению памяти для вашей программы.
7. Код возврата: 0 = успех, 4 = предупреждение, 8 = ошибка, 12 = серьезная ошибка, 16 = неисправимая ошибка.

## Выходной поток SYSPRINT прекомпилятора

Выходной поток SYSPRINT создаются прекомпилятором DB2 при использовании процедуры для прекомпиляции программы. Список процедур JCL, поставляемых с DB2,смотрите в разделе Табл. 49 на стр. 470.

Если для подготовки и выполнения программы используются панели подготовки программы, DB2 размещает SYSPRINT в соответствии с заданной опцией TERM (на строке 12 панели подготовки программы: компиляция, прекомпоновка, компоновка и выполнение). Если используется командная процедура DSNH (CLIST), вместо этого можно задать PRINT(TERM), чтобы данные SYSPRINT выводились на терминал, или задать PRINT(квалификатор), чтобы данные SYSPRINT выводились в набор данных с именем *ID\_авторизации.квалификатор.PCLIST*. Если для параметра PRINT не заданы значения LEAVE, NONE или TERM, при завершении работы прекомпилятора DB2 выдает сообщение, в котором указывается, где можно найти распечатки прекомпилятора. Это помогает быстро и легко найти данные диагностики.

В выходной поток SYSPRINT направляется информация об исходном модуле прекомпиляции, если при запуске прекомпилятора DB2 заданы опции SOURCE и XREF.

Вывод SYSPRINT имеют следующий формат:

- Список опций прекомпилятора DB2 (рис. 124 ), влияющих на прекомпиляцию (если не задано NOOPTIONS).

```
DB2 SQL PRECOMPILER          Версия 6
OPTIONS SPECIFIED: HOST(PLI),XREF,SOURCE1
OPTIONS USED - SPECIFIED OR DEFAULTED2
APOST
APOSTSQL
    CONNECT(2)
    DEC(15)
    FLAG(I)
NOGRAPHIC
    HOST(PLI)
    NOT KATAKANA
    LINECOUNT(60)
    MARGINS(2,72)
    ONEPASS
    OPTIONS
PERIOD
    SOURCE
    STDSQL(NO)
    SQL(DB2)
    XREF
```

Рисунок 124. Вывод SYSPRINT прекомпилятора DB2: раздел опций

### Примечания к рис. 124:

1. В этом разделе перечислены опции, заданные во время прекомпиляции. Если среди опций прекомпилятора задана опция NOOPTIONS, этот список не выводится.
2. В этом разделе выводятся используемые опции, в том числе опции по умолчанию, индуцированные значения и заданные опции.

Прекомпилятор DB2 переопределяет или игнорирует все заданные опции, которые не соответствуют языку хоста.

- Исходный текст (рис. 125 ) операторов (только если задана опция SOURCE).

```
DB2 SQL PRECOMPILER      TMN5P40:PROCEDURE OPTIONS (MAIN):      PAGE 2
1      TMN5P40:PROCEDURE OPTIONS(MAIN) ;          00000100
2  *****/*****00000200
3  *      описание программы и пролог          00000300
:
1324  *****/*****00132400
1325  /* ПОЛУЧАЕМ ИНФОРМАЦИЮ О ПРОЕКТЕ ИЗ      */
1326  /* ТАБЛИЦЫ ПРОЕКТОВ                      */      00132500
1327  *****/*****00132600
1328  EXEC SQL SELECT ACTNO, PREQPROJ, PREQACT    00132800
1329      INTO PROJ_DATA                      00132900
1330      FROM TPREREQ                         00133000
1331      WHERE PROJNO = :PROJ_NO;                00133100
1332                                         00133200
1333  *****/*****00133300
1334  /* ПРОЕКТ ЗАВЕРШЕН. УДАЛЯЕМ ЕГО.          */      00133400
1335  *****/*****00133500
1336                                         00133600
1337  EXEC SQL DELETE FROM PROJ                 00133700
1338      WHERE PROJNO = :PROJ_NO;                00133800
:
1523  END;                                     00152300
```

Рисунок 125. Вывод SYSPRINT прекомпилятора DB2: раздел исходного текста операторов

#### Примечание к рис. 125:

- Последовательные номера в левом столбце генерируются прекомпилятором DB2 и используются в списке символьических перекрестных ссылок, в сообщениях об ошибках прекомпилятора и в сообщениях об ошибках связывания.
- Последовательные номера в правом столбце – производные от последовательных номеров входных операторов.
- Список (рис. 126 на стр. 523 ) символьических имен, использованных в операторах SQL (этот список появляется, только если задана опция XREF).

DB2 SQL PRECOMPIILER	SYMBOL CROSS-REFERENCE LISTING		PAGE 29
DATA NAMES	DEFN	REFERENCE	
"ACTNO"	****	FIELD 1328	
"PREQACT"	****	FIELD 1328	
"REQPROJ"	****	FIELD 1328	
"PROJNO"	****	FIELD 1328 1331 1338	
...			
PROJ_DATA	495	CHARACTER(35) 1329	
PROJ_NO	496	CHARACTER(3) 1331 1338	
"TPREREQ"	****	TABLE 1330 1337	

Рисунок 126. Вывод SYSPRINT прекомпилятора DB2: раздел символьических перекрестных ссылок

#### Примечания к рис. 126:

##### DATA NAMES

Указывает символические имена, использованные в исходных операторах. Имена, заключенные в двойные кавычки ("") или в простые кавычки ('') – это имена элементов SQL, таких как таблицы, столбцы и ID авторизации. Другие имена – это имена переменных хоста.

##### DEFN

Это номер строки, сгенерированный прекомпилятором для этого имени. \*\*\*\* означает, что этот объект не был определен или прекомпилятор не смог распознать эти объявления.

##### REFERENCE

Содержит два типа информации: тип объекта, для которого в программе задано это символическое имя, и номера строк, в которых используется это символическое имя. Если это символическое имя соответствует переменной хоста, в этом списке также указывается тип данных или STRUCTURE.

- Сводка (рис. 127 на стр. 524 ): общее число ошибок, обнаруженных прекомпилятором DB2, и список сообщений об ошибках, выданных прекомпилятором DB2.

```
DB2 SQL PRECOMPILER          STATISTICS

SOURCE STATISTICS
  SOURCE LINES READ: 15231
  NUMBER OF SYMBOLS: 1282
  SYMBOL TABLE BYTES EXCLUDING ATTRIBUTES: 64323

  THERE WERE 1 MESSAGES FOR THIS PROGRAM.4
  THERE WERE 0 MESSAGES SUPPRESSED.5
  65536 BYTES OF STORAGE WERE USED BY THE PRECOMPILER.6
  RETURN CODE IS 8.7
  DSNH104I E LINE 590 COL 64 ILLEGAL SYMBOL: 'X'; VALID SYMBOLS ARE:,FROM8
```

Рисунок 127. Вывод SYSPRINT прекомпилиатора DB2: раздел сводки

**Примечания к рис. 127:**

1. Общее число строк исходной программы.
2. Общее число символьических имен в таблице символьических имен (имена SQL и имена переменных хоста).
3. Число байт, использованных для таблицы символьических имен.
4. Общее число обнаруженных ошибок.
5. Общее число обнаруженных ошибок, для которых не выведены сообщения. Эта строка появляется, если задана опция FLAG.
6. Число байт рабочей памяти, использованных прекомпилиатором DB2 при обработке исходных операторов.
7. Код возврата: 0 = успех, 4 = предупреждение, 8 = ошибка, 12 = серьезная ошибка, 16 = неисправимая ошибка.
8. Сообщения об ошибках (в этом примере обнаружена только одна ошибка).

## Глава 6–3. Обработка программ среды DL/I batch

В разделах этой главы рассказывается о поддержке программ DL/I batch в DB2:

- “Планирование работы с DL/I batch”
- “Особенности разработки программы” на стр. 526
- “Наборы входных и выходных данных” на стр. 528
- “Особенности подготовки программы” на стр. 530
- “Перезапуск и восстановление” на стр. 533

### Планирование работы с DL/I batch

Ниже в разделе Свойства и функции поддержки DL/I batch в DB2 рассказано, что можно делать в программе DL/I batch. В разделе “Требования к DB2 в задании DL/I batch” на стр. 526 говорится, как это сделать.

### Свойства и функции поддержки DL/I batch в DB2

Программа DL/I batch может выполнять:

- Любой вызов IMS batch, кроме вызовов ROLS, SETS и SYNC. Вызовы ROLS и SETS служит для обработки промежуточной точки резервирования, которую DB2 не поддерживает. Вызов SYNC служит для обработки точки принятия без идентификации точки принятия каким–либо значением. IMS не допускает вызовы SYNC в пакетной среде, поэтому они не включены в поддержку DL/I batch в DB2.

Выполнение вызова ROLS, SETS или SYNC в прикладной программе приводит к аварийному завершению X'04E' системы с кодом причины X'00D44057' в регистре 15.

- Вызовы GSAM.
- Служебные вызовы системы IMS.
- Все операторы SQL, кроме COMMIT и ROLLBACK. В средах IMS и CICS эти операторы SQL не поддерживаются. В программе для принятия данных должен использоваться вызов CHKP среды IMS, а для отката изменений – вызов ROLL или ROLB среды IMS.

Попытка выполнить оператор COMMIT вызывает SQLCODE –925; попытка выполнить оператор ROLLBACK вызывает SQLCODE –926. Эти операторы также возвращают SQLSTATE '2D521'.

- Все вызовы стандартных или традиционных видов доступа (например, QSAM, VSAM и т.д.)

Базы данных DB2 и IMS, так же, как и последовательные наборы данных, доступные через GSAM, можно перезапускать с помощью средства Checkpoint and Restart среды IMS.

В DB2 для доступа как к данным DB2, так и к данным DL/I служат следующие средства DB2 и IMS:

- Вызовы синхронизации IMS – для принятия и аварийного завершения единиц восстановления

- Утилита подключения IMS DB2 – для управления протоколом двухфазного принятия, позволяющее обеим системам синхронизировать единицу восстановления при перезапуске после аварийного завершения
- Журнал IMS, используемый для записи момента принятия.

## Требования к DB2 в задании DL/I batch

Если DB2 используется в задании DL/I batch, в программу и в шаг задания JCL надо внести следующие изменения:

- Надо добавить в программу операторы SQL для получения доступа к данным DB2. Затем необходимо прекомпилировать программу и скомпоновать полученный DBRM в план или пакет, как описано в разделе “Глава 6–1. Подготовка прикладной программы к запуску” на стр. 435.
- Перед тем как запускать программу, задайте JOBLIB, STEPLIB или книгу компоновки для доступа к загрузочной библиотеке DB2, чтобы можно было загружать модули DB2.
- Либо задайте с помощью оператора DDITV02 DD набор входных данных, либо укажите компонент подсистемы с параметром SSM= в процедуре вызова DL/I batch. Каждый из них задает информацию о шаге пакетного задания и DB2. Подробную информациюсмотрите в разделе “Ввод DL/I batch в DB2” на стр. 528.
- По желанию можно с помощью оператора DDOTV02 DD задать набор выходных данных. Этот набор данных может потребоваться для получения сообщений от утилиты подключения IMS о неоднозначной и диагностической информации.

## Авторизация

Когда пакетная программа пытается запустить первый оператор SQL, DB2 проверяет, имеет ли ID авторизации привилегию EXECUTE для данного плана. DB2 использует этот же ID для последующих проверок авторизации, а также для идентификации записей из трассировок учетных записей и производительности.

Первичный ID авторизации – это значение параметра USER в операторе задания, если он доступен. Если задание передано на выполнение, это имя регистрации TSO. Иначе используется имя PSB среды IMS. Но в этом случае ID не должен начинаться с “SYSADM” – это приводит к аварийному завершению задания. При попытке изменить ID авторизации в обработчике пользователя пакетное задание отклоняется.

## Особенности разработки программы

Использование DL/I batch надо учитывать при разработке и программировании в случаях, описанных ниже.

## **Адресные пространства**

Регион DL/I batch не зависит ни от управляющего региона IMS, ни от адресного пространства CICS. Регион DL/I batch загружает код DL/I в регион прикладной программы вместе с самой программой.

## **Принятие**

Чаще сохраняйте изменения в программах IMS batch, чтобы не занимать ресурсы на длительное время. Если для восстановления требуются координированное принятие, посмотрите раздел *Раздел 4 (Том 1) DB2 Administration Guide*.

## **Операторы SQL и вызовы IMS**

Нельзя пользоваться операторами SQL COMMIT и ROLLBACK – это приведет к возврату кода ошибки SQL. Кроме того, нельзя пользоваться вызовами ROLS, SETS и SYNC, которые приводят к аварийному завершению прикладной программы.

## **Вызовы контрольных точек**

Используйте в программе операторы SQL и вызовы DL/I, и вызывайте контрольные точки. Все контрольные точки, вызванные пакетной программой, должны быть уникальны. Частота контрольных точек зависит от структуры программы. В контрольной точке позиционирование DL/I теряется, указатели DB2 закрываются (возможно, кроме указателей, заданных с условием WITH HOLD), блокировки на время принятия освобождаются (тоже с некоторыми исключениями), а изменения баз данных с этого момента считаются постоянными и для IMS, и для DB2.

## **Синхронизация прикладной программы**

Можно разработать программу, не используя контрольные точки IMS. В этом случае, если программа заканчивается аварийно до завершения, DB2 отменяет все изменения, а для отмены изменений DL/I можно воспользоваться утилитой возврата IMS batch.

Можно также выполнить динамический возврат изменений IMS в том же самом задании. Нужно задать для параметра ВКО значение 'Y' и разместить журнал IMS на устройстве памяти с прямым доступом (DASP).

Вы можете столкнуться с проблемой, если ошибка в системе возникнет после завершения программы, но до завершения шага задания. Если до завершения программы нет вызова контрольной точки, DB2 вносит изменения в единицу работы без использования IMS. Если ошибка возникает в системе до внесения DL/I изменений в данные, данные DB2 будут рассинхронизированы с изменениями DL/I. Если ошибка в системе возникает во время обработки принятия DB2, данные DB2 могут быть неоднозначны.

*Рекомендуется в конце любого задания, изменяющего данные, всегда вызывать символьическую контрольную точку для координации принятия выполняющейся единицы работы для IMS и DB2. При перезапуске программы нужно воспользоваться вызовом XRST, чтобы получить информацию о контрольной точке и разрешить все неоднозначные единицы работы DB2.*

## **Контрольная точка и использование XRST**

Если используется вызов XRST, DB2 считает, что выполняемая контрольная точка является символической. Опции вызова символической контрольной точки отличаются от опций вызова обычной контрольной точки. Использование неверной формы вызова контрольной точки может привести к ошибкам.

Если вызов XRST не используется, DB2 считает любой вызов контрольной точки вызовом обычной контрольной точки.

Чтобы перезапуск был проще, ID контрольных точек должны состоять из символов EBCDIC.

Если программа должна быть перезапускаемой, нужно воспользоваться символической контрольной точкой и вызовами XRST. Если используется вызов XRST, он должен быть первым выполняемым вызовом IMS и иметь место до выполнения любого из операторов SQL. Кроме того, вызов XRST должен быть только один.

## **Аварийные завершения вызова синхронизации**

Если программа содержит неверный вызов синхронизации IMS (CHKP, ROLB, ROLL или XRST), приводящий к тому, что IMS выдает плохой код состояния в PCB, DB2 аварийно завершает программу. Убедитесь, что эти вызовы протестированы, прежде чем использовать программу в промышленном режиме.

---

## **Наборы входных и выходных данных**

Обратите внимание на два набора данных:

- DDITV02 – для ввода
- DDOTV02 – для вывода.

## **Ввод DL/I batch в DB2**

Ввод для программы можно задать двумя способами:

- При помощи оператора DD DDITV02, в котором описан файл со следующими опциями DCB: LRECL=80 и RECFM=F или FB.
- При помощи указания имени компонента подсистемы IMS в параметре SSM= вызова DL/I в DB2. Имя компонента подсистемы – это значение параметра SSM, присоединенное к значению параметра IMSID.

Если использовать и оператор DD DDITV02, и задание компонента подсистемы, оператор DD DDITV02 переопределяет заданный компонент подсистемы. Если не задать ничего, DB2 аварийно завершает программу с кодом аварийного завершения системы X'04E' и уникальным кодом причины в регистре 15.

Поля в спецификации компонента подсистемы или в наборе данных DDITV02 указываются позиционно и разделяются запятыми:

SSN,LIT,ESMT,RTT,ERR,CRC,CONNECTION\_NAME,PLAN,PROG

<b>Поле</b>	<b>Содержимое</b>
SSN	Имя подсистемы DB2, задается обязательно. Для установления соединения с DB2 надо задать имя.
LIT	Длина значения SSN – от одного до четырех символов. Для DB2 нужен признак языкового интерфейса для маршрутизации операторов SQL при обработке в среде IMS оперативного режима. Так как пакетная программа может соединиться только с одной системой DB2, DB2 не использует значение LIT. Рекомендуется задать значение SYS1, однако его можно пропустить (введите SSN,,ESMT).
ESMT	Длина значения LIT – от нуля до четырех символов. Имя модуля инициации DB2 – DSNNMIN10, задается обязательно.
RTT	Длина значения ESMT – восемь символов. Задание таблицы трансляции ресурсов необязательно.
ERR	Длина RTT – от нуля до восьми символов. Опция ошибок региона задает, что делать, если DB2 не работает или план недоступен. Есть три возможности: <ul style="list-style-type: none"> <li>• <i>R</i> (по умолчанию) возвращает программе код возврата SQL. Наиболее распространенный SQLCODE в этом случае – –923 (SQLSTATE '57015').</li> <li>• <i>Q</i> вызывает аварийное завершение в пакетной среде, однако в среде оперативного режима помещает входное сообщение снова в очередь.</li> <li>• <i>A</i> вызывает аварийное завершение и в пакетной среде, и в среде оперативного режима.</li> </ul> Если программа использует вызов XRST, и в вызове XRST требуется координированное восстановление, ERR игнорируется. В этом случае если DB2 не работает, программа завершается аварийно.
CRC	Длина значения ERR – от нуля до одного символа. Так как команды DB2 не поддерживаются в среде DL/I batch, символ распознавания команды здесь не используется.
CONNECTION_NAME	Длина значения CRC – от нуля до одного символа. Имя соединения задавать не обязательно. Это имя шага задания, координирующего активность DB2. Если не задать эту опцию, по умолчанию будут использоваться следующие имена соединений: <ul style="list-style-type: none"> <li><b>Тип программы Имя соединения по умолчанию</b></li> <li><b>Пакетное задание Имя задания</b></li> <li><b>Запущенная задача Имя запущенной задачи</b></li> <li><b>Пользователь TSO ID авторизации TSO</b></li> </ul>

Если пакетное задание, изменяющее данные, завершается неудачно, нужно использовать отдельное задание для перезапуска пакетного задания. Имя соединения для задания перезапуска должно быть именем пакетного задания, завершившегося неудачно. Если же используется имя соединения по умолчанию, у задания перезапуска должно быть имя пакетного задания по изменению данных, завершившегося неудачно.

Для DB2 требуются уникальные имена соединений. Если с DB2 пытаются соединиться две программы с одним именем соединения, вторая из них не сможет соединиться с DB2.

Длина значения CONNECTION\_NAME – от 1 до 8 символов.

**PLAN** Имя плана DB2 задавать не обязательно. Если не задать имя плана, имя модуля программы проверяется по таблице трансляции ресурсов. Если в таблице трансляции ресурсов найдено соответствие, в качестве имени плана DB2 используется транслируемое имя. Если совпадений нет, в качестве имени плана DB2 используется имя модуля программы.

Длина значения PLAN – от 0 до 8 символов.

**PROG** Имя прикладной программы задается обязательно. Оно идентифицирует программу, которая должна быть загружена и принять управление.

Длина значения PROG – от 1 до 8 символов.

Ниже приведен пример полей в записи:

DSN,SYS1,DSNMIN10,,R,-,BATCH001,DB2PLAN,PROGA

## Вывод DL/I batch в DB2

В среде IMS оперативного режима DB2 посыпает незатребованные сообщения о состоянии MTO (master terminal operator – оператор главного терминала) и записывает информацию о неоднозначной обработке и диагностическую информацию в журнал IMS. В пакетной среде DB2 направляет эту информацию в выходной набор данных, заданный в операторе DD DDOTV02. Набор выходных данных должен иметь опции DCB RECFM=V или VB, LRECL=4092 и BLKSIZE не меньше LRECL + 4. Если оператор DD пропущен, DB2 выдает сообщение IEC130I и продолжает обработку без всякого вывода.

Может потребоваться сохранить и вывести этот набор данных, поскольку информация полезна для диагностики. Для вывода записей набора данных переменной длины и в шестнадцатеричном, и в символьном виде можно воспользоваться модулем DFSERA10 среды IMS.

---

## Особенности подготовки программы

При подготовке программы с доступом к DB2 и DL/I из пакетной программы надо иметь в виду следующее.

## Прекомпиляция

При добавлении операторов SQL в программу нужно прекомпилировать программу и скомпоновать полученный модуль DBRM в план или пакет, как описано в разделе “Глава 6–1. Подготовка прикладной программы к запуску” на стр. 435.

## Связывание

У владельца плана или пакета должны быть все привилегии, требуемые для выполнения встроенных в них операторов SQL. До того, как пакетная программа сможет выполнять операторы SQL, должен существовать план DB2.

Имя плана для DB2 можно задать одним из следующих способов:

- Во входном наборе данных DDITV02.
- В спецификации элементов подсистемы.
- По умолчанию; в этом случае именем плана будет имя загрузочного модуля программы, уже заданное в DDITV02.

DB2 передает имя плана в пакет привязки среды IMS. Если имя плана не задать в DDITV02, и таблицы трансляции ресурсов (RTT) не существует, или данного имени нет в RTT, DB2 использует в качестве имени плана переданное имя. Если данное имя есть в RTT, это имя транслируется в план, заданный для RTT.

Рекомендуемый подход – дать плану DB2 имя загрузочного модуля программы, что используется утилитой подключения IMS по умолчанию. Это имя плана должно совпадать с именем программы.

## **Компоновка**

В DB2 есть подпрограммы языкового интерфейса для каждой отдельной поддерживаемой среды. Для DL/I batch DB2 требуется подпрограмма языкового интерфейса IMS. Также требуется DFLI000, скомпонованный с прикладной программой.

## **Загрузка и запуск**

Для запуска программы с DB2 нужен план DB2. План DB2 создается в процессе компоновки. Сначала DB2 проверяет, может ли шаг пакетного задания связаться с пакетным заданием DB2. Затем DB2 проверяет, может ли прикладная программа получить доступ к DB2 и выполнить идентификацию пользователя пакетных заданий, обращающихся к DB2.

Есть два способа передачи программ DL/I batch на выполнение в DB2:

- Процедура DL/I batch может запустить в качестве программы модуль DSNMTV01. DSNMTV01 загружает “реальную” программу. Пример задания JCL для передачи на выполнение программы DL/I batch этим способом смотрите в разделе “Передача на выполнение программы DL/I batch с использованием DSNMTV01” на стр. 532.
- Процедура DL/I batch может запустить программу без использования модуля DSNMTV01. Для этого сделайте следующее:
  - В процедуре DL/I batch задайте SSM= .
  - В регионе пакетной среды для JCL программы задайте:
    - MBR=имя–программы
    - SSM=имя подсистемы DB2

Пример задания JCL для передачи на выполнение программы DL/I batch этим способом смотрите в разделе “Передача на выполнение программы DL/I batch без использования DSNMTV01” на стр. 532.

## **Передача на выполнение программы DL/I batch с использованием DSNMTV01**

В следующем типовом примере задания JCL показана программа на языке COBOL, IVP8CP22, которая запускается с использованием поддержки DL/I batch в DB2.

- На первом шаге используется стандартная процедура DLIBATCH среды IMS.
- Второй шаг показывает, как использовать программу DFSEERA10 среды IMS для вывода содержания выходного набора данных DDOTV02.

```
//ISOCS04 JOB 3000,ISOIR,MSGLEVEL=(1,1),NOTIFY=ISOIR,
//          MSGCLASS=T,CLASS=A
//JOBLIB   DD  DISP=SHR,
//          DSN=prefix.SDSNLOAD
//***** *****
//** СЛЕДУЮЩИЙ ШАГ ПЕРЕДАЕТ НА ВЫПОЛНЕНИЕ ЗАДАНИЕ COBOL IVP8CP22,
//** КОТОРОЕ ИЗМЕНЯЕТ И БАЗЫ ДАННЫХ DB2, И БАЗЫ ДАННЫХ DL/I.
//**
//***** *****
//UPDTE    EXEC DLIBATCH,DBRC=Y,LOGT=SYSDA,COND=EVEN,
//          MBR=DSNMTV01,PSB=IVP8CA,BKO=Y,IRLM=N
//G.STEPLIB DD
//          DD
//          DSN=prefix.SDSNLOAD,DISP=SHR
//          DD  DSN=prefix.RUNLIB.LOAD,DISP=SHR
//          DD  DSN=SYS1.COB2LIB,DISP=SHR
//          DD  DSN=IMS.PGMLIB,DISP=SHR
//G.STEPAT  DD  DSN=IMSCAT,DISP=SHR
//G.DDOTV02  DD  DSN=&TEMP1,DISP=(NEW,PASS,DELETE),
//          SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,
//          DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02  DD  *
//          SSDQ,SYS1,DSNMIN10,,A,-,BATCH001,,IVP8CP22
/*
//***** *****
//*** ВСЕГДА ПЫТАТЬСЯ НАПЕЧАТАТЬ НАБОР ДАННЫХ DDOTV02 ***
//***** *****

//STEP3    EXEC PGM=DFSEERA10,COND=EVEN
//STEPLIB  DD  DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSNAME=&TEMP1,DISP=(OLD,DELETE)
//SYSIN    DD *
CONTROL  CNTL K=000,H=8000
OPTION   PRINT
/*
//
```

## **Передача на выполнение программы DL/I batch без использования DSNMTV01**

В следующем примере типового задания JCL показана программа на языке COBOL, IVP8CP22, которая запускается с использованием поддержки DL/I batch в DB2.

```

//TEPCTEST JOB 'USER=ADMF001',MSGCLASS=A,MSGLEVEL=(1,1),
//           TIME=1440,CLASS=A,USER=SYSADM,PASSWORD=SYSADM
//*****
//BATCH EXEC DLIBATCH,PSB=IVP8CA,MBR=IVP8CP22,
//       BKO=Y,DBRC=N,IRLM=N,SSM=SSDQ
//*****
//SYSPRINT DD   SYSOUT=A
//REPORT  DD SYSOUT=*
//G.DDOTV02 DD DSN=&TEMP,DISP=(NEW,PASS,DELETE),
//   SPACE=(CYL,(10,1),RLSE),
//   UNIT=SYSDA,DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD *
//SSDQ,SY1,DSNMIN10,,Q,",DSNMTE1,,IVP8CP22
//G.SYSIN DD *
/*
//*****
//*** ВСЕГДА ПЫТАТЬСЯ НАПЕЧАТАТЬ НАБОР ДАННЫХ DDOTV02
//*****
//PRTLOG EXEC PGM=DFSER10,COND=EVEN
//STEPLIB  DD   DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSUT1   DD DSN=&TEMP,DISP=(OLD,DELETE)
//SYSIN    DD *
CONTROL CNTL K=000,H=8000
OPTION PRINT
/*

```

## Перезапуск и восстановление

Для перезапуска пакетной программы, изменяющей данные, сначала нужно запустить утилиту возврата IMS batch, а затем – задание перезапуска с указанием ID последней успешно пройденной контрольной точки.

- Образец JCL для этой утилиты находится в разделе “Пример JCL пакетного возврата.”
- Образец JCL для задания перезапуска приведен в разделе “Пример JCL перезапуска задания DL/I batch” на стр. 534.
- Инструкции по поиску последней успешно пройденной контрольной точкисмотрите в разделе “Поиск ID контрольной точки DL/I batch” на стр. 535.

## Пример JCL пакетного возврата

В следующем типовом примере задания JCL показан пакетный возврат для PSB=IVP8CA.

```

//ISOCS04 JOB 3000,ISOIR,MSGLEVEL=(1,1),NOTIFY=ISOIR,
//           MSGCLASS=T,CLASS=A
//***** * * * * * * * * * * * * * * * * * * * * * * * * * * *
//*
//** ВОЗВРАТ К ПОСЛЕДНЕЙ КОНТРОЛЬНОЙ ТОЧКЕ.
//**          ЕСЛИ RC=0028, ЗАПИСАТЬ С УКАЗАНИЕМ NO-UPDATE *
//*
//* - - - - - - - - - - - - - - - - - - - - - - - - - - - *
//BACKOUT EXEC PGM=DFSRRC00,
//           PARM='DLI,DFSBBO00,IVP8CA,,,,,,,Y,N,,Y',
//           REGION=2600K,COND=EVEN
//*                                     |
//*                                     ---> DBRC ВКЛЮЧЕН
//STEPLIB  DD   DSN=IMS.RESLIB,DISP=SHR
//STEPCAT DD  DSN=IMSCAT,DISP=SHR
//IMS      DD   DSN=IMS.PSBLIB,DISP=SHR
//          DD   DSN=IMS.DBDLIB,DISP=SHR

```

```

/*
/* IMSLOGR DD набор данных обязателен
/* IEFRDER DD набор данных обязателен
//DFSVSAMP DD *
OPTIONS,LTWA=YES
2048,7
1024,7
/*
//SYSIN    DD DUMMY
/*

```

## Пример JCL перезапуска задания DL/I batch

Операционные процедуры могут перезапустить шаг задания DL/I batch для прикладной программы, использующей вызовы XRST и символические контрольные точки среды IMS.

Программу BMP нельзя перезапустить в пакетной среде DL/I в DB2. Записи символьических контрольных точек в этом случае недоступны, что приводит к пользовательскому аварийному завершению U0102 среды IMS.

Для перезапуска пакетного задания, завершающегося аварийно или преждевременно, найдите ID контрольной точки этого задания в журнале системы MVS или в списке SYSOUT задания, завершившегося с ошибкой. Перед перезапуском шага задания укажите ID контрольной точки в опции CKPTID=значение процедуры DLIBATCH, затем запустите задание. Если используется имя соединения по умолчанию (т.е. в наборе входных данных DDITV02 не задана опция с именем этого соединения), имя перезапускаемого задания должно быть именем задания, завершившегося с ошибкой. Смотрите следующий типовой пример, в котором значение последнего ID контрольной точки — IVP80002:

```

//ISOCS04 JOB 3000,OJALA,MSGLEVEL=(1,1),NOTIFY=OJALA,
//      MSGCLASS=T,CLASS=A
//***** ****
/*
//** СЛЕДУЮЩИЙ ШАГ ПЕРЕЗАПУСКАЕТ ПРОГРАММУ НА ЯЗЫКЕ COBOL IVP8CP22, КОТОРАЯ
//** ИЗМЕНЯЕТ И БАЗЫ ДАННЫХ DB2, И БАЗЫ ДАННЫХ DL/I, С ТОЧКИ CKPTID=IVP80002.
/*
//***** ****
//RSTRT EXEC DLIBATCH,DBRC=Y,COND=EVEN,LOGT=SYSDA,
// MBR=DSMMTV01,PSB=IVP8CA,BKO=Y,IRLM=N,CKPTID=IVP80002
//G.STEPLIB DD
//      DD DSN=prefix.SDSNLOAD,DISP=SHR
//      DD DSN=prefix.RUNLIB.LOAD,DISP=SHR
//      DD DSN=SYS1.COB2LIB,DISP=SHR
//      DD DSN=IMS.PGMLIB,DISP=SHR
//**      другие программные библиотеки
//** G.IEFRDER набор данных обязателен
//G.STEPCAT DD DSN=IMSCAT,DISP=SHR
//** G.IMSLOGR набор данных обязателен
//G.DDITV02 DD DSN=&TEMP2,DISP=(NEW,PASS,DELETE),
//      SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,
//      DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD *
//      DB2X,SYS1,DSNMIN10,,A,-,BATCH001,,IVP8CP22
/*
//***** ****
//*** ВСЕГДА ПЫТАТЬСЯ НАПЕЧАТАТЬ НАБОР ДАННЫХ DDOTV02 ***
//*****

```

```

//STEP8      EXEC PGM=DFSER10,COND=EVEN
//STEPLIB    DD   DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT   DD   SYSOUT=A
//SYSUT1     DD   DSNAME=&TEMP2,DISP=(OLD,DELETE)
//SYSIN      DD *
CONTROL CNTL K=000,H=8000
OPTION PRINT
/*
//

```

## Поиск ID контрольной точки DL/I batch

Когда прикладная программа выполняет вызов контрольной точки среды IMS, IMS посыпает ID контрольной точки на консоль MVS и в поток SYSOUT в сообщении DFS0540I. IMS также записывает ID контрольной точки в запись журнала IMS типа X'41'. Вызовы символических контрольных точек также помещают в журнал IMS одну или несколько записей типа X'18'. XRST использует записи журнала типа X'18' для репозиционирования баз данных DL/I и возврата информации в прикладную программу.

В процессе принятия ID контрольной точки программы передается в DB2. Если во время принятия происходит ошибка и возникает неоднозначная единица работы, DB2 запоминает ID контрольной точки. Для поиска ID последней контрольной точки можно пользоваться следующими способами:

- Найдите в выводе SYSOUT для этого шага задания сообщение DFS0540I, содержащее ID выполненных контрольных точек. Используйте последний указанный ID контрольной точки.
- Найдите в журнале консоли MVS сообщение (сообщения) DFS0540I, с ID контрольных точек, выполненных этой пакетной программой. Используйте последний указанный ID контрольной точки.
- Выполните утилиту пакетного возврата (Batch Backout) среды IMS для возврата баз данных DL/I к последнему (по умолчанию) ID контрольной точки. По окончании пакетного возврата в сообщении DFS395I будет указан ID последней успешно выполненной контрольной точки среды IMS. При перезапуске укажите этот ID контрольной точки.
- При перезапуске DB2 оператор может ввести команду -DISPLAY THREAD(\*) TYPE(INDOUBT), чтобы получить возможную неоднозначную единицу работы (имя соединения и ID контрольной точки). При перезапуске с ID этой контрольной точки программа будет работать, так как эта контрольная точка записывается в журнал IMS, однако может возникнуть ошибка с пользовательским аварийным завершением U102 среды IMS, поскольку IBS не завершила регистрацию информации перед неудачным завершением. В этом случае перезапустите программу с ID предыдущей контрольной точки.

При перезапуске DB2 автоматически выполняет одно из двух действий, если ошибка возникла вне периода неоднозначности: либо возвращает единицу работы к предшествующей контрольной точке, либо выполняет принятие данных без посторонней помощи. Если оператор после этого введет команду -DISPLAY THREAD(\*) TYPE(INDOUBT) никакой информации о единице работы не будет выведено.



---

## Раздел 7. Дополнительные приемы программирования

<b>Глава 7–1. Кодирование динамического SQL в прикладных программах</b>	543
Выбор между статическим и динамическим SQL	544
Переменные хоста делают статический SQL гибким	544
Динамический SQL абсолютно гибок	545
Чего не может динамический SQL	545
Что делает прикладная программа, использующая динамический SQL	545
Производительность статического и динамического SQL	545
Кэширование операторов динамического SQL	547
Использование кэша динамических операторов	548
Сохранение подготовленных операторов после точек принятия	549
Ограничение динамического SQL при помощи утилиты ограничения ресурсов	552
Как задать в программе обработку реактивного ограничения	553
Как задать в программе обработку прогностического ограничения	553
Использование прогностического ограничения для устаревших реквестеров DRDA	554
Использование прогностического ограничения на поддерживающих реквестерах	554
Выбор языка хоста для прикладных программ с динамическим SQL	554
Динамический SQL без операторов SELECT	555
Динамическое выполнение с использованием EXECUTE IMMEDIATE	555
Динамическое выполнение с использованием PREPARE и EXECUTE	556
Динамический SQL для операторов SELECT с фиксированным списком	559
Что должна делать ваша прикладная программа	560
Динамический SQL для операторов SELECT с переменным списком	562
Что должна делать ваша прикладная программа	562
Подготовка оператора SELECT с переменным списком	563
Динамическое выполнение оператора SELECT с переменным списком	573
Выполнение произвольных операторов с маркерами параметров	574
Как опция связывания REOPT(VARS) влияет на динамический SQL	576
Использование динамического SQL в языке COBOL	577
<b>Глава 7–2. Использование хранимых процедур в системах клиент–сервер</b>	579
Введение в хранимые процедуры	579
Пример простой хранимой процедуры	581
Настройка среды хранимых процедур	584
Определение хранимой процедуры для DB2	585
Обновление среды хранимых процедур (для системных администраторов)	590
Перенос хранимых процедур в среду, созданную WLM (для системных администраторов)	591
Переопределение хранимых процедур, определенных в SYSIBM.SYSPROCEDURES	592
Написание и подготовка хранимой процедуры	593
Требования к языкам хранимой процедуры и вызывающей программы	593
Вызов других программ	594
Использование повторно–входимого кода	594

Написание хранимой процедуры в виде основной программы или подпрограммы . . . . .	595
Ограничения для хранимых процедур . . . . .	599
Использование в хранимой процедуре специальных регистров . . . . .	599
Обращение в хранимой процедуре к другим узлам . . . . .	601
Написание хранимой процедуры, обращающейся к базам данных IMS . . . . .	602
Написание хранимой процедуры, возвращающей наборы результатов клиенту DRDA . . . . .	603
Подготовка хранимой процедуры . . . . .	604
Связывание хранимой процедуры . . . . .	606
<b>Написание и подготовка прикладной программы, использующей хранимые процедуры . . . . .</b>	<b>607</b>
Формы оператора CALL . . . . .	607
Полномочия на выполнение хранимых процедур . . . . .	609
Соглашения по компоновке . . . . .	609
Использование переменных–индикаторов для ускорения обработки . . . . .	631
Объявление типов данных для передаваемых параметров . . . . .	632
Написание клиентской программы DB2 for OS/390, получающей наборы результатов . . . . .	637
Подготовка клиентской программы . . . . .	642
<b>Выполнение хранимой процедуры . . . . .</b>	<b>643</b>
Как DB2 определяет, какую версию хранимой процедуры нужно выполнять . . . . .	644
Вызов разных версий хранимой процедуры из одной программы . . . . .	645
Одновременное выполнение нескольких хранимых процедур . . . . .	646
Обращение к другим ресурсам (не DB2) . . . . .	648
<b>Тестирование хранимой процедуры . . . . .</b>	<b>649</b>
Отладка хранимой процедуры как отдельной программы на рабочей станции . . . . .	650
Отладка при помощи Debug Tool и IBM VisualAge® COBOL . . . . .	650
Отладка при помощи CODE/370 . . . . .	651
Использование опции времени выполнения MSGFILE . . . . .	653
Использование специального драйвера . . . . .	653
Использование операторов SQL INSERT . . . . .	653
<b>Глава 7–3. Настройка запросов . . . . .</b>	<b>655</b>
<b>Общие советы и вопросы . . . . .</b>	<b>655</b>
Нельзя ли упростить запрос? . . . . .	655
Правильно ли написаны все предикаты? . . . . .	655
Есть ли в запросе подзапросы? . . . . .	656
Есть ли в запросе функции столбцов? . . . . .	657
Есть ли в предикате статического запроса SQL входные переменные? . . . . .	657
Проблемы, связанные с корреляцией столбцов . . . . .	658
Можно ли написать запрос без использования выражений столбцов? . . . . .	658
<b>Написание эффективных предикатов . . . . .</b>	<b>658</b>
Свойства предикатов . . . . .	659
Предикаты в условии ON . . . . .	662
<b>Общие правила проверки предикатов . . . . .</b>	<b>662</b>
Порядок проверки предикатов . . . . .	663
Сводка результатов по обработке предикатов . . . . .	663
Примеры свойств предикатов . . . . .	668
Показатели фильтрации предикатов . . . . .	669
Модификация предикатов DB2 . . . . .	674
Корреляция столбцов . . . . .	678

Эффективное использование переменных хоста . . . . .	682
Использование опции REOPT(VARS) для изменения пути доступа во время выполнения . . . . .	682
Изменение запросов, чтобы повлиять на выбор пути доступа . . . . .	683
Написание эффективных подзапросов . . . . .	687
Связанные подзапросы . . . . .	687
Несвязанные подзапросы . . . . .	688
Преобразование подзапроса в операцию объединения . . . . .	690
Настройка подзапроса . . . . .	691
Особые способы повлиять на выбор пути доступа . . . . .	693
Получение информации о пути доступа . . . . .	693
Минимизация затрат при получении небольшого числа строк: OPTIMIZE FOR n ROWS . . . . .	694
Уменьшение числа согласованных с индексом столбцов . . . . .	696
Добавление дополнительных локальных предиктов . . . . .	698
Изменение статистики каталога . . . . .	699
<b>Глава 7—4. Использование EXPLAIN для повышения производительности SQL . . . . .</b>	<b>701</b>
Получение информации PLAN_TABLE с помощью EXPLAIN . . . . .	702
Создание PLAN_TABLE . . . . .	703
Заполнение и поддержание таблицы планов . . . . .	709
Упорядочивание строк таблицы планов . . . . .	710
Первые вопросы о доступе к данным . . . . .	711
Используется ли для доступа индекс? (ACCESSTYPE – I, II, N или MX) . . . . .	712
Используется ли для доступа несколько индексов? (ACCESSTYPE=M) . . . . .	712
Сколько столбцов индекса используется при согласованном просмотре? (MATCHCOLS=n) . . . . .	713
Можно ли выполнить запрос, используя только индекс? (INDEXONLY=Y) . . . . .	714
Возможен ли прямой доступ к строке? (PRIMARY_ACESSTYPE = D) . . . . .	714
Материализуются ли производная таблица или вложенное табличное выражение? . . . . .	718
Ограничивается ли просмотр определенными разделами? (PAGE_RANGE=Y) . . . . .	718
Какие виды предварительной выборки используются? (PREFETCH = L, S или пусто) . . . . .	719
Используется ли параллельная обработка данных? (PARALLELISM_MODE – I, C или X) . . . . .	720
Применяется ли сортировка? . . . . .	720
Преобразуется ли подзапрос в операцию объединения? . . . . .	721
Когда выполняются функции столбцов? (COLUMN_FN_EVAL) . . . . .	721
Интерпретация доступа к одной таблице . . . . .	721
Просмотр табличных пространств (ACCESSTYPE=R PREFETCH=S) . . . . .	722
Индексный путь доступа . . . . .	723
UPDATE с использованием индекса . . . . .	728
Объяснение доступа к нескольким таблицам . . . . .	728
Определения и примеры . . . . .	729
Объединение с вложенными циклами (METHOD=1) . . . . .	731
Объединение просмотром слияния (METHOD=2) . . . . .	733
Гибридное объединение (METHOD=4) . . . . .	736
Предварительная выборка данных . . . . .	738
Последовательная предварительная выборка (PREFETCH=S) . . . . .	738
Предварительная выборка списка (PREFETCH=L) . . . . .	739

Последовательное обнаружение во время выполнения . . . . .	741
Информация о сортировках . . . . .	743
Сортировка данных . . . . .	743
Сортировка списка RID . . . . .	744
Влияние сортировки на оператор OPEN CURSOR . . . . .	744
Обработка производных таблиц и вложенных табличных выражений . . . . .	745
Слияние . . . . .	745
Материализация . . . . .	746
Как с помощью EXPLAIN определить, когда происходит материализация . . . . .	748
Производительность слияния и материализации . . . . .	748
Оценка стоимости оператора . . . . .	749
Создание таблицы операторов . . . . .	749
Заполнение и поддержка таблицы операторов . . . . .	751
Получение строк из таблицы операторов . . . . .	751
Смысл категорий стоимости . . . . .	752
<b>Глава 7–5. Параллельные операции и производительность запросов</b>	<b>753</b>
Сравнение методов параллелизма . . . . .	754
Разрешение параллельной обработки . . . . .	757
Когда параллелизм не используется . . . . .	758
Интерпретация выходных данных EXPLAIN . . . . .	759
Как по столбцам таблицы PLAN_TABLE проверить использование параллелизма . . . . .	759
Примеры таблицы PLAN_TABLE, показывающей использование параллелизма . . . . .	760
Настройка параллельной обработки . . . . .	762
Запрещение параллелизма выполнения запроса . . . . .	763
<b>Глава 7–6. Программирование для утилиты ISPF (Interactive System Productivity Facility – интерактивная утилита производительности системы)</b>	<b>765</b>
Использование ISPF и командного процессора DSN . . . . .	765
Вызов одной программы SQL через ISPF и DSN . . . . .	766
Вызов нескольких программ SQL через ISPF и DSN . . . . .	767
Вызов нескольких программ SQL посредством ISPF и CAF . . . . .	768
<b>Глава 7–7. Программирование для утилиты подключения по вызову (CAF)</b>	<b>769</b>
Возможности и ограничения утилиты подключения по вызову . . . . .	769
Возможности при использовании CAF . . . . .	769
Требования CAF . . . . .	771
Как использовать CAF . . . . .	773
Краткое описание функций соединения . . . . .	774
Обращение к языковому интерфейсу CAF . . . . .	776
Основные свойства соединений CAF . . . . .	778
Описания функций CAF . . . . .	779
Функция CONNECT: синтаксис и использование . . . . .	781
Функция OPEN: синтаксис и использование . . . . .	785
Функция CLOSE: синтаксис и использование . . . . .	787
Функция DISCONNECT: синтаксис и использование . . . . .	789
Функция TRANSLATE: синтаксис и использование . . . . .	791
Сводка поведения CAF . . . . .	792
Примеры сценариев . . . . .	794

Одиночное задание с неявными соединениями . . . . .	794
Одиночное задание с явными соединениями . . . . .	794
Несколько заданий . . . . .	794
Обработчики для прикладной программы . . . . .	795
Обработчики ситуаций внимания . . . . .	795
Процедуры восстановления . . . . .	795
Сообщения об ошибках и набор данных DSNTRACE . . . . .	796
Коды возврата и коды причины CAF . . . . .	796
Коды подкомпоненты поддержки подсистемы (Х'00F3') . . . . .	797
Примеры программ . . . . .	797
Пример задания JCL, использующей CAF . . . . .	798
Пример программы на ассемблере, использующей CAF . . . . .	798
Загрузка и удаление языкового интерфейса CAF . . . . .	798
Установление соединения с DB2 . . . . .	799
Проверка кодов возврата и кодов причины . . . . .	800
Использование фиктивной точки входа DSNHLI . . . . .	804
Объявления переменных . . . . .	804
<b>Глава 7–8. Программирование для утилиты подключения служб управления восстановимыми ресурсами (RRSAF)</b> . . . . .	807
Возможности и ограничения RRSAF . . . . .	807
Возможности прикладных программ RRSAF . . . . .	807
Требования RRSAF . . . . .	809
Как использовать RRSAF . . . . .	810
Обращение к языковому интерфейсу RRSAF . . . . .	811
Основные свойства соединений RRSAF . . . . .	813
Сводка функций соединения . . . . .	814
Описание функций RRSAF . . . . .	815
Сводка поведения RRSAF . . . . .	839
Примеры сценариев . . . . .	841
Отдельное задание . . . . .	841
Несколько заданий . . . . .	841
Вызов функции SIGNON для повторного использования потока DB2 . . . . .	842
Переключение потоков DB2 между задачами . . . . .	842
Коды возврата и коды причин RRSAF . . . . .	843
Примеры программ . . . . .	844
Пример задания JCL, использующего RRSAF . . . . .	844
Загрузка и удаление языкового интерфейса RRSAF . . . . .	844
Использование фиктивной точки входа DSNHLI . . . . .	845
Установление соединения с DB2 . . . . .	846
<b>Глава 7–9. Особенности программирования для CICS</b> . . . . .	849
Управление утилитой подключения CICS из прикладной программы . . . . .	849
Повышение показателя повторного использования потоков . . . . .	849
Проверка рабочего состояния утилиты подключения CICS . . . . .	850
<b>Глава 7–10. Приемы программирования: Вопросы и ответы</b> . . . . .	853
Задание уникального ключа для таблицы . . . . .	853
Просмотр ранее полученных данных . . . . .	853
Сохранение копии данных . . . . .	853
Получение с начала . . . . .	854
Получение с середины . . . . .	854
Получение данных в обратном порядке . . . . .	855
Обновление ранее полученных данных . . . . .	857

Обновление данных во время получения их из базы данных . . . . .	857
Обновление тысяч строк . . . . .	857
Получение тысяч строк . . . . .	858
Использование SELECT * . . . . .	858
Оптимизация получения небольшого количества строк . . . . .	858
Добавление данных в конец таблицы . . . . .	859
Перевод требований конечных пользователей в операторы SQL . . . . .	859
Изменение определения таблицы . . . . .	859
Хранение данных не в табличном формате . . . . .	860
Поиск нарушенного реляционного или проверочного ограничения . . . . .	860

---

## Глава 7–1. Кодирование динамического SQL в прикладных программах

Прежде чем принимать решение об использовании динамического SQL, следует оценить, что больше подходит для вашей прикладной программы – статический SQL или динамический SQL.

Для большинства пользователей DB2 *статический SQL* – программа, встроенная в программу на языке хоста и связанная перед запуском этой программы – прямой и эффективный путь к данным DB2. Статический SQL можно использовать, когда еще до запуска прикладной программы известно, какие операторы SQL ей придется выполнять.

*Динамический SQL* подготавливает и выполняет операторы SQL прямо в программе, во время ее работы. Существует четыре вида динамического SQL:

- Встроенный динамический SQL

Прикладная программа помещает исходный текст SQL в переменные хоста и включает операторы PREPARE и EXECUTE, которые сообщают DB2, что во время выполнения надо подготовить и выполнить содержимое этих переменных хоста. Программы, содержащие встроенный динамический SQL, необходимо прокомпилировать и связывать.

- Интерактивный SQL

Пользователь вводит операторы SQL через SPUFI. DB2 подготавливает и выполняет эти операторы, как операторы динамического SQL.

- Отложенный встроенный SQL

Операторы отложенного встроенного SQL не являются полностью ни статическими, ни динамическими. Как и статические операторы, операторы отложенного встроенного SQL встраиваются в прикладные программы, но подготавливаются они во время выполнения, как динамические операторы. DB2 обрабатывает операторы отложенного встроенного SQL в соответствии с правилами времени связывания. Например, DB2 использует идентификатор авторизации и спецификатор, который во время связывания был задан, как владелец плана или пакета. Операторы отложенного встроенного SQL используются для доступа к удаленным данным по собственному протоколу DB2.

- Динамический SQL, выполняемый через функции ODBC

В прикладной программе содержатся вызовы функций ODBC, которые передают операторы динамического SQL как аргументы. Программы, использующие вызовы функций CLI, не нуждаются в прокомпиляции и связывании. Информацию о ODBC смотрите в руководстве *DB2 ODBC Guide and Reference*.

Некоторые советы по использованию статического и динамического SQL приводятся в разделе “Выбор между статическим и динамическим SQL” на стр. 544.

В остальной части этой главы рассказывается, как кодировать динамический SQL в прикладных программах, содержащих три вида операторов SQL:

- “Динамический SQL без операторов SELECT” на стр. 555. К числу таких операторов относятся DELETE, INSERT и UPDATE.
- “Динамический SQL для операторов SELECT с фиксированным списком” на стр. 559. Оператор SELECT называют оператором *с фиксированным списком*, если заранее известно число и тип элементов данных в каждой из строк результата.
- “Динамический SQL для операторов SELECT с переменным списком” на стр. 562. Оператор SELECT является оператором *с переменным списком*, если нельзя сказать заранее, сколько элементов данных он допускает и каких типов будут эти данные.

## Выбор между статическим и динамическим SQL

В этом разделе содержится следующая информация, которая может помочь решить, следует ли использовать в прикладной программе операторы динамического SQL:

- “Переменные хоста делают статический SQL гибким”
- “Динамический SQL абсолютно гибок” на стр. 545
- “Что делает прикладная программа, использующая динамический SQL” на стр. 545
- “Чего не может динамический SQL” на стр. 545
- “Производительность статического и динамического SQL” на стр. 545
- “Кэширование операторов динамического SQL” на стр. 547
- “Ограничение динамического SQL при помощи утилиты ограничения ресурсов” на стр. 552
- “Выбор языка хоста для прикладных программ с динамическим SQL” на стр. 554

## Переменные хоста делают статический SQL гибким

При использовании статического SQL нельзя изменять форму операторов SQL, не внося изменений в программу. Тем не менее можно повысить гибкость этих операторов, используя переменные хоста.

В следующем примере оператор UPDATE может изменять оклад каждого из сотрудников. Во время связывания вам известно, что заработка плата должна быть изменена, но до времени выполнения вы не знаете, у кого из сотрудников она будет изменена и на какую сумму.

```

01 IOAREA.
  02 EMPID          PIC X(06).
  02 NEW-SALARY      PIC S9(7)V9(2) COMP-3.
:
(Dругие объявления)
READ CARDIN RECORD INTO IOAREA
  AT END MOVE 'N' TO INPUT-SWITCH.
:
(Dругие операторы COBOL)
  EXEC SQL
    UPDATE DSN8610.EMP
      SET SALARY = :NEW-SALARY
        WHERE EMPNO = :EMPID
          END-EXEC.

```

Ни оператор (UPDATE), ни его основная структура не изменяются, однако результаты оператора UPDATE будут зависеть от введенного значения.

## Динамический SQL абсолютно гибок

Что, если программа должна использовать операторы SQL разных типов и структур? Если типов и структур настолько много, что программа не может содержать в себе модель каждого из них, возможно, в ней надо использовать динамический SQL.

Один из примеров такой программы – Query Management Facility (QMF), предоставляющая альтернативный интерфейс для DB2, который принимает почти все операторы SQL. Другой пример – SPUFI; она принимает операторы SQL из входного набора данных, а затем обрабатывает и выполняет их динамически.

## Чего не может динамический SQL

Динамически можно использовать только небольшую часть операторов SQL. Информацию о том, какие из операторов DB2 SQL можно подготовить динамически,смотрите в таблице в разделе Приложение G, “Характеристики операторов SQL в DB2 for OS/390” на стр. 967.

## Что делает прикладная программа, использующая динамический SQL

Программа, использующая динамический SQL, должна принимать в качестве ввода или генерировать оператор SQL в форме строки символов.

Программирование упростится, если можно спланировать программу так, чтобы она не использовала операторы SELECT или использовала только такие операторы SELECT, которые возвращают известное число значений с известными типами. В наиболее общем случае, когда об операторах SQL, которые будут выполняться, заранее ничего не известно, программа обычно выполняет следующие этапы:

1. Транслирует входные данные, включая все маркеры параметров, в оператор SQL
2. Подготавливает оператор SQL к выполнению и получает описание таблицы результатов
3. Получает для операторов SELECT основную память в объеме, достаточном для хранения полученных данных
4. Выполняет оператор или производит выборку строк данных
5. Обрабатывает возвращаемую информацию
6. Обрабатывает коды возврата SQL.

## Производительность статического и динамического SQL

Для доступа к данным DB2 оператору SQL нужен путь доступа. Два фактора, в значительной степени влияющие на производительность оператора SQL – это время, которое DB2 использует для определения пути доступа во время выполнения, и эффективность этого пути доступа. DB2 определяет путь доступа для оператора в один из двух моментов:

- При связывании плана или пакета, содержащего оператор SQL

- При выполнении оператора SQL

Момент, когда DB2 определяет путь доступа, зависит от следующих факторов:

- Выполняется ли оператор статически или динамически
- Содержит ли оператор входные переменные хоста

### **Операторы статического SQL без входных переменных хоста**

Для операторов статического SQL, не содержащих входных переменных хоста, DB2 определяет путь доступа при связывании плана или пакета. Это приводит к максимальной производительности, поскольку при выполнении программы путь доступа уже определен.

### **Операторы статического SQL с входными переменными хоста**

Для таких операторов время определения DB2 пути доступа зависит от того, какая задана опция связывания – NOREOPT(VARS) или REOPT(VARS). По умолчанию устанавливается NOREOPT(VARS).

Если задать NOREOPT(VARS), DB2 определяет путь доступа во время связывания, то есть так же, как и при отсутствии входных переменных.

Если задать REOPT(VARS), DB2 определяет путь доступа во время связывания, а затем еще раз во время выполнения, используя значения в следующих типах входных переменных:

- Переменные хоста
- Маркеры параметров
- Специальные регистры

Это означает, что DB2 должна затратить дополнительное время на определение пути доступа для операторов во время выполнения, но если DB2 с использованием значений этих переменных определит существенно лучший путь доступа, может быть достигнуто общее увеличение производительности. В общем случае использование REOPT(VARS) может привести к такой же производительности операторов статического SQL с входными переменными, как у операторов динамического SQL с константами. Дополнительную информацию об использовании REOPT(VARS) для изменения путей доступа смотрите в разделе “Эффективное использование переменных хоста” на стр. 682.

### **Операторы динамического SQL**

Для операторов динамического SQL DB2 определяет путь доступа во время выполнения при подготовке каждого из операторов. Это может привести к снижению производительности по сравнению с операторами статического SQL. Однако если один и тот же оператор SQL выполняется часто, число подготовок динамических операторов можно сократить, используя кэш динамических операторов. Дополнительную информацию смотрите в разделе “Производительность статического и динамического SQL” на стр. 545.

**Операторы динамического SQL с входными переменными хоста:** В общем случае при связывании прикладных программ, содержащих операторы динамического SQL с входными переменными хоста, рекомендуется использовать опцию REOPT(VARS). При этом для снижения непроизводительных затрат следует кодировать операторы PREPARE. При

использовании REOPT(VARS) DB2 подготавливает оператор SQL одновременно с выполнением операций OPEN или EXECUTE для этого оператора. Это означает, что DB2 обрабатывает оператор, как при задании DEFER(PREPARE). Однако если вы выполняете в программе оператор DESCRIBE перед оператором PREPARE или используете оператор PREPARE с параметром INTO, DB2 подготавливает этот оператор дважды. В первый раз DB2 определяет путь доступа без использования значений входных переменных, а во второй раз – с использованием. Такая дополнительная подготовка может снизить производительность. Для оператора, использующего указатель, двойной подготовки можно избежать, поместив в программе оператор DESCRIBE после оператора OPEN.

Если используется прогностическое ограничение ресурсов и оператор динамического SQL, связанный с REOPT(VARS), превышает порог предупреждения, прикладная программа не получит SQLCODE предупреждения. Однако если оператор превысит порог ошибки прогностического ограничения ресурсов, прикладная программа получит ошибки от оператора EXECUTE или OPEN SQLCODE.

## Кэширование операторов динамического SQL

Поскольку возможности оптимизации SQL в DB2 выросли, выросла и стоимость подготовки оператора динамического SQL. Прикладным программам, использующим динамический SQL, может понадобиться идти на эти расходы несколько раз. Выполнив операцию принятия, программа должна будет выполнить другой оператор PREPARE, если этот же оператор SQL встретится еще раз. Для оператора SELECT определенную помочь оказывает возможность объявления указателя с условием WITH HOLD, но для этого он должен быть открыт в точке принятия. WITH HOLD также приводит к определенным блокировкам, которые будут наложены на все объекты, от которых находится в зависимости подготовленный оператор. Кроме того, WITH HOLD никак не влияет на прочие операторы SQL (не операторы SELECT).

DB2 может сохранить подготовленные динамические операторы в кэше. Это единый кэш для всей подсистемы DB2 в пуле EDM, который все прикладные процессы могут использовать для хранения и считывания подготовленных динамических операторов. Когда оператор SQL подготовлен и автоматически сохранен в кэше, последующий запрос на подготовку того же самого оператора SQL может обойтись без дорогостоящего процесса подготовки, используя оператор из кэша. Операторы в кэше могут использоваться совместно различными потоками, планами и пакетами.

Например:

PREPARE STMT1 FROM ... EXECUTE STMT1 COMMIT : PREPARE STMT1 FROM ... EXECUTE STMT1 COMMIT :	Оператор подготовлен; подготовленный оператор помещен в кэш.  Идентичный оператор. DB2 использует подготовленный оператор из кэша.
--	--

**Кэшируемые операторы:** Кэширование возможно для следующих операторов SQL:

```
SELECT  
UPDATE  
INSERT  
DELETE
```

Можно помещать в кэш распределенные и локальные операторы SQL. Подготовленные динамические операторы, использующие доступ по собственному протоколу DB2, допускают кэширование.

**Ограничения:** Несмотря на то, что статические операторы, использующие доступ по собственному протоколу DB2, на удаленном узле являются динамическими, они не подходят для кэширования.

Операторы в планах или пакетах, связанных с опцией REOPT(VARS), не подходят для кэширования. Дополнительную информацию о REOPT(VARS) смотрите в разделе “Как опция связывания REOPT(VARS) влияет на динамический SQL” на стр. 576.

Подготовленные операторы нельзя использовать совместно членам группы совместного использования данных. Поскольку у каждого из компонентов есть свой собственный пул EDM, оператор в кэше на одном из членов недоступен прикладной программе, работающей на другом члене группы.

## Использование кэша динамических операторов

Чтобы сделать возможным кэширование подготовленных операторов, введите YES в поле CACHE DYNAMIC SQL панели установки DSNTIP4. Дополнительную информацию смотрите в разделе 2 руководства *DB2 Installation Guide*.

### Условия для совместного использования операторов

Предположим, что S1 и S2 – исходные операторы, а P1 – подготовленная версия S1. P1 находится в кэше подготовленных операторов.

Чтобы DB2 могла использовать оператор P1 вместо подготовки оператора S2, должны быть удовлетворены следующие условия:

- S1 и S2 должны быть идентичными. Эти операторы должны совпадать посимвольно и иметь одинаковую длину. При невыполнении одного из этих условий DB2 не сможет использовать оператор из кэша.

Например, если и S1, и S2 имеют вид

```
'UPDATE EMP SET SALARY=SALARY+50 '
```

то DB2 может использовать P1 вместо подготовки S2. Однако если S1 имеет вид

```
'UPDATE EMP SET SALARY=SALARY+50 '
```

а S2 имеет вид

```
'UPDATE EMP SET SALARY=SALARY+50 '
```

DB2 не может использовать P1.

В этом случае DB2 готовит S2 и помещает подготовленную версию S2 в кэш.

- Идентификатор авторизации, использованный для подготовки S1, должен быть использован и для подготовки S2:

- Если план или пакет используют стратегию запуска, ID авторизации — это текущее значение SQLID.

Для вторичных идентификаторов авторизации:

- У прикладного процесса, который ведет поиск в кэше, должен быть такой же список вторичных идентификаторов авторизации, что и у процесса, поместившего запись в кэш, или первый список должен включать в себя второй.
  - Если процесс, который первоначально подготовил оператор и поместил его в кэш, использовал для подготовки одну из привилегий, поддерживаемых первичным идентификатором авторизации, этот идентификатор должен либо быть частью списка вторичных идентификаторов авторизации ведущего поиск в кэше процесса, либо быть первичным идентификатором авторизации этого процесса.
  - Если план или пакет используют стратегию связывания, ID авторизации — это ID владельца плана. Для потока сервера DDF идентификатор авторизации — это ID владельца пакета.
  - Если пакет использует стратегию определения, ID авторизации — это ID владельца пользовательской функции или хранимой процедуры.
  - Если пакет использует стратегию вызова, ID авторизации — это ID, под которым выполнялся оператор, вызвавший пользовательскую функцию или хранимую процедуру.

Объяснение стратегий связывания, запуска, определения и вызова смотрите в разделе “Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES” на стр. 457.

- При связывании плана или пакета, содержащего S2, значения следующих опций связывания должны быть теми же самыми, что и при связывании плана или пакета, содержащего S1:

CURRENTDATA  
DYNAMICRULES  
ISOLATION  
SQLRULES  
QUALIFIER

- При подготовке S2 значения специальных регистров CURRENT DEGREE, CURRENT RULES и CURRENT PRECISION должны быть теми же самими, что и при подготовке S1.

## Сохранение подготовленных операторов после точек принятия

Опция связывания KEEPDYNAMIC(YES) позволяет сохранять динамические операторы после точки принятия для процесса прикладной программы.

Прикладная программа может выполнить PREPARE для оператора один раз, а остальные PREPARE для этого оператора пропустить. На рис. 128 на стр. 550 показана прикладная программа, использующая KEEPDYNAMIC(YES).

```

PREPARE STMT1 FROM ...      Оператор подготовлен.
EXECUTE STMT1
    COMMIT
:
EXECUTE STMT1              Прикладная программа не выполняет PREPARE.
    COMMIT
:
EXECUTE STMT1              Снова нет необходимости в PREPARE.
    COMMIT

```

*Рисунок 128. Динамического SQL, использующий опцию связывания KEEPDYNAMIC(YES)*

Для понимания действия опции связывания KEEPDYNAMIC важно понять разницу между исполняемой формой оператора динамического SQL – **подготовленным оператором** и оператором в форме строки символов – **строкой оператора**.

**Взаимосвязь между KEEPDYNAMIC(YES) и кэшированием операторов:**

Если кэш динамических операторов не активен, и вы запускаете прикладную программу, связанную с опцией KEEPDYNAMIC(YES), после операции принятия для подготовленного оператора DB2 сохраняет только строку оператора. При последующих OPEN, EXECUTE или DESCRIBE DB2 должна подготовить этот оператор еще раз перед тем, как выполнить запрошенную операцию. Это показано на рис. 129.

```

PREPARE STMT1 FROM ...      Оператор подготовлен и помещен в память.
EXECUTE STMT1
    COMMIT
:
EXECUTE STMT1              Прикладная программа не выполняет PREPARE.
    COMMIT                  DB2 подготавливает оператор еще раз.
:
EXECUTE STMT1              Снова нет необходимости в PREPARE.
    COMMIT

```

*Рисунок 129. Использование KEEPDYNAMIC(YES) при неактивном кэше динамических операторов*

При активном кэше динамических операторов и запуске прикладной программы, связанной с KEEPDYNAMIC(YES), DB2 сохраняет как копию подготовленного оператора, так и копию строки оператора. Для прикладного процесса подготовленный оператор кэшируется локально. Вероятно, что этот оператор также кэшируется глобально в пуле EDM, чтобы им могли воспользоваться другие прикладные процессы. Если прикладная программа выполняет OPEN, EXECUTE или DESCRIBE после операции принятия, этот прикладной процесс использует свою локальную копию подготовленного оператора, чтобы избежать подготовки и поиска в кэше. На рис. 130 на стр. 551 показан этот процесс.

PREPARE STMT1 FROM ...	Оператор подготовлен и помещен в память.
EXECUTE STMT1	
COMMIT	
⋮	
EXECUTE STMT1	Прикладная программа не выполняет PREPARE.
COMMIT	DB2 использует подготовленный оператор из памяти.
⋮	
EXECUTE STMT1	Снова нет необходимости в PREPARE.
COMMIT	DB2 использует подготовленный оператор из памяти.
⋮	
PREPARE STMT1 FROM ...	Оператор подготовлен и помещен в память.

*Рисунок 130. Использование KEEPDYNAMIC(YES) при активном кэше динамических операторов*

Локальный экземпляр подготовленного оператора SQL хранится в памяти *ssnmDBM1*, пока не произойдет одно из следующих событий:

- Закончится прикладной процесс.
- Будет произведена операция отката.
- Прикладная программа выполнит явно оператор PREPARE с тем же самым именем оператора.

Если прикладная программа выполнит оператор PREPARE с тем же самым именем оператора SQL, с которым связан динамический оператор в кэше, этот оператор исключается из кэша, и DB2 готовит новый оператор.

- Оператор будет удален из памяти, если он не был использован в течение некоторого времени, а число хранимых операторов динамического SQL достигло заданного во время установки предела.

**Ошибки неявной обработки операторов PREPARE:** Если оператор требуется во время жизни прикладного процесса, но был удален из локального кэша, DB2, возможно, удастся извлечь его из глобального кэша. Если оператора нет в глобальном кэше, DB2 должна неявно образом подготовить этот оператор еще раз. У прикладной программы нет необходимости выполнять оператор PREPARE. Однако если прикладная программа выполняет для этого оператора OPEN, EXECUTE или DESCRIBE, эта прикладная программа должна быть готова к тому, что DB2 подготовит оператор *неявно*. Все ошибки, произошедшие во время такой подготовки, возвращаются при операциях OPEN, EXECUTE или DESCRIBE.

**Как KEEPDYNAMIC влияет на прикладные программы, использующие распределенные данные:** Если реквестер прикладной программы не выполняет PREPARE после COMMIT, пакет на сервере DB2 for OS/390 должен быть связан с KEEPDYNAMIC(YES). Если и реквестер, и сервер – системы DB2 for OS/390, реквестер DB2 полагает, что значение KEEPDYNAMIC для пакета на сервере то же, что и значение для плана на реквестере.

Опция KEEPDYNAMIC оказывает влияние на производительность для клиентов DRDA, которые задают указатели с условием WITH HOLD:

- Если указано KEEPDYNAMIC(NO), когда клиент DRDA выполняет для указателя SQL CLOSE, требуется отдельное сетевое сообщение.

- Если указано KEEPDYNAMIC(YES), сервер DB2 for OS/390 автоматически закрывает указатель при обнаружении SQLCODE +100, то есть клиент не должен посылать отдельное сообщение для закрытия этого указателя. Это уменьшает сетевой трафик для прикладных программ DRDA, использующих указатели с условием WITH HOLD. Это также уменьшает продолжительность блокировок, связанных с таким указателем.

**Использование RELEASE(DEALLOCATE) с KEEPDYNAMIC(YES) и динамическим кэшированием операторов:** Информацию о взаимодействиях между опциями связывания RELEASE(DEALLOCATE) и KEEPDYNAMIC(YES) смотрите в разделе “Опция RELEASE и кэширование динамических операторов” на стр. 370.

**Особенности совместного использования данных:** Если у одного из членов группы совместного использования данных кэширование разрешено, а у другого – запрещено, и программа связана с KEEPDYNAMIC(YES), DB2 должна неявно подготовить оператор еще раз, если этот оператор выполняется на члене без кэша. Это приводит к некоторому снижению производительности.

---

## Ограничение динамического SQL при помощи утилиты ограничения ресурсов

Утилита ограничения ресурсов ограничивает процессорное время, которое может потребовать выполнение оператора SQL, что предотвращает выполнение длительных действий операторами SQL. Функция прогностического ограничения утилиты ограничения ресурсов перед запуском операторов SQL дает оценку стоимости их обработки. Чтобы предсказать стоимость оператора SQL, выполните операцию EXPLAIN, которая поместит информацию о стоимости в таблицу DSN\_STATEMENT\_TABLE. Информацию о создании, заполнении и интерпретации содержимого DSN\_STATEMENT\_TABLE смотрите в разделе “Оценка стоимости оператора” на стр. 749.

Утилита ограничения ресурсов управляет только операторами динамического SQL SELECT, UPDATE, DELETE и INSERT. Для каждого из используемых в программе операторов динамического SQL действуют одни и те же ограничения. Ограничение может быть пределом *реактивного* ограничения или пределом *прогностического* ограничения. Если оператор превышает предел реактивного ограничения, возвращается код ошибки SQL. Если оператор превышает предел прогностического ограничения, возвращается код ошибки или код предупреждения SQL. Дополнительная информация о кодах SQL прогностического ограничения приводится в разделе “Как задать в программе обработку прогностического ограничения” на стр. 553.

Системный администратор установить ограничения для отдельных планов или пакетов, для отдельных пользователей или для всех пользователей, у которых нет персональных ограничений.

Чтобы добавить, удалить или изменить записи в таблице ограничения ресурсов, следуйте процедурой, определенным для вашей системы. Дополнительную информацию о таблицах ограничения ресурсов смотрите в Разделе 5 (Том 2) руководства *DB2 Administration Guide*.

## Как задать в программе обработку реактивного ограничения

Когда оператор динамического SQL превышает порог реактивного ограничения, прикладная программа получает SQLCODE –905. После этого прикладная программа должна определить, что делать дальше.

Если неудавшийся оператор использовал указатель SQL, положение этого указателя остается неизменным. Прикладная программа может затем закрыть этот указатель. Прочие операции с этим указателем не выполняются, они приводят к ошибке SQL с тем же кодом.

Если неудавшийся оператор SQL не использовал указатель, перед возвратом кода ошибки прикладной программе все изменения, внесенные этим оператором, отменяются. Прикладная программа может либо выполнить другой оператор SQL, либо принять всю уже сделанную работу.

## Как задать в программе обработку прогностического ограничения

Если на вашей установке используется прогностическое ограничение, необходимо модифицировать прикладные программы, чтобы они производили проверку на коды SQLCODE +495 и –495, которые прогностическое ограничение может генерировать при выполнении оператора PREPARE. Код SQLCODE +495 в сочетании с отложенной подготовкой требует от DB2 выполнения некоторой специальной обработки, чтобы этот новый код SQLCODE предупреждения не повлиял на существующие программы.

Информацию о настройке утилиты ограничения ресурсов для прогностического ограничения смотрите в Разделе 5 (Том 2) руководства *DB2 Administration Guide*.

### Обработка кода SQLCODE +495

Если реквестер использует отложенную подготовку, момент получения программой кода SQLCODE +495 зависит от наличия маркеров параметров. Если маркеры параметров присутствуют, DB2 не может выполнить обработку PREPARE, OPEN и FETCH в одном сообщении. Если возвращен SQLCODE +495, обработка OPEN или FETCH не производится до тех пор, пока ее не запросит прикладная программа.

- Если присутствуют маркеры параметров, код +495 возвращает операция OPEN (а не PREPARE).
- Если маркеров параметров нет, код +495 возвращает PREPARE.

Обычно при отложенной подготовке реквестеру возвращаются PREPARE, OPEN и первая операция FETCH для данных. При получении кода +495 от утилиты прогностического ограничения ресурсов идеально было бы иметь возможность выбора, надо ли выполнять операции OPEN и FETCH для данных. Но у старых реквестеров такой возможности нет. Прогностическое ограничение поддерживается в DRDA, начиная с уровня 4.

Поддержка DRDA для прогностического ограничения включена в DB2 for OS/390 Версия 6 и DB2 Connect™ Версии 5.2 с соответствующими опциями. Все прочие реквестеры, что касается поддержки прогностического ограничения через DRDA, можно считать устаревшими.

## **Использование прогностического ограничения для устаревших реквестеров DRDA**

Если реквестеру возвращен SQLCODE +495, обработка OPEN продолжается, но первый блок данных при операции OPEN не возвращается. Таким образом, если прикладная программа не продолжает обработку запроса, вы потеряете стоимость обработки OPEN.

## **Использование прогностического ограничения на поддерживающих реквестерах**

Если ваша программа не откладывает подготовку, реквестеру возвращается код SQLCODE +495, и обработка OPEN не производится.

Если же программа откладывает подготовку, она получает код +495 в обычное время (при OPEN или PREPARE). Если при отложенной подготовке используются маркеры параметров, вы, как обычно, получите код +495 в момент обработки OPEN. Однако обмена дополнительными сообщениями при этом не происходит.

**Рекомендация:** Не используйте отложенную подготовку для программ, которые используют маркеры параметров и прогностическое ограничение ресурсов на стороне сервера.

---

## **Выбор языка хоста для прикладных программ с динамическим SQL**

Программы, использующие динамический SQL, обычно пишут на ассемблере, C, PL/I и на версиях COBOL, кроме OS/VS COBOL. Все операторы, кроме SELECT с переменным списком, можно писать на любом языке, поддерживаемом DB2. Программы, содержащие оператор SELECT с переменным списком, трудно писать на языке FORTRAN, поскольку такие программы нельзя запустить без помощи подпрограммы, которая будет управлять адресными переменными (указателями) и размещением памяти.

Для большинства примеров в этом разделе используется PL/I. В разделе “Использование динамического SQL в языке COBOL” на стр. 577 показана техника использования языка COBOL. Более длинные примеры можно посмотреть среди программ примеров:

- |                 |  |
|-----------------|--|
| <b>DSNTEP2</b>  | Обрабатывает динамически как операторы SELECT, так и другие операторы. (PL/I). |
| <b>DSNTIAD</b>  | Обрабатывает динамически операторы без операторов SELECT. (Ассемблер).         |
| <b>DSNTIAUL</b> | Обрабатывает динамически операторы SELECT. (Ассемблер).                        |

Эти примеры программ находятся в библиотеке префикс. SDSNSAMP. Можно посмотреть эти программы на экране или напечатать их при помощи ISPF, IEBPTPCH или вашей программы печати.

---

## Динамический SQL без операторов SELECT

При простейшем способе использования динамического SQL операторы SELECT динамически не используются. Поскольку нет необходимости в динамическом выделении основной памяти, программу можно написать на любом языке хоста, включая OS/VS COBOL и FORTRAN. На рис. 215 на стр. 913 приводится пример программы на языке С с динамическим SQL без операторов SELECT.

Программа должна выполнить следующие действия:

1. Включить SQLCA. Требования для области связи SQL (SQLCA) такие же, как и для статических операторов SQL.
2. Загрузить входной оператор SQL в область данных. Процедура создания или чтения входного оператора SQL здесь не обсуждается; этот оператор зависит от среды и источников информации. Можно либо прочитать операторы SQL полностью, либо получить информацию для построения оператора из наборов данных, от пользователя за терминалом, из ранее заданных переменных программы или из таблиц в базе данных.

Если предпринять попытку динамически выполнить оператор SQL, не поддерживающий DB2, вы получите ошибку SQL.

3. Выполнить оператор. Можно использовать любой из следующих способов:
  - “Динамическое выполнение с использованием EXECUTE IMMEDIATE”
  - “Динамическое выполнение с использованием PREPARE и EXECUTE” на стр. 556.
4. Обработать все возможные ошибки. Требования – те же самые, что и для статических операторов SQL. Код возврата последнего выполненного оператора SQL записывается в переменные хоста SQLCODE и SQLSTATE или в соответствующих полях в SQLCA. Информацию относительно SQLCA и ее полей смотрите в разделе “Проверка выполнения операторов SQL” на стр. 116.

## Динамическое выполнение с использованием EXECUTE IMMEDIATE

Предположим, что вы разработали программу для чтения с терминала операторов SQL DELETE, например, таких:

```
DELETE FROM DSN8610.EMP WHERE EMPNO = '000190'  
DELETE FROM DSN8610.EMP WHERE EMPNO = '000220'
```

После чтения оператора программа немедленно выполняет его.

Напомним, что перед тем как использовать статические операторы SQL, их необходимо подготовить (прекомпилировать и связать). Заблаговременная подготовка операторов динамического SQL невозможна. Оператор SQL EXECUTE IMMEDIATE запускает подготовку и выполнение оператора SQL динамически, во время выполнения.

## **Оператор EXECUTE IMMEDIATE**

Для выполнения этих операторов:

```
< Считываем оператор DELETE в переменную хоста DSTRING.>
      EXEC SQL
      EXECUTE IMMEDIATE :DSTRING;
```

DSTRING – переменная хоста типа символьной строки. EXECUTE IMMEDIATE вызывает немедленную подготовку и выполнение оператора DELETE.

### **Переменная хоста**

DSTRING – имя переменной хоста; это не зарезервированное слово DB2. В ассемблере, COBOL и С ее следует объявлять как переменную типа строка переменной длины. В языке FORTRAN она должна быть переменной типа строка фиксированной длины. В PL/I она может быть либо переменной типа строка, как фиксированной длины, так и переменной длины, либо любым выражением PL/I, при вычислении которого получается символьная строка. Дополнительную информацию о переменных типа строка переменной длины смотрите в разделе “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145.

## **Динамическое выполнение с использованием PREPARE и EXECUTE**

Предположим, что вы хотите выполнить операторы DELETE несколько раз, используя список номеров сотрудников. Если бы вы писали оператор DELETE как статический, вы сделали бы это так:

```
< Читаем значение для EMP из списка. >
DO UNTIL (EMP = 0);
      EXEC SQL
      DELETE FROM DSN8610.EMP WHERE EMPNO = :EMP ;
< Читаем значение для EMP из списка. >
      END;
```

Этот цикл повторяется до тех пор, пока при чтении EMP не будет получен 0.

Если бы вы знали заранее, что будет использоваться только оператор DELETE и только таблица DSN8610.EMP, можно было бы использовать более эффективный статический SQL. Но предположим, что есть несколько различных таблиц со строками, идентифицирующимися номерами сотрудников, и что пользователи вводят для удаления как имя таблицы, так и список сотрудников. Переменные могут содержать номера сотрудников, но не могут содержать имена таблиц, поэтому весь оператор следует конструировать и выполнять динамически. В вашу программу надо внести следующие изменения:

- Использовать маркеры параметров вместо переменных хоста
- Использовать оператор PREPARE
- Использовать EXECUTE вместо EXECUTE IMMEDIATE.

## Использование маркеров параметров

В динамических операторах SQL нельзя использовать переменные хоста. Поэтому оператор SQL с переменными хоста нельзя выполнить динамически. Каждую из переменных хоста в операторе надо заменить на *маркер параметра*, обозначаемый вопросительным знаком (?).

Можно указать DB2, что маркер параметра представляет собой переменную хоста с определенным типом данных, задав маркер параметра как аргумент функции CAST. Во время выполнения оператора DB2 преобразует переменную хоста в тип данных в функции CAST. Маркер параметра, включенный вами в функцию CAST, называется *типовизированным* маркером параметра. Маркер параметра без функции CAST называется *нетипизированным* маркером параметра.

Поскольку DB2 может вычислить оператор SQL с типизированными маркерами параметров эффективнее, чем оператор с нетипизированными маркерами параметров, во всех возможных случаях рекомендуется использовать типизированные маркеры параметров. При определенных условиях их использование абсолютно необходимо. Правила использования типизированных и нетипизированных маркеров параметровсмотрите в Главе 6 руководства *DB2 SQL Reference*.

**Пример:** Для подготовки оператора:

```
DELETE FROM DSN8610.EMP WHERE EMPNO = :EMP ;
```

надо составить примерно такую строку:

```
DELETE FROM DSN8610.EMP WHERE EMPNO = CAST(? AS CHAR(6))
```

При выполнении подготовленного оператора вы связываете переменную хоста :EMP с маркером параметра. Предположим, что S1 – подготовленный оператор. В таком случае оператор EXECUTE будет выглядеть так:

```
EXECUTE S1 USING :EMP;
```

## Оператор PREPARE

Можно представить себе PREPARE и EXECUTE, как выполнение EXECUTE IMMEDIATE в два этапа. Первый этап, PREPARE, превращает строку символов в оператор SQL, а затем присваивает ему имя по вашему выбору.

Например, пусть переменная :DSTRING имеет значение “DELETE FROM DSN8610.EMP WHERE EMPNO = ?”. Чтобы подготовить оператор SQL из этой строки и присвоить ему имя S1, введите:

```
EXEC SQL PREPARE S1 FROM :DSTRING;
```

В подготовленном операторе все еще стоит маркер параметра, значение для которого необходимо задать во время выполнения оператора. Когда оператор подготовлен, имя таблицы уже определено, но маркер параметра позволяет многократно выполнять этот оператор с различными значениями номеров сотрудников.

## **Оператор EXECUTE**

EXECUTE выполняет подготовленный оператор SQL, указывая имя списка одной или нескольких переменных хоста или структур хоста, где содержатся значения для всех маркеров параметров.

После подготовки оператора в пределах одной и той же единицы работы его можно выполнять несколько раз. В большинстве случаев COMMIT или ROLLBACK разрушают подготовленные в единице работы операторы. Поэтому перед повторным выполнением их надо подготавливать еще раз. Однако если объявить указатель для динамического оператора с условием WITH HOLD, операция принятия не разрушит подготовленный оператор, если указатель все еще будет открыт. Этот оператор можно выполнить в следующей единице работы без повторной подготовки.

Для однократного выполнения подготовленного оператора S1 с использованием значения параметра из переменной хоста :EMP введите:

```
EXEC SQL EXECUTE S1 USING :EMP;
```

## **Полный пример**

Мы начали обсуждение с примера цикла DO, в котором выполняется оператор SQL:

```
< Читаем значение для EMP из списка. >
DO UNTIL (EMP = 0);
    EXEC SQL
        DELETE FROM DSN8610.EMP WHERE EMPNO = :EMP ;
< Читаем значение для EMP из списка. >
    END;
```

Теперь можно написать эквивалентный пример для динамического оператора SQL:

```
< Заносим в DSTRING оператор с маркерами параметров.>
EXEC SQL PREPARE S1 FROM :DSTRING;
< Читаем значение для EMP из списка. >
DO UNTIL (EMPNO = 0);
    EXEC SQL EXECUTE S1 USING :EMP;
< Читаем значение для EMP из списка. >
    END;
```

Оператор PREPARE готовит оператор SQL и дает ему имя S1.  
Оператор EXECUTE несколько раз выполняет S1 с разными значениями EMP.

## **Несколько маркеров параметров**

Подготовленный оператор (в примере – S1) может содержать несколько маркеров параметров. В таком случае условие USING оператора EXECUTE задает список переменных структуры хоста. В этих переменных в соответствующем порядке могут содержаться значения, число и тип данных которых совпадают с числом и типами параметров в S1. Необходимо либо предварительно знать число и тип параметров и объявлять эти переменные в программе, либо использовать SQLDA (область дескриптора SQL).

## Использование DESCRIBE INPUT для помещения информации о маркере параметра в SQLDA

Оператор DESCRIBE INPUT можно использовать, чтобы позволить DB2 помещать в SQLDA информацию о типах данных для маркеров параметров.

Перед тем как выполнить DESCRIBE INPUT, необходимо разместить SQLDA с числом SQLVAR, достаточным для задания всех маркеров параметров в операторах SQL, которые вы хотите описать.

После выполнения DESCRIBE INPUT прикладная программа кодируется, как и любая другая прикладная программа, в которой подготовленный оператор выполняется с использованием SQLDA. Сначала вы получаете адреса входных переменных хоста и их переменные индикаторов и вставляете эти адреса в поля SQLDATA и SQLIND. Затем подготовленный оператор SQL выполняется.

Например, предположим, что вы хотите динамически выполнить следующий оператор:

```
DELETE FROM DSN8610.EMP WHERE EMPNO = ?
```

Код для настройки SQLDA, получения информации о параметре с использованием DESCRIBE INPUT и выполнения оператора выглядит примерно так:

```
SQLAPTR=ADDR(INSQLDA);          /* Получаем указатель в SQLDA */
SQLDAID='SQLDA';                /* Заполняем метку SQLDA */
SQLDABC=LENGTH(INSQLDA);        /* Заносим длину SQLDA */
SQLN=1;                         /* Указываем число SQLVAR */
SQLD=0;                          /* Инициализируем число использ. SQLVAR */
DO IX=1 TO SQLN;               /* Инициализируем SQLVAR */
      SQLTYPE(IX)=0;
      SQLLEN(IX)=0;
      SQLNAME(IX)='';
      END;
SQLSTMT='DELETE FROM DSN8610.EMP WHERE EMPNO = ?';
EXEC SQL PREPARE SQLOBJ FROM SQLSTMT;
EXEC SQL DESCRIBE INPUT SQLOBJ INTO :INSQLDA;
SQLDATA(1)=ADDR(HVEMP);         /* Получаем адрес входных данных */
SQLIND(1)=ADDR(HVEMPIND);       /* Получаем адрес индикатора */
EXEC SQL EXECUTE SQLOBJ USING DESCRIPTOR :INSQLDA;
```

---

## Динамический SQL для операторов SELECT с фиксированным списком

Оператор SELECT с фиксированным списком возвращает строки, содержащие известное число значений известного типа. При его использовании заранее точно известно, какие виды переменных хоста следует объявлять для сохранения результатов. (Противоположная ситуация, в которой заранее *неизвестно*, какая структура переменных хоста понадобится, описана в разделе “Динамический SQL для операторов SELECT с переменным списком” на стр. 562.)

Термин “фиксированный список” не подразумевает, что предварительно известно, сколько будет возвращено строк данных; тем не менее, вы должны знать число столбцов и типы данных этих столбцов. Оператор SELECT с

фиксированным списком возвращает таблицу результатов, содержащую произвольное число строк; ваша программа просматривает все эти строки по одной при помощи оператора FETCH. Каждая из удачных выборок возвращает одинаковое число значений, и типы данных этих значений совпадают. Поэтому переменные хоста можно задавать точно так же, как и для статического SQL.

Преимущество SELECT с фиксированным списком заключается в том, что его можно писать на любом поддерживаемом DB2 языке программирования. Для динамических операторов SELECT с переменным списком необходим ассемблер, C, PL/I или версия языка COBOL, отличная от OS/VS COBOL.

Пример программы на языке C, иллюстрирующей динамический SQL с операторами SELECT с фиксированным списком, смотрите на рис. 215 на стр. 913.

## Что должна делать ваша прикладная программа

Чтобы динамически выполнить оператор SELECT с фиксированным списком, ваша программа должна:

1. Включить SQLCA
2. Загрузить входной оператор SQL в область данных

Эти два этапа совпадают с описанными в разделе “Динамический SQL без операторов SELECT” на стр. 555.

3. Объявить указатель для имени оператора, как описано в разделе “Объявляем указатель для имени оператора” на стр. 561.
4. Подготовить этот оператор, как описано в разделе “Подготавливаем оператор” на стр. 561.
5. Открыть указатель, как описано в разделе “Открываем указатель” на стр. 561.
6. Произвести выборку строк из таблицы результатов, как описано в разделе “Выбираем строки из таблицы результатов” на стр. 561.
7. Закрыть указатель, как описано в разделе “Закрываем указатель” на стр. 562.
8. Обработать все полученные ошибки. Этот этап совпадает с этапом для статического SQL, за исключением числа и типа возможных ошибок.

Предположим, что ваша программа получает обновленные имена и телефонные номера, динамически выполняя операторы SELECT следующего вида:

```
SELECT LASTNAME, PHONENO FROM DSN8610.EMP  
WHERE ... ;
```

Программы считывают операторы с терминала, а пользователь определяет условие WHERE.

Как и в случае других операторов (не SELECT), программа помещает эти операторы в символьную переменную с переменной длиной под названием DSTRING. В конечном счете, из DSTRING следует подготовить оператор, но

сначала необходимо объявить указатель для этого оператора и присвоить ему имя.

### **Объявляем указатель для имени оператора**

Динамические операторы SELECT не могут использовать условие INTO; поэтому для того, чтобы поместить результаты в переменные хоста, следует использовать указатель. При объявлении этого указателя используйте имя оператора (назовем его STMT) и задайте имя собственно указателя (например, C1):

```
EXEC SQL DECLARE C1 CURSOR FOR STMT;
```

### **Подготавливаем оператор**

Теперь подготовим оператор (STMT) из DSTRING. Вот один из возможных операторов PREPARE:

```
EXEC SQL PREPARE STMT FROM :DSTRING;
```

Как и другие операторы, SELECT с фиксированным списком может содержать маркеры параметров. Однако в данном примере они не нужны.

Чтобы выполнить STMT, ваша программа должна открыть указатель, считать строки из таблицы результатов и закрыть указатель. Как это сделать, описано в следующих разделах.

### **Открываем указатель**

Оператор OPEN вычисляет оператор SELECT с именем STMT. Например:

*Без маркеров параметров:* EXEC SQL OPEN C1;

*Если в STMT есть маркеры параметров,* для задания значений для всех маркеров параметров в STMT необходимо использовать условие USING оператора OPEN. Если в STMT четыре маркера параметров, надо написать:

```
EXEC SQL OPEN C1 USING :PARM1, :PARM2, :PARM3, :PARM4;
```

### **Выбираем строки из таблицы результатов**

Ваша программа может в цикле выполнять, например, такой оператор:

```
EXEC SQL FETCH C1 INTO :NAME, :PHONE;
```

Ключевая особенность этого оператора – использование списка переменных хоста для получения значений, возвращаемых FETCH. В этом списке содержится известное число элементов (два – :NAME и :PHONE) с известными типами данных (символьные строки длиной 15 и 4, соответственно).

Этот список можно использовать в операторе FETCH только из–за того, что вы запланировали в программе использование SELECT только с фиксированным списком. Каждая из строк, на которую указывает указатель C1, должна содержать ровно два символьных значения соответствующей длины. Если эта программа должна обработать что–нибудь еще, в ней должна использоваться технология, описанная в разделе Динамический SQL для операторов SELECT с переменным списком.

### **Закрываем указатель**

Этот этап аналогичен этапу для статического SQL. Например, оператор WHENEVER NOT FOUND может указать процедуру, в которой будет такой оператор:

```
EXEC SQL CLOSE C1;
```

---

## **Динамический SQL для операторов SELECT с переменным списком**

Оператор SELECT с переменным списком возвращает строки, содержащие неизвестное число значений с неизвестными типами. При его использовании заранее точно *неизвестно*, какие виды переменных хоста надо объявить для сохранения результатов. (Существенно более простая ситуация, когда это *известно*, описана в разделе “Динамический SQL для операторов SELECT с фиксированным списком” на стр. 559.) Поскольку оператору SELECT с переменным списком требуются переменные указателей для области дескриптора SQL, его нельзя выполнять в программах на языках FORTRAN или OS/VS COBOL. Если вам требуется использование оператора SELECT с переменным списком, программа на языках FORTRAN или OS/VS COBOL может вызвать подпрограмму, написанную на языке, поддерживающем переменные указателей (например, на PL/I или на ассемблере).

### **Что должна делать ваша прикладная программа**

Для динамического выполнения оператора SELECT с переменным списком ваша программа должна выполнить следующие действия:

1. Включить SQLCA
2. Загрузить входной оператор SQL в область данных

Первые два этапа полностью совпадают с тем, что описано в разделе “Динамический SQL без операторов SELECT” на стр. 555; следующий этап – новый:

3. Подготовить и выполнить оператор. Этот этап сложнее, чем для SELECT с фиксированным списком. Подробности смотрите в разделах “Подготовка оператора SELECT с переменным списком” на стр. 563 и “Динамическое выполнение оператора SELECT с переменным списком” на стр. 573. Он включает в себя следующие стадии:
  - a. Включить SQLDA (область дескриптора SQL).
  - b. Объявить указатель и подготовить оператор переменной.
  - c. Получить информацию о типах данных каждого из столбцов таблицы результатов.
  - d. Определить основную память для строки считанных данных.
  - e. Поместить в SQLDA адреса памяти, куда надо помещать каждый из элементов полученных данных.
  - f. Открыть указатель.
  - g. Произвести выборку строки.
  - h. Наконец, закрыть указатель и освободить основную память.

Для операторов с маркерами параметров эти операции еще сложнее.

4. Обработать все возможные ошибки.

## Подготовка оператора SELECT с переменным списком

Предположим, что ваша программа динамически выполняет операторы SQL, но на этот раз без каких-либо ограничений на их форму. Программа считывает операторы с терминала, и предварительно вы о них ничего не знаете. Эти операторы могут даже не быть операторами SELECT.

Как и в случае других операторов (не SELECT), программа помещает эти операторы в символьную переменную с переменной длиной под названием DSTRING. Программа продолжает подготовку оператора из переменной, а затем присваивает этому оператору некоторое имя, например, STMT.

Здесь требуется новая операция. Программа должна выяснить, является ли оператор оператором SELECT. Если это оператор SELECT, программа должна выяснить число значений в каждой из строк и тип их данных. Такая информация берется из *области дескриптора SQL* (SQLDA).

### Область дескриптора SQL (SQLDA)

SQLDA – структура для связи с вашей программой, и память для нее обычно выделяется динамически во время выполнения.

Чтобы включить SQLDA в программу на PL/I или на C, используйте оператор:  
EXEC SQL INCLUDE SQLDA;

Для ассемблера укажите в области определения памяти CSECT:

EXEC SQL INCLUDE SQLDA

Для COBOL, за исключением OS/VS COBOL, используйте оператор:

EXEC SQL INCLUDE SQLDA END-EXEC.

SQLDA нельзя включить в программу на языке OS/VS COBOL или FORTRAN.

Полная структура SQLDA и описаний, задаваемых операторами INCLUDE, описаны в Приложении С руководства *DB2 SQL Reference*.

### Получение информации об операторе SQL

В SQLDA может содержаться переменное число вхождений SQLVAR, каждое из которых представляет собой набор из пяти полей, описывающих один столбец в таблице результатов оператора SELECT.

Число вхождений SQLVAR зависит от следующих факторов:

- Числа столбцов в таблице результатов, которую вы хотите описать.
- Хотите ли вы, чтобы PREPARE или DESCRIBE помещал в SQLDA и имена столбцов, и метки. Это задается опцией USING BOTH в операторе PREPARE или DESCRIBE.
- Есть ли в таблице результатов столбцы типа большой объект (LOB) или пользовательского типа.

В Табл. 52 на стр. 564 показано минимальное число SQLVAR для таблицы результатов из *n* столбцов.

Таблица 52. Минимальное число SQLVAR для таблицы результатов из  $n$  столбцов

Тип описания и содержания набора результатов	Без USING BOTH	C USING BOTH
Нет пользовательских типов или типов большой объект (LOB)	$n$	$2*n$
Есть пользовательские типы, но не типы большой объект (LOB)	$2*n$	$3*n$
Есть типы большой объект (LOB), но не пользовательские типы	$2*n$	$2*n$
Есть и типы большой объект (LOB), и пользовательские типы	$2*n$	$3*n$

SQLDA с  $n$  SQLVAR называется *одинарной SQLDA*, SQLDA с  $2*n$  SQLVAR называется *двойной SQLDA*, SQLDA с  $3*n$  SQLVAR называется *тройной SQLDA*.

У программы, допускающей динамическое выполнение любых видов операторов SQL, есть два варианта:

- Выделить самую большую SQLDA, которая ей может когда-либо потребоваться. Максимальное число столбцов в таблице результатов – 750, поэтому SQLDA с 750 столбцами занимает 33 016 байт для одинарной SQLDA, 66 016 байт для двойной SQLDA и 99 016 байт для тройной SQLDA. Большинство SELECT не возвращает 750 столбцов, поэтому программа обычно не использует большую часть этого пространства.
- Выделить меньшую SQLDA с меньшим числом SQLVAR. В этом случае программа может выяснить, задан ли оператор SELECT, и, если да, то сколько столбцов в его таблице результатов. Если столбцов в результате больше, чем может вместить SQLDA, DB2 не возвращает описаний. Когда это происходит, программа должна получить для второй SQLDA достаточно большую память, чтобы вместить описания столбцов, и запросить у DB2 описания еще раз. Несмотря на то, что для программы такая технология сложнее, чем предыдущая, именно она используется чаще всего.

Сколько столбцов следует разрешить? Надо выбрать число, которое будет достаточно большим для большинства операторов SELECT, но не приведет к излишней потере пространства; хороший компромисс – 40 столбцов. Покажем, что делать, если оператор возвращает больше столбцов, чем разрешено; в следующем примере используется SQLDA, которая вмещает по крайней мере 100 столбцов.

### Объявление указателя для оператора

Как и раньше, необходим указатель для динамического SELECT. Например, введите:

```
EXEC SQL
DECLARE C1 CURSOR FOR STMT;
```

## Подготовка оператора с использованием минимальной SQLDA

Предположим, что программа объявляет структуру SQLDA с именем MINSQLDA, в которой 100 вхождений SQLVAR, и SQLN имеет значение 100. Чтобы подготовить оператор из строки символов в DSTRING и ввести его описание в MINSQLDA, введите:

```
EXEC SQL PREPARE STMT FROM :DSTRING;  
EXEC SQL DESCRIBE STMT INTO :MINSQLDA;
```

Можно также использовать условие INTO в операторе PREPARE:

```
EXEC SQL  
PREPARE STMT INTO :MINSQLDA FROM :DSTRING;
```

Ни в каком из этих примеров не используйте условие USING. Пока используется только минимальная структура SQLDA. На рис. 131 показано ее содержимое.



Рисунок 131. Минимальная структура SQLDA

## SQLN определяет, что получает SQLVAR

Поле SQLN, значение которого необходимо задать перед использованием DESCRIBE (или PREPARE INTO), определяет, сколько вхождений SQLVAR включено в SQLDA. Если для DESCRIBE требуется больше, результаты DESCRIBE зависят от содержимого таблицы результатов. Пусть  $n$  – число столбцов в таблице результатов. Тогда:

- Если в таблице результатов есть по крайней мере один столбец с пользовательским типом, но нет столбцов типа большой объект (LOB), не указано USING BOTH и  $n \leq SQLN < 2^n$ , DB2 возвращает основную информацию SQLVAR в первых  $n$  SQLVAR, но не возвращает информацию о пользовательских типах. К основной информации SQLVAR относится:
  - Код типа данных
  - Атрибут длины (за исключением типа большой объект)
  - Имя или метка столбца
  - Адрес переменной хоста
  - Адрес переменной–индикатора
- Если же SQLN меньше, чем минимальное число SQLVAR, указанное в Табл. 52 на стр. 564, DB2 не возвращает в SQLVAR никакой информации.

Независимо от того, достаточно ли велика ваша SQLDA, при любом выполнении DESCRIBE DB2 возвращает следующие значения, которые можно использовать для построения SQLDA правильного размера:

- SQLD

0, если оператор SQL не является оператором SELECT. В противном случае равно числу столбцов в таблице результатов. Число необходимых для SELECT вхождений SQLVAR зависит от значения в 7-м байте SQLDAID.

- 7–й байт SQLDAID

2, если каждый из столбцов в таблице результатов требует 2 записи SQLVAR. 3, если каждый из столбцов в таблице результатов требует 3 записи SQLVAR.

### **Если оператор не является оператором SELECT**

Чтобы выяснить, является ли оператор оператором SELECT, программа может запросить поле SQLD в MINSQLDA. Если в этом поле находится 0, оператор *не является* оператором SELECT, этот оператор уже подготовлен, и программа может его выполнить. Если в этом операторе нет маркеров параметров, можно использовать:

```
EXEC SQL EXECUTE STMT;
```

(Если в операторе есть маркеры параметров, необходимо использовать область дескриптора SQL; указания смотрите в разделе “Выполнение произвольных операторов с маркерами параметров” на стр. 574.)

### **Запрос памяти для второй SQLDA, если она необходима**

Теперь можно выделить память для второй, полноразмерной SQLDA; назовем ее FULSQLDA. На рис. 132 на стр. 567 показана ее структура.

У FULSQLDA есть заголовок с фиксированной длиной 16 байтов, за которым следует раздел переменной длины, состоящий из структур с форматом SQLVAR. Если в таблице результатов есть столбцы с типом большой объект (LOB) или с пользовательским типом, раздел с переменной длиной, состоящий из структур с форматом SQLVAR2, идет за структурами с форматом SQLVAR. Все структуры SQLVAR и SQLVAR2 имеют длину 44 байта. Подробности об этих двух форматах SQLVAR смотрите в Приложении С руководства *DB2 SQL Reference*. Число необходимых элементов SQLVAR и SQLVAR2 находится в поле SQLD MINSQLDA, а общая необходимая для FULSQLDA длина (16 + SQLD \* 44) находится в поле SLDABC MINSQLDA. Выделите это количество памяти.



\* Длина символьной строки в SQLNAME.  
SQLNAME - 30-байтная область сразу после поля длины.

\* Длина символьной строки в SQLTNAME.  
SQLTNAME - 30-байтная область сразу после поля длины.

Рисунок 132. Структура SQLDA

### Описываем оператор SELECT еще раз

После выделения достаточного пространства для FULSQLDA программа должна выполнить следующие этапы:

1. Поместить общее число вхождений SQLVAR и SQLVAR2 в FULSQLDA в поле SQLN FULSQLDA. Это число появляется в поле SQLD MINSQLDA.
2. Описать этот оператор еще раз в новой SQLDA:

```
EXEC SQL DESCRIBE STMT INTO :FULSQLDA;
```

После выполнения оператора DESCRIBE в каждом из SQLVAR в полноразмерной SQLDA (в нашем примере FULSQLDA) содержится описание одного столбца таблицы результатов в пяти полях. Если во вхождении SQLVAR описан столбец с типом большой объект (LOB) или столбец пользовательского типа, в соответствующем вхождении SQLVAR2 содержится специфичная для типа большой объект (LOB) или пользовательского типа дополнительная информация. На рис. 133 показана SQLDA, описывающая два обычных столбца (не типа большой объект и не пользовательского типа). Пример описания таблицы результатов со столбцами типа большой объект (LOB) или столбцами пользовательского типа смотрите в разделе “Описание таблиц с типами большой объект (LOB) и пользовательскими типами” на стр. 571.

Заголовок SQLDA	SQLDA			8816	200	200
Элемент SQLVAR 1 (44 байта)	452	3	Не определена	0	8	WORKDEPT
Элемент SQLVAR 2 (44 байта)	453	4	Не определена	0	7	PHONENO

Рисунок 133. Содержимое FULSQLDA после выполнения DESCRIBE

## Запрос памяти под строку

Перед выборкой строк таблицы результатов программа должна:

1. Проанализировать каждое из описаний SQLVAR, чтобы определить, сколько места требуется для значения столбца.
2. Сгенерируйте адрес некоторой области памяти необходимого размера.
3. Занесите этот адрес в поле SQLDATA.

Если поле SQLTYPE указывает на то, что значение может быть пустым, программа должна также поместить в поле SQLIND адрес переменной—индикатора.

На рис. 134, рис. 135 и рис. 136 на стр. 569 показана область дескриптора SQL после выполнения определенных действий. Табл. 53 на стр. 569 описывает значения в области дескрипторов. На рис. 134, оператор DESCRIBE вставил все значения, за исключением первого вхождения числа 200. Программа вставила число 200 перед тем как выполнила DESCRIBE, чтобы сообщить, сколько разрешить вхождений SQLVAR. Если в таблице результатов SELECT больше столбцов, поля SQLVAR не описывают ничего.

Следующий набор из пяти значений, первый SQLVAR, соответствует первому столбцу таблицы результатов (столбец WORKDEPT). Элемент 1 SQLVAR содержит строки символов фиксированной длины и не допускает пустых значений (SQLTYPE=452); атрибут имеет длину 3. Информацию о значениях SQLTYPE смотрите в Приложении С руководства *DB2 SQL Reference*.

Заголовок SQLDA	→	SQLDA			8816	200	200
Элемент SQLVAR 1 (44 байта)	→	452	3	Не определена	0	8	WORKDEPT
Элемент SQLVAR 2 (44 байта)	→	453	4	Не определена	0	7	PHONENO

Рисунок 134. Область дескриптора SQL после выполнения DESCRIBE

Заголовок SQLDA	→	SQLDA			8816	200	200
Элемент SQLVAR 1 (44 байта)	→	452	3	Addr FLDA	Addr FLDAI	8	WORKDEPT
Элемент SQLVAR 2 (44 байта)	→	453	4	Addr FLDB	Addr FLDBI	7	PHONENO

Переменные-индикаторы  
(полуслово)  
FLDAI      FLDBI

FLDA      FLDB  
CHAR(3)      CHAR(4)

[ ]      [ ]      [ ]      [ ]

Рисунок 135. Область дескриптора SQL после анализа описаний и запроса памяти

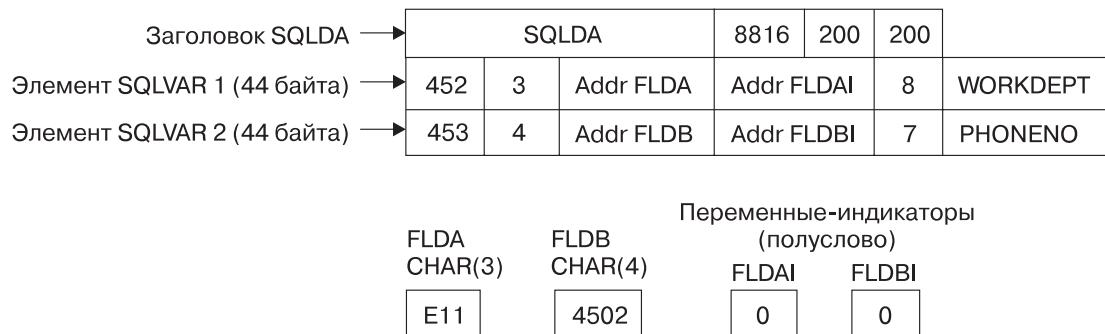


Рисунок 136. Область дескриптора SQL после выполнения *FETCH*

Таблица 53. Значения, вставленные в SQLDA

Значение	Поле	Описание
SQLDA	SQLDAID	“Метка”
8816	SQLDABC	Размер SQLDA в байтах (16 + 44 * 200)
200	SQLN	Число вхождений SQLVAR, заданное программой
200	SQLD	Число вхождений SQLVAR, реально использованных оператором DESCRIBE
452	SQLTYPE	Значение SQLTYPE в первом вхождении SQLVAR. Указывает, что в первом столбце содержатся строки символов с фиксированной длиной, и не допускается пустых значений.
3	SQLLEN	Атрибут длины столбца
Неопределенное значение или значение CCSID	SQLDATA	Байты 3 и 4 для столбца строкового типа содержат CCSID. Не определено для всех других типов столбцов.
Неопределенное значение	SQLIND	
8	SQLNAME	Число символов в имени столбца
WORKDEPT	SQLNAME+2	Имя первого столбца

### Занесение адресов памяти в SQLDA

После анализа описания каждого из столбцов программа должна заменить содержимое каждого поля SQLDATA на адрес области памяти, достаточно большой, чтобы вместить значения из этого столбца. Аналогично для каждого из столбцов, допускающих пустые значения, программа должна заменить содержимое поля SQLIND. Это содержимое должно стать адресом полуслова, которое можно использовать в качестве переменной-индикатора для этого столбца. Программа, конечно, может запросить выделить память для этой цели, но использованные области памяти не обязательно должны быть смежными.

На рис. 135 на стр. 568 показано содержимое области дескриптора перед тем, как программа получит какую-либо из строк таблицы результатов. Адреса полей и переменные указателей уже находятся в SQLVAR.

## **Изменение CCSID для прочитанных данных**

У всех строковых данных DB2, если не задано условие FOR BIT DATA, есть кодовая схема и связанный с ней CCSID. Когда вы выбираете строковые данные из таблицы, у выбранных данных обычно та же кодовая схема и CCSID, что и у таблицы, с одним исключением: если выполняется запрос к таблице DB2 for OS/390, определенной как ASCII, считанные данные кодируются в EBCDIC.

При использовании SQLDA для динамического выбора данных из таблицы можно изменить кодовую схему для считанных данных. Эту возможность можно использовать, чтобы из таблицы, определенной как ASCII, считывать данные в ASCII.

Чтобы изменить схему кодирования считанных данных, настройте SQLDA, как вы бы настроили любой другой оператор SELECT с переменным списком. Затем внесите в SQLDA дополнительные изменения:

1. Поместите символ + в шестой байт SQLDAID поля.
2. Для каждой из записей SQLVAR:
  - Задайте 8 в поле длины SQLNAME.
  - В первых двух байтах поля данных SQLNAME задайте X'0000'.
  - В третьем и четвертом байтах поля данных SQLNAME задайте в шестнадцатеричном формате CCSID для выходных результатов. Можно указать любой CCSID, для которого есть строка в каталоге SYSSTRINGS с соответствующим значением для OUTCCSID.

Если вы модифицируете CCSID для вывода содержимого таблицы ASCII в ASCII в системе DB2 for OS/390 и раньше вы выполнили оператор DESCRIBE для оператора SELECT, который используется для вывода таблицы ASCII, в полях SQLDATA в SQLDA для DESCRIBE содержится CCSID ASCII для этой таблицы. Чтобы задать порцию данных поля SQLNAME для SELECT, перенесите содержимое каждого из полей SQLDATA в SQLDA из DESCRIBE в каждое из полей SQLNAME в SQLDA для SELECT. Если вы используете одну и ту же SQLDA и для DESCRIBE, и для SELECT, перед тем, как модифицировать поле SQLDATA для SELECT, не забудьте перенести содержимое поля SQLDATA в SQLNAME.

Например, предположим, что таблица, содержащая WORKDEPT и PHONENO, определена с CCSID ASCII. Чтобы получить данные для столбцов WORKDEPT и PHONENO в ASCII CCSID 437 (X'01B5'), измените SQLDA, как показано на рис. 137 на стр. 571.



Рисунок 137. Область дескриптора SQL для получения данных в ASCII CCSID 437

### Использование меток столбцов

По умолчанию DESCRIBE описывает каждый из столбцов в поле SQLNAME его именем. Вместо этого можно заставить этот оператор использовать метки столбцов, написав:

```
EXEC SQL
DESCRIBE STMT INTO :FULSQLDA USING LABELS;
```

В этом случае SQLNAME для столбца без метки не содержит ничего. Если вы предпочитаете использовать метки, когда они существуют, и имена столбцов, когда меток нет, введите USING ANY. (У некоторых столбцов, например, у производных от функций или выражений нет ни имен, ни меток; для таких столбцов SQLNAME не содержит ничего. Однако если столбец является результатом UNION, SQLNAME содержит имена столбцов первого операнда UNION.)

Можно также написать USING BOTH для получения и имени, и метки при их одновременном существовании. Однако для этого необходим второй набор SQLVAR в FULSQLDA. В первом наборе будут описания всех столбцов с использованием имен; во втором наборе — описания с использованием меток. Это означает, что необходимо выделить более длинную SQLDA для второго оператора DESCRIBE — (16 + SQLD \* 88 байт) вместо (16 + SQLD \* 44). Необходимо также удвоить число столбцов (SLQD \* 2) в поле SQLN второй SQLDA. В противном случае, если не хватит места, DESCRIBE не введет описаний ни для одного столбца.

### Описание таблиц с типами большой объект (LOB) и пользовательскими типами

В общем случае этапы, которые вы выполняете при подготовке SQLDA для выбора строк из таблицы со столбцами типа большой объект (LOB) или с пользовательским типом, похожи на этапы, выполняемые в том случае, когда в таблице нет столбцов с такими типами. Единственное отличие состоит в том, что для столбцов с типом большой объект (LOB) или пользовательским типом необходимо анализировать дополнительные поля в SQLDA.

Для примера предположим, что вы хотите выполнить оператор SELECT:

```
SELECT USER, A_DOC FROM DOCUMENTS;
```

где USER не может содержать пустых значений и является идентификатором пользовательского типа, определяемым так:

```
CREATE DISTINCT TYPE SCHEMA1.ID AS CHAR(20);
```

а A\_DOC может содержать пустые значения и имеет тип символьный большой объект (CLOB(1M)).

Таблица результатов для этого оператора содержит два столбца, но в SQLDA требуется четыре SQLVAR, поскольку таблица результатов содержит тип большой объект (LOB) и пользовательский тип. Предположим, что вы подготовили и описали этот оператор в FULSQLDA, достаточно большой для размещения четырех SQLVAR. FULSQLDA выглядит, как на рис. 138.

Заголовок SQLDA	SQLDA 2			192	4	4
Элемент SQLVAR 1 (44 байта)	452	20	Не определена	0	4	USER
Элемент SQLVAR 2 (44 байта)	409	0	Не определена	0	5	A_DOC
Элемент SQLVAR2 1 (44 байта)					7	SCH1.ID
Элемент SQLVAR2 2 (44 байта)	1048576				11	SYSIBM.CLOB

Рисунок 138. Область дескриптора SQL после описания типа символьный большой объект (CLOB) и пользовательского типа

Следующие этапы такие же, как и для таблиц результатов без типов большой объект (LOB) и пользовательских типов:

1. Проанализируйте каждое из описаний SQLVAR для определения максимального пространства, необходимого для значения столбца.

Для типа большой объект (LOB) получите длину из поля SQLLONGL вместо поля SQLLEN.

2. Сгенерируйте адрес некоторой области памяти необходимого размера.

Для типа данных большой объект (LOB), кроме того, нужна 4–байтная область памяти для длины данных типа большой объект (LOB). Можно разместить эту 4–байтную область в начале данных типа большой объект (LOB) или в другом положении.

3. Занесите этот адрес в поле SQLDATA.

Для данных типа большой объект (LOB), если вы выделили отдельную область для хранения длины данных типа большой объект (LOB), поместите этот адрес поля длины в SQLDATA. Если поле длины расположено в начале области данных типа большой объект (LOB), поместите 0 в SQLDATA.

4. Если поле SQLTYPE указывает на то, что значение может быть пустым, программа должна также поместить в поле SQLIND адрес переменной–индикатора.

На рис. 139 на стр. 573 и рис. 140 на стр. 573 показано содержимое FULSQLDA после заполнения адресов памяти и выполнения FETCH.

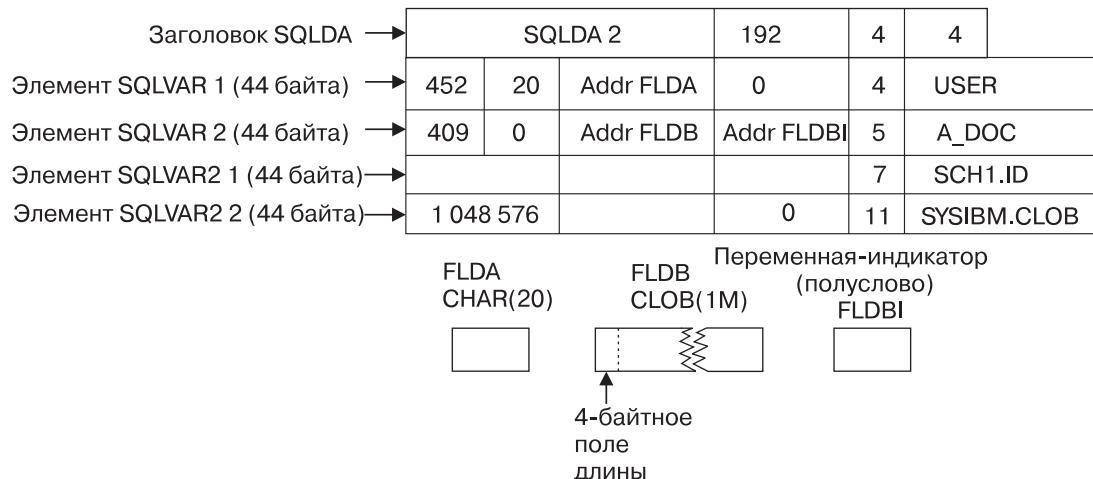


Рисунок 139. Область дескриптора SQL после анализа описаний CLOB и пользовательского типа, а также запроса памяти

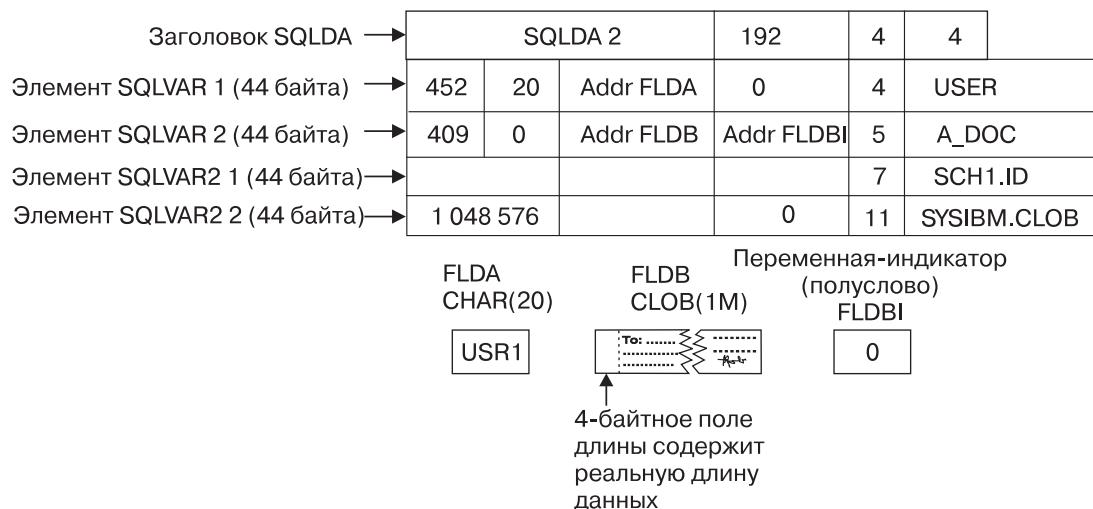


Рисунок 140. Область дескриптора SQL после выполнения FETCH для таблицы со столбцами CLOB и пользовательского типа

## Динамическое выполнение оператора SELECT с переменным списком

Получить строки таблицы результатов при использовании оператора SELECT с переменным списком достаточно легко. Эти операторы только немного отличаются от операторов в примере с фиксированным списком.

### Открываем указатель

Если в операторе SELECT нет маркеров параметров, это делается достаточно просто. Например:

```
EXEC SQL OPEN C1;
```

Если маркеры параметров есть, смотрите далее раздел “Выполнение произвольных операторов с маркерами параметров” на стр. 574.

## **Выбираем строки из таблицы результатов**

Этот оператор отличается от соответствующего для SELECT с фиксированным списком. введите:

```
EXEC SQL  
FETCH C1 USING DESCRIPTOR :FULSQLDA;
```

Ключевая особенность этого оператора – условие USING DESCRIPTOR :FULSQLDA. Это условие присваивает имя области дескрипторов SQL, в которой SQLVAR указывают на другие области. Эти другие области получают значения, которые возвращает FETCH. Такое условие можно использовать только потому, что ранее вы настроили FULSQLDA, и она выглядит, как на рис. 134 на стр. 568.

На рис. 136 на стр. 569 показан результат FETCH. Области данных, указанные в полях SQLVAR, получают значения из единственной строки таблицы результатов.

Последующие выполнения того же оператора FETCH помещают в эти области значения из последующих строк таблицы результатов.

## **Закрываем указатель**

Этот этап такой же, как и для фиксированного списка. Когда все строки будут обработаны, выполните следующий оператор:

```
EXEC SQL CLOSE C1;
```

Когда COMMIT заканчивает единицу работы, содержащую OPEN, оператор в STMT возвращается в неподготовленное состояние. Если вы не определили указатель с условием WITH HOLD, перед тем как повторно открывать указатель, необходимо будет подготовить оператор еще раз.

## **Выполнение произвольных операторов с маркерами параметров**

Рассмотрим в качестве примера программу, которая выполняет операторы динамического SQL нескольких видов, включая операторы SELECT с переменным списком и переменным количеством маркеров параметров. Эта программа может давать пользователям меню для выбора операции (обновить, выбрать, удалить); поля ввода имен таблиц и столбцов для выбора или изменения. Программа также позволяет пользователям вводить списки номеров сотрудников для выбранной операции. По введенной информации программа конструирует несколько форм операторов SQL, например, такую:

```
SELECT .... FROM DSN8610.EMP  
WHERE EMPNO IN (?, ?, ?, ..., ?);
```

Затем программа выполняет эти операторы динамически.

## **Когда известны число и типы параметров**

В приведенном выше примере вам заранее неизвестно число маркеров параметров и, возможно, представляемые ими виды параметров. Если число и типы параметров вам известны, можно использовать прежнюю технологию, как в следующих примерах:

- Если оператор SQL не является оператором SELECT, укажите имя списка переменных хоста в операторе EXECUTE:

**Неправильно:** EXEC SQL EXECUTE STMT;

**Правильно:** EXEC SQL EXECUTE STMT USING :VAR1, :VAR2, :VAR3;

- Если оператор SQL является оператором SELECT, укажите имя списка переменных хоста в операторе OPEN:

**Неправильно:** EXEC SQL OPEN C1;

**Правильно:** EXEC SQL OPEN C1 USING :VAR1, :VAR2, :VAR3;

В обоих случаях число и типы названных переменных хоста должны совпадать с числом маркеров параметров в STMT и типами соответствующих параметров. У первой переменной (VAR1 в примерах) должен быть тип, ожидаемый для первого маркера параметра в операторе, у второй переменной должен быть тип, ожидаемый для второго маркера и так далее. Число переменных должно быть не меньше числа маркеров параметров.

### **Когда число и типы параметров неизвестны**

Когда неизвестны число и типы параметров, можно настроить область дескрипторов SQL. Число SQLDA, включенных в вашу программу, не ограничено, и их можно использовать для различных целей. Предположим, что набор параметров описывает SQLDA под названием DPARM.

Структура DPARM такая же, как и у любой другой SQLDA. Число SQLVAR может быть разным, как в предыдущих примерах. В данном случае для каждого из маркеров параметров должно быть одно вхождение SQLVAR. Каждое вхождение SQLVAR описывает одну переменную хоста, которая во время выполнения заменяет один маркер параметра. Это происходит либо при открытии указателя для оператора SELECT, либо при выполнении для другого оператора (не SELECT).

Перед использованием EXECUTE или OPEN необходимо заполнить определенные поля в DPARM; остальные поля можно проигнорировать.

Поле	Использование при описании переменных хоста для маркеров параметров
SQLDAID	Седьмой байт указывает, одна или несколько записей SQLVAR используется для каждого из маркеров параметров. Если этот байт непустой, по крайней мере один из маркеров параметров представляет значение пользовательского типа или типа большой объект (LOB), поэтому в SQLDA есть несколько наборов записей SQLVAR.
SQLDABC	Длина SQLDA, равна SQLN * 44 + 16
SQLN	Число вхождений SQLVAR, выделенное для DPARM
SQLD	Реально использованное число вхождений SQLVAR. Оно должно быть меньше, чем число маркеров параметров. В каждый SQLVAR поместите следующую информацию, так же, как для оператора DESCRIBE:
SQLTYPE	Код типа переменной и допустимость пустых значений
SQLLEN	Длина переменной хоста
SQLDATA	Адрес переменной хоста
SQLIND	Адрес переменной-индикатора, если он необходим
SQLNAME	Игнорируется

## **Использование SQLDA с EXECUTE или OPEN**

Чтобы указать, что SQLDA с именем DPARM описывает переменные хоста, которые заменяют во время выполнения маркеры параметров, используйте с EXECUTE или OPEN условие USING DESCRIPTOR.

- Для оператора, отличного от SELECT, введите:

```
EXEC SQL EXECUTE STMT USING DESCRIPTOR :DPARM;
```

- Для оператора SELECT введите:

```
EXEC SQL OPEN C1 USING DESCRIPTOR :DPARM;
```

## **Как опция связывания REOPT(VARS) влияет на динамический SQL**

Когда вы указываете опцию связывания REOPT(VARS), DB2 переоптимизирует путь доступа во время выполнения для операторов SQL, содержащих переменные хоста, маркеры параметров и специальные регистры. Опция REOPT(VARS) оказывает на операторы динамического SQL следующее влияние:

- Когда вы указываете опцию REOPT(VARS), DB2 автоматически использует DEFER(PREPARE), что означает, что DB2 ожидает подготовки оператора до тех пор, пока не встретит оператор EXECUTE или OPEN.
- Если выполнить оператор DESCRIBE, а затем оператор EXECUTE не для оператора SELECT, DB2 подготавливает этот оператор дважды: первый раз для оператора DESCRIBE и еще раз для оператора EXECUTE. DB2 использует значения во входных переменных только во время второго PREPARE. Если в вашей программе много динамических операторов не—SELECT, такие многочисленные PREPARE могут снизить производительность. Чтобы повысить производительность, попробуйте выделить код с этими операторами в отдельный пакет и связывать этот пакет с опцией NOREOPT(VARS).
- Если выполнить оператор DESCRIBE до того, как открыть для него указатель, DB2 подготавливает этот оператор дважды. Однако если выполнить оператор DESCRIBE после открытия указателя, DB2 подготавливает его только один раз. Чтобы повысить производительность программы, связанной с опцией REOPT(VARS), выполняйте оператор DESCRIBE *после* того, как открываете указатель. Чтобы не выполнять DESCRIBE автоматически перед открытием указателя, не используйте оператор PREPARE с условием INTO.
- Если вы используете для прикладных программ, связанных с REOPT(VARS), прогностическое ограничение, DB2 не возвращает код предупреждения SQL, если операторы динамического SQL превышают порог предупреждения этого прогностического ограничения. DB2 не возвращает SQLCODE ошибки, когда операторы динамического SQL превышают порог ошибки прогностического ограничения. DB2 возвращает код ошибки SQL для оператора EXECUTE или OPEN.

---

## **Использование динамического SQL в языке COBOL**

Все формы динамического SQL можно использовать во всех версиях языка COBOL, за исключением OS/VS COBOL. Программы OS/VS COBOL, использующие SQLDA, для управления переменными адресов и для выделения памяти должны использовать подпрограмму на ассемблере. Подробное описание и работающий пример такого способасмотрите в разделе “Пример программы на языке COBOL с динамическим SQL” на стр. 899.



---

## Глава 7–2. Использование хранимых процедур в системах клиент–сервер

В этой главе рассматриваются следующие темы:

- “Введение в хранимые процедуры”
- “Пример простой хранимой процедуры” на стр. 581
- “Настройка среды хранимых процедур” на стр. 584
- “Написание и подготовка хранимой процедуры” на стр. 593
- “Написание и подготовка прикладной программы, использующей хранимые процедуры” на стр. 607
- “Выполнение хранимой процедуры” на стр. 643
- “Тестирование хранимой процедуры” на стр. 649

---

### Введение в хранимые процедуры

*Хранимая процедура* – это скомпилированная программа, которая хранится на локальном или удаленном сервере DB2 и может выполнять операторы SQL. Типичная хранимая процедура содержит несколько операторов SQL и какие-либо блоки обработки данных на языке хоста. В прикладной программе клиента для вызова хранимой процедуры используется оператор CALL языка SQL.

Хранимые процедуры имеет смысл использовать в прикладных программах для среды клиент–сервер, которые обладают по крайней мере одной из следующих особенностей:

- Выполняет много удаленных операторов SQL.

Удаленные операторы SQL могут вызывать много сетевых операций передачи и приема, что увеличивает загрузку процессора.

При использовании хранимых процедур для выполнения множества операторов SQL прикладной программы на сервер DB2 передается только одно сообщение, что позволяет уменьшить сетевой трафик.

- Обращается к переменным хоста, для которых нужно гарантировать безопасность и целостность.

Хранимые процедуры позволяют убрать программы SQL с рабочей станции и тем самым запретить пользователям рабочей станции изменять содержимое важнейших операторов SQL и переменных хоста.

На рис. 141 на стр. 580 и рис. 142 на стр. 580 показана разница между вариантом с использованием хранимых процедур и вариантом без их использования.

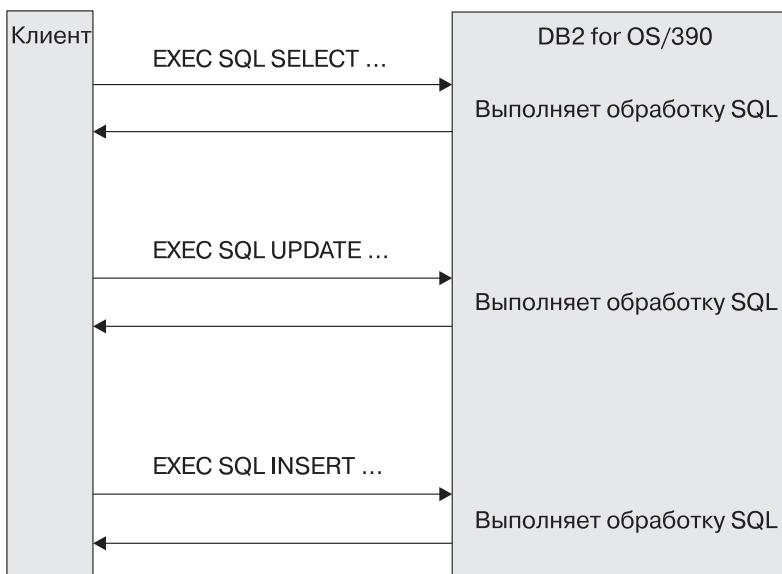


Рисунок 141. Работа без использования хранимых процедур. Операторы SQL содержатся в самой прикладной программе и при выполнении каждого оператора происходит отдельное обращение к серверу.

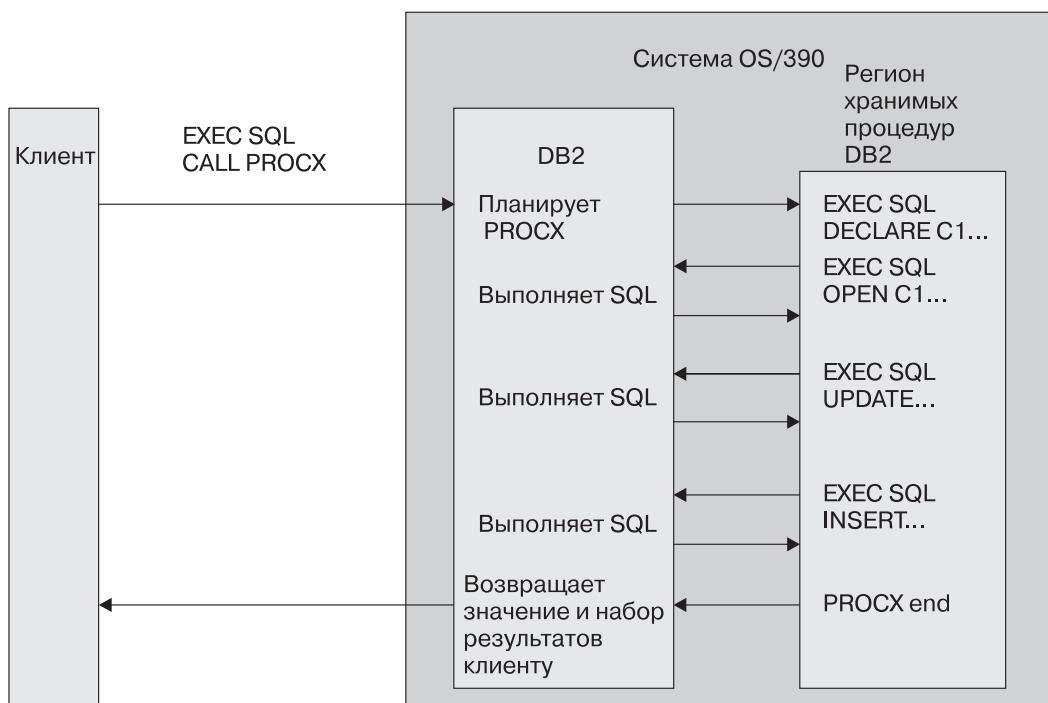


Рисунок 142. Работа с использованием хранимых процедур. Для выполнения той же последовательности операторов SQL достаточно одной операции передачи или приема.

---

## Пример простой хранимой процедуры

Предположим, что прикладная программа выполняется на клиентской рабочей станции и вызывает хранимую процедуру *A* на сервере DB2 системы LOCA. Хранимая процедура *A* выполняет следующие действия:

1. Получает набор параметров, содержащий данные для одной строки таблицы DSN8610.EMPPROJACT, в которой хранится информация о сотрудниках для конкретных проектов. Эти параметры – входные параметры оператора SQL CALL:
  - EMP: номер сотрудника
  - PRJ: номер проекта
  - ACT: ID работы
  - EMT: требуемый процент рабочего времени для сотрудника
  - EMS: дата начала работы
  - EME: предполагаемая дата завершения работы
2. Объявляет указатель C1 с опцией WITH RETURN, который используется для возвращения вызывающей программе набора результатов, содержащего все строки таблицы EMPPROJACT.
3. Обращается к таблице EMPPROJACT, чтобы определить, существует ли в ней строка, значения столбцов PROJNO, ACTNO, EMSTDATE и EMPNO которой совпадают со значениями параметров PRJ, ACT, EMS и EMP. (Наборы значений в этих столбцах уникальны. Существует не более одной строки таблицы с этими значениями.)
4. Если такая строка существует, выполняет оператор SQL UPDATE, записывающий в столбцы EMPTIME и EMENDATE значения параметров EMT и EME.
5. Если такая строка не найдена, выполняет оператор SQL INSERT, вставляющий в таблицу новую строку, значения столбцов которой задаются параметрами процедуры.
6. Открывает указатель C1. Теперь по окончании этой хранимой процедуры вызывающей программе будет возвращен набор результатов.
7. Возвращает два параметра, содержащих следующие значения:
  - Код, указывающий тип последнего выполненного оператора SQL: UPDATE или INSERT.
  - SQLCODE этого оператора.

На рис. 143 на стр. 582 показаны шаги выполнения этой хранимой процедуры.

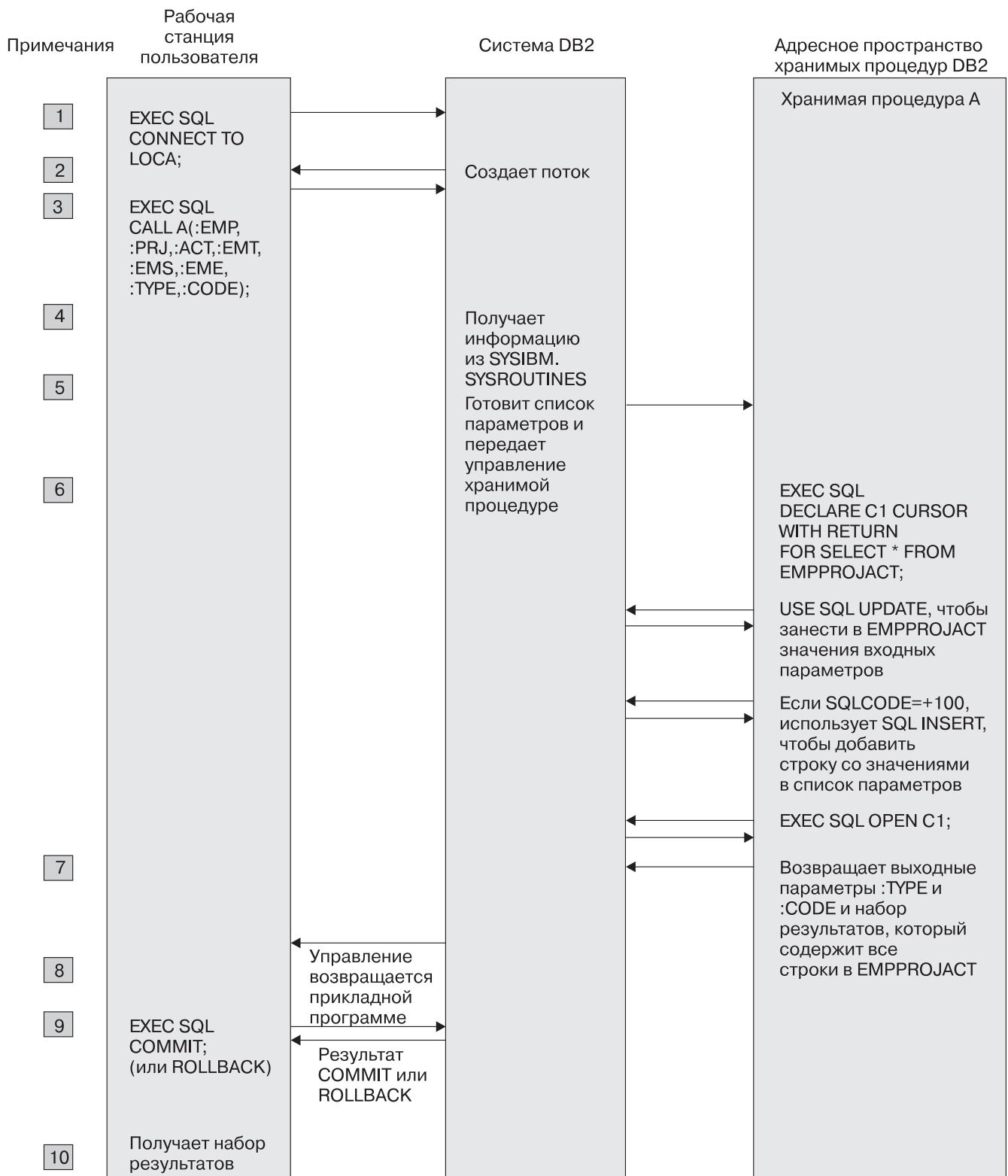


Рисунок 143. Выполнение хранимой процедуры

#### Примечания к рисунку рис. 143:

1. Прикладная программа рабочей станции использует оператор SQL CONNECT для установления диалога с DB2.

2. DB2 создает поток DB2 для обработки требований SQL.
  3. Оператор SQL CALL сообщает серверу DB2, что прикладная программа хочет выполнить хранимую процедуру. Вызывающая программа задает необходимые параметры.
  4. План для прикладной программы клиента содержит информацию из таблицы каталога SYSROUTINES о хранимой процедуре A. DB2 сохраняет в кэше все строки таблицы, связанной с A, чтобы при последующих вызовах A для этой таблицы не требовалась операции ввода–вывода.
  5. DB2 записывает в адресное пространство хранимых процедур информацию о требовании, и начинается выполнение хранимой процедуры.
  6. Хранимая процедура выполняет операторы SQL.

DB2 проверяет, что владелец пакета или плана, содержащего данный оператор SQL CALL, обладает полномочиями EXECUTE для пакета, связанного с хранимой процедурой DB2.

Один из операторов SQL открывает указатель, объявленный с опцией WITH RETURN. В результате прикладной программе рабочей станции будет возвращен набор результатов.
  7. Хранимая процедура присваивает значения выходным параметрам и завершает работу. Управление возвращается в адресное пространство хранимых процедур DB2 и затем системе DB2. Если в определении хранимой процедуры задано COMMIT ON RETURN NO, DB2 не выполняет принятие или откат для любых изменений, вызванных операторами SQL в хранимой процедуре, пока вызывающая программа не выполнит явный оператор COMMIT или ROLLBACK. Если для COMMIT\_ON\_RETURN задано значение YES и хранимая процедура выполняется успешно, DB2 производит принятие всех изменений.
  8. Управление возвращается вызывающей прикладной программе, которая получает выходные параметры и набор результатов. Затем DB2:
    - Закрывает все указатели, открытые хранимой процедурой, за исключением указателя, открытого хранимой процедурой для возврата набора результатов.
    - Отбрасывает все операторы SQL, подготовленные хранимой процедурой.
    - Возвращает в систему рабочую память, использовавшуюся хранимой процедурой.
- Прикладная программа может продолжать вызывать хранимые процедуры или выполнять операторы SQL. DB2 получает и обрабатывает требование COMMIT или ROLLBACK. Эта операция COMMIT или ROLLBACK относится ко всем операторам SQL в данной рабочей единице независимо от того, выполнены они прикладной программой или хранимой процедурой.
- Если прикладная программа работает в среде IMS или CICS, будут использоваться точки синхронизации IMS или CICS, а не операторы SQL COMMIT или ROLLBACK.
9. DB2 возвращает прикладной программе ответное сообщение, описывающее результат операции COMMIT или ROLLBACK.

10. Чтобы получить содержимое таблицы EMPPROJACT, возвращенное хранимой процедурой в наборе результатов, прикладная программа рабочей станции выполняет следующие действия:
- Объявляет локатор набора результатов для возвращенного набора результатов.
  - Выполняет оператор ASSOCIATE LOCATORS, чтобы связать этот локатор набора результатов с данным набором результатов.
  - Выполняет оператор ALLOCATE CURSOR, чтобы связать указатель с этим набором результатов.
  - Несколько раз выполняет оператор FETCH с этим указателем, чтобы получить строки из набора результатов.

---

## Настройка среды хранимых процедур

В этом разделе рассматриваются необходимые предварительные действия для выполнения хранимых процедур. Большая часть этой информации предназначена для системных администраторов, но разработчики прикладных программ должны прочитать раздел “Определение хранимой процедуры для DB2” на стр. 585. В этом разделе объясняется, как при помощи оператора CREATE PROCEDURE определить хранимую процедуру для DB2.

Чтобы подготовить подсистему DB2 для выполнения хранимых процедур, выполните следующие действия:

- Решите, какие адресные пространства должны использоваться для хранимых процедур: созданные WLM или созданные DB2.

Сравнение этих двух сред смотрите в разделе 5 (том 2) руководства *DB2 Administration Guide*.

Если вы использовали адресные пространства, созданные DB2, и хотите перейти к адресным пространствам, созданным WLM, смотрите информацию о необходимых для этого действиях в разделе “Перенос хранимых процедур в среду, созданную WLM (для системных администраторов)” на стр. 591.

- Определите процедуры JCL для адресных пространств хранимых процедур.

В компоненте DSNTIJMV набора данных DSN610.SDSNSAMP содержится пример процедур JCL для запуска адресных пространств, созданных WLM и DB2. Если ввести имя процедуры WLM или имя процедуры DB2 в панели установки DSNTIPX, DB2 настроит для вас процедуру JCL. Подробную информацию смотрите в разделе 2 руководства *DB2 Installation Guide*.

- Для адресных пространств, созданных WLM, определите среды прикладных программ WLM для групп хранимых процедур и свяжите с каждой средой прикладных программ процедуру запуска JCL.

Информацию о том, как это сделать, смотрите в разделе 5 (том 2) руководства *DB2 Administration Guide*.

- Если вы собираетесь использовать хранимые процедуры с интерфейсом ODBA для обращения к базам данных IMS, измените процедуры запуска для адресных пространств, где будут выполняться эти хранимые процедуры, следующим образом:

- В конец STEPLIB добавьте имя набора данных для набора данных IMS, содержащего вызываемый код интерфейса ODBA (обычно это IMS.RESLIB).
- После оператора STEPLIB DD добавьте оператор DFSRESLB DD с именем набора данных IMS, содержащего вызываемый код интерфейса ODBA.
- Установите языковую среду и соответствующие компиляторы.

Сведения об установке языковая среда смотрите в руководстве *OS/390 Language Environment for OS/390 & VM Customization*.

Минимальные требования к компилятору и языковой среде смотрите в разделе “Требования к языкам хранимой процедуры и вызывающей программы” на стр. 593.

Для каждой хранимой процедуры выполните следующие действия:

- Убедитесь, что библиотека, содержащая эту хранимую процедуру, задана в STEPLIB для процедуры запуска адресного пространства хранимых процедур.
- Используйте оператор CREATE PROCEDURE, чтобы определить эту хранимую процедуру для DB2, и оператор ALTER PROCEDURE, чтобы изменить это определение.

Подробную информацию смотрите в разделе “Определение хранимой процедуры для DB2.”

- Выполните операции по защите для этой хранимой процедуры.

Дополнительную информацию смотрите в Разделе 3 руководства *DB2 Administration Guide*.

## Определение хранимой процедуры для DB2

Чтобы хранимая процедура могла выполняться, ее необходимо определить для DB2. Для этого используется оператор SQL CREATE PROCEDURE. Чтобы изменить определение, используйте оператор ALTER PROCEDURE.

В Табл. 54 перечислены характеристики хранимой процедуры и соответствующие им параметры операторов CREATE PROCEDURE и ALTER PROCEDURE.

Таблица 54 (Стр. 1 из 2). Характеристики хранимой процедуры

Характеристика	Параметр оператора CREATE/ALTER PROCEDURE
Имя хранимой процедуры	PROCEDURE
Объявления параметров	
Внешнее имя	EXTERNAL NAME
Язык	LANGUAGE ASSEMBLE LANGUAGE C LANGUAGE COBOL LANGUAGE PLI
Детерминированная или недетерминированная	NOT DETERMINISTIC DETERMINISTIC

Таблица 54 (Стр. 2 из 2). Характеристики хранимой процедуры

Характеристика	Параметр оператора CREATE/ALTER PROCEDURE
Типы операторов SQL в этой хранимой процедуре	NO SQL CONTAINS SQL READS SQL DATA MODIFIES SQL DATA
Форма задания параметров	PARAMETER STYLE DB2SQL PARAMETER STYLE GENERAL PARAMETER STYLE GENERAL WITH NULLS
Адресное пространство для хранимых процедур	FENCED
Собрание пакетов	NO COLLID COLLID <i>ID</i> <i>собрания</i>
Среда WLM	WLM ENVIRONMENT <i>имя</i> WLM ENVIRONMENT <i>имя</i> ,*
Разрешенная длительность выполнения хранимой процедуры	ASUTIME NO LIMIT ASUTIME LIMIT <i>целое число</i>
Сохранение загружаемого модуля в памяти	STAY RESIDENT NO STAY RESIDENT YES
Тип программы	PROGRAM TYPE MAIN PROGRAM TYPE SUB
Защита	SECURITY DB2 SECURITY USER SECURITY DEFINER
Опции времени выполнения	RUN OPTIONS <i>опции</i>
Максимальное число возвращаемых наборов результатов	RESULT SETS <i>целое число</i>
Принятие единицы работы при возвращении из хранимой процедуры	COMMIT ON RETURN YES COMMIT ON RETURN NO
Вызов с пустыми аргументами	CALLED ON NULL INPUT
Передача информации о среде DB2	NO DBINFO DBINFO <sup>1</sup>

Примечания к Табл. 54 на стр. 585:

1. DBINFO допустима только с PARAMETER STYLE DB2SQL.

Полное описание параметров операторов CREATE PROCEDURE или ALTER PROCEDURE смотрите в разделе Глава 6 *DB2 SQL Reference*.

Информацию о параметрах для оператора CREATE PROCEDURE и ALTER PROCEDURE смотрите в Главе 6 руководства *DB2 SQL Reference*.

### Передача хранимой процедуре информации о среде

Если вы задаете параметр DBINFO при определении хранимой процедуры с PARAMETER STYLE DB2SQL, DB2 передает хранимой процедуре структуру, которая содержит информацию о среде. Поскольку та же самая структура используется и для пользовательских функций, некоторые поля в этой структуре для хранимых процедур не используются. Структура DBINFO содержит следующую информацию:

#### **Длина имени положения**

Поле типа двухбайтное целое без знака. Содержит длину имени положения в следующем поле.

#### **Имя положения**

Поле типа символ, 128 байтов. Содержит имя положения, с которым в настоящее время соединена вызывающая программа.

#### **Длина ID авторизации**

Поле типа двухбайтное целое без знака. Содержит длину ID авторизации в следующем поле.

#### **ID авторизации**

Поле типа символ, 128 байтов. Содержит дополненный пробелами справа ID авторизации программы, вызвавшей хранимую процедуру. Если хранимая процедура функция вложена в другие подпрограммы (пользовательские функции или хранимые процедуры), в этом поле содержится ID программы, вызвавшей подпрограмму самого высокого уровня.

#### **Кодовая страница подсистемы**

Структура из 48 байтов, состоящая из семи полей типа целое и 20 зарезервированных байтов. Эти поля содержат информацию о CCSID и схеме кодировки подсистемы, из которой вызвана хранимая процедура. Семь полей типа целое содержат:

- EBCDIC SBCS CCSID
- EBCDIC MIXED CCSID
- EBCDIC DBCS CCSID
- ASCII SBCS CCSID
- ASCII MIXED CCSID
- ASCII DBCS CCSID
- Схему кодирования

#### **Длина спецификатора таблицы**

Двухбайтное целое без знака. Это поле содержит 0.

#### **Спецификатор таблицы**

Символьное поле длиной 128 байт. Для хранимых процедур не используется.

#### **Длина имени таблицы**

Поле типа двухбайтное целое без знака. Это поле содержит 0.

#### **Имя таблицы**

Символьное поле длиной 128 байт. Для хранимых процедур не используется.

#### **Длина имени столбца**

Поле типа двухбайтное целое без знака. Это поле содержит 0.

#### **Имя столбца**

Символьное поле длиной 128 байт. Для хранимых процедур не используется.

#### **Информация о продукте**

8–байтное символьное поле, характеризует продукт, где выполняется хранимая процедура. Данное поле имеет вид *pppvvvrrmt*, где:

- *ppp* – трехбайтный код продукта:
  - DSN** DB2 for OS/390
  - ARI** DB2 Server for VSE & VM
  - QSQ** DB2 for AS/400
  - SQL** DB2 UDB
- *vv* идентификатор версии, 2 цифры.
- *rr* идентификатор выпуска, 2 цифры.
- *m* идентификатор уровня модификации, 1 цифра.

#### **Операционная система**

Поле типа целое, 4 байта. Характеризует операционную систему, в которой выполняется программа, вызывавшая пользовательскую функцию. Может иметь следующие значения:

- 0** Неизвестно
- 1** OS/2
- 3** Windows
- 4** AIX
- 5** Windows NT
- 6** HP-UX
- 7** Solaris
- 8** OS/390
- 13** Siemens Nixdorf
- 15** Windows 95
- 16** SCO Unix

#### **Количество записей в списке столбцов табличной функции**

Поле типа двухбайтное целое без знака. Это поле содержит 0.

#### **Зарезервированная область**

24 байта.

#### **Указатель на список столбцов табличной функции**

Для хранимых процедур не используется.

#### **Уникальный идентификатор программы**

Это поле – указатель на строку, служащую уникальным идентификатором соединения программы с DB2. Для каждого соединения с DB2 эта строка порождается заново.

Строка – LUWID, состоящий из полного сетевого имени LU, за которым следует точка и номер экземпляра логической единицы работы. Имя сети LU состоит из ID сети от 1 до 8 символов, точки и сетевого имени LU от 1 до 8 символов. Номер экземпляра LUW состоит из 12 шестнадцатеричных символов, которые уникально идентифицируют единицу работы.

### **Зарезервированная область**

20 байт.

Смотрите в разделе “Соглашения по компоновке” на стр. 609 пример кодирования списка параметров DBINFO в хранимой процедуре.

### **Пример определения хранимой процедуры**

Предположим, у вас есть написанная и подготовленная хранимая процедура со следующими характеристиками:

- Имя процедуры – В.
- У процедуры есть два параметра:
  - Входной параметр V1, тип – целое число
  - Выходной параметр V2, тип – символьная строка длиной 9
- Процедура написана на языке С.
- Она не содержит операторов SQL.
- Вызовы с одинаковыми входными параметрами всегда возвращают одинаковые выходные значения.
- Имя загружаемого модуля – SUMMOD.
- Имя собрания пакетов – SUMCOLL.
- Процедура не должна выполняться более 900 служебных единиц процессорного времени.
- Параметры процедуры могут иметь пустые значения.
- После выполнения процедура должна удаляться из памяти.
- Требуемые для этой процедуры опции времени выполнения для языковой среды:  
`MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)`
- Процедура является частью среды прикладных программ WLM с именем PAYROLL.
- Она выполняется в виде основной программы.
- Она не обращается к ресурсам вне DB2, тем самым ей не требуется особая среда RACF.
- Она может возвращать до 10 наборов результатов.
- При возвращении управления к программе клиента система DB2 не должна автоматически выполнять принятие изменений.

Следующий оператор CREATE PROCEDURE определяет эту процедуру для DB2:

```
CREATE PROCEDURE B(V1 INTEGER IN, CHAR(9) OUT)
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME SUMMOD
COLLID SUMCOLL
ASUTIME LIMIT 900
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT PAYROLL
PROGRAM TYPE MAIN
SECURITY DB2
RESULT SETS 10
COMMIT ON RETURN NO;
```

Предположим, позже потребовалось изменить определение этой хранимой процедуры:

- Она берет данные из таблиц DB2, но не изменяет данных DB2.
- Параметры могут иметь пустые значения и хранимая процедура может возвращать диагностическую строку.
- Продолжительность выполнения хранимой процедуры не ограничена.
- Если эта хранимая процедура вызывается другой хранимой процедурой или пользовательской функцией, она использует среду WLM вызывающей процедуры или функции.

Чтобы задать эти изменения, используйте следующий оператор ALTER PROCEDURE:

```
ALTER PROCEDURE B
READS SQL
ASUTIME NO LIMIT
PARAMETER STYLE DB2SQL
WLM ENVIRONMENT (PAYROLL,*);
```

## Обновление среды хранимых процедур (для системных администраторов)

В зависимости от произведенных в среде хранимых процедур изменений может потребоваться выполнение одного или нескольких из следующих действий:

- Обновление языковой среды.

Выполните это обновление после изменения загрузочного модуля для хранимой процедуры, загруженной к кэш адресного пространства хранимых процедур. При обновлении языковой среды этот загрузочный модуль удаляется из кэша. При следующем вызове этой хранимой процедуры загружается новый модуль.

- Перезапуск адресного пространства хранимых процедур.

Если нужно внести изменения в запускающую процедуру JCL, следует остановить и затем запустить адресное пространство хранимых процедур.

Как это сделать, зависит от того, какие используются адресные пространства – созданные WLM или созданные DB2.

**Для адресных пространств, созданных DB2:** Для выполнения этих операций используйте команды DB2 START PROCEDURE и STOP PROCEDURE.

**Для адресных пространств, созданных WLM:**

- Если WLM работает в целевом режиме:

- Используйте команду MVS

```
VARY WLM,APPLENV=имя,REFRESH
```

чтобы обновить языковую среду, если нужно загрузить новую версию хранимой процедуры. *имя* – имя среды прикладных программ WLM, связанной с группой хранимых процедур. Это означает, что выполнение этой команды затрагивает все хранимые процедуры, связанные с данной средой прикладных программ.

- Используйте команду MVS

```
VARY WLM,APPLENV=имя,QUIESCE
```

чтобы остановить все адресные пространства хранимых процедур, связанные со средой прикладных программ WLM *имя*.

- Используйте команду MVS

```
VARY WLM,APPLENV=имя,RESUME
```

чтобы запустить все адресные пространства хранимых процедур, связанные со средой прикладных программ WLM *имя*.

Дополнительную информацию о команде VARY WLM смотрите в руководстве *OS/390 MVS Planning: Workload Management*.

- Если WLM работает в режиме совместимости:

- Используйте команду MVS

```
CANCEL имя-адресного-пространства
```

чтобы остановить адресное пространство хранимых процедур, созданное WLM.

- Используйте команду MVS

```
START имя-адресного-пространства
```

чтобы запустить адресное пространство хранимых процедур, созданное WLM.

В режиме совместимости при обновлении языковой среды необходимо остановить и запустить адресные пространства хранимых процедур.

## Перенос хранимых процедур в среду, созданную WLM (для системных администраторов)

Если подсистема DB2 установлена в системе OS/390 выпуска 3 или более позднего, можно выполнять некоторые или все из хранимых процедур в адресных пространствах, созданных WLM. Чтобы перенести хранимые процедуры из среды, созданной DB2, в среду, созданную WLM, выполните следующие действия:

1. Определите процедуры JCL для адресных пространств хранимых процедур

В элементе DSNTIJMV набора данных DSN610.SDSNSAMP содержится пример процедур JCL для запуска адресных пространств, созданных WLM.

2. Определите среды прикладных программ WLM для групп хранимых процедур и свяжите с каждой средой прикладных программ процедуру запуска JCL.

Информацию о том, как это сделать, смотрите в разделе 5 (том 2) руководства *DB2 Administration Guide*.
3. Введите команду DB2 STOP PROCEDURE(\*), чтобы остановить все операции адресном пространстве хранимых процедур, созданном DB2.
4. Для каждой хранимой процедуры выполните команду ALTER PROCEDURE с параметром WLM ENVIRONMENT, чтобы задать имя среды прикладных программ.
5. Заново свяжите все существующие хранимые процедуры с DSNRLI, модулем языкового интерфейса для средства подключений службы управления восстановимыми ресурсами (RRSAF). Используйте JCL и операторы управления редактора связей, как это показано на рис. 144.

```
//LINKRRS EXEC PGM=IEWL,  
//      PARM='LIST,XREF,RENT,AMODE=31,RMODE=ANY'  
//SYSPRINT DD SYSOUT=*  
//SYSLIB   DD DISP=SHR,DSN=USER.RUNLIB.LOAD  
//          DD DISP=SHR,DSN=DSN610.SDSNLOAD  
//SYSLMOD  DD DISP=SHR,DSN=USER.RUNLIB.LOAD  
//SYSUT1   DD SPACE=(1024,(50,50)),UNIT=SYSDA  
//SYSLIN   DD DISP=SHR,DSN=DSN610.SDSNLOAD  
      ENTRY STORPROC  
      REPLACE DSNALI(DSNRLI)  
      INCLUDE SYSLMOD(STORPROC)  
      NAME STORPROC(R)
```

Рисунок 144. Связывание существующих хранимых процедур с RRSAF

6. Если WLM работает в режиме совместимости, используйте для запуска новых адресных пространств хранимых процедур, созданных WLM, команду MVS

START имя-адресного-пространства

Если WLM работает в целевом режиме, адресные пространства запускаются автоматически.

## Переопределение хранимых процедур, определенных в SYSIBM.SYSPROCEDURES

В предыдущих версиях DB2 (до Версии 6) для определения хранимых процедур для DB2 использовалось добавление строк в таблицу каталога SYSIBM.SYSPROCEDURES. При перенастройке в DB2 Версии 6 система DB2 автоматически создает новые определения для старых хранимых процедур в SYSIBM.SYSROUTINES и определения для параметров хранимых процедур в SYSIBM.SYSPARMS. Но если в определении старой хранимой процедуры заданы значения AUTHID или LUNAME, DB2 не сможет создать для нее новое определение; такую хранимую процедуру необходимо переопределить вручную.

Чтобы проверить, не заданы ли в определении хранимой процедуры непустые значения для AUTHID или LUNAME, выполните следующий запрос:

```
SELECT * FROM SYSIBM.SYSPROCEDURES  
WHERE AUTHID<>' ' OR LUNAME<>' ';
```

Затем используйте команду CREATE PROCEDURE, чтобы создать определения для всех хранимых процедур, найденных этим оператором SELECT. Не задавайте в CREATE PROCEDURE AUTHID или LUNAME. AUTHID и LUNAME позволяли определить различные версии хранимой процедуры, например, отладочную и рабочую версии. Для этой цели можно задать уникальные имена схем для хранимых процедур с одним именем. Например, для хранимой процедуры INVENTORY можно определить TEST.INVENTORY и PRODTN.INVENTORY.

## Написание и подготовка хранимой процедуры

Хранимая процедура – это прикладная программа DB2, которая выполняется в адресном пространстве хранимых процедур. Она может содержать большинство операторов, используемых в обычных прикладных программах.

Хранимая процедура очень похожа на любую другую программу SQL. Она может содержать статические или динамические операторы SQL, вызовы IFI и команды DB2, передаваемые через IFI. В этом разделе рассматриваются следующие темы:

- Требования к языкам хранимой процедуры и вызывающей программы
- “Вызов других программ” на стр. 594
- “Использование повторно–входимого кода” на стр. 594
- “Написание хранимой процедуры в виде основной программы или подпрограммы” на стр. 595
- “Обращение в хранимой процедуре к другим узлам” на стр. 601
- “Написание хранимой процедуры, возвращающей наборы результатов клиенту DRDA” на стр. 603
- “Подготовка хранимой процедуры” на стр. 604
- “Связывание хранимой процедуры” на стр. 606

## Требования к языкам хранимой процедуры и вызывающей программы

Хранимая процедура может быть написана на языках ассемблер, C, C++, COBOL или PL/I. Все программы должны быть разработаны для работы с использованием языковой среды. Хранимые процедуры, написанные на языках COBOL и C++, могут содержать объектно–ориентированные расширения. Информацию о включении объектно–ориентированных расширений в прикладные программы SQL смотрите в разделах “Особенности C++” на стр. 181 и “Особенности объектно–ориентированных расширений в языке COBOL” на стр. 204. Список минимальных требований к компилятору и языковой среде смотрите в руководстве *DB2 Release Guide*.

Программа, вызывающая хранимую процедуру, может быть написана на любом языке, поддерживающем оператор SQL CALL. В прикладных программах ODBC для передачи DB2 вызова хранимой процедуры можно использовать условие ESCAPE.

## Вызов других программ

Хранимая процедура может состоять из нескольких программ, у каждой из которых есть свой собственный пакет. Хранимая процедура может вызывать другие программы, хранимые процедуры и пользовательские функции. Для вызова других программ применяются средства используемого языка программирования.

Если хранимая процедура вызывает другие программы, содержащие операторы SQL, у каждой из этих вызываемых программ должен быть пакет DB2. Владелец пакета или плана, содержащего оператор CALL, должен обладать полномочиями EXECUTE для всех пакетов, используемых другими программами.

Когда хранимая процедура вызывает другую программу, DB2 определяет, к какому собранию принадлежит пакет вызываемой программы, используя один из следующих способов:

- Если хранимая процедура выполняет SET CURRENT PACKAGESET, пакет вызываемой программы относится к собранию, заданному в SET CURRENT PACKAGESET.
- Если хранимая процедура не выполняет SET CURRENT PACKAGESET,
  - Если определение хранимой процедуры содержит NO COLLID, DB2 использует ID собрания для пакета, содержащего оператор SQL CALL.
  - Если определение хранимой процедуры содержит COLLID *ID*—собрания, DB2 использует *ID*—собрания.

При возвращении управления из хранимой процедуры DB2 восстанавливает значение специального регистра CURRENT PACKAGESET, записывая в него значение, которое он имел перед выполнением оператора SQL CALL программой клиента.

## Использование повторно–входимого кода

Во всех случаях, когда это возможно, делайте хранимые процедуры повторно–входимыми. Использование таких процедур может улучшить производительность, поскольку:

- Повторно–входимые процедуры не требуется загружать в память при каждом их вызове.
- Одна копия хранимой процедуры в адресном пространстве хранимых процедур может одновременно использоваться несколькими задачами. При этом уменьшается объем виртуальной памяти, используемой для размещения кода в адресном пространстве хранимых процедур.

Чтобы подготовить повторно–входимую хранимую процедуру, скомпилируйте ее как повторно–входимую и скомпонуйте как повторно–входимую и повторно используемую.

Инструкции по компиляции повторно–входимых программ смотрите в руководстве по соответствующему языку программирования. Информацию об использовании компоновщика для создания повторно–входимых и повторно используемых загрузочных модулей смотрите в руководстве *DFSMS/MVS®: Program Management*.

Чтобы сделать повторно–входимую хранимую процедуру постоянно хранящейся в памяти, задайте STAY RESIDENT YES в операторе CREATE PROCEDURE или ALTER PROCEDURE для этой хранимой процедуры.

Если хранимая процедура не может быть повторно–входимой, скомпонуйте как не повторно–входимую и не повторно используемую. Атрибут невозможности повторного использования предотвращает одновременное использование одной копии хранимой процедуры несколькими задачами. Не повторно–входимые хранимые процедуры не должны оставаться в памяти. Поэтому необходимо задать STAY RESIDENT NO в операторе CREATE PROCEDURE или ALTER PROCEDURE для этой хранимой процедуры.

## Написание хранимой процедуры в виде основной программы или подпрограммы

Хранимая процедура, выполняющаяся в адресном пространстве, созданном WLM, и использующая языковую среду Выпуска 1.7 или более позднего, может быть или основной программой, или подпрограммой. Хранимая процедура–подпрограмма может работать производительнее, так как языковая среда выполняет для нее меньшее число операций.

В общем случае подпрограмма должна выполнить следующие дополнительные операции, которые для основной программы выполняются языковой средой:

- Инициализация и очистка данных
- Выделение и освобождение памяти
- Закрытие перед выходом из подпрограммы всех открытых файлов

При написании хранимой процедуры в виде подпрограммы следуйте приводимым ниже правилам:

- Следуйте правилам данного языка программирования для написания подпрограмм. Например, нельзя выполнять операции ввода–вывода в подпрограммах на языке PL/I.
- Избегайте использования операторов, которые при завершении программы вызывают окончание работы экземпляра языковой среды. Примеры таких операторов: STOP или EXIT в подпрограмме на языке PL/I, STOP RUN в подпрограмме на языке COBOL. Если экземпляр завершает работу при завершении хранимой процедуры, а программа клиента вызывает другую хранимую процедуру, выполняющуюся в виде подпрограммы, языковая среда должна будет создать новый экземпляр. В результате теряются преимущества от написания хранимой процедуры в виде подпрограммы.

В Табл. 55 кратко описано, как задать основную программу или же подпрограмму.

Таблица 55 (Стр. 1 из 2). Характеристики основных программ и подпрограмм

Язык	Основная программа	Подпрограмма
Ассемблер	В вызове макрокоманды CEEENTRY задано MAIN=YES.	В вызове макрокоманды CEEENTRY задано MAIN=NO.

Таблица 55 (Стр. 2 из 2). Характеристики основных программ и подпрограмм

Язык	Основная программа	Подпрограмма
C	Содержит функцию main(). Ее параметры передаются через argc и argv.	Вызываемая функция. Ее параметры передаются явно.
COBOL	Программа на языке COBOL, которая не завершается GOBACK	Динамически загружаемая подпрограмма, которая завершается GOBACK
PL/I	Содержит процедуру, объявленную с OPTIONS(MAIN)	Процедура, объявленная с OPTIONS(FETCHABLE)

На рис. 145 показан пример хранимой процедуры, написанной на языке С в виде подпрограммы.

```
/********************************************/
/* Эта подпрограмма на языке С – хранимая          */
/* процедура с методом передачи параметров GENERAL,   */
/* принимающая 3 параметра.                         */
/********************************************/
#pragma linkage(cfunc,fetchable)
#include <stdlib.h>
void cfunc(char p1[11],long *p2,short *p3)
{
   /********************************************/
    /* Объявление переменных, используемых операторами SQL.      */
    /* Эти переменные являются локальными для подпрограммы;        */
    /* значения параметров нужно скопировать в эти локальные       */
    /* переменные и потом обратно.                                     */
   /****************************************/>
EXEC SQL BEGIN DECLARE SECTION;
    char parm1[11];
    long int parm2;
    short int parm3;
EXEC SQL END DECLARE SECTION;
```

Рисунок 145 (Часть 1 из 2). Хранимая процедура на языке С в виде подпрограммы

```

/*******************************/
/* Копирование значений входных параметров в локальные */
/* переменные.                                              */
/*******************************/
strcpy(parm1,p1);
parm2 = *p2;
parm3 = *p3;
/*******************************/
/* Выполнение операций с локальными переменными.          */
/*******************************/

:::

/*******************************/
/* Задание значений, возвращаемых вызывающей программе. */
/*******************************/
strcpy(parm1,"SETBYSP");
parm2 = 100;
parm3 = 200;
/*******************************/
/* Копирование значений в выходные параметры.           */
/*******************************/
strcpy(p1,parm1);
*p2 = parm2;
*p3 = parm3;
}

```

*Рисунок 145 (Часть 2 из 2). Хранимая процедура на языке C в виде подпрограммы*

На рис. 146 на стр. 598 показан пример хранимой процедуры, написанной на языке C++ в виде подпрограммы.

```

/*****************/
/* Эта подпрограмма на языке C++ - хранимая */
/* процедура с методом передачи параметров GENERAL, */
/* принимающая 3 параметра. */
/* Оператор extern обязателен. */
/*****************/
extern "C" void cppfunc(char p1[11],long *p2,short *p3);
#pragma linkage(cppfunc,fetchable)
#include <stdlib.h>
    EXEC SQL INCLUDE SQLCA;
void cppfunc(char p1[11],long *p2,short *p3)
{
/*****************/
/* Обявление переменных, используемых операторами SQL. */
/* Эти переменные являются локальными для подпрограммы; */
/* значения параметров нужно скопировать в эти локальные */
/* переменные и потом обратно. */
/*****************/
EXEC SQL BEGIN DECLARE SECTION;
    char parm1[11];
    long int parm2;
    short int parm3;
EXEC SQL END DECLARE SECTION;
/*****************/
/* Копирование значений входных параметров в локальные */
/* переменные. */
/*****************/
strcpy(parm1,p1);
parm2 = *p2;
parm3 = *p3;
/*****************/
/* Выполнение операций с локальными переменными. */
/*****************/
:
/*****************/
/* Задание значений, возвращаемых вызывающей программе. */
/*****************/
strcpy(parm1,"SETBYSP");
parm2 = 100;
parm3 = 200;
/*****************/
/* Копирование значений в выходные параметры. */
/*****************/
strcpy(p1,parm1);
*p2 = parm2;
*p3 = parm3;
}

```

*Рисунок 146. Хранимая процедура на языке C++ в виде подпрограммы*

Хранимая процедура, выполняющаяся в адресном пространстве, созданном DB2, должна содержать основную программу.

## Ограничения для хранимых процедур

- Не используйте в хранимой процедуре явных вызовов средства подключения. Хранимые процедуры, выполняемые в адресном пространстве, созданном DB2, используют неявные вызовы средства подключения вызовов (CAF). Хранимые процедуры, выполняемые в адресном пространстве, созданном WLM, используют неявные вызовы средства подключений службы управления восстановимыми ресурсами (RRSAF). Если хранимая процедура выполняет явный вызов средства подключения, DB2 отклоняет этот вызов.
- Не включайте в хранимую процедуру следующие операторы SQL:
  - COMMIT
  - SET CURRENT SQLID

Если DB2 обнаруживает такие операторы, она помещает поток DB2 в состояние *необходим откат*. Когда управление возвращается вызывающей программе, она должна выполнить одно из следующих действий:

- Выполнить оператор ROLLBACK, чтобы после выполнения этого оператора можно было выполнять другие операторы SQL.
- Завершить работу, вызывая тем самым автоматический откат рабочей единицы.

**Использование в хранимой процедуре операторов ROLLBACK:** Можно поместить в хранимую процедуру операторы ROLLBACK, обеспечивающие откат работы DB2 в ситуации ошибки. Например, предположим, что создается хранимая процедура, выбирающая значения из таблицы TABLEA и использующая эти значения для обновления таблицы TABLEB. Если в таблице TABLEA обнаружено некоторое значение – признак того, что эта таблица содержит неправильные данные, необходимо отменить соответствующие изменения в таблице TABLEB. Если в хранимой процедуре выполняется оператор типа:

```
IF VALUEA=BADVAL THEN  
    EXEC SQL ROLLBACK;
```

оператор ROLLBACK в случае ошибки переведет поток для этой хранимой процедуры в состояние *необходим откат*. В результате программа, вызывающая эту хранимую процедуру, отменит изменения в TABLEB.

## Использование в хранимой процедуре специальных регистров

В хранимой процедуре можно использовать все специальные регистры. Однако изменять можно только значения некоторых из этих специальных регистров. После завершения хранимой процедуры система DB2 восстанавливает значения всех специальных регистров, возвращая им те значения, которые они имели перед вызовом процедуры.

В Табл. 56 на стр. 600 представлена информация об использовании специальных регистров в хранимой процедуре.

Таблица 56. Характеристики специальных регистров для хранимой процедуры

Специальный регистр	Начальное значение	Может ли использовать оператор SET для изменения значения?
CURRENT DATE	Новое значение для каждого оператора SQL в пакете хранимой процедуры <sup>1</sup>	Нет <sup>4</sup>
CURRENT DEGREE	Значение наследуется от вызывающей программы <sup>2</sup>	Да
CURRENT LOCALE LC_TYPE	Значение наследуется от вызывающей программы	Да
CURRENT OPTIMIZATION HINT	Значение опции связывания OPTHINT для пакета хранимой процедуры или значение, которое наследуется от вызывающей программы <sup>5</sup>	Да
CURRENT PACKAGESET	Значение наследуется от вызывающей программы <sup>3</sup>	Да
CURRENT PATH	Значение опции связывания PATH для пакета хранимой процедуры или значение, которое наследуется от вызывающей программы <sup>5</sup>	Да
CURRENT PRECISION	Значение наследуется от вызывающей программы	Да
CURRENT RULES	Значение наследуется от вызывающей программы	Да
CURRENT SERVER	Значение наследуется от вызывающей программы	Да
CURRENT SQLID	Первичный ID авторизации процесса прикладной программы или значение, которое наследуется от вызывающей программы <sup>6</sup>	Да <sup>7</sup>
CURRENT TIME	Новое значение для каждого оператора SQL в пакете хранимой процедуры <sup>1</sup>	Нет <sup>4</sup>
CURRENT TIMESTAMP	Новое значение для каждого оператора SQL в пакете хранимой процедуры <sup>1</sup>	Нет <sup>4</sup>
CURRENT TIMEZONE	Значение наследуется от вызывающей программы	Нет <sup>4</sup>
CURRENT USER	Первичный ID авторизации процесса прикладной программы	Нет <sup>4</sup>

Примечания к Табл. 56:

1. Если хранимая процедура вызывается из области действия триггера, DB2 использует временную отметку оператора SQL триггера в качестве временной отметки для всех операторов SQL в пакете хранимой процедуры.
2. DB2 допускает параллелизм только на одном уровне вложенного оператора SQL. Если специальный регистр CURRENT DEGREE имеет значение ANY, а параллелизм запрещен, DB2 не учитывает значения CURRENT DEGREE.
3. Если для хранимой процедуры задано значение для COLLID в операторе CREATE PROCEDURE, DB2 присваивает специальному регистру CURRENT PACKAGESET значение COLLID.
4. Не используется, так как отсутствует оператор SET для этого специального регистра.
5. Если какая–либо программа в области видимости вызывающей программы выдает оператор SET для этого специального регистра перед вызовом этой хранимой процедуры, этот специальный регистр наследует значение, заданное этим оператором SET. В противном случае этот специальный регистр содержит значение, заданное опцией связывания для пакета хранимой процедуры.
6. Если какая–либо программа в области видимости вызывающей программы выдает оператор SET CURRENT SQLID для этого специального регистра перед вызовом этой хранимой процедуры, этот специальный регистр наследует значение, заданное этим оператором SET. В противном случае специальный регистр CURRENT SQLID содержит ID авторизации процесса прикладной программы.
7. Если пакет хранимой процедуры использует значение, отличающееся от значения RUN для опции связывания DYNAMICRULES, оператор SET CURRENT SQLID может быть выполнен, но не повлияет на ID авторизации, использующийся для динамических операторов SQL в пакете хранимой процедуры. Значение DYNAMICRULES определяет ID авторизации, используемый для динамических операторов SQL. Дополнительные сведения о значениях DYNAMICRULES и ID авторизациисмотрите в разделе “Выбор стратегии динамических операторов SQL с помощью DYNAMICRULES” на стр. 457.

## Обращение в хранимой процедуре к другим узлам

Хранимая процедура может обращаться к таблицам на других системах DB2, используя трехчастные имена объектов или оператор CONNECT. Если используется оператор CONNECT, для обращения к таблицам используется доступ через DRDA. Если используются трехчастные имена объектов или алиасы для трехчастных имен объектов, метод доступа к распределенным данным зависит от значения DBPROTOCOL, заданного при связывании пакета хранимой процедуры. Если параметр связывания DBPROTOCOL не задан, метод доступа к распределенным данным зависит от значения поля DATABASE PROTOCOL панели установки DSNTIP5. Значение PRIVATE указывает DB2, что для обращения к удаленным данным для хранимой процедуры нужно использовать доступ через собственный протокол DB2. Значение DRDA указывает DB2, что нужно использовать доступ через DRDA.

Если локальная прикладная программа DB2 вызывает хранимую процедуру, эта хранимая процедура не может обращаться через собственный протокол

DB2 к какими-либо узлам DB2, уже соединенными с вызывающей программой через DRDA.

Локальная прикладная программа DB2 не может использовать доступ через DRDA для соединения с какими-либо узлами, к которым хранимая процедура уже обращалась через собственный протокол DB2. Перед образованием соединения через собственный протокол DB2, локальная прикладная программа DB2 должна сначала выполнить оператор RELEASE, чтобы завершить соединение через собственный протокол DB2 и затем выполнить принятие рабочей единицы.

## **Написание хранимой процедуры, обращающейся к базам данных IMS**

Поддержка IMS Open Database Access (ODBA) позволяет хранимой процедуре DB2 соединяться с системой IMS DBCTL или IMS DB/DC и выполнять вызовы DL/I для обращения к базам данных IMS.

Поддержка ODBA использует OS/390 RRS для управления точками синхронизации ресурсов DB2 и IMS. Поэтому хранимые процедуры, использующие ODBA, могут работать только в адресных пространствах, управляемых WLM.

Когда вы пишете хранимую процедуру, использующую ODBA, соблюдайте правила написания программ IMS, выполняющих вызовы DL/I. Сведения о написании программ DL/I смотрите в руководствах *IMS/ESA Application Programming: Database Manager* и *IMS/ESA Application Programming: Transaction Manager*.

Действия IMS, которые выполняются в хранимой процедуре, находятся в той же области принятия, что и сама хранимая процедура. Как и для любой другой хранимой процедуры, принятие выполняет вызывающая программа.

Хранимая процедура, которая использует ODBA, должна выполнить вызов DPSB PREP, чтобы освободить PSB после завершения работы IMS под этим PSB. Ключевое слово PREP сообщает IMS, что текущую работы надо перевести в неоднозначное состояние. Если работа находится в неоднозначном состоянии, IMS не требует активации обработки точек синхронизации при выполнении вызова DPSB. IMS выполняет принятие или откат работы, как часть процесса двухфазного принятия RRS, когда вызывающая хранимую процедуру программа выполняет COMMIT или ROLLBACK.

Пример хранимой процедуры на языке COBOL и клиентской программы показывает обращение к данным IMS с использованием интерфейса ODBA. Исходный текст хранимой процедуры находится в компоненте DSN8EC1 и подготавливается заданием DSNTEJ61. Исходный текст вызывающей программы находится в компоненте DSN8EC1 и подготавливается заданием DSNTEJ62. Все тексты находятся в наборе данных DSN610.SDSNSAMP.

Процедура запуска для адресного пространства хранимых процедур, использующих ODBA, должна включать оператор DFSRESLB DD и дополнительный набор данных в STEPLIB. Дополнительную информацию смотрите в разделе “Настройка среды хранимых процедур” на стр. 584.

## **Написание хранимой процедуры, возвращающей наборы результатов клиенту DRDA**

Хранимая процедура может возвращать клиенту DRDA несколько наборов результатов запроса, если удовлетворяются следующие условия:

- Клиент поддерживает коды DRDA, используемые для возвращения наборов результатов запроса.
- RESULT SETS в определении хранимой процедуры имеет значение больше 0.

Для каждого возвращаемого набора результатов хранимая процедура должна:

- Объявить указатель с опцией WITH RETURN.
- Открыть этот указатель.
- Оставить этот указатель открытым.

При завершении хранимой процедуры DB2 возвращает клиенту строки в наборе результатов запроса.

DB2 не возвращает наборы результатов для указателей, которые были закрыты до завершения хранимой процедуры. Хранимая процедура должна выполнить оператор CLOSE для каждого указателя, связанного с набором результатов, который не должен быть возвращен клиенту DRDA.

**Пример:** Предположим, что должен возвращаться набор результатов, содержащий записи для всех сотрудников отдела D11. Сначала объявитите указатель, описывающий это подмножество сотрудников:

```
EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR  
SELECT * FROM DSN8610.EMP  
      WHERE WORKDEPT='D11';
```

Затем откройте этот указатель:

```
EXEC SQL OPEN C1;
```

DB2 возвращает клиенту набор результатов и имя этого указателя SQL в хранимой процедуре.

**Используйте осмысленные имена указателей для возвращаемого набора результатов:** Имя указателя, используемого для возвращения набора результатов, становится доступным прикладной программе клиента посредством расширений оператора DESCRIBE. Дополнительную информацию смотрите в разделе “Написание клиентской программы DB2 for OS/390, получающей наборы результатов” на стр. 637.

Используйте имена указателей, осмысленные в контексте прикладной программы клиента DRDA, особенно если хранимая процедура возвращает несколько наборов результатов.

**Объекты, из которых могут возвращаться наборы результатов:** В операторе SELECT, связанном с указателем для набора результатов, можно использовать любые из следующих объектов:

- Таблицы, синонимы, производные таблицы, временные таблицы и алиасы, определенные в локальной системе DB2
- Таблицы, синонимы, производные таблицы, временные таблицы и алиасы, определенные на удаленных системах DB2 for OS/390, доступных через собственный протокол DB2

**Возвращение клиенту подмножества строк:** Если с указателем набора результатов выполняются операторы FETCH, DB2 не возвращает программе клиента выбранные строки. Например, если объявлен указатель с опцией WITH RETURN и затем выполняются операторы OPEN, FETCH, FETCH, клиент получит данные, начинающиеся с третьей строки набора результатов.

**Использование временной таблицы для возвращения наборов результатов:** Для возвращения наборов результатов из хранимой процедуры можно использовать временную таблицу. Эту возможность можно использовать для возвращения нереляционных данных клиенту DRDA.

Например, из хранимой процедуры можно обращаться к данным IMS следующим образом:

- Используйте MVS/APPC, чтобы запустить транзакцию IMS.
- Получите ответное сообщение IMS, содержащее данные, которые нужно вернуть клиенту.
- Вставьте данные из этого ответного сообщения во временную таблицу.
- Откройте указатель на эту временную таблицу. После завершения хранимой процедуры строки этой временной таблицы возвращаются клиенту.

## Подготовка хранимой процедуры

Существует ряд задач, которые нужно выполнить перед использованием хранимой процедуры, чтобы ее можно было выполнять на сервере MVS. Часть этих задач выполняет системный администратор. Необходимые действия системного администратора описаны в разделе “Определение хранимой процедуры для DB2” на стр. 585 и разделе 2 руководства *DB2 Installation Guide*.

Выполните следующие шаги:

1. Выполните прекомпиляцию и компиляцию прикладной программы.  
Если хранимая процедура – это программа на языке COBOL, ее нужно скомпилировать с опцией NODYNAM.
2. Скомпонуйте прикладную программу. Для хранимой процедуры должен быть скомпонован или загружен один из следующих модулей языкового интерфейса:

**DSNALI** Модуль языкового интерфейса для средства подключения по вызову (CAF). Скомпонуйте или загрузите этот модуль, если хранимая процедура выполняется в адресном пространстве, созданном DB2. Дополнительную информацию смотрите в разделе “Обращение к языковому интерфейсу CAF” на стр. 776.

**DSNRLI** Модуль языкового интерфейса для средства подключений службы управления восстановимыми ресурсами. Скомпонуйте или загрузите этот модуль, если хранимая процедура выполняется в адресном пространстве, созданном DB2. Модуль DSNRLI необходимо скомпоновать или загрузить, если хранимая процедура обращается к большим объектам или особым типам. Дополнительную информацию смотрите в разделе “Обращение к языковому интерфейсу RRSAF” на стр. 811.

Если хранимая процедура выполняется в адресном пространстве, созданном WLM, необходимо задать при компоновке параметр AMODE(31).

3. Свяжите DBRM с DB2 при помощи команды BIND PACKAGE. Для хранимых процедур требуется только пакет на сервере. Не нужно связывать план. Дополнительную информацию смотрите в разделе “Связывание хранимой процедуры” на стр. 606.
4. Определите хранимую процедуру для DB2.
5. Используйте команду GRANT EXECUTE, чтобы задать соответствующим пользователям полномочия на использование этой хранимой процедуры. Например,

```
GRANT EXECUTE ON PROCEDURE SPSCHEMA.STORPRCA TO JONES;
```

Эта команда разрешает прикладным программам, выполняемым с ID авторизации JONES, вызывать хранимую процедуру SPSCHEMA.STORPRCA.

**Подготовка хранимой процедуры для выполнения в качестве авторизованной программы:** Если хранимая процедура выполняется в адресном процессе, созданном WLM, ее можно выполнять как авторизованную программу MVS. Чтобы подготовить хранимую процедуру для работы в качестве авторизованной программы, выполните следующие дополнительные действия:

- При компоновке хранимой процедуры:
  - Задав значение параметра AC=1, укажите, что загрузочный модуль может использовать ограниченные средства системных служб.
  - Поместите загрузочный модуль для этой хранимой процедуры в библиотеку авторизованных APF программ.
- Убедитесь, что хранимая процедура выполняется в адресном пространстве, для которого все библиотеки, заданные в параметре STEPLIB процедуры запуска, авторизованы APF. Задайте параметр среди прикладных программ WLM ENVIRONMENT оператора CREATE PROCEDURE или ALTER PROCEDURE для хранимой процедуры, чтобы хранимая процедура выполнялась именно в таком адресном пространстве.

## Связывание хранимой процедуры

Для хранимой процедуры не требуется план DB2. Хранимая процедура выполняется в потоке вызывающей программы, используя план этой вызывающей клиентской программы клиента.

Для выполнения оператора CALL вызывающая прикладная программа может использовать пакет или план DB2. Хранимая процедура должна использовать пакет DB2, как это показано на рис. 147.

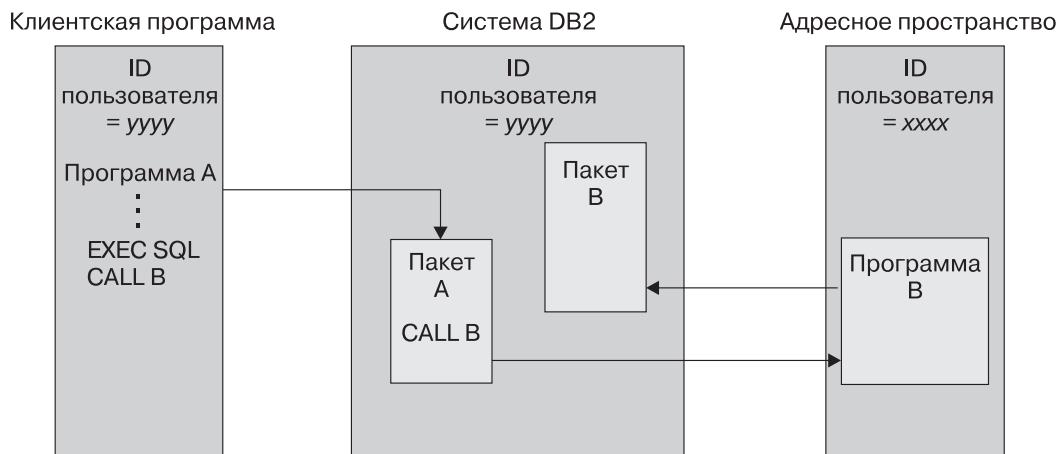


Рисунок 147. Среда выполнения хранимых процедур

При связывании хранимой процедуры:

- Для связывания хранимой процедуры используйте команду BIND PACKAGE. Если для управления доступом к пакету хранимой процедуры используется опция ENABLE, необходимо разрешить системный тип соединений для прикладной программы, выполняющей оператор CALL.
- Пакет хранимой процедуры не требуется связывать с планом программы, вызывающей эту процедуру, за исключением случаев, когда хранимая процедура и вызывающая ее программа расположены в одной и той же системе.
- Владелец пакета, содержащего оператор SQL CALL, должен обладать полномочиями EXECUTE для всех пакетов, к которым обращается эта хранимая процедура, включая пакеты, заданные в SET CURRENT PACKAGESET.

Как показано на рис. 147, на сервере должны существовать:

- План или пакет, содержащий оператор SQL CALL. Этот пакет связан с программой клиента.
- Пакет связанный с хранимой процедурой.

Программа сервера может использовать несколько пакетов. Эти пакеты берутся из двух источников:

- DBRM, связанного несколько раз с различными версиями одного пакета с тем же самым именем; они могут быть расположены в разных собраниях. Хранимая процедура может переключаться с одной версии пакета на другую при помощи оператора SET CURRENT PACKAGESET.

- Пакет, связанный с другой программой, содержащей операторы SQL и вызываемой хранимой процедурой.

## Написание и подготовка прикладной программы, использующей хранимые процедуры

Для вызова хранимой процедуры и передачи ей списка параметров используйте оператор SQL CALL. Синтаксис и полное описание оператора CALL смотрите в главе 6 руководства *DB2 SQL Reference*.

Прикладная программа, вызывающая хранимую процедуру, может:

- Вызывать несколько хранимых процедур
- Выполнить оператор CALL локально или послать оператор CALL на сервер. Прикладная программа выполняет оператор CONNECT для соединения с сервером и затем выполняет оператор CALL или использует трехчастное имя для явного соединения с сервером, на котором находится хранимая процедура.
- После соединения сервером смешать операторы CALL с другими операторами SQL.

Используйте для выполнения оператора CALL один из этих методов:

- Выполните статический оператор CALL.
- Используйте условие ESCAPE в прикладной программе ODBC для передачи DB2 оператора CALL.

- Использовать любые средства подключения DB2

## Формы оператора CALL

Простейшая форма оператора CALL выглядит следующим образом:

EXEC SQL CALL A (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME, :TYPE, :CODE);

где :EMP, :PRJ, :ACT, :EMT, :EMS, :EME, :TYPE и :CODE – переменные хоста, объявленные ранее в прикладной программе. Используемый оператор CALL может отличаться от приведенной выше формы:

- Вместо передачи каждого параметра отдельно их можно передать вместе в виде структуры хоста. Например, если определить такую структуру хоста:

```
struct {  
    char EMP[7];  
    char PRJ[7];  
    short ACT;  
    short EMT;  
    char EMS[11];  
    char EME[11];  
} empstruc;
```

оператор CALL будет выглядеть так:

EXEC SQL CALL A (:empstruc, :TYPE, :CODE);

- Предположим, что A – схема SCHEMAA на удаленной системе LOCA. Для обращения к A можно использовать один из следующих методов:

- Выполнить оператор CONNECT для соединения с LOCA и затем выполнить оператор CALL:

```
CONNECT TO LOCA;
EXEC SQL CALL SCHEMAA.A (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME,
                           :TYPE, :CODE);
```

- Задать в операторе CALL трехчастное имя для A:

```
EXEC SQL CALL LOCA.SCHEMAA.A (:EMP, :PRJ, :ACT, :EMT, :EMS,
                               :EME, :TYPE, :CODE);
```

Преимущество второго метода в том, что нет необходимости выполнять оператор CONNECT. Недостаток – в том, что такая форма оператора CALL не может быть перенесена на другие платформы.

Если программа выполняет операторы ASSOCIATE LOCATORS или DESCRIBE PROCEDURE, необходимо использовать одну и ту же форму для имени процедуры в операторе CALL и в операторе ASSOCIATE LOCATORS или DESCRIBE PROCEDURE.

- В приведенном выше примере предполагается, что ни один из входных параметров не может иметь пустого значения. Чтобы разрешить пустые значения, запишите оператор в следующем виде:

```
EXEC SQL CALL A (:EMP :IEMP, :PRJ :IPRJ, :ACT :IACT,
                  :EMT :IEMT, :EMS :IEMS, :EME :IEME,
                  :TYPE :ITYPE, :CODE :ICODE);
```

где :IEMP, :IPRJ, :IACT, :IEMT, :IEMS, :IEME, :ITYPE и :ICODE – индикаторы пустых значений для параметров.

- Хранимой процедуре можно передать константы типа целое число, текстовая строка или же пустые значения, как показано в следующем примере:

```
EXEC SQL CALL A ('000130', 'IF1000', 90, 1.0, NULL, '1982-10-01',
                  :TYPE, :CODE);
```

- Имя хранимой процедуры можно задать как переменную хоста:

```
EXEC SQL CALL :procnm (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME,
                        :TYPE, :CODE);
```

Допустим, хранимая процедура называется 'A'. Переменная хоста *procnm* – символьная переменная длины 255 или меньше, содержащая значение 'A'. Этот метод можно использовать, если имя хранимой процедуры заранее не известно, хотя список ее параметров известен.

- Если желательно передавать параметры в виде одной структуры, а не в виде отдельных переменных хоста, можно использовать такую форму оператора:

```
EXEC SQL CALL A USING DESCRIPTOR :sqlda;
```

где *sqlda* – имя SQLDA.

- Можно выполнить оператор CALL, используя переменную хоста для имени хранимой процедуры и SQLDA для списка параметров:

```
EXEC SQL CALL :procnm USING DESCRIPTOR :sqlda;
```

Такая форма оператора обеспечивает особую гибкость, так как можно использовать один оператор CALL для вызовов различных хранимых процедур с различными списками параметров.

Программа клиента перед вызовом SQL CALL должна присвоить переменной хоста *procspmt* имя хранимой процедуры и загрузить в SQLDA информацию о параметрах.

Все приведенные выше примеры оператора CALL используют SQLDA. Если SQLDA не задана явно, препроцессор генерирует SQLDA на основе переменных в списке параметров.

## Полномочия на выполнение хранимых процедур

Для выполнения хранимой процедуры требуются два типа полномочий:

- Полномочия на выполнение оператора CALL
- Полномочия на выполнение пакета хранимой процедуры и всех пакетов, вызываемых в пакете хранимой процедуры.

Необходимые полномочия зависят от того, какая форма оператора CALL используется: CALL литерал или CALL :*переменная–хоста*.

Если хранимая процедура вызывает пользовательские функции или триггеры, требуются дополнительные полномочия на выполнение этих триггеров, пользовательских функций и пакетов этих пользовательских функций.

Дополнительную информацию смотрите в описании оператора CALL в главе 6 руководства *DB2 SQL Reference*.

## Соглашения по компоновке

Когда прикладная программа выполняет оператор CALL, DB2 создает список параметров для хранимой процедуры, используя параметры и значения, заданные в этом операторе. DB2 получает информацию о параметрах из определения хранимой процедуры, созданного при выполнении команды CREATE PROCEDURE. Параметры могут относиться к одному из следующих типов:

- IN** Параметры только для входных значений, передаваемых хранимой процедуре
- OUT** Параметры только для выходных значений, возвращаемых хранимой процедурой вызывающей программе
- INOUT** Параметры для входных или выходных значений, передаваемых хранимой процедуре или возвращаемых из нее.

Если хранимая процедура не может присвоить значения одному или нескольким выходным параметрам, DB2 не обнаруживает ошибку в хранимой процедуре. Вместо этого DB2 возвращает вызывающей программе эти выходные параметры с теми значениями, которые они имели на входе в хранимую процедуру.

**Инициализация выходных параметров:** Для хранимых процедур, выполняемых локально, перед вызовом нет необходимости инициализировать значения выходных параметров. Однако при вызове хранимой процедуры с удаленного узла локальная подсистема DB2 не может отличить входные (IN) параметры от выходных (OUT или INOUT). Поэтому перед вызовом хранимой процедуры с удаленного узла надо инициализировать значения всех выходных параметров.

При инициализации выходных параметров типа большой объект рекомендуется задавать для них нулевую длину. Это увеличит производительность программы.

DB2 поддерживает три метода передачи параметров. DB2 выбирает метод передачи параметров, исходя из значения параметра PARAMETER STYLE в определении хранимой процедуры: **GENERAL**, **GENERAL WITH NULLS** или **DB2SQL**.

- **GENERAL:** Используйте значение GENERAL, если не нужно, чтобы вызывающая программа передавала хранимой процедуре пустые значения входных параметров (IN или INOUT). В хранимой процедуре должно содержаться объявления переменных для всех параметров, передаваемых в операторе CALL.

На рис. 148 показана структура списка параметров для PARAMETER STYLE GENERAL.



Рисунок 148. Метод передачи параметров GENERAL для хранимой процедуры

- **GENERAL WITH NULLS:** Используйте значение GENERAL WITH NULLS, чтобы разрешить вызывающей программе задавать пустые значения для каких-либо параметров, передаваемых хранимой процедуре. При использовании метода передачи параметров GENERAL WITH NULLS хранимая процедура должна сделать следующее:

- Объявить переменную для каждого параметра, передаваемого в операторе CALL.
- Объявить структуру индикаторов пустых значений, содержащую индикатор пустого значения для каждого параметра.
- На входе в процедуру проверить все индикаторы пустых значений, связанные с входными параметрами, чтобы определить, какие параметры содержат нулевые значения.
- На выходе из процедуры присвоить значения всем индикаторам пустых значений, связанным с выходными переменными. Значение индикатора пустого значения для выходной переменной, возвращающей вызывающей программе пустое значение, должно быть отрицательным числом. В противном случае это значение должно быть равно 0.

В операторе CALL после каждого параметра задайте индикатор пустого значения, используя одну из следующих форм:

переменная-хоста :переменная-индикатор  
или  
переменная-хоста INDICATOR :индикатор-пустого-значения.

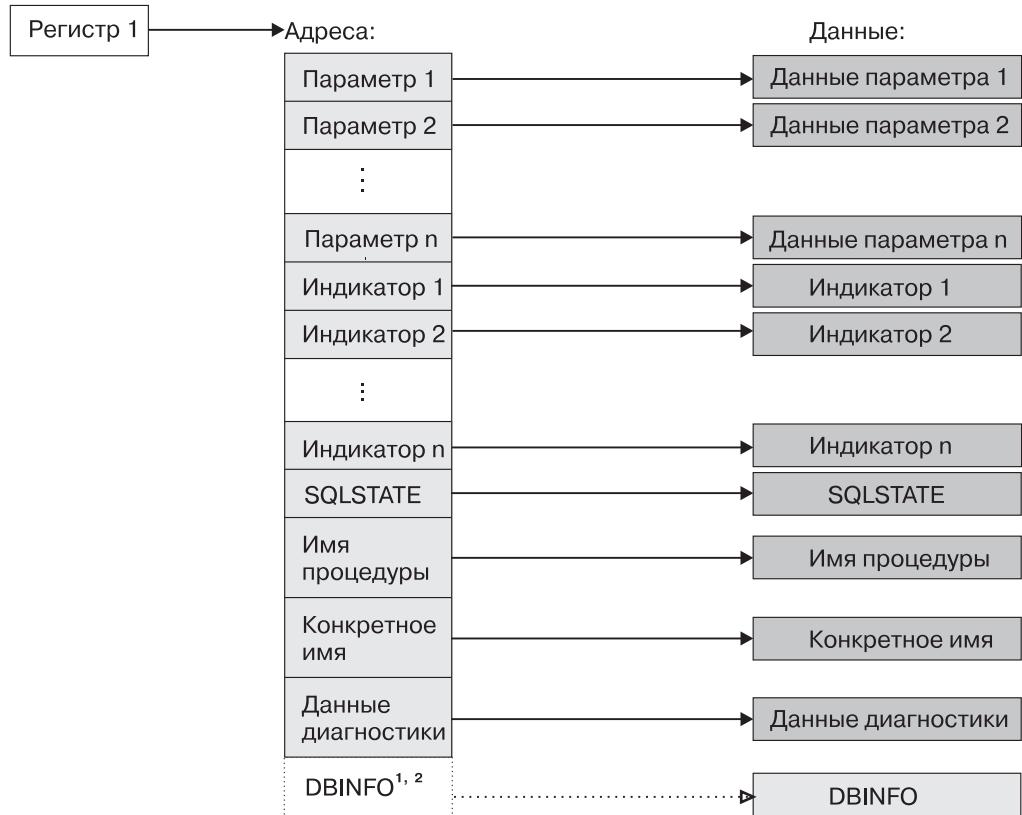
На рисунке рис. 149 на стр. 611 показана структура списка параметров для PARAMETER STYLE GENERAL WITH NULLS.



Рисунок 149. Метод передачи параметров GENERAL WITH NULLS для хранимой процедуры

- **DB2SQL:** Подобно GENERAL WITH NULLS, опция DB2SQL позволяет задавать пустые значения для любого параметра, передаваемого хранимой процедуре. Кроме того, DB2 передает хранимой процедуре входные и выходные параметры, содержащие следующую информацию:
  - SQLSTATE, который возвращается системе DB2. Это параметр CHAR(5), который может иметь те же значения, что и возвращаемые пользовательской функцией. Допустимые значения SQLSTATEсмотрите в разделе “Передача значений параметров в пользовательскую функцию и из пользовательской функции” на стр. 278.
  - Полное имя хранимой процедуры. Это значение VARCHAR(27).
  - Уникальное имя хранимой процедуры. Собственное имя – это значение VARCHAR(18), совпадающее с неспецифицированным именем.
  - Стока диагностики SQL, которая возвращается системе DB2. Это значение VARCHAR(70). Оно используется для передачи вызывающей программе описания ошибки или предупреждения.

На рис. 150 на стр. 612 показана структура списка параметров для PARAMETER STYLE DB2SQL.



<sup>1</sup> Для PL/I это значение - адрес указателя на данные DBINFO.

<sup>2</sup> Передается, если в определении пользовательской функции задана опция DBINFO

Рисунок 150. Метод передачи параметров DB2SQL для хранимой процедуры

### Примеры использования метода передачи параметров GENERAL для хранимой процедуры

В следующих примерах показано, как использовать метод передачи параметров GENERAL для входных параметров в хранимых процедурах, написанных на языках ассемблер, С, COBOL и PL/I. Примеры полного кода хранимых процедур и вызывающих их прикладных программ смотрите в разделе “Примеры использования хранимых процедур” на стр. 932.

В этих примерах предполагается, что прикладная программа на языке COBOL содержит следующие объявления параметров и оператор CALL:

```
*****
* ПАРАМЕТРЫ ДЛЯ ВЫЗОВА ОПЕРАТОРА SQL CALL *
*****
01 V1 PIC S9(9) USAGE COMP.
01 V2 PIC X(9).
⋮
EXEC SQL CALL A (:V1, :V2) END-EXEC.
```

В операторе CREATE PROCEDURE параметры определяются следующим образом:

```
V1 INT IN, V2 CHAR(9) OUT
```

На рисунках рис. 151, рис. 152, рис. 153 и рис. 154 показано, как хранимые процедуры, написанные на разных языках, получают такие параметры.

```
*****
* КОД ХРАНИМОЙ ПРОЦЕДУРЫ НА АССЕМБЛЕРЕ, ИСПОЛЬЗУЮЩИЙ МЕТОД      *
* ПЕРЕДАЧИ ПАРАМЕТРОВ GENERAL.                                         *
*****
A      CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
      USING PROGAREA,R13
*****
* ЗАДАНИЕ ЯЗЫКОВОЙ СРЕДЫ.                                              *
*****
:
*****
* ПОЛУЧЕНИЕ ПЕРЕДАННЫХ ЗНАЧЕНИЙ ПАРАМЕТРОВ. МЕТОД ПЕРЕДАЧИ      *
* ПАРАМЕТРОВ GENERAL СЛЕДУЕТ СТАНДАРТНОМУ МЕТОДУ ПЕРЕДАЧИ          *
* ПАРАМЕТРОВ ДЛЯ ЯЗЫКА АССЕМБЛЕР:                                     *
* НА ВХОДЕ В ПРОЦЕДУРУ РЕГИСТР 1 УКАЗЫВАЕТ НА СПИСОК УКАЗАТЕЛЕЙ   *
* НА ПАРАМЕТРЫ.                                                       *
*****
L      R7,0(R1)           ПОЛУЧИТЬ УКАЗАТЕЛЬ НА V1
MVC    LOCV1(4),0(R7)     СКОПИРОВАТЬ V1 В ЛОКАЛЬНУЮ ПЕРЕМЕННУЮ
:
L      R7,4(R1)           ПОЛУЧИТЬ УКАЗАТЕЛЬ НА V2
MVC    0(9,R7),LOCV2      СКОПИРОВАТЬ ЗНАЧЕНИЕ В ВЫХОДНОЙ ПАРАМЕТР V2
:
CEETERM RC=0
*****
* ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ И ЭКВИВАЛЕНТНЫХ ИМЕН                      *
*****
R1      EQU    1           РЕГИСТР 1
R7      EQU    7           РЕГИСТР 7
PPA     CEEPPA ,          КОНСТАНТЫ, ОПИСЫВАЮЩИЕ ЭТУ БЛОК КОДА
          LTORG ,
PROGAREA DSECT
          ORG    **+CEEDSASZ   ОСТАВИТЬ МЕСТО ДЛЯ ПОСТОЯННОЙ ЧАСТИ DSA
LOCV1   DS     F           ЛОКАЛЬНАЯ КОПИЯ ПАРАМЕТРА V1
LOCV2   DS     CL9          ЛОКАЛЬНАЯ КОПИЯ ПАРАМЕТРА V2
:
PROGSIZE EQU    *-PROGAREA
          CEEDSA ,          ОТображение области динамического хранения
          CEECAA ,          ОТображение области общей привязки
END    A
```

Рисунок 151. Пример использования метода передачи параметров GENERAL в программе на ассемблере

---

```
#pragma runopts(PLIST(OS))
#pragma options(RENT)
#include <stdlib.h>
#include <stdio.h>
/*********************************************************/
/* Код хранимой процедуры на языке C, использующей метод      */
/* передачи параметров GENERAL.                                     */
/*********************************************************/
main(argc,argv)
    int argc;                      /* Число переданных параметров */
    char *argv[];                  /* Массив строк, содержащих      */
                                    /* значения параметров         */
{
    long int locv1;                /* Локальная копия V1           */
    char locv2[10];                /* Локальная копия V2           */
                                    /* (с нулевым символом-ограничителем) */

    :
    /* Получение переданных параметров. Метод передачи параметров */
    /* GENERAL следует стандартным правилам передачи параметров */
    /* в языке C:                                                 */
    /* - argc содержит число переданных параметров               */
    /* - argv[0] - указатель на имя хранимой процедуры          */
    /* - argv[1] до argv[n] - указатели на n параметров        */
    /* в операторе SQL CALL.                                     */
    /* *********************************************************/
    if(argc==3)                   /* Нужно получить 3 параметра: */
    {                            /* procname, V1, V2            */
        locv1 = *(int *) argv[1];
                                    /* Получить локальную копию V1 */

        :
        strcpy(argv[2],locv2);
                                    /* Присвоить значение V2      */

        :
    }
}
```

---

Рисунок 152. Пример использования метода передачи параметров *GENERAL* в программе на языке C

---

```

CBL RENT
IDENTIFICATION DIVISION.
*****
* КОД ХРАНИМОЙ ПРОЦЕДУРЫ НА ЯЗЫКЕ COBOL, ИСПОЛЬЗУЮЩЕЙ      *
* МЕТОД ПЕРЕДАЧИ ПАРАМЕТРОВ GENERAL.                      *
*****
PROGRAM-ID.      A.

:
DATA DIVISION.

:
LINKAGE SECTION.
*****
* ОБЯВЛЕНИЕ ПАРАМЕТРОВ, ПЕРЕДАННЫХ ОПЕРАТОРОМ SQL CALL.   *
*****
01 V1  PIC S9(9) USAGE COMP.
01 V2  PIC X(9).

:
PROCEDURE DIVISION USING V1, V2.
*****
* USING УКАЗЫВАЕТ, ЧТО ПЕРЕМЕННЫЕ V1 И V2 БЫЛИ           *
* ПЕРЕДАНЫ ВЫЗЫВАЮЩЕЙ ПРОГРАММОЙ.                         *
*****
:

* ПРИСВАИВАНИЕ ЗНАЧЕНИЯ ВЫХОДНОЙ ПЕРЕМЕННОЙ V2 *
*****
MOVE '123456789' TO V2.

```

---

*Рисунок 153. Пример использования метода передачи параметров GENERAL в программе на языке COBOL*

---

```

*PROCESS SYSTEM(MVS);
A: PROC(V1, V2) OPTIONS(MAIN NOEXECOPS REENTRANT);
/* Код хранимой процедуры на языке PL/I, использующей метод      */
/* передачи параметров GENERAL.                                     */
/*****
/* В операторе PROCEDURE указывается, что процедура имеет два   */
/* параметра, передаваемых оператором SQL CALL. Затем ниже    */
/* заданы объявления этих параметров.                            */
/*****
DCL V1 BIN FIXED(31),
V2 CHAR(9);

:
V2 = '123456789';      /* Присваивание значения выходной переменной V2 */

```

---

*Рисунок 154. Пример использования метода передачи параметров GENERAL в программе на языке PL/I*

## **Примеры использования метода передачи параметров GENERAL WITH NULLS для хранимой процедуры**

В следующих примерах показано, как использовать метод передачи параметров GENERAL WITH NULLS для входных параметров в хранимых процедурах, написанных на языках ассемблер, С, COBOL и PL/I. Примеры полного кода хранимых процедур и вызывающих их прикладных программ смотрите в разделе “Примеры использования хранимых процедур” на стр. 932.

В этих примерах предполагается, что прикладная программа на языке С содержит следующие объявления параметров и оператор CALL:

```
/********************************************/
/* Параметры для оператора SQL CALL          */
/********************************************/
long int v1;                                /* Дополнительный байт для      */
                                             /* нулевого символа-ограничителя */
/********************************************/
/* Структура индикаторов                      */
/********************************************/
short int procnm_ind;
short int ind1;
short int ind2;
} indstruc;

:
indstruc.ind1 = 0;                          /* Не забудьте инициализировать   */
                                             /* индикаторы для входных параметров */
                                             /* перед выполнением               */
                                             /* оператора CALL                  */
                                             /*
EXEC SQL CALL B (:v1 :indstruc.ind1, :v2 :indstruc.ind2);

:
```

В операторе CREATE PROCEDURE параметры определяются следующим образом:

V1 INT IN, V2 CHAR(9) OUT

На рис. 155, рис. 156, рис. 157 и рис. 158 показано, как хранимые процедуры, написанные на разных языках, получают такие параметры.

---

```

*****
* КОД ХРАНИМОЙ ПРОЦЕДУРЫ НА АССЕМБЛЕРЕ, ИСПОЛЬЗУЮЩЕЙ МЕТОД      *
* ПЕРЕДАЧИ ПАРАМЕТРОВ GENERAL WITH NULLS.                           *
*****
B      CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
      USING PROGAREA,R13
*****
* ЗАДАНИЕ ЯЗЫКОВОЙ СРЕДЫ.                                         *
*****
:
*****
* ПОЛУЧЕНИЕ ПЕРЕДАННЫХ ЗНАЧЕНИЙ ПАРАМЕТРОВ. МЕТОД ПЕРЕДАЧИ      *
* ПАРАМЕТРОВ GENERAL WITH NULLS:                                     *
* ПАРАМЕТРОВ DB2SQL:                                                 *
* НА ВХОДЕ В ПРОЦЕДУРУ РЕГИСТР 1 УКАЗЫВАЕТ НА СПИСОК УКАЗАТЕЛЕЙ.   *
* ЕСЛИ ПЕРЕДАНО N ПАРАМЕТРОВ, ИСПОЛЬЗУЕТСЯ N+1 УКАЗАТЕЛЬ. ПЕРВЫЕ   *
* ПЕРВЫЕ N УКАЗАТЕЛЕЙ - ЭТО АДРЕСА N ПАРАМЕТРОВ, ТАК ЖЕ КАК В       *
* ПЕРЕДАЧИ ПАРАМЕТРОВ GENERAL. (N+1)-Й УКАЗАТЕЛЬ - ЭТО АДРЕС      *
* СПИСКА, СОДЕРЖАЩЕГО N ИНДИКАТОРОВ ПУСТЫХ ЗНАЧЕНИЙ.             *
*****
L      R7,0(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА V1
MVC    LOCV1(4),0(R7)    СКОПИРОВАТЬ V1 В ЛОКАЛЬНУЮ ПЕРЕМЕННУЮ
L      R7,8(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА МАССИВ ИНДИКАТОРОВ
MVC    LOCIND(2*2),0(R7) СКОПИРОВАТЬ ИХ ЗНАЧЕНИЯ В ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ
LH     R7,LOCIND         ПОЛУЧИТЬ ЗНАЧЕНИЕ ИНДИКАТОРА ДЛЯ V1
LTR    R7,R7              ПРОВЕРИТЬ ЗНАК ЭТОГО ИНДИКАТОРА
BM     NULLIN            ЕСЛИ ОН ОТРИЦАТЕЛЬНЫЙ, V1 - ПУСТОЕ ЗНАЧЕНИЕ
:
L      R7,4(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА V2
MVC    0(9,R7),LOCV2    СКОПИРОВАТЬ ЗНАЧЕНИЕ В ВЫХОДНОЙ ПАРАМЕТР V2
L      R7,8(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА МАССИВ ИНДИКАТОРОВ
MVC    2(2,R7),=H(0)    ПРИСВОИТЬ ИНДИКАТОРУ ДЛЯ V2 ЗНАЧЕНИЕ 0
:
CEETERM RC=0
*****
* ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ И ЭКВИВАЛЕНТНЫХ ИМЕН                  *
*****
R1     EQU   1           РЕГИСТР 1
R7     EQU   7           РЕГИСТР 7
PPA    CEEPPA ,          КОНСТАНТЫ, ОПИСЫВАЮЩИЕ ЭТУТ БЛОК КОДА
          LTORG ,
PROGAREA DSECT          ПОМЕСТИТЕ ЗДЕСЬ ПУЛ ЛИТЕРАЛОВ
          ORG   **CEEDSASZ  ОСТАВИТЬ МЕСТО ДЛЯ ПОСТОЯННОЙ ЧАСТИ DSA
LOCV1   DS    F           ЛОКАЛЬНАЯ КОПИЯ ПАРАМЕТРА V1
LOCV2   DS    CL9          ЛОКАЛЬНАЯ КОПИЯ ПАРАМЕТРА V2
LOCIND  DS    2H           ЛОКАЛЬНАЯ КОПИЯ МАССИВА ИНДИКАТОРОВ
:
PROGSIZE EQU   *-PROGAREA
          CEEDSA ,          ОТОБРАЖЕНИЕ ОБЛАСТИ ДИНАМИЧЕСКОГО ХРАНЕНИЯ
          CEECAA ,          ОТОБРАЖЕНИЕ ОБЛАСТИ ОБЩЕЙ ПРИВЯЗКИ
          END   B

```

---

Рисунок 155. Пример использования метода передачи параметров *GENERAL WITH NULLS* в программе на ассемблере

---

```

#pragma options(RENT)
#pragma runopts(PLIST(OS))
#include <stdlib.h>
#include <stdio.h>
/*********************************************************/
/* Код хранимой процедуры на языке C, использующей метод */
/* передачи параметров GENERAL WITH NULLS. */
/*********************************************************/
main(argc,argv)
    int argc;           /* Число переданных параметров */
    char *argv[];       /* Массив строк, содержащих      */
                        /* значения параметров        */
{
    long int locv1;          /* Локальная копия V1          */
    char locv2[10];          /* Локальная копия V2          */
                            /* (с нулевым символом-ограничителем) */
    short int locind[2];     /* Локальная копия массива    */
                            /* индикаторов                */
    short int *tempint;      /* Используется для получения */
                            /* массива индикаторов        */
                            /* */

    :
    /* **** */
    /* Получение переданных параметров. Метод передачи параметров */
    /* GENERAL WITH NULLS: */
    /* DB2SQL: */
    /* - argc содержит число переданных параметров */
    /* - argv[0] - указатель на имя хранимой процедуры */
    /* - argv[1] до argv[n] - указатели на n параметров */
    /* в операторе SQL CALL. */
    /* - argv[n+1] - указатель на массив индикаторов пустых */
    /* значений */
    /* **** */
    if(argc==4)
    {
        /* Нужно получить 4 параметра: */
        /* procname, V1, V2, */
        /* массива индикаторов */
        locv1 = *(int *) argv[1];
        tempint = argv[3];
        locind[0] = *tempint;
        locind[1] = *(++tempint);
        if(locind[0]<0)
        {
            /* Получить локальную копию V1 */
            /* Получить указатель на */
            /* индикаторов */
            /* Получить 1-й индикатор */
            /* Получить 2-й индикатор */
            /* Если 1-й индикатор меньше 0,*/
            /* V1 - пустое значение */
            :
        }
        :
        strcpy(argv[2],locv2);
        *(++tempint) = 0;
        /* Присвоить значение V2 */
        /* Присвоить 0 индикатору */
        /* для параметра V2 */
    }
}

```

---

*Рисунок 156. Пример использования метода передачи параметров GENERAL WITH NULLS в программе на языке C*

---

```

CBL RENT
IDENTIFICATION DIVISION.
*****
* КОД ХРАНИМОЙ ПРОЦЕДУРЫ НА ЯЗЫКЕ COBOL, ИСПОЛЬЗУЮЩЕЙ      *
* МЕТОД ПЕРЕДАЧИ ПАРАМЕТРОВ GENERAL WITH NULLS.          *
*****
PROGRAM-ID.      B.

:
DATA DIVISION.

:
LINKAGE SECTION.
*****
* ОБЪЯВЛЕНИЕ ПАРАМЕТРОВ И МАССИВА ИНДИКАТОРОВ, ПЕРЕДАННЫХ   *
* ОПЕРАТОРОМ SQL CALL.                                         *
*****
01 V1  PIC S9(9) USAGE COMP.
01 V2  PIC X(9).
*
01 INDARRAY.
10 INDVAR  PIC S9(4) USAGE COMP OCCURS 2 TIMES.

:
PROCEDURE DIVISION USING V1, V2, INDARRAY.
*****
* ФРАЗА USING УКАЗЫВАЕТ, ЧТО ПЕРЕМЕННЫЕ V1, V2 И МАССИВ      *
* ИНДИКАТОРОВ БЫЛИ ПЕРЕДАНЫ ВЫЗЫВАЮЩЕЙ ПРОГРАММОЙ.        *
*****
:
* ПРОВЕРКА V1 НА НУЛЕВОЕ ЗНАЧЕНИЕ    *
*****
IF INDARRAY(1) < 0
PERFORM NULL-PROCESSING.

:
*****
* ПРИСВАИВАНИЕ ЗНАЧЕНИЯ ВЫХОДНОЙ ПЕРЕМЕННОЙ V2   *
* И ЕЕ ИНДИКАТОРУ НУЛЕВОГО ЗНАЧЕНИЯ                 *
*****
MOVE '123456789' TO V2.
MOVE ZERO TO INDARRAY(2).

```

*Рисунок 157. Пример использования метода передачи параметров GENERAL WITH NULLS в программе на языке COBOL*

---

```
*PROCESS SYSTEM(MVS);
A: PROC(V1, V2, INDSTRUC) OPTIONS(MAIN NOEXECOPS REENTRANT);
/*****************************************/
/* Код хранимой процедуры на языке PL/I, использующей метод      */
/* передачи параметров GENERAL WITH NULLS.                           */
/*****************************************/
/*****************************************/
/* В операторе PROCEDURE указывается, что процедура имеет два   */
/* параметра и структуру индикаторов пустых значений,          */
/* передаваемые оператором SQL CALL. Затем ниже заданы        */
/* объявления этих параметров.                                     */
/* В языке PL/I необходимо объявлять структуру индикаторов,    */
/* а не массив.                                                 */
/*****************************************/
DCL V1 BIN FIXED(31),
      V2 CHAR(9);
DCL
  01 INDSTRUC,
  02 IND1 BIN FIXED(15),
  02 IND2 BIN FIXED(15);

:
IF IND1 < 0 THEN
  CALL NULLVAL;      /* Если индикатор имеет отрицательное */
                     /* значение, то V1 - пустое значение */

:
V2 = '123456789';    /* Присваивание значения выходной переменной V2 */
IND2 = 0;             /* Присваивание индикатору для V2 значения 0 */

:
```

*Рисунок 158. Пример использования метода передачи параметров GENERAL WITH NULLS в программе на языке PL/I*

---

### **Примеры использования метода передачи параметров DB2SQL для хранимой процедуры**

В следующих примерах показано, как использовать метод передачи параметров DB2SQL для входных параметров в хранимых процедурах, написанных на языках ассемблер, С, COBOL и PL/I. В этих примерах показано также, как хранимая процедура получает структуру DBINFO.

В этих примерах предполагается, что прикладная программа на языке С содержит следующие объявления параметров и оператор CALL:

```

/*****************/
/* Параметры для оператора SQL CALL */
/*****************/
long int v1;                                /* Дополнительный байт для      */
                                              /* нулевого символа-ограничителя */
/*****************/
/* Переменные для индикаторов пустых значений */
/*****************/
short int ind1;                             /* Не забудьте инициализировать */
                                              /* индикаторы для входных параметров */
                                              /* перед выполнением */
                                              /* оператора CALL */
EXEC SQL CALL B (:v1 :indstruc.ind1, :v2 :ind1, :ind2);

:

```

В операторе CREATE PROCEDURE параметры определяются так:

V1 INT IN, V2 CHAR(9) OUT

На рис. 159, рис. 160, рис. 161, рис. 162 и рис. 163 показано, как хранимые процедуры, написанные на разных языках, получают такие параметры.

---

```

*****
* КОД ХРАНИМОЙ ПРОЦЕДУРЫ НА АССЕМБЛЕРЕ, ИСПОЛЬЗУЮЩЕЙ МЕТОД      *
* ПЕРЕДАЧИ ПАРАМЕТРОВ DB2SQL.                                         *
*****
B      CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
      USING PROGAREA,R13
*****
* ЗАДАНИЕ ЯЗЫКОВОЙ СРЕДЫ.                                         *
*****
      ::

*****
* ПОЛУЧЕНИЕ ПЕРЕДАННЫХ ЗНАЧЕНИЙ ПАРАМЕТРОВ. МЕТОД ПЕРЕДАЧИ      *
* ПАРАМЕТРОВ DB2SQL:                                                 *
* НА ВХОДЕ В ПРОЦЕДУРУ РЕГИСТР 1 УКАЗЫВАЕТ НА СПИСОК УКАЗАТЕЛЕЙ.   *
* ЕСЛИ ПЕРЕДАНО N ПАРАМЕТРОВ, ИСПОЛЬЗУЕТСЯ 2N+4 УКАЗАТЕЛЬ.       *
* ПЕРВЫЕ N УКАЗАТЕЛЕЙ - ЭТО АДРЕСА N ПАРАМЕТРОВ, ТАК ЖЕ КАК В      *
* МЕТОДЕ ПЕРЕДАЧИ ПАРАМЕТРОВ GENERAL. СЛЕДУЮЩИЕ N УКАЗАТЕЛЕЙ -   *
* АДРЕСА ЗНАЧЕНИЙ ПЕРЕМЕННЫХ-ИНДИКАТОРОВ. ПОСЛЕДНИЕ             *
* 4 УКАЗАТЕЛЯ (5, ЕСЛИ ПЕРЕДАЕТСЯ DBINFO) - АДРЕСНАЯ             *
* ИНФОРМАЦИЯ О СРЕДЕ ХРАНИМОЙ ПРОЦЕДУРЫ И РЕЗУЛЬТАТАХ           *
* ВЫПОЛНЕНИЯ.                                                       *
*****
L      R7,0(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА V1
MVC    LOCV1(4),0(R7)    СКОПИРОВАТЬ V1 В ЛОКАЛЬНУЮ ПЕРЕМЕННУЮ
L      R7,8(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА 1-Й ИНДИКАТОР
MVC    LOCI1(2),0(R7)    СКОПИРОВАТЬ ЕГО ЗНАЧЕНИЕ В ЛОКАЛЬНУЮ ПЕРЕМЕННУЮ
L      R7,20(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА ИМЯ ХРАНИМОЙ ПРОЦЕДУРЫ
MVC    LOCSPNM(20),0(R7) СКОПИРОВАТЬ ЭТО ЗНАЧЕНИЕ В ЛОКАЛЬНУЮ ПЕРЕМЕННУЮ
L      R7,24(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА DBINFO
MVC    LOCDBINF(DBINFLN),0(R7)
*          КОПИРУЕМ ЗНАЧЕНИЕ В ЛОКАЛЬНУЮ ПЕРЕМЕННУЮ
LH     R7,LOCI1          ПОЛУЧИТЬ ЗНАЧЕНИЕ ИНДИКАТОРА ДЛЯ V1
LTR    R7,R7              ПРОВЕРИТЬ ЗНАК ЭТОГО ИНДИКАТОРА
BM    NULLIN             ЕСЛИ ОН ОТРИЦАТЕЛЬНЫЙ, V1 - ПУСТОЕ ЗНАЧЕНИЕ
      ::

L      R7,4(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА V2
MVC    0(9,R7),LOCV2    СКОПИРОВАТЬ ЗНАЧЕНИЕ В ВЫХОДНОЙ ПАРАМЕТР V2
L      R7,12(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА ИНДИКАТОРОВ ДЛЯ V2
MVC    0(2,R7),=H'0'    ПРИСВОИТЬ ИНДИКАТОРУ ДЛЯ V2 ЗНАЧЕНИЕ 0
L      R7,16(R1)          ПОЛУЧИТЬ УКАЗАТЕЛЬ НА SQLSTATE
MVC    0(5,R7),=CL5'xxxxx' ЗАНЕСТИ xxxx в SQLSTATE
      ::

      CEETERM RC=0

```

---

*Рисунок 159 (Часть 1 из 2). Пример использования метода передачи параметров DB2SQL в программе на ассемблере*

---

```

*****
* ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ И ЭКВИВАЛЕНТНЫХ ИМЕН *
*****
R1      EQU  1          РЕГИСТР 1
R7      EQU  7          РЕГИСТР 7
PPA     CEEPPA ,        КОНСТАНТЫ, ОПИСЫВАЮЩИЕ ЭТОТ БЛОК КОДА
          LTORG ,
PROGAREA DSECT
          ORG  **+CEEDSASZ   ОСТАВИТЬ МЕСТО ДЛЯ ПОСТОЯННОЙ ЧАСТИ DSA
LOCV1   DS   F          ЛОКАЛЬНАЯ КОПИЯ ПАРАМЕТРА V1
LOCV2   DS   CL9         ЛОКАЛЬНАЯ КОПИЯ ПАРАМЕТРА V2
LOCI1   DS   H          ЛОКАЛЬНАЯ КОПИЯ ИНДИКАТОРА 1
LOCI2   DS   H          ЛОКАЛЬНАЯ КОПИЯ ИНДИКАТОРА 2
LOCSQLST DS  CL5        ЛОКАЛЬНАЯ КОПИЯ SQLSTATE
LOCSPNM  DS  H,CL27    ЛОКАЛЬНАЯ КОПИЯ ИМЕНИ ХРАНИМОЙ ПРОЦЕДУРЫ
LOCSPSNM DS  H,CL18    ЛОКАЛЬНАЯ КОПИЯ УНИКАЛЬНОГО ИМЕНИ
LOCDIAG  DS  H,CL70    ЛОКАЛЬНАЯ КОПИЯ ДАННЫХ ДИАГНОСТИКИ
LOCDBINF DS  0H         ЛОКАЛЬНАЯ КОПИЯ ДАННЫХ DBINFO
DBNAMELN DS  H          ДЛИНА ИМЕНИ БАЗЫ ДАННЫХ
DBNAME   DS  CL128     ИМЯ БАЗЫ ДАННЫХ
AUTHIDLN DS  H          ДЛИНА ID АВТОРИЗАЦИИ ПРОГРАММЫ
AUTHID   DS  CL128     ID АВТОРИЗАЦИИ ПРОГРАММЫ
EBC_SBCS DS  F          EBCDIC SBCS CCSID
EBC_DBCS DS  F          EBCDIC DBCS CCSID
EBC_MIXD DS  F          EBCDIC MIXED CCSID
ASC_SBCS DS  F          ASCII SBCS CCSID
ASC_DBCS DS  F          ASCII DBCS CCSID
ASC_MIXD DS  F          ASCII MIXED CCSID
ENCODE   DS  F          СХЕМА КОДИРОВАНИЯ ПРОЦЕДУРЫ
RESERVO  DS  CL20       РЕЗЕРВ
TBQUALLN DS  H          ДЛИНА СПЕЦИФИКАТОРА ТАБЛИЦЫ
TBQUAL   DS  CL128     СПЕЦИФИКАТОР ТАБЛИЦЫ
TBNAMELN DS  H          ДЛИНА ИМЕНИ ТАБЛИЦЫ
TBNAME   DS  CL128     ИМЯ ТАБЛИЦЫ
CLNAMELN DS  H          ДЛИНА ИМЕНИ СТОЛБЦА
COLNAME   DS  CL128     ИМЯ СТОЛБЦА
RELVER   DS  CL8         ВЫПУСК И ВЕРСИЯ DBMS
PLATFORM  DS  F          ОПЕРАЦИОННАЯ СИСТЕМА DBMS
NUMTFCOL DS  H          ЧИСЛО ИСПОЛЬЗУЕМЫХ СТОЛБЦОВ ТАБЛИЧНОЙ ФУНКЦИИ
RESERV1  DS  CL24       РЕЗЕРВ
TFCOLNUM DS  A          УКАЗАТЕЛЬ НА СПИСОК СТОЛБЦОВ ТАБЛИЧНОЙ ФУНКЦИИ
APPLID   DS  A          УКАЗАТЕЛЬ НА ID ПРОГРАММЫ
RESERV2  DS  CL20       РЕЗЕРВ
DBINFLN  EQU  *-LOCDBINF  ДЛИНА DBINFO

:
PROGSIZE EQU  *-PROGAREA
          CEEDSA ,
          CEECAA ,
          END   B          ОТОБРАЖЕНИЕ ОБЛАСТИ ДИНАМИЧЕСКОГО ХРАНЕНИЯ
                           ОТОБРАЖЕНИЕ ОБЛАСТИ ОБЩЕЙ ПРИВЯЗКИ

```

---

*Рисунок 159 (Часть 2 из 2). Пример использования метода передачи параметров DB2SQL в программе на ассемблере*

---

```
#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>

main(argc,argv)
    int argc;
    char *argv[];
{
    int parml;
    short int ind1;
    char p_proc[28];
    char p_spec[19];
    /***** *****/
    /* Предположим, что оператор SQL CALL содержит в */
    /* списке параметров 3 входных/выходных параметра. */
    /* Вектор argv будет содержать следующие записи: */
    /*      argv[0]      1  загрузочный модуль      */
    /*      argv[1-3]     3  входных/выходных параметра */
    /*      argv[4-6]     3  индикатора пустых значений */
    /*      argv[7]       1  переменная SQLSTATE      */
    /*      argv[8]       1  полное имя процедуры      */
    /*      argv[9]       1  собственное имя процедуры */
    /*      argv[10]      1  сообщение об ошибке      */
    /*      argv[11]      + 1 dbinfo                  */
    /*      -----          */
    /*      12   для переменной argc      */
    /***** *****/
    if argc<>12 {

    ::

    /* Закончить здесь, если выполнен вызов с неверным числом параметров */
}
```

---

*Рисунок 160 (Часть 1 из 2). Пример использования метода передачи параметров DB2SQL для хранимой процедуры на языке C, написанной как главная программа*

---

```
*****
/* Предположим, что тип первого параметра - целое. */
/* Ниже показано, как скопировать этот параметр */
/* в память программы. */
*****
parm1 = *(int *) argv[1];
*****
/* К индикатору пустого значения для первого */
/* параметра SQL CALL можно обратиться так: */
*****
ind1 = *(short int *) argv[4];
*****
/* Следующее выражение приписывает значение */
/* 'xxxxx' SQLSTATE, возвращаемому вызывающей */
/* программе оператора SQL CALL. */
*****
strcpy(argv[7],"xxxxx/0");
*****
/* Получаем полное имя функции */
/* с помощью следующего выражения. */
*****
strcpy(p_proc,argv[8]);
*****
/* Получаем собственное имя функции */
/* с помощью следующего выражения. */
*****
strcpy(p_spec,argv[9]);
*****
/* Следующее выражение приписывает значение */
/* 'ууууууу' строке диагностики, возвращаемой */
/* в SQLDA, связанной с оператором CALL. */
*****
strcpy(argv[10],"ууууууу/0");
:
}
```

---

Рисунок 160 (Часть 2 из 2). Пример использования метода передачи параметров DB2SQL для хранимой процедуры на языке C, написанной как главная программа

---

```
#pragma linkage(myproc,fetchable)
#include <stdlib.h>
#include <stdio.h>
struct sqlsp_dbinfo
{
    unsigned short dbnamelen; /* длина имени базы данных */
    unsigned char dbname[128]; /* имя базы данных */
    unsigned short authidlen; /* длина id авт. программы */
    unsigned char authid[128]; /* ID авт. программы */
    unsigned long ebdic_sbccs; /* EBCDIC SBCS CCSID */
    unsigned long ebdic_dbccs; /* EBCDIC MIXED CCSID */
    unsigned long ebdic_mixed; /* EBCDIC DBCS CCSID */
    unsigned long ascii_sbccs; /* ASCII SBCS CCSID */
    unsigned long ascii_dbccs; /* ASCII MIXED CCSID */
    unsigned long ascii_mixed; /* ASCII DBCS CCSID */
    unsigned long encode; /* схема кодировки UDF */
    unsigned char reserv0[20]; /* резерв для последующего использования*/
    unsigned short tbqualiflen; /* длина спецификатора таблицы */
    unsigned char tbqualif[128]; /* имя спецификатора таблицы */
    unsigned short tbnamelen; /* длина имени таблицы */
    unsigned char tbname[128]; /* имя таблицы */
    unsigned short colnamelen; /* длина имени столбца */
    unsigned char colname[128]; /* имя столбца */
    unsigned char relver[8]; /* Версия и выпуск базы данных */
    unsigned long platform; /* Платформа базы данных */
    unsigned short numtfcol; /* число столбцов табличной функции */
    unsigned char reserv1[24]; /* зарезервировано */
    unsigned short *tfcolnum; /* список столбцов табличной функции */
    unsigned short *appl_id; /* LUWID для соединения с DB2 */
    unsigned char reserv2[20]; /* зарезервировано */
};
```

---

*Рисунок 161 (Часть 1 из 2). Пример использования метода передачи параметров DB2SQL для хранимой процедуры на языке C, написанной как подпрограмма*

---

```
void myproc(*parm1 int,           /* пусть PARM1 типа INT          */
            parm2 char[11],      /* а PARM2 - CHAR(10)          */
            :
            *p_ind1 short int,   /* индикатор пустого значения для parm1 */
            *p_ind2 short int,   /* индикатор пустого значения для parm2 */
            :
            p_sqlstate char[6],   /* SQLSTATE, возвращаемый DB2          */
            p_proc char[28],      /* Полное имя хранимой процедуры    */
            p_spec char[19],      /* Собственное имя хранимой процедуры */
            p_diag char[71],      /* Страна диагностики                */
            struct sqlsp_dbinfo *sp_dbinfo);  /* DBINFO                           */
{
    int l_p1;
    char[11] l_p2;
    short int l_ind1;
    short int l_ind2;
    char[6] l_sqlstate;
    char[28] l_proc;
    char[19] l_spec;
    char[71] l_diag;
    sqlsp_dbinfo *lsp_dbinfo;
    :
    /*****
     * Каждый параметр из списка параметров          */
     * копируется в локальную переменную, чтобы        */
     * показать обращение к этим параметрам.          */
     ****/
    l_p1 = *parm1;

    strcpy(l_p2,parm2);

    l_ind1 = *p_ind1;

    l_ind1 = *p_ind2;

    strcpy(l_sqlstate,p_sqlstate);

    strcpy(l_proc,p_proc);

    strcpy(l_spec,p_spec);

    strcpy(l_diag,p_diag);
    memcpy(&lsp_dbinfo,sp_dbinfo,sizeof(lsp_dbinfo));
    :
}
```

---

Рисунок 161 (Часть 2 из 2). Пример использования метода передачи параметров DB2SQL для хранимой процедуры на языке C, написанной как подпрограмма

---

```
CBL RENT
    IDENTIFICATION DIVISION.
    :
    DATA DIVISION.
    :
    LINKAGE SECTION.
* Объявляем входные параметры
01 PARM1 ...
01 PARM2 ...
:
* Объявляем индикатор пустого значения для каждого входного параметра
01 P-IND1 PIC S9(4) USAGE COMP.
01 P-IND2 PIC S9(4) USAGE COMP.
:
* Объявляем SQLSTATE, который задает хранимая процедура
01 P-SQLSTATE PIC X(5).
* Объявляем полное имя функции
01 P-PROC.
    49 P-PROC-LEN PIC 9(4) USAGE BINARY.
    49 P-PROC-TEXT PIC X(27).
* Объявляем собственное имя функции
01 P-SPEC.
    49 P-SPEC-LEN PIC 9(4) USAGE BINARY.
    49 P-SPEC-TEXT PIC X(18).
* Объявляем маркер диагностического сообщения SQL
01 P-DIAG.
    49 P-DIAG-LEN PIC 9(4) USAGE BINARY.
    49 P-DIAG-TEXT PIC X(70).
*****
* объявление структуры DBINFO
*****
01 SP-DBINFO.
* Имя положения и его длина
    02 UDF-DBINFO-LOCATION.
        49 UDF-DBINFO-LLEN PIC 9(4) USAGE BINARY.
        49 UDF-DBINFO-LOC PIC X(128).
* ID авторизации и его длина
    02 UDF-DBINFO-AUTHORIZATION.
        49 UDF-DBINFO-ALEN PIC 9(4) USAGE BINARY.
        49 UDF-DBINFO-AUTH PIC X(128).
* CCSID для DB2 for OS/390
    02 UDF-DBINFO-CCSID PIC X(48).
    02 UDF-DBINFO-CCSID-REDEFINE UDF-DBINFO-CCSID.
        03 UDF-DBINFO-ESBCS PIC 9(9) USAGE BINARY.
        03 UDF-DBINFO-EMIXED PIC 9(9) USAGE BINARY.
        03 UDF-DBINFO-EDBCS PIC 9(9) USAGE BINARY.
        03 UDF-DBINFO-ASBCS PIC 9(9) USAGE BINARY.
        03 UDF-DBINFO-AMIXED PIC 9(9) USAGE BINARY.
        03 UDF-DBINFO-ADBCS PIC 9(9) USAGE BINARY.
        03 UDF-DBINFO-ENCODE PIC 9(9) USAGE BINARY.
        03 UDF-DBINFO-RESERV0 PIC X(20).
```

---

Рисунок 162 (Часть 1 из 2). Пример использования метода передачи параметров DB2SQL в программе на языке COBOL

---

```
*      Имя схемы и его длина
02 UDF-DBINFO-SCHEMA0.
49 UDF-DBINFO-SLEN PIC 9(4) USAGE BINARY.
49 UDF-DBINFO-SCHEMA PIC X(128).
*
*      Имя таблицы и его длина
02 UDF-DBINFO-TABLE0.
49 UDF-DBINFO-TLEN PIC 9(4) USAGE BINARY.
49 UDF-DBINFO-TABLE PIC X(128).
*
*      Имя столбца и его длина
02 UDF-DBINFO-column0.
49 UDF-DBINFO-CLEN PIC 9(4) USAGE BINARY.
49 UDF-DBINFO-COLUMN PIC X(128).
*
*      Уровень выпуска DB2
02 UDF-DBINFO-VERREL PIC X(8).
*      Не используется
02 FILLER          PIC X(2).
*
*      Платформа базы данных
02 UDF-DBINFO-PLATFORM PIC 9(9) USAGE BINARY.
*
*      число элементов в списке столбцов табличной функции
02 UDF-DBINFO-NUMTFCOL PIC 9(4) USAGE BINARY.
*
*      зарезервировано
02 UDF-DBINFO-RESERV1 PIC X(24).
*
*      Не используется
02 FILLER          PIC X(2).
*
*      Указатель на список столбцов табличной функции
02 UDF-DBINFO-TFCOLUMN PIC 9(9) USAGE BINARY.
*
*      Указатель на ID программы
02 UDF-DBINFO-APPLID PIC 9(9) USAGE BINARY.
*
*      зарезервировано
02 UDF-DBINFO-RESERV2 PIC X(20).

*:
:
PROCEDURE DIVISION USING PARM1, PARM2,
P-IND1, P-IND2,
P-SQLSTATE, P-PROC, P-SPEC, P-DIAG,
SP-DBINFO.
:
:
```

Рисунок 162 (Часть 2 из 2). Пример использования метода передачи параметров DB2SQL в программе на языке COBOL

---

```

*PROCESS SYSTEM(MVS);
MYMAIN: PROC(PARM1, PARM2, ...,
              P_IND1, P_IND2, ...,
              P_SQLSTATE, P_PROC, P_SPEC, P_DIAG, SP_DBINFO)
              OPTIONS(MAIN NOEXECOPS REENTRANT);

      DCL PARM1 ...          /* первый параметр */
      DCL PARM2 ...          /* второй параметр */
      :
      DCL P_IND1 BIN FIXED(15);/* индикатор для первого параметра */
      DCL P_IND2 BIN FIXED(15);/* индикатор для второго параметра */
      :
      DCL P_SQLSTATE CHAR(5); /* SQLSTATE, возвращаемый DB2 */
      DCL 01 P-PROC CHAR(27) /* Полное имя процедуры */
                  VARYING;
      DCL 01 P-SPEC CHAR(18) /* Собственное имя процедуры */
                  VARYING;
      DCL 01 P-DIAG CHAR(70) /* Страна диагностики */
                  VARYING;

      DCL DBINFO PTR;
      DCL 01 SP_DBINFO BASED(DBINFO),           /* Dbinfo */
            03 UDF_DBINFO_LLEN BIN FIXED(15),    /* длина положения */
            03 UDF_DBINFO_LOC CHAR(128),         /* имя положения */
            03 UDF_DBINFO_ALEN BIN FIXED(15),    /* длина ID авт. */
            03 UDF_DBINFO_AUTH CHAR(128),        /* ID авторизации */
            03 UDF_DBINFO_CCSID,                /* CCSIDs for DB2 for OS/390*/
                  05 R1      BIN FIXED(15), /* Зарезервировано */
                  05 UDF_DBINFO_ESBCS BIN FIXED(15), /* EBCDIC SBCS CCSID */
                  05 R2      BIN FIXED(15), /* Зарезервировано */
                  05 UDF_DBINFO_EMIXED BIN FIXED(15), /* EBCDIC MIXED CCSID*/
                  05 R3      BIN FIXED(15), /* Зарезервировано */
                  05 UDF_DBINFO_EDBCS BIN FIXED(15), /* EBCDIC DBCS CCSID */
                  05 R4      BIN FIXED(15), /* Зарезервировано */
                  05 UDF_DBINFO_ASBCS BIN FIXED(15), /* ASCII SBCS CCSID */
                  05 R5      BIN FIXED(15), /* Зарезервировано */
                  05 UDF_DBINFO_AMIXED BIN FIXED(15), /* ASCII MIXED CCSID */
                  05 R6      BIN FIXED(15), /* Зарезервировано */
                  05 UDF_DBINFO_ADBCS BIN FIXED(15), /* ASCII DBCS CCSID */
                  05 UDF_DBINFO_ENCODE BIN FIXED(31), /* схема кодировки UDF */
                  05 UDF_DBINFO_RESERVO CHAR(20),   /* зарезервировано */
            03 UDF_DBINFO_SLEN BIN FIXED(15),   /* длина схемы */
            03 UDF_DBINFO_SCHEMA CHAR(128),     /* имя схемы */
            03 UDF_DBINFO_TLEN BIN FIXED(15),   /* длина таблицы */
            03 UDF_DBINFO_TABLE CHAR(128),      /* имя таблицы */
            03 UDF_DBINFO_CLEN BIN FIXED(15),   /* длина столбца */
            03 UDF_DBINFO_COLUMN CHAR(128),     /* имя столбца */
            03 UDF_DBINFO_RELVER CHAR(8),       /* версия и выпуск DB2 */
            03 UDF_DBINFO_PLATFORM BIN FIXED(31), /* платформа базы данных */
            03 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* число столб. табл. функц. */
            03 UDF_DBINFO_RESERV1 CHAR(24),      /* зарезервировано */
            03 UDF_DBINFO_TFCOLUMN PTR,         /* -> список столб. табл. функц. */
            03 UDF_DBINFO_APPLID PTR,          /* -> id программы */
            03 UDF_DBINFO_RESERV2 CHAR(20);    /* Зарезервировано */
      :

```

---

*Рисунок 163. Пример использования метода передачи параметров DB2SQL в программе на языке PL/I*

## Особенности использования языка C

Чтобы методы передачи параметров правильно работали в хранимой процедуре на языке C, выполняемой в системе MVS, необходимо включить в код программы строку

```
#pragma runopts(PLIST(OS))
```

Однако эта опция не может применяться на других платформах. Если предполагается использование этой хранимой процедуры, написанной на языке C, не только в системе MVS, но и на других plataформах, следует использовать условную компиляцию, как показано на рис. 164, чтобы эта опция включалась только при компиляции в системе MVS.

```
#ifdef MVS  
#pragma runopts(PLIST(OS))  
#endif
```

-- или --

```
#ifndef WKSTN  
#pragma runopts(PLIST(OS))  
#endif
```

Рисунок 164. Использование условной компиляции для включения или исключения оператора

## Особенности использования языка PL/I

Чтобы методы передачи параметров правильно работали в хранимой процедуре на языке PL/I, выполняемой в системе MVS, необходимо сделать следующее:

- Включить в исходный код опцию времени выполнения NOEXECOPS.
- Задать опцию времени компиляции SYSTEM(MVS).

Информацию о задании опций времени компиляции и опций времени выполнения для языка PL/I смотрите в руководстве *IBM PL/I MVS & VM Programming Guide*.

## Использование переменных–индикаторов для ускорения обработки

Если какой–либо из выходных параметров занимает очень большой объем памяти, нет смысла передавать хранимой процедуре всю эту область памяти. В программе,зывающей хранимую процедуру, можно использовать индикатор пустого значения, чтобы передавать хранимой процедуре только двухбайтную область, а получать от хранимой процедуры всю область памяти. Чтобы сделать это, объявите индикатор пустого значения для каждого большого выходного параметра в операторе SQL CALL. (Если используется метод передачи параметров GENERAL WITH NULLS или DB2SQL, необходимо объявить индикаторы пустых значений для всех параметров, тем самым нет необходимости объявлять другие индикаторы пустых значений.) Присвойте отрицательное значение каждому индикатору пустого значения, связанному с большой выходной переменной. Затем включите эти индикаторы пустых значений в оператор CALL. Эта техника может использоваться для всех методов передачи параметров (GENERAL, GENERAL WITH NULLS или DB2SQL).

Например, предположим, что у хранимой процедуры, определенной с методом передачи параметров GENERAL, есть один входной параметр типа целое число и один выходной параметр длиной 6000. Вы не хотите передавать хранимой процедуре область памяти длиной 6000. Программа на языке PL/I, содержащая эти операторы, передает хранимой процедуре для выходной переменной только два байта, а получает от хранимой процедуры все 6000 байт:

```
DCL INTVAR BIN FIXED(31);          /* Это входная переменная */
DCL BIGVAR(6000);                  /* Это выходная переменная */
DCL I1 BIN FIXED(15);              /* Это индикатор пустого значения */
:
I1 = -1;                           /* Присваивание I1 значения -1 */
/* в результате чего хранимой */
/* процедуре только 2-байтная */
/* область, описываемая */
/* индикатором I1, вместо 6000- */
/* байтной области для BIGVAR */
EXEC SQL CALL PROCX(:INTVAR, :BIGVAR INDICATOR :I1);
```

## Объявление типов данных для передаваемых параметров

В хранимой процедуре должны быть объявлены все передаваемые ей параметры. Кроме того, определение хранимой процедуры должно содержать для каждого из параметров объявление совместимых с SQL типа данных. Информацию о создании определения хранимой процедурысмотрите в разделе “Определение хранимой процедуры для DB2” на стр. 585.

Для всех типов данных, кроме больших объектов, ROWID и локаторов, смотрите таблицы в разделе Табл. 57, где указана совместимость типов данных хоста с типами данных в определении хранимой процедуры. Для больших объектов, ROWID и локаторов смотрите таблицы Табл. 58, Табл. 59 на стр. 633, Табл. 60 на стр. 634 и Табл. 61 на стр. 635.

*Таблица 57. Список таблиц совместимости типов данных*

Язык	Таблица совместимых типов данных
Ассемблер	Табл. 9 на стр. 155
C	Табл. 11 на стр. 172
COBOL	Табл. 14 на стр. 197
PL/I	Табл. 18 на стр. 226

*Таблица 58 (Стр. 1 из 2). Совместимые объявления на ассемблере для больших объектов, ROWID и локаторов*

Тип данных SQL в определении	объявление на ассемблере
TABLE LOCATOR	DS FL4
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	

Таблица 58 (Стр. 2 из 2). Совместимые объявления на ассемблере для больших объектов, ROWID и локаторов

Тип данных SQL в определении	объявление на ассемблере
BLOB( <i>n</i> )	Если <i>n</i> <= 65535: var DS 0FL4 var_length DS FL4 var_data DS CLn Если <i>n</i> > 65535: var DS 0FL4 var_length DS FL4 var_data DS CL65535 ORG var_data+( <i>n</i> -65535)
CLOB( <i>n</i> )	Если <i>n</i> <= 65535: var DS 0FL4 var_length DS FL4 var_data DS CLn Если <i>n</i> > 65535: var DS 0FL4 var_length DS FL4 var_data DS CL65535 ORG var_data+( <i>n</i> -65535)
DBCLOB( <i>n</i> )	Если <i>m</i> (=2* <i>n</i> ) <= 65534: var DS 0FL4 var_length DS FL4 var_data DS CLm Если <i>m</i> > 65534: var DS 0FL4 var_length DS FL4 var_data DS CL65534 ORG var_data+( <i>m</i> -65534)
ROWID	DS HL2,CL40

Таблица 59 (Стр. 1 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке С

Тип данных SQL в определении	Объявление на языке С
TABLE LOCATOR	unsigned long
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	
BLOB( <i>n</i> )	struct {unsigned long length; char data[n];} } var;
CLOB( <i>n</i> )	struct {unsigned long length; char var_data[n];} } var;
DBCLOB( <i>n</i> )	struct {unsigned long length; wchar_t data[n];} } var;

*Таблица 59 (Стр. 2 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке C*

<b>Тип данных SQL в определении</b>	<b>Объявление на языке C</b>
ROWID	<pre>struct {     short int length;     char data[40]; } var;</pre>

*Таблица 60 (Стр. 1 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке COBOL*

<b>Тип данных SQL в определении</b>	<b>Объявление на языке COBOL</b>
TABLE LOCATOR	01 var PIC S9(9) USAGE IS BINARY.
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	
BLOB( <i>n</i> )	<p>Если <i>n</i> &lt;= 32767:</p> <pre>01 var.     49 var-LENGTH PIC 9(9)         USAGE COMP.     49 var-DATA PIC X(<i>n</i>).</pre> <p>Если <i>n</i> &gt; 32767:</p> <pre>01 var.     02 var-LENGTH PIC S9(9)         USAGE COMP.     02 var-DATA.         49 FILLER             PIC X(32767).         49 FILLER             PIC X(32767).     :     49 FILLER         PIC X(mod(<i>n</i>,32767)).</pre>
CLOB( <i>n</i> )	<p>Если <i>n</i> &lt;= 32767:</p> <pre>01 var.     49 var-LENGTH PIC 9(9)         USAGE COMP.     49 var-DATA PIC X(<i>n</i>).</pre> <p>Если <i>n</i> &gt; 32767:</p> <pre>01 var.     02 var-LENGTH PIC S9(9)         USAGE COMP.     02 var-DATA.         49 FILLER             PIC X(32767).         49 FILLER             PIC X(32767).     :     49 FILLER         PIC X(mod(<i>n</i>,32767)).</pre>

*Таблица 60 (Стр. 2 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке COBOL*

<b>Тип данных SQL в определении</b>	<b>Объявление на языке COBOL</b>
DBCLOB( <i>n</i> )	<p>Если <i>n</i> &lt;= 32767:</p> <p>01 var.          49 var-LENGTH PIC 9(9)          USAGE COMP.          49 var-DATA PIC G(<i>n</i>)          USAGE DISPLAY-1.</p> <p>Если <i>n</i> &gt; 32767:</p> <p>01 var.          02 var-LENGTH PIC S9(9)          USAGE COMP.          02 var-DATA.          49 FILLER          PIC G(32767)          USAGE DISPLAY-1.          49 FILLER          PIC G(32767).          USAGE DISPLAY-1.</p> <p>:</p> <p>49 FILLER          PIC G(mod(<i>n</i>,32767))          USAGE DISPLAY-1.</p>
ROWID	<p>01 var.          49 var-LEN PIC 9(4)          USAGE COMP.          49 var-DATA PIC X(40).</p>

*Таблица 61 (Стр. 1 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке PL/I*

<b>Тип данных SQL в определении</b>	<b>PL/I</b>
TABLE LOCATOR	BIN FIXED(31)
BLOB LOCATOR	
CLOB LOCATOR	
DBCLOB LOCATOR	
BLOB( <i>n</i> )	<p>Если <i>n</i> &lt;= 32767:</p> <p>01 var,          03 var_LENGTH          BIN FIXED(31),          03 var_DATA          CHAR(<i>n</i>);</p> <p>Если <i>n</i> &gt; 32767:</p> <p>01 var,          02 var_LENGTH          BIN FIXED(31),          02 var_DATA,          03 var_DATA1(<i>n</i>)          CHAR(32767),          03 var_DATA2          CHAR(mod(<i>n</i>,32767));</p>

*Таблица 61 (Стр. 2 из 2). Совместимые объявления для больших объектов, ROWID и локаторов на языке PL/I*

Тип данных SQL в определении	PL/I
CLOB( <i>n</i> )	<pre> Если n &lt;= 32767: 01 var,  03 var_LENGTH     BIN FIXED(31),  03 var_DATA     CHAR(<i>n</i>); Если n &gt; 32767: 01 var,  02 var_LENGTH     BIN FIXED(31),  02 var_DATA,  03 var_DATA1(<i>n</i>)     CHAR(32767),  03 var_DATA2     CHAR(mod(<i>n</i>,32767)); </pre>
DBCLOB( <i>n</i> )	<pre> If n &lt;= 16383: 01 var,  03 var_LENGTH     BIN FIXED(31),  03 var_DATA     GRAPHIC(<i>n</i>); Если n &gt; 16383: 01 var,  02 var_LENGTH     BIN FIXED(31),  02 var_DATA,  03 var_DATA1(<i>n</i>)     GRAPHIC(16383),  03 var_DATA2     GRAPHIC(mod(<i>n</i>,16383)); </pre>
ROWID	CHAR(40) VAR

**Таблицы результатов:** Каждое определение языка высокого уровня для параметров хранимых процедур поддерживает только одно значение (скалярное значение) параметра. Не поддерживаются параметры типа структура, массив или вектор. По этой причине оператор SQL CALL ограничивает возможности прикладной программы в возвращении некоторых типов таблиц. Например, в прикладной программе может потребоваться возвращать таблицу, которая содержит по несколько значений для одного или нескольких параметров, передаваемых хранимой процедуре. Поскольку оператор SQL CALL не может возвращать более одного набора параметров, для возвращения такой таблицы используйте одну из следующих техник:

- Поместите возвращаемые прикладной программой данные в таблицу DB2. Вызывающая программа может получить эти данные одним из следующих способов:
  - Вызывающая программа может напрямую выбрать строки из этой таблицы. Задайте FOR FETCH ONLY или FOR READ ONLY в операторе SELECT, используемемся для получения данных из этой таблицы. Выборка блоком позволяет эффективно получать требуемые данные.

- Хранимая процедура может возвращать содержимое этой таблицы в виде набора результатов. Дополнительную информацию смотрите в разделах “Написание хранимой процедуры, возвращающей наборы результатов клиенту DRDA” на стр. 603 и “Написание клиентской программы DB2 for OS/390, получающей наборы результатов” на стр. 637.
- Преобразуйте табличные данные в формат строки и верните их вызывающей программе в виде параметра типа символьная строка. Вызывающая программа и хранимая процедура могут использовать соглашение о формате содержимого такой символьной строки. Например, оператор SQL CALL может передать хранимой процедуре в качестве параметра символьную строку длиной 1920 байт, позволяя хранимой процедуре вернуть вызывающей программе образ экрана 24x80.

## **Написание клиентской программы DB2 for OS/390, получающей наборы результатов**

Если прикладная программа вызывает хранимую процедуру, возвращающую наборы результатов, необходимо включить в прикладную программу код, который:

- Определяет число возвращенных наборов результатов
- Определяет содержание каждого набора результатов
- Получает все наборы результатов

Если известны число и содержание наборов результатов, возвращаемых хранимой процедурой, программу можно сделать проще. Однако если код написан для более общего случая, в котором число и содержание наборов результатов неизвестны, при изменении хранимой процедуры не придется вносить серьезных изменений в клиентскую программу.

Для получения наборов результатов выполните семь основных действий:

1. Объявите локатор для каждого возвращаемого набора результатов.

Если неизвестно сколько будет возвращаться наборов результатов, объягите достаточное число локаторов наборов результатов для максимального числа наборов результатов, которые могут быть возвращены.

2. Вызовите хранимую процедуру и проверьте код возврата SQL.

Если SQLCODE из оператора CALL равен +466, хранимая процедура вернула наборы результатов.

3. Определите число возвращенных хранимой процедурой наборов результатов.

Если число возвращаемых хранимой процедурой наборов результатов уже известно, этот шаг можно пропустить.

Для определения числа наборов результатов используйте оператор SQL DESCRIBE PROCEDURE. Этот оператор помещает информацию о наборах результатов в SQLDA. Сделайте эту SQLDA достаточно большой для сохранения информации о максимальном числе наборов результатов, которые может вернуть хранимая процедура. После выполнения оператора DESCRIBE PROCEDURE поля в SQLDA содержат следующие значения:

- SQLD содержит число наборов результатов, возвращаемых хранимой процедурой.
- Каждый элемент SQLVAR содержит информацию об одном наборе результатов. В элементе SQLVAR:
  - Поле SQLNAME содержит имя указателя SQL, использованного хранимой процедурой для возвращения этого набора результатов.
  - Поле SQLIND содержит значение –1. Этого означает, что отсутствует оценка числа строк в этом наборе результатов.
  - Поле SQLDATA содержит значение локатора набора результатов, являющееся адресом этого набора результатов.

#### 4. Свяжите локаторы наборов результатов с наборами результатов.

Для связывания локаторов наборов результатов с наборами результатов можно использовать оператор SQL ASSOCIATE LOCATORS. Этот оператор присваивает значения переменным локаторов наборов результатов. Если число заданных локаторов больше, чем число возвращенных наборов результатов, DB2 игнорирует избыточные локаторы.

Чтобы использовать оператор ASSOCIATE LOCATORS, его необходимо встроить в прикладную программу. Оператор ASSOCIATE LOCATORS нельзя выполнять динамически.

Если перед этим был выполнен оператор DESCRIBE PROCEDURE, поля SQLDATA в SQLDA будут содержать значения локатора набора результатов. Значения полей SQLDATA можно вручную скопировать в локаторы наборов результатов или же для этого можно использовать оператор ASSOCIATE LOCATORS.

Имя хранимой процедуры, задаваемое в операторе ASSOCIATE LOCATORS или DESCRIBE PROCEDURE, должно совпадать с именем хранимой процедуры в операторе CALL, который возвращает эти наборы результатов. Это значит, что:

- Если в ASSOCIATE LOCATORS или DESCRIBE PROCEDURE задано простое имя хранимой процедуры (имя без квалификаторов), имя хранимой процедуры в операторе CALL также должно быть простым именем.
- Если в ASSOCIATE LOCATORS или DESCRIBE PROCEDURE задано имя хранимой процедуры с именем схемы, имя хранимой процедуры в операторе CALL также должно задаваться с именем схемы.
- Если в ASSOCIATE LOCATORS или DESCRIBE PROCEDURE задано имя хранимой процедуры с именем системы и именем схемы, имя хранимой процедуры в операторе CALL также должно задаваться с именем системы и именем схемы.

#### 5. Выделите указатели для выборки строк из наборов результатов.

Чтобы связать каждый набор результатов с указателем, используйте оператор SQL ALLOCATE CURSOR. Выполните оператор ALLOCATE CURSOR для каждого набора результатов. Эти имена указателей могут отличаться имен указателей в хранимой процедуре.

Чтобы использовать оператор ALLOCATE CURSOR, его необходимо встроить в прикладную программу. Оператор ALLOCATE CURSOR нельзя выполнять динамически.

6. Определите содержание наборов результатов.

Если формат набора результатов уже известен, этот шаг можно пропустить.

Используйте оператор SQL DESCRIBE CURSOR, чтобы определить формат набора результатов и поместить эту информацию в SQLDA. Для каждого набора результатов необходимо использовать достаточно большую SQLDA, чтобы она могла вместить описания всех столбцов в наборе результатов.

Оператор DESCRIBE CURSOR можно использовать только для тех указателей, для которых перед этим был выполнен оператор ALLOCATE CURSOR.

Если указатель для набора результатов объявлен с опцией WITH HOLD, после выполнения оператора DESCRIBE CURSOR старший бит восьмого байта поля SQLDAID в SQLDA будет установлен в 1.

7. Выберите строки из набора результатов в переменные хоста, используя указатели, выделенные операторами ALLOCATE CURSOR.

Если оператор DESCRIBE CURSOR выполнен, перед выборкой строк выполните следующие шаги:

- a. Выделите память для переменных хоста и индикаторов пустых значений. Чтобы определить объем памяти, требуемый для каждой переменной хоста, используйте содержимое SQLDA, полученное в результате выполнения оператора DESCRIBE CURSOR.
- b. Поместите адрес памяти для каждой переменной хоста в соответствующее поле SQLDATA в SQLDA.
- c. Поместите адрес памяти для каждого индикатора пустого значения в соответствующее поле SQLIND в SQLDA.

Выборка строк из набора результатов аналогична выборке строк из таблицы.

При выполнении этих операторов нет необходимости в соединении с удаленной системой:

- DESCRIBE PROCEDURE
- ASSOCIATE LOCATORS
- ALLOCATE CURSOR
- DESCRIBE CURSOR
- FETCH
- CLOSE

Синтаксис локаторов наборов результатов для каждого языка смотрите в разделе “Глава 3–4. Встроенные операторы SQL в языках хоста” на стр. 145. Синтаксис операторов ASSOCIATE LOCATORS, DESCRIBE PROCEDURE, ALLOCATE CURSOR и DESCRIBE CURSOR смотрите в главе 6 руководства *DB2 SQL Reference*.

На рис. 165 на стр. 640 и рис. 166 на стр. 641 показаны программы на языке C, выполняющие каждый из этих шагов. Для других языков программы пишутся аналогичным образом. Более полный пример

программы на языке С, получающей набор результатов, смотрите в разделе “Примеры использования хранимых процедур” на стр. 932.

На рис. 165 показано, как получать наборы результатов в случае, когда известны число возвращаемых наборов результатов и содержание каждого набора результатов.

```
*****
/* Объявление локаторов наборов результатов. В этом этом */
/* примере предполагается, что известно, что возвращаются */
/* два набора результатов. Также предполагается, что */
/* известен формат обоих наборов результатов */
*****
EXEC SQL BEGIN DECLARE SECTION;
    static volatile SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2;
EXEC SQL END DECLARE SECTION;

:
*****
/* Вызов хранимой процедуры P1. */
/* Проверка, что SQLCODE имеет значение +466, означающее, */
/* что были возвращены наборы результатов. */
*****
EXEC SQL CALL P1(:parm1, :parm2, ...);
if(SQLCODE==+466)
{
    ****
    /* Задание при помощи оператора ASSOCIATE LOCATORS связей */
    /* между каждым набором результатов и его локатором. */
    ****
    EXEC SQL ASSOCIATE LOCATORS (:loc1, :loc2) WITH PROCEDURE P1;
:

:
    ****
    /* Связывание указателя с каждым набором результатов. */
    ****
    EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
    EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :loc2;
    ****
    /* Выборка строк наборов результатов в переменные хоста. */
    ****
    while(SQLCODE==0)
    {
        EXEC SQL FETCH C1 INTO :order_no, :cust_no;
        :
        }
        while(SQLCODE==0)
        {
            EXEC SQL FETCH C2 :order_no, :item_no, :quantity;
        :
        }
    }
}
*****
```

Рисунок 165. Получение неизвестных наборов результатов

На рис. 166 на стр. 641 показано, как получать наборы результатов в случае, когда число возвращаемых наборов результатов и содержание каждого набора результатов неизвестны.

---

```

/*****************/
/* Объявление локаторов наборов результатов. В этом этом      */
/* примере предполагается, что известно, что возвращается      */
/* не более трех наборов результатов, поэтому объявляется      */
/* три локатора. Также предполагается, что формат наборов      */
/* результатов неизвестен.                                         */
/*****************/
EXEC SQL BEGIN DECLARE SECTION;
    static volatile SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2, *loc3;
EXEC SQL END DECLARE SECTION;

::

/*****************/
/* Вызов хранимой процедуры P2.                                     */
/* Проверка, что SQLCODE имеет значение +466, означающее,      */
/* что были возвращены наборы результатов.                         */
/*****************/
EXEC SQL CALL P2(:parm1, :parm2, ...);
if(SQLCODE==+466)
{
/*****************/
/* Определение числа возвращенных процедурой P2 наборов      */
/* результатов, для чего используется оператор                  */
/* DESCRIBE PROCEDURE.                                           */
/* :proc_da - SQLDA достаточного размера для                   */
/* вмещения трех записей SQLVAR.                                */
/*****************/
EXEC SQL DESCRIBE PROCEDURE P2 INTO :proc_da;

::

/*****************/
/* Теперь, когда известно число возвращенных результатов,      */
/* задаются связи между каждым набором результатов и его      */
/* локатором, для чего используется оператор                  */
/* ASSOCIATE LOCATORS. В этом примере предполагается, что      */
/* возвращено три набора результатов.                           */
/*****************/
EXEC SQL ASSOCIATE LOCATORS (:loc1, :loc2, :loc3) WITH PROCEDURE P2;

::

/*****************/
/* Связывание указателя с каждым набором результатов.      */
/*****************/
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :loc2;
EXEC SQL ALLOCATE C3 CURSOR FOR RESULT SET :loc3;

```

---

*Рисунок 166 (Часть 1 из 2). Получение неизвестных наборов результатов*

```
*****
/* Используется оператор DESCRIBE CURSOR для определения      */
/* формата каждого набора результатов.                          */
*****
EXEC SQL DESCRIBE CURSOR C1 INTO :res_da1;
EXEC SQL DESCRIBE CURSOR C2 INTO :res_da2;
EXEC SQL DESCRIBE CURSOR C3 INTO :res_da3;

:::

*****  

/* Присваивание значений полям SQLDATA и SQLIND в SQLDA,      */
/* которые использовались в операторах DESCRIBE CURSOR.      */
/* Эти значения – адреса переменных хоста и индикаторов      */
/* пустых значений, в которые DB2 будет помещать строки      */
/* наборов результатов.                                         */
*****  

:::  

*****  

/* Выборка строк наборов результатов в области памяти, на    */
/* которые указывают SQLDA.                                       */
*****  

while(SQLCODE==0)
{
    EXEC SQL FETCH C1 USING :res_da1;  

:::  

}  

    while(SQLCODE==0)
{
    EXEC SQL FETCH C2 USING :res_da2;  

:::  

}  

    while(SQLCODE==0)
{
    EXEC SQL FETCH C3 USING :res_da3;  

:::  

}
```

Рисунок 166 (Часть 2 из 2). Получение неизвестных наборов результатов

## Подготовка клиентской программы

Вызывающую программу необходимо подготовить, проведя на системе клиента ее препроцессинг, компиляцию и компоновку.

Перед тем как можно будет вызывать хранимую процедуру из встроенных операторов SQL прикладной программы, необходимо связать пакет для этой программы клиента на удаленной системе. Для связывания пакета с удаленной системой можно использовать возможности удаленного связывания DRDA на системе клиента DRDA.

Если имеются пакеты, содержащие операторы SQL CALL и связанные до установки DB2 версии 6, можно улучшить производительность выполнения этих пакетов, заново связав их в DB2 версии 6 или более поздней. Такое повторное связывание позволяет DB2 получить во время связывания

некоторую информацию из каталога, которая в версиях до версии 6 получалась во время выполнения. Следовательно, после повторного связывания пакетов эти пакеты выполняются более эффективно, поскольку уменьшается число операций поиска в каталоге, которые DB2 производит во время выполнения.

Перед тем как можно будет выполнять прикладную программу ODBC или CLI, необходимо связать с DB2 пакеты и план DB2, связанные с драйвером ODBC. Информацию о построении прикладных программ клиента, обращающихся к хранимым процедурам, на платформах, отличающихся от DB2 for OS/390,смотрите в следующих документах:

- *DB2 UDB Application Building Guide*
- *DB2 for OS/400® SQL Programming*

Клиент MVS может связать DBRM с удаленным сервером, задав имя системы в команде BIND PACKAGE. Например, предположим, программа клиента должна вызывать хранимую процедуру в системе LOCA. Проведите прокомпиляцию программы, чтобы создать DBRM A. Затем можно использовать команду

BIND PACKAGE (LOCA.COLLA) MEMBER(A)

для связывания DBRM A с собранием пакетов COLLA в системе LOCA.

План для пакета располагается только на системе клиента.

---

## Выполнение хранимой процедуры

Хранимая процедура выполняется или в виде основной программы, или в виде подпрограммы. В разделе “Написание хранимой процедуры в виде основной программы или подпрограммы” на стр. 595 представлена информация о требованиях для каждого из этих типов хранимых процедур.

Если хранимая процедура выполняется в виде *основной программы*, языковая среда перед каждым вызовом этой хранимой процедуры заново инициализирует память, используемую хранимой процедурой. Программные переменные для этой хранимой процедуры не сохраняются между вызовами.

Если хранимая процедура выполняется в виде *подпрограммы*, языковая среда не инициализирует память между вызовами. Программные переменные для этой хранимой процедуры могут сохраняться между вызовами. Однако не следует предполагать, что значения программных переменных из одного вызова будут доступны в другом вызове, так как:

- Хранимые процедуры, вызванные другими пользователями, могут выполняться в данном экземпляре языковой среды между двумя выполнениями вашей хранимой процедуры.
- Последовательные выполнения хранимой процедуры могут выполняться в различных адресных пространствах хранимых процедур.
- Оператор MVS может обновить языковую среду между двумя выполнениями вашей хранимой процедуры.

DB2 выполняет хранимые процедуры в потоке DB2 вызывающей программы, делая хранимые процедуры частью рабочей единицы вызывающей программы.

Если обе среды прикладных программ на клиенте и на сервере поддерживают двухфазное принятие, координатор управляет обновлениями между прикладной программой, сервером и хранимыми процедурами. Если одна из сторон не поддерживает двухфазное принятие, обновления будут неудачными.

Если хранимая процедура завершается ненормально:

- Вызывающая программа получает сообщение об ошибке SQL, уведомляющее об ошибке выполнения хранимой процедуры.
- DB2 переводит рабочую единицу вызывающей программы в состояние необходимости отката.
- Если хранимая процедура не обрабатывает условие ненормального завершения, DB2 обновляет языковую среду, чтобы восстановить используемую программой память. В большинстве случаев нет необходимости в перезапуске языковой среды.
- Если в процедуре JCL, запускающей адресное пространство хранимых процедур, выделен набор данных для имени DD CEECDUMP, языковая среда записывает в этот набор данных небольшой диагностический дамп. Чтобы получить эту информацию дампа, обратитесь к системному администратору. Описание способов диагностики ошибок смотрите в разделе “Тестирование хранимой процедуры” на стр. 649.

## Как DB2 определяет, какую версию хранимой процедуры нужно выполнять

Хранимая процедура однозначно определяется комбинацией имени схемы и имени хранимой процедуры. Если при выполнении оператора CALL для вызова хранимой процедуры задано полное имя хранимой процедуры с квалификаторами, оно определяет единственную хранимую процедуру. Однако если задано имя хранимой процедуры без квалификаторов, DB2 использует для определения хранимой процедуры следующий метод:

1. DB2 просматривает слева направо список имен схем в значении опции связывания PATH или в специальном регистре CURRENT PATH, пока не найдет имя схемы, для которой существует определение хранимой процедуры с именем, заданным в операторе CALL. DB2 использует имена схем из опции связывания PATH для операторов CALL вида  
*CALL литерал*  
Для операторов CALL вида  
*CALL переменная-хоста*  
DB2 использует имена схем из специального регистра CURRENT PATH.
2. Если DB2 находит определение хранимой процедуры, DB2 выполняет эту хранимую процедуру, если выполнены следующие условия:
  - Вызывающая программа обладает полномочиями на выполнение этой хранимой процедуры.

- Число параметров этой хранимой процедуры совпадает с числом параметров, заданных в операторе CALL.

Если хотя бы одно из этих условий не выполняется, DB2 продолжает просмотр списка схем, пока не найдет хранимую процедуру, удовлетворяющую обоим условиям, или не достигнет конца списка.

3. Если DB2 не может найти подходящую хранимую процедуру, DB2 возвращает для оператора CALL код ошибки SQL.

## Вызов разных версий хранимой процедуры из одной программы

Если вы хотите вызывать из одной прикладной программы различные версии хранимой процедуры с совпадающими именами загрузочного модуля, выполните следующие действия:

1. При определении каждой версии хранимой процедуры используйте одно и то же имя процедуры, но разные имена схем и среды WLM.
2. В программе, которая вызывает хранимую процедуру, задайте в операторе CALL неспецифицированное имя процедуры.
3. При помощи пути SQL укажите, какую из версий процедуры должна вызывать клиентская программа. Выбрать путь SQL можно несколькими способами:
  - Если клиентская программа не использует ODBC или JDBC, примените один из следующих методов:
    - Используйте оператор CALL в виде *CALL имя-процедуры*. Когда вы связываете планы или пакеты для программы, которая вызывает хранимую процедуру, свяжите по плану или пакету для каждой версии хранимой процедуры, которую вы собираетесь вызывать. В опции связывания PATH для каждого плана или пакета укажите имя схемы для хранимой процедуры, которую вы собираетесь вызывать.
    - Используйте оператор CALL в виде *CALL переменная-хоста*. В клиентской программе используйте оператор SET CURRENT PATH для задания имени схемы хранимой процедуры, которую вы собираетесь вызывать.
  - Если клиентская программа использует ODBC или JDBC, примените один из следующих методов:
    - Используйте оператор SET CURRENT PATH для задания имени схемы хранимой процедуры, которую вы собираетесь вызывать.
    - При связывании пакетов хранимых процедур укажите различные собрания для каждого пакета хранимых процедур. В клиентской программе используйте оператор SET CURRENT PACKAGESET для задания собрания пакетов с хранимой процедурой, которую вы собираетесь вызывать.
4. При запуске хранимой процедуры укажите план или пакет со значением PATH, которое соответствует имени схемы для хранимой процедуры, которую вы собираетесь вызывать.

Предположим, например, что вы хотите написать одну программу PROGY, которая вызывает одну из двух версий хранимой процедуры PROCX. Загрузочный модуль для каждой из версий хранимой процедуры называется

SUMMOD. Каждая из версий SUMMOD находится в своей загрузочной библиотеке. Эти хранимые процедуры выполняются в разных средах WLM, и JCL запуска для каждой из сред WLM содержит STEPLIB, который указывает правильную загрузочную библиотеку для модуля хранимой процедуры.

Сначала определим две хранимые процедуры в разных схемах и в разных средах WLM:

```
CREATE PROCEDURE TEST.PROCX(V1 INTEGER IN, CHAR(9) OUT)
LANGUAGE C
EXTERNAL NAME SUMMOD
WLM ENVIRONMENT TESTENV;
```

```
CREATE PROCEDURE PROD.PROCX(V1 INTEGER IN, CHAR(9) OUT)
LANGUAGE C
EXTERNAL NAME SUMMOD
WLM ENVIRONMENT PRODENV;
```

В операторах CALL для процедуры PROCX в программе PROGY используйте неспецифицированную форму имени хранимой процедуры:

```
CALL PROCX(V1,V2);
```

Свяжите два плана для PROGY. В одном операторе BIND укажите PATH(TEST). В другом операторе BIND укажите PATH(PROD).

Чтобы вызвать TEST.PROCX, выполните PROGY с планом, который связывался с PATH(TEST). Чтобы вызвать PROD.PROCX, выполните PROGY с планом, который связывался с PATH(PROD).

## Одновременное выполнение нескольких хранимых процедур

Можно выполнять нескольких хранимых процедур одновременно, каждую процедуру под ее собственной задачей MVS (TCB). Максимальное число хранимых процедур, которые могут одновременно выполняться в одном адресном пространстве, задается во время установки DB2 на панели DSNTIPX.

Дополнительную информацию смотрите в Разделе 2 руководства *DB2 Installation Guide*.

Это значение можно переопределить следующими способами:

- Для адресных пространств хранимых процедур, созданных WLM или DB2:
  - Задайте параметр NUMTCB в команде MVS START, запускающей адресные пространства хранимых процедур.
  - Измените значение параметра NUMTCB в тексте процедур JCL, запускающих адресные пространства хранимых процедур.
- Для адресных пространств, созданных WLM, задайте при настройке среды прикладных программ WLM параметр

`NUMTCB=число-TCB`

в поле Start Parameters (Параметры запуска) панели Create An Application Environment (Создать среду прикладных программ).

Чтобы максимально увеличить число хранимых процедур, которые могут выполняться одновременно, используйте следующие основные правила:

- Задайте значение 0 для параметра REGION в процедурах запуска для адресных пространств хранимых процедур, чтобы получить максимально возможный объем памяти в области ниже 16 Мбайт.
- Ограничьте объем памяти, требуемый прикладным программам в области ниже 16 Мбайт:
  - Скомпонуйте программы для работы в области выше 16 Мбайт, задав атрибуты AMODE(31) и RMODE(ANY)
  - Используйте опции компилятора RES и DATA(31) для программ на языке COBOL.
- Ограничьте объем памяти, требуемый для языковой среды IBM, используя следующие опции времени выполнения:
  - HEAP(,,ANY) чтобы выделить динамическую память в области выше 16 Мбайт
  - STACK(,,ANY,) чтобы выделить программный стек в области выше 16 Мбайт
  - STORAGE(,,4K) чтобы уменьшить до 4 Кбайт размер зарезервированной области памяти в области ниже 16 Мбайт
  - BELOWHEAP(4K,,) чтобы уменьшить до 4 Кбайт размер динамической области памяти в области ниже 16 Мбайт
  - LIBSTACK(4K,,) чтобы уменьшить до 4 Кбайт размер стека библиотек в области ниже 16 Мбайт
  - ALL31(ON) чтобы указать, что все программы, входящие в хранимую процедуру, должны выполняться с атрибутами AMODE(31) и RMODE(ANY).

Эти опции можно задать в списке значений параметра RUN OPTIONS оператора CREATE PROCEDURE или ALTER PROCEDURE, если они не являются заданными при установке значениями по умолчанию для языковой среды. Например, для параметре RUN OPTIONS можно задать опции:

```
H(,,ANY),STAC(,,ANY,),STO(,,,4K),BE(4K,,),LIBS(4K,,),ALL31(ON)
```

Дополнительную информацию о создании определения хранимой процедуры смотрите в разделе “Определение хранимой процедуры для DB2” на стр. 585.

- Если для хранимых процедур используются адресные пространства, созданные WLM, для одной среды прикладных программ WLM назначайте хранимые процедуры со сходным поведением. Если хранимые процедуры в среде WLM имеют существенно различные характеристики скорости выполнения, WLM может не суметь правильно определить характеристики рабочей загрузки WLM. В итоге WLM может создать слишком мало или слишком много адресных пространств. В обоих случаях могут возрасти время выполнения хранимых процедур и прикладных программ DB2.

Дополнительную информацию о назначении хранимых процедур для WLM сред прикладных программ смотрите в Разделе 5 (Том 2) руководства *DB2 Administration Guide*.

## Обращение к другим ресурсам (не DB2)

Прикладные программы, которые выполняются в адресном пространстве хранимых процедур, могут обращаться к любым ресурсам, доступным в адресных пространствах MVS, таким как файлы VSAM, плоские файлы, диалогам MVS/APPC и транзакциям IMS или CICS.

При разработке хранимых процедур, обращающихся к другим ресурсам (не DB2), имейте в виду следующее:

- Если хранимая процедура выполняется в адресном пространстве хранимых процедур, созданном DB2, DB2 не согласовывает действия по принятию и откату для восстановимых ресурсов, таких как транзакции IMS или CICS, и сообщениями MQI. DB2 не имеет информации об отношениях между хранимой процедурой и восстановимым ресурсом и, следовательно, не может ими управлять.
- Если хранимая процедура выполняется в адресном пространстве хранимых процедур, созданном WLM, для управления принятиями хранимая процедура использует менеджер транзакций и службу управления восстанавливаемыми ресурсами OS/390 (OS/390 RRS). При выполнении в этих средах операций принятия и отката DB2 согласовывает все обновления восстановимых ресурсов, производимые в системе MVS другими менеджерами ресурсов, совместимых с OS/390 RRS.
- Если хранимая процедура выполняется в адресном пространстве хранимых процедур, созданном DB2, MVS не знает, что адресное пространство хранимых процедур выполняет работу для DB2. В частности, это значит, что MVS обращается к защищенным RACF ресурсам, используя ID пользователя, связанный с задачей MVS (*ssnmSPAS*) для хранимых процедур, а не ID пользователя для клиента.
- Если хранимая процедура выполняется в адресном пространстве хранимых процедур, созданном WLM, для доступа к ресурсам других систем (не DB2) DB2 может создать среду RACF. Полномочия, используемые при обращении хранимой процедуры к защищенным ресурсом MVS, зависят от значения EXTERNAL SECURITY в определении хранимой процедуры:
  - Если SECURITY имеет значение DB2, используются ID авторизации, связанный с адресным пространством хранимых процедур.
  - Если SECURITY имеет значение USER, используются ID авторизации, под которым выполняется оператор CALL.
  - Если SECURITY имеет значение DEFINER, используются ID авторизации, под которым выполнялся оператор CREATE PROCEDURE.
- Не все прочие (не DB2) ресурсы допускают одновременное обращение нескольких TCB в одном адресном пространстве. Может потребоваться сделать эти вызовы в прикладной программе последовательными.

## CICS

Прикладные программы хранимых процедур могут обращаться к CICS одним из следующих методов:

- Интерфейс очереди сообщений (MQI): для асинхронного выполнения транзакций CICS
- Внешний интерфейс CICS (EXCI): для синхронного выполнения транзакций CICS
- Протокол APPC с использованием интерфейса прикладного программирования CPI Communications

В адресных пространствах, созданных DB2, прикладная программа CICS выполняется в виде отдельной рабочей единицы, отличной от рабочей единицы, в которой выполняется хранимая процедура. Следовательно, результаты работы CICS не влияют на завершение работы хранимой процедуры. Например, транзакция CICS в хранимой процедуре, для которой производится откат рабочей единицы, не мешает хранимой процедуре выполнить принятие для рабочей единицы DB2. Аналогично откат рабочей единицы DB2 не отменяет успешного принятия транзакции CICS.

Если используются адресные пространства, созданные WLM, и система выполняется в выпуске CICS, использующем OS/390 RRS, OS/390 RRS управляет принятием всех ресурсов.

## IMS

Если система выполняется в выпуске IMS, не использующем OS/390 RRS, для доступа из хранимой процедуры к данным DL/I можно использовать один из следующих методов:

- Используйте интерфейс CICS EXCI для синхронного выполнения транзакций CICS. Такие транзакции CICS могут по очереди обращаться к данным DL/I.
- Запустите транзакции IMS асинхронно, используя MQI.
- Используйте APPC через интерфейс прикладного программирования CPI Communications

## Тестирование хранимой процедуры

Обычно используемые средства отладки, такие как TSO TEST, не доступны в среде, в которой выполняются хранимые процедуры. Ниже описаны некоторые альтернативные методы тестирования.

## Отладка хранимой процедуры как отдельной программы на рабочей станции

Если на рабочей станции есть средства отладки, можно провести большую часть работ по созданию и отладке на рабочей станции до установки хранимой процедуры в MVS. В результате в MVS остается выполнить очень немного отладочных операций.

## Отладка при помощи Debug Tool и IBM VisualAge® COBOL

Если на вашей рабочей станции установлен VisualAge COBOL, а на системе OS/390 – Debug Tool, можно использовать компонент Edit/Compile/Debug VisualAge COBOL совместно с Debug Tool для отладки хранимых процедур на языке COBOL, выполняемых в адресном пространстве, созданном WLM. Подробную информацию о Debug Toolсмотрите в книге *Debug Tool User's Guide and Reference*.

Написав хранимую процедуру на языке COBOL и задав среду WLM, выполните следующие действия, чтобы протестировать хранимую процедуру при помощи Debug Tool:

1. При компиляции хранимой процедуры укажите опции TEST и SOURCE.

Задайте сохранение листинга исходного текста в постоянном наборе данных. VisualAge COBOL выводит этот листинг исходного текста в сеансе отладки.

2. При определении хранимой процедуры укажите в аргументе RUN OPTIONS опцию времени выполнения TEST с подопцией VADTCPIP&*ipaddr*.

VADTCPIP& сообщает Debug Tool о взаимодействии с рабочей станцией, на которой выполняется VisualAge COBOL и которая сконфигурирована для связи TCP/IP с вашей системой OS/390. Здесь *ipaddr* – IP–адрес рабочей станции, на которую вы выводите отладочную информацию. Например, значение RUN OPTIONS в определении следующей хранимой процедуры указывает, что отладочную информацию надо направлять на рабочую станцию с IP–адресом 9.63.51.17:

```
CREATE PROCEDURE WLMCOB
  (IN INTEGER, INOUT VARCHAR(3000), INOUT INTEGER)
  MODIFIES SQL DATA
  LANGUAGE COBOL EXTERNAL
  PROGRAM TYPE MAIN
  WLM ENVIRONMENT WLMENV1
  RUN OPTIONS 'POSIX(ON),TEST(,,,VADTCPIP&9.63.51.17:*)'
```

3. В процедуре запуска JCL для адресного пространства хранимых процедур, созданного WLM, добавьте имя набора данных загрузочной библиотеки Debug Tool в список STEPLIB. Предположим, например, что ENV1PROC – процедура JCL для среды прикладных программ WLMENV1. Исправленная процедура JCL для ENV1PROC может выглядеть так:

```

//DSNWLM    PROC RGN=OK,APPLENv=WLMENV1,DB2SSN=DSN,NUMTCB=8
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//           PARM='&DB2SSN,&NUMTCB,&APPLENv'
//STEPLIB   DD  DISP=SHR,DSN=DSN610.RUNLIB.LOAD
//           DD  DISP=SHR,DSN=CEE.SCEERUN
//           DD  DISP=SHR,DSN=DSN610.SDSNLOAD
//           DD  DISP=SHR,DSN=EQAW.SEQAMOD <== DEBUG TOOL

```

4. На рабочей станции запустите демон VisualAge Remote Debugger.

Этот демон ожидает входящие требования от TCP/IP.

5. Вызовите хранимую процедуру.

Когда хранимая процедура будет запущена, на рабочей станции будет открыто окно сеанса отладки. Можно отлаживать вашу хранимую процедуру, выполняя команды Debug Tool.

## Отладка при помощи CODE/370

Для проверки хранимых процедур MVS, написанных на любом поддерживаемом языке, можно использовать лицензированную программу CoOperative Development Environment/370, которая работает совместно с языковая среда. Программу CODE/370 можно использовать в интерактивном или в пакетом режиме.

**Использование CODE/370 в интерактивном режиме:** Для интерактивного тестирования хранимой процедуры при помощи CODE/370, необходимо использовать на рабочей станции отладчик PWS программы CODE/370. Также необходимо, чтобы программа CODE/370 была установлена в системе MVS, в которой выполняется хранимая процедура. Для отладки хранимой процедуры при помощи отладчика PWS сделайте следующее:

- Скомпилируйте хранимую процедуру с опцией TEST. В программу будет помещена информация, используемая отладчиком во время сеанса отладки.
- Запустите отладчик. Один из способов сделать это – задать для языковой среды опцию времени выполнения TEST. Опция TEST определяет, когда и как запускается отладчик. Самое удобное место для задания опций времени выполнения – параметр RUN OPTIONS оператора CREATE PROCEDURE или ALTER PROCEDURE для хранимой процедуры.

Например, если задать опцию:

TEST(ALL,\*,PROMPT,JBJONES%SESSNA:)

значения параметров задают, что:

**ALL**

Отладчик получает управление в случае прерывания внимания, условия аварийной остановки (ABEND) или при возникновении в программе или в языковой среде условия с уровнем серьезности 1 или выше.

Команды отладки будут вводиться с терминала.

**PROMPT**

Отладчик отладки запускается сразу после инициализации языковой среды.

**JBJONES%SESSNA:** CODE/370 запускает на рабочей станции сеанс, определяемый для APPC/MVS как JBJONES с ID сеанса SESSNA.

- Если нужно сохранять выходную информацию сеанса отладки, введите команду, задающую имя файла журнала. Например,

```
SET LOG ON FILE dbgtool.log;
```

вызывает запись информации в расположенный на этой рабочей станции файл журнала с именем dbgtool.log. Это должна быть первая команда, вводимая с терминала или включенная в командный файл.

**Использование CODE/370 в пакетном режиме:** Для тестирования хранимой процедуры в пакетном режиме необходимо, чтобы в системе MVS, в которой выполняется хранимая процедура, был установлен отладчик MFI CODE/370. Чтобы отладить хранимую процедуру в пакетном режиме, используя отладчик MFI, сделайте следующее:

- Если для запуска CODE/370 планируется использовать опцию времени выполнения языковой среды TEST, скомпилируйте хранимую процедуру с опцией TEST. В программу будет помещена информация, используемая отладчиком во время сеанса отладки.
- Выделите набор данных журнала для выходных данных CODE/370. Поместите оператор DD для этого набора журнала данных в процедуру запуска для адресного пространства хранимых процедур. Чтобы задать этот командный набор данных для CODE/370, задайте имя этого набора командных данных или имя DD в опции времени выполнения TEST. Например,

```
TEST(ALL,TESTDD,PROMPT,*)
```

указывает CODE/370, что команды нужно искать в наборе данных, связанном с именем DD TESTDD.

Первой командой в командном наборе данных должна быть команда:

```
SET LOG ON FILE имяDD;
```

Эта команда направляет вывод информации из сеанса отладки в журнальный набор данных, определенный на предыдущем шаге. Например, если в процедуре запуска адресного пространства хранимых процедур определен журнальный набор данных с именем DD INSPLOG, первой командой должна быть команда:

```
SET LOG ON FILE INSPLOG;
```

- Запустите отладчик. Можно использовать два метода:
  - Задайте опцию времени выполнения TEST. Удобнее всего это сделать в параметре RUN OPTIONS оператора CREATE PROCEDURE или ALTER PROCEDURE для хранимой процедуры.
  - Поместите в исходный код хранимой процедуры вызовы CEETEST. При использовании такого метода для существующей хранимой процедуры необходимо ее перекомпилировать, заново скомпоновать и

связать, а затем ввести команды STOP PROCEDURE и START PROCEDURE, чтобы заново загрузить эту хранимую процедуру.

Можно одновременно использовать опцию времени выполнения TEST и вызовы CEETEST. Например, можно использовать опцию TEST для задания имени командного набора данных, а вызовы CEETEST – чтобы задать места передачи управления отладчику.

Дополнительную информацию о CODE/370 смотрите в руководстве *CoOperative Development Environment/370: Debug Tool*.

## Использование опции времени выполнения MSGFILE

Опция времени выполнения языковая среда MSGFILE позволяет задать оператор JCL DD, используемый для записи сообщений отладки. Опцию MSGFILE можно использовать для направления сообщений отладки в файл DASD или в файл с буферизацией JES. При этом надо иметь в виду:

- Для каждого аргумента MSGFILE необходимо добавить в процедуру JCL, используемую для запуска адресного пространства хранимых процедур DB2, оператор DD.
- Выполните оператор ALTER PROCEDURE с параметром RUN OPTIONS, чтобы добавить опцию MSGFILE в список опций времени выполнения для этой хранимой процедуры.
- Поскольку в адресном пространстве хранимых процедур DB2 могут быть активны несколько TCB, операции ввода–вывода для набора данных, заданного в опции MSGFILE, необходимо сделать последовательными.  
Например:
  - Чтобы не допустить совместного использования набора данных несколькими хранимыми процедурами, для каждой процедуры можно задать в опции MSGFILE уникальное имя DD.
  - Если отладка прикладной программы производится редко или выполняется в тестовой системе DB2, можно сделать операции ввода–вывода последовательными, временно выполняя адресное пространство хранимых процедур DB2, задав опцию NUMTCB=1 в процедуре запуска этого адресного пространства хранимых процедур. Чтобы сделать это, обратитесь за помощью к системному администратору.

## Использование специального драйвера

Можно написать небольшой драйвер, который вызовет хранимую процедуру как подпрограмму и передаст ей список параметров, поддерживаемый этой хранимой процедурой. Затем можно протестировать и отладить эту хранимую процедуру под TSO, как обычную прикладную программу DB2. Можно использовать TSO TEST и другие обычно используемые средства отладки.

## Использование операторов SQL INSERT

Можно использовать операторы SQL, чтобы вставить отладочную информацию в таблицу DB2. Это позволит другим компьютерам в сети (например, рабочим станциям) легко обращаться к данным в этой таблице, используя DRDA.

DB2 отбрасывает отладочную информацию в прикладной программе, выполняющей оператор ROLLBACK. Чтобы избежать потери отладочных

данных, напишите вызывающую программу так, чтобы она получала эти отладочные данные перед выполнением оператора ROLLBACK.

---

## Глава 7–3. Настройка запросов

В этой главе рассказывается, как повысить эффективность запросов. Она начинается с раздела:

- “Общие советы и вопросы”

Более подробные сведения и предложения смотрите в разделах:

- “Написание эффективных предикатов” на стр. 658
- “Эффективное использование переменных хоста” на стр. 682
- “Написание эффективных подзапросов” на стр. 687

Если вы попробовали применить предложения из этих разделов, но проблемы с работой запросов еще остаются, можно попробовать другие, более рискованные методы. Смотрите раздел “Особые способы повлиять на выбор пути доступа” на стр. 693.

---

### Общие советы и вопросы

**Рекомендация:** Если ваш запрос работает плохо, посмотрите сначала следующий список и проверьте, не пропустили ли вы что-нибудь.

#### Нельзя ли упростить запрос?

Проверьте, нельзя ли написать запрос проще и эффективнее. Проверьте, не выбираются ли столбцы, которые не используются, и нет ли ненужных условий ORDER BY или GROUP BY.

#### Правильно ли написаны все предикаты?

**Индексируемые предикаты:** Убедитесь, что все предикаты, которые, по вашему мнению, должны быть индексируемыми, запрограммированы как индексируемые. В Табл. 62 на стр. 664 приводятся сведения, какие предикаты индексируемые, а какие нет.

**Избыточные или ненужные предикаты:** Постарайтесь удалить лишние и ненужные предикаты, которые вы ненамеренно поместили в текст запроса; они замедляют работу.

**Объявленные длины переменных хоста:** Убедитесь, что указанные вами размеры переменных хоста не превышают длину столбца данных, с которым они сравниваются. Если указан больший размер, предикат выполняется на 2 этапе и не может быть согласован с просмотром индекса.

Например, допустим, что переменная хоста и столбец SQL определены так:

**объявление на ассемблере определение SQL**  
MYHOSTV DS PLn 'значение' COL1 DECIMAL(6,3)

Когда используется 'n', точность переменной хоста – '2n-1'. Если n = 4 и значение = '123.123', тогда для проверки предиката WHERE COL1 = :MYHOSTV нельзя использовать согласованный просмотр индекса, потому что точности не совпадают. Чтобы избавиться от неэффективных предикатов,

использующих десятичные переменные хоста, объявит переменную хоста без опции 'Ln':

```
MYHOSTV DS P'123.123'
```

Это гарантирует, что переменная хоста будет задана так же, как и столбец SQL.

## Есть ли в запросе подзапросы?

Если в запросе используются подзапросы, обратитесь к разделу “Написание эффективных подзапросов” на стр. 687, где рассказывается, как DB2 их выполняет. Определенных правил, определяющих, как программировать подзапросы и нужно ли это делать, не существует. Однако есть некоторые общие принципы:

- Если для таблиц подзапроса существуют эффективные индексы, лучше использовать связанный подзапрос.
- Если для таблиц подзапроса нет доступных эффективных индексов, лучше будет работать несвязанный подзапрос.
- Если в родительском запросе несколько подзапросов, постарайтесь их расположить наиболее эффективным способом.

Рассмотрим следующий пример. Предположим, что в MAIN\_TABLE 1000 строк.

```
SELECT * FROM MAIN_TABLE  
WHERE TYPE IN (подзапрос 1)  
    AND  
    PARTS IN (подзапрос 2);
```

Если подзапросы 1 и 2 одного и того же типа (либо связанные, либо несвязанные), DB2 выполняет предикаты подзапроса в том порядке, в котором они встречаются в условии WHERE. Подзапрос 1 отвергает 10% всех строк, а подзапрос 2 отвергает 80% всех строк.

Предикат из подзапроса 1 (обозначим его P1) проверяется 1000 раз, а предикат из подзапроса 2 (P2) проверяется 900 раз, что в сумме дает 1900 проверок. Если же переставить предикаты, P2 будет проверяться 1000 раз, а P1 – только 200 раз, и всего будет 1200 проверок.

Получается, что размещение P2 перед P1 эффективнее (при условии, что P1 и P2 выполняются одинаковое время). Однако если P1 выполняется в 100 раз быстрее P2, тогда первым имеет смысл поместить подзапрос 1. Если вы заметили снижение производительности, попробуйте поменять подзапросы местами и посмотрите результаты. В разделе “Написание эффективных подзапросов” на стр. 687 описывается, какие причины влияют на скорость выполнения предикатов.

Если вы сомневаетесь, запустите EXPLAIN для этого запроса со связанным и несвязанным подзапросами. Посмотрев результаты работы EXPLAIN и проанализировав распределение данных и операторы SQL, вы сможете определить, какой способ эффективнее.

Этот общий принцип применим к предикатам всех типов. Но поскольку предикаты подзапросов могут использовать процессор и ввод–вывод в

тысячи раз интенсивнее, чем остальные предикаты, больше внимания следует уделить правильному расположению именно этих предикатов.

DB2 всегда выполняет несвязанные предикаты подзапроса до связанных предикатов, независимо от их расположения.

В разделе “Модификация предикатов DB2” на стр. 674 описывается, в каком порядке DB2 проверяет предикаты и в каком случае можно управлять порядком проверки.

## Есть ли в запросе функции столбцов?

Если в запросе используются функции столбцов, постарайтесь максимально упростить их; это увеличит вероятность того, что они будут выполняться при получении данных, а не после. Вообще лучше всего, когда функции столбцов выполняются во время обращения к данным, немного хуже – во время сортировки. Наименее предпочтительный вариант, когда они выполняются после выбора данных. В разделе “Когда выполняются функции столбцов? (COLUMN\_FN\_EVAL)” на стр. 721 можно прочитать, как получить нужную информацию с помощью EXPLAIN.

Чтобы функции столбцов выполнялись во время получения данных, все эти функции в запросе должны удовлетворять следующим условиям:

- Для условий GROUP BY не должна требоваться сортировка. Проверьте это по результатам, возвращенным командой EXPLAIN.
- Не должно быть предикатов 2 этапа (остаточных). Проверьте это по своей программе.
- Не должно быть функций особого набора типа COUNT(DISTINCT C1).
- Если это запрос объединения, все функции набора должны относиться к последней объединяемой таблице. Проверьте это по результатам, возвращенным командой EXPLAIN.
- Все функции столбцов должны работать с одним столбцом и не содержать арифметических выражений.

Если запрос содержит функции MAX или MIN, посмотрите на “Доступ с одной выборкой (ACCESSTYPE=I1)” на стр. 727, дает ли этот метод какие-то преимущества.

## Есть ли в предикате статического запроса SQL входные переменные?

Если в запросе используются входные переменные или маркеры параметров, фактические значения при связывании пакета или плана, содержащего запрос, не известны. Поэтому DB2 использует для определения оптимального пути доступа для оператора SQL показатель фильтрации по умолчанию. Если выбранный путь оказался неэффективным, получить лучший путь доступа можно несколькими способами.

Более подробные сведения смотрите в разделе “Эффективное использование переменных хоста” на стр. 682.

## Проблемы, связанные с корреляцией столбцов

Два столбца в таблице называются связанными, если значения столбцов не могут изменяться независимо.

DB2 может не выбрать оптимальный путь доступа, если запрос содержит связанные столбцы. Если вы подозреваете, что у вас есть проблемы со связанными столбцами, посмотрите в разделе “Корреляция столбцов” на стр. 678, что можно сделать.

## Можно ли написать запрос без использования выражений столбцов?

Следующий предикат прибавляет к столбцу SALARY значения, которые требуют повторного обращения к этому столбцу:

```
WHERE SALARY + (:hv1 * SALARY) > 50000
```

Если переписать этот предикат по—другому, DB2 сможет оценивать его эффективнее:

```
WHERE SALARY > 50000/(1 + :hv1)
```

В такой форме столбец встречается один раз с одной стороны оператора, а все прочие значения стоят с другой стороны оператора. Выражение справа называется *нестолбцовыми выражениями*. DB2 может оценивать многие предикаты с нестолбцовыми выражениями на раннем этапе обработки (этап 1), и такой запрос будет выполняться быстрее.

Дополнительную информацию о нестолбцовых выражениях и этапе 1 обработки смотрите в разделе “Свойства предикатов” на стр. 659.

---

## Написание эффективных предикатов

**Определение:** Предикаты встречаются в условиях WHERE, HAVING или ON операторов SQL; они описывают свойства данных. Обычно они определяются для столбцов таблицы и либо отбирают строки (пользуясь индексом), либо исключают строки (возвращаемые при просмотре). Полученный в результате набор отобранных или исключенных строк не зависит от выбранного для этой таблицы пути доступа.

**Пример:** В приведенном ниже запросе три предиката: предикат равенства со столбцом C1, предикат BETWEEN со столбцом C2 и предикат LIKE со столбцом C3.

```
SELECT * FROM T1  
  WHERE C1 = 10 AND  
        C2 BETWEEN 10 AND 20 AND  
        C3 NOT LIKE 'A%'
```

**Влияние на путь доступа:** В этом разделе рассказывается о том, как предикаты влияют на путь доступа. Поскольку в SQL один и тот же запрос можно представить разными способами, зная, каким образом выбор пути доступа зависит от предикатов, вы сможете писать более эффективные запросы.

В этом разделе описываются:

- “Свойства предикатов” на стр. 659
- “Общие правила проверки предикатов” на стр. 662
- “Показатели фильтрации предикатов” на стр. 669
- “Модификация предикатов DB2” на стр. 674
- “Корреляция столбцов” на стр. 678

## **Свойства предикатов**

Предикаты в условии HAVING не используются при выборе пути доступа; поэтому в данном разделе термин “предикат” относится к предикатам после ключевых слов WHERE или ON.

На выбор пути доступа влияет:

- **Тип** предиката, как описывается в разделе “Типы предикатов”
- **Индексируем** ли он – смотрите описание в разделе “Индексируемые и неиндексируемые предикаты” на стр. 660
- Проверяется ли предикат на **1 этапе** или на **2 этапе**
- Используется ли в предикате столбец ROWID – смотрите описание в разделе “Возможен ли прямой доступ к строке? (PRIMARY\_ACCESTYPE = D)” на стр. 714

Для предикатов в условии ON существуют дополнительные соображения.

**Определения:** Все предикаты можно разделить на:

### **Простые и составные**

*Составной* предикат – это предикат, полученный в результате применения к двум предикатам, простым или составным, логической операции AND или OR. Остальные предикаты – *простые*.

### **Локальный предикат или объединение**

В локальных предикатах используется только одна таблица. Они локальны по отношению к таблице и ограничивают число возвращаемых пользователю строк этой таблицы. В предикатах *объединения* используется несколько таблиц, которые указываются явно или с помощью связанной ссылки. Они определяют способ объединения строк из двух и более таблиц. Примеры использованиясмотрите в разделе “Объяснение доступа к нескольким таблицам” на стр. 728.

### **Логический терм**

Предикат, который не содержится в составном предикате с операцией OR, называется *логическим термом*. Если логический терм для какой–то строки при проверке дает ложный результат, все условие WHERE признается ложным для этой строки.

### **Типы предикатов**

Тип предиката зависит от его операции и синтаксиса, как показано ниже. Тип предиката определяет способ его обработки при проверке.

<b>Тип</b>	<b>Определение</b>
Подзапрос	Любой предикат, содержащий оператор SELECT. Например: C1 IN (SELECT C10 FROM TABLE1)

Предикат равенства Любой предикат, не являющийся подзапросом и содержащий операцию равенства и не содержащий операции NOT. Или предикат вида C1 IS NULL. Пример: C1=100

Предикат диапазона Любой предикат, не являющийся подзапросом и содержащий одну из следующих операций: >, >=, <, <=, LIKE или BETWEEN. Пример: C1>100

IN—список Предикат вида столбец IN (список значений). Например: C1 IN (5,10,15)

NOT Любой предикат, не являющийся подзапросом и содержащий операцию NOT. Пример: COL1 <> 5 or COL1 NOT BETWEEN 10 AND 20.

**Пример: Влияние типа предиката на выбор пути доступа:** Следующие два примера показывают, как тип предиката может влиять на выбор пути доступа DB2. В каждом примере предполагается, что для таблицы T1 (C1, C2) существует уникальный индекс I1 (C1), и все значения C1 целые положительные.

Запрос

```
SELECT C1, C2 FROM T1 WHERE C1 >= 0;
```

содержит предикат диапазона. Однако этот предикат не исключает ни одной строки из T1. Поэтому DB2 может определить при связывании, что просмотр таблицы эффективнее, чем просмотр индекса.

Запрос

```
SELECT * FROM T1 WHERE C1 = 0;
```

содержит предикат равенства. В этом случае DB2 выбирает доступ к индексу, потому что для получения результата требуется только одно обращение.

## Индексируемые и неиндексируемые предикаты

**Определение:** Индексируемые предикаты можно вычислить, используя значения индекса, предикаты другого типа так вычислить нельзя.

Индексируемые предикаты могут не быть предикатами, согласованными с индексом; это зависит от имеющихся индексов и пути доступа, выбранного при связывании.

**Примеры:** Если в таблице сотрудников есть индекс по столбцу LASTNAME, согласованным с индексом предикатом может быть такой предикат:

```
SELECT * FROM DSN8610.EMP WHERE LASTNAME = 'SMITH';
```

Следующий предикат не может быть согласованным с индексом предикатом, потому что он неиндексируемый.

```
SELECT * FROM DSN8610.EMP WHERE SEX <> 'F';
```

**Рекомендация:** Чтобы сделать запросы более эффективными, используйте в запросах индексируемые предикаты и создайте для таблиц соответствующие индексы. Индексируемые предикаты делают возможным согласованный просмотр индекса — в большинстве случаев этот путь доступа очень эффективен.

## **Предикаты 1 этапа и 2 этапа**

**Определение:** Строки, выбранные для запроса, обрабатываются в два этапа.

1. Предикаты 1 этапа (*согласованные с аргументом поиска*) могут применяться на первом этапе.
2. Предикаты 2 этапа (*не согласованные с аргументом поиска или остаточные*) могут применяться только на втором этапе.

Предикат 1 этапа определяется по следующим признакам:

- По синтаксису предиката

Смотрите в Табл. 62 на стр. 664 список простых предикатов и их типов. В разделе Примеры свойств предикатов приводятся сведения о типах составных предикатов.

- По типу и размеру констант в предикате

Простой предикат, который по синтаксису можно отнести к 1 этапу, может не быть предикатом 1 этапа, если он содержит константы или столбцы, типы и размеры которых не согласуются. Например, следующий предикат – не 1 этапа:

- CHARCOL='ABCDEFG', где CHARCOL определено как CHAR(6)
- SINTCOL>34.5, где SINTCOL определено как SMALLINT

Первый предикат – не предикат 1 этапа, потому что длина столбца меньше размера константы. Второй предикат – не предикат 1 этапа, потому что у столбца и константы разные типы данных.

- Проверяет ли DB2 предикат до операции объединения или после нее. Предикат, который проверяется после операции объединения – всегда предикат 2 этапа.

**Примеры:** Все индексируемые предикаты – предикаты 1 этапа. Предикат C1 LIKE %BC – тоже предикат 1 этапа, однако он не индексируем.

**Рекомендация:** По возможности используйте предикаты 1 этапа.

## **Предикаты – логические термы**

**Определение:** Предикат – логический терм, или BT–предикат – это простой или составной предикат, удовлетворяющий условию: если этот предикат дает ложный результат для какой–то строки, то и все условие WHERE для этой строки будет ложно.

**Примеры:** В приведенном ниже запросе P1, P2 и P3 – простые предикаты:

```
SELECT * FROM T1 WHERE P1 AND (P2 OR P3);
```

- P1 – простой BT–предикат.
- P2 и P3 – простые предикаты, не являющиеся BT–предикатами.
- P2 OR P3 – составной BT–предикат.
- P1 AND (P2 OR P3) – составной BT–предикат.

**Влияние на путь доступа:** При доступе с одним индексом только предикаты – логические термы выбираются в качестве согласованных с индексом предикатов. Только для индексируемых BT–предикатов может использоваться согласованный просмотр индекса. Согласованный просмотр индекса

предикатами, отличными от BT–предикатов, возможно в DB2 только при многоиндексном доступе.

При операциях объединения предикаты – логические термы могут исключать строки на более ранней стадии, чем предикаты, не являющиеся логическими термами.

**Рекомендация:** Для операций объединения по возможности используйте предикаты – логические термы.

## Предикаты в условии ON

Ключевое слово ON задает условие для внешнего объединения. Для полного объединения в условии могут использоваться только предикаты равенства. Для внешних объединений других типов можно использовать любые предикаты, кроме предикатов с подзапросами.

Для правых и левых внешних объединений и для внутренних объединений предикаты объединения в условии ON обрабатываются так же, как другие предикаты 1 и 2 этапов. Предикат 2 этапа в условии ON обрабатывается, как предикат 2 этапа внутренней таблицы.

При полном внешнем объединении условие ON проверяется при операции объединения как предикат 2 этапа.

Для внешнего объединения предикаты, оцениваемые после объединения, считаются предикатами 2 этапа. Предикаты в табличном выражении могут быть проверены до объединения и поэтому могут быть предикатами 1 этапа.

Например, в операторе

```
SELECT * FROM (SELECT * FROM DSN8610.EMP  
      WHERE EDLEVEL > 100) AS X FULL JOIN DSN8610.DEPT  
      ON X.WORKDEPT = DSN8610.DEPT.DEPTNO;
```

предикат “EDLEVEL > 100” проверяется перед полным объединением и поэтому является предикатом 1 этапа. Более подробную информацию о способах объединения смотрите в разделе “Объяснение доступа к нескольким таблицам” на стр. 728.

---

## Общие правила проверки предикатов

### *Рекомендации:*

1. С точки зрения использования ресурсов лучше, чтобы предикат проверялся как можно раньше.
2. Предикаты 1 этапа предпочтительнее предикатов 2 этапа, потому что они раньше отбирают строки и снижают объем обработки на 2 этапе.
3. По возможности старайтесь при написании запросов помещать первыми предикаты с наиболее жесткими условиями. Когда предикаты с наибольшим показателем фильтрации проверяются первыми, это позволяет исключить наибольшее число ненужных строк на самой ранней стадии, тем самым снизив затраты на обработку на более позднем этапе. Однако эти соображения применимы только к предикатам одного типа и

одного этапа проверки. Дальнейшие сведения о показателях фильтрации смотрите в разделе “Показатели фильтрации предикатов” на стр. 669.

## Порядок проверки предикатов

Порядок проверки предикатов определяют две группы правил.

Первая группа:

1. Первыми проверяются индексируемые предикаты. Все согласованные с индексом предикаты для столбцов ключа выполняются первыми и проверяются во время обращения к индексу.

Первыми применяются к индексу предикаты 1 этапа, которые не были выбраны в качестве согласованных предикатов, но ссылаются на столбцы индекса. Это называется *экранированием индекса*.

2. Затем проверяются другие предикаты 1 этапа.

После обращения к странице данных предикаты 1 этапа применяются к данным.

3. Наконец, к полученным в результате строкам применяются предикаты 2 этапа.

Вторая группа правил определяет порядок проверки предикатов внутри каждого из перечисленных выше этапов:

1. Все предикаты равенства (включая предикаты вида столбец IN список, где список состоит из одного элемента).
2. Все предикаты диапазона вида столбец IS NOT NULL
3. Все остальные предикаты.

После применения правил из этих двух групп порядок проверки задается последовательностью расположения правил в запросе. Располагая предикаты в разном порядке, вы можете управлять последовательностью проверки.

## Сводка результатов по обработке предикатов

В Табл. 62 на стр. 664 перечислены многие простые предикаты, и про каждый из них сказано, индексируем ли он и может ли проверяться на 1 этапе.

Используются следующие обозначения:

- *несьв подз* означает несвязанный подзапрос.
- *связ подз* означает связанный подзапрос.
- *оп* – одна из операций  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $\neg>$ ,  $\neg<$ .
- *значение* – константа, переменная хоста или специальный регистр.
- *шаблон* – любая символьная строка, не начинающаяся символами процента (%) или подчеркивания (\_).
- *строка* – любая символьная строка, не содержащая символов процента (%) или подчеркивания (\_).
- *выражение* – любое выражение, содержащее арифметические операции, скалярные функции, функции столбцов, операции конкатенации, столбцы, константы, переменные хоста, специальные регистры или значения даты или времени.

- *нест выраж* – нестолбцовое выражение, то есть любое выражение, в котором не участвуют столбцы. Это выражение может содержать арифметические операции, скалярные функции, операции конкатенации, константы, переменные хоста, специальные регистры и значения даты или времени.

Пример такого выражения –

CURRENT DATE - 50 DAYS

- *предикат* – любой предикат.

Вообще, если составной предикат составлен из нескольких простых предикатов, соединенных при помощи операции OR, свойства результата операции совпадают со свойствами предиката, который выполняется последним. Например, если оба операнда операции OR – индексируемые предикаты, результирующий предикат тоже индексируем. Если это операция OR над предикатом 1 этапа и предикатом 2 этапа, результат – предикат 2 этапа.

Таблица 62 (Стр. 1 из 3). Типы предикатов и их обработка

Тип предиката	Индексируем?	1 этап?	Прим.
COL = знач	Д	Д	13
COL = <i>нест выраж</i>	Д	Д	9, 11, 12
Стлб IS NULL	Д	Д	
Стлб оп значение	Д	Д	
COL оп <i>нест выраж</i>	Д	Д	9, 11
COL BETWEEN значение1 AND значение2	Д	Д	
COL BETWEEN <i>нест выраж1</i> AND <i>нест выраж2</i>	Д	Д	9, 11
значение BETWEEN COL1 AND COL2	Н	Н	
COL BETWEEN COL1 AND COL2	Н	Н	10
COL BETWEEN выражение1 AND выражение2	Н	Н	7
COL LIKE 'шаблон'	Д	Д	6
COL IN (список)	Д	Д	14
COL <> значение	Н	Д	8
COL <> <i>нест выраж</i>	Н	Д	8, 11
COL IS NOT NULL	Н	Д	
COL NOT BETWEEN значение1 AND значение2	Н	Д	
COL NOT BETWEEN <i>нест выраж1</i> AND <i>нест выраж2</i>	Н	Д	11
значение NOT BETWEEN COL1 AND COL2	Н	Н	
COL NOT IN (список)	Н	Д	

Таблица 62 (Стр. 2 из 3). Типы предикатов и их обработка

Тип предиката	Индекс– сируем?	1 этап?	Прим.
COL NOT LIKE ' строка'	Н	Д	6
COL LIKE '%строка'	Н	Д	1, 6
COL LIKE '_строка'	Н	Д	1, 6
COL LIKE переменная хоста	Д	Д	2, 6
T1.COL = T2.COL	Д	Д	16
T1.COL оп T2.COL	Д	Д	3
T1.COL <> T2.COL	Н	Д	3
T1.COL1 = T1.COL2	Н	Н	4
T1.COL1 оп T1.COL2	Н	Н	4
T1.COL1 <> T1.COL2	Н	Н	4
COL=(несв подз)	Д	Д	15
COL = ANY (несв подз)	Н	Н	
COL = ALL (несв подз)	Н	Н	
COL оп (несв подз)	Д	Д	15
COL оп ANY (несв подз)	Д	Д	
COL оп ALL (несв подз)	Д	Д	
COL <> (несв подз)	Н	Д	
COL <> ANY (несв подз)	Н	Н	
COL <> ALL (несв подз)	Н	Н	
COL IN (несв подз)	Д	Д	
COL NOT IN (несв подз)	Н	Н	
COL = (связ подз)	Н	Н	5
COL = ANY (связ подз)	Н	Н	
COL = ALL (связ подз)	Н	Н	
COL оп (связ подз)	Н	Н	5
COL оп ANY (связ подз)	Н	Н	
COL оп ALL (связ подз)	Н	Н	
COL <> (связ подз)	Н	Н	5
COL <> ANY (связ подз)	Н	Н	
COL <> ALL (связ подз)	Н	Н	
COL IN (связ подз)	Н	Н	
COL NOT IN (связ подз)	Н	Н	
EXISTS (подз)	Н	Н	
NOT EXISTS (подз)	Н	Н	
COL = выражение	Д	Д	7
выражение = значение	Н	Н	
выражение <> значение	Н	Н	
выражение оп значение	Н	Н	

Таблица 62 (Стр. 3 из 3). Типы предикатов и их обработка

Тип предиката	Индексируем?	1 этап?	Прим.
выражение от (подзапрос)	Н	Н	

**Примечания к Табл. 62 на стр. 664:**

1. Индексируем, только если указан символ ESCAPE и используется условие LIKE. Например, COL LIKE '+%строка' ESCAPE '+' – индексируемый предикат.
2. Индексируем, только если шаблон в переменной хоста содержит индексируемую константу (например, переменная хоста='строка%').
3. В каждом операторе столбцы одного типа. Примеры столбцов разных типов:
  - Разные типы данных, например, INTEGER и DECIMAL
  - Разные длины числовых столбцов, например, DECIMAL(5,0) и DECIMAL(15,0)
  - Разные десятичные точности, например, DECIMAL(7,3) и DECIMAL(7,4).

В следующих случаях столбцы считаются столбцами одного типа:

- Столбцы одного типа данных, но разных подтипов.
  - Столбцы одного типа данных, но с разными атрибутами допустимости пустых значений. (Например, в один столбец можно записывать пустые значения, а в другой – нельзя.)
  - Символьные или графические столбцы различных длин, например, CHAR(5) и CHAR(20)
4. Если и COL1, и COL2 из одной таблицы, для таких предикатов доступ по индексу ни для одного из этих столбцов не производится. Исключение – запросы такого типа:

```
SELECT * FROM T1 A, T1 B WHERE A.C1 = B.C2;
```

В этом запросе благодаря использованию внутриоператорных имен одна и та же таблица воспринимается как две разные. Поэтому для столбцов C1 и C2 возможен индексный доступ.

5. Если подзапрос уже проверен для данного значения внутриоператорного имени, DB2 может не проверять его еще раз.
6. Неиндексируемый и не 1 этапа, если для этого столбца существует процедура полей.
7. Предикат индексируем и 1 этапа, если выполняется одно из следующих условий:
  - COL – типа INTEGER или SMALLINT, и выражение имеет вид целая-константа1 арифметическая-операция целая-константа2
  - COL – типа DATE, TIME или TIMESTAMP, и:
    - выражение имеет вид:

дата-время-скалярная-функция  
(символьная-константа) или  
дата-время-скалярн.-функция (симв.-конст.) + длительн.-для-врем.-отметок  
дата-время-скалярн.-функция (симв.-конст.) - длительн.-для-врем.-отметок

- Тип *дата–время–скалярная–функция(символьная–константа)* совпадает с типом COL.
  - Численная часть *длительность–для–временных–отметок* – целое.
  - *символьная–константа*:
    - Содержит больше 7 символов для скалярной функции DATE; например, '1995-11-30'.
    - Содержит больше 14 символов для скалярной функции TIMESTAMP; например, '1995-11-30-08.00.00'.
    - Любой длины для скалярной функции TIME.
8. Условие WHERE NOT COL = значение обрабатывается так же, как условие WHERE COL <> значение и так далее.
9. Если *нест выражение*, *нест выражение1* или *нест выражение2* имеют одну из следующих форм, то предикат не индексируем:
- *нест выражение + 0*
  - *нест выражение - 0*
  - *нест выражение \* 1*
  - *нест выражение / 1*
  - *нест выражение CONCAT пустая строка*
10. COL, COL1 и COL2 могут обозначать один и тот же столбец или разные столбцы. Столбцы могут находиться в одной таблице или в разных таблицах.
11. Чтобы предикат был индексируемым и 1 этапа, у столбца и результата выражения должны быть одинаковые тип данных и длина. Например, чтобы предикат:
- COL оп *скалярная–функция*
- , где скалярная функция – HEX, SUBSTR, DIGITS, CHAR или CONCAT, был индексируемым и 1 этапа, тип и длина результата скалярной функции должны быть такими же, как соответственно тип и длина столбца.
12. Предикат будет предикатом 2 этапа в следующих случаях:
- *нест выражение* – выражение case.
  - *нест выражение* – произведение или частное двух нестолбцовых выражений – целое значение, а COL столбец типа FLOAT или DECIMAL.
13. Если тип данных COL – ROWID, DB2 пытается использовать прямой доступ к строке вместо индексного доступа или просмотра табличного пространства.
14. Если тип данных COL – ROWID, и для COL определен индекс, DB2 пытается использовать прямой доступ к строке, а не индексный доступ.
15. Неиндексируемый и не 1 этапа, если COL не допускает пустых значений, а в условии несвязанного подзапроса SELECT допустимы пустые значения.
16. Если столбцы – числовые, чтобы предикат был индексируемым предикатом 1 этапа, у них должны быть одинаковые типы данных, длины и точности. Символьные столбцы могут иметь разные типы и длины. Например, предикаты со столбцами следующих типов и длин – индексируемые предикаты 1 этапа:
- CHAR(5) и CHAR(20)

- VARCHAR(5) и CHAR(5)
- VARCHAR(5) и CHAR(20)

## Примеры свойств предикатов

Предположим, что предикаты P1 и P2 – простые, индексируемые предикаты 1 этапа, тогда:

P1 AND P2 – составной индексируемый предикат 1 этапа.

P1 OR P2 – составной предикат 1 этапа, индексируемый только с помощью объединения списков RID из двух индексов.

Следующие примеры предикатов иллюстрируют общие правила, приведенные в Табл. 62 на стр. 664. В каждом случае предполагается, что для столбцов таблицы (C1,C2,C3,C4) существуют индексы, и наименьшее значение в каждом столбце – 0.

- WHERE C1=5 AND C2=7

Оба простых предиката – предикаты 1 этапа, и составной предикат индексируем. Можно использовать согласованный просмотр индекса по столбцам C1 и C2.

- WHERE C1=5 AND C2>7

Оба простых предиката – предикаты 1 этапа, и составной предикат индексируем. Можно использовать согласованный просмотр индекса по столбцам C1 и C2.

- WHERE C1>5 AND C2=7

Оба простых предиката – предикаты 1 этапа, но только первый согласован с индексом. Можно использовать согласованный просмотр индекса по столбцу C1.

- WHERE C1=5 OR C2=7

Оба предиката – 1 этапа, но не логические термы. Составной предикат индексируем. Если DB2 использует многоиндексный доступ для составного предиката, C1 и C2 могут использоваться для согласованного просмотра. При одноиндексном доступе C1 и C2 могут использоваться только для экранирования индекса.

- WHERE C1=5 OR C2<>7

Первый предикат индексируемый и 1 этапа, второй предикат – 1 этапа, но не индексируемый. Составной предикат – 1 этапа и неиндексируемый.

- WHERE C1>5 OR C2=7

Оба предиката – 1 этапа, но не логические термы. Составной предикат индексируем. Если DB2 использует многоиндексный доступ для составного предиката, C1 и C2 могут использоваться для согласованного просмотра. При одноиндексном доступе C1 и C2 могут использоваться только для экранирования индекса.

- WHERE C1 IN (подзапрос) AND C2=C1

Оба предиката – 2 этапа и не индексируемые. Согласованный просмотр индекса невозможен, и оба предиката проверяются на 2 этапе.

- WHERE C1=5 AND C2=7 AND (C3 + 5) IN (7,8)

Только первые два предиката – 1 этапа и индексируемые. Допускается согласованный просмотр индекса, и все строки, удовлетворяющие первым двум предикатам, передаются на 2 этап для проверки третьего предиката.

- WHERE C1=5 OR C2=7 OR (C3 + 5) IN (7,8)

Третий предикат – 2 этапа. Составной предикат – 2 этапа, и все три предиката проверяются на 2 этапе. Простые предикаты – не логические термы, а составной предикат не индексируем.

- WHERE C1=5 OR (C2=7 AND C3=C4)

Третий предикат – 2 этапа. Составные предикаты (C2=7 AND C3=C4) и (C1=5 OR (C2=7 AND C3=C4)) – 2 этапа. Все предикаты проверяются на 2 этапе.

- WHERE (C1>5 OR C2=7) AND C3 = C4

Составной предикат (C1>5 OR C2=7) индексируем и 1 этапа. Простой предикат C3=C4 – не 1 этапа; поэтому согласованный просмотр индекса невозможен. Строки, удовлетворяющие составному предикату (C1>5 OR C2=7), передаются на 2 этап для проверки предиката C3=C4.

- WHERE T1.COL1=T2.COL1 AND T1.COL2=T2.COL2

Предположим, что у T1.COL1 и T2.COL1 – одинаковые типы данных, у T1.COL2 и T2.COL2 – тоже. Если у T1.COL1 и T2.COL1 разные атрибуты допустимости пустых значений, а у T1.COL2 и T2.COL2 – одинаковые, и DB2 выбирает для проверки составного предиката объединение просмотром, составной предикат – 1 этапа. Но если у T1.COL2 и T2.COL2 тоже разные атрибуты допустимости пустых значений, и DB2 выбирает объединение просмотром, составной предикат – не 1 этапа.

## Показатели фильтрации предикатов

**Определение:** показатель фильтрации предиката – это число от 0 до 1, выражающее долю строк таблицы, для которых предикат верен. Про эти строки говорят, что они *отбираются* этим предикатом.

**Пример:** Предположим, что DB2 может определить, что столбец C1 таблицы T содержит только пять различных значений: A, D, Q, W и X. Если нет другой информации, DB2 считает, что значение столбца C1 равно D в одной пятой всех строк. Тогда показатель фильтрации предиката C1='D' для таблицы T равен 0,2.

**Как DB2 использует показатели фильтрации:** Показатели фильтрации влияют на выбор пути доступа, давая возможность оценить число отобранных предикатом строк.

Для простых предикатов показатель фильтрации – функция трех переменных:

1. Литерального значения предиката; например, 'D' в предыдущем примере.
2. Операции предиката; в предыдущем примере это '=', а в отрицании этого предиката – '<>'.
3. Статистики для столбца предиката. В предыдущем примере это может быть информация, что столбец T.C1 содержит только пять значений.

**Рекомендация:** Первые две из этих переменных задаете вы, когда пишете предикат. Понимание того, как DB2 использует показатели фильтрации поможет вам писать предикаты более эффективно.

Значения третьей переменной, статистики столбца, хранятся в каталоге DB2. Вы можете изменить многие из этих значений, запустив утилиту RUNSTATS или выполнив команду UPDATE для таблицы каталога. О том, как пользоваться RUNSTATS, можно узнать из раздела, посвященного поддержке статистики в каталоге в руководстве Раздел 4 (Том 1) *DB2 Administration Guide*. Об изменении каталога вручную можно прочитать в разделе “Изменение статистики каталога” на стр. 699.

Чтобы заменить статистику каталога на собственную, нужно понимать, как DB2 использует:

- “Показатели фильтрации по умолчанию для простых предикатов”
- “Показатели фильтрации для однородных распределений данных”
- “Интерполяционные формулы” на стр. 671
- “Показатели фильтрации для всех распределений данных” на стр. 673

### **Показатели фильтрации по умолчанию для простых предикатов**

В Табл. 63 перечислены показатели фильтрации по умолчанию для разных типов предикатов. DB2 использует эти значения, когда нет другой статистики.

**Пример:** показатель фильтрации по умолчанию для предиката  $C1 = 'D'$  – 1/25 (0,04). Если на самом деле в столбце C1 встречается только пять значений, включая D, использование умолчания не приведет к выбору оптимального пути доступа.

*Таблица 63. показатели фильтрации DB2 по умолчанию для разных типов предикатов*

Тип предиката	Показатель фильтрации
Стлб = литерал	1/25
Стлб IS NULL	1/25
Стлб IN (список литералов)	(число литералов)/25
Стлб оп литерал	1/3
Стлб LIKE литерал	1/10
Стлб BETWEEN литерал1 и литерал2	1/10

#### **Примечание:**

*оп* – одна из операций <, <=, >, >=.

литерал – любое постоянное значение, известное при связывании.

### **Показатели фильтрации для однородных распределений данных**

DB2 использует показатели фильтрации, приведенные в Табл. 64 на стр. 671, если:

- Столбце COLCARDF таблицы каталога SYSIBM.SYSCOLUMNS содержит положительное значение для столбца “Стлб.”
- Для столбца “Стлб” в SYSIBM.SYSCOLDIST больше нет никакой статистики.

**Пример:** Если в столбце всего пять значений, и D – одно из них, RUNSTATS занесет в столбец COLCARDF таблицы SYSCOLUMNS значение 5. Если больше никакой статистики нет, показатель фильтрации для предиката C1 = 'D' – 1/5 (0,2).

Таблица 64. Однородные показатели фильтрации DB2 для разных типов предикатов

Тип предиката	Показатель фильтрации
Стлб = литерал	1/COLCARDF
Стлб IS NULL	1/COLCARDF
Стлб IN (список литералов)	число литералов /COLCARDF
Стлб <i>оп1</i> литерал	интерполяционная формула
Стлб <i>оп2</i> литерал	интерполяционная формула
Стлб LIKE литерал	интерполяционная формула
Стлб BETWEEN литерал1 и литерал2	интерполяционная формула

**Примечание:**

*оп1* – < или <=, и литерал – не переменная хоста.

*оп2* – > или >=, и литерал – не переменная хоста.

литерал – любое постоянное значение, известное при связывании.

**Показатели фильтрации для предикатов других типов:** В Табл. 63 на стр. 670 и Табл. 64 перечислены только наиболее распространенные типы предикатов. Если P1 – предикат и F – его показатель фильтрации, тогда показатель фильтрации предиката NOT P1 равен (1 – F). Однако так как при вычислении показателя фильтрации учитываются многие факторы, невозможно привести показатели фильтрации для всех типов предикатов.

## Интерполяционные формулы

**Определение:** Для предиката, содержащего диапазон значений, DB2 вычисляет показатель фильтрации с помощью *интерполяционной формулы*. Формула основана на оценке отношения числа значений в диапазоне к общему числу значений в столбце таблицы.

**Формулы:** Формулы, которые приводятся ниже, – приблизительные оценки, которые потом модифицируются DB2. Они применяются к предикатам вида Стлб *оп.* литерал. Величина (Всего значений) в каждой формуле определяется по значениям столбцов HIGH2KEY и LOW2KEY в таблице каталога SYSIBM.SYSCOLUMNS для столбца Стлб: Всего значений = (значение HIGH2KEY – значение LOW2KEY).

- Для операций < и <=, где литерал – не переменная хоста:  
(Значение литерала – значение LOW2KEY) / (Всего значений)
- Для операций > и >=, где литерал – не переменная хоста:  
(значение HIGH2KEY – значение литерала) / (Всего значений)
- Для LIKE или BETWEEN:  
(Верхнее значение литерала – нижнее значение литерала) / (Всего значений)

**Пример:** Предположим, что для столбца C2 предиката значение HIGH2KEY – 1400, а значение LOW2KEY – 200. Для C2 DB2 вычисляет (Всего значений) = 1200.

Для предиката C1 BETWEEN 800 AND 1100, DB2 вычисляет показатель фильтрации F по формуле:

$$F = (1100 - 800) / 1200 = 1/4 = 0,25$$

**Интерполяция для LIKE:** DB2 обрабатывает предикат LIKE так же, как предикат BETWEEN. Границы диапазона, указываемого в предикате, определяются по лiteralной строке предиката. При определении границ учитываются только символы до символа выделения ('%' или '\_'). Поэтому, если символ выделения стоит в строке первым, принимается, что показатель фильтрации равен 1, а предикат не исключает ни одной строки.

**Умолчания для интерполяции:** В некоторых случаях DB2 может не применять интерполяцию; вместо этого она использует показатель фильтрации по умолчанию. Умолчания для интерполяции:

- Применяются только для предикатов диапазона, включая LIKE и BETWEEN
- Используются, только когда интерполяция неадекватна
- Определяются по значению COLCARDF
- Используются, когда для столбца существует однородная или дополнительная статистика распределений значений, если верно одно из следующих условий:
  - Предикат не содержит констант
  - COLCARDF < 4.

В Табл. 65 приведены умолчания интерполяции для операций <, <=, >, >=, а также LIKE и BETWEEN.

Таблица 65. Показатели фильтрации по умолчанию для интерполяции

COLCARDF	Показатель для оп	Показатель для LIKE или BETWEEN
≥100000000	1/10000	3/100000
≥10000000	1/3000	1/10000
≥1000000	1/1000	3/10000
≥100000	1/300	1/1000
≥10000	1/100	3/1000
≥1000	1/30	1/100
≥100	1/10	3/100
≥0	1/3	1/10

**Примечание:** оп – одна из операций <, <=, >, >=.

## **Показатели фильтрации для всех распределений данных**

RUNSTATS может сгенерировать дополнительную статистику для столбца или набора столбцов, входящих в составной ключ индекса. DB2 может использовать эту информацию для вычисления показателей фильтрации. DB2 собирает статистику распределений значений двух видов:

Частота	Процент строк таблицы, содержащих определенное значение столбца или комбинацию значений столбцов составного ключа
Мощность	Число различных сочетаний значений столбцов

**Когда они используются:** В Табл. 66 перечислены типы предикатов, для которых используется эта статистика.

*Таблица 66. Предикаты, для которых используется статистика распределений*

Тип статистики	Столбец или столбцы ключа	предикаты
Частота	Столбец	Стлб=литерал Стлб IS NULL Стлб IN (список–литералов) Стлб оп литерал Стлб BETWEEN литерал AND литерал
Частота	Столбцы	Стлб=литерал
Мощность	Столбец	Стлб=литерал Стлб IS NULL Стлб IN (список–литералов) Стлб оп литерал Стлб BETWEEN литерал AND литерал Стлб=переменная–хоста Стлб1=Стлб2
Мощность	Столбцы	Стлб=литерал Стлб=переменная–хоста Стлб1=Стлб2

**Примечание:** оп – одна из операций <, <=, >, >=.

**Как они используются:** Столбцы COLVALUE и FREQUENCYF в таблице SYSCOLDIST содержат статистику распределений данных. Независимо от числа значений в этих столбцах RUNSTATS удаляет существующие значения и вставляет строки для наиболее часто встречающихся значений. Если RUNSTATS запущена без опции FREQVAL, она вставляет строки для 10 наиболее часто встречающихся значений первого столбца указанного индекса. Если RUNSTATS запущена с опцией FREQVAL и двумя ключевыми словами NUMCOLS и COUNT, она вставляет строки для столбцов составного индекса. В ключевом слове COUNT указывается число наиболее часто встречающихся значений. Более подробные сведения о RUNSTATSсмотрите в Разделе 2 *DB2 Utility Guide and Reference*. DB2 использует частоту из столбца FREQUENCYF для предикатов, использующих значения из столбца COLVALUE и предполагает, что остальные данные распределены равномерно.

### **Пример: Показатель фильтрации для одного столбца**

Рассмотрим предикат C1 IN ('3','5') и допустим, что SYSCOLDIST содержит следующие значения для столбца C1:

COLVALUE	FREQUENCYF
'3'	.0153
'5'	.0859
'8'	.0627

Показатель фильтрации равен  $.0153 + .0859 = .1012$ .

#### **Пример: показатель фильтрации для связанных столбцов**

Предположим, что столбцы C1 и C2 связаны и входят в составной ключ индекса. Предположим также, что есть предикат C1='3' AND C2='5', и SYSCOLDIST содержит следующие значения для столбцов C1 и C2:

COLVALUE	FREQUENCYF
'1' '1'	.1176
'2' '2'	.0588
'3' '3'	.0588
'3' '5'	.1176
'4' '4'	.0588
'5' '3'	.1764
'5' '5'	.3529
'6' '6'	.0588

Показатель фильтрации – .1176.

## **Модификация предикатов DB2**

В некоторых случаях DB2 модифицирует предикаты или генерирует дополнительные предикаты. Хотя эти изменения не видны пользователям, они непосредственно влияют на выбор пути доступа и результаты в PLAN\_TABLE. Это объясняется тем, что DB2 всегда использует индексный доступ, если это может снизить затраты. При генерации дополнительных предикатов индексируемых предикатов может стать больше, что позволяет выбрать более эффективный путь доступа.

Поэтому для понимания результатов PLAN\_TABLE необходимо знать, как DB2 обрабатывает предикаты. Полезную информацию можно также найти в Табл. 62 на стр. 664.

#### **Модификация предикатов с условием IN—список**

Если в списке IN предиката только один элемент, предикат преобразуется в предикат равенства.

Набор простых предикатов – логических термов равенства с одним и тем же столбцом, соединенных с помощью операции OR, может быть преобразован в предикат включения. Например: C1=5 or C1=10 or C1=15 преобразуется в C1 IN (5,10,15).

#### **Когда DB2 упрощает операции объединения**

Поскольку полные внешние объединения менее эффективны, чем левые или правые объединения, а последние менее эффективны, чем внутренние объединения, следует по возможности использовать в запросах самую простую операцию объединения. Однако если DB2 может упростить операцию объединения, она пытается это сделать. Вообще DB2 может упростить операцию объединения, если запрос содержит предикат или условие ON, которые исключают пустые строки с пустыми значениями, полученные при объединении.

Рассмотрим, например, такой запрос:

```
SELECT * FROM T1 X FULL JOIN T2 Y  
  ON X.C1=Y.C1  
 WHERE X.C2>12;
```

В результате внешнего объединения новая таблица будет содержать следующие строки:

- Строки с совпадающими значениями C1 в таблицах T1 и T2 (результат внутреннего объединения)
- Строки из T1, для которых в T2 нет строк с соответствующим значением C1
- Строки из T2, для которых в T1 нет строк с соответствующим значением C1

Однако после применения предиката из таблицы результата будут удалены все строки из T2, для которых в T1 нет строк с соответствующим значением C1. DB2 преобразует полное объединение в более эффективное левое объединение:

```
SELECT * FROM T1 X LEFT JOIN T2 Y  
  ON X.C1=Y.C1  
 WHERE X.C2>12;
```

В следующем примере предикат X.C2>12 исключает все пустые значения, которые получаются после правого объединения:

```
SELECT * FROM T1 X RIGHT JOIN T2 Y  
  ON X.C1=Y.C1  
 WHERE X.C2>12;
```

Поэтому DB2 может преобразовать правое объединение в более эффективное внутреннее объединение, которое дает тот же результат:

```
SELECT * FROM T1 X INNER JOIN T2 Y  
  ON X.C1=Y.C1  
 WHERE X.C2>12;
```

Чтобы DB2 могла преобразовать внешнее объединение в более простое внешнее или во внутреннее объединение, предикат, который следует за операцией объединения, должен удовлетворять следующим условиям:

- Это логический терм.
- Он ложен, если одна из таблиц при объединении дает пустые значения для всех своих столбцов.

Вот примеры предикатов, которые могут вызвать упрощение операции объединения:

- T1.C1 > 10
- T1.C1 IS NOT NULL
- T1.C1 > 10 OR T1.C2 > 15
- T1.C1 > T2.C1
- T1.C1 IN (1,2,4)
- T1.C1 LIKE 'ABC%'
- T1.C1 BETWEEN 10 AND 100
- 12 BETWEEN T1.C1 AND 100

Следующий пример показывает, как DB2 может упростить операцию объединения, если запрос содержит условие ON, которое исключает строки с несовпадающими значениями:

```
SELECT * FROM T1 X LEFT JOIN T2 Y  
    FULL JOIN T3 Z ON Y.C1=Z.C1  
    ON X.C1=Y.C1;
```

Поскольку последнее условие ON исключает из результирующей таблицы все строки с пустыми значениями столбцов из T1 или из T2, DB2 может заменить полное объединение на более эффективное левое объединение с тем же результатом:

```
SELECT * FROM T1 X LEFT JOIN T2 Y  
    LEFT JOIN T3 Z ON Y.C1=Z.C1  
    ON X.C1=Y.C1;
```

Есть один случай, когда DB2 преобразует полное внешнее объединение в левое объединение, которое невозможно задать программно. Это случай, когда производная таблица задает полное внешнее объединение, но для последующего запроса к этой производной таблице требуется только левое внешнее объединение. Например, рассмотрим такую производную таблицу:

```
CREATE VIEW V1 (C1,T1C2,T2C2) AS  
    SELECT COALESCE(T1.C1, T2.C1), T1.C2, T2.C2  
    FROM T1 X FULL JOIN T2 Y  
    ON T1.C1=T2.C1;
```

Эта таблица содержит строки с пустыми значениями C2, полученными из T1. Однако после выполнения следующего запроса эти строки будут исключены:

```
SELECT * FROM V1  
    WHERE T1C2 > 10;
```

Поэтому для данного запроса оптимальным было бы левое объединение T1 и T2. DB2 может выполнить этот запрос так, как если бы производная таблица V1 была создана с помощью левого внешнего объединения, чтобы сделать запрос более эффективным.

## Генерация предикатов транзитивного замыкания

Когда набор предикатов запроса логически подразумевает другие предикаты, DB2 может сгенерировать дополнительные предикаты, чтобы предоставить больше информации для выбора пути доступа.

**Правила генерации предикатов:** Для запросов с одной таблицей или запросов внутреннего объединения, DB2 генерирует предикаты транзитивного замыкания, если:

- Запрос содержит предикат равенства: Стлб1=Стлб2. Это может быть:
  - Локальный предикат
  - Предикат объединения
- Запрос содержит также предикат – логический терм с одним из столбцов первого предиката в одном из следующих форматов:
  - Стлб1 *оп* значение

*ОП* – =, <>, >, >=, < ИЛИ <=.

значение – константа, переменная хоста или специальный регистр.

- Стлб1 (NOT) BETWEEN значение1 AND значение2
- Стлб1=Стлб3

Для запросов внешнего объединения DB2 генерирует предикаты для транзитивного замыкания, если в запросе есть условие ON вида Стлб1=Стлб2 и перед операцией объединения находится предикат одного из следующих видов:

- Стлб1 оп значение
  - оп – =, <>, >, >=, < ИЛИ <=
- Стлб1 (NOT) BETWEEN значение1 AND значение2

DB2 генерирует предикат транзитивного замыкания для запроса внешнего объединения, только если новый предикат не ссылается на таблицу со строками, не имеющими соответствия в другой таблице. Это значит, что сгенерированный предикат не должен ссылаться на левую таблицу при левом внешнем объединении или на правую таблицу при правом внешнем объединении.

Если предикат отвечает условиям транзитивного замыкания, DB2 генерирует новый предикат, независимо от того, существует ли он уже в условии WHERE.

Сгенерированные предикаты могут иметь следующий вид:

- Стлб оп значение
  - оп – =, <>, >, >=, < ИЛИ <=.
  - значение – константа, переменная хоста или специальный регистр.
- Стлб (NOT) BETWEEN значение1 AND значение2
- Стлб1=Стлб2 (только для запросов с одной таблицей или внутренних объединений)

#### **Пример транзитивного замыкания для внутреннего объединения:**

Предположим, что вы написали запрос, удовлетворяющий условиям транзитивного замыкания:

```
SELECT * FROM T1, T2  
WHERE T1.C1=T2.C1 AND  
T1.C1>10;
```

DB2 генерирует дополнительный предикат, чтобы получить более эффективный запрос:

```
SELECT * FROM T1, T2  
WHERE T1.C1=T2.C1 AND  
T1.C1>10 AND  
T2.C1>10;
```

#### **Пример транзитивного замыкания для внешнего объединения:**

Предположим, вы написали такой запрос внешнего объединения:

```
SELECT * FROM (SELECT * FROM T1 WHERE T1.C1>10) X  
LEFT JOIN T2  
ON X.C1 = T2.C1;
```

Предикат перед операцией объединения, T1.C1>10, удовлетворяет условиям транзитивного замыкания, поэтому DB2 создает такой запрос:

```
SELECT * FROM
  (SELECT * FROM T1 WHERE T1.C1>10 AND T2.C1>10) X
  LEFT JOIN T2
  ON X.C1 = T2.C1;
```

**Избыточность предикатов:** Предикат избыточен, если выполнение других предикатов запроса дает тот же результат, что и весь запрос. Избыточные предикаты может написать пользователь или их может сгенерировать DB2. DB2 не отслеживает избыточные предикаты в запросах. Все предикаты запроса проверяются во время выполнения запроса, независимо от того, избыточны они или нет. Избыточные предикаты, которые DB2 генерирует для того, чтобы помочь выбрать путь доступа, игнорируются во время выполнения.

**Добавление предикатов:** DB2 выполняет транзитивное замыкание только для предикатов равенства и диапазона. Другие предикаты, например, IN или LIKE, могут применяться в следующем случае:

```
SELECT * FROM T1,T2
  WHERE T1.C1=T2.C1
    AND T1.C1 LIKE 'A%';
```

В данном случае добавляется предикат T2.C1 LIKE 'A%'.

## Корреляция столбцов

Два столбца данных одной таблицы, A и B, называются связанными, если значения в столбце A не могут изменяться независимо от значений столбца B.

Ниже приведен фрагмент одной большой таблицы. Столбцы CITY и STATE сильно связаны, а столбцы DEPTNO и SEX полностью независимы.

TABLE CREWINFO

CITY	STATE	DEPTNO	SEX	EMPNO	ZIPCODE
Fresno	CA	A345	F	27375	93650
Fresno	CA	J123	M	12345	93710
Fresno	CA	J123	F	93875	93650
Fresno	CA	J123	F	52325	93792
New York	NY	J123	M	19823	09001
New York	NY	A345	M	15522	09530
Miami	FL	B499	M	83825	33116
Miami	FL	A345	F	35785	34099
Los Angeles	CA	X987	M	12131	90077
Los Angeles	CA	A345	M	38251	90091

В этом простом примере для каждого значения 'Fresno' столбца CITY в столбце STATE стоит одно и то же значение ('CA').

## Как обнаружить связанные столбцы

На наличие связанных столбцов может указывать плохое время ответа при неправильном выборе пути доступа DB2. Если вам кажется, что два столбца в таблице (CITY и STATE в таблице CREWINFO) связаны, вызовите два следующих запроса SQL, результаты которых отражают связь между столбцами:

```
SELECT COUNT (DISTINCT CITY) FROM CREWINFO; (RESULT1)
SELECT COUNT (DISTINCT STATE) FROM CREWINFO; (RESULT2)
```

Результат подсчета числа значений в каждом столбце – значение COLCARDF в таблице SYSCOLUMNS каталога DB2. Перемножьте два этих значения, чтобы получить первый результат:

RESULT1 x RESULT2 = **ANSWER1**

Затем вызовите следующий оператор SQL:

```
SELECT COUNT(*) FROM
  (SELECT DISTINCT CITY,STATE
   FROM CREWINFO) AS V1;          (ANSWER2)
```

Сравните результаты (ANSWER2 и ANSWER1). Если ANSWER2 меньше ANSWER1, эти столбцы связаны.

## Влияние корреляции столбцов

DB2 может неправильно определить путь доступа, порядок таблиц или способ объединения, если в запросе используются сильно связанные столбцы. Наличие связанных столбцов могут привести к заниженной оценке стоимости операторов. Связанные столбцы влияют как на запросы с одной таблицей, так и на запросы объединения.

### *Корреляция столбцов, наиболее подходящих для индексного доступа:*

Следующий запрос выбирает строки с женщинами–сотрудниками из отдела A345 из Фресно, штат Калифорния (Fresno, California). В таблице определены 2 индекса, индекс 1 (CITY,STATE,ZIPCODE) и индекс 2 (DEPTNO,SEX).

#### Запрос 1

```
SELECT ... FROM CREWINFO WHERE
  CITY = 'FRESNO' AND STATE = 'CA'           (предикат1)
  AND DEPTNO = 'A345' AND SEX = 'F';        (предикат2)
```

Обратите внимание на два составных предиката (обозначенные предикат1 и предикат2), их фильтрующее действие (долю строк, которые они выбирают) и их показатели фильтрации в DB2. Если нужная статистика отсутствует в каталоге, показатели фильтрации вычисляются, как если бы столбцы предиката были абсолютно независимы (не связаны).

Таблица 67. Влияние корреляции столбцов на согласованные с индексом столбцы

	ИНДЕКС 1	ИНДЕКС 2
Согласованные с индексом предикаты CITY=FRESNO AND STATE=CA	предикат1 CITY=FRESNO AND STATE=CA	предикат2 DEPTNO=A345 AND SEX=F
Согласованные с индексом столбцы	2	2
оценка DB2 для согласованных с индексом столбцов (показатель фильтрации)	столбец=CITY, COLCARDF=4 Показатель фильтрации=1/4 столбец=STATE, COLCARDF=3 Показатель фильтрации=1/3	column=DEPTNO, COLCARDF=4 Показатель фильтрации=1/4 столбец=SEX, COLCARDF=2 Показатель фильтрации=1/2
Составной показатель фильтрации для согласованных с индексом столбцов	$1/4 \times 1/3 = 0.083$	$1/4 \times 1/2 = 0.125$
Специфицированных конечных страниц по оценке DB2	$0.083 \times 10 = 0.83$ ВЫБРАННЫЙ ИНДЕКС (.8 < 1.25)	$0.125 \times 10 = 1.25$
Реальный показатель фильтрации на основе распределения данных	4/10	2/10
Реальное число специфицированных конечных страниц на основе составного предиката	$4/10 \times 10 = 4$	$2/10 \times 10 = 2$ ЛУЧШИЙ ВЫБОР ИНДЕКСА (2 < 4)

DB2 выбирает индекс, возвращающий наименьшее число строк, что определяется, в числе других факторов, наименьшим показателем фильтрации для согласованных с индексом столбцов. Предположим, что на путь доступа влияет только показатель фильтрации. Объединенный показатель фильтрации столбцов CITY и STATE, на первый взгляд, очень хороший, в то время как столбцы, входящие во второй индекс, похоже, фильтруются хуже. Поэтому DB2 выбирает для запроса 1 в качестве пути доступа индекс 1.

Проблема в том, что степень фильтрации столбцов CITY и STATE на самом деле хуже. Столбец STATE почти не дает фильтрации. Поскольку столбцы DEPTNO и SEX отфильтровывают больше строк, DB2 должна предпочесть индексу 1 индекс 2.

### **Корреляция столбцов, используемых для экранирования индекса**

Несогласованные с индексом столбцы, используемые для экранирования индекса, также могут быть связаны. Более подробно об этом можно прочитать в разделе “Несогласованный просмотр индекса (ACCESSTYPE=1 и MATCHCOLS=0)” на стр. 725. Предикаты индексного экранирования позволяют уменьшить число строк данных, отобранных при просмотре индекса. Однако если предикаты индексного экранирования связаны, они отфильтровывают меньше строк данных, чем должны были бы в соответствии с показателем фильтрации. Чтобы проверить это, воспользуйтесь запросом 1 (смотрите страницу 679) со следующими индексами таблицы CREWINFO (страница 678):

индекс 3 (EMPNO,CITY,STATE)  
индекс 4 (EMPNO,DEPTNO,SEX)

Если выбран индекс 3, из-за того, что столбцы CITY и STATE предиката 1 связаны, предикаты экранирования не настолько улучшают доступ по индексу, как это предполагалось, и, видимо, в данном случае лучше использовать

индекс 4. (Заметим, что экранирование индекса используется также для индексов, где есть согласованные с индексом столбцы.)

**Объединения нескольких таблиц:** В запросе 2 к исходному запросу добавляется дополнительная таблица (смотрите запрос 1 на странице 679), чтобы продемонстрировать влияние корреляции столбцов на запросы объединения.

#### **TABLE DEPTINFO**

CITY	STATE	MANAGER	DEPT	DEPTNAME
FRESNO	CA	SMITH	J123	ADMIN
LOS ANGELES	CA	JONES	A345	LEGAL

#### **Запрос 2**

```
SELECT ... FROM CREWINFO T1,DEPTINFO T2  
      WHERE T1.CITY = 'FRESNO' AND T1.STATE='CA'          (предикат 1)  
        AND T1.DEPTNO = T2.DEPT AND T2.DEPTNAME = 'LEGAL';
```

Порядок доступа к таблицам в операторе объединения влияет на производительность. Предполагаемый составной показатель фильтрации предиката 1 меньше его реального показателя фильтрации. Поэтому хотя на первый взгляд может показаться, что лучше начинать с таблицы CREWINFO, на самом деле это может оказаться не так.

Кроме того, из-за меньшего, согласно оценке, размера таблицы CREWINFO может быть выбрано вложенное объединение. Но если из-за того, что предикат 1 отфильтровывает меньше строк, чем оценивалось, из таблицы CREWINFO отбирается слишком много строк, возможно, лучше было бы использовать другой способ объединения.

#### **Что делать, если есть связанные столбцы**

Если из-за корреляции столбцов DB2 выбирает неверный путь доступа, попробуйте применить один из следующих приемов для изменения пути доступа:

- Если связанные столбцы входят в составной ключ индекса, запустите утилиту RUNSTATS с опциями KEYCARD и FREQVAL. Это наиболее предпочтительный способ.
- Изменение статистики каталога вручную.
- Используйте SQL, который вынуждает DB2 выбрать доступа по нужному индексу.

Два последних способа описываются в разделе “Особые способы повлиять на выбор пути доступа” на стр. 693.

Утилита RUNSTATS собирает статистику, которая требуется DB2, чтобы выбирать правильные способы обработки запросов. С помощью RUNSTATS можно собирать статистику для столбцов составных ключей индекса и определять число различных сочетаний значений для этих столбцов. Это дает DB2 необходимую информацию для подсчета показателя фильтрации для запроса.

RUNSTATS собирает полезную для запросов статистику примерно так:

```
SELECT * FROM T1  
WHERE C1 = 'a' AND C2 = 'b' AND C3 = 'c' ;
```

где:

- Используются первые три ключа индекса (MATCHCOLS = 3).
- Индекс существует для C1, C2, C3, C4, C5.
- Некоторые или все столбцы индекса в той или иной степени связаны.

Прочтайте в Разделе 5 (Том 2) *DB2 Administration Guide*, как использовать RUNSTATS, чтобы повлиять на выбор пути доступа.

---

## Эффективное использование переменных хоста

**Для переменных хоста требуются показатели фильтрации по умолчанию:**

При связывании статического оператора SQL, который содержит переменные хоста, DB2 использует показатель фильтрации по умолчанию, чтобы определить наилучший путь доступа для этого оператора. Более подробные сведения о показателях фильтрации, в том числе о значениях по умолчанию,смотрите в разделе “Показатели фильтрации предикатов” на стр. 669.

DB2 часто выбирает путь доступа, при котором запрос с несколькими переменными хоста работает хорошо. Однако в новой версии или после проведения каких-либо работ DB2 может выбрать другой путь доступа, который окажется хуже. В большинстве случаев изменение пути доступа определяют показатели фильтрации по умолчанию, изменение которых может привести к тому, что DB2 будет оптимизировать запрос по-другому.

Изменить путь доступа для запроса, содержащего переменные хоста, можно двумя способами:

- Связать пакет или план, содержащий запрос, с опцией REOPT(VARS).
- Переписать запрос по-другому.

## Использование опции REOPT(VARS) для изменения пути доступа во время выполнения

Используйте опцию связывания REOPT(VARS), если хотите, чтобы DB2 определяла путь доступа и при связывании, и во время выполнения для операторов, которые содержат следующие элементы:

- переменные хоста
- маркеры параметров
- специальные регистры

Во время выполнения DB2 использует значения этих переменных для определения пути доступа.

Поскольку повторная оптимизация пути доступа во время выполнения связана с определенными затратами, опцию связывания REOPT(VARS) следует использовать только для пакетов и планов, которые работают неэффективно.

Будьте внимательны при использовании опции REOPT(VARS) для операторов, выполняемых в цикле; оптимизация происходит при каждом выполнении оператора. Однако если используется указатель, можно включить в цикл

операторы FETCH, потому что оптимизация производится только при открытии указателя.

Чтобы более эффективно использовать опцию REOPT(VARS), сначала определите, какие операторы SQL работают плохо. Выделите эти операторы в отдельные единицы, которые свяжите в пакеты с опцией REOPT(VARS). Остальные операторы свяжите с опцией NOREOPT(VARS). Затем свяжите план с опцией NOREOPT(VARS). Только операторы в пакетах, связанных с опцией REOPT(VARS), могут быть повторно оптимизированы во время выполнения.

Чтобы определить, какие запросы в планах и пакетах, связанных с опцией REOPT(VARS), будут повторно оптимизированы во время выполнения, выполните следующие операторы SELECT:

```
SELECT PLNAME,
CASE WHEN STMTNOI <> 0
    THEN STMTNOI
    ELSE STMTNO
END AS STMTNUM,
SEQNO, TEXT
FROM SYSIBM.SYSSTMT
WHERE STATUS IN ('B','F','G','J')
ORDER BY PLNAME, STMTNUM, SEQNO;

SELECT COLLID, NAME, VERSION,
CASE WHEN STMTNOI <> 0
    THEN STMTNOI
    ELSE STMTNO
END AS STMTNUM,
SEQNO, STMT
FROM   SYSIBM.SYSPACKSTMT
WHERE STATUS IN ('B','F','G','J')
ORDER BY COLLID, NAME, VERSION, STMTNUM, SEQNO;
```

Если указана опция связывания VALIDATE(RUN), и оператор в плане или пакете не был связан успешно, этот оператор дополнительно связывается во время выполнения. Если указана также опция REOPT(VARS), DB2 повторно оптимизирует путь доступа при дополнительном связывании.

Чтобы определить, какие планы и пакеты содержат операторы, которые будут связаны при выполнении, выполните следующие операторы SELECT:

```
SELECT DISTINCT NAME
  FROM SYSIBM.SYSSTMT
 WHERE STATUS = 'F' OR STATUS = 'H';

SELECT DISTINCT COLLID, NAME, VERSION
  FROM   SYSIBM.SYSPACKSTMT
 WHERE STATUS = 'F' OR STATUS = 'H';
```

## Изменение запросов, чтобы повлиять на выбор пути доступа

Следующие примеры демонстрируют возможные проблемы с работой запросов и предлагают способы настройки запросов. Однако, перед тем как изменить запрос, проверьте, нельзя ли решить эту проблему с помощью опции REOPT(VARS). Дополнительные сведения об опции REOPT(VARS) смотрите в

разделе “Использование опции REOPT(VARS) для изменения пути доступа во время выполнения” на стр. 682.

### ***Пример 1: Предикат равенства***

Показатель фильтрации по умолчанию для предиката равенства – 1/COLCARDF. Реально показатель фильтрации может быть другим.

#### ***Запрос:***

```
SELECT * FROM DSN8610.EMP  
WHERE SEX = :HV1;
```

**Предположения:** Поскольку в столбце SEX (пол) только два различных значения – 'M' и 'F', значение COLCARDF для SEX равно 2. Если число сотрудников–мужчин и сотрудников–женщин разное, реальный показатель фильтрации либо больше, либо меньше значения по умолчанию, в зависимости от значения :HV1 ('M' или 'F').

**Рекомендация:** Выбрать более правильный путь доступа можно одним из следующих способов:

- Связать пакет или план, содержащий запрос, с опцией REOPT(VARS). Это заставит DB2 выполнить повторную оптимизацию во время выполнения запроса с использованием введенных значений.
- Написать предикаты так, чтобы, зная настоящие показатели фильтрации, повлиять на выбор пути доступа DB2. Например, приведенный выше запрос можно разбить на три запроса, в двух из которых используются константы. В большинстве случаев DB2 сможет тогда определить точный показатель фильтрации при связывании плана.

```
SELECT (HV1);  
WHEN ('M')  
DO;  
EXEC SQL SELECT * FROM DSN8610.EMP  
WHERE SEX = 'M';  
END;  
WHEN ('F')  
DO;  
EXEC SQL SELECT * FROM DSN8610.EMP  
WHERE SEX = 'F';  
END;  
OTHERWISE  
DO:  
EXEC SQL SELECT * FROM DSN8610.EMP  
WHERE SEX = :HV1;  
END;  
END;
```

### ***Пример 2: Известные диапазоны***

У таблицы T1 два индекса: T1X1 для столбца C1 и T1X2 для столбца C2.

#### ***Запрос:***

```
SELECT * FROM T1  
WHERE C1 BETWEEN :HV1 AND :HV2  
AND C2 BETWEEN :HV3 AND :HV4;
```

**Предположения:** Известно, что:

- В прикладной программе всегда задается небольшой диапазон значений для C1 и большой диапазон для C2.
- Предпочтительный путь доступа – по индексу T1X1.

**Рекомендация:** Если DB2 не выбирает индекс T1X1, перепишите запрос следующим образом, чтобы DB2 не выбирала индекс T1X2 на C2:

```
SELECT * FROM T1
    WHERE C1 BETWEEN :HV1 AND :HV2
        AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

### **Пример 3: Переменные диапазоны**

У таблицы T1 два индекса: T1X1 для столбца C1 и T1X2 для столбца C2.

**Запрос:**

```
SELECT * FROM T1
    WHERE C1 BETWEEN :HV1 AND :HV2
        AND C2 BETWEEN :HV3 AND :HV4;
```

**Предположения:** Известно, что в прикладной программе для C1 и C2 могут задаваться как небольшие, так и большие диапазоны значений. Поэтому, пользуясь показателями фильтрации по умолчанию, DB2 не сможет выбрать оптимальный путь доступа во всех случаях. Например, при небольшом диапазоне значений C1 лучше использовать индекс T1X1 по C1, при небольшом диапазоне C2 предпочтителен индекс T1X2 по C2, а при больших диапазонах и C1, и C2 предпочтительнее просмотр табличного пространства.

**Рекомендация:** Если DB2 выбирает неоптимальный путь доступа, попробуйте изменить прикладную программу одним из следующих способов:

- Используйте динамический оператор SQL и задайте в операторе диапазоны C1 и C2. Зная границы диапазона, DB2 может определить реальные показатели фильтрации для запроса. Подготовка оператора при каждом его выполнении требует дополнительных действий, однако это оправдано, если в запросе обрабатывается большое количество данных.
- Включите в запрос простые логические проверки диапазонов C1 и C2, а затем выполните один из следующих статических операторов SQL, в зависимости от диапазонов C1 и C2:

```
SELECT * FROM T1
    WHERE C1 BETWEEN :HV1 AND :HV2
        AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

```
SELECT * FROM T1 WHERE C2 BETWEEN :HV3 AND :HV4
    AND (C1 BETWEEN :HV1 AND :HV2 OR 0=1);
```

```
SELECT * FROM T1 WHERE (C1 BETWEEN :HV1 AND :HV2 OR 0=1)
    AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

### **Пример 4: ORDER BY**

У таблицы T1 два индекса: T1X1 для столбца C1 и T1X2 для столбца C2.

**Запрос:**

```
SELECT * FROM T1
  WHERE C1 BETWEEN :HV1 AND :HV2
  ORDER BY C2;
```

В этом примере DB2 может выбрать одно из следующих действий:

- Сканировать индекс T1X1 и затем отсортировать результаты по столбцу C2
- Сканировать табличное пространство, в котором расположена T1, а затем отсортировать результаты по столбцу C2
- Сканировать индекс T1X2, а затем применить предикат к каждой строке данных, отказавшись от сортировки

Какой способ лучше, зависит от следующих факторов:

- Числа строк, которые удовлетворяют предикату диапазона
- У какого индекса наибольшее отношение кластеризации

Если реальное число строк, которые удовлетворяют предикату диапазона, сильно отличается от оценки, DB2 может выбрать не лучший путь доступа.

**Предположения:** Вас не устраивает выбор DB2.

**Рекомендация:** Используйте в прикладной программе динамический оператор SQL и задайте в операторе диапазон C1. Это позволит DB2 использовать истинный показатель фильтрации, а не заданный по умолчанию, но потребует дополнительных затрат на выполнение оператора PREPARE.

### **Пример 5: Операция объединения**

В каждой из таблиц A, B и есть индексы для столбцов C1, C2, C3 и C4.

**Запрос:**

```
SELECT * FROM A, B, C
  WHERE A.C1 = B.C1
    AND A.C2 = C.C2
    AND A.C2 BETWEEN :HV1 AND :HV2
    AND A.C3 BETWEEN :HV3 AND :HV4
    AND A.C4 < :HV5
    AND B.C2 BETWEEN :HV6 AND :HV7
    AND B.C3 < :HV8
    AND C.C2 < :HV9;
```

**Предположения:** Истинные показатели фильтрации для таблицы A гораздо больше показателей по умолчанию. Поэтому DB2 использует заниженную оценку числа строк, отобранных из таблицы A, и неправильно выбирает ее в качестве первой таблицы для объединения.

**Рекомендации:** Вы можете:

- Снизить оценку размера таблицы A, добавив предикаты
- Запретить использование индекса при объединении, сделав предикат объединения неиндексируемым

В приведенном ниже запросе применяется второй способ.

```
SELECT * FROM T1 A, T1 B, T1 C
  WHERE (A.C1 = B.C1 OR 0=1)
        AND A.C2 = C.C2
        AND A.C2 BETWEEN :HV1 AND :HV2
        AND A.C3 BETWEEN :HV3 AND :HV4
        AND A.C4 < :HV5
        AND B.C2 BETWEEN :HV6 AND :HV7
        AND B.C3 < :HV8
        AND C.C2 < :HV9;
```

Так как предикат объединения таблиц А и В стал неиндексируемым (его нельзя использовать для одноиндексного доступа), использовать индекс по столбцу С1 невозможно. Это, в свою очередь, может привести к тому, что DB2 будет вначале обращаться к таблицам А или В. Или же DB2 изменит тип доступа для таблицы А или В, что повлияет на последовательность объединения других таблиц.

## Написание эффективных подзапросов

**Определения:** Подзапрос – это оператор SELECT внутри условия WHERE или HAVING другого оператора SQL.

**Требуется решение:** Часто можно написать два или несколько операторов SQL, которые дают один и тот же результат, особенно если используются подзапросы. При этом для операторов могут использоваться разные пути доступа и их производительность также может быть разной.

**Обзор тем:** Дальше рассказывается о различных способах получения результата, предусмотренного подзапросом, и о том, что в каждом случае делает DB2. Эти сведения помогут вам определить, какой способ лучше использовать для конкретного запроса.

В первых двух способах используются различные типы подзапросов:

- “Связанные подзапросы”
- “Несвязанные подзапросы” на стр. 688

Подзапрос иногда можно преобразовать в операцию объединения. Иногда это делает DB2, чтобы выбрать лучший путь доступа, а иногда вы можете это сделать самостоятельно, чтобы получить лучшие результаты. Третий способ –

- “Преобразование подзапроса в операцию объединения” на стр. 690

Посмотрите сравнение этих трех способов применительно к одной задаче в разделе

- “Настройка подзапроса” на стр. 691

## Связанные подзапросы

**Определение:** Связанный подзапрос ссылается по крайней мере на один столбец внешнего запроса.

Любой предикат, который содержит связанный подзапрос – это предикат 2 этапа.

**Пример:** В приведенном ниже запросе внутриоператорное имя X – пример ссылки из подзапроса на внешний блок запроса.

```
SELECT * FROM DSN8610.EMP X
  WHERE JOB = 'DESIGNER'
    AND EXISTS (SELECT 1
                 FROM DSN8610.PROJ
                 WHERE DEPTNO = X.WORKDEPT
                   AND MAJPROJ = 'MA2100');
```

**Действия DB2:** Связанный подзапрос проверяется для каждой отобранный строки внешнего запроса, на которую есть ссылки. При выполнении запроса DB2:

1. Читает строку из таблицы EMP, где JOB='DESIGNER'.
2. Ищет значение WORKDEPT из этой строки в таблице, хранящейся в памяти.  
Таблица в памяти позволяет выполнять подзапрос меньшее число раз. Если подзапрос уже был выполнен с данным значением WORKDEPT, результат был сохранен в таблице, и для текущей строки DB2 не выполняет подзапрос еще раз. Вместо этого DB2 переходит к шагу 5.
3. Выполняет подзапрос, если данного значения WORKDEPT нет в памяти. Для этого требуется поиск в таблице PROJ, чтобы проверить, есть ли проект, для которого MAJPROJ равно 'MA2100' и за который отвечает текущий WORKDEPT.
4. Сохраняет значение WORKDEPT и результат подзапроса в памяти.
5. Возвращает прикладной программе текущую строку таблицы EMP.

DB2 повторяет этот процесс для каждой отобранный строки таблицы EMP.

**Замечания относительно таблицы в памяти:** Таблица в памяти создается, если в предикате, содержащем подзапрос, есть одна из следующих операций:

<, <=, >, >=, =, <>, EXISTS, NOT EXISTS

Но таблица не используется, если:

- Если в подзапросе более 16 связанных столбцов
- Суммарная длина связанных столбцов превышает 256 байт
- На наборе связанных столбцов таблицы из внешнего запроса существует уникальный индекс

Таблица в памяти – "циклическая" таблица, которая не гарантирует сохранения результатов для всех возможных повторений подзапроса.

## Несвязанные подзапросы

**Определение:** Несвязанный подзапрос не содержит ссылок на внешние запросы.

**Пример:**

```
SELECT * FROM DSN8610.EMP
  WHERE JOB = 'DESIGNER'
    AND WORKDEPT IN (SELECT DEPTNO
                      FROM DSN8610.PROJ
                     WHERE MAJPROJ = 'MA2100');
```

**Действия DB2:** Несвязанный подзапрос выполняется при открытии указателя для запроса. Действия DB2 зависят от того, одно значение возвращает запрос или несколько. Запрос из последнего примера может возвращать несколько значений.

### Подзапросы с одним значением

Если подзапрос содержится в предикате с простой операцией, он должен возвращать 1 или 0 строк. Простая операция — это одна из операций:

<, <=, >, >=, =, <>, EXISTS, NOT EXISTS

Следующий несвязанный подзапрос возвращает одно значение:

```
SELECT *
  FROM DSN8610.EMP
  WHERE JOB = 'DESIGNER'
    AND WORKDEPT <= (SELECT MAX(DEPTNO)
                      FROM DSN8610.PROJ);
```

**Действия DB2:** Подзапрос выполняется при открытии указателя. Если подзапрос возвращает несколько строк, DB2 генерирует ошибку. Предикат, содержащий подзапрос, воспринимается как простой предикат, в котором указанна константа, например, WORKDEPT <= 'значение'.

**Обработка на 1 и на 2 этапе:** Правила определения, на каком этапе выполняется предикат с несвязанным подзапросом, возвращающим одно значение, в основном такие же, как для такого же предиката с одной переменной. Однако предикат выполняется на 2 этапе, если:

- Подзапрос может возвращать пустые значения, а столбец внешнего запроса не допускает пустые значения.
- Тип данных подзапроса выше типа данных столбца внешнего запроса. Например, следующий предикат — предикат 2 этапа:

WHERE SMALLINT\_COL < (SELECT INTEGER\_COL FROM ...)

### Подзапросы с несколькими значениями

Подзапрос может возвращать несколько значений, если используется одна из следующих операций:

on ANY on ALL on SOME IN EXISTS

где оп — одна из операций >, >=, <, or <=.

**Действия DB2:** Если возможно, DB2 заменяет подзапрос, возвращающий несколько строк, на подзапрос, возвращающий только одну строку. Так происходит, если вместе с ключевыми словами ANY, ALL или SOME есть операция сравнения. Пример — такой запрос:

```
SELECT * FROM DSN8610.EMP
  WHERE JOB = 'DESIGNER'
    AND WORKDEPT <= ANY (SELECT DEPTNO
                           FROM   DSN8610.PROJ
                           WHERE MAJPROJ = 'MA2100');
```

DB2 вычисляет максимальное значение DEPTNO по таблице DSN8610.PROJ и удаляет из запроса ключевое слово ANY. После этого преобразования подзапрос воспринимается как подзапрос с одним значением.

Такое преобразование может быть произведено с *максимальным значением*, если используется операция сравнения:

- > или >= с условием ALL
- < или <= с условием ANY или SOME

Преобразование может быть произведено с *минимальным значением*, если используется операция сравнения:

- < или <= с условием ALL
- > или >= с условием ANY или SOME

Этап выполнения для полученного в результате предиката определяется по тем же правилам, как и для предиката с подзапросом, который возвращает одно значение.

**Когда для подзапроса выполняется сортировка:** Результаты несвязанного подзапроса сортируются в порядке убывания, если используется операция сравнения IN, NOT IN, = ANY, <> ANY, = ALL или <> ALL. Сортировка облегчает проверку предиката, сокращая просмотр результатов подзапроса. Когда значение подзапроса становится меньше или равно выражению в левой части, просмотр может быть прекращен, а предикат оценен как истинный или ложный.

Если результат запроса символьного типа, и в левой части предиката стоит выражение типа даты и времени, результат помещается в рабочий файл без сортировки. Для некоторых несвязанных подзапросов, в которых используются перечисленные выше операции сравнения, DB2 может более точно указать входную точку для рабочего файла, тем самым сокращая необходимый просмотр.

**Результаты EXPLAIN:** Сведения о результате в таблице плана для сортируемого подзапроса смотрите в разделе “Когда выполняются функции столбцов? (COLUMN\_FN\_EVAL)” на стр. 721.

## Преобразование подзапроса в операцию объединения

Подзапрос может быть преобразован в операцию объединения таблицы—результата подзапроса и таблицы—результата внешнего запроса, если это преобразование не приводит к избыточности.

DB2 выполняет это преобразование, только если:

- Подзапрос находится в условии WHERE.
- Подзапрос не содержит условий GROUP BY, HAVING или функций столбцов.

- В условии FROM подзапроса только одна таблица.
- После преобразования в объединении участвуют не больше 15 таблиц
- Список SELECT подзапроса состоит только из одного столбца, содержащего уникальные значения, что обеспечивается уникальным индексом.
- Оператор сравнения предиката, содержащего подзапрос – IN, = ANY или = SOME.
- В случае несвязанного подзапроса левая часть предиката – один столбец того же типа данных и длины, что и столбец подзапроса. (Для связанного подзапроса левая часть может быть любым выражением.)

**Пример:** Следующий подзапрос может быть преобразован в операцию объединения:

```
SELECT * FROM EMP
  WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT
                    WHERE LOCATION IN ('SAN JOSE', 'SAN FRANCISCO')
                    AND DIVISION = 'MARKETING');
```

Если в торговом подразделении есть отдел с филиалами в Сан–Хосе и в Сан–Франциско, результат данного оператора SQL будет отличаться от результата операции объединения. При объединении каждый сотрудник этого отдела будет учитываться дважды, первый раз для Сан–Хосе, второй раз для Сан–Франциско, хотя это один и тот же отдел. Поэтому, очевидно, для преобразования подзапроса в операцию объединения необходимо, чтобы список SELECT подзапроса был уникaлен. В данном примере это может обеспечить уникальность индекса для одного из следующих сочетаний столбцов:

- (DEPTNO)
- (DIVISION, DEPTNO)
- (DEPTNO, DIVISION).

Полученный в результате запрос:

```
SELECT EMP.* FROM EMP, DEPT
  WHERE EMP.DEPTNO = DEPT.DEPTNO AND
        DEPT.LOCATION IN ('SAN JOSE', 'SAN FRANCISCO') AND
        DEPT.DIVISION = 'MARKETING';
```

**Результаты EXPLAIN:** О результате в таблице плана для подзапроса, преобразованного в операцию объединения, можно прочитать в разделе “Преобразуется ли подзапрос в операцию объединения?” на стр. 721.

## Настройка подзапроса

Следующие три запроса возвращают в качестве результата один и тот же набор строк. В этих запросах выбираются данные обо всех разработчиках из отделов, отвечающих за проекты, входящие в главный проект MA2100. Этот пример демонстрирует возможность получения нужного результата несколькими способами.

### Запрос А: Объединение двух таблиц

```
SELECT DSN8610.EMP.* FROM DSN8610.EMP, DSN8610.PROJ  
  WHERE JOB = 'DESIGNER'  
    AND WORKDEPT = DEPTNO  
    AND MAJPROJ = 'MA2100';
```

### **Запрос B: Связанный подзапрос**

```
SELECT * FROM DSN8610.EMP X  
  WHERE JOB = 'DESIGNER'  
    AND EXISTS (SELECT 1 FROM DSN8610.PROJ  
      WHERE DEPTNO = X.WORKDEPT  
        AND MAJPROJ = 'MA2100');
```

### **Query C: Несвязанный подзапрос**

```
SELECT * FROM DSN8610.EMP  
  WHERE JOB = 'DESIGNER'  
    AND WORKDEPT IN (SELECT DEPTNO FROM DSN8610.PROJ  
      WHERE MAJPROJ = 'MA2100');
```

Если на выходе требуются столбцы и из таблицы EMP, и из PROJ, необходимо использовать объединение.

Если в подзапросе значения DEPTNO могут повторяться, его нельзя преобразовать в эквивалентную операцию объединения.

В общем случае наиболее эффективным, наверное, окажется запрос А. Однако если в таблице нет индекса по DEPTNO, более эффективным может быть запрос С. Если нельзя использовать операцию объединения, и у таблицы PROJ есть индекс по DEPTNO, лучшим может оказаться запрос В.

Если возникли проблемы с подзапросом, посмотрите, нельзя ли его написать по-другому, или создать индекс, чтобы увеличить производительность подзапроса.

Важно также знать последовательность выполнения различных предикатов подзапроса, а также других предикатов запроса. Если предикат подзапроса неэффективен, возможно, перед ним можно выполнить другой предикат, чтобы уменьшить число строк, с которыми работает данный предикат.

## Особые способы повлиять на выбор пути доступа

### ВНИМАНИЕ

В этом разделе излагаются способы изменения запросов и статистики каталога с целью повлиять на выбор пути доступа DB2. В будущих версиях DB2 способ выбора может измениться, и ваши изменения могут привести к снижению производительности. Сохраните старую статистику каталога или текст программы SQL перед тем как производить какие-либо изменения, чтобы повлиять на выбор пути доступа. До и после сделанных изменений проверьте работу запроса. После переноса в следующую версию снова проверьте, как работает запрос. Если производительность снизилась, возможно, следует отменить внесенные изменения.

В этом разделе содержится следующая информация об определении и изменении пути доступа:

- Получение информации о пути доступа
- “Минимизация затрат при получении небольшого числа строк: OPTIMIZE FOR n ROWS” на стр. 694
- “Уменьшение числа согласованных с индексом столбцов” на стр. 696
- “Добавление дополнительных локальных предикатов” на стр. 698
- “Изменение статистики каталога” на стр. 699

## Получение информации о пути доступа

Получить информацию о пути доступа DB2 можно несколькими путями:

- Использовать Visual Explain

Утилиту DB2 Visual Explain, которая вызывается с рабочей станции клиента, можно использовать для вывода и анализа информации о путях доступа, выбираемых DB2. Утилита поддерживает удобный интерфейс для работы с данными PLAN\_TABLE и позволяет вызывать EXPLAIN для динамических операторов SQL. Она также дает доступ к статистике каталога для определенных объектов, которые влияют на выбор пути доступа. Кроме того, утилита позволяет архивировать выходные данные EXPLAIN для предыдущих операторов SQL, что можно использовать для анализа изменений в среде SQL. Более подробные сведения смотрите в *оперативной справке по DB2 Visual Explain*.

- Получить отчеты об учетных записях DB2 Performance Monitor.

Другой способ проследить за работой запросов – с помощью отчетов об учетных записях DB2 Performance Monitor. В отчете в кратком формате перечисляются основные рабочие характеристики, упорядоченные по PLANNAME. Сравните планы, содержащие операторы SQL, для которых вы пытались повлиять на выбор путей доступа. Если время работы, время TCB или число требований getpage резко возрастает без соответствующего увеличения активности SQL, это может означать наличие проблемы. Можно использовать DB2 PM Online Monitor, чтобы проследить за событиями после внесения изменений и сразу выяснить, как сказываются ваши изменения.

- Указать опцию связывания EXPLAIN

Можно также использовать опцию EXPLAIN при связывании или повторном связывании плана или пакета. Сравните новый план или пакет для данного оператора со старым. Если в новом плане или пакете используется просмотр табличного пространства или несогласованный просмотр индекса, а в старом нет, проблема, скорее всего, в операторе. Проверьте изменения в пути доступа в новом плане или пакете; они могли привести к повышению или снижению производительности. Если ни учетный отчет, упорядоченный по PLANNAME или PACKAGE, ни оператор EXPLAIN не позволяют определить действия по исправлению ситуации, используйте отчеты об активности DB2 PM SQL, чтобы получить дополнительную информацию. Более подробно об использовании EXPLAIN читайте в разделе “Получение информации PLAN\_TABLE с помощью EXPLAIN” на стр. 702.

## **Минимизация затрат при получении небольшого числа строк: OPTIMIZE FOR n ROWS**

Когда прикладная программа выполняет оператор SELECT, DB2 предполагает, что программа получит все отобранные строки. Это предположение вполне оправдано при пакетной обработке. Однако для интерактивных прикладных программ SQL, как например, SPUFI, запросы обычно определяют очень большой потенциальный набор результатов, но программа получает только первые несколько строк. Путь доступа, выбираемый DB2, может оказаться неоптимальным для таких интерактивных программ.

В этом разделе рассказывается, как условие OPTIMIZE FOR n ROWS сказывается на производительности интерактивных прикладных программ SQL. Эти сведения относятся к локальным прикладным программам, если особо не оговорено обратное. Более подробные сведения об использовании условия OPTIMIZE FOR n ROWS в распределенных прикладных программахсмотрите в разделе “Задание опции OPTIMIZE FOR n ROWS” на стр. 425.

**Применение условия OPTIMIZE FOR n ROWS:** Условие OPTIMIZE FOR n ROWS позволяет прикладной программе сообщить о своем намерении:

- Получить только часть набора результатов или
- Сделать приоритетным получение нескольких первых строк

DB2 использует условие OPTIMIZE FOR n ROWS, чтобы выбрать пути доступа, минимизирующие время ответа для получения нескольких первых строк. Для распределенных запросов значение *n* определяет число строк, которые DB2 посыпает клиенту при каждой передаче данных по сети DRDA. Смотрите в разделе “Задание опции OPTIMIZE FOR n ROWS” на стр. 425 более подробные сведения об использовании условия OPTIMIZE FOR n ROWS в распределенной среде.

**Используйте условие OPTIMIZE FOR 1 ROW, чтобы избежать сортировки:** Использование OPTIMIZE FOR 1 ROW в наибольшей степени может повлиять на выбор пути доступа. Это условие говорит DB2, что нужно выбрать путь доступа, который можно быстрее вернет первую отобранныю строку. Это означает, что DB2, когда это возможно, отвергает пути доступа, использующие сортировку. Если указано значение *n*, отличное от 1, DB2 выбирает путь

доступа, исходя из стоимости, при этом не обязательно отказываясь от сортировки.

**Как задать условие *OPTIMIZE FOR n ROWS* для прикладной программы**

**CLI:** Для прикладной программы интерфейса уровня вызовов (CLI) можно указать, чтобы DB2 использовала OPTIMIZE FOR n ROWS для всех запросов. Для этого укажите в файле инициализации ключевое слово OPTIMIZEFORNROWS. Более подробно об этом можно прочитать в Глава 4 *DB2 ODBC Guide and Reference*.

**Сколько строк можно получить, если указано *OPTIMIZE FOR n ROWS*:**

Даже если указано условие OPTIMIZE FOR n ROWS, можно получить все отобранные строки. Однако при этом общее время получения всех строк может быть значительно выше, чем если бы DB2 выполняла оптимизацию в расчете на получение всего набора результатов.

**Когда эффективно условие *OPTIMIZE FOR n ROWS*:** Условие OPTIMIZE FOR n ROWS эффективно только для запросов, которые могут выполняться поэтапно. Если для запроса DB2 должна собрать весь набор результатов перед тем как возвратить первую строку, DB2 игнорирует условие OPTIMIZE FOR n ROWS, как, например, в следующих ситуациях:

- Запрос использует SELECT DISTINCT или функцию со спецификатором DISTINCT, например, COUNT(DISTINCT C1).
- Используется условие GROUP BY или ORDER BY, и нет индекса, который бы задавал нужный порядок.
- Есть функция столбцов, но нет условия GROUP BY.
- В запросе используется UNION.

**Пример:** Предположим, вы регулярно запрашиваете таблицу сотрудников, чтобы определить сотрудников с наибольшими окладами. Можно использовать такой запрос:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY  
      FROM EMPLOYEE  
      ORDER BY SALARY DESC;
```

Для столбца EMPNO определен индекс, поэтому записи сотрудников упорядочены по EMPNO. Если вы также определили убывающий индекс по столбцу SALARY, этот индекс, скорее всего, кластеризован очень плохо. Чтобы избежать большого количества синхронных операций ввода–вывода прямого доступа, DB2 скорее всего просмотрит табличное пространство, затем отсортирует строки по SALARY. При этом может возникнуть задержка перед тем как первые отобранные строки будут возвращены прикладной программе. Если добавить в оператор условие OPTIMIZE FOR n ROWS, как показано ниже:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY  
      FROM EMPLOYEE  
      ORDER BY SALARY DESC  
      OPTIMIZE FOR 20 ROWS;
```

DB2 скорее всего воспользуется непосредственно индексом SALARY, поскольку вы указали, что, возможно, будут получены только записи для 20 наиболее оплачиваемых сотрудников. При этом удастся избежать дорогостоящей операции сортировки.

### **Влияние условия *OPTIMIZE FOR n ROWS*:**

- Может измениться способ объединения. Наиболее вероятный выбор – объединение с помощью вложенных циклов, потому что при этом невысоки затраты и этот способ представляется наиболее эффективным, если планируется получение только одной строки.
- Скорее всего, будет использован индекс, согласованный с условием ORDER BY, потому что при этом для условия ORDER BY не потребуется сортировка.
- Предварительная выборка списка будет использоваться с меньшей вероятностью.
- Последовательная предварительная выборка будет затребована DB2 с меньшей вероятностью, так как DB2 считает, что планируется получить только небольшое число строк.
- В запросе объединения таблица со столбцами из условия ORDER BY, скорее всего, будет выбрана в качестве внешней таблицы, если в этой таблице есть индекс, упорядоченный в соответствии с условием ORDER BY.

**Рекомендация:** И для локальных, и для распределенных запросов указывайте условие OPTIMIZE FOR *n* ROWS только в прикладных программах, которые постоянно выбирают только небольшую часть строк из набора результатов. Например, прикладная программа может читать только столько строк, сколько требуется для заполнения экрана терминала конечного пользователя. В подобных случаях прикладная программа редко читает оставшуюся часть набора результатов запроса. Для такой программы условие OPTIMIZE FOR *n* ROWS может повысить производительность благодаря тому, что:

- Это вынудит DB2 выбирать путь доступа SQL, при котором как можно быстрее отбираются первые *n* строк
- Это ограничивает число строк, передаваемых по сети при каждой передаче данных

Чтобы максимально повлиять на выбор пути доступа, укажите OPTIMIZE FOR 1 ROW. Это значение не оказывает негативного влияния на распределенные запросы. Чтобы увеличить число строк, возвращаемых при одной передаче данных по сети, можете указать большее значение *n*, например, число строк, которые умещаются на экране терминала, что не влияет на выбор пути доступа негативно.

## **Уменьшение числа согласованных с индексом столбцов**

Старайтесь помешать DB2 использовать неудачно работающий индекс, исключив из согласованного с индексом предиката его первый столбец. Посмотрите пример на рис. 167 на стр. 698, где DB2 выбирает далеко не оптимальный индекс.

DB2 выбирает для доступа к данным индекс IX2, хотя IX1 работал бы примерно в 10 раз быстрее. Дело в том, что 50% всех деталей из центра 3 все еще остаются в центре 3. Предположим, что в таблице каталога SYSCOLDIST нет статистики для связанных столбцов. В этом случае DB2 предполагает, что детали из центра 3 равномерно распределены между 50 центрами.

Изменив запрос, можно добиться выбора нужного пути доступа. Чтобы помешать использованию индекса IX2 для данного запроса, можно сделать третий предикат неиндексируемым.

```
SELECT * FROM PART_HISTORY
WHERE
    PART_TYPE = 'BB'
    AND W_FROM = 3
    AND (W_NOW = 3 + 0)      --<-- ПРЕДИКАТ ТЕПЕРЬ НЕ ИНДЕКСИРУЕМ
```

Теперь индекс I2 не выбирается, поскольку в нем только один согласованный столбец. Выбирается нужный нам индекс I1. Третий предикат не индексируем, поэтому для составного предиката индекс не используется.

Сделать предикат неиндексируемым можно многими способами. Рекомендуется следующий способ: если в предикате используется числовое значение, прибавить к нему 0, если используется символьное значение, сделать конкатенацию с пустой строкой.

Индексируемый	Неиндексируемый
T1.C3=T2.C4	(T1.C3=T2.C4 CONCAT ssq;ssq;)
T1.C1=5	T1.C1=5+0

При этом результат запроса не изменится, а затраты возрастут лишь слегка.

Предпочтительный способ улучшения пути доступа, когда в таблице есть связанные столбцы – сгенерировать статистику каталога для связанных столбцов. Это можно сделать, либо запустив RUNSTATS, либо изменив вручную таблицу каталога SYSCOLDIST или SYSCOLDISTSTATS.

```

CREATE TABLE PART_HISTORY (
    PART_TYPE CHAR(2),
    PART_SUFFIX CHAR(10),
    W_NOW INTEGER,
    W_FROM INTEGER,
    DEVIATIONS INTEGER,
    COMMENTS CHAR(254),
    DESCRIPTION CHAR(254),
    DATE1 DATE,
    DATE2 DATE,
    DATE3 DATE);

CREATE UNIQUE INDEX IX1 ON PART_HISTORY
(PART_TYPE,PART_SUFFIX,W_FROM,W_NOW);
CREATE UNIQUE INDEX IX2 ON PART_HISTORY
(W_FROM,W_NOW,DATE1);

```

Статистика для таблицы		Статистика для индекса		IX1	IX2
CARDF	100,000	FIRSTKEYCARDF	1000	50	
NPAGES	10,000	FULLKEYCARDF	100,000	100,000	
		CLUSTERRATIO	99%	99%	
		NLEAF	3000	2000	
		NLEVELS	3	3	
		столбец	мощность	HIGH2KEY	LOW2KEY
		Part_type	1000	'ZZ'	'AA'
		w_now	50	1000	1
		w_from	50	1000	1

Q1:  
SELECT \* FROM PART\_HISTORY -- SELECT ALL PARTS  
WHERE PART\_TYPE = 'BB' P1 -- THAT ARE 'BB' TYPES  
AND W\_FROM = 3 P2 -- THAT WERE MADE IN CENTER 3  
AND W\_NOW = 3 P3 -- AND ARE STILL IN CENTER 3

Показатели фильтрации предикатов.					
P1	= 1/1000= .001				
P2	= 1/50 = .02				
P3	= 1/50 = .02				
ПРЕДПОЛАГАЕМЫЕ ЗНАЧЕНИЯ			РЕАЛЬНЫЕ ЗНАЧЕНИЯ		
показатель			показатель		
индекс	соотв.ст.	фильтрации строк	индекс	соотв.ст.	фильтрации строк
ix2	2	.02*.02	40	ix2	2
ix1	1	.001	100	ix1	1
				.02*.50	1000
				.001	100

Рисунок 167. Уменьшение значения MATCHCOLS

## Добавление дополнительных локальных предикатов

Добавление локальных предикатов со столбцами, которые не используются в других предикатах, может повлиять на запросы объединения следующим образом.

1. Таблица, используемая в дополнительных предикатах, скорее всего, будет выбрана в качестве внешней таблицы. Это объясняется тем, что DB2 считает, что чем больше предикатов ссылаются на таблицу, тем меньше из

нее будет выбрано строк. Обычно в качестве внешней таблицы наиболее эффективно использовать таблицу, из которой отбирается наименьшее число строк.

2. В качестве способа объединения скорее всего будет выбрано объединение с вложенными циклами, потому что этот способ более эффективен для небольшого количества данных, а большее число предикатов заставит DB2 считать, что будет получено меньше данных.

Добавляемые предикаты должны быть вида WHERE TX.CX=TX.CX.

При этом результат запроса не изменится. Это справедливо для столбца любого типа данных и приводит к минимальным издержкам. Однако DB2 использует для каждого столбца только лучший показатель фильтрации. Поэтому если TX.CX уже используется в другом предикате равенства, добавление такого предиката ничего не даст. Нужно добавить лишний локальный предикат для столбца, который еще не используется в других предикатах. Следует также учесть, что если возможен доступ к таблице только по индексу, то не стоит добавлять такой предикат, который помешает DB2 использовать доступ только по индексу.

## Изменение статистики каталога

Имея соответствующие полномочия, можно повлиять на выбор пути доступа, изменив с помощью оператора SQL UPDATE или INSERT статистические данные в каталоге DB2. Однако это рекомендуется делать только в крайнем случае. Даже если изменение статистики каталога может улучшить работу определенного запроса, на других запросах это может оказаться отрицательно. Кроме того, операторы UPDATE придется вызывать снова после того, как RUNSTATS переустановит значения в каталоге. Изменять статистику нужно очень осторожно.

В примере на рис. 167 на стр. 698 есть такой запрос:

```
SELECT * FROM PART_HISTORY -- SELECT ALL PARTS
WHERE PART_TYPE = 'BB'      P1 -- THAT ARE 'BB' TYPES
    AND W_FROM = 3          P2 -- THAT WERE MADE IN CENTER 3
    AND W_NOW = 3           P3 -- AND ARE STILL IN CENTER 3
```

для которого наблюдается проблема с корреляцией данных. DB2 не известно, что 50% деталей, изготовленных в центре 3, все еще остаются в центре 3. Сделав предикат неиндексируемым, мы решили эту проблему. Но предположим, что подобные запросы пишут сотни пользователей. Сделать так, чтобы все пользователи изменили свои запросы, невозможно. В такой ситуации лучше всего изменить статистику каталога.

Для запроса на рис. 167 на стр. 698, где связанные столбцы входят в составной ключ индекса, можно изменить статистику каталога одним из двух способов:

- Запустить утилиту RUNSTATS и затребовать статистику для связанных столбцов W\_FROM и W\_NOW. Этот способ более предпочтителен. Подробнее о поддержке статистики в каталоге рассказывается в разделах Раздел 5 (Том 2) *DB2 Administration Guide* и Раздел 2 *DB2 Utility Guide and Reference*.
- Изменение статистики каталога вручную.

**Изменение каталога, чтобы учесть корреляцию столбцов:** Одна из таблиц каталога, которую можно изменить, – SYSIBM.SYSCOLDIST, которая содержит информацию о первом столбце ключа или столбцах составного ключа индекса. Предположим, что, поскольку столбцы W\_NOW и W\_FROM связаны, есть только 100 различных сочетаний значений двух столбцов, а не 2500 (50 для W\_FROM \* 50 для W\_NOW). Вставьте следующую строку, чтобы задать новое количество значений:

```
INSERT INTO SYSIBM.SYSCOLDIST
  (FREQUENCY, FREQUENCYF, IBMREQD,
   TBOWNER, TBNAME, NAME, COLVALUE,
   TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
VALUES(0, -1, 'N',
      'USRTO01','PART_HISTORY','W_FROM',' ',
      'C',100,X'00040003',2);
```

Поскольку W\_FROM и W\_NOW – столбцы, входящие в составной ключ индекса, эту информацию можно занести в SYSCOLDIST с помощью утилиты RUNSTATS. Смотрите более подробные сведения в книге *DB2 Utility Guide and Reference*.

Сообщить DB2 о частоте определенного сочетания значений столбцов можно, изменив SYSIBM.SYSCOLDIST. Например, можно указать, что в 1% строк в PART\_HISTORY значения W\_FROM и W\_NOW равны 3, вставив в SYSCOLDIST следующую строку:

```
INSERT INTO SYSIBM.SYSCOLDIST
  (FREQUENCY, FREQUENCYF, STATSTIME, IBMREQD,
   TBOWNER, TBNAME, NAME, COLVALUE,
   TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
VALUES(0, .0100, '1996-12-01-12.00.00.000000','N',
      'USRTO01','PART_HISTORY','W_FROM',X'00800000030080000003',
      'F',-1,X'00040003',2);
```

#### **Изменение каталога для объединений с помощью табличных функций:**

Обязательно помните, что изменение статистики каталога **может вызвать серьезные проблемы в работе**, если эти изменения произведены некорректно. Следите за работой и будьте готовы восстановить прежнее состояние статистики, если такие проблемы возникнут.

## Глава 7–4. Использование EXPLAIN для повышения производительности SQL

Информация, начиная с этого заголовка и до конца главы, относится к продуктозависимому интерфейсу программирования и сопутствующей руководящей информации, как оговорено в разделе Приложение I, "Замечания" на стр. 977.

**Определения и назначение:** EXPLAIN – средство текущего контроля, которое дает информацию о:

- Плане, пакете или операторе SQL при их связывании. Данные помещаются в созданную вами таблицу PLAN\_TABLE, которая называется *таблицей планов*. Опытные пользователи могут использовать PLAN\_TABLE, чтобы давать DB2 советы по оптимизации.
- Предполагаемой стоимости операторов SQL SELECT, INSERT, UPDATE или DELETE. Данные помещаются в создаваемую вами таблицу DSN\_STATEMENT\_TABLE, которая называется *таблицей оператора*. Дополнительные сведения о таблицах операторовсмотрите в "Оценка стоимости оператора" на стр. 749.
- Пользовательских функциях, используемых в операторе, включая конкретное имя функции и схему. Данные помещаются в создаваемую вами таблицу с именем DSN\_FUNCTION\_TABLE, которая называется *таблицей функций*. Дополнительные сведения о таблицах функцийсмотрите в разделе "Правильный выбор функции для выполнения DB2" на стр. 318.

**Другие средства:** Отладить запросы SQL могут помочь также следующие средства:

- DB2 Visual Explain

Visual Explain это графическая утилита DB2 для рабочих станций, которая дает:

- Удобное для понимания отображение выбранного пути доступа
- Предложения для изменения оператора SQL
- Возможность вызова EXPLAIN для динамических операторов SQL
- Возможность изменения статистики каталога DB2 для объектов, от которых зависит выбор пути доступа
- Программу просмотра параметров подсистемы с возможностью поиска

Использование DB2 Visual Explain, поставляемого на отдельном CD-ROM по лицензии DB2 Версия 6, описано в *электронной справке Visual Explain*.

- DB2 Performance Monitor (PM)

DB2 PM – средство текущего контроля, предоставляющее пользователям рабочие данные в определенном формате. Информация DB2 PM объединяет информацию из EXPLAIN и из каталога DB2. Он отображает пути доступа, индексы, таблицы, табличные пространства, планы, пакеты, DBRM, определения переменных хоста, порядок, последовательность

доступа к таблицам и объединений, типы блокировок. Данные выводятся не в виде таблицы, а в диалоговом режиме, что облегчает чтение и понимание информации.

- DB2 Estimator

DB2 Estimator for Windows – удобное в работе автономное средство для оценки производительности прикладных программ DB2 for OS/390. С его помощью можно прогнозировать поведение и стоимость прикладных программ, транзакций, операторов SQL и утилит. Например, с помощью DB2 Estimator можно оценить, как скажется добавление или удаление индекса для таблицы, как изменится время ответа после увеличения ресурсов процессора и какое время займет выполнение задания утилиты.

**Обзор тем:** В этой главе рассматриваются следующие темы:

- “Получение информации PLAN\_TABLE с помощью EXPLAIN”
- “Оценка стоимости оператора” на стр. 749
- “Первые вопросы о доступе к данным” на стр. 711
- “Интерпретация доступа к одной таблице” на стр. 721
- “Объяснение доступа к нескольким таблицам” на стр. 728
- “Предварительная выборка данных” на стр. 738
- “Информация о сортировках” на стр. 743
- “Обработка производных таблиц и вложенных табличных выражений” на стр. 745
- “Глава 7–5. Параллельные операции и производительность запросов” на стр. 753

---

## Получение информации PLAN\_TABLE с помощью EXPLAIN

Информация в таблице PLAN\_TABLE поможет:

- Спроектировать базы данных, индексы и прикладные программы
- Определить, когда нужно повторно связать прикладную программу
- Определить, какой путь доступа выбран для запроса

При работе с одной таблицей EXPLAIN сообщает, можно ли использовать индексы или нужно просматривать табличное пространство. Если используются индексы, EXPLAIN сообщает, сколько столбцов индекса и какие методы ввода–вывода используются для чтения страниц. Для объединений таблиц EXPLAIN сообщает способ и тип объединения, порядок доступа к таблицам при объединении, а также когда и почему DB2 сортирует строки.

Основное назначение EXPLAIN – отслеживать пути доступа для подоператоров выбора SELECT в операторах SQL. Для команд UPDATE и DELETE WHERE CURRENT OF, а также для INSERT в таблицу планов помещается меньше информации. Некоторые виды доступа EXPLAIN не описывает: например, доступ к большим объектам, которые хранятся отдельно от базовой таблицы, и обращения к родительским и зависимым таблицам, требуемые для соблюдения реляционных ограничений.

Пути доступа для запросов, приведенных в этой главе в качестве примеров, служат только для иллюстрации этих примеров. Если выполнить эти запросы в реальной системе, могут быть выбраны другие пути доступа.

**Действия для получения информации PLAN\_TABLE:** Краткий перечень действий для получения информации от EXPLAIN таков:

1. Обеспечьте нужный доступ к таблице планов. Как создать таблицу, описано в разделе “Создание PLAN\_TABLE.”
2. Заполните таблицу нужной информацией. Смотрите инструкции в разделе “Заполнение и поддержание таблицы планов” на стр. 709.
3. Выберите нужную информацию из таблицы. Инструкциисмотрите в разделе “Упорядочивание строк таблицы планов” на стр. 710.

## Создание PLAN\_TABLE

Чтобы воспользоваться EXPLAIN, нужно создать таблицу с именем PLAN\_TABLE для сохранения результатов. Оператор для создания таблицы есть в библиотеке примеров DB2 в компоненте DSNTESC. (Если вам не требуется информация из таблиц функций и операторов, создавать эти таблицы для работы с EXPLAIN необязательно.)

На рис. 168 на стр. 704 показан формат таблицы планов. В Табл. 68 на стр. 704 показано содержимое каждого столбца.

Таблица планов допускает много форматов, однако предпочтительнее формат с 49 столбцами, потому что в этом случае вы получаете наиболее полную информацию. При добавлении столбцов к существующей таблице укажите для них NOT NULL WITH DEFAULT, чтобы в существующие строки были записаны значения по умолчанию. Однако, как видно из рис. 168 на стр. 704, некоторые столбцы не допускают пустых значений. Не указывайте для этих столбцов NOT NULL WITH DEFAULT.

QUERYNO	INTEGER	NOT NULL	PREFETCH	CHAR(1)	NOT NULL WITH DEFAULT
QBLOCKNO	SMALLINT	NOT NULL	COLUMN_FN_EVAL	CHAR(1)	NOT NULL WITH DEFAULT
APPLNAME	CHAR(8)	NOT NULL	MIXOPSEQ	SMALLINT	NOT NULL WITH DEFAULT
PROGNAME	CHAR(8)	NOT NULL	-----28-столбцовый формат -----		
PLANNO	SMALLINT	NOT NULL	VERSION	VARCHAR(64)	NOT NULL WITH DEFAULT
METHOD	SMALLINT	NOT NULL	COLLID	CHAR(18)	NOT NULL WITH DEFAULT
CREATOR	CHAR(8)	NOT NULL	-----30-столбцовый формат -----		
TNAME	CHAR(18)	NOT NULL	ACCESS_DEGREE	SMALLINT	
TABNO	SMALLINT	NOT NULL	ACCESS_PGROUP_ID	SMALLINT	
ACCESSTYPE	CHAR(2)	NOT NULL	JOIN_DEGREE	SMALLINT	
MATCHCOLS	SMALLINT	NOT NULL	JOIN_PGROUP_ID	SMALLINT	
ACCESSCREATOR	CHAR(8)	NOT NULL	-----34-столбцовый формат -----		
ACCESSNAME	CHAR(18)	NOT NULL	SORTC_PGROUP_ID	SMALLINT	
INDEXONLY	CHAR(1)	NOT NULL	SORTN_PGROUP_ID	SMALLINT	
SORTN_UNIQ	CHAR(1)	NOT NULL	PARALLELISM_MODE	CHAR(1)	
SORTN_JOIN	CHAR(1)	NOT NULL	MERGE_JOIN_COLS	SMALLINT	
SORTN_ORDERBY	CHAR(1)	NOT NULL	CORRELATION_NAME	CHAR(18)	
SORTN_GROUPBY	CHAR(1)	NOT NULL	PAGE_RANGE	CHAR(1)	NOT NULL WITH DEFAULT
SORTC_UNIQ	CHAR(1)	NOT NULL	JOIN_TYPE	CHAR(1)	NOT NULL WITH DEFAULT
SORTC_JOIN	CHAR(1)	NOT NULL	GROUP_MEMBER	CHAR(8)	NOT NULL WITH DEFAULT
SORTC_ORDERBY	CHAR(1)	NOT NULL	IBM_SERVICE_DATA	VARCHAR(254)	NOT NULL WITH DEFAULT
SORTC_GROUPBY	CHAR(1)	NOT NULL	-----43-столбцовый формат -----		
TSLOCKMODE	CHAR(3)	NOT NULL	WHEN_OPTIMIZE	CHAR(1)	NOT NULL WITH DEFAULT
TIMESTAMP	CHAR(16)	NOT NULL	QBLOCK_TYPE	CHAR(6)	NOT NULL WITH DEFAULT
REMARKS	VARCHAR(254)	NOT NULL	BIND_TIME	TIMESTAMP	NOT NULL WITH DEFAULT
-----25-столбцовый формат -----					
-----46-столбцовый формат -----					
			OPTHINT	CHAR(8)	NOT NULL WITH DEFAULT
			HINT_USED	CHAR(8)	NOT NULL WITH DEFAULT
			PRIMARY_ACESSTYPE	CHAR(1)	NOT NULL WITH DEFAULT
-----49-столбцовый формат -----					

| Рисунок 168. Формат таблицы PLAN\_TABLE

Таблица 68 (Стр. 1 из 6). Описание столбцов в таблице PLAN\_TABLE

Название столбца	Описание
QUERYNO	<p>Число, идентифицирующее объясняемый оператор. Для строки, созданной оператором EXPLAIN, это число можно задать в условии SET QUERYNO. Для строк, созданных другими операторами, его можно задать с помощью условия SET QUERYNO, которое употребляется как необязательное в операторах SELECT, INSERT, UPDATE и DELETE. Если оно не задано, DB2 назначает число на основе номера строки данного оператора SQL в исходной программе.</p> <p>Предикатам FETCH не приписываются индивидуальные QUERYNO.DB2 использует QUERYNO оператора DECLARE CURSOR для всех относящихся к этому указателю операторов FETCH.</p> <p>Когда значение QUERYNO основано на номере оператора в исходной программе, номера свыше 32767 передаются как 0. Поэтому в очень длинной программе это значение может оказаться не уникальным. Если оно не уникально, уникально значение TIMESTAMP.</p>
QBLOCKNO	Положение запроса в объясняемом операторе (1 для самого внешнего запроса, 2 для следующего, и так далее). Для улучшения производительности DB2 может слить один блок запроса с другим. В этом случае номер позиции такого блока запроса не будет указан в QBLOCKNO.
APPLNAME	Имя плана прикладной программы для данной строки. Применяется только ко вложенным операторам EXPLAIN, которые выполняются в плане, или к операторам, выполняемым при связывании плана. Если не применяется, столбец содержит пробел.

Таблица 68 (Стр. 2 из 6). Описание столбцов в таблице PLAN\_TABLE

Название столбца	Описание
PROGNAME	Имя программы или пакета, содержащего объясняемый оператор. Применяется только ко вложенным операторам EXPLAIN и к операторам, выполняемым в результате связывания плана или пакета. Если не применяется, столбец содержит пробел.
PLANNO	Номер шага, на котором был обработан запрос, указанный в столбце QBLOCKNO. Данный столбец отражает порядок выполнения шагов.
METHOD	Число (0, 1, 2, 3 или 4), обозначающее режим объединения на данном шаге: <ul style="list-style-type: none"> <li>0      Обращение к первой таблице, к продолжению предыдущей таблицы, или не используется.</li> <li>1      Объединение со <i>вложенным циклом</i>. Для каждой строки текущей составной таблицы находятся и присоединяются совпадающие строки новой таблицы.</li> <li>2      Объединение с <i>просмотром слияния</i>. Текущая составная таблица и новая таблица просматриваются в порядке объединяемых столбцов, и совпадающие строки объединяются.</li> <li>3      Сортировки, которых требуют ORDER BY, GROUP BY, SELECT DISTINCT, UNION, предикаты с кванторами или предикаты IN. На этом шаге нет обращения к новой таблице.</li> <li>4      <i>Гибридное объединение</i>. Текущая таблица просматривается в порядке строк соединяемых столбцов новой таблицы. Доступ к новой таблице производится с помощью предварительной выборки списка.</li> </ul>
CREATOR	Создатель новой таблицы, к которой осуществляется обращение на данном шаге; столбец содержит пробел, если значение в столбце METHOD = 3.
TNAME	Имя таблицы, временной таблицы, материализованной производной таблицы, табличного выражения или промежуточной таблицы результатов для внешнего объединения, к которой осуществляется обращение на данном шаге; столбец содержит пробел, если значение в столбце METHOD = 3.  Для внешнего объединения данный столбец содержит имя временной таблицы рабочего файла в форме DSNWFQB( <i>qblockno</i> ). Для объединенных производных таблиц выводятся имена базовых таблиц и внутриоператорные имена. Материализованная производная таблица представляет собой другой блок запроса с собственными таблицами, материализованными производными таблицами и так далее.
TABNO	Значения только для IBM.
ACCESTYPE	Метод доступа к новой таблице: <ul style="list-style-type: none"> <li><b>I</b>      По индексу (указанному в ACCESSCREATOR и ACCESSNAME)</li> <li><b>I1</b>     Просмотр индекса за одну выборку</li> <li><b>N</b>      Просмотр индекса, когда совпадающий предикат содержит ключевое слово IN</li> <li><b>R</b>      Просмотр табличного пространства</li> <li><b>M</b>      Просмотр нескольких индексов; дальше следует MX, MI или MU</li> <li><b>MX</b>     Просмотр индекса, указанного в ACCESSNAME</li> <li><b>MI</b>     По пересечению нескольких индексов</li> <li><b>MU</b>     По объединению нескольких индексов</li> </ul> <b>пробел</b> Не применяется к текущей строке.

Таблица 68 (Стр. 3 из 6). Описание столбцов в таблице PLAN\_TABLE

Название столбца	Описание
MATCHCOLS	Для режимов доступа ACESSTYPE I, I1, N или MX – количество ключей индекса, используемых при просмотре индекса; иначе 0.
ACCESSCREATOR	Для режимов доступа ACESSTYPE I, I1, N или MX – создатель индекса; иначе пробел.
ACCESSNAME	Для режимов доступа ACESSTYPE I, I1, N или MX – имя индекса; иначе пробел.
INDEXONLY	Указывает, достаточно ли на данном шаге только доступа к индексу, или требуется также доступ к данным. Y=Да; N=Нет.
SORTN_UNIQ	Сортируется ли новая таблица для устранения одинаковых строк. Y=Да; N=Нет.
SORTN_JOIN	Сортируется ли новая таблица для режимов объединения 2 и 4. Y=Да; N=Нет.
SORTN_ORDERBY	Сортируется ли новая таблица по условию ORDER BY. Y=Да; N=Нет.
SORTN_GROUPBY	Сортируется ли новая таблица по условию GROUP BY. Y=Да; N=Нет.
SORTC_UNIQ	Сортируется ли составная таблица для удаления одинаковых строк. Y=Да; N=Нет.
SORTC_JOIN	Сортируется ли составная таблица для режимов объединения 1, 2 и 4. Y=Да; N=Нет.
SORTC_ORDERBY	Сортируется ли составная таблица по условию ORDER BY или для предиката с кванторами. Y=Да; N=Нет.
SORTC_GROUPBY	Сортируется ли составная таблица по условию GROUP BY. Y=Да; N=Нет.
TSLOCKMODE	Указание на режим блокировки, которая будет установлена для новой таблицы, ее табличного пространства или разделов табличного пространства. Если изоляцию можно определить во время связывания, столбец имеет следующие значения:  IS      Неявная совместно используемая блокировка IX      Неявная монопольная блокировка S      Совместно используемая блокировка U      Блокировка изменения X      Монопольная блокировка SIX     Совместно используемая с неявной монопольной блокировкой N      изоляция UR; блокировки нет  Если изоляцию нельзя определить во время связывания, режим блокировки, определяемый изоляцией во время выполнения, обозначается следующими значениями:  NS      Для изоляции UR блокировки нет; для CS, RS или RR – блокировка S. NIS     Для изоляции UR блокировки нет; для CS, RS или RR – блокировка IS. NSS     Для изоляции UR блокировки нет; для CS и RS – блокировка IS; для RR – блокировка S. SS      Для изоляции UR, CS или RS – блокировка IS; для RR – блокировка S.  Данные в этом столбце выравнены по правому краю. Например, IX выглядит как последовательность из пробела, I и X. Если столбец содержит одни пробелы, блокировка не устанавливается.
TIMESTAMP	Обычно время обработки строки до сотой доли секунды. При необходимости DB2 добавляет к этому значению одну сотую секунды, чтобы различить строки для следующих друг за другом запросов.
REMARKS	Комментарий. В этом поле можно поместить строку любых символов не длиннее 254.

Таблица 68 (Стр. 4 из 6). Описание столбцов в таблице PLAN\_TABLE

Название столбца	Описание
PREFETCH	Будут ли страницы данных считываться заранее с помощью предварительной выборки. S = чисто последовательная выборка; L = выборка через список страниц; пробел = выборка неизвестна или не производится.
COLUMN_FN_EVAL	Когда вычисляется значение функции столбца SQL. R = во время чтения данных из таблицы или индекса; S = во время сортировки по условию GROUP BY; пробел = после извлечения данных и после всех сортировок.
MIXOPSEQ	Порядковый номер шага в операции со многими индексами. 1, 2, ... n      Для шагов в процедуре со многими индексами (ACCESSTYPE = MX, MI или MU.) 0                 Для всех остальных строк (ACCESSTYPE = I, I1, M, N, R или пробел.)
VERSION	Идентификатор версии для пакета. Применяется только ко вложенному оператору EXPLAIN, выполняемому в пакете, или к оператору, объясняемому во время связывания пакета. Если не применяется, столбец содержит пробел.
COLLID	ID собрания для пакета. Применяется только ко вложенному оператору EXPLAIN, выполняемому в пакете, или к оператору, объясняемому во время связывания пакета. Если не применяется, столбец содержит пробел..
<b>Примечание:</b> Следующие 9 столбцов, от ACCESS_DEGREE до CORRELATION_NAME, содержат пустые значения, если план или пакет был связан с помощью таблицы плана, имеющей меньше 43 столбцов. Кроме того, значение каждого из них может быть пустым в случае неприменимости обозначаемого им метода.	
ACCESS_DEGREE	Число параллельных задач или операций, которые активирует запрос. Значение устанавливается во время связывания; реальное количество параллельных операций во время выполнения может быть другим. Столбец содержит 0, если есть переменная хоста.
ACCESS_PGROUP_ID	Идентификатор группы параллельности, обращающейся к новой таблице. Группа параллельности – набор параллельно выполняемых последовательных операций, имеющих одинаковое количество параллельных заданий. Значение устанавливается во время связывания и может измениться во время выполнения.
JOIN_DEGREE	Количество параллельных операций или заданий при объединении составной и новой таблиц. Значение устанавливается во время связывания и равно 0, если есть переменная хоста. Реальное количество параллельных операций или заданий во время выполнения может быть другим.
JOIN_PGROUP_ID	Идентификатор группы параллельности при объединении составной и новой таблиц. Значение устанавливается во время связывания; оно может измениться во время выполнения.
SORTC_PGROUP_ID	Идентификатор группы параллельности для параллельной сортировки составной таблицы.
SORTN_PGROUP_ID	Идентификатор группы параллельности для параллельной сортировки новой таблицы.
PARALLELISM_MODE	Вид параллелизма, если он используется во время связывания: I            Параллелизм ввода/вывода запроса C            Параллелизм запросов СР X            Параллелизм запросов Sysplex
MERGE_JOIN_COLS	Количество столбцов, объединяемых во время объединения с просмотром слиянием (второй метод).

Таблица 68 (Стр. 5 из 6). Описание столбцов в таблице PLAN\_TABLE

Название столбца	Описание																		
CORRELATION_NAME	Внутриоператорное имя таблицы или производной таблицы, указанное в операторе. Если внутриоператорное имя не задано, столбец содержит пробел.																		
PAGE_RANGE	Предназначена ли таблица для экранирования диапазона страниц, чтобы планы не просматривали лишних разделов. Y = Да; пробел = Нет.																		
JOIN_TYPE	<p>Тип внешнего объединения:</p> <p>F      полное внешнее объединение      L      левое внешнее объединение      пробел    внутреннее объединение или без объединения</p> <p>Правое внешнее объединение при использовании преобразуется в левое внешнее объединение, так что столбец JOIN_TYPE получает значение L.</p>																		
GROUP_MEMBER	Имя члена группы DB2, выполнившего оператор EXPLAIN. Столбец содержит пробел, если подсистема DB2 работала не в среде совместного использования данных во время выполнения оператора EXPLAIN.																		
IBM_SERVICE_DATA	Значения только для IBM.																		
WHEN_OPTIMIZE	<p>Когда был определен путь доступа:</p> <p>пробел    Во время связывания с показателем фильтра по умолчанию для всех переменных хоста, маркеров параметров или специальных регистров.</p> <p>B      Во время связывания, с показателем фильтра по умолчанию для всех переменных хоста, маркеров параметров или специальных регистров; но во время выполнения будет произведена повторная оптимизация оператора с использованием значений входных переменных для входных переменных хоста, маркеров параметров или особых регистров. Для повторной оптимизации необходимо указать параметр связывания REOPT(VARS).</p> <p>R      Во время выполнения, с использованием входных переменных для всех переменных хоста, маркеров параметров или специальных регистров. Для этого необходимо указать параметр связывания REOPT(VARS).</p>																		
QBLOCK_TYPE	Для каждого блока запроса, тип выполненной операции SQL. Для самого внешнего запроса указывает на тип оператора. Возможные значения:																		
	<table> <tbody> <tr> <td><b>SELECT</b></td> <td><b>SELECT</b></td> </tr> <tr> <td><b>INSERT</b></td> <td><b>INSERT</b></td> </tr> <tr> <td><b>UPDATE</b></td> <td><b>UPDATE</b></td> </tr> <tr> <td><b>DELETE</b></td> <td><b>DELETE</b></td> </tr> <tr> <td><b>SELUPD</b></td> <td><b>SELECT c FOR UPDATE OF</b></td> </tr> <tr> <td><b>DELCUR</b></td> <td><b>DELETE WHERE CURRENT OF CURSOR</b></td> </tr> <tr> <td><b>UPDCUR</b></td> <td><b>UPDATE WHERE CURRENT OF CURSOR</b></td> </tr> <tr> <td><b>CORSUB</b></td> <td><b>Связанный подзапрос</b></td> </tr> <tr> <td><b>NCOSUB</b></td> <td><b>Несвязанный подзапрос</b></td> </tr> </tbody> </table>	<b>SELECT</b>	<b>SELECT</b>	<b>INSERT</b>	<b>INSERT</b>	<b>UPDATE</b>	<b>UPDATE</b>	<b>DELETE</b>	<b>DELETE</b>	<b>SELUPD</b>	<b>SELECT c FOR UPDATE OF</b>	<b>DELCUR</b>	<b>DELETE WHERE CURRENT OF CURSOR</b>	<b>UPDCUR</b>	<b>UPDATE WHERE CURRENT OF CURSOR</b>	<b>CORSUB</b>	<b>Связанный подзапрос</b>	<b>NCOSUB</b>	<b>Несвязанный подзапрос</b>
<b>SELECT</b>	<b>SELECT</b>																		
<b>INSERT</b>	<b>INSERT</b>																		
<b>UPDATE</b>	<b>UPDATE</b>																		
<b>DELETE</b>	<b>DELETE</b>																		
<b>SELUPD</b>	<b>SELECT c FOR UPDATE OF</b>																		
<b>DELCUR</b>	<b>DELETE WHERE CURRENT OF CURSOR</b>																		
<b>UPDCUR</b>	<b>UPDATE WHERE CURRENT OF CURSOR</b>																		
<b>CORSUB</b>	<b>Связанный подзапрос</b>																		
<b>NCOSUB</b>	<b>Несвязанный подзапрос</b>																		
BIND_TIME	Время, когда был связан план или пакет данного оператора или блока запроса. Для статических операторов SQL это максимально точная отметка времени. Для динамических операторов SQL это значение из поля TIMESTAMP таблицы PLAN_TABLE, дополненное четырьмя нулями.																		
OPTHINT	Строка, используемая для идентификации этой строки как подсказки для DB2. DB2 использует эту строку как входные данные при выборе пути доступа.																		
HINT_USED	Использовав одну из подсказок, DB2 помещает ее идентификатор (значение столбца OPTHINT) в этот столбец.																		

Таблица 68 (Стр. 6 из 6). Описание столбцов в таблице PLAN\_TABLE

Название столбца	Описание
PRIMARY_ACESSTYPE	Указывает, будет ли сначала предприниматься попытка прямого доступа к строке:  D DB2 попробует прямо обратиться к строке. Если DB2 не может использовать прямой доступ к строке во время выполнения, будет использован путь доступа, описанный в столбце ACESSTYPE таблицы PLAN_TABLE.  пробел DB2 не будет пытаться прямо обратиться к строке.

## Заполнение и поддержание таблицы планов

Описание двух разных способов заполнения таблицы планов смотрите в разделах:

- “Выполнение оператора SQL EXPLAIN”
- “Связывание с опцией EXPLAIN(YES)”

При заполнении таблицы планов с помощью EXPLAIN триггеры INSERT для таблицы не активируются. Они активируются только когда вы сами вставляете строки.

Советы по поддержанию растущей таблицы планов смотрите в разделе “Поддержание таблицы планов” на стр. 710.

### Выполнение оператора SQL EXPLAIN

Таблицу PLAN\_TABLE можно заполнить с помощью оператора SQL EXPLAIN. В условии FOR укажите один оператор SQL, для которого вы хотите получить объяснения.

Оператор EXPLAIN можно вызвать либо статически из прикладной программы, либо динамически с помощью QMF или SPUFI. Инструкции и подробности авторизации, требуемые для PLAN\_TABLE, смотрите в справочнике *DB2 SQL Reference*.

### Связывание с опцией EXPLAIN(YES)

Вы можете заполнить таблицу планов при связывании или повторном связывании плана или пакета. Укажите опцию EXPLAIN(YES). EXPLAIN получает информацию о путях доступа для всех объясняемых операторов SQL в пакете или модуле DBRM плана. Информация помещается в таблицу владелец\_пакета.PLAN\_TABLE или владелец\_плана.PLAN\_TABLE. Для динамически подготавливаемого SQL спецификатор PLAN\_TABLE – это текущий SQLID.

**О производительности:** Опция связывания EXPLAIN не влияет на производительность. Выбор пути доступа производится одинаково, независимо от задания EXPLAIN(YES) или EXPLAIN (NO). Указание EXPLAIN(YES) приводит лишь к небольшим издержкам, связанным с записью результатов в таблицу планов.

Если план или таблица, которые были ранее связаны с опцией EXPLAIN(YES), автоматически повторно связываются, запуск EXPLAIN зависит от значения поля EXPLAIN PROCESSING на панели установки DSNTIPO.

**EXPLAIN при удаленном связывании:** Удаленный реквестер, обращающийся к DB2, может указать опцию EXPLAIN(YES) при связывании пакета на сервере DB2. Информация записывается в таблицу планов на сервере, а не на реквестере. Если реквестер не поддерживает распространение опции EXPLAIN(YES), повторно свяжите пакет на реквестере с этой опцией, чтобы получить информацию о путях доступа. Нельзя получить информацию о путях доступа для операторов SQL, использующих собственный протокол.

### Поддержание таблицы планов

DB2 добавляет строки к PLAN\_TABLE, когда вы запускаете утилиту; она не удаляет строки автоматически. Чтобы удалить из таблицы устаревшие строки, используйте команду DELETE, как при работе с любой другой таблицей. Можно также использовать команду DROP TABLE, чтобы отбросить таблицу планов целиком.

## Упорядочивание строк таблицы планов

Вставлять строки в таблицу планов могут несколько процессов. Чтобы определить пути доступа, необходимо получить строки для определенного запроса в нужном порядке.

### Получение строк, относящихся к плану

Строки, относящиеся к определенному плану, определяются значением APPLNAME. Приведенный ниже запрос к таблице планов возвращает строки для всех объясняемых операторов плана в их логической последовательности:

```
SELECT * FROM JOE.PLAN_TABLE  
WHERE APPLNAME = 'APPL1'  
ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

После сортировки по условию ORDER BY можно увидеть, есть ли в таблице:

- Несколько QBLOCKNO для одного QUERYNO
- Несколько PLANNO для одного QBLOCKNO
- Несколько MIXOPSEQ для одного PLANNO

Все строки с одинаковыми непустыми значениями QBLOCKNO и одинаковыми значениями QUERYNO относятся к определенному этапу запроса. Блоки запросов (QBLOCKNO) не обязательно выполняются в том порядке, в котором они показаны в PLAN\_TABLE. Однако внутри одного QBLOCKNO подэтапы в столбце PLANNO показываются в порядке их выполнения.

Для каждого подэтапа столбец TNAME определяет таблицу, к которой производится доступ. Сортировки могут быть показаны как часть обращения к таблице или как отдельный этап.

**Когда QUERYNO=0?** Если в тексте программы больше 32767 строк, все значения QUERYNO, большие 32767, показываются как 0. Для записей с QUERYNO=0 можно различать разные операторы по отметке времени, которая всегда уникальна.

## Получение строк для пакета

Строки, относящиеся к определенному пакету, определяются значениями PROGNAME, COLLID и VERSION. Эти столбцы соответствуют элементам четырехчастного имени пакета:

LOCATION.COLLECTION.PACKAGE\_ID.VERSION

COLLID представляет собой имя СОБРАНИЯ, а PROGNAME – PACKAGE\_ID. Приведенный ниже запрос к таблице планов возвращает строки для всех объясняемых операторов пакета в их логической последовательности:

```
SELECT * FROM JOE.PLAN_TABLE
  WHERE PROGNAME = 'PACK1' AND COLLID = 'COLL1' AND VERSION = 'PROD1'
    ORDER BY QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

---

## Первые вопросы о доступе к данным

При изучении результатов EXPLAIN постарайтесь ответить на следующие вопросы:

- “Используется ли для доступа индекс? (ACCESSTYPE – I, I1, N или MX)” на стр. 712
- “Используется ли для доступа несколько индексов? (ACCESSTYPE=M)” на стр. 712
- “Сколько столбцов индекса используется при согласованном просмотре? (MATCHCOLS=n)” на стр. 713
- “Можно ли выполнить запрос, используя только индекс? (INDEXONLY=Y)” на стр. 714
- “Возможен ли прямой доступ к строке? (PRIMARY\_ACCESTYPE = D)” на стр. 714
- “Материализуются ли производная таблица или вложенное табличное выражение?” на стр. 718
- “Ограничивается ли просмотр определенными разделами? (PAGE\_RANGE=Y)” на стр. 718
- “Какие виды предварительной выборки используются? (PREFETCH = L, S или пусто)” на стр. 719
- “Используется ли параллельная обработка данных? (PARALLELISM\_MODE – I, C или X)” на стр. 720
- “Применяется ли сортировка?” на стр. 720
- “Преобразуется ли подзапрос в операцию объединения?” на стр. 721
- “Когда выполняются функции столбцов? (COLUMN\_FN\_EVAL)” на стр. 721

В этом разделе рассказывается, как ответить на эти вопросы, зная значения столбцов таблицы планов.

## Используется ли для доступа индекс? (ACCESSTYPE – I, I1, N или MX)

Если в столбце ACCESSTYPE таблицы планов стоит одно из этих значений, DB2 использует для доступа к таблице с именем в столбце TNAME индекс. Индекс определяют столбцы ACCESSCREATOR и ACCESSNAME. Описание способов работы с индексамисмотрите в разделе “Индексный путь доступа” на стр. 723.

## Используется ли для доступа несколько индексов? (ACCESSTYPE=M)

Это значение указывает на то, что DB2 использует для доступа к одной таблице набор индексов. Строки таблицы планов содержат информацию о многоиндексном доступе. Столбец MIXOPSEQ содержит номера строк в порядке выполнения этапов многоиндексного доступа. (Если полученные строки упорядочены по MIXOPSEQ, результат аналогичен постфиксной арифметической нотации.)

В примерах, приведенных на рис. 169 и рис. 170 на стр. 713, есть индексы IX1 по T(C1) и IX2 по T(C2). В процессе выполнения запроса DB2 можно выделить следующие этапы:

1. Получение всех идентификаторов записи (RID), для которых C1=1, с использованием индекса IX1.
2. Получение списка RID всех записей, удовлетворяющих условию C2=1, с использованием индекса IX2. Пересечение списков RID и будет окончательным набором RID.
3. Обращение к страницам данных для получения отобранных строк по окончательному списку RID.

```
SELECT * FROM T  
WHERE C1 = 1 AND C2 = 1;
```

TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	PREFETCH	MIXOP-SEQ
T	M	0		Н	L	0
T	MX	1	IX1	Д		1
T	MX	1	IX2	Д		2
T	MI	0		Н		3

Рисунок 169. Данные PLAN\_TABLE для примера с операцией пересечения (AND)

Один и тот же индекс при многоиндексном доступе может просматриваться несколько раз, потому что с индексом может быть согласовано несколько предикатов, как на рис. 170 на стр. 713.

```

SELECT * FROM T
  WHERE C1 BETWEEN 100 AND 199 OR
        C1 BETWEEN 500 AND 599;

```

TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	PREFETCH	MIXOP-SEQ
T	M	0		Н	L	0
T	MX	1	IX1	Д		1
T	MX	1	IX1	Д		2
T	MU	0		Н		3

Рисунок 170. Данные PLAN\_TABLE для примера с операцией объединения (OR)

Этапы обработки:

1. Получение всех RID, для которых C1 между 100 и 199, с использованием индекса IX1.
2. Получение всех RID, для которых C1 между 500 и 599, с использованием того же индекса IX1. Объединение списков RID дает окончательный список RID.
3. Получение отобранных строк с использованием окончательного списка RID.

## Сколько столбцов индекса используется при согласованном просмотре? (MATCHCOLS=n)

Если MATCHCOLS равно 0, способ доступа называется *несогласованным просмотром индекса*. Читаются все ключи индекса и их RID.

Если MATCHCOLS больше 0, способ доступа называется *согласованным просмотром индекса*: в запросе используются предикаты, согласованные со столбцами индекса.

В общем случае согласованные предикаты с ведущими столбцами индекса – это предикаты равенства или включения (IN). Предикат, согласованный с последним столбцом индекса, может быть предикатом равенства, включения (IN) или диапазона (<, <=, >, >=, LIKE или BETWEEN).

Ниже приводится пример запроса, содержащего согласованные с индексом предикаты:

```

SELECT * FROM EMP
  WHERE JOBCODE = '5' AND SALARY > 60000 AND LOCATION = 'CA';
INDEX XEMP5 on (JOBCODE, LOCATION, SALARY, AGE);

```

В качестве пути доступа для этого запроса выбран индекс XEMP5 с MATCHCOLS = 3. Здесь два предиката равенства с первыми двумя столбцами индекса и предикат диапазона с третьим столбцом. Хотя в индексе четыре столбца, согласованными с индексом могут быть только три из них.

## **Можно ли выполнить запрос, используя только индекс? (INDEXONLY=Y)**

В этом случае способ доступа называется *доступом только через индекс*. При операции SELECT все столбцы, используемые в запросе, можно найти в индексе, и DB2 не обращается к таблице. При операции UPDATE или DELETE для чтения выбранной строки требуется только индекс.

Доступ к данным только по индексу невозможен для этапов, в которых используется предварительная выборка списка (описание в разделе “Какие виды предварительной выборки используются? (PREFETCH = L, S или пусто)” на стр. 719). Доступ только по индексу невозможен, когда в наборе результатов возвращаются данные переменной длины или в предикате LIKE используется столбец типа VARCHAR, если только на панели установки DSNTIP4 в поле VARCHAR FROM INDEX не было задано YES, а план или пакеты после этого были повторно связаны. Смотрите дополнительные сведения в разделе *Раздел 2 DB2 Installation Guide*.

При доступе по нескольким индексам для этапа с типом доступа MX поле INDEXONLY содержит Y, так как обращение к страницам данных происходит только после завершения всех операций пересечения (MI) и объединения (MU).

Когда программа SQL использует доступ только через индекс к столбцу ROWID, программа делает заявку на табличное пространство или на раздел табличного пространства. В результате может произойти конфликт между этой программой SQL и утилитой, которая бронирует это табличное пространство или раздел. Доступ только через индекс к таблице со столбцом ROWID невозможен, если соответствующее табличное пространство или раздел находится в несовместимом состоянии. Например, программа SQL может сделать заявку на чтение табличного пространства, только если состояние этого табличного пространства допускает чтение.

## **Возможен ли прямой доступ к строке? (PRIMARY\_ACCESTYPE = D)**

Если прикладная программа выбирает из таблицы строку, содержащую столбец ROWID, значение ID строки неявно указывает на положение строки. Если это значение ID строки используется в условии поиска в последующих операциях SELECT, DELETE или UPDATE, DB2 сможет сразу перейти к строке. Этот способ доступа называется *прямым доступом к строке*.

Прямой доступ к строке работает очень быстро, так как DB2 не нужно просматривать индекс или табличное пространство, чтобы найти строку. Он может использоваться для таблиц, содержащих столбец ROWID.

Чтобы использовать прямой доступ к строке, сначала нужно поместить значения строки в переменные хоста. Значение из столбца ROWID содержит положение строки. Затем, при выполнении запросов, обращающихся к этой строке, включите значение ID строки в условие поиска. Если DB2 определит, что можно использовать прямой доступ к строке, она обращается непосредственно к строке, используя ее ID. Смотрите пример запроса в разделе “Пример: Использование в программе ID строк для прямого доступа к строке” на стр. 717.

## **Для каких предикатов может быть использован прямой доступ к строке?**

Чтобы для запроса мог использоваться прямой доступ к строке, условие поиска должно быть предикатом – логическим термом 1 этапа, для которого верно одно из следующих утверждений:

1. Это простой предикат – логический терм вида **COL=нестолбцовое выражение**, где COL – типа ROWID и **нестолбцовое выражение** содержит ID строки
2. Это простой предикат – логический терм вида **COL IN список**, где COL – типа ROWID, список состоит из ID строк, и для COL определен индекс
3. Это составной логический терм, составленный из нескольких простых предикатов с помощью операции AND, и один из простых предикатов подходит под описание 1 или 2

Однако если для запроса подходит прямой доступ к строке, это не означает, что для него всегда выбирается этот тип доступа. Если DB2 находит лучший путь доступа, прямой доступ к строке не используется.

**Примеры:** В следующем примере ID – столбец ROWID в таблице T1. Для этого столбца существует уникальный индекс. Переменные хоста – типа ROWID.

```
WHERE ID IN (:hv_rowid1,:hv_rowid2,:hv_rowid3)
```

Следующий предикат также подходит для прямого доступа к строке:

```
WHERE ID = ROWID(X'F0DFD230E3C0D80D81C201AA0A280100000000000203')
```

**Поиск распространяемых строк:** Если строки переносятся из одной таблицы в другую, нельзя использовать ID строки из входной таблицы для поиска такой же строки в выходной таблице или наоборот. Прямой доступ к строке в данном случае в качестве пути доступа использовать нельзя. Например, допустим, что в приведенном ниже примере переменная хоста содержит ID строки из таблицы SOURCE:

```
SELECT * FROM TARGET  
WHERE ID = :hv_rowid
```

Поскольку положение строки выходной таблице с тем же ID строки не такое, как во входной таблице, DB2 скорее всего не найдет эту строку. Чтобы получить нужную строку, примените поиск по другому столбцу.

## **Использование пути доступа из ACCESSTYPE**

Если даже DB2 планирует использовать прямой доступ к строке, обстоятельства могут заставить DB2 отказаться от него во время выполнения. DB2 запоминает положение строки после обращения к ней. Однако после этого положение этой строки может поменяться (например, после REORG), и DB2 не сможет при следующем обращении использовать для поиска строки прямой доступ к строке. Вместо этого DB2 использует путь доступа, который указан в столбце ACCESSTYPE таблицы PLAN\_TABLE.

Если предикат, для которого планировался прямой доступ к строке, неиндексируемый, и DB2 не может использовать этот способ доступа, она применяет для поиска строки просмотр табличного пространства. Это может сильно повлиять на производительность прикладных программ, которые

рассчитывают на прямой доступ к строке. При написании прикладных программ следует учитывать возможность того, что прямой доступ к строке будет невозможен. Можно, например:

- Сделать, чтобы прикладная программа не запоминала столбцы ROWID при реорганизациях табличного пространства.

Когда программа выполняет принятие, она освобождает табличное пространство; в этот момент может быть запущена утилита REORG, которая переместит строку, что сделает прямой доступ к строке невозможным. Поэтому при написании программы следует учитывать принятие – использовать сохраненный ID строки только до принятия или после принятия выбирать ID строки заново.

Если вы сохраняете столбцы ROWID из другой таблицы, необходимо исправлять их значения после реорганизации этой таблицы.

- Создайте индекс для столбца ROWID, чтобы DB2 могла использовать индекс, если прямой доступ невозможен.
- Добавьте к предикату со столбцом ROWID другой предикат, который позволяет DB2 использовать существующий индекс таблицы. Например, после чтения строки прикладная программа может выполнить такое изменение:

```
EXEC SQL UPDATE EMP  
SET SALARY = :hv_salary + 1200  
WHERE EMP_ROWID = :hv_emp_rowid  
AND EMPNO = :hv_emprno;
```

Если для EMPNO существует индекс, DB2 сможет использовать индексный доступ, когда прямой доступ к строке невозможен. Дополнительный предикат гарантирует, что DB2 не выберет просмотр табличного пространства.

## Прямой доступ к строке и другие способы доступа

**Параллелизм:** Прямой доступ к строке и параллельная обработка несовместимы друг с другом. Если для запроса подходит и прямой доступ к строке, и параллельная обработка, используется первый способ. Если прямой доступ к строке невозможен, DB2 не выбирает параллельную обработку; вместо этого она использует резервный тип доступа (указанный в столбце ACSESSTYPE таблицы PLAN\_TABLE). В результате она может выбрать просмотр табличного пространства. Чтобы избежать этого, добавьте к предикату индексируемый столбец.

**Обработка списков RID:** Прямой доступ к строке и обработка списков RID взаимно исключают друг друга. Если для запроса подходит и прямой доступ к строке, и обработка списков RID, используется первый способ доступа. Если же его невозможно использовать, DB2 не использует обработку списков RID; вместо этого она выбирает резервный тип доступа.

## Пример: Использование в программе ID строк для прямого доступа к строке

Фрагмент программы на С, демонстрирующий, как получить значение ID строки и затем использовать его для поиска строки, которую нужно изменить.

```
*****  
/* Объявление переменных хоста */  
*****  
EXEC SQL BEGIN DECLARE SECTION;  
    SQL TYPE IS BLOB_LOCATOR hv_picture;  
    SQL TYPE IS CLOB_LOCATOR hv_resume;  
    SQL TYPE IS ROWID hv_emp_rowid;  
    short hv_dept, hv_id;  
    char hv_name[30];  
    decimal hv_salary[5,2];  
EXEC SQL END DECLARE SECTION;  
  
*****  
/* Возвращаем изображение и сводку из таблицы PIC_RES */  
*****  
strcpy(hv_name, "Jones");  
EXEC SQL SELECT PR.PICTURE, PR.RESUME INTO :hv_picture, :hv_resume  
    FROM PIC_RES PR  
    WHERE PR.Name = :hv_name;  
  
*****  
/* Вставляем в таблицу EMPDATA, строку, которая содержит */  
/* изображение и сводку, полученные из таблицы PIC_RES */  
*****  
EXEC SQL INSERT INTO EMPDATA  
    VALUES (DEFAULT,9999,'Jones', 35000.00, 99,  
    :hv_picture, :hv_resume);  
  
*****  
/* Теперь получаем некоторую информацию об этой строке, */  
/* в том числе значение ROWID. */  
*****  
hv_dept = 99;  
EXEC SQL SELECT E.SALARY, E.EMP_ROWID  
    INTO :hv_salary, :hv_emp_rowid  
    FROM EMPDATA E  
    WHERE E.DEPTNUM = :hv_dept AND E.NAME = :hv_name;
```

Рисунок 171 (Часть 1 из 2). Пример использования ID строки для прямого доступа к строке

```

/*
* Изменяем столбцы SALARY, PICTURE и RESUME. Используем *
* значение ROWID, полученное предыдущим оператором, для *
* обращения к строке, которую надо изменить.           */
/* smiley_face и update_resume - пользовательские          */
/* функции, которые здесь не показаны.                   */
/*
EXEC SQL UPDATE EMPDATA
    SET SALARY = :hv_salary + 1200,
        PICTURE = smiley_face(:hv_picture),
        RESUME = update_resume(:hv_resume)
    WHERE EMP_ROWID = :hv_emp_rowid;

/*
* Используем значение ROWID, чтобы получить ID           */
/* сотрудника из той же самой записи.                   */
/*
EXEC SQL SELECT E.ID INTO :hv_id
    FROM EMPDATA E
    WHERE E.EMP_ROWID = :hv_emp_rowid;

/*
* Используем значение ROWID, чтобы удалить запись о      */
/* сотруднике из таблицы.                                */
/*
EXEC SQL DELETE FROM EMPDATA
    WHERE EMP_ROWID = :hv_emp_rowid;

```

*Рисунок 171 (Часть 2 из 2). Пример использования ID строки для прямого доступа к строке*

## Материализуются ли производная таблица или вложенное табличное выражение?

Если столбец TNAME содержит имя производной таблицы или вложенного табличного выражения, (NTE), это означает, что производная таблица или вложенное табличное выражение материализованы. *Материализация* означает, что строки производной таблицы или NTE помещены в рабочий файл, чтобы с ними можно было работать, как с таблицей. Если производная таблица или вложенное табличное выражение материализуются, этот этап заносится в таблицу планов с собственным значением QBLOCKNO и именем производной таблицы или вложенного табличного выражения в TNAME. Если DB2 может работать с производной таблицей или вложенным табличным выражением, обращаясь только к базовой таблице, в столбец TNAME не заносится имя производной таблицы или вложенного табличного выражения. (Более подробное описание материализациисмотрите в разделе “Обработка производных таблиц и вложенных табличных выражений” на стр. 745.)

## Ограничиваются ли просмотр определенными разделами? (PAGE\_RANGE=Y)

DB2 может ограничиться при просмотре данных в многораздельной таблице одним или несколькими разделами. Такой способ называется *ограниченным просмотром разделов*. Запрос должен содержать предикат с первым столбцом индекса разделения. Только первый столбец ключа значим для ограничения набора разделов для просмотра.

Ограниченный просмотр разделов может сочетаться с другими способами доступа. Например, рассмотрим такой запрос:

```
SELECT .. FROM T  
  WHERE (C1 BETWEEN '2002' AND '3280'  
        OR C1 BETWEEN '6000' AND '8000')  
        AND C2 = '6';
```

Предположим, что у таблицы Т есть индекс разделения по столбцу С1, и все значения С1 от 2002 до 3280 находятся в разделах 3 и 4, а значения от 6000 до 8000 находятся в разделах 8 и 9. Предположим также, что у Т есть другой индекс по столбцу С2. DB2 может выбрать один из следующих способов доступа:

- Согласованный просмотр индекса по столбцу С1. При этом данные читаются только из разделов 3, 4, 8 и 9. (PAGE\_RANGE=N)
- Согласованный просмотр индекса по столбцу С2. (DB2 может выбрать его, если число строк, для которых С2=6, невелико.) При этом из индекса по С2 читаются все RID, для которых С2=6, и соответствующие страницы данных из разделов 3, 4, 8 и 9. (PAGE\_RANGE=Y)
- Сканирование табличного пространства по Т. DB2 при этом будет читать страницы данных только из разделов 3, 4, 8 и 9. (PAGE\_RANGE=Y)

**Объединения:** Для каждой таблицы, используемой в запросе, при объединении может применяться ограниченный просмотр разделов.

**Ограничения:** Ограниченный просмотр разделов не применяется, когда в первом ключе первичного индекса используются переменные хоста или маркеры параметров. Причина в том, что набор разделов для просмотра неизвестен во время связывания. Если вы считаете, что выгодно использовать ограниченный просмотр разделов, но у вас есть переменные хоста или маркеры параметров, используйте опцию связывания REOPT(VARS).

Если у вас есть предикаты с операцией OR, и один из предикатов ссылается на столбец в индексе разделения, не являющийся первым столбцом ключа, DB2 не использует ограниченный просмотр разделов.

## Какие виды предварительной выборки используются? (PREFETCH = L, S или пусто)

Предварительная выборка – способ, при котором заранее известно, что будет использоваться определенный набор страниц данных, и весь набор читается в буфер с помощью одной асинхронной операции ввода–вывода. Если значение PREFETCH:

- S, этот способ называется *последовательной предварительной выборкой*. Заранеечитываются страницы данных, расположенные последовательно. При просмотре табличного пространства всегда используется последовательная предварительная выборка. При просмотре индекса она может не использоваться. Более полное описание смотрите в разделе “Последовательная предварительная выборка (PREFETCH=S)” на стр. 738.
- L, такой способ называется *предварительной выборкой списка*. Один или несколько индексов используются для выбора RID, чтобы получить список страниц данных, которые будут прочитаны заранее; страницы не обязательно располагаются последовательно. Обычно RID сортируются.

Исключение – гибридное объединение (описывается в разделе “Гибридное объединение (METHOD=4)” на стр. 736), когда значение столбца SORTN\_JOIN равно N. Более полное описание смотрите в разделе “Предварительная выборка списка (PREFETCH=L)” на стр. 739.

- Пустое, предварительная выборка не используется в качестве способа доступа. Однако в зависимости от характера доступа к страницам данные могут считываться заранее при помощи процесса, называемого *последовательным обнаружением*. Описание смотрите в разделе “Последовательное обнаружение во время выполнения” на стр. 741.

## Используется ли параллельная обработка данных? (PARALLELISM\_MODE – I, C или X)

Параллельная обработка используется только для запросов, допускающих только чтение.

Если режим:	DB2 планирует использовать:
I	Параллельные операции ввода–вывода
C	Параллельная процессорная обработка
X	Параллелизм запросов Sysplex

Непустые значения в столбцах ACCESS\_DEGREE и JOIN\_DEGREE задают для DB2 степень параллелизма. Однако реально DB2 может не использовать параллельную обработку, или использовать ее в меньшей степени, чем было запланировано. Более полное описание смотрите в разделе “Глава 7–5.

Параллельные операции и производительность запросов” на стр. 753.  
Дополнительную информацию об этом методе смотрите в разделе Глава 7 *DB2 Data Sharing: Planning and Administration*.

## Применяется ли сортировка?

**SORTN\_JOIN и SORTC\_JOIN:** SORTN\_JOIN означает, что новая таблица объединения сортируется перед объединением. (Для гибридного объединения это сортировка списка RID.) Если и SORTN\_JOIN, и SORTC\_JOIN содержат 'Y', при объединении выполняются две сортировки. Сортировки для объединения указываются в строке, описывающей доступ к новой таблице.

**Сортировки METHOD 3:** Используются в условиях ORDER BY, GROUP BY, SELECT DISTINCT, UNION или предикатах с кванторами. (Предикаты с кванторами – это предикаты вида 'столбец = ANY (подвыбор)' или 'столбец = SOME (подвыбор)'). Они указываются в отдельной строке. В одной строке таблицы планов могут быть указаны две сортировки составной таблицы, однако реально выполняется только одна сортировка.

**SORTC\_UNIQ и SORTC\_ORDERBY:** SORTC\_UNIQ указывает на сортировку для удаления повторяющихся значений, которая может потребоваться для оператора SELECT со спецификаторами DISTINCT или UNION. SORTC\_ORDERBY обычно указывает на сортировку для условия ORDER BY. Но SORTC\_UNIQ и SORTC\_ORDERBY также могут указывать на сортировку результатов несвязанного подзапроса, как для удаления повторяющихся значений, так и для упорядочивания.

## Преобразуется ли подзапрос в операцию объединения?

Для лучшего выбора пути доступа DB2 иногда преобразует подзапросы в операции объединения, как описывается в разделе “Преобразование подзапроса в операцию объединения” на стр. 690. В таблице планов на то, что подзапрос преобразуется в операцию объединения, указывает значение в столбце QBLOCKNO.

- Если подзапрос не преобразуется в объединение, это означает, что он выполняется как отдельная операция, и его значение QBLOCKNO больше того же значения для внешнего запроса.
- Если подзапрос преобразуется в объединение, у него и у внешнего запроса одинаковые значения QBLOCKNO. На объединение также указывают значения 1, 2 или 4 в столбце METHOD.

## Когда выполняются функции столбцов? (COLUMN\_FN\_EVAL)

То, когда выполняются функции столбцов, зависит от пути доступа, выбранного для оператора SQL.

- Если столбец ACESSTYPE содержит I1, функции MAX и MIN могут быть выполнены путем обращения к индексу, указанному в ACCESSNAME.
- Для других значений ACESSTYPE момент выполнения функций столбца указывается в столбце COLUMN\_FN\_EVAL.

Значение	Функции выполняются ...
S	При сортировке для условия GROUP BY
R	При чтении данных из таблицы или индекса
пробел	После получения данных и всех сортировок

В общем случае значения R или S лучше для производительности, чем пустое значение.

**Будьте осторожны при использовании функций дисперсии и стандартного отклонения:** Функции VARIANCE и STTDEV всегда выполняются на последнем этапе (то есть COLUMN\_FN\_EVAL для них пусто). Это приводит к тому, что другие функции в блоке запроса также выполняются на последнем этапе. Например, в следующем запросе функция sum выполняется позже, чем если бы не было функции дисперсии:

```
SELECT SUM(C1), VARIANCE(C1) FROM T1;
```

## Интерпретация доступа к одной таблице

В последующих разделах описываются различные пути доступа, которые могут указываться в таблице планов, а также предложения по улучшению путей доступа для выбора DB2. Описаны следующие темы:

- Просмотр табличных пространств (ACESSTYPE=R PREFETCH=S)
- “Индексный путь доступа” на стр. 723
- “UPDATE с использованием индекса” на стр. 728

## Просмотр табличных пространств (**ACCESSTYPE=R PREFETCH=S**)

Просмотр табличного пространства чаще всего используется по следующим причинам:

- Производится доступ через временную таблицу. (Доступ через индекс для временных таблиц невозможен.)
- Невозможен согласованный просмотр индекса, поскольку индекс недоступен или нет подходящих предикатов.
- Высока доля возвращаемых строк таблицы. В этом случае использование индекса не дает преимущества, так как большинство строк приходится читать и так.
- У индексов, для которых есть согласованные предикаты, низкое отношение кластеризации, и поэтому их использование эффективно только для данных небольшого объема.

Предположим, что у таблицы T нет индекса по C1. Вот пример запроса, для которого используется просмотр табличного пространства:

```
SELECT * FROM T WHERE C1 = VALUE;
```

В данном случае надо проверить каждую строку T, чтобы установить, равно ли значение C1 указанному в запросе значению.

### Просмотр несегментированных табличных пространств

DB2 читает и проверяет каждую страницу табличного пространства, независимо от того, какой таблице принадлежит страница. Она может также читать страницы, не содержащие данных или содержащие данные, которые были удалены.

### Просмотр сегментированных табличных пространств

Если табличное пространство сегментировано, DB2 сначала определяет, какие сегменты нужно читать. Затем она читает только сегменты табличного пространства, содержащие строки T. Если объем данных, считываемых при предварительной выборке, который определяется размером пула буферов, больше SEGSIZE, и если сегменты T не расположены непрерывно, DB2 может считывать ненужные страницы. Используйте как можно большее значение SEGSIZE, подходящее для конкретного размера данных. Большое значение SEGSIZE лучше всего подходит для поддержания кластеризации строк данных. Для очень маленьких таблиц укажите значение SEGSIZE, равное числу страниц, занимаемых таблицей.

**Рекомендуемые значения SEGSIZE:** В Табл. 69 приводятся рекомендуемые значения SEGSIZE для различных размеров таблиц.

Таблица 69. Рекомендации для SEGSIZE

Число страниц	Рекомендуемое значение SEGSIZE
≤ 28	4 – 28
> 28 < 128 страниц	32
≥ 128 страниц	64

## **Просмотр многораздельных табличных пространств**

Многораздельные табличные пространства несегментированы. Просмотр таких пространств более эффективен, чем просмотр однораздельных табличных пространств. DB2 может использовать ограниченный просмотр разделов, как описывается в разделе “Ограничиваются ли просмотр определенными разделами? (PAGE\_RANGE=Y)” на стр. 718.

## **Просмотр табличного пространства и последовательная предварительная выборка**

Независимо от типа табличного пространства DB2 планирует использование для просмотра табличного пространства последовательной предварительной выборки. Для сегментированного табличного пространства DB2 может и не использовать при выполнении предварительную выборку, если она определит, что обращение должно производиться менее, чем к четырем страницам. Более подробно о последовательной предварительной выборке читайте в разделе “Последовательная предварительная выборка (PREFETCH=S)” на стр. 738.

Если вы не хотите использовать последовательную предварительную выборку для какого-то запроса, можете добавить условие OPTIMIZE FOR 1 ROW.

## **Индексный путь доступа**

DB2 использует следующие индексные пути доступа:

- “Согласованный просмотр индекса (MATCHCOLS>0)”
- “Индексное экранирование” на стр. 724
- “Несогласованный просмотр индекса (ACCESSTYPE=I и MATCHCOLS=0)” на стр. 725
- “Просмотр индекса с IN-списком (ACCESSTYPE=N)” на стр. 725
- “Многоиндексный доступ (ACCESSTYPE – M, MX, MI или MU)” на стр. 725
- “Доступ с одной выборкой (ACCESSTYPE=I1)” на стр. 727
- “Доступ только через индекс (INDEXONLY=Y)” на стр. 727
- “Индекс уникальности с равенством (MATCHCOLS=числу столбцов индекса)” на стр. 728

## **Согласованный просмотр индекса (MATCHCOLS>0)**

При согласованном просмотре индекса предикаты ссылаются на ведущие или на все столбцы ключа индекса. Эти предикаты выполняют фильтрацию; обращение производится только к определенным страницам индекса и данных. Если степень фильтрации высокая, согласованный просмотр индекса эффективен.

В общем случае правила для определения числа согласованных столбцов прости, хотя из них есть несколько исключений.

- Будем просматривать столбцы индекса, начиная с ведущего столбца. Для каждого столбца индекса поищем индексируемый предикат – логический терм с этим столбцом (определение логического термасмотрите в разделе “Свойства предикатов” на стр. 659). Если такой предикат есть, он может использоваться в качестве согласованного с индексом предиката.

Столбец MATCHCOLS в таблице планов показывает, сколько столбцов индекса согласованы с предикатами.

- Если для столбца нет согласованного предиката, поиск согласованных предикатов заканчивается.
- Если согласованный предикат – предикат диапазона, больше согласованных столбцов быть не может. В примере согласованного просмотра индекса, который приводится ниже, на предикате диапазона C2>1 поиск согласованных предикатов прекращается.

Исключения:

- Только один предикат включения может быть согласован с данным индексом
- Для доступа с ACESSTYPE=MX и индексного доступа с предварительной выборкой списка предикаты включения не могут быть согласованными с индексом.
- Предикаты объединения не могут быть согласованными предикатами при объединении слиянием (METHOD=2). Например, T1.C1=T2.C1 не может быть согласованным предикатом при объединении слиянием, хотя любой локальный предикат, как например, C1='5', может.

Предикаты объединения могут использоваться в качестве согласованных предикатов для внутренней таблицы объединения с вложенными циклами или гибридного объединения.

**Пример согласованного просмотра индекса:** Предположим, что существует индекс по T(C1,C2,C3,C4):

```
SELECT * FROM T  
      WHERE C1=1 AND C2>1  
            AND C3=1;
```

В этом примере два согласованных столбца. Первый – в предикате C1=1, а второй – в предикате C2>1. Предикат диапазона со столбцом C2 не позволяет использовать C3 в качестве согласованного столбца.

## Индексное экранирование

При **индексном экранировании** предикаты задаются для столбцов индекса, но они не согласованы с индексом. Эти предикаты улучшают индексный доступ, уменьшая число строк, отбираемых при поиске в индексе. Например, пусть существует индекс T(C1,C2,C3,C4):

```
SELECT * FROM T  
      WHERE C1 = 1  
            AND C3 > 0 AND C4 = 2  
            AND C5 = 8;
```

C3>0 и C4=2 – предикаты индексного экранирования. Они могут применяться к индексу, но они не согласованы с индексом. C5=8 – не предикат индексного экранирования, он проверяется после получения данных. Значение MATCHCOLS в таблице планов равно 1.

EXPLAIN не сообщает явно, когда применяется индексное экранирование; однако если MATCHCOLS меньше числа столбцов в ключе индекса, это указывает на то, что может быть применено экранирование.

## **Несогласованный просмотр индекса (ACCESSTYPE=I и MATCHCOLS=0)**

При *несогласованном просмотре индекса* нет согласованных с индексом столбцов. Поэтому проверяются все ключи индекса.

Поскольку несогласованный индекс обычно не дает фильтрации, этот путь доступа может быть эффективен лишь в немногих случаях. Например, в таких ситуациях:

- Когда есть предикаты индексного экранирования  
В этом случае обращение происходит не ко всем страницам данных.
- Когда используется условие OPTIMIZE FOR n ROWS  
При этом условии иногда может выбираться несогласованный индекс, особенно если он обеспечивает порядок, заданный условием ORDER BY.
- Когда в несегментированном табличном пространстве больше одной таблицы  
В этом случае при просмотре табличного пространства читаются ненужные строки. При чтении строк через несогласованный индекс будет прочитано меньше строк.

## **Просмотр индекса с IN–списком (ACCESSTYPE=N)**

*Просмотр индекса с IN–списком* – особый вид согласованного просмотра индекса, при котором один индексируемый предикат включения используется как согласованный предикат равенства.

Просмотр индекса с IN–списком можно рассматривать как последовательность согласованных просмотров индекса, для каждого из которых используются значения из списка IN. В следующем примере, где существует индекс по (C1,C2,C3,C4), может использоваться просмотр индекса по списку включения:

```
SELECT * FROM T
  WHERE C1=1 AND C2 IN (1,2,3)
        AND C3>0 AND C4<100;
```

Таблица планов показывает MATCHCOLS = 3 и ACCESSTYPE = N. Просмотр по списку включения разбивается на три согласованных просмотра индекса:

(C1=1, C2=1, C3>0), (C1=1, C2=2, C3>0), (C1=1, C2=3, C3>0)

## **Многоиндексный доступ (ACCESSTYPE – M, MX, MI или MU)**

При *многоиндексном доступе* для доступа к таблице используется несколько индексов. Это хороший путь доступа, когда:

- Нет эффективного пути доступа по одному индексу.
- Эффективен комбинированный доступ по нескольким индексам.

Для каждого используемого индекса строятся списки RID. Путем объединения или пересечения этих списков получается окончательный список RID, с помощью которого строки результатачитываются с использованием предварительной выборки списка. Многоиндексный доступ можно рассматривать как расширенную предварительную выборку списка с более сложными операциями получения списка RID на первом этапе. Более сложные операции – это объединение и пересечение.

DB2 выбирает многоиндексный доступ для следующего запроса:

```
SELECT * FROM EMP
  WHERE (AGE = 34) OR
        (AGE = 40 AND JOB = 'MANAGER');
```

В этом запросе:

- EMP – таблица со столбцами EMPNO, EMPNAME, DEPT, JOB, AGE и SAL.
- EMPX1 – индекс EMP по столбцу AGE.
- EMPX2 – индекс EMP по столбцу JOB.

В таблице планов доступ описывается последовательностью строк. Для этого запроса значения ACESSTYPE следующие:

### Значение Смысл

M	Начало обработки при многоиндексном доступе
MX	После просмотра индексов применяется объединение или пересечение
MI	Применяется пересечение (AND)
MU	Применяется объединение (OR)

Ниже перечислены этапы выполнения предыдущего запроса; соответствующие значения таблицы планов показаны в рис. 172:

1. Предикат AGE= 34, согласованный с индексом EMPX1, отбирает список строк–кандидатов на включение в набор результатов запроса. Значение MIXOPSEQ равно 1.
2. Предикат AGE= 40, согласованный с индексом EMPX1, также отбирает набор строк–кандидатов. Значение MIXOPSEQ равно 2.
3. Предикат JOB='MANAGER', согласованный с индексом EMPX2, также отбирает набор строк–кандидатов. Значение MIXOPSEQ равно 3.
4. Вычисляется первое пересечение (AND), значение MIXOPSEQ равно 4, ACESSTYPE равно MI. При пересечении двух первых списков строк (MIXOPSEQ 2 и 3) формируется промежуточный список IR1, который не показан в таблице PLAN\_TABLE.
5. Последний шаг, со значением MIXOPSEQ 5 – объединение (OR) двух списков строк, IR1 и списка, полученного на первом шаге (MIXOPSEQ 1). Это объединение и дает набор результатов запроса.

PLAN-NO	TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	PREFETCH	MIXOP-SEQ
1	EMP	M	0		L	0
1	EMP	MX	1	EMPX1		1
1	EMP	MX	1	EMPX1		2
1	EMP	MI	0			3
1	EMP	MX	1	EMPX2		4
1	EMP	MU	0			5

Рисунок 172. Содержимое таблицы планов для запроса, использующего многоиндексный доступ. В зависимости от показателей фильтрации предикатов порядок выполнения отдельных шагов может быть различным.

В этом примере этапы многоиндексного доступа выполняются в порядке расположения предикатов в запросе. Так бывает не всегда. Порядок выполнения отдельных шагов при многоиндексном доступе выбирается так, чтобы эффективнее использовать пространство пула RID и затратить как можно меньше времени.

### **Доступ с одной выборкой (ACCESSTYPE=I1)**

При доступе к индексу с одной выборкой возвращается только одна строка. Это самый лучший путь доступа, который выбирается всегда, когда это возможно. Он используется для операторов, содержащих функцию столбцов MIN или MAX: порядок индекса позволяет получить результат функции в виде одной строки.

Доступ к индексу с одной выборкой возможен, если:

- В запросе используется только одна таблица.
- Есть только одна функция столбцов (или MIN, или MAX).
- Либо ни одного, либо все предикаты согласованы с индексом.
- Нет условия GROUP BY.
- В случае функции MIN существует возрастающий индекс по нужному столбцу, в случае MAX – убывающий индекс.
- Функции столбцов определяются на:
  - Первом столбце индекса, если нет предикатов
  - Последнем согласованном столбце индекса, если последний согласованный предикат – предикат диапазона
  - Следующем столбце индекса (после последнего согласованного), если все согласованные предикаты – предикаты равенства.

#### **Запросы, использующие доступ к индексу с одной выборкой:**

Следующие запросы используют просмотр индекса с одной выборкой (с индексом T(C1,C2 DESC,C3)):

```
SELECT MIN(C1) FROM T;
SELECT MIN(C1) FROM T WHERE C1>5;
SELECT MIN(C1) FROM T WHERE C1>5 AND C1<10;
SELECT MAX(C2) FROM T WHERE C1=5;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2>5;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2>5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2 BETWEEN 5 AND 10;
```

### **Доступ только через индекс (INDEXONLY=Y)**

При доступе только через индекс не требуется доступ к страницам данных, поскольку вся нужная информация есть в индексе. Если же оператор SQL запрашивает столбец, которого нет в индексе, изменяет столбец таблицы или удаляет строку, DB2 вынуждена обращаться к страницам данных. Поскольку индекс почти всегда меньше самой таблицы, доступ только по индексу обычно эффективнее.

Если задан индекс T(C1,C2), доступ только через индекс можно использовать для следующих запросов:

```
SELECT C1, C2 FROM T WHERE C1 > 0;
SELECT C1, C2 FROM T;
SELECT COUNT(*) FROM T WHERE C1 = 1;
```

### Индекс уникальности с равенством (MATCHCOLS=числу столбцов индекса)

Индекс, который полностью согласован и уникален, и для которого все согласованные предикаты – предикаты равенства, называется *индексом уникальности с равенством*. При этом в результате будет только одна строка. Если невозможен доступ к индексу с одной выборкой, этот способ доступа считается самым эффективным. (Индексом уникальности считается только индекс, определенный как индекс уникальности.)

Иногда DB2 может определить, что индекс, который не полностью согласован, на самом деле может быть индексом уникальности с равенством. Рассмотрим следующий случай:

```
Индекс_уникальности1: (C1, C2)
Индекс_уникальности2: (C2, C1, C3)
```

```
SELECT C3 FROM T
  WHERE C1 = 1 AND
        C2 = 5;
```

Индекс1 – полностью согласованный индекс уникальности с равенством. Но индекс2 – тоже индекс уникальности с равенством, хотя он не полностью согласован. Индекс2 – лучший выбор, так как он еще и делает возможным доступ только по индексу.

### UPDATE с использованием индекса

Если столбцы индекса не изменяются, при операции UPDATE можно использовать индекс.

Чтобы при изменении индекса можно было использовать согласованный просмотр индекса, должны выполняться следующие условия:

- Для каждого изменяемого столбца ключа должен существовать предикат вида "столбец\_индекса = константа" или "столбец\_индекса IS NULL".
- если используется производная таблица, должно быть указано условие WITH CHECK OPTION.

При предварительной выборке списка или многоиндексном доступе в операции UPDATE может использоваться любой индекс или индексы (конечно, если они обеспечивают эффективный доступ к данным).

---

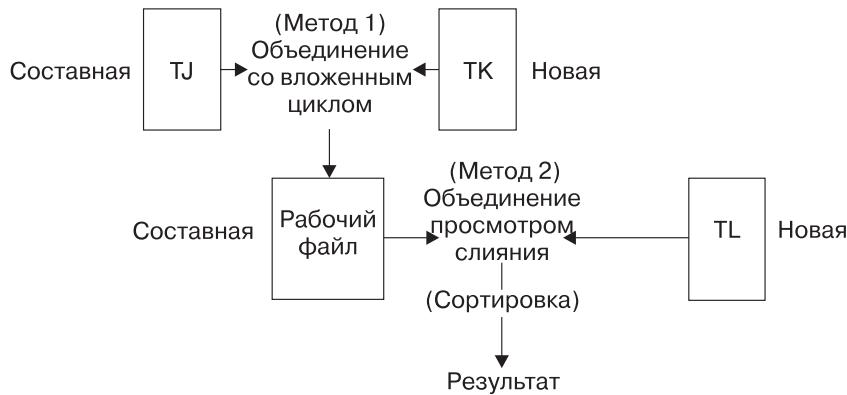
### Объяснение доступа к нескольким таблицам

При операции *объединения* СУБД получает строки из двух или большего числа таблиц и соединяет их. В операции указываются по крайней мере две таблицы, хотя среди таблиц могут быть совпадающие.

Этот раздел начинается с пункта “Определения и примеры” на стр. 729; за ним следуют описания способов объединения, которые могут встретиться в таблице планов:

- “Объединение с вложенными циклами (METHOD=1)” на стр. 731
- “Объединение просмотром слияния (METHOD=2)” на стр. 733
- “Гибридное объединение (METHOD=4)” на стр. 736

## Определения и примеры



METHOD	TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	TSLOCK-MODE
0	TJ	I	1	TJX1	Н	IS
1	TK	I	1	TKX1	Н	IS
2	TL	I	0	TLX1	Д	S
3			0		Н	

SORTN UNIQ	SORTN JOIN	SORTN ORDERBY	SORTN GROUPBY	SORTC UNIQ	SORTC JOIN	SORTC ORDERBY	SORTC GROUPBY
Н	Н	Н	Н	Н	Н	Н	Н
Н	Н	Н	Н	Н	Н	Н	Н
Н	Д	Н	Н	Н	Д	Н	Н
Н	Н	Н	Н	Н	Н	Д	Н

Рисунок 173. Способы объединения и их описание в таблице планов

В операции объединения могут участвовать и более двух таблиц. Но при этом операция выполняется в несколько этапов, на каждом из которых объединяются две таблицы.

**Определения:** Составная таблица (или внешняя таблица) операции объединения – это таблица, полученная на предыдущем этапе, или, если это первый этап – таблица, доступ к которой производится первым.(На первом этапе, таким образом, составная таблица "составлена" только из одной таблицы.) Новая таблица (или внутренняя таблица) операции объединения – это таблица, которая на данном этапе используется впервые.

**Пример:** На рис. 173 показан набор столбцов таблицы. DB2 выполняет 4 этапа:

1. Обращается к первой таблице (METHOD=0) под именем TJ (TNAME), которая становится составной таблицей на этапе 2.

2. Объединяет TJ с новой таблицей TK, формируя новую составную таблицу.
3. Сортирует новую таблицу TL (SORTN\_JOIN=Y) и составную таблицу (SORTC\_JOIN=Y), затем объединяет две отсортированных таблицы.
4. Сортирует окончательную составную таблицу (TNAME пусто) в нужном порядке (SORTC\_ORDERBY=Y).

**Определения:** При операции объединения строкам одной таблицы на основании критерия объединения сопоставляются строки другой таблицы. Например, критерий может состоять в том, что значение столбца A одной таблицы равно значению столбца X другой таблицы (WHERE T1.A = T2.X).

Объединения можно разделить на два типа в зависимости от того, что делается со строками одной таблицы, которым согласно критерию объединения не соответствует ни одной строки в другой таблице:

- При *внутреннем объединении* отбрасываются все строки из каждой таблицы, которым не соответствует ни одна строка в другой таблице.
- При *внешнем объединении* строки, не имеющие соответствий, из одной или другой таблицы или из обеих таблиц сохраняются. Строки составной таблицы, полученные из таких строк, дополняются пустыми значениями. Внешние объединения различаются по тому, какие строки, не имеющие соответствий, они сохраняют.

Таблица 70. Типы объединений и сохраняемые строки без соответствий

Вид внешнего объединения:	Сохраняет строки без соответствий из:
Левое внешнее объединение	Составной (внешней) таблицы
Правое внешнее объединение	Новой (внутренней) таблицы
Полное внешнее объединение	Обеих таблиц

**Пример:** На рис. 174 на стр. 731 показано внешнее объединение и значения в таблице планов. В столбце JOIN\_TYPE тип внешнего объединения задается одним из следующих значений:

- F для ПОЛНОГО ВНЕШНЕГО ОБЪЕДИНЕНИЯ
- L для ЛЕВОГО ВНЕШНЕГО ОБЪЕДИНЕНИЯ
- Пустое для ВНУТРЕННЕГО ОБЪЕДИНЕНИЯ или когда нет объединения

Во время выполнения DB2 преобразует каждое правое внешнее объединение в левое внешнее объединение, поэтому JOIN\_TYPE никогда не указывает отдельно на правое внешнее объединение.

```

EXPLAIN PLAN SET QUERYNO = 10 FOR
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
  FROM PROJECTS LEFT JOIN
       (SELECT PART,
              COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
              PRODUCTS.PRODUCT
     FROM PARTS FULL OUTER JOIN PRODUCTS
       ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
      ON PROJECTS.PROD# = PRODNUM

```

QUERYNO	QBLOCKNO	PLANNO	TNAME	JOIN_TYPE
10	1	1	PROJECTS	
10	1	2	TEMP	L
10	2	1	PRODUCTS	
10	2	2	PARTS	F

Рисунок 174. Данные таблицы планов для примера с внешними объединениями

**Материализация при внешнем объединении:** Иногда DB2 вынуждена материализовать набор результатов, если внешнее объединение используется в сочетании с другими объединениями, производными таблицами или вложенными табличными выражениями. Это можно определить по столбцу таблицы планов TNAME, куда для материализованной таблицы заносится имя DSNWFQB(xx); здесь xx – номер блока запроса (QBLOCKNO), который создал рабочий файл.

## Объединение с вложенными циклами (METHOD=1)

В этом разделе описывается этот распространенный способ объединения.

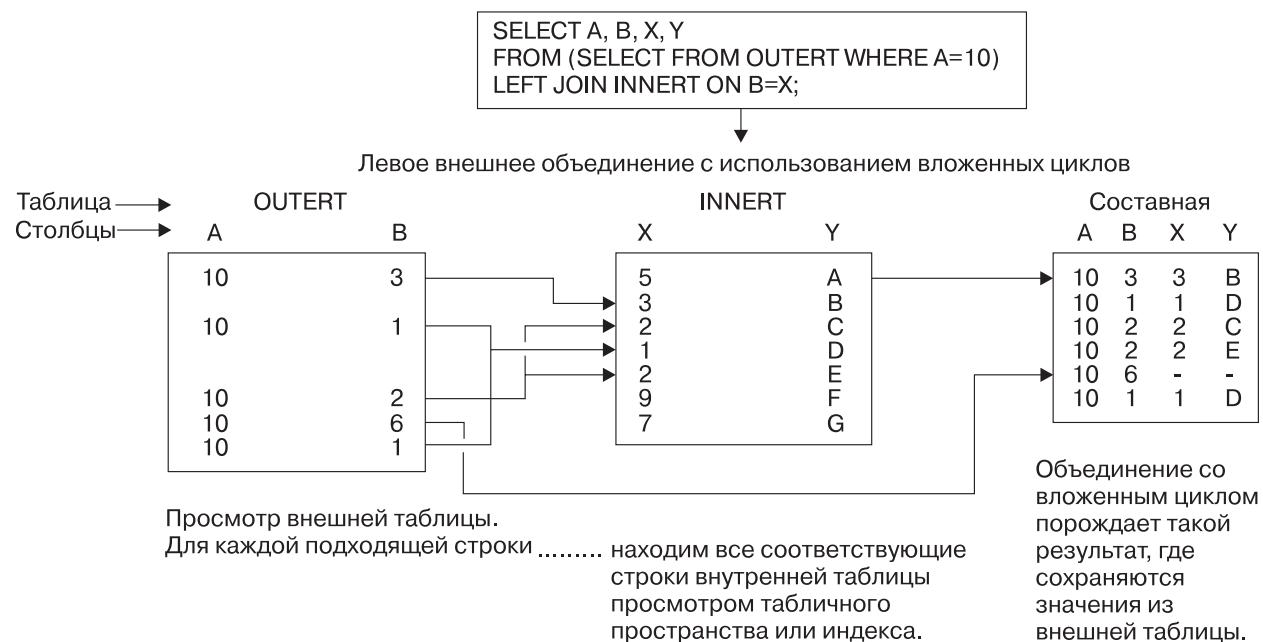


Рисунок 175. Объединение с вложенными циклами при левом внешнем объединении

## **Способ объединения**

DB2 сканирует составную (внешнюю) таблицу. Для каждой отобранный строки этой таблицы (удовлетворяющей всем предикатам для этой таблицы), DB2 ищет соответствующие строки в новой (внутренней) таблице. Все найденные строки она соединяет (выполняет конкатенацию) с текущей строкой составной таблицы. Если ни одной строки не найдено:

В случае внутреннего объединения DB2 отбрасывает текущую строку.

В случае внешнего объединения DB2 присоединяет строку с пустыми значениями.

Предикаты 1 этапа и 2 этапа исключают неподходящие строки во время объединения. (Определение этих типов предикатов смотрите в разделе “Предикаты 1 этапа и 2 этапа” на стр. 661.) DB2 может просматривать каждую таблицу, используя любой доступный способ, включая просмотр табличного пространства.

## **Соображения производительности**

При объединении с вложенным циклом DB2 многократно просматривает внутреннюю таблицу. При этом DB2 один раз просматривает внешнюю таблицу, а просмотр внутренней таблицы выполняется столько раз, сколько строк отбираются из внешней таблицы. Поэтому объединение с вложенным циклом – как правило, самый эффективный способ объединения, если значения столбца объединения, передаваемые внутренней таблице, последовательны и индекс по этому столбцу во внутренней таблице кластеризован, или число строк внутренней таблицы, полученных через индекс, мало.

## **Когда используется**

Объединение с вложенным циклом часто используется, если:

- Внешняя таблица невелика.
- Предикаты с небольшими показателями фильтрации значительно уменьшают число отбираемых строк во внешней таблице.
- Существует эффективный сильно кластеризованный индекс по столбцам объединения во внутренней таблице.
- Во внутренней таблице происходит обращение к небольшому числу страниц данных.

**Пример: левое внешнее объединение:** Пример на рис. 175 на стр. 731 иллюстрирует левое внешнее объединение с вложенным циклом. Внешнее объединение сохраняет не имеющую соответствия строку с A=10 и B=6 в таблице OUTERT. Внутреннее объединение отличается только тем, что эта строка отбрасывается.

**Пример: приоритет таблицы с одной строкой:** В приведенном ниже примере с индексом уникальности по T1.C2 DB2 определяет, что в T1 есть только одна строка, удовлетворяющая условию поиска. При объединении с вложенным циклом DB2 обращается первой к таблице T1.

```
SELECT * FROM T1, T2  
  WHERE T1.C1 = T2.C1 AND  
        T1.C2 = 5;
```

**Пример: Декартово объединение с первой малой таблицей:** Декартово объединение – вид объединения с вложенным циклом без предиката объединения двух таблиц. DB2 обычно избегает использовать декартово объединение, однако иногда, как на следующем примере, это самый эффективный метод. В данном запросе используются три таблицы: T1 с 2 строками, T2 с 3 строками и T3 с 10 миллионами строк.

```
SELECT * FROM T1, T2, T3  
  WHERE T1.C1 = T3.C1 AND  
        T2.C2 = T3.C2 AND  
        T3.C3 = 5;
```

Между T1 и T3, а также между T2 и T3 есть предикаты объединения. Между T1 и T2 предикатов объединения нет.

Предположим, что в 5 миллионах строк T3 значение C3=5. Время обработки будет очень большим, если T3 – внешняя таблица объединения, и обращения к T1 и T2 будут выполняться для каждой из 5 строк.

Однако если все строки T1 и T2 объединить без предиката объединения, обращение к 5 миллионам строк будет производиться всего 6 раз, по одному разу для каждой из строк декартова объединения T1 и T2. Трудно сказать, какой из способов доступа будет эффективнее. DB2 оценивает различные варианты и может выбрать, например, такой порядок доступа к таблицам: T1, T2, T3.

**Сортировка составной таблицы:** Таблица планов может показывать объединение с вложенным циклом, включающее сортировку составной таблицы. DB2 может сортировать составную таблицу (внешнюю таблицу в примере рис. 175), если:

- Столбцы объединения в составной и новой таблицах упорядочены по-разному.
- В составной таблице нет индекса по столбцу объединения.
- Индекс есть, но он слабо кластеризован.

Объединение с вложенным циклом с сортировкой составной таблицы эффективно использует последовательное обнаружение для предварительной выборки страниц данных новой таблицы, снижая число синхронных операций ввода–вывода и время выполнения.

## Объединение просмотром слияния (METHOD=2)

Объединение просмотром слияния также называется *объединением слиянием* или *объединением сортировкой*. Для использования этого способа в запросе должен быть предикат или предикаты вида TABLE1.COL1=TABLE2.COL2, где у столбцов COL1 и COL2 одинаковые тип и длина

### Способ объединения

На рис. 176 на стр. 734 показано объединение просмотром слияния.

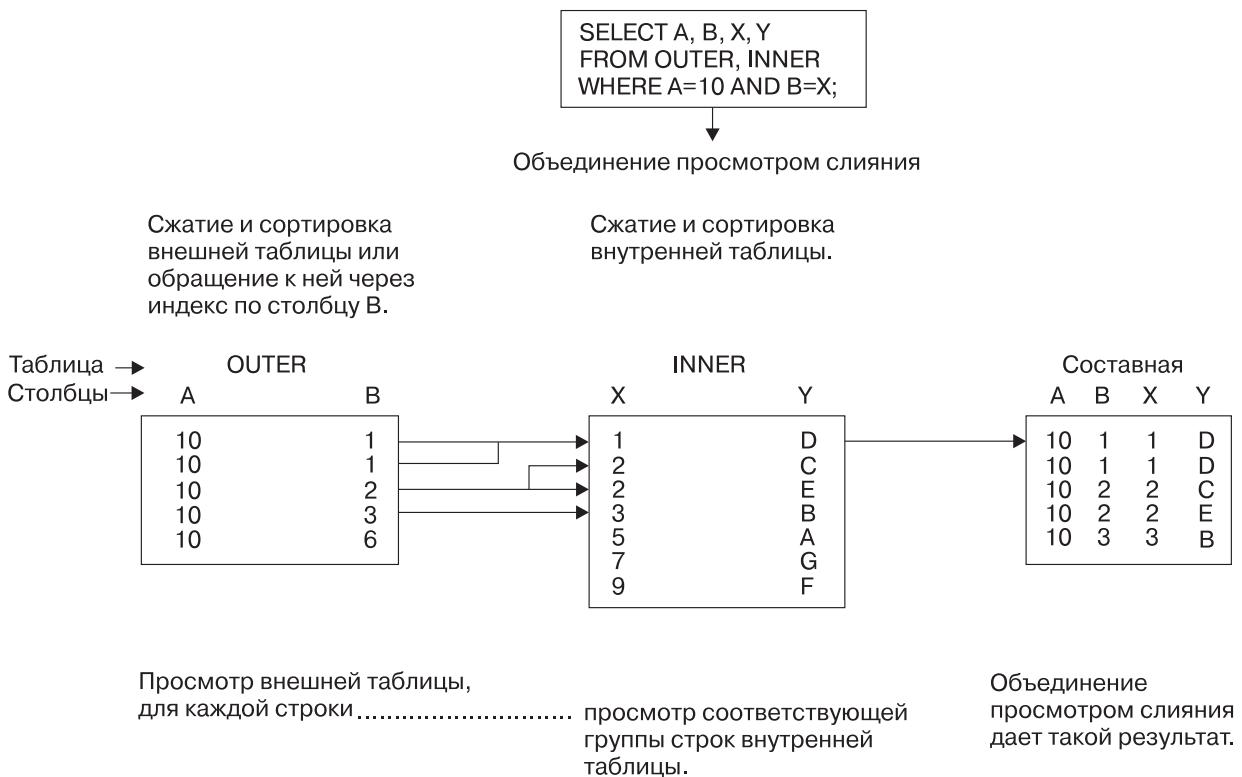


Рисунок 176. Объединение просмотром слияния

DB2 просматривает обе таблицы в порядке, задаваемом столбцами объединения. Если нет эффективных индексов, обеспечивающих порядок, DB2 может отсортировать внешнюю или внутреннюю таблицу или обе таблицы. Внутренняя таблица помещается в рабочий файл; внешняя таблица помещается в рабочий файл, только если для нее нужна сортировка. Если строка внешней таблицы соответствует строке внутренней таблицы, DB2 объединяет строки.

Затем DB2 читает следующую строку внутренней таблицы, которая может соответствовать той же строке внешней таблицы, и продолжает читать строки, пока есть соответствие. Когда соответствия больше нет, DB2 читает следующую строку внешней таблицы.

- Если в этой строке то же значение столбца объединения, DB2 снова читает группу соответствующих записей из внутренней таблицы. Таким образом группа записей с повторяющимся значением столбца объединения во внутренней таблице просматривается столько раз, сколько соответствующих записей во внешней таблице.
- Если в строке внешней таблицы новое значение столбца объединения, DB2 производит поиск вперед во внутренней таблице. Она может найти:
  - Несоответствующие строки внутренней таблицы, с меньшими значениями столбца объединения.
  - Новую строку с совпадающим значением. DB2 затем начинает процесс снова.
  - Строку с большим значением столбца объединения. Теперь строка внешней таблицы осталась без соответствия. DB2 продвигается вперед во внешней таблице и может найти:

- Несоответствующие строки внешней таблицы.
- Новую соответствующую строку. DB2 затем начинает процесс снова.
- Внешнюю строку с большим значением столбца объединения. Теперь строка внутренней таблицы осталась без соответствия, и DB2 возобновляет поиск во внутренней таблице.

Если DB2 находит несоответствующую строку:

При внутреннем объединении DB2 отбрасывает строку.

При левом внешнем объединении DB2 отбрасывает строку, если она из внутренней таблицы и оставляет, если она из внешней таблицы.

При полном внешнем объединении DB2 оставляет строку.

Когда DB2 оставляет строку без соответствия, она добавляет к ней набор пустых значений, как если бы в другой таблице была соответствующая строка. Для полного внешнего объединения используется объединение просмотром слияния.

### **Соображения производительности**

Полное внешнее объединение этим способом использует все предикаты из условия ON, чтобы установить соответствие между двумя таблицами, и читает во время объединения все строки. При внутреннем и левом внешнем объединении для этого используются только предикаты 1 этапа из условия ON. Если соответствие таблиц задается несколькими столбцами, эффективнее поместить все предикаты объединения в условие ON, а не оставлять некоторые из них в условии WHERE.

При внутреннем объединении DB2 может при связывании вывести дополнительные предикаты для внутренней таблицы и применить их во время выполнения к отсортированной внешней таблице. Эти предикаты позволяют уменьшить размер рабочего файла для внутренней таблицы.

Если DB2 использовала эффективный индекс по столбцам объединения для считывания строк из внутренней таблицы, эти строки уже упорядочены нужным образом. DB2 помещает данные непосредственно в рабочий файл без сортировки внутренней таблицы, что снижает время выполнения.

### **Когда используется**

Просмотр слияния часто используется, если:

- Из внешней и внутренней таблиц отбирается много строк, а предикаты объединений не обеспечивают большой фильтрации; то есть при объединении "многих с многими".
- Таблицы большие и нет индексов по столбцам, задающим соответствие.
- Из внутренних таблиц выбирается немного строк. В этом случае DB2 применяет сортировку. Сортировка тем эффективнее, чем меньше столбцов.

## Гибридное объединение (METHOD=4)

Этот способ используется только при внутреннем объединении и требует наличия индекса по столбцу объединения для внутренней таблице.

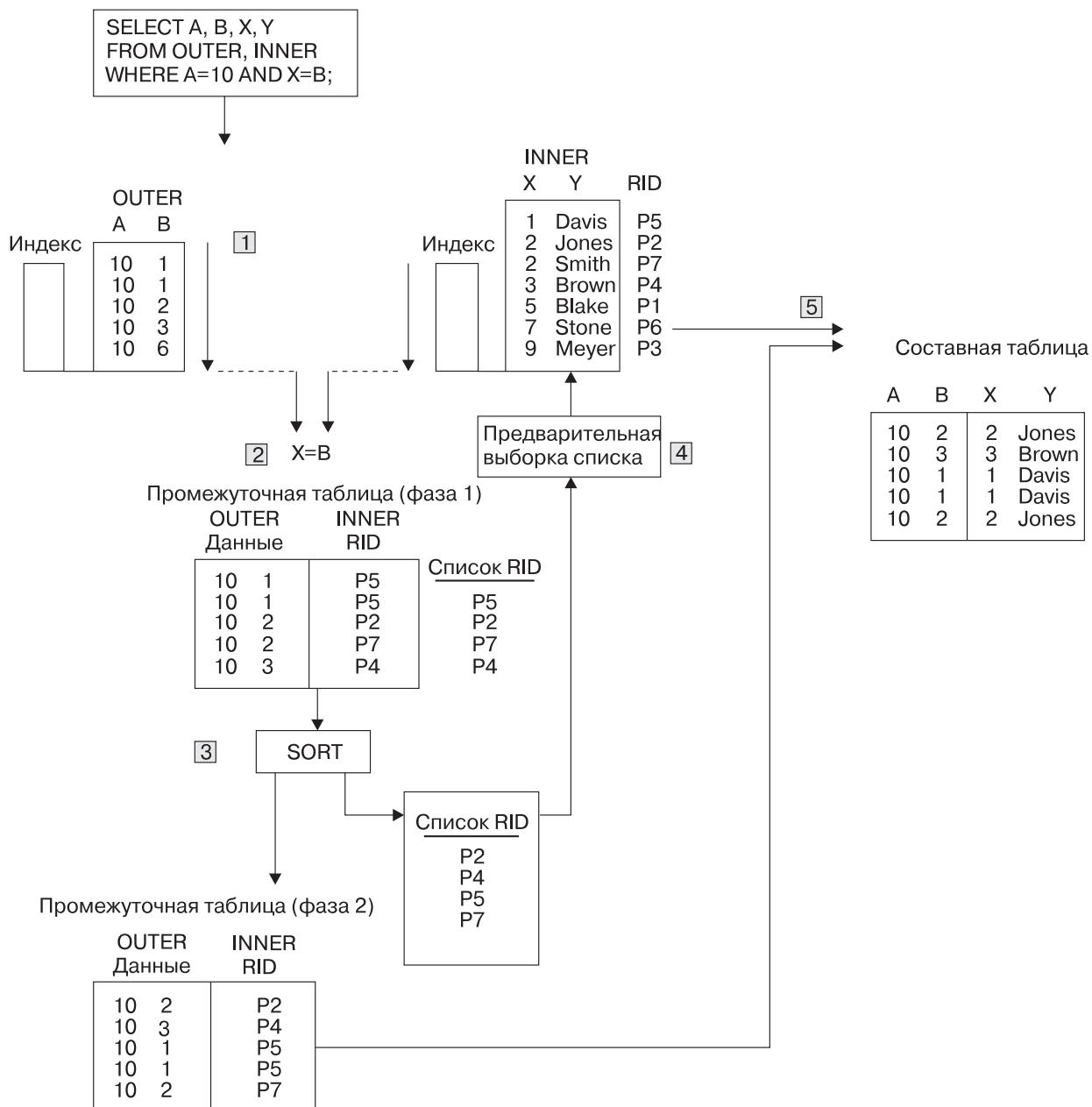


Рисунок 177. Гибридное объединение (SORTN\_JOIN='Y')

### Способ объединения

Этот способ требует получения RID в порядке, который позволяет использовать предварительную выборку списка. Этапы показаны на рис. 177. В данном примере как во внешней (OUTER), так и во внутренней (INNER) таблице существуют индексы по столбцам объединения.

DB2 выполняет следующие действия:

- 1** Просматривает внешнюю таблицу (OUTER).
- 2** Объединяет внешние таблицы с RID из индекса внутренней таблицы. Результат – промежуточная таблица фазы 1. Индекс внутренней таблицы просматривается для каждой строки внешней таблицы.
- 3** Сортирует данные внешней таблицы и RID, создавая отсортированный список RID и промежуточную таблицу фазы 2. Сортировка индицируется значением Y в столбце SORTN\_JOIN таблицы планов. Если индекс внутренней таблицы – индекс кластеризации, DB2 может не выполнять сортировку; значение SORTN\_JOIN тогда равно N.
- 4** Считывает данные из внутренней таблицы, используя предварительную выборку списка.
- 5** Объединяет данные из внутренней таблицы и промежуточную таблицу фазы 2 в окончательную составную таблицу.

## Возможные результаты EXPLAIN для гибридного объединения

Значение столбца	Объяснение
<b>METHOD='4'</b>	Было использовано гибридное объединение.
<b>SORTC_JOIN='Y'</b>	Составная таблица была отсортирована.
<b>SORTN_JOIN='Y'</b>	Промежуточная таблица была отсортирована в соответствии с порядком RID внутренней таблицы. Список RID был получен с помощью некластеризованного индекса.
<b>SORTN_JOIN='N'</b>	Сортировки списка RID промежуточной таблицы не было. Для получения списка RID внутренней таблицы использовался кластеризованный индекс, и RID уже были упорядочены нужным образом.
<b>PREFETCH='L'</b>	Страницы считаны с использованием предварительной выборки списка.

## Соображения производительности

Гибридное объединение использует предварительную выборку списка более эффективно, чем объединение с вложенным циклом, особенно, если для предиката объединения есть индексы с низким отношением кластеризации. При этом повторяющиеся значения обрабатываются эффективнее, поскольку внутренняя таблица сканируется только раз для каждого набора повторяющихся значений столбца объединения внешней таблицы.

Если индекс внутренней таблицы сильно кластеризован, сортировать промежуточную таблицу не нужно (SORTN\_JOIN=N). Промежуточная таблица размещается в памяти, а не в рабочем файле.

## Когда используется

Гибридное объединение, как правило, используется, если:

- Используется некластеризованный индекс или индексы по столбцу объединения внутренней таблицы
- Из внешней таблицы отбираются строки с повторяющимися значениями

## Предварительная выборка данных

*Предварительная выборка* – механизм чтения набора страниц, обычно 32, в пул буферов одной асинхронной операцией ввода–вывода. Предварительная выборка позволяет сэкономить время работы процессора и затраты на ввод–вывод. Чтобы достичь этой экономии, контролируйте предварительную выборку.

В таблице планов отображаются два вида предварительной выборки:

- “Последовательная предварительная выборка (PREFETCH=S)”
- “Предварительная выборка списка (PREFETCH=L)” на стр. 739

Если DB2 не выбрала предварительную выборку списка при связывании, она все–таки может использовать ее при выполнении запроса. Этот способ описывается в разделе:

- “Последовательное обнаружение во время выполнения” на стр. 741

### Последовательная предварительная выборка (PREFETCH=S)

При *последовательной предварительной выборке* происходит чтение последовательного набора страниц. Максимальное число страниц, считываемое по требованию из прикладной программы, определяется размером используемого пула буферов:

- Для пула 4–килобайтных буферов число страниц, считываемых заранее, показано в приведенной ниже таблице.

Таблица 71. Число страниц, считываемое при предварительной выборке, для 4–килобайтных буферов

Размер пула буферов	Считывается страниц
<=223 буферов	8 страниц для каждой асинхронной операции ввода–вывода
224–999 буферов	16 страниц для каждой асинхронной операции ввода–вывода
1000+ буферов	32 страниц для каждой асинхронной операции ввода–вывода

- Для пула 8–килобайтных буферов число страниц, считываемых при предварительной выборке, показано в следующей таблице.

Таблица 72. Число страниц, считываемое при предварительной выборке, для 8–килобайтных буферов

Размер пула буферов	Считывается страниц
<=112 буферов	4 страницы для каждой асинхронной операции ввода–вывода
113–499 буферов	8 страниц для каждой асинхронной операции ввода–вывода
500+ буферов	16 страниц для каждой асинхронной операции ввода–вывода

- Для пула 16–килобайтных буферов число страниц, считываемых при предварительной выборке, показано в следующей таблице.

*Таблица 73. Число страниц, считываемое при предварительной выборке, для 16–килобайтных буферов*

<b>Размер пула буферов</b>	<b>Считывается страниц</b>
<=56 буферов	2 страницы для каждой асинхронной операции ввода–вывода
57–249 буферов	4 страницы для каждой асинхронной операции ввода–вывода
250+ буферов	8 страниц для каждой асинхронной операции ввода–вывода

- Для пула 16–килобайтных буферов число страниц, считываемых при предварительной выборке, показано в следующей таблице.

*Таблица 74. Число страниц, считываемое при предварительной выборке, для 32–килобайтных буферов*

<b>Размер пула буферов</b>	<b>Считывается страниц</b>
<=16 буферов	0 страниц (предварительная выборка отключена)
17–99 буферов	2 страницы для каждой асинхронной операции ввода–вывода
100+ буферов	4 страницы для каждой асинхронной операции ввода–вывода

В некоторых утилитах (LOAD, REORG, RECOVER) при предварительной выборке может считываться вдвое больше страниц.

**Когда используется:** Последовательная предварительная выборка, как правило, используется при просмотре табличного пространства.

При просмотре индекса, когда считывается 8 и более страниц, DB2 выбирает последовательную предварительную выборку при связывании. Отношение кластеризации индекса должно быть не меньше 80%. Предварительно выбираются и страницы данных, и страницы индекса.

## Предварительная выборка списка (PREFETCH=L)

При предварительной выборке списка считывается набор страниц данных, определяемый списком RID, полученных из индекса. Страницы данных не обязательно должны быть последовательными. Одной операцией при предварительной выборке списка может быть считано максимум 32 страницы (для утилит 64).

Предварительная выборка списка может использоваться и при одноиндексном, и при многоиндексном доступе.

### Способ доступа

Предварительная выборка списка выполняется в три этапа:

1. Получение RID: Список RID для определения нужных страниц получается путем согласованного просмотра одного или нескольких индексов.
2. Сортировка RID: Список RID сортируется в порядке возрастания по номеру страницы.

3. Получение данных: Нужные страницы данныхчитываются по порядку по отсортированному списку RID.

Предварительная выборка списка не сохраняет порядок данных, определяемый индексом. Поскольку список RID сортируется по номеру страницы перед обращением к данным, порядок полученных данных не согласован ни с каким столбцом. Если данные должны быть отсортированы в силу условия ORDER BY или по другой причине, требуется дополнительная сортировка.

При гибридном объединении, если индекс сильно кластеризован, номера страниц могут не сортироваться перед обращением к данным.

Предварительная выборка списка может использоваться большинством согласованных предикатов для просмотра индекса. Исключение – предикаты включения (IN); они не могут быть согласованными предикатами, если используется предварительная выборка списка.

### **Когда используется**

Предварительная выборка списка используется:

- Обычно с одним индексом с отношением кластеризации меньше 80%.
- Иногда с индексами с высоким отношением кластеризации, если предполагаемый объем считываемых данных слишком мал для последовательной предварительной выборки, но для его считывания требуется более одного обращения.
- Всегда при многоиндексном доступе.
- Всегда для доступа к данным из внутренней таблицы при гибридном объединении.

### **Предельные значения времени связывания и времени выполнения**

DB2 не использует предварительную выборку списка, если предполагается, что список RID займет более 50% пула RID при выполнении запроса. Размер пула RID можно изменить, изменив поле RID POOL SIZE на панели установки DSNTIPC. Максимальный размер пула RID – 1000Мбайт. Максимальное число RID в одном списке – приблизительно 16 миллионов.

Во время выполнения DB2 прекращает предварительную выборку списка, если должны быть считаны более 25% строк таблицы (но не меньше 4075). Была ли прекращена предварительная выборка списка, показывает запись IFCID 0125 в данных трассировки, преобразованных макрокомандой DSNDQW01.

Когда предварительная выборка списка прекращается, выполнение запроса продолжается способом, зависящим от текущего пути доступа.

- При доступе по одному индексу или по объединению списков RID из двух индексов применяется просмотр табличного пространства.
- При индексном доступе до того, как сформировано пересечение списков RID, выполнение продолжается со следующего этапа многоиндексного доступа. Если это последний этап и ни одного списка RID не было сформировано, применяется просмотр табличного пространства.

При пересечении списков RID, если какой-либо список содержит не больше 32 RID, операция пересечения прекращается, а этот список используется для доступа к данным.

## Последовательное обнаружение во время выполнения

Если DB2 не выбрала предварительную выборку списка при связывании, она все-таки может использовать ее при выполнении запроса. Способ называется *последовательным обнаружением*.

### Когда используется

DB2 может использовать последовательное обнаружение для конечных страниц индекса и страниц данных. Чаще всего оно используется по отношению к внутренней таблице объединения с вложенным циклом при последовательном доступе к данным.

Если к таблице постоянно происходят обращения из одного и того же оператора (например, DELETE в цикле do-while), возможен последовательный доступ к страницам данных или конечным страницам индекса. Это обычно в среде пакетной обработки. Последовательное обнаружение может использоваться для следующих операторов:

- SELECT или FETCH
- UPDATE и DELETE
- INSERT при последовательном доступе к существующим страницам данных

DB2 может использовать последовательное обнаружение, если она не выбрала при связывании последовательную предварительную выборку из-за неточной оценки числа страниц для доступа.

Последовательное обнаружение не используется для операторов SQL, на которые могут накладываться реляционные ограничения.

### Как узнать, было ли использовано

Последовательное обнаружение определяется только во время выполнения и не показывается в таблице планов. Определить, было ли использовано последовательное обнаружение, можно по записи IFCID 0003 в данных учетной трассировки или записи IFCID 0006 в данных трассировки производительности.

### Как определить, можно ли использовать

Когда прикладная программа просматривает данные DB2, пользуясь индексом, определяется характер доступа к странице. При этом можно обнаружить ситуации, когда доступ становится последовательным или близким к последовательному.

Отслеживаются последние 8 страниц. Страница считается последовательной, если она находится за текущей страницей на расстоянии не более чем R/2 страниц, где R – объем предварительной выборки. Обычно R равно 32.

Если страница последовательная, DB2 определяет, является ли доступ к данным последовательным или близким к последовательному. Доступ к данным считается последовательным, если из последних 8 страниц более 4 были последовательными; то же относится и к доступу только по индексу.

Такое слежение применяется постоянно, позволяя прекращать и возобновлять последовательный доступ.

После того, как впервые был объявлен последовательный доступ, что называется **начальным последовательным доступом к данным**, вычисляются следующие три диапазона страниц:

- Пусть была затребована страница A. RUN1 – диапазон страниц длиной P/2 страниц, начиная с A.
- Пусть B – страница A + P/2. RUN2 – диапазон страниц длиной P/2, начиная с B.
- Пусть C – страница B + P/2. RUN3 – диапазон страниц длиной P страниц, начинающийся с C.

Например, если A – страница 10, диапазоны страниц, вычисляемые DB2, выглядят, как на следующем рисунке.

	A	B	C	
	RUN1	RUN2	RUN3	
Номер страницы	10	26	42	
P=32 страницы	16   16   32			

*Рисунок 178. Начальные диапазоны страниц для определения возможности использования предварительной выборки*

При начальном последовательном доступе к данным запрашивается предварительная выборка для P страниц, начиная со страницы A (RUN1 и RUN2). Объем предварительной выборки всегда P страниц.

Для последующих страниц, где страница 1)странично–последовательная и 2)действует последовательный доступ к данным, предварительная выборка запрашивается следующим образом:

- Если нужная страница – в диапазоне RUN1, предварительная выборка не включается, потому что она уже была включена, когда был объявлен последовательный доступ.
- Если нужная страница в диапазоне RUN2, предварительная выборка включается для RUN3, и RUN2 становится RUN1, RUN3 становится RUN2, а диапазон страниц, начинающийся с C+P длиной P страниц становится RUN3.

Если после того, как последовательный доступ был выключен, характер доступа становится последовательным, как описано выше, снова объявляется начальный последовательный доступ с соответствующей обработкой.

Поскольку при связывании можно только оценить число считываемых страниц, последовательное обнаружение – это своего рода страховка, которая применяется, когда доступ к данным становится последовательным.

В крайних ситуациях, когда достигаются определенные предельные значения для пула буферов, последовательная предварительная выборка может быть выключена. Описание пулов буферов и предельных значений смотрите в разделе Раздел 5 (Том 2) *DB2 Administration Guide*.

---

## Информация о сортировках

Существует два общих типа сортировок, которые DB2 может использовать при доступе к данным. Первый тип – сортировки строк данных: другой – сортировка идентификаторов строк (RID) в списке RID.

### Сортировка данных

После работы EXPLAIN информация о сортировках DB2 появляется в таблице PLAN\_TABLE. Это могут быть сортировки составной или новой таблицы. Если в одной строке PLAN\_TABLE значение 'Y' стоит в нескольких столбцах, описывающих сортировку составной таблицы, значит, одна сортировка выполняет две функции. (DB2 не выполняет двух сортировок, если в одной строке два 'Y'.) Например, если и SORTC\_ORDERBY, и SORTC\_UNIQ содержат 'Y' в одной строке PLAN\_TABLE, одна сортировка упорядочивает строки и удаляет повторяющиеся строки.

Новую таблицу DB2 может сортировать только для операции объединения, что указывается в поле SORTN\_JOIN.

#### Сортировки по условиям GROUP BY и ORDER BY

Эти сортировки указываются в полях SORTC\_ORDERBY и SORTC\_GROUPBY таблицы PLAN\_TABLE. Если указаны оба условия – GROUP BY и ORDER BY, и каждый элемент в списке условия ORDER BY входит также в список GROUP BY, выполняется только одна сортировка, которая указывается в поле SORTC\_ORDERBY.

Сортировка по условию GROUP BY выполняется лучше, когда запрос использует одну таблицу и когда нет индекса по столбцу, входящему в GROUP BY.

#### Сортировка для удаления повторяющихся значений

Сортировка этого типа используется в запросах с SELECT DISTINCT с функцией столбцов типа COUNT(DISTINCT COL1) или для удаления записей с повторяющимися значениями при выполнении условия UNION. Она указывается в поле SORTC\_UNIQ таблицы PLAN\_TABLE.

#### Сортировки при операциях объединения

Перед объединением двух таблиц часто требуется сначала отсортировать одну или обе таблицы. При гибридном объединении (METHOD 4) и объединении с вложенным циклом (METHOD 1) сортировка составной таблицы может сделать операцию объединения эффективнее. При объединении слиянием (METHOD 2) и составная, и новая таблицы должны быть отсортированы, если нет индекса, который обеспечивает доступ к таблицам в нужном порядке. Сортировки, требуемые для операций объединения, указываются в полях SORTN\_JOIN и SORTC\_JOIN таблицы PLAN\_TABLE.

## **Сортировки, необходимые для выполнения подзапросов**

Если в запросе есть несвязанный подзапрос IN или NOT IN, результаты запроса сортируются и записываются в рабочий файл, который будет использовать родительский запрос. Результаты подзапроса сортируются для более эффективного выполнения предиката IN или NOT IN родительским запросом. Повторяющиеся значения в рабочем файле не нужны и поэтому удаляются. Несвязанные подзапросы с ключевыми словами =ANY или =ALL, или NOT=ANY, или NOT=ALL требуют сортировки того же типа, что и подзапросы IN и NOT IN. Когда выполняется сортировка для несвязанного подзапроса, в таблице PLAN\_TABLE заполняются два поля – SORTC\_ORDERBY и SORTC\_UNIQUE, так как DB2 удаляет повторяющиеся значения и выполняет сортировку.

SORTN\_GROUPBY, SORTN\_ORDERBY и SORTN\_UNIQ в настоящее время в DB2 не используются.

## **Сортировка списка RID**

DB2 сортирует список RID по возрастанию номеров страниц для предварительной выборки списка. Эта сортировка очень быстрая и выполняется полностью в памяти. Сортировка RID обычно не отображается в таблице PLAN\_TABLE, однако, как правило, она выполняется, если используется предварительная выборка списка. Единственное исключение – когда выполняется гибридное объединение и для внутренней таблицы используется единственный, сильно кластеризованный индекс. В этом случае SORTN\_JOIN содержит 'N', показывающее, что сортировки списка RID для внутренней таблицы не было.

## **Влияние сортировки на оператор OPEN CURSOR**

Тип требуемой сортировки указателя влияет на время выполнения DB2 оператора OPEN CURSOR. В этом разделе в общих чертах описывается влияние сортировки и параллельной обработки на оператор OPEN CURSOR.

### ***Без параллельной обработки:***

- Если сортировка не требуется, OPEN CURSOR не обращается к данным. Данные возвращаются при первой выборке.
- Если требуется сортировка, OPEN CURSOR вызывает материализацию результата таблицы. Управление возвращается прикладной программе после материализации таблицы–результата. Если указатель, требующий сортировки, закрывается, а затем открывается снова, снова выполняется сортировка.
- Если применяется сортировка RID, но не данных, построение списка RID по индексу и возврат первой записи данных происходит только после выборки первой строки.

### ***С параллельной обработкой:***

- По оператору OPEN CURSOR асинхронно начинается параллельная обработка, независимо от того, требуется сортировка или нет. Сразу после этого управление возвращается прикладной программе.
- Если применяется сортировка RID, но не данных, параллельная обработка начинается только после первой выборки. Действия такие же, как и без параллельной обработки.

## **Обработка производных таблиц и вложенных табличных выражений**

В этом разделе описывается, как DB2 обрабатывает производные таблицы и вложенные табличные выражения. Вложенное табличное выражение (называемое здесь *табличным выражением*) – это подзапрос в условии FROM оператора SQL SELECT. Табличные выражения обрабатываются аналогично производным таблицам. Для выполнения запросов, использующих производные таблицы или табличные выражения, используются два способа:

- Слияние
  - Материализация

Определить, какой способ был использован, можно, выполнив EXPLAIN для оператора, содержащего производную таблицу или табличное выражение. Ниже приводятся сведения, которые помогут разобраться в информации утилиты EXPLAIN о производных таблицах и табличных выражениях и отладить запросы, в которых они используются.

## Слияние

Слияние эффективнее материализации, как описывается в разделе “Производительность слияния и материализации” на стр. 748. При слиянии оператор, ссылающийся на производную таблицу или табличное выражение, объединяется с подвыбором, задающим производную таблицу или табличное выражение. Полученный логически эквивалентный оператор применяется к базе данных.

Рассмотрим следующие операторы, один из которых определяет производную таблицу, а другой ссылается на нее:

**Ссылка на производную таблицу:**

```
CREATE VIEW VIEW1 (VC1,VC21,VC32) AS      SELECT VC1,VC21
SELECT C1,C2,C3 FROM T1                  FROM VIEW1
WHERE C1 > C3:                           WHERE VC1 IN (A,B,C);
```

Подвыбор оператора, определяющего производную таблицу, может быть объединен сзывающимся на нее оператором, в результате чего получится следующий логически эквивалентный оператор:

Объединенный оператор:

```
SELECT C1,C2 FROM T1  
WHERE C1 > C3 AND C1 IN (A,B,C);
```

Вот еще один пример, когда возможно слияние производной таблицы и табличного выражения:

```
SELECT * FROM V1 X
LEFT JOIN
  (SELECT * FROM T2) Y ON X.C1=Y.C1
    LEFT JOIN T3 Z ON X.C1=Z.C1;
```

## Материализация

Слияние производных таблиц и табличных выражений возможно не всегда.  
Рассмотрим следующие операторы:

Определение производной таблицы: Ссылка на производную таблицу:

```
CREATE VIEW VIEW1 (VC1,VC2) AS  
SELECT SUM(C1),C2 FROM T1  
GROUP BY C2;
```

```
SELECT MAX(VC1)  
FROM VIEW1;
```

Столбец VC1 встречается в качестве аргумента функции столбцов в операторе, ссылающемся на производную таблицу. Значения VC1 задаются при определении производной таблицы как результат применения функции столбцов SUM(C1) к группам после группировки базовой таблицы T1 по столбцу C2. Эквивалентного оператора SQL SELECT, который можно применить к базовой таблице T1 для получения того же результата, не существует, так как нельзя указать, что функции столбцов должны применяться последовательно.

### Два этапа материализации

В предыдущем примере DB2 материализует производную таблицу или табличное выражение; этот процесс выполняется в два этапа.

1. Подвыбор, определяющий производную таблицу или табличное выражение, применяется к базе данных, и результаты помещаются во временную копию таблицы результатов.
2. Затем оператор, ссылающийся на производную таблицу или табличное выражение, применяется к временной копии таблицы результатов, чтобы получить нужный результат.

Нужна ли материализация, зависит от свойств оператора, содержащего ссылку, или логически эквивалентного оператора, содержащего ссылку, получившегося в результате объединения, и свойств подвыбора, определяющего производную таблицу или табличное выражение.

### Когда материализуются производные таблицы и табличные выражения

Обычно DB2 использует материализацию, когда имеются ссылки на производную таблицу или табличное выражение и при этом используется групповая обработка (группировка, функции столбцов, DISTINCT) в подвыборе, а также в ссылающемся операторе или в операции объединения, в которой участвует производная таблица или табличное выражение. В Табл. 75 приведены случаи, когда используется материализация. DB2 может также использовать материализацию в операторах, содержащих несколько внешних объединений или внешние и внутренние объединения, или если слияние приводит к объединению более чем 15 таблиц.

Таблица 75 (Стр. 1 из 2). Случаи применения DB2 материализации производной таблицы или табличного выражения. "X" означает, что применяется материализация. Примечания приведены после таблицы.

В операторе SELECT FROM для производной таблицы или табличного выражения используется...(1)	В определении производной таблицы или табличного выражения используется...(2)			
	GROUP BY	DISTINCT	Функция столбцов	Функция столбцов с DISTINCT
Объединения (3)	X	X	X	X

Таблица 75 (Стр. 2 из 2). Случаи применения DB2 материализации производной таблицы или табличного выражения. "X" означает, что применяется материализация. Примечания приведены после таблицы.

В операторе <b>SELECT FROM</b> для производной таблицы или табличного выражения используется...(1)	В определении производной таблицы или табличного выражения используется...(2)			
	GROUP BY	DISTINCT	Функция столбцов	Функция столбцов с DISTINCT
GROUP BY	X	X	X	X
DISTINCT	—	X	—	X
Функция столбцов	X	X	X	X
Функция столбцов с DISTINCT	X	X	X	X
SELECT поднабор столбцов производной таблицы или табличного выражения	—	X	—	—

#### Примечания к Табл. 75 на стр. 746:

- Если производная таблица – назначение команд INSERT, UPDATE или DELETE, для разрешения ссылки на производную таблицу используется слияние. Производные таблицы, являющиеся назначением этих операторов, должны допускать изменение. В каком случае производные таблицы допускают только чтение, описано в разделе Глава 6 *DB2 SQL Reference*.

Оператор SQL может ссылаться на производную таблицу несколько раз, и для некоторых ссылок может использоваться слияние, а для других – материализация.

- Если список SELECT содержит переменную хоста в табличном выражении, происходит материализация. Например:

```
SELECT C1 FROM
  (SELECT :HV1 AS C1 FROM T1) X;
```

Если определение производной таблицы или вложенного табличного выражения содержит пользовательскую функцию, которая определена как NOT DETERMINISTIC или EXTERNAL ACTION, эта производная таблица или табличное выражение всегда материализуются.

- Дополнительные сведения о материализации при внешних объединениях:

- Если в производной таблице или табличном выражении есть условие WHERE, не содержащее столбцов, происходит материализация. Например:

```
SELECT X.C1 FROM
  (SELECT C1 FROM T1
   WHERE 1=1) X LEFT JOIN T2 Y
   ON X.C1=Y.C1;
```

- Если при полном внешнем объединении список SELECT для производной таблицы или вложенного табличного выражения не состоит из одного столбца, используемого в условии ON внешнего объединения, происходит материализация. Например:

```
SELECT X.C1 FROM
  (SELECT C1+10 AS C2 FROM T1) X FULL JOIN T2 Y
   ON X.C2=Y.C2;
```

- Если в списке SELECT для производной таблицы или вложенного табличного выражения нет столбцов, происходит материализация. Например:

```
SELECT X.C1 FROM
  (SELECT 1+2+HV1. AS C1 FROM T1) X LEFT JOIN T2 Y
  ON X.C1=Y.C1;
```

## Как с помощью EXPLAIN определить, когда происходит материализация

Для каждой материализованной ссылки на производную таблицу или табличное выражение в таблицу PLAN\_TABLE добавляются строки, описывающие пути доступа для обоих этапов процесса материализации. Это пути доступа для формирования промежуточного результата подвыбора производной таблицы и для доступа к этому промежуточному результату из оператора, содержащего ссылку. Подвыбор также может содержать ссылки на производные таблицы и табличные выражения, которые должны быть материализованы.

На материализацию производной таблицы указывает также имя производной таблицы или табличного выражения в поле TNAME в одной из строк, описывающих путь доступа для блока запроса со ссылкой. Когда DB2 выбирает слияние, EXPLAIN помещает в PLAN\_TABLE; только имена базовых таблиц производной таблицы или табличного выражения.

## Производительность слияния и материализации

Производительность слияния выше, чем материализации. При материализации DB2 использует для доступа к материализованному промежуточному результату просмотр табличного пространства. DB2 материализует производную таблицу или табличное выражение, только если она не может использовать слияние.

Как описано выше, материализация состоит из двух этапов, на первом из которых формируется промежуточный результат. Чем он меньше, тем эффективнее второй этап. Чтобы уменьшить размер промежуточного результата, DB2 пытается проверить определенные предикаты из условия WHERE ссылающегося оператора на первом этапе, а не на втором. Для этой цели подходят только определенные предикаты. Во-первых, предикат должен быть простым логическим термом. Во-вторых, он должен быть предикатом одного из видов, показанных в Табл. 76.

Таблица 76. Предикаты, которые можно проверить на первом этапе

Предикат	Пример
COL оп литерал	V1.C1 > hv1
COL IS (NOT) NULL	V1.C1 IS NOT NULL
COL (NOT) BETWEEN литерал AND литерал	V1.C1 BETWEEN 1 AND 10
COL (NOT) LIKE константа (ESCAPE константа)	V1.C2 LIKE 'p\%%' ESCAPE '\'

**Примечание:** Здесь "оп" – =, <>, >, <, <= или >=, а литерал – переменная хоста, константа или специальный регистр. Литералы в предикате BETWEEN могут быть разными.

Подразумеваемые предикаты, сгенерированные с помощью транзитивного замыкания, также можно проверить на первом этапе.

## Оценка стоимости оператора

Утилита EXPLAIN позволяет одновременно с заполнением таблицы PLAN\_TABLE заполнить таблицу `владелец.DSN_STATEMNT_TABLE`. DB2 дает оценки стоимости в служебных единицах и в миллисекундах для операторов SELECT, INSERT, UPDATE и DELETE, как статических, так и динамических. Оценки не учитывают некоторых факторов, в том числе влияния параллелизма, триггеров или пользовательских функций.

Информацию в таблице операторов можно использовать, чтобы:

- Определить операторы, которые не будут выполняться в соответствии со служебными требованиями и исправить их.

DB2 относит свои оценки стоимости к двум *категориям стоимости*: категории А и категории В. К категории А относятся оценки, для получения которых достаточно информации. Они могут быть точными не на 100%, однако они, как правило, точнее оценок из категории В.

К категории В DB2 относит оценки, полученные с использованием значений по умолчанию, например, когда нет статистики или когда в запросе есть переменные хоста. Более подробно о том, как DB2 определяет категорию стоимости, говорится в описании столбца REASON в Табл. 77 на стр. 750.

- Предоставить системному программисту основные значения в служебных единицах для ограничения динамических операторов.

О прогностическом ограничении можно прочитать в разделе Раздел 5 (Том 2) *DB2 Administration Guide*.

В этом разделе описываются следующие задачи по получению и использованию оценок стоимости с помощью утилиты EXPLAIN:

1. “Создание таблицы операторов”
2. “Заполнение и поддержка таблицы операторов” на стр. 751
3. “Получение строк из таблицы операторов” на стр. 751
4. “Смысл категорий стоимости” на стр. 752

Смотрите в разделе Раздел 7 книги *Руководство по прикладному программированию и языку SQL для DB2*, как изменить прикладную программу, чтобы обрабатывать коды возврата SQL, относящиеся к прогностическому ограничению.

## Создание таблицы операторов

Чтобы получить информацию о предполагаемой стоимости оператора, создайте таблицу под именем DSN\_STATEMNT\_TABLE, в которую утилита EXPLAIN поместит результаты. Набор операторов для создания таблицы есть в примере библиотеки DB2 в члене DSNTESC.

На рис. 179 на стр. 750 показан формат таблицы операторов.

```

CREATE TABLE DSN_STATEMNT_TABLE
( QUERYNO          INTEGER      NOT NULL WITH DEFAULT,
  APPLNAME         CHAR(8)     NOT NULL WITH DEFAULT,
  PROGNAME         CHAR(8)     NOT NULL WITH DEFAULT,
  COLLID           CHAR(18)    NOT NULL WITH DEFAULT,
  GROUP_MEMBER    CHAR(8)     NOT NULL WITH DEFAULT,
  EXPLAIN_TIME    TIMESTAMP   NOT NULL WITH DEFAULT,
  STMT_TYPE        CHAR(6)     NOT NULL WITH DEFAULT,
  COST_CATEGORY   CHAR(1)     NOT NULL WITH DEFAULT,
  PROCMS           INTEGER     NOT NULL WITH DEFAULT,
  PROCSU           INTEGER     NOT NULL WITH DEFAULT,
  REASON           VARCHAR(254) NOT NULL WITH DEFAULT);

```

*Рисунок 179. Формат таблицы DSN\_STATEMNT\_TABLE*

В Табл. 77 показано содержимое каждого столбца. Первые пять столбцов DSN\_STATEMNT\_TABLE такие же, как в PLAN\_TABLE.

*Таблица 77 (Стр. 1 из 2). Описания столбцов DSN\_STATEMNT\_TABLE*

Имя столбца	Описание														
QUERYNO	Число–идентификатор описываемого столбца. Описание столбца QUERYNO смотрите в Табл. 68 на стр. 704. Если QUERYNO не уникально, уникально значение EXPLAIN_TIME.														
APPLNAME	Имя плана прикладной программы для данной строки или пустое значение. Смотрите описание столбца APPLNAME в Табл. 68 на стр. 704.														
PROGNAME	Имя программы или пакета, содержащих оператор или пустое значение. Смотрите описание столбца PROGNAME в Табл. 68 на стр. 704.														
COLLID	ID собрания для пакета, или пустое значение. Смотрите описание столбца COLLID в Табл. 68 на стр. 704.														
GROUP_MEMBER	Имя члена DB2, выполнившего EXPLAIN, или пустое значение. Смотрите описание столбца GROUP_MEMBER в Табл. 68 на стр. 704.														
EXPLAIN_TIME	Время, когда выполнялся оператор. Это время аналогично времени в столбце BIND_TIME таблицы PLAN_TABLE.														
STMT_TYPE	Тип оператора. Возможны следующие значения: <table> <tr> <td><b>SELECT</b></td> <td>SELECT</td> </tr> <tr> <td><b>INSERT</b></td> <td>INSERT</td> </tr> <tr> <td><b>UPDATE</b></td> <td>UPDATE</td> </tr> <tr> <td><b>DELETE</b></td> <td>DELETE</td> </tr> <tr> <td><b>SELUPD</b></td> <td>SELECT с FOR UPDATE OF</td> </tr> <tr> <td><b>DELCUR</b></td> <td>DELETE WHERE CURRENT OF CURSOR</td> </tr> <tr> <td><b>UPDCUR</b></td> <td>UPDATE WHERE CURRENT OF CURSOR</td> </tr> </table>	<b>SELECT</b>	SELECT	<b>INSERT</b>	INSERT	<b>UPDATE</b>	UPDATE	<b>DELETE</b>	DELETE	<b>SELUPD</b>	SELECT с FOR UPDATE OF	<b>DELCUR</b>	DELETE WHERE CURRENT OF CURSOR	<b>UPDCUR</b>	UPDATE WHERE CURRENT OF CURSOR
<b>SELECT</b>	SELECT														
<b>INSERT</b>	INSERT														
<b>UPDATE</b>	UPDATE														
<b>DELETE</b>	DELETE														
<b>SELUPD</b>	SELECT с FOR UPDATE OF														
<b>DELCUR</b>	DELETE WHERE CURRENT OF CURSOR														
<b>UPDCUR</b>	UPDATE WHERE CURRENT OF CURSOR														
COST_CATEGORY	Показывает, пришлось ли DB2 использовать значения по умолчанию при получении оценок. Возможные значения: <table> <tr> <td><b>A</b></td> <td>Показывает, что у DB2 было достаточно информации, чтобы оценить стоимость, не пользуясь значениями по умолчанию.</td> </tr> <tr> <td><b>B</b></td> <td>Показывает, что определенные причины заставили DB2 использовать значения по умолчанию. Посмотрите значение в поле REASON, чтобы определить, почему DB2 не смогла получить оценку категории A.</td> </tr> </table>	<b>A</b>	Показывает, что у DB2 было достаточно информации, чтобы оценить стоимость, не пользуясь значениями по умолчанию.	<b>B</b>	Показывает, что определенные причины заставили DB2 использовать значения по умолчанию. Посмотрите значение в поле REASON, чтобы определить, почему DB2 не смогла получить оценку категории A.										
<b>A</b>	Показывает, что у DB2 было достаточно информации, чтобы оценить стоимость, не пользуясь значениями по умолчанию.														
<b>B</b>	Показывает, что определенные причины заставили DB2 использовать значения по умолчанию. Посмотрите значение в поле REASON, чтобы определить, почему DB2 не смогла получить оценку категории A.														

Таблица 77 (Стр. 2 из 2). Описания столбцов DSN\_STATEMNT\_TABLE

Имя столбца	Описание	
PROCMS	Оценка стоимости процессорного времени в миллисекундах для оператора SQL. Оценка округляется вверх до ближайшего целого числа. Максимальное значение – 2147483647 миллисекунд, что приблизительно равно 24,8 дням. Если оценка выше этого максимума, проставляется максимальное значение.	
PROCSU	Оценка стоимости процессорного времени в служебных единицах для оператора SQL. Оценка округляется вверх до ближайшего целого числа. Максимальное значение – 2147483647 служебных единиц. Если оценка выше этого максимума, проставляется максимальное значение.	
REASON	Строка, в которой указывается причина, по которой оценка стоимости была отнесена к категории В.  <b>HOST VARIABLES</b>  <b>TABLE CARDINALITY</b>  <b>UDF</b>  <b>TRIGGERS</b>  <b>REFERENTIAL CONSTRAINTS</b>	В операторе используются переменные хоста, маркеры параметров или специальные регистры.  Отсутствует статистика мощности для одной или нескольких таблиц, используемых в операторе.  В операторе используются пользовательские функции.  Для таблицы назначения оператора INSERT, UPDATE или DELETE определены триггеры.  Для таблицы назначения оператора DELETE существуют реляционные связи типа CASCADE или SET NULL.

## Заполнение и поддержка таблицы операторов

Таблица операторов заполняется одновременно с заполнением соответствующей таблицы планов. Более подробные сведениясмотрите в разделе “Заполнение и поддержание таблицы планов” на стр. 709.

Как и в случае таблицы планов, DB2 только добавляет строки в таблицу операторов и не удаляет строки автоматически. Триггеры INSERT не активируются, за исключением случаев, когда вы сами вставляете строки оператором SQL INSERT.

Чтобы очистить таблицу от устаревших строк, как и при работе с любой другой таблицей, используйте оператор DELETE. Можно также использовать DROP TABLE, чтобы полностью отбросить таблицу операторов.

## Получение строк из таблицы операторов

Чтобы получить все строки таблицы операторов, можно использовать следующий запрос, который возвратит все строки, относящиеся к оператору с номером запроса 13:

```
SELECT * FROM JOE.DSN_STATEMNT_TABLE  
WHERE QUERYNO = 13;
```

Столбцы QUERYNO, APPLNAME, PROGNAME, COLLID и EXPLAIN\_TIME содержат те же значения, что и соответствующие столбцы таблицы

PLAN\_TABLE для данного плана. Можно объединить таблицы планов и операторов по этим столбцам:

```
SELECT A.* , PROCMS , COST_CATEGORY
  FROM JOE.PLAN_TABLE A, JOE.DSN_STATEMNT_TABLE B
 WHERE A.APPLNAME = 'APPL1' AND
       A.APPLNAME = B.APPLNAME AND
       A.PROGNAME = B.PROGNAME AND
       A.COLLID    = B.COLLID AND
       A.BIND_TIME = B.EXPLAIN_TIME
 ORDER BY A.QUERYNO, A.QBLOCKNO, A.PLANNO, A.MIXOPSEQ;
```

## Смысл категорий стоимости

Категории стоимости – способ, которым DB2 отличает оценки, для получения которых было достаточно информации, от оценок, для которых информации было недостаточно. Вы вряд ли будете тратить много времени, чтобы отлаживать запрос, основываясь на оценках категории В, потому что реальная стоимость может сильно различаться в зависимости от значения переменной хоста или числа уровней вложенных триггеров или пользовательских функций.

Так же и системные администраторы при заполнении таблицы ограничения ресурсов (для прогностического или реактивного ограничения), видимо, воспримут оценки категории В скорее ориентировочно по сравнению с более определенными оценками категории А.

Из-за неопределенности операторы категории В хорошо подходят для реактивного ограничения.

**Что попадает в категорию В?** DB2 относит оценку стоимости оператора в категорию стоимости В, если выполняются следующие условия:

- В операторе есть пользовательские функции
- Для таблицы назначения определены триггеры:
  - Триггеры вставки в случае оператора INSERT
  - Триггеры изменения в случае оператора UPDATE
  - Триггеры удаления в случае оператора DELETE
- Для таблицы назначения оператора удаления определены реляционные связи как для родительской таблицы, а правила удаления – либо CASCADE, либо SET NULL
- Предикат в условии WHERE имеет одну из следующих форм:
  - COL op литерал, где литерал – переменная хоста, маркер параметра или специальный регистр. Операция (оп) – >, >=, <, <=, LIKE или NOT LIKE.
  - COL BETWEEN литерал AND литерал, где каждый литерал – переменная хоста, маркер параметра или специальный регистр
  - LIKE с условием ESCAPE, содержащим переменную хоста
- Отсутствует статистика мощности для одной или нескольких таблиц, используемых в таблице.

**Что попадает в категорию А?** DB2 относит к категории А все оценки, которые не попадают в категорию В.

---

## Глава 7–5. Параллельные операции и производительность запросов

Когда система DB2 планирует обращение к данным из таблицы или индекса в распределенном табличном пространстве, она может запустить несколько параллельных операций. Так можно существенно сократить время выполнения запросов, обрабатывающих много данных или требующих большой загрузки процессора.

Параллелизм ввода–вывода для запроса управляет одновременно выполняемыми требованиями ввода–вывода для одного запроса, в параллельном режиме читающими страницы в пул буферов. Такой тип обработки может значительно улучшить производительность выполнения запросов, скорость выполнения которых определяется операциями ввода–вывода. Параллелизм ввода–вывода используется только тогда, когда нельзя использовать какой–либо другой режим параллелизма.

Параллелизм процессоров для запроса обеспечивает действительную многозадачность при выполнении запроса. Запрос большого объема можно разбить на несколько меньших запросов. Эти меньшие запросы выполняются одновременно на нескольких процессорах, обращающихся к данным в параллельном режиме. Это уменьшает время выполнения запроса.

Чтобы еще более увеличить производительность обработки запросов с большим числом операций процессора, система DB2 может поделить выполнение большого запроса между разными подсистемами DB2 в группе совместного использования данных. Этот метод называется параллелизмом запросов Sysplex. Дополнительную информацию об этом методе смотрите в разделе Глава 7 книги *DB2 Data Sharing: Planning and Administration*.

DB2 может использовать параллельные операции для обработки:

- Статических и динамических запросов.
- Обращений к локальным и удаленным данным.
- Запросов, использующих операции просмотра одной таблицы и объединения нескольких таблиц.
- Обращений к данным посредством индексов с использованием просмотра табличного пространства или предварительной выборки списка.
- Операций сортировки.

Параллельные операции обычно работают по крайней мере с одной таблицей в распределенном табличном пространстве. Наибольший выигрыш в производительности достигается при просмотре больших распределенных табличных пространств, поскольку и операции ввода–вывода, и операции центрального процессора (СР) могут выполняться в параллельном режиме.

**Параллелизм для многораздельных и однораздельных табличных пространств:** Параллелизм запросов может давать выигрыш как для многораздельных, так и для однораздельных табличных пространств. Теперь при параллелизме могут использоваться индексы, не являющиеся индексами кластеризации. Это значит, что доступ к таблице может производиться в параллельном режиме, если программа связана с опцией ANY и обращение к таблице происходит через такие индексы.

В этой главе рассматриваются следующие темы:

- “Сравнение методов параллелизма”
- “Разрешение параллельной обработки” на стр. 757
- “Когда параллелизм не используется” на стр. 758
- “Интерпретация выходных данных EXPLAIN” на стр. 759
- “Настройка параллельной обработки” на стр. 762
- “Запрещение параллелизма выполнения запроса” на стр. 763

---

## Сравнение методов параллелизма

На рисунках в этом разделе показаны методы параллелизма в сравнении с последовательной предварительной выборкой и в сравнении друг с другом. Для всех трех методов подразумевается, что производится обращение к табличному пространству с тремя разделами – P1, P2 и P3. P1, P2 и P3 – это обозначения разделов табличного пространства. R1, R2, R3 и т.д. – это обозначение требований для последовательной предварительной выборки. Используются также их комбинации, например, P2R1 означает первое требование для раздела 2.

На рис. 180 показана **последовательная обработка**. При последовательной обработке DB2 обрабатывает три раздела по порядку, переходя к разделу 2 после завершения работы с разделом 1 и переходя к разделу 3 после завершения работы с разделом 2. Последовательная предварительная выборка позволяет совмещать работу процессора с операциями ввода–вывода, но операции ввода–вывода не могут выполняться одновременно друг с другом. В примере на рис. 180 для выполнения требования на предварительную выборку уходит больше времени, чем на обработку данных. Процессор часто ожидает выполнения операций ввода–вывода.

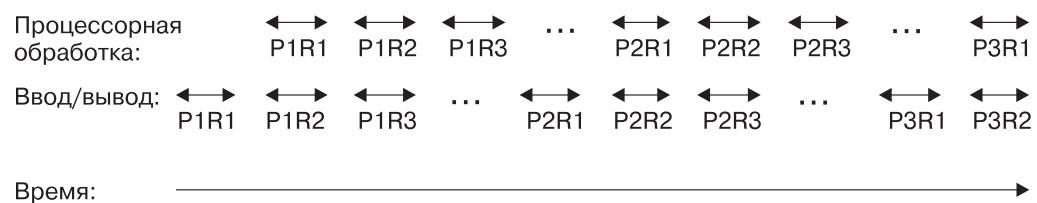
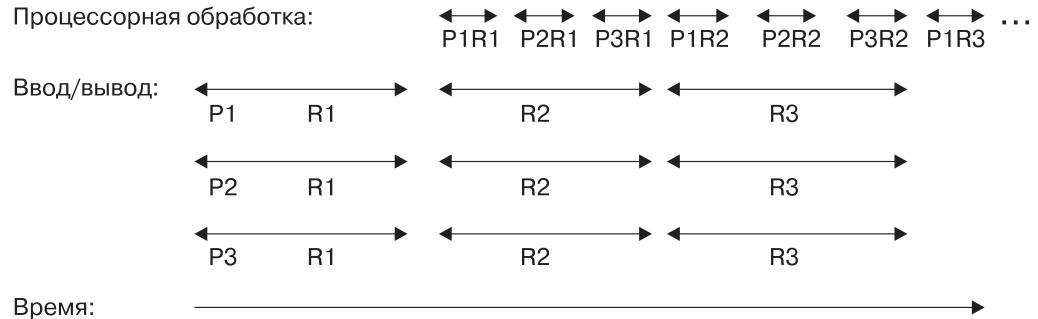


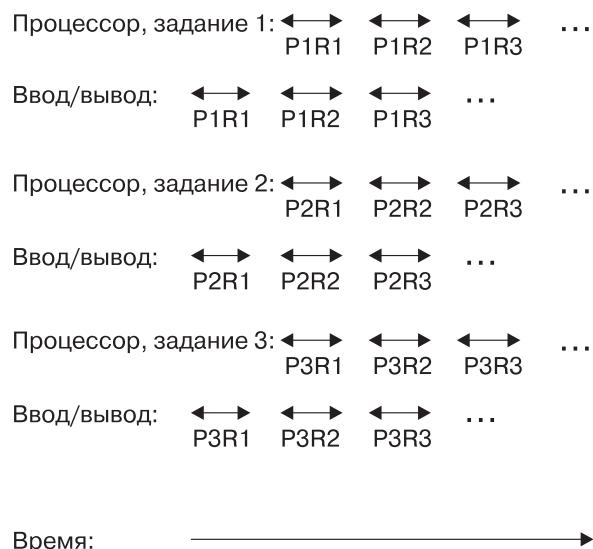
Рисунок 180. Методы выполнения операций процессора и операций ввода–вывода. Последовательная обработка.

На рис. 181 на стр. 755 показаны **параллельные операции ввода–вывода**. При использовании параллельных операций ввода–вывода DB2 одновременно выбирает данные из трех разделов. Процессор выполняет первое требование для каждого раздела, затем второе требование для каждого раздела и т.д. Процессор не ожидает выполнения операций ввода–вывода, но используется только одно задание обработки.



*Рисунок 181. Методы выполнения операций процессора и операций ввода–вывода. Параллельные операции ввода–вывода.*

На рис. 182 показаны **параллельные операции процессоров**. При параллелизме процессоров DB2 может для обработки запроса использовать несколько параллельных заданий. Три одновременно работающих задания могут сильно уменьшить затраты времени на выполнение запросов, обрабатывающих много данных или требующих большой загрузки процессора. Тот же принцип используется и для **параллелизма запросов Sysplex**, но в этом случае работа может выполняться и не на одном центральном процессорном комплексе.



*Рисунок 182. Методы выполнения операций процессора и операций ввода–вывода. Выполнение запроса с использованием параллелизма процессоров. Задания могут выполняться на одном центральном процессорном комплексе или на нескольких членах группы совместного использования данных.*

**Запросы, для которых можно достичь выигрыша при использовании параллельных операций:** Использование параллельных операций может принести выгоду для следующих запросов:

- Запросы, при выполнении которых DB2 тратит большую часть времени на выборку страниц – запросы с большим количеством операций ввода–вывода.

Типичный запрос с большим количеством операций ввода–вывода выглядит примерно так (подразумевается, что при просмотре табличного пространства используется много страниц):

```
SELECT COUNT(*) FROM ACCOUNTS  
  WHERE BALANCE > 0 AND  
    DAYS_OVERDUE > 30;
```

- Запросы, при выполнении которых DB2 для обработки строк требуется много времени процессора, а также, возможно, времени на выполнение операций ввода–ввода . Сюда входят:
  - *Запросы с интенсивным просмотром данных и высокой избирательностью.* Для выполнения таких запросов необходимо просмотреть большой объем данных, но относительно малое число строк удовлетворяет условию поиска.
  - *Запросы, содержащие функции столбцов.* Функции столбцов (такие как MIN, MAX, SUM, AVG и COUNT) обычно требуют просмотра большого количества данных, но возвращают только один общий результат.
  - *Запросы, обращающиеся к длинным строкам данных.* Такие запросы обращаются к таблицам с длинными строками и число строк на страницу очень мало (например, одна строка в странице).
  - *Запросы, требующие больших затрат времени центрального процессора.* Это могут быть запросы только для чтения сложной структуры, обрабатывающие много данных или включающие сортировку.

Типичный запрос, требующий больших затрат времени процессора:

```
SELECT MAX(QTY_ON_HAND) AS MAX_ON_HAND,  
       AVG(PRICE) AS AVG_PRICE,  
       AVG(DISCOUNTED_PRICE) AS DISC_PRICE,  
       SUM(TAX) AS SUM_TAX,  
       SUM(QTY SOLD) AS SUM_QTY_SOLD,  
       SUM(QTY_ON_HAND - QTY_BROKEN) AS QTY_GOOD,  
       AVG(DISCOUNT) AS AVG_DISCOUNT,  
       ORDERSTATUS,  
       COUNT(*) AS COUNT_ORDERS  
  FROM ORDER_TABLE  
 WHERE SHIPPER = 'OVERNIGHT' AND  
       SHIP_DATE < DATE('1996-01-01')  
 GROUP BY ORDERSTATUS  
 ORDER BY ORDERSTATUS;
```

**Терминология:** Когда термин *задание* используется при описании параллельной обработки, необходимо учитывать контекст. Если для запросов используется параллелизм процессоров или параллелизм Sysplex, задание будет представлять собой единицу выполнения MVS, используемую для обработки запроса. Если используется параллелизм операций ввода–вывода, термин "задание" будет обозначать просто обработку одного из параллельных потоков ввода–вывода.

**Параллельная группа** – это термин, используемый для обозначения конкретного набора параллельных операций (параллельных заданий или параллельных операций ввода–вывода). Для запроса может существовать

несколько параллельных групп, но каждая параллельная группа внутри запроса идентифицируется по ее уникальному номеру ID.

**Степень параллелизма** – это число параллельных заданий или операций ввода–вывода, используемых DB2 для операций параллельной группы.

## Разрешение параллельной обработки

Запросы могут использовать преимущества параллелизма, только если параллельная обработка была разрешена. Чтобы разрешить параллельную обработку:

- Для **статических операторов SQL** задайте опцию DEGREE(ANY) в команде BIND или REBIND. Эта опция связывания влияет только на статические операторы SQL и не разрешает параллелизм для динамических операторов.
- Для **динамических операторов SQL** задайте для специального регистра CURRENT DEGREE значение 'ANY'. Значение этого специального регистра влияет только на динамические операторы. Оно не влияет на статические операторы SQL. Также нужно убедиться, что параллелизм не запрещен в RLST для плана, пакета или ID авторизации. Для задания значения этого специального регистра можно использовать следующий оператор SQL:

```
SET CURRENT DEGREE='ANY';
```

Можно также изменить значение по умолчанию для этого специального регистра с 1 на ANY для всей подсистемы DB2, изменив значение поля CURRENT DEGREE панели установки DSNTIP4.

- При связывании с уровнем изоляции CS выберите также опцию CURRENTDATA(NO), если это возможно. Эта опция позволяет улучшить производительность в целом, а также гарантирует, что DB2 будет рассматривать использование параллелизма для неоднозначных указателей. Если при связывании была задана опция CURRENTDATA(YES) и DB2 не может определить, что указатель используется только для чтения, DB2 не использует параллелизм. Лучше всего всегда явно указывать, когда указатель используется только для чтения, задавая FOR FETCH ONLY или FOR READ ONLY в операторе DECLARE CURSOR.
- Значение порога параллелизма виртуального пула буферов (VPPSEQT) должно быть достаточно велико, чтобы обеспечить достаточный объем пула буферов для параллельной обработки. Описание пула буферов и их порогов смотрите в Разделе 5 (Том 2) руководства *DB2 Administration Guide*.

Если параллельная обработка разрешена и DB2 оценивает затраты ресурсов ввода–вывода и процессора для данного запроса как высокие, она может запустить несколько параллельных заданий, если предполагает, что в результате этого время выполнения может быть уменьшено.

| **Специальные требования для параллелизма процессора:** DB2 должна выполнять на центральном процессорном комплексе, содержащем два или несколько тесно связанных процессоров (иногда называемых центральными процессорами или СР). Если при связывании запроса активен только один

процессор, DB2 может использовать только параллельные операции ввода—вывода.

DB2 может также использовать только параллельные операции ввода—вывода, если указатель объявлен с WITH HOLD и при связывании задан уровень изоляции RR или RS. Информацию о дополнительных ограничениях на параллельностьсмотрите в разделе Табл. 78.

Для сложных запросов при запуске в параллельном режиме на члене группы совместного использования данных с параллелизмом запросов Sysplex используйте мощности группы совместного использования данных для обработки отдельных сложных запросов на нескольких членах группы. Дополнительную информацию о том, как использовать мощности группы совместного использования данных для обработки сложных запросов,смотрите в разделе Глава 7 *DB2 Data Sharing: Planning and Administration*.

## Когда параллелизм не используется

Параллелизм используется не для всех запросов; для некоторых путей доступа нет смысла в дополнительных расходах на параллелизм. При выборке из временной таблицы параллелизм для нее также не используется. Обнаружив, что параллелизм не используется, обратитесь к разделу Табл. 78, чтобы проверить, не использует ли ваш запрос какой-либо из путей доступа, для которых параллелизм не разрешен.

Таблица 78 (Стр. 1 из 2). Контрольный список для режимов параллельности и ограничений на запросы

Если запрос использует...	Будет ли разрешен параллелизм?			
	В-в	Центр. проц.	Sysplex	Комментарии
Доступ через список RID (предварительная выборка списка и множественный доступ к индексам)	Да	Да	Нет	Указывается значением "L" в столбце PREFETCH в PLAN_TABLE или значениями M, MX, MI или MQ в столбце ACESSTYPE в PLAN_TABLE.
Запросы, возвращающие значения больших объектов	Да	Да	Нет	
Объединение просмотров слияния для нескольких столбцов	Нет	Нет	Нет	
Запросы, в которых задан прямой доступ к столбцам	Нет	Нет	Нет	Указывается значением D в столбце PRIMARY_ACCESS_TYPE в PLAN_TABLE
Выражения с материализованными производными таблицами или материализованными вложенными таблицами во время использования.	Нет	Нет	Нет	

Таблица 78 (Стр. 2 из 2). Контрольный список для режимов параллельности и ограничений на запросы

Если запрос использует...	Будет ли разрешен параллелизм?			
	В-в	Центр. проц.	Sysplex	Комментарии
EXISTS в предикате WHERE	Нет	Нет	Нет	

**Если разрешен параллелизм, DB2 не использует некоторые гибридные объединения:** Чтобы гарантировать возможность использования преимуществ параллелизма, DB2 не использует один из типов гибридных объединений (SORTN\_JOIN=Y), если план или пакет связаны с опцией CURRENT DEGREE=ANY или если специальный регистр CURRENT DEGREE имеет значение 'ANY'.

**Влияние нераспределенных индексов:** DB2 не использует параллелизм, если для первой базовой таблицы в параллельной группе используется прямой доступ через нераспределенный индекс.

**Разъяснения для доступу через список IN:** DB2 может использовать параллелизм, только если доступ через список условия IN используется для внутренней таблицы параллельной группы. Например, предположим, что в следующем запросе для объединения таблиц T1 и T2 используется объединение с вложенным циклом. Для доступа через список IN для T2 может использоваться параллелизм:

```
SELECT COUNT(*)
  FROM T1, T2
 WHERE T1.C1 = T2.C1 AND
       T1.C2 > 5      AND
       T2.C2  IN ( 6 , 7 , 9 ) ;
```

## Интерпретация выходных данных EXPLAIN

Чтобы понять, как DB2 планирует использование параллелизма, посмотрите выходные данные в таблице PLAN\_TABLE. (Все столбцы таблицы PLAN\_TABLE подробно описаны в Табл. 68 на стр. 704.) В этом разделе описывается, как по столбцам таблицы PLAN\_TABLE проверить использование параллелизма, и даны некоторые примеры.

## Как по столбцам таблицы PLAN\_TABLE проверить использование параллелизма

Чтобы по выходным данным проверить использование параллелизма, выполните следующие действия:

### 1. Определите, планирует ли DB2 использовать параллелизм:

Ненулевое значение столбца ACCESS\_DEGREE или JOIN\_DEGREE для каждого блока запроса (QBLOCKNO) в запросе (QUERYNO) означает, что планируется какая-то степень параллелизма.

### 2. Определите параллельные группы в запросе:

Все операции (PLANNO) с одинаковыми значениями ACCESS\_PGROUP\_ID, JOIN\_PGROUP\_ID, SORTN\_PGROUP\_ID или

SORTC\_PGROUP\_ID входят в одну параллельную группу. Обычно в такой набор операций содержит различные типы методов объединения и операций сортировки. ID параллельных групп могут находиться в тех же строках выходной таблицы PLAN\_TABLE или в других строках, в зависимости от выполняемой операции. Это поясняется на примерах в разделе “Примеры таблицы PLAN\_TABLE, показывающей использование параллелизма.”

### 3. Определите режим параллелизма:

В столбце PARALLELISM\_MODE указывается планируемый режим параллелизма (I, C или X, т.е. параллелизм ввода–вывода, процессоров или Sysplex). Внутри блока запроса не могут смешиваться режимы параллелизма “I” и “C.” Однако операторы, использующие несколько блоков запроса (например, UNION) могут иметь режим параллелизма “I” для одного блока запроса и “C” – для другого. Режимы параллелизма “C” и “X” могут смешиваться в одном блоке запроса, но не в одной параллельной группе.

Если при связывании оператора эта система DB2 была членом группы совместного использования данных, столбец PARALLELISM\_MODE может содержать значение “X,” даже если активен только этот один член группы DB2. Это позволяет DB2 использовать преимущества дополнительной мощности процессоров, которые могут быть доступны во время выполнения. Если другие члены группы недоступны во время выполнения, DB2 выполняет этот запрос на одном члене группы DB2.

## Примеры таблицы PLAN\_TABLE, показывающей использование параллелизма

Другие значения в этих примерах не зависят от значения PARALLELISM\_MODE – I, C или X.

- **Пример 1: обращение к одной таблице**

Предположим, что во время связывания DB2 решает запустить три одновременных требования для получения данных из таблицы T1. Часть таблицы PLAN\_TABLE будет выглядеть следующим образом. Если DB2 решает не использовать параллельные операции для какого–то шага, столбцы ACCESS\_DEGREE и ACCESS\_PGROUP\_ID будут содержать пустые значения.

TNAME	METHOD	ACCESS_DEGREE	ACCESS_PGROUP_ID	JOIN_DEGREE	JOIN_PGROUP_ID	SORTC_PGROUP_ID	SORTN_PGROUP_ID
T1	0	3	1	(null)	(null)	(null)	(null)

- **Пример 2: объединение с вложенным циклом**

Рассмотрим запрос, создающий серию объединений с вложенным циклом для трех таблиц T1, T2 и T3. T1 – это самая внешняя таблица, а T3 – самая внутренняя. Во время связывания DB2 решает запустить три одновременных требования для получения данных из каждой из этих трех таблиц. Для метода объединения с вложенным циклом все операции получения данных будут входить в одну параллельную группу. Часть таблицы PLAN\_TABLE будет выглядеть так:

TNAME	METHOD	ACCESS_DEGREE	ACCESS_PGROUP_ID	JOIN_DEGREE	JOIN_PGROUP_ID	SORTC_PGROUP_ID	SORTN_PGROUP_ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	1	3	1	3	1	(null)	(null)
T3	1	3	1	3	1	(null)	(null)

- **Пример 3: объединение просмотров слияния**

Рассмотрим запрос, создающий объединение просмотров слияния для двух таблиц T1 и T2. Во время связывания DB2 решает запустить три одновременных требования для T1 и шесть одновременных требований для T2. Операции просмотра и сортировки для таблицы T1 будут входить в одну параллельную группу. Операции просмотра и сортировки для таблицы T2 будут входить в другую параллельную группу. Кроме того, фазы слияния могут в принципе выполняться параллельно. При этом третья параллельная группа используется для запуска трех одновременных требований для каждой промежуточной отсортированной таблицы. Часть таблицы PLAN\_TABLE будет выглядеть так:

TNAME	METHOD	ACCESS_DEGREE	ACCESS_PGROUP_ID	JOIN_DEGREE	JOIN_PGROUP_ID	SORTC_PGROUP_ID	SORTN_PGROUP_ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	2	6	2	3	3	1	2

- **Пример 4: гибридное объединение**

Рассмотрим запрос, создающий гибридное объединение для двух таблиц T1 и T2. Кроме того, необходимо отсортировать таблицу T1; в итоге строка T2 таблицы PLAN\_TABLE содержит SORTC\_JOIN=Y. Во время связывания DB2 решает запустить три одновременных требования для T1 и шесть одновременных требований для T2. Параллельные операции используются для объединения с использованием индекса кластеризации таблицы T2.

Поскольку RID таблицы T2 могут быть получены при помощи одновременных требований к распределенному индексу, фаза объединения выполняется параллельно. Операции получения RID таблицы T2 и строк таблицы T2 входят в одну параллельную группу. Часть таблицы PLAN\_TABLE будет выглядеть так:

TNAME	METHOD	ACCESS_DEGREE	ACCESS_PGROUP_ID	JOIN_DEGREE	JOIN_PGROUP_ID	SORTC_PGROUP_ID	SORTN_PGROUP_ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	4	6	2	6	2	1	(null)

## Настройка параллельной обработки

Значительная часть информации в этом разделе относится также к параллелизму запросов Sysplex. Дополнительную информацию смотрите в Главе 7 руководства *DB2 Data Sharing: Planning and Administration*.

Параллельная группа может выполняться с меньшей степенью параллельности, чем указано в выходных данных в таблице PLAN\_TABLE. На уменьшение степени параллелизма могут влиять следующие причины:

- Степень доступности пула буферов
- Логический конфликт.

Рассмотрим объединение с вложенным циклом. Внутренняя таблица может находиться в распределенном или нераспределенном табличном пространстве, но DB2 с большей вероятностью будет использовать параллельную операцию объединения, если внешняя таблица является распределенной.

- Физические конфликты
- Переменные хоста во время выполнения

Положение разделов таблицы для данного запроса может задаваться переменной хоста. В таких случаях DB2 откладывает планирование степени параллелизма до времени выполнения, когда будет известно значение этой переменной хоста.

- Обновляемый указатель

Во время выполнения DB2 может определить, что неоднозначный указатель может быть обновляемым.

- Изменение в конфигурации активных процессоров

Если во время выполнения активно меньшее число процессоров, DB2 может изменить степень параллельности.

### **Особенности блокировок для прикладных программ с уровнем изоляции многократное чтение:**

При использовании параллелизма процессоров каждое задание получает блокировки независимо. Учтите, что это может привести к увеличению общего числа блокировок для прикладных программ, которые:

- Используют уровень изоляции многократное чтение
- Используют параллелизм процессоров
- Многократно обращаются к табличному пространству, используя режим блокировки IS и не выполняя операции COMMIT

Для всех прикладных программ с изоляцией многократное чтение рекомендуется часто выполнять операции COMMIT, чтобы освобождать удерживаемые блокировками ресурсы. Уровни изоляции многократное чтение и стабильность чтения нельзя использовать с параллелизмом запросов Sysplex.

---

## Запрещение параллелизма выполнения запроса

Чтобы запретить параллельные операции, выполните какое-либо из следующих действий:

- Для статических операторов SQL выполните повторное связывание, изменив опцию DEGREE(ANY) на DEGREE(1). Чтобы сделать это, можно использовать панели DB2I, подкоманды DSN или DSNH CLIST. Значение по умолчанию – DEGREE(1).
- Для динамических операторов SQL выполните следующий оператор SQL:

```
SET CURRENT DEGREE = '1';
```

Значение по умолчанию для специального регистра CURRENT DEGREE равно 1, если это значение по умолчанию не было изменено для данной системы.

Для запрещения параллелизма можно также использовать управляющие элементы системы. Этот вариант описан в Разделе 5 (Том 2) руководства *DB2 Administration Guide*.



---

## **Глава 7–6. Программирование для утилиты ISPF (Interactive System Productivity Facility – интерактивная утилита производительности системы)**

Утилита ISPF помогает проектировать и выполнять диалоги. В DB2 имеется прикладная программа примера, которая демонстрирует, как использовать ISPF через утилиту подключения по вызову CAF. Инструкции по компиляции, печати и использованию этой программы приводятся в *Разделе 2 DB2 Installation Guide*. В этой главе описывается, как строить прикладные программы для использования вместе с ISPF.

В следующих разделах обсуждаются сценарии взаимодействия между вашей программой, DB2 и ISPF. С точки зрения эффективности, легкости написания и обслуживания и общей гибкости, у каждого из них есть свои достоинства и недостатки.

---

### **Использование ISPF и командного процессора DSN**

В любой структуре существуют определенные ограничения на способ соединения и рассоединения с DB2. Если используется опция PGM ISPF SELECT, ISPF передает управление вашему модулю загрузки с помощью макрокоманды LINK; если используется CMD, ISPF передает управление с помощью макрокоманды ATTACH.

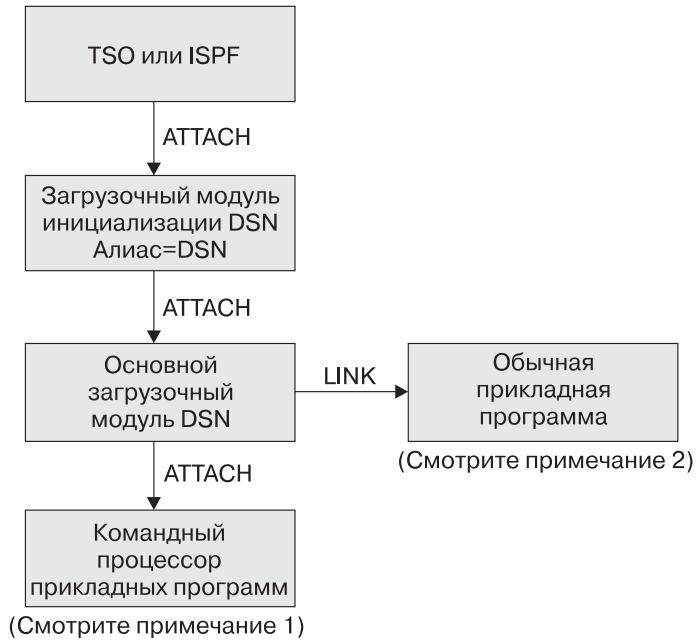
Командный процессор DSN (смотрите раздел “Командный процессор DSN” на стр. 465) допускает только соединения с одним блоком управления задачами (TCB – task control block). Проследите за тем, чтобы TCB не был изменен после выполнения первого оператора SQL. Службы ISPF SELECT изменяют TCB, если DSN был запущен под ISPF, поэтому с их помощью нельзя передавать управление от одного модуля загрузки к другому. Вместо этого надо использовать LINK, XCTL или LOAD.

На рис. 183 на стр. 766 показаны блоки управления задачами, получаемые в результате подключения командного процессора DSN под TSO или ISPF.

Если вы находитесь в ISPF и работаете под DSN, можно выполнить ISPLINK на другую программу, которая вызывает CLIST. В свою очередь, CLIST использует DSN и другую прикладную программу. При каждом таком использовании DSN в DB2 создается отдельная единица восстановления (процесс или транзакция).

Все инициированные таким образом единицы работы DSN не связаны между собой по отношению изоляции (блокировки) и восстановления (принятия). При этом возможна тупиковая ситуация между вашими собственными программами; так, одна из единиц (DSN) может затребовать последовательный ресурс (например, страницу данных), которым другая единица (DSN) владеет в монопольном режиме.

Оператор COMMIT в одной из программ применяется только к этому процессу. Средство координации этих процессов отсутствует.



*Рисунок 183. Структура задач DSN. Каждый прямоугольник представляет блок управления задачами (TCB).*

Примечания к рис. 183:

1. Команда RUN с опцией CP заставляет DSN подключить вашу программу и создать новый TCB.
2. Команда RUN без опции CP заставляет DSN связаться с вашей программой.

## Вызов одной программы SQL через ISPF и DSN

С помощью этой структуры пользователь вашей прикладной программы вызывает сначала ISPF, которая выводит на экран данные и панели выбора. Когда пользователь выбирает программу на панели выбора, ISPF вызывает CLIST, который запускает эту программу. Соответствующий CLIST может содержать выражение:

```

DSN
  RUN PROGRAM(MYPROG) PLAN(MYPLAN)
END

```

У этой прикладной программы будет один большой загрузочный модуль и один план.

**Недостатки:** Для больших программ этого типа возникает потребность модульной разработки, что делает план более гибким и легким в обслуживании. Если у вас один большой план, необходимо повторно связывать его целиком при каждом изменении модуля, содержащего операторы SQL.<sup>6</sup> Передача управления другому загрузочному модулю,

<sup>6</sup> Чтобы достичь модульности строения программы, все части которой используют SQL, попробуйте применить пакеты. Посмотрите раздел "Глава 5–1. Планирование прекомпиляции и связывания" на стр. 341.

который вызывает SQL, с помощью ISPLINK невозможна; вместо этого надо использовать LINK, XCTL или LOAD и BALR.

Если требуется использовать ISPLINK, вызовите ISPF для запуска под DSN:  
DSN

```
RUN PROGRAM(ISPF) PLAN(MYPLAN)
END
```

В этом случае надо выйти из ISPF, прежде чем можно будет запустить вашу прикладную программу.

Кроме того, программа в целом зависит от DB2; если DB2 не запущена, ни одна из частей программы не сможет начать или продолжать работу.

---

## Вызов нескольких программ SQL через ISPF и DSN

Большую прикладную программу можно разбить на несколько различных функций, взаимодействующих через общий пул совместно используемых переменных под управлением ISPF. Некоторые функции можно написать как раздельно компилируемые и загружаемые программы, другие – как модули EXEC или CLIST. Любую из этих программ или функций можно запустить через службу ISPF SELECT, которая запускается из программы, CLIST или панели выбора ISPF.

При использовании службы ISPF SELECT можно указать, должна ли ISPF создать новый пул переменных ISPF перед вызовом функции. Большую прикладную программу можно также разбить на несколько независимых частей, у каждой из которых есть собственный пул переменных ISPF.

Различные части программы можно вызывать разными способами. Например, можно использовать опцию PGM из ISPF SELECT:

PGM(*имя программы*) PARM(*параметры*)

Можно также использовать опцию CMD:

CMD(*команда*)

Для части, которая обращается к DB2, эта команда может называться CLIST, запускающий DSN:

```
DSN
RUN PROGRAM(PART1) PLAN(PLAN1) PARM(ввод с панели)
END
```

Разбиение прикладной программы на отдельные модули делает ее более гибкой и облегчает ее обслуживание. Кроме того, некоторые модули могут не зависеть от DB2; тогда части прикладной программы, не вызывающие DB2, могут работать, даже если DB2 не запущена. Не работающая DB2 в этом случае не мешает работе частей программы, которые обращаются только к другим базам данных.

**Недостатки:** Модульная программа должна выполнить больше различных операций. Она вызывает несколько CLIST, причем каждый из них надо найти, загрузить, проанализировать, интерпретировать и выполнить. Она также устанавливает и разрывает соединения с DB2 чаще, чем отдельный модуль загрузки. В результате эффективность может снизиться.

---

## Вызов нескольких программ SQL посредством ISPF и CAF

Для вызова DB2 можно использовать утилиту подключения по вызову (CAF); подробности смотрите в разделе “Глава 7–7. Программирование для утилиты подключения по вызову (CAF)” на стр. 769. Программы менеджера соединений примера ISPF/CAF (DSN8SPM и DSN8SCM) используют преимущества службы ISPLINK SELECT, позволяя каждой подпрограмме устанавливать свое соединение с DB2, свой поток и свой план.

При той же модульной структуре, что и в предыдущем примере, использование CAF позволит повысить производительность за счет снижения числа CLIST. Это не означает, однако, что любая функция DB2 будет выполняться быстрее.

**Недостатки:** По сравнению с модульной структурой, использующей DSN, структура, использующая CAF, потребует более сложной программы, а для нее, возможно, будут нужны процедуры на ассемблере. Дополнительную информацию смотрите в разделе “Глава 7–7. Программирование для утилиты подключения по вызову (CAF)” на стр. 769.

---

## Глава 7–7. Программирование для утилиты подключения по вызову (CAF)

Утилита подключения по вызову – это часть программного кода DB2, позволяющая другим программам соединяться с DB2 и использовать DB2 для обработки операторов SQL, команд или вызовов интерфейса инструментальных средств (IFI). Используя средство подключения вызовов (CAF), прикладная программа может устанавливать свои собственные соединения с DB2 и может управлять этими соединениями. CAF могут использовать программы, выполняющиеся в пакетной среде MVS, программы, выполняющиеся в виде приоритетного процесса TSO и в виде фонового процесса TSO.

Пакетные прикладные программы IMS также могут обращаться к базам данных DB2 через CAF, но этот метод не позволяет согласовывать принятия единиц работы между системами IMS и DB2. Мы настоятельно рекомендуем использовать для пакетных прикладных программ IMS пакетную поддержку DL/I DB2.

Прикладные программы CICS должны использовать средство подключения CICS; прикладные программы IMS – средство подключения IMS. Программы, выполняющиеся в виде приоритетного процесса TSO или в виде фонового процесса TSO, могут использовать или командный процессор DSN, или CAF; каждый из этих методов имеет свои преимущества и недостатки.

**Необходимые знания:** Аналитики и программисты, собирающиеся использовать CAF, должны быть хорошо знакомы с принципами и возможностями MVS в следующих областях:

- Макрокоманда CALL и стандартные соглашения о связывании модулей
- Опции адресации и расположения программы в памяти (AMODE и RMODE)
- Создание заданий и управление заданиями; многозадачность
- Средства функционального восстановления, такие как ESTAE, ESTAI и FRR
- Асинхронные события и обработчики ситуаций внимания TSO (STAX)
- Техники синхронизации, такие как WAIT/POST.

---

### Возможности и ограничения утилиты подключения по вызову

Чтобы принять решение, использовать ли утилиту подключения по вызову, рассмотрите описанные ниже возможности и ограничения.

### Возможности при использовании CAF

Используя CAF, программа может:

- Обращаться к DB2 из адресных пространств MVS при отсутствии TSO, IMS или CICS.
- Обращаться к DB2 из нескольких заданий MVS в одном адресном пространстве.
- Обращаться к IFI DB2.

- Выполняться, когда система DB2 не активна (хотя в этом случае программа не сможет выполнять операторы SQL).
- Выполняться с программой монитора терминала (TMR) TSO или без нее.
- Выполняться, не являясь подзаданием командного процессора DSN (или какого-либо кода DB2).
- Выполняться в области памяти выше или ниже 16 Мегабайт. (Код CAF располагается в области памяти ниже 16 Мегабайт.)
- Устанавливать явное соединение с DB2, используя интерфейс *CALL* и управляя точным состоянием соединения.
- Устанавливать неявное соединение с DB2 без предварительного вызова CAF, используя операторы SQL или вызовы IFI со значениями по умолчанию для имени плана и имени подсистемы.
- Проверять, что используется правильный выпуск DB2.
- Задавать блоки управления событиями (ECB), которые система DB2 передает при запуске или при завершении ее работы.
- Перехватывать коды возврата, коды причины и коды аварийного завершения от DB2 и преобразовывать их в текстовые сообщения.

## **Возможности заданий**

Любое задание в адресном пространстве может установить соединение с DB2 через CAF. Может существовать только одно соединение для каждого блока управления заданием (TCB). Требование к службам DB2, выданное программой из конкретного задания, использует соединение с DB2 этого задания. Это требование к службам выполняются независимо от любых других операций DB2 для других заданий.

Каждое соединенное задание может выполнять план. Несколько заданий в одном адресном пространстве могут использовать один план, но каждый экземпляр плана выполняется независимо от других. Задание может завершить свой план и выполнять другой план, не прекращая полностью своего соединения с DB2.

CAF не создает структуры задания и не содержит обработчиков ситуаций внимания и процедур функционального восстановления. Можно задать нужные обработчики ситуаций внимания и процедуры функционального восстановления, но при этом необходимо использовать процедуры восстановления типа ESTAE/ESTAI, а не процедуры EUT FRR.

При использовании нескольких одновременных соединений увеличивается вероятность тупиковых ситуаций и конфликтов ресурсов DB2. Учитывайте это при разработке прикладных программ.

## **Язык программирования**

Прикладную программу CAF можно писать на языках ассемблер, С, COBOL, FORTRAN и PL/I. При выборе языка для написания прикладной программы учтите следующие ограничения:

- Если нужно использовать макрокоманды MVS (ATTACH, WAIT, POST и т.п.), необходимо выбрать язык программирования, который их поддерживает, или встроить в программу модули на ассемблере.

- Функция CAF TRANSLATE недоступна из языка FORTRAN. Чтобы использовать эту функцию, вызывайте ее из процедуры, написанной на другом языке, а затем вызовите эту процедуру из языка FORTRAN.

Примеры программ на ассемблере (DSN8CA) и на языке COBOL (DSN8CC), использующих утилиту подключения по вызову, приводятся в библиотеке префикс.SDSNSAMP. Прикладная программа на языке PL/I (DSN8SPM) вызывает DSN8CA, а прикладная программа на языке COBOL (DSN8SCM) вызывает DSN8CC. Дополнительную информацию о примерах прикладных программ и их исходных текстах смотрите в разделе Приложение В, "Примеры прикладных программ" на стр. 883.

### **Утилита трассировки**

Утилита трассировки генерирует диагностические сообщения, помогающие при отладке программ и поиске ошибок в программах CAF. В частности, попытки неправильного использования CAF вызывают появление в данных трассировки сообщений об ошибках.

### **Подготовка программ**

Подготовка прикладной программы для выполнения в CAF аналогична подготовке программы для выполнения в других средах, таких как CICS, IMS и TSO. Подготовку прикладной программы CAF можно произвести в пакетной среде или используя процесс подготовки программы DB2. Систему подготовки программы можно использовать или через DB2I, или через DSNH CLIST.

Примеры и рекомендации по подготовке программ смотрите в разделе "Глава 6–1. Подготовка прикладной программы к запуску" на стр. 435.

## **Требования CAF**

При написании программ, использующих CAF, учитывайте следующие характеристики.

### **Размер программы**

Для кода CAF требуется около 16 Кбайт виртуальной памяти для каждого адресного пространства и дополнительные 10 Кбайт для каждого TCB, использующего CAF.

### **Использование макрокоманды LOAD**

CAF использует макрокоманду MVS SVC LOAD для загрузки двух модулей в процессе инициализации после первого требования на обслуживание. Модуль загружается в защищенную от выборки память, имеющую ключ защиты шага задания. Если локальная среда перехватывает и замещает вызов LOAD SVC, необходимо убедиться, что используемая версия LOAD работает с цепочками элементов списка загрузки (LLE) и элементов каталога содержимого (CDE) аналогично стандартной макрокоманде MVS LOAD.

### **Использование CAF в пакетной среде IMS**

При использовании CAF из пакетной среды IMS необходимо в одной единице работы записывать данные только в одну систему. Если в одной единице работы записываются данные в две системы, системная ошибка может вызвать несовместимость баз данных и невозможность автоматического восстановления. Чтобы завершить единицу работы в DB2, выполните оператор SQL COMMIT; чтобы завершить единицу работы в IMS, выполните команду SYNCPOINT.

## **Среда выполнения**

Прикладные программы, обращающиеся к службам DB2, должны удовлетворять некоторым требованиям среды выполнения. Эти требования должны выполняться независимо от используемого средства подключения. Они относятся не только к CAF.

- Прикладная программа должна выполняться в режиме TCB. Режим SRB не поддерживается.
- При обращениях прикладной программы к службам DB2 не должны быть активны EUT FRR. Если активен какой-либо EUT FRR, может возникнуть ошибка функционального восстановления DB2 и прикладная программа может получить непредсказуемый код аварийного завершения.
- Различные средства подключения не могут быть одновременно активны в одном адресном пространстве. Это значит, что:
  - Прикладная программа не должна использовать CAF в адресном пространстве CICS или IMS.
  - Прикладная программа, выполняющаяся в адресном пространстве, которое имеет соединение с DB2 при помощи CAF, не может соединяться с DB2 при помощи RRSAF.
  - Прикладная программа, выполняющаяся в адресном пространстве, имеющем соединение с DB2 при помощи RRSAF, не может соединяться с DB2 при помощи CAF.
- Одно средство подключения не может запускать другое средство подключения. Это означает, что прикладная программа CAF не может использовать DSN, а подкоманда DSN RUN не может вызывать прикладную программу CAF.
- Модуль DSNALI языкового интерфейса для CAF поставляется с атрибутами компоновки AMODE(31) и RMODE(ANY). Если прикладные программы загружают CAF в область ниже 16 МБайт, необходимо заново скомпоновать модуль DSNALI.

## **Выполнение прикладных программ DSN под CAF**

Можно, хотя не рекомендуется, выполнять существующие прикладные программы DSN с использованием CAF, просто позволив им устанавливать неявные соединения с DB2. Чтобы сделать это, имя плана прикладной программы должно совпадать с именем модуля требований к базе данных (DBRM), построенному при препроцессинге модуля, делающего первый вызов SQL. Необходимо также заменить модуль языкового интерфейса DSNALI на модуль языкового интерфейса для TSO (DSNELI).

Выполнение прикладных программ DSN с CAF не дает значительного выигрыша, а потеря служб DSN может влиять на работу программ. Не рекомендуется выполнять прикладные программы DSN с CAF, если у вас нет контроллера прикладных программ, управляющего прикладной программой DSN и замещающего необходимые функции DSN. Но даже в этом случае может потребоваться изменение прикладной программы для правильного взаимодействия с контроллером при ошибках соединения.

---

## Как использовать CAF

Чтобы использовать CAF, необходимо сначала сделать доступным загружаемый модуль DSNALI – языковый интерфейс утилиты подключения по вызову. Информацию о загрузке и компоновке этого модулясмотрите в разделе “Обращение к языковому интерфейсу CAF” на стр. 776.

Если языковый интерфейс доступен, в программе есть два способа использования CAF:

- Неявно, просто включив в программу операторы SQL или вызовы IFI. CAF устанавливает соединения с DB2, используя значения по умолчанию для соответствующих параметров, описанных в разделе “Неявные соединения” на стр. 775.
- Явно, написав оператор CALL DSNALI с соответствующими опциями. Общую форму этого операторасмотрите в разделе “Описания функций CAF” на стр. 779.

Первый элемент каждого списка опций – функция, описывающая действие, которое должна выполнить CAF. Список возможных функций и описания их действийсмотрите в разделе “Краткое описание функций соединения” на стр. 774. Результат выполнения любой функции в какой–то степени зависит от того, какие функции уже были выполнены программой. Перед использованием какой–либо функции не забудьте прочитать описание ее использования. Прочтите также раздел “Сводка поведения CAF” на стр. 792, в котором описывается влияние выполненных ранее функций.

Пример возможной конфигурации CAF:

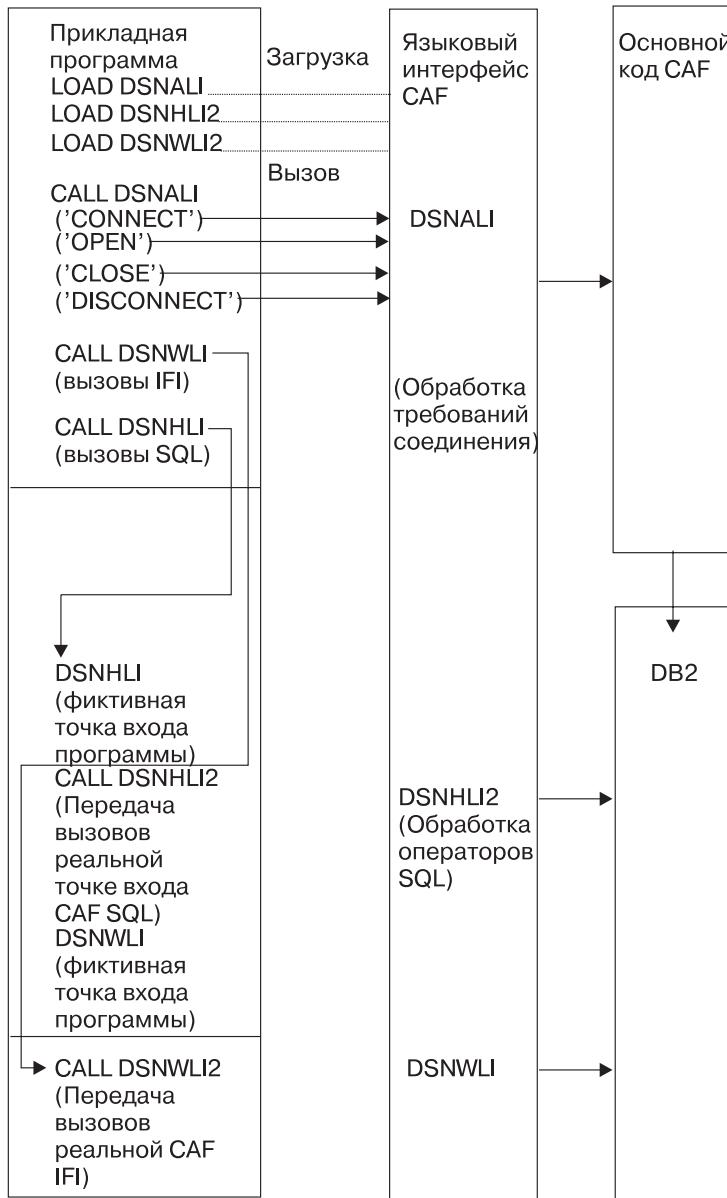


Рисунок 184. Пример конфигурации утилиты подключения по вызову

В оставшейся части этой главы описываются:

- Краткое описание функций соединения
- “Примеры сценариев” на стр. 794
- “Обработчики для прикладной программы” на стр. 795
- “Сообщения об ошибках и набор данных DSNTRACE” на стр. 796
- “Примеры программ” на стр. 797.

## Краткое описание функций соединения

В операторе CALL DSNALI можно использовать следующие функции:

### CONNECT

Задает задание (TCB) как пользователя указанной подсистемы DB2. Когда первое задание в адресном пространстве выдает требование на соединение, это адресное пространство также инициализируется как

пользователь DB2. Смотрите раздел “Функция CONNECT: синтаксис и использование” на стр. 781.

#### **OPEN**

Выделяет план DB2. Чтобы система DB2 могла обрабатывать операторы SQL, необходимо выделить план. Если функция CONNECT не была вызвана, функция OPEN неявно задает это задание (и, возможно, адресное пространство) как пользователя DB2. Смотрите раздел “Функция OPEN: синтаксис и использование” на стр. 785.

#### **CLOSE**

Освобождает план (возможно, выполнив принятие или отмену изменений базы данных). Если функция OPEN неявно вызывает функцию CONNECT, функция CLOSE удаляет задание (и, возможно, адресное пространство) из числа пользователей DB2. Смотрите раздел “Функция CLOSE: синтаксис и использование” на стр. 787.

#### **DISCONNECT**

Удаляет задание из числа пользователей DB2 и, если это последнее или единственное задание в данном адресном пространстве, имеющее соединение с DB2, завершает соединение с DB2 этого адресного пространства. Смотрите раздел “Функция DISCONNECT: синтаксис и использование” на стр. 789.

#### **TRANSLATE**

Возвращает (в SQLCA) SQLCODE и текст, описывающие шестнадцатеричный код причины ошибки DB2. Смотрите раздел “Функция TRANSLATE: синтаксис и использование” на стр. 791. Функцию TRANSLATE нельзя вызывать из программ на языке FORTRAN.

### **Неявные соединения**

Если в операторе CALL DSNALI в прикладной программе CAF не заданы явно выполняемые операторы SQL, CAF инициирует неявные требования CONNECT и OPEN к DB2. Хотя CAF выполняет эти требования соединения, используя описанные ниже значения по умолчанию, для этих требований возвращаются те же коды возврата и коды причины DB2, что и для явно заданных требований.

Для неявных соединений используются следующие значения по умолчанию:

#### **Имя подсистемы**

Имя по умолчанию задается в модуле DSNHDECP. CAF использует стандартный модуль DSNHDECP, если вы не установите собственный модуль DSNHDECP в библиотеке в STEPLIB, заданной в JOBLIB или в списке компоновки. В группе совместного использования данных имя подсистемы по умолчанию — это имя подключения группы.

#### **Имя плана**

Имя модуля требований к базе данных (DBRM), связанного с модулем, который делает первый вызов SQL. Если программа может выполнять свой первый вызов SQL из разных модулей с различными DBRM, нельзя использовать имя плана по умолчанию; необходимо использовать явный вызов функции OPEN.

Если прикладная программа содержит вызовы SQL и вызовы IFI, перед вызовами IFI необходимо выполнить хотя бы один вызов SQL. Это обеспечивает использование прикладной программой правильного плана.

Существуют различные типы неявных соединений. В простейшем случае прикладная программа не использует ни функцию CONNECT, ни функцию OPEN. Можно также использовать только функцию CONNECT или только функцию OPEN. В каждом из этих случаев устанавливается неявное соединение прикладной программы с DB2. Чтобы завершить неявное соединение, необходимо использовать соответствующие вызовы. Подробную информацию смотрите в Табл. 84 на стр. 793.

Перед выполнением любых вызовов SQL с использованием точки входа DSNHLI CAF прикладная программа должна успешно установить соединение, явно или неявно. Поэтому прикладная программа должна сначала определить, были ли успешными или неудачными предыдущие неявные требования соединения.

Для неявных требований соединения регистр 15 содержит код возврата, а регистр 0 – код причины. Код возврата и код причины содержатся также в тексте сообщения для SQLCODE –991. Прикладная программа должна проверить код возврата и код причины сразу после первого выполняемого в прикладной программе оператора SQL. Есть два способа сделать это:

- Непосредственно проверить содержимое регистров 0 и 15.
- Проверить SQLCA, и если SQLCODE равен 991, получить код возврата и код причины из текста сообщения. Код возврата – первый элемент, а код причины – второй элемент сообщения.

Если неявное соединение установлено успешно, прикладная программа может проверить SQLCODE для первого и последующих операторов SQL.

## Обращение к языковому интерфейсу CAF

Часть утилиты подключения по вызову – загрузочный модуль DB2 DSNALI, модуль языкового интерфейса средства подключения вызовов. Алиасы DSNALI – DSNHLI2 и DSNWLI2. Этот модуль имеет пять точек входа: DSNALI, DSNHLI, DSNHLI2, DSNWLI и DSNWLI2:

- Точка входа DSNALI используется для обработки явных требований соединения DB2.
- DSNHLI и DSNHLI2 используются для обработки вызовов SQL (DSNHLI применяется, если прикладная программа использует скомпонованный модуль CAF; DSNHLI2 применяется, если прикладная программа использует загружаемый модуль CAF).
- DSNWLI и DSNWLI2 используются для обработки вызовов IFI (DSNWLI применяется, если прикладная программа использует скомпонованный модуль CAF; DSNWLI2 применяется, если прикладная программа использует загружаемый модуль CAF).

Для обращения к модулю DSNALI можно использовать явные требования LOAD при выполнении программы или же включить в процессе компоновки программы модуль DSNALI в ее загрузочный модуль. Каждый из этих способов имеет свои достоинства и недостатки.

## **Явная загрузка DSNALI**

Чтобы загрузить DSNALI, выполните макрокоманду MVS LOAD для точек входа DSNALI и DSNHLI2. Если используются функции IFI, необходимо также загрузить DSNWLI2. Сохраните адрес точки входа, возвращаемый макрокомандой LOAD, и используйте его в макрокомандах CALL.

Явная загрузка модуля DSNALI позволяет сделать обслуживание прикладной программы независимым от будущих изменений в языковом интерфейсе. Если языковый интерфейс будет изменен, это скорее всего не повлияет на работу вашего загрузочного модуля.

Необходимо указать DB2, какую точку входа следует использовать. Это можно сделать одним из двух способов:

- Задайте опцию прекомпилятора ATTACH(CAF).  
DB2 сгенерирует вызовы для точки входа DSNHLI2. Эту опцию нельзя использовать в прикладных программах на языке FORTRAN.
- Задайте в программе для вашего загрузочного модуля фиктивную точку входа с именем DSNHLI.

Если не задана опция прекомпилятора ATTACH, прекомпилятор DB2 генерирует для каждого оператора SQL вызовы для точки входа DSNHLI. Прекомпилятор не знает, какое будет использоваться средство подключения DB2, и его работа не зависит от этого. Когда вызовы, сгенерированные прекомпилятором DB2, передают управление точке входа DSNHLI, ваш код, соответствующий этой фиктивной точке входа, должен сохранить список опций, передаваемый в R1, и вызвать DSNHLI2 с тем же списком опций. Пример программного кода для фиктивной точки входа DSNHLI смотрите в разделе “Использование фиктивной точки входа DSNHLI” на стр. 804.

## **Компоновка DSNALI**

Модуль DSNALI языкового интерфейса CAF можно включить в ваш загрузочный модуль при компоновке. Этот модуль должен входить в библиотеку загрузочных модулей, входящей в SYSLIB, или в другую библиотеку INCLUDE, заданную в компоновщике JCL. Поскольку все модули языковых интерфейсов содержат объявление точки входа DSNHLI, компоновщик JCL должен иметь управляющий оператор компоновки INCLUDE для DSNALI (например, INCLUDE DB2LIB(DSNALI)). Это позволяет избежать компоновки неправильного модуля языкового интерфейса.

Если не требуется явно вызывать DSNALI для функций CAF, включение модуля DSNALI в загрузочный модуль имеет некоторые преимущества. Если DSNALI включен во время компоновки, не нужно включать в программу описанную выше фиктивную точку входа DSNHLI или задавать опцию прекомпилятора ATTACH. Модуль DSNALI содержит точку входа для DSNHLI, которая идентична точке входа DSNHLI2, и точку входа DSNWLI, которая идентична точке входа DSNWLI2.

Недостаток компоновки DSNALI в загрузочный модуль в том, что если IBM внесет в модуль DSNALI какие-либо изменения, необходимо будет заново скомпоновать программу.

## Основные свойства соединений CAF

Некоторые основные свойства соединения утилиты подключения по вызову с DB2:

- **Имя соединения:** DB2CALL. Можно использовать команду DISPLAY THREAD, чтобы получить список прикладных программ CAF с именем соединения DB2CALL.
- **Тип соединения:** BATCH. Соединения BATCH используют однофазное принятие, управляемое системой DB2. Прикладные программы могут также использовать операторы SQL COMMIT и ROLLBACK.
- **ID авторизации:** При обработке соединения для каждого задания DB2 устанавливает идентификаторы авторизации для соединения этого задания. Для типа соединения BATCH система DB2 создает список ID авторизации на основе ID авторизации, связанного с адресным пространством; это один и тот же список для каждого задания. Локальная система может обеспечивать соединение DB2 процедурой обработки, которая используется для изменения списка ID авторизации. Информацию об ID авторизации и процедурах обработки авторизациисмотрите в приложении В (том 2) руководства *DB2 Administration Guide*.
- **Область видимости:** CAF обрабатывает соединения таким образом, как будто каждое задание работает совершенно независимо. Когда задание вызывает какую-либо функцию, CAF передает этот вызов системе DB2 независимо от состояния соединений других заданий в этом адресном пространстве. Однако прикладная программа и подсистема DB2 имеют доступ к информации о состоянии соединений всех заданий в адресном пространстве.

### Завершение задания

Если задание, использующее соединение, завершается нормально до того, как функция CLOSE освободит план, DB2 выполняет принятие всех изменений, сделанных потоком после последней точки принятия. Если задание, использующее соединение, завершается ненормально до того, как функция CLOSE освободит план, DB2 выполняет откат всех изменений, сделанных после последней точки принятия.

В обоих случаях перед завершением задания DB2 освобождает план, если это необходимо, и завершает соединение для этого задания.

### Аварийное завершение DB2

Если в процессе выполнения прикладной программы происходит аварийное завершение DB2, для прикладной программы производится откат к последней точке принятия. Если работа DB2 завершается во время обработки требования принятия, система DB2 произведет принятие или откат всех изменений при последующем ее запуске. Выполняемая при этом операция зависит от состояния выполнения требования принятия в момент завершения работы системы DB2.

## Описания функций CAF

При использовании функций CAF в программах на языках С, COBOL, FORTRAN или PL/I следуйте правилам конкретного языка для реализации вызовов процедур на ассемблере. Задавайте в списке параметров для каждого вызова CAF параметры кода возврата и кода причины.

Далее описываются соглашения об использовании регистров и о списках параметров для функций CAF для языка ассемблер. После этого приводится описание синтаксиса и параметров для конкретных функций.

### Соглашение об использовании регистров

Если в вызове CAF не заданы параметры кода возврата и кода причины, CAF помещает код возврата в регистр 15, а код причины – в регистр 0. CAF поддерживает также языки высокого уровня, в которых невозможно обращаться к отдельным регистрам. Дополнительную информацию смотрите в рис. 185 на стр. 780 и последующем обсуждении. Во время вызовов сохраняется содержимое регистров со 2 по 14. Программа должна использовать следующие стандартные соглашения о вызовах:

Регистр	Использование
R1	Указатель на список параметров (подробное описание смотрите в разделе “Список параметров вызовов DSNALI”)
R13	Адрес области сохранения вызывающей программы
R14	Адрес возврата вызывающей программы
R15	Адрес точки входа CAF

### Список параметров вызовов DSNALI

Используйте стандартный список параметров для вызовов MVS CALL. Регистр 1 указывает на список полных адресов, указывающих на сами параметры. Последний адрес должен содержать в старшем бите значение 1. На рис. 185 на стр. 780 показан пример структуры списка параметров для функции CONNECT.

При написании операторов CALL DSNALI задавайте все параметры, идущие перед кодом возврата. Нельзя пропускать какие-либо из этих параметров, задавая нули или пробелы. Для этих параметров для явных требований соединения не существуют значения по умолчанию. Значения по умолчанию используются только для неявных соединений.

Все параметры, начиная с кода возврата, необязательны.

Для всех языков, кроме ассемблера, задайте для параметра в операторе CALL DSNALI значение 0, если нужно, чтобы для этого параметра использовалось значение по умолчанию, а после него задайте последующие параметры. Допустим, например, что вы пишите вызов CONNECT в программе на языке COBOL. Нужно задать все параметры, кроме кода возврата. Запишите этот вызов так:

```
CALL 'DSNALI' USING FUNCTN SSID TECB SECB RIBPTR  
      BY CONTENT ZERO BY REFERENCE REASCODE SRDURA EIBPTR.
```

Для вызова на ассемблере задайте запятую для параметра в операторе CALL DSNALI, если нужно, чтобы для этого параметра использовалось значение по

умолчанию, а после него задайте последующие параметры. Например, чтобы задать все необязательные параметры, кроме кода возврата, запишите такой вызов CONNECT:

```
CALL DSNALI,(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,,REASCODE,SRDURA,EIBPTR)
```

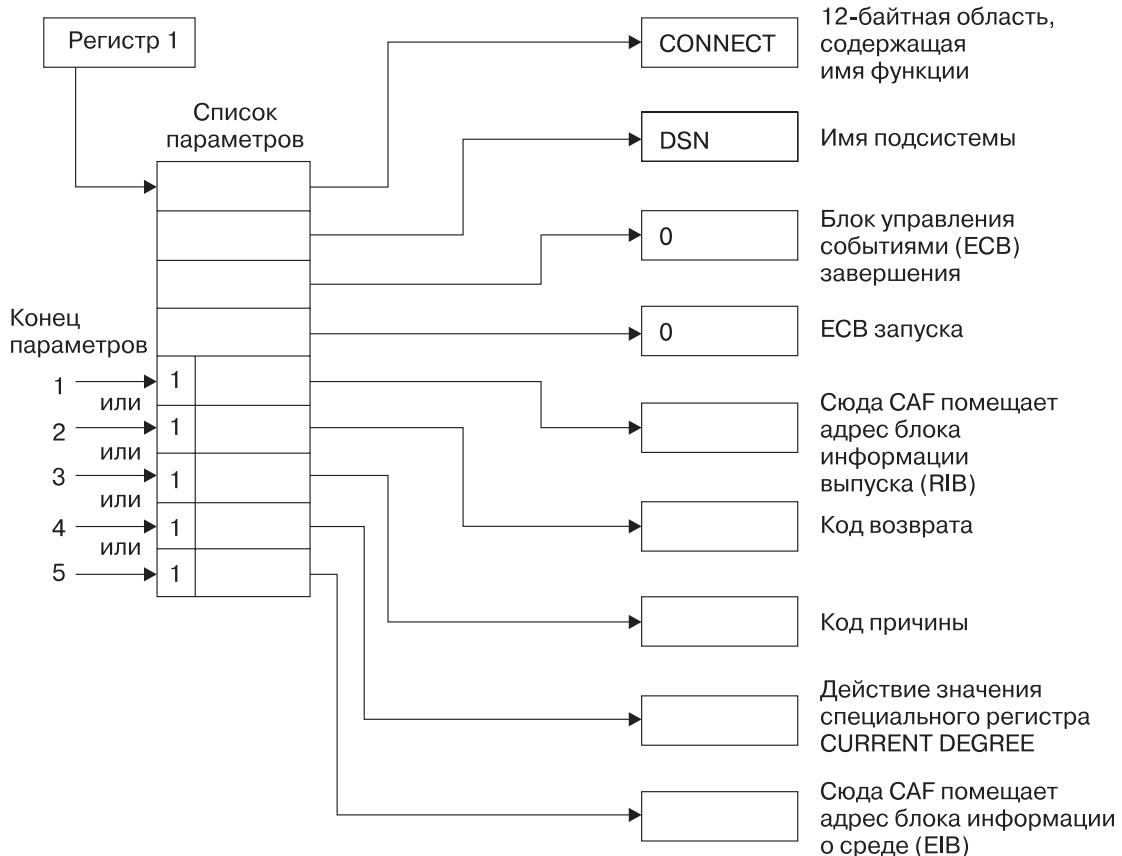


Рисунок 185. Список параметров для вызова CONNECT

На рис. 185 показано, как можно использовать индикатор конца списка параметров для управления полями кода возврата и кода причины, идущими после вызова CAF CONNECT. Далее описаны точки завершения списка параметров для список параметров CAF:

- Список параметров завершается без задания параметров *код\_возврата*, *код\_причины* и *srdura*; код возврата помещается в регистр 15, а код причины – в регистр 0.  
Такое завершение списка параметров обеспечивает совместимость с программами CAF, передающими код возврата в регистре 15 и код причины в регистре 0.
- Список параметров завершается после параметра кода возврата; код возврата помещается в параметр в списке параметров, а код причины – в регистр 0.  
Такое завершение списка параметров позволяет прикладной программе выполнять действия, исходя из кода возврата, но не проверяя соответствующий код причины.
- Список параметров завершается после параметра кода причины; коды возврата и причины помещаются в параметры в списке параметров.

Такое завершение списка параметров обеспечивает поддержку языков высокого уровня, которые не могут работать с содержимым отдельных регистров.

При написании прикладной программы CAF на языке ассемблер можно задать параметр кода причины, пропустив параметр кода возврата. Чтобы сделать это, задайте запятую вместо пропущенного параметра кода возврата.

4. Список параметров завершается после параметра *srdura*.

При написании прикладной программы CAF на ассемблере можно задать этот параметр, пропустив параметр кода возврата и кода причины. Чтобы сделать это, задайте запятые вместо пропущенных параметров.

5. Список параметров завершается после параметра *адрес\_EIB*.

При написании прикладной программы CAF на ассемблере можно задать этот параметр, пропустив параметр кода возврата и кода причины. Чтобы сделать это, задайте запятые вместо пропущенных параметров.

Даже если в списке параметров задан параметр для кода возврата, код возврата помещается и в регистр 15 для языков высокого уровня, поддерживающих специальную обработку кода возврата.

## Функция CONNECT: синтаксис и использование

Функция CONNECT инициализирует соединение с DB2. Не путайте функцию CONNECT утилиты подключения по вызову с оператором DB2 CONNECT, который используется для доступа к удаленной системе внутри DB2.

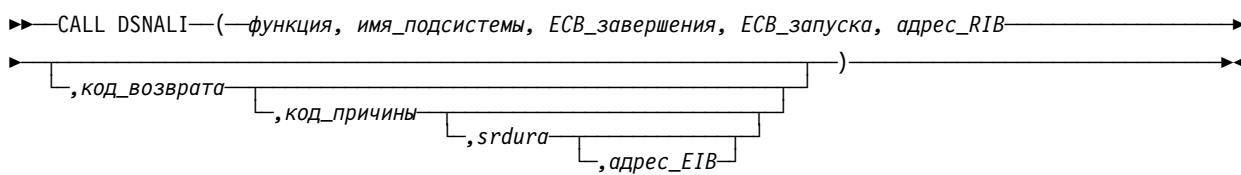


Рисунок 186. Функция DSNALI CONNECT

Параметры указывают на следующие области:

### функция

12–байтная область, содержащая слово *CONNECT*, после которого идут пять пробелов.

### имя\_подсистемы

4–байтное имя подсистемы DB2 или имя группового подключения (если используется группа совместного использования данных), с которым установлено соединение.

Если задается имя группового подключения, программа соединяется с DB2 в системе MVS, в которой выполняется эта программа. Если задается имя группового подключения и ECB запуска, DB2 игнорирует этот ECB запуска. Если нужно использовать ECB запуска, задайте имя подсистемы, а не имя группового подключения. Это имя подсистемы не должно совпадать с именем группового подключения.

Если *имя\_подсистемы* короче четырех символов, дополните его справа пробелами до длины четыре символа.

#### *ECB\_завершения*

Адрес блока управления событиями (ECB) для этой прикладной программы, используемого при завершении DB2. DB2 передает этот ECB, если оператор системы вводит команду STOP DB2 или если DB2 завершается ненормально. Код POST этого ECB указывает тип завершения:

Код POST	Тип завершения
8	QUIESCE
12	FORCE
16	ABTERM

В прикладной программе CAF перед проверкой *ECB\_завершения* сначала проверьте код возврата и код причины, возвращенные вызовом CONNECT, чтобы убедиться, что этот вызов успешно выполнен. Дополнительную информациюсмотрите в разделе “Проверка кодов возврата и кодов причины” на стр. 800.

#### *ECB\_запуска*

Адрес ECB запуска для этой прикладной программы. Если в момент, когда прикладная программа выдает этот вызов, система DB2 еще не была запущена, DB2 передает этот ECB при успешном завершении запуска DB2. DB2 передает максимум один ECB запуска для адресного пространства. Передаваемый ECB связан с самым последним вызовом CONNECT из этого адресного пространства. Прикладная программа должна проверить все ненулевые коды причины CAF/DB2, прежде чем выполнять команду WAIT для этого ECB.

Если *имя\_подсистемы* – это имя группового подключения, DB2 игнорирует ECB запуска.

#### *адрес\_RIB*

4–байтная область, в которую CAF помещает после вызова адрес блока информации о выпуске (RIB). Поле RIBREL можно использовать для определения используемого в настоящее время уровня выпуска DB2. Если RIB недоступен (например, если указанная подсистема не существует), DB2 заполняет эту 4–байтную область нулями.

Область, на которую указывает параметр *адрес\_RIB*, находится в области памяти ниже 16 МБайт.

Параметр *адрес\_RIB* является обязательным, хотя прикладная программа может не использовать блок информации о выпуске.

Макрокоманда DSNDRIB определяет структуру блока информации о выпуске (RIB). Ее можно найти в префикс.SDSNMACS(DSNDRIB).

#### *код\_возврата*

4–байтная область, в которую CAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, CAF помещает код возврата в регистр 15 и код причины в регистр 0.

#### *код\_причины*

4-байтная область, в которую CAF помещает код причины. Если этот параметр не задан, CAF помещает код причины в регистр 0.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметр *код\_возврата*.

#### *srdura*

10-байтная область, содержащая строку 'SRDURA(CD)'. Это необязательный параметр. Если он задан, значение специального регистра CURRENT DEGREE действует в промежутке времени от вызова CONNECT до вызова DISCONNECT. Если он не задан, значение специального регистра CURRENT DEGREE действует от вызова OPEN до вызова CLOSE. Если этот параметр задан в каком-либо языке, кроме ассемблера, необходимо также задать параметры *код\_возврата* и *код\_причины*. В программе на ассемблере можно пропустить параметры *код\_возврата* и *код\_причины*, задав вместо них запятые.

#### *адрес\_EIB*

4-байтная область, в которую CAF помещает после вызова адрес блока информации о среде (EIB). EIB содержит информацию, которую можно использовать при соединении с подсистемой DB2, входящей в группу совместного использования данных. Например, можно определить имя группы совместного использования данных и члена этой группы, с которой производится соединение. При соединении с подсистемой DB2, не входящей в группу совместного использования данных, поля в EIB, относящиеся к совместному использованию данных, остаются пустыми. Если EIB недоступен (например, если заданно имя несуществующей подсистемы), DB2 заполняет эту 4-байтную область нулями.

Область, на которую указывает параметр *адрес\_EIB*, находится в области памяти ниже 16 Мбайт.

В вызовах CONNECT можно не задавать этот параметр.

Если этот параметр задан в каком-либо языке, кроме ассемблера, необходимо также задать параметры *код\_возврата*, *код\_причины* и *srdura*. В программе на ассемблере можно пропустить параметры *код\_возврата*, *код\_причины* и *srdura*, задав вместо них запятые.

Макрокоманда DSNDEIB определяет структуру EIB. Она находится в префикс.SDSNMACS(DSNDEIB).

**Использование:** Функция CONNECT задает вызывающее ее задание как пользователя службы DB2. Если другие задания в этом адресном пространстве в настоящее время не соединены с этой подсистемой (имя которой задается параметром *имя\_подсистемы*), функция CONNECT также инициализирует это адресное пространство для связи с адресными пространствами DB2. Функция CONNECT устанавливает полномочия для перекрестных обращений к памяти от данного адресного пространства к DB2 и создает управляющие блоки адресных пространств.

Вызовы функции CONNECT не являются обязательными. Первое требование от задания (вызов функции OPEN или вызов SQL или IFI) приведет к тому, что CAF выполнит неявный вызов функции CONNECT. Если соединение задания установлено неявно, это соединение с DB2 завершается при выполнении функции CLOSE или при завершении задания.

Установление соединений на уровне задания и на уровне адресного пространства – это операция инициализации, отнимающая много ресурсов. Если для установления соединения задания используется явный вызов функции CONNECT, это соединение завершается при выполнении функции DISCONNECT или при завершении задания. Явный вызов функции CONNECT уменьшает накладные расходы, так как соединение с DB2 сохраняется после того, как функция CLOSE освобождает план.

Функцию CONNECT можно вызывать из какого–либо задания или из всех заданий в адресном пространстве, но инициализация на уровне адресного пространства производится только один раз – при соединении первого задания.

Если задание не выдает явные вызовы функций CONNECT или OPEN, при первом вызове SQL или IFI соединение устанавливается неявно с использованием имени по умолчанию для подсистемы DB2. Это значение по умолчанию имени подсистемы задается системным программистом или системным администратором при установке DB2. Вы должны знать это имя по умолчанию и должны быть уверены, что именно это имя подсистемы DB2 нужно использовать.

На практике не следует смешивать в одном адресном пространстве явные вызовы функций CONNECT и OPEN с неявно установленными соединениями. Или явно задайте подсистему DB2, которую нужно использовать, или позвольте всем требованиям использовать подсистему по умолчанию.

Используйте функцию CONNECT, когда:

- Нужно задать конкретное имя подсистемы, не совпадающее с именем по умолчанию (*имя\_подсистемы*).
- Нужно, чтобы значение специального регистра CURRENT DEGREE действовало в течение всего соединения (*srdura*).
- Нужно отслеживать ECB запуска DB2 (*ECB\_запуска*), ECB завершения DB2 (*ECB\_завершения*) или уровень выпуска DB2.
- Несколько заданий в адресном пространстве будут открывать и закрывать планы.
- Одно задание в адресном пространстве будет открывать и закрывать планы несколько раз.

Другие параметры функции CONNECT позволяют вызывающей программе узнать:

- Что оператор ввел команду DB2 STOP. В этом случае DB2 передает ECB завершения (*ECB\_завершения*). Прикладная программа может или ожидать получения этого ECB, или просто проверять его содержимое.
- Произошло аварийное завершение DB2. В этом случае DB2 передает ECB завершения (*ECB\_завершения*).
- Что система DB2 вновь доступна (после попытки соединения, которая была неудачной из–за недоступности DB2). Программа ожидает получения ECB запуска (*ECB\_запуска*) или проверяет его содержимое. DB2 игнорирует этот ECB, если она была активна во время вызова функции CONNECT или если в вызове CONNECT задано имя группового подключения.

- Текущий уровень выпуска DB2. Используйте поле RIBREL в блоке информации о выпуске (RIB).

Не вызывайте функцию CONNECT из TCB, для которого уже имеется активное соединение с DB2. (Дополнительную информацию об ошибках CAF смотрите в разделах “Сводка поведения CAF” на стр. 792 и “Сообщения об ошибках и набор данных DSNTRACE” на стр. 796.)

В Табл. 79 показаны вызовы функции CONNECT в разных языках.

*Таблица 79. Примеры вызовов функции CAF CONNECT*

Язык	Пример вызова
Ассемблер	CALL DSNALI,(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR, RETCODE,REASCODE,SRDURA,EIBPTR)
C	fnret=dsnali(&functn[0],&ssid[0], &tecb, &secb,&ribptr,&retcode, &reascode, &srdu[0],&eibptr);
COBOL	CALL 'DSNALI' USING FUNCTN SSID TERMECB STARTECB RIBPTR RETCODE REASCODE SRDURA EIBPTR.
FORTRAN	CALL DSNALI(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR, RETCODE,REASCODE,SRDURA,EIBPTR)
PL/I	CALL DSNALI(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,RETCODE, REASCODE,SRDURA,EIBPTR);

**Примечание:** DSNALI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках С и PL/I следующие директивы компилятора:

```

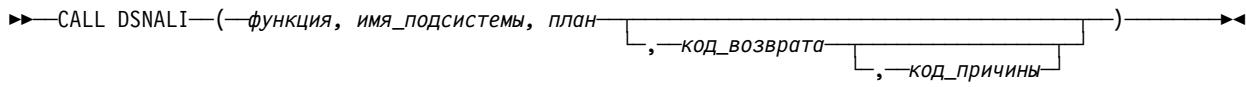
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
            int DSNALI(
                char * functn,
                ...); }

PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);

```

## Функция OPEN: синтаксис и использование

Функция OPEN выделяет ресурсы для выполнения заданного плана. Функция OPEN может также запрашивать соединение с DB2 для вызывающей программы.



*Рисунок 187. Функция DSNALI OPEN*

Параметры указывают на следующие области:

*функция*

12-байтная область, содержащая слово OPEN, после которого идут восемь пробелов.

*имя\_подсистемы*

4–байтное имя подсистемы DB2 или имя группового подключения (если используется группа совместного использования данных). При необходимости функция OPEN устанавливает соединение с подсистемой DB2, имя которой задано этим параметром. Если *имя\_подсистемы* короче четырех символов, дополните его справа пробелами до длины четыре символа.

*план*

8–байтное имя плана DB2.

*код\_возврата*

4–байтная область, в которую CAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, CAF помещает код возврата в регистр 15 и код причины в регистр 0.

*код\_причины*

4–байтная область, в которую CAF помещает код причины. Если этот параметр не задан, CAF помещает код причины в регистр 0.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметр *код\_возврата*.

**Использование:** Функция OPEN выделяет ресурсы DB2, необходимые для выполнения плана и требований IFI. Если для вызывающего задания еще не установлено соединение с указанной подсистемой DB2, функция OPEN устанавливает это соединение.

Функция OPEN выделяет план для подсистемы DB2, имя которой задается параметром *имя\_подсистемы*. Параметр *имя\_подсистемы*, как и другие параметры, является обязательным, даже если задание вызвало функцию CONNECT. Если задание вызывает функцию CONNECT и после нее функцию OPEN, в обоих вызовах должно быть задано одно имя подсистемы.

Вызовы функции OPEN не являются обязательными. Если функция OPEN не используется в программе, ее действия будут выполнены при первом вызове SQL или IFI из этого задания, при этом будут использоваться значения по умолчанию, перечисленные в разделе “Неявные соединения” на стр. 775.

Не используйте функцию OPEN, если для задания уже выделен план.

В Табл. 80 на стр. 787 показаны вызовы функции OPEN в разных языках.

Таблица 80. Примеры вызовов функции CAF OPEN

Язык	Пример вызова
Ассемблер	CALL DSNALI,(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE)
C	fnret=dsnali(&functn[0],&ssid[0], &plannname[0],&retcode, &reascode);
COBOL	CALL 'DSNALI' USING FUNCTN SSID PLANNAME RETCODE REASCODE.
FORTRAN	CALL DSNALI(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE)
PL/I	CALL DSNALI(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE);

**Примечание:** DSNALI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках С и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
          int DSNALI(
            char * functn,
            ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

## Функция CLOSE: синтаксис и использование

Функция CLOSE освобождает план и может также завершать соединения с DB2 задания и, возможно, адресного пространства.

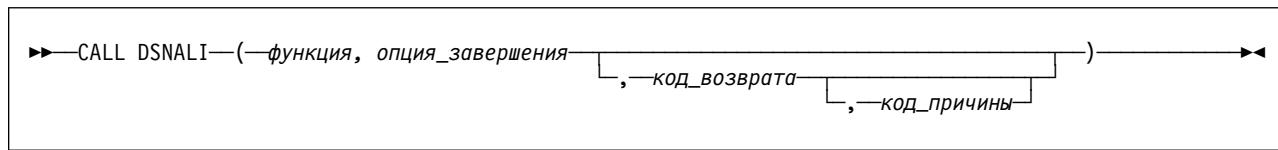


Рисунок 188. Функция DSNALI CLOSE

Параметры указывают на следующие области:

### функция

12-байтная область, содержащая слово CLOSE, после которого идут семь пробелов.

### опция\_завершения

4-байтная опция завершения, содержащая одно из следующих значений:

**SYNC**      Выполнить принятие всех измененных данных

**ABRT**      Выполнить откат до предыдущей точки принятия.

### код\_возврата

4-байтная область, в которую CAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, CAF помещает код возврата в регистр 15 и код причины в регистр 0.

### код\_причины

4-байтная область, в которую CAF помещает код причины. Если этот параметр не задан, CAF помещает код причины в регистр 0.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметр **код\_возврата**.

**Использование:** Функция CLOSE освобождает план, созданный явным вызовом функции OPEN или созданный неявно при первом вызове SQL.

Если для задания не вызывалась функция CONNECT, функция CLOSE также удаляет соединение этого задания с DB2. Если другие задания в этом адресном пространстве не имеют активных соединений с DB2, DB2 также удаляет структуры блоков управления, созданные для этого адресного пространства и удаляет полномочия для перекрестных обращений к памяти.

Не используйте функцию CLOSE, если для текущего задания не выделен план.

Вызовы функции CLOSE не являются обязательными. Если эта функция не вызвана явно, DB2 выполняет те же действия при завершении задания, используя параметр SYNC при нормальном и параметр ABRT при ненормальном завершении. (Это неявный вызов функции CLOSE.) Если некоторые обстоятельства вызывают завершение прикладной программы, можно улучшить производительность завершения, явно вызвав функцию CLOSE перед завершением задания.

Если нужно использовать новый план, необходимо использовать явный вызов функции CLOSE, после которого вызвать функцию OPEN с новым именем плана.

Если работа DB2 завершена, задание, которое не вызывало функцию CONNECT, должно явно вызвать функцию CLOSE, чтобы CAF могло выполнить сброс блоков управления для последующих соединений. Такой вызов функции CLOSE возвращает код возврата *выполнен сброс* (+004) и код причины X'00C10824'. Если не выполнен вызов CLOSE, а система DB2 вновь становится активной, следующее требование на соединение этого задания закончится неудачно. Будет выдано сообщение *Your TCB does not have a connection* (У данного TCB нет соединения) с кодом причины X'00F30018' в регистре 0 или сообщение об ошибке CAF DSNA201I или DSNA202I, в зависимости от действия, которое пыталось выполнить прикладная программа. Задание должно вызвать функцию CLOSE, прежде чем оно сможет вновь соединиться с DB2.

Задание, которое явно вызвало функцию CONNECT, должно вызвать функцию DISCONNECT, чтобы CAF выполнила сброс блоков управления при завершении DB2. В этом случае не обязательно вызывать функцию CLOSE.

В Табл. 81 на стр. 789 показаны вызовы функции CLOSE в разных языках.

Таблица 81. Примеры вызовов функции CAF CLOSE

Язык	Пример вызова
Ассемблер	CALL DSNALI,(FUNCTN,TERMOP,RETCODE, REASCODE)
C	fnret=dsnali(&functn[0], &termop[0], &retcode,&reascode);
COBOL	CALL 'DSNALI' USING FUNCTN TERMOP RETCODE REASCODE.
FORTRAN	CALL DSNALI(FUNCTN,TERMOP, RETCODE,REASCODE)
PL/I	CALL DSNALI(FUNCTN,TERMOP, RETCODE,REASCODE);

**Примечание:** DSNALI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках С и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
          int DSNALI(
            char * functn,
            ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

## Функция DISCONNECT: синтаксис и использование

Функция DISCONNECT завершает соединение с DB2.

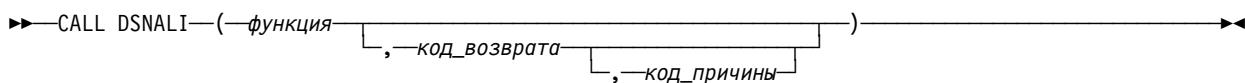


Рисунок 189. Функция DSNALI DISCONNECT

Параметры указывают на следующие области:

### функция

12–байтная область, содержащая слово DISCONNECT, после которого идут два пробела.

### код\_возврата

4–байтная область, в которую CAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, CAF помещает код возврата в регистр 15 и код причины в регистр 0.

### код\_причины

4–байтная область, в которую CAF помещает код причины. Если этот параметр не задан, CAF помещает код причины в регистр 0.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметр *код\_возврата*.

**Использование:** Функция DISCONNECT удаляет соединение вызывающего задания с DB2. Если другие задания в этом адресном пространстве не имеют активных соединений с DB2, DB2 также удаляет структуры блоков управления, созданные для этого адресного пространства и удаляет полномочия для перекрестных обращений к памяти.

Функцию DISCONNECT могут вызывать только те задания, которые явно вызвали функцию CONNECT. Если функция CONNECT не была вызвана, функция DISCONNECT вернет код ошибки.

Если во время вызова функции DISCONNECT имеется выделенный план, CAF неявно вызывает функцию CLOSE с параметром SYNC.

Вызовы функции DISCONNECT не являются обязательными. Если они не используются, DB2 выполняет те же действия при завершении задания. (Это неявный вызов функции DISCONNECT.) Если некоторые обстоятельства вызывают завершение прикладной программы, можно улучшить производительность завершения, явно вызвав функцию DISCONNECT перед завершением задания.

Если работа DB2 завершена, задание, которое вызвало функцию CONNECT, должно вызвать функцию DISCONNECT для сброса блоков управления CAF. Такой вызов функции DISCONNECT возвращает код возврата и код причины выполнения сброса (+004 и X'00C10824') и обеспечивает возможность выполнения требований соединения от этого задания при будущей активизации DB2.

Задание, которое не использовало явный вызов функции CONNECT, должно вызвать функцию CLOSE для сброса блоков управления CAF при завершении работы DB2.

В Табл. 82 показаны вызовы функции DISCONNECT в разных языках.

Таблица 82. Примеры вызовов функции CAF DISCONNECT

Язык	Пример вызова
Ассемблер	CALL DSNALI(FUNCTN,RETCODE,REASCODE)
C	fnret=dsnali(&functn[0], &retcode, &reascode);
COBOL	CALL 'DSNALI' USING FUNCTN RETCODE REASCODE.
FORTRAN	CALL DSNALI(FUNCTN,RETCODE,REASCODE)
PL/I	CALL DSNALI(FUNCTN,RETCODE,REASCODE);

**Примечание:** DSNALI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках С и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++     extern "OS" {
          int DSNALI(
              char * functn,
              ...); }

PL/I    DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

## Функция TRANSLATE: синтаксис и использование

Функцию TRANSLATE можно использовать для преобразования шестнадцатеричного кода причины DB2 в SQLCODE (целое число со знаком) и текст сообщения об ошибке. SQLCODE и текст сообщения помещаются в SQLCA вызывающей программы. Функцию TRANSLATE нельзя вызывать из программ на языке FORTRAN.

Функцию TRANSLATE имеет смысл использовать только при обнаружении ошибки после вызова функции OPEN, перед которой использовался явный вызов функции CONNECT. Для ошибок, возникающих при выполнении требований SQL или IFI, функция TRANSLATE выполняется автоматически.

```
►►CALL DSNALI—(—функция, sqlca
    [ ,—код_возврата
        [ ,—код_причины
    ] ]
)►►
```

*Рисунок 190. Функция DSNALI TRANSLATE*

Параметры указывают на следующие области:

## ФУНКЦИЯ

12-байтная область, содержащая слово TRANSLATE, после которого идут три пробела.

*sglca*

Область связи SQL (SQLCA) программы.

## *код возврата*

4-байтная область, в которую CAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, CAF помещает код возврата в регистр 15 и код причины в регистр 0.

## **код причины**

4-байтная область, в которую CAF помещает код причины. Если этот параметр не задан, CAF помещает код причины в регистр 0.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметр `код возврата`.

**Использование:** Используйте функцию TRANSLATE, чтобы получить соответствующий код ошибки SQL и текст сообщения для кода причины ошибки DB2, возвращаемого CAF в регистре 0 после вызова функции OPEN. DB2 помещает эту информацию в переменные хоста SQLCODE и SQLSTATE или в соответствующие поля SQLCA.

Функция TRANSLATE может преобразовывать коды причины CAF, начинающиеся с X'00F3', но не может преобразовывать коды причины, начинающиеся с X'00C1'. Если после вызова функции OPEN получен код причины ошибки X'00F30040' (*ресурс недоступен*) функция TRANSLATE возвращает в последних 44 символах поля SQLERRM имя недоступного объекта базы данных. Если данный код причины ошибки неизвестен функции DB2 TRANSLATE, она возвращает SQLCODE -924 (SQLSTATE '58006') и помещает в поле SQLERRM текстовую информацию о коде исходной функции DB2 и кодах возврата и причины ошибки. Содержимое регистров 0 и 15 не изменяется, за исключением случаев, когда возникает ошибка функции

TRANSLATE; в таких случаях в регистр 0 заносится значение X'C10205', а в регистр 15 – значение 200.

В Табл. 83 показаны вызовы функции TRANSLATE в разных языках.

Таблица 83. Примеры вызовов функции CAF TRANSLATE

Язык	Пример вызова
Ассемблер	CALL DSNALI,(FUNCTN,SQLCA,RETCODE, REASCODE)
C	fnret=dsnali(&functn[0], &sqlca, &retcode, &reascode);
COBOL	CALL 'DSNALI' USING FUNCTN SQLCA RETCODE REASCODE.
PL/I	CALL DSNALI(FUNCTN,SQLCA,RETCODE, REASCODE);

**Примечание:** DSNALI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках С и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
        int DSNALI(
            char * functn,
            ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

## Сводка поведения CAF

В Табл. 84 на стр. 793 кратко описано поведение CAF после получения от прикладных программ различных требований. Используйте эту информацию при разработке структуры программы, а также чтобы понять, когда могут возникать ошибки CAF. Внимательно используя эту информацию, можно избежать главных структурных ошибок в программах.

В этой таблице ошибка обозначается как *Ошибкаппп*. Соответствующий код причины – X'00C10'ппп; номер сообщения – DSNAпппI или DSNAпппE. Список кодов причины смотрите в разделе “Коды возврата и коды причины CAF” на стр. 796.

Таблица 84. Результаты вызовов CAF в зависимости от предыдущих вызовов

Предыдущие вызовы	Следующий вызов					
	CONNECT	OPEN	SQL	CLOSE	DISCONNECT	TRANSLATE
Отсутствует: это первый вызов	CONNECT	OPEN	CONNECT, OPEN, затем вызов SQL или IFI	Ошибка 203	Ошибка 204	Ошибка 205
CONNECT	Ошибка 201	OPEN	OPEN, затем вызов SQL или IFI	Ошибка 203	DISCONNECT	TRANSLATE
CONNECT, затем OPEN	Ошибка 201	Ошибка 202	Вызов SQL или IFI	CLOSE <sup>1</sup>	DISCONNECT	TRANSLATE
CONNECT, затем вызов SQL или IFI	Ошибка 201	Ошибка 202	Вызов SQL или IFI	CLOSE <sup>1</sup>	DISCONNECT	TRANSLATE
OPEN	Ошибка 201	Ошибка 202	Вызов SQL или IFI	CLOSE <sup>2</sup>	Ошибка 204	TRANSLATE
Вызов SQL или IFI	Ошибка 201	Ошибка 202	Вызов SQL или IFI	CLOSE <sup>2</sup>	Ошибка 204	TRANSLATE <sup>3</sup>

**Примечания:**

1. Соединения задания и адресного пространства остаются активными. Если из-за завершения DB2 возникает ошибка функции CLOSE, выполняется сброс блоков управления CAF, функция генерирует код ошибки 4 и код причины 'XX'00C10824' и CAF сможет выполнять другие требования соединения, когда DB2 будет снова активна.
2. Соединение для задания завершается. Если в этом адресном пространстве нет других соединенных заданий, завершается и соединение на уровне адресного пространства.
3. Можно вызвать функцию TRANSLATE, но в данном случае это излишне. CAF автоматически вызывает функцию TRANSLATE при ошибках требований SQL или IFI.

Табл. 84 имеет следующую структуру:

- В *верхней строке* перечислены возможные функции CAF, которые могут вызываться программами.
- В *первом столбце* перечислены самые последние требования соединения. Например, CONNECT, затем OPEN означает, что задание вызвало функцию CONNECT и затем функцию OPEN (не вызывая в промежутке между ними других функций CAF).
- На *пересечении строки и столбца* показан результат следующего вызова, если он идет после соответствующих предыдущих вызовов. Например, если это вызов OPEN и предыдущие вызовы – это CONNECT, результат будет OPEN: выполняется функция OPEN. Если это вызов SQL и предыдущие вызовы отсутствуют (то есть этот вызов SQL – это первый вызов CAF в программе), результатом будет неявное выполнение функций CONNECT и OPEN и затем выполнение функции SQL.

---

## Примеры сценариев

В этом разделе представлены примеры сценариев для соединения заданий с DB2.

### Одиночное задание с неявными соединениями

Самый простой сценарий соединения – одиночное задание,зывающее DB2 и не использующее явных операторов DSNALI CALL. Для этого задания неявно устанавливается соединение с подсистемой с именем по умолчанию и используется имя плана по умолчанию.

При завершении задания:

- Выполняется принятие всех изменений в базе данных (при нормальном завершении) или откат изменений (при ненормальном завершении).
- Освобождается активный план и все ресурсы базы данных.
- Завершаются соединения задания и адресного пространства с DB2.

### Одиночное задание с явными соединениями

Более сложный сценарий, но также с одним заданием:

```
CONNECT  
  OPEN      выделить план  
  вызов SQL или IFI  
  
:  
  CLOSE     освободить текущий план  
  OPEN      выделить новый план  
  вызов SQL или IFI  
  
:  
CLOSE  
DISCONNECT
```

Задание в каждый момент времени может иметь соединение с одной и только одной подсистемой DB2. Если имя подсистемы для функции OPEN не совпадает с именем подсистемы для функции CONNECT, возникнет ошибка CAF. Чтобы переключиться на другую подсистему, прикладная программа должна отсоединиться от текущей подсистемы, а затем выдать требование на соединение с новым именем подсистемы.

### Несколько заданий

В этом сценарии несколько заданий в адресном пространстве используют службы DB2. Каждое задание должно явно задавать одно и то же имя подсистемы с требованиями функций CONNECT или OPEN. Задание 1 не делает вызовов SQL или IFI. Оно предназначено для отслеживания ECB завершения и ECB запуска DB2, а также для проверки уровня выпуска DB2.

ЗАДАНИЕ 1	ЗАДАНИЕ 2	ЗАДАНИЕ 3	ЗАДАНИЕ n
CONNECT			
OPEN	OPEN	OPEN	
SQL	SQL	SQL	
...	...	...	
CLOSE	CLOSE	CLOSE	
OPEN	OPEN	OPEN	
SQL	SQL	SQL	
...	...	...	
CLOSE	CLOSE	CLOSE	
DISCONNECT			

## Обработчики для прикладной программы

Для прикладной программ можно задать обработчики для описанных ниже ситуаций.

### Обработчики ситуаций внимания

Обработчик ситуации внимания позволяет программе получать назад управление от DB2 во время долгого выполнения ошибочных требований, отсоединяя TCB, который в настоящее время ожидает выполнения требования SQL или IFI. DB2 обнаруживает ситуацию аварийного завершения, вызванную оператором DETACH, и выполняет для этого задания обработку завершения (включая ROLLBACK).

Утилита подключения по вызову не имеет обработчиков ситуаций внимания. При необходимости можно задать свои собственные обработчики. Однако DB2 использует процедуры FRR EUT, поэтому если обработчик ситуации внимания задается во время выполнения DB2, этот обработчик может не получить управление.

### Процедуры восстановления

Утилита подключения по вызову не имеет процедур восстановления при аварийном завершении.

Программа может задать процедуру обработки аварийного завершения. Она должна использовать индикаторы слежения, чтобы определить когда в процессе обработки DB2 возникает аварийное завершение. Если аварийное завершение возникает, когда управление имеет DB2, можно:

- Разрешить выполнение завершения задания. Не пытайтесь повторить выполнение программы. DB2 обнаруживает завершение задания и завершает поток, используя параметр ABRT. Производится откат и теряются все изменения базы данных, сделанные после последней точки SYNC или COMMIT.

Это единственное действие, которое можно выполнить для ситуаций аварийного завершения, вызванных CANCEL или DETACH. В таком случае нельзя использовать дополнительные операторы SQL. При попытке выполнить другой оператор SQL из прикладной программы или из ее процедуры восстановления возвращаются код возврата +256 и код причины X'00F30083'.

- Вызвать в процедуре ESTAE функцию CLOSE с параметром ABRT и затем функцию DISCONNECT. Процедура обработчика ESTAE может вызывать

повторное выполнение, поэтому нет необходимости перезапускать задание прикладной программы.

Стандартные процедуры функционального восстановления MVS (FRR) могут выполняться только в режиме блока требований обслуживания (SRB).

Поскольку DB2 не поддерживает вызовы из процедур в режиме SRB, в процедурах, вызывающих DB2, можно использовать только процедуры FRR EUT.

При использовании CAF для обработки требований SQL или вызовов IFI не должно быть активных EUT FRR.

Процедура EUT FRR может быть активной, но она не может повторять неудачные требования DB2. При повторении, вызванном процедурой EUT FRR, обходятся процедуры ESTAE системы DB2. Любое следующее требование DB2 (включая DISCONNECT) вызовет ошибку с кодом возврата +256 и код причинны X'00F30050'.

Если в среде MVS есть активная процедура EUT FRR, ошибки будут возникать при всех требованиях DB2, включая начальные вызовы CONNECT или OPEN. Эти ошибки вызваны тем, что DB2 всегда создает процедуру ESTAE типа ARR, а MVS/ESA не разрешает создание процедур ESTAE типа ARR при наличии активной процедуры FRR.

---

## Сообщения об ошибках и набор данных DSNTRACE

CAF не генерирует сообщения об ошибках, если не выделен набор данных DSNTRACE. Если набор данных DSNTRACE выделен динамически или оператором //DSNTRACE DD в задании JCL, CAF записывает в него диагностические сообщения трассировки. Пример задания JCL, выделяющей набор данных DSNTRACE, смотрите в разделе “Пример задания JCL, использующей CAF” на стр. 798. Номера сообщений трассировки содержат 3 последние цифры кодов причины.

---

## Коды возврата и коды причины CAF

CAF возвращает коды возврата и коды причины или в соответствующие параметры, заданные в вызове CAF, или, если эти параметры не заданы, в регистры 15 и 0. Подробное описание кодов причины смотрите в руководстве *DB2 Сообщения и коды*.

Если код причины начинается с X'00F3' (за исключением X'00F30006'), можно использовать функцию CAF TRANSLATE, чтобы получить текст сообщения об ошибке для печати или вывода на экран.

Для вызовов SQL утилита CAF возвращает в SQLCA стандартные коды SQLCODE. Список таких кодов возврата и их значений смотрите в разделе 2 руководства *DB2 Сообщения и коды*. CAF возвращает коды возврата и коды причины IFI в области IFCA.

Таблица 85 (Стр. 1 из 2). Коды возврата и коды причины CAF

Код возврата	Код причины	Объяснение
0	X'00000000'	Успешное выполнение.

Таблица 85 (Стр. 2 из 2). Коды возврата и коды причины CAF

Код возврата	Код причины	Объяснение
4	X'00C10823'	Несоответствие уровней выпусков DB2 и утилиты подключения по вызову.
4	X'00C10824'	Завершен сброс переменных CAF. Система готова к установлению нового соединения.
200 (примечание 1)	X'00C10201'	Получен второй вызов функции CONNECT от того же TCB. Первый вызов CONNECT мог быть неявным или явным.
200 (примечание 1)	X'00C10202'	Получен второй вызов функции OPEN от того же TCB. Первый вызов OPEN мог быть неявным или явным.
200 (примечание 1)	X'00C10203'	Вызвана функция CLOSE, но нет активного плана.
200 (примечание 1)	X'00C10204'	Вызвана функция DISCONNECT, но нет активного соединения.
200 (примечание 1)	X'00C10205'	Вызвана функция TRANSLATE, но нет соединения с DB2.
200 (примечание 1)	X'00C10206'	Неверное число параметров или не установлен бит конца списка.
200 (примечание 1)	X'00C10207'	Неправильный параметр функции.
200 (примечание 1)	X'00C10208'	От одного TCB получены требования на обращения к двум различным подсистемам DB2.
204 (примечание 2)		Системная ошибка CAF. Возможно, ошибка подключения или DB2.

**Примечания:**

1. Ошибка CAF, возможно, вызвана ошибками в списках параметров, передаваемыхзывающими программами. Ошибки CAF не изменяют текущее состояние соединения с DB2; можно продолжать обработку, используя правильное требование.
2. Системные ошибки вызывают аварийное завершение. Объяснение кодов причины аварийного завершения смотрите в разделе 4 руководства *DB2 Сообщения и коды*. Если включена трассировка, поясняющие сообщения записываются в набор данных DSNTRACE непосредственно перед аварийным завершением.

## Коды подкомпоненты поддержки подсистемы (X'00F3')

Эти коды причины генерируются поддержкой подсистемы для внешних областей памяти, частью подкомпонента поддержки подсистемы DB2, который обслуживает все соединения DB2 и требования. Дополнительную информацию об этих кодах, а также о кодах причины аварийного завершения и завершения подсистемы, выдаваемых другими частями поддержки подсистемы, смотрите в разделе 4 руководства *DB2 Сообщения и коды*.

## Примеры программ

Далее представлены примеры программ JCL и программ на ассемблере, обращающихся к утилите подключения по вызову (CAF).

## Пример задания JCL, использующей CAF

Ниже показан пример задания JCL – модели использования CAF в пакетной среде (не TSO). Оператор DSNTRACE, показанный в этом примере, необязателен.

```
//имя_задания      JOB    данные_карты_job_MVS
//CAFJCL          EXEC   PGM=прикладная_программа_CAF
//STEPLIB          DD     DSN=загрузочная_библиотека_программы
//                  DD     DSN=загрузочная_библиотека_DB2
//
//:
//SYSPRINT DD  SYSOUT=*
//DSNTRACE   DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
```

## Пример программы на ассемблере, использующей CAF

В следующих разделах показаны части примера программы на ассемблере, использующей утилиту подключения по вызову. В ней показаны основные способы реализации вызовов CAF, но не показаны код и макрокоманды MVS, необходимые для поддержки этих вызовов. Например, многие прикладные программы должны иметь структуру, состоящую из двух заданий, чтобы процедура обработки ситуаций внимания могла отсоединять соединенные задания с целью вернуть управление от DB2 к прикладной программе. Такая структура не показана в приведенном ниже примере.

В этой программе подразумевается существование макрокоманды WRITE. Там, где эта макрокоманда встречается в тексте этого примера программы, вместо нее можно задать собственные операторы. Решите, что должна делать прикладная программа в этих ситуациях; возможно, вывод сообщений об ошибках не требуется.

## Загрузка и удаление языкового интерфейса CAF

В следующем фрагменте кода показано, как прикладная программа загружает модули языкового интерфейса утилиты подключения по вызову с точками входа DSNALI и DSNHLI2. Сохранение этих точек входа в переменные LIALI и LISQL гарантирует, что прикладная программа загружает эти модули только один раз.

Удалите загруженный модуль, когда окончена его работа с DB2.

```
***** ПОЛУЧИТЬ АДРЕСА ТОЧЕК ВХОДА ЯЗЫКОВОГО ИНТЕРФЕЙСА ****
LOAD EP=DSNALI      Загрузить модуль с точкой входа функций CAF
ST   R0,LIALI        Сохранить эту точку входа для вызовов CAF
LOAD EP=DSNHLI2      Загрузить модуль с точкой входа CAF для SQL
ST   R0,LISQL        Сохранить эту точку входа для вызовов SQL
*   .
*   .      Вставьте здесь вызовы функций соединения и вызовы SQL
*   .
DELETE EP=DSNALI     Для правильного значения счетчика использования
DELETE EP=DSNHLI2     Для правильного значения счетчика использования
```

## Установление соединения с DB2

На рис. 191 показано, как выдаются явные требования для некоторых действий (CONNECT, OPEN, CLOSE, DISCONNECT и TRANSLATE) и используется процедура CHEKCODE для проверки кодов причины, возвращаемых CAF:

```
***** Вызов CONNECT *****
L R15,LIALI      Получить адрес языкового интерфейса
MVC FUNCTN,CONNECT  Получить код вызываемой функции
CALL (15),(FUNCTN,SSID,TECB,SECB,RIBPTR),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE   Проверить коды возврата и причины
CLC CONTROL,CONTINUE  Все в порядке?
BNE EXIT          Если нет (CONTROL не 'CONTINUE'), выход
USING R8,RIB       Подготовка к обращению к RIB
L R8,RIBPTR        Обращение к RIB, чтобы получить уровень
*                  выпуска DB2
WRITE 'Текущий уровень выпуска DB2 - ' RIBREL

***** Вызов OPEN *****
L R15,LIALI      Получить адрес языкового интерфейса
MVC FUNCTN,OPEN    Получить код вызываемой функции
CALL (15),(FUNCTN,SSID,PLAN),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE   Проверить коды возврата и причины

***** Вызовы SQL *****
* Вставьте здесь свои вызовы SQL. Прекомпилятор DB2
* генерирует вызовы точки входа DSNHLI. Нужно задать
* опцию прекомпилятора ATTACH(CAF) или создать фиктивную
* точку входа с этим именем для перехвата всех вызовов SQL.
* Фиктивная точка входа DSNHLI показана ниже.

***** Вызов CLOSE *****
CLC CONTROL,CONTINUE  Все в порядке?
BNE EXIT          Если нет (CONTROL не 'CONTINUE'), выход
MVC TRMOP,ABRT     Предполагаем завершение с параметром ABRT
L R4,SQLCODE       Поместить SQLCODE в регистр
C R4,CODE0         Проверить SQLCODE
BZ SYNCTERM        Если 0, вызвать CLOSE с параметром SYNC
C R4,CODE100        SQLCODE равен 100?
BNE DISC          Если нет, вызвать CLOSE в параметром ABRT
SYNCTERM MVC TRMOP,SYNC  Хороший код, завершение с параметром SYNC
DISC DS 0H          Создать список параметров CAF
L R15,LIALI        Получить адрес языкового интерфейса
MVC FUNCTN,CLOSE   Получить код вызываемой функции
CALL (15),(FUNCTN,TRMOP),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE   Проверить коды возврата и причины

***** Вызов DISCONNECT *****
CLC CONTROL,CONTINUE  Все в порядке?
BNE EXIT          Если нет (CONTROL не 'CONTINUE'), выход
L R15,LIALI        Получить адрес языкового интерфейса
MVC FUNCTN,DISCON  Получить код вызываемой функции
CALL (15),(FUNCTN),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE   Проверить коды возврата и причины
```

Рисунок 191. Процедура CHEKCODE для соединения с DB2

Здесь не показано задание, ожидающее ECB завершения DB2. При необходимости можно написать программу для такого задания и использовать макрокоманду MVS WAIT для слежения за ECB. Это задание может отсоединять код этого примера при получении ECB завершения. Задание может также ожидать ECB запуска DB2. Этот пример ожидает ECB запуска на уровне своего задания.

В начале этой программы примера подразумевается, что уже заданы некоторые переменные:

#### **Переменная хоста Использование**

<b>LIALI</b>	Точка входа, используемая для обработки требований соединения с DB2.
<b>LISQL</b>	Точка входа, используемая для обработки вызовов SQL.
<b>SSID</b>	Идентификатор подсистемы DB2.
<b>TECB</b>	Адрес ECB завершения DB2.
<b>SECB</b>	Адрес ECB запуска DB2.
<b>RIBPTR</b>	Полное слово, в которое CAF заносит адрес RIB.
<b>PLAN</b>	Имя плана, используемого в вызове OPEN.
<b>CONTROL</b>	Используется для обработки завершения, вызванного ошибочными кодами возврата или причины. Значение переменной CONTROL задается в процедуре CHEKCODE.
<b>CAFCALL</b>	Область, содержащая список параметров для макрокоманды CALL.

#### **Проверка кодов возврата и кодов причины**

На рис. 192 на стр. 801 показан способ проверки кодов возврата и ECB завершения DB2 после каждого требования соединения или вызова SQL. Эта процедура задает значение переменной CONTROL, используемой для управления работой программы.

```

*****
* ПСЕВДОКОД ПРОЦЕДУРЫ CHEKCODE
*****
*IF передан ECB завершения с кодом ABTERM или FORCE
* THEN
*   CONTROL = 'SHUTDOWN'
*   WRITE 'DB2 обнаружила FORCE или ABTERM, завершаем работу'
* ELSE                                /* ECB завершения не передан */
*   SELECT (RETCODE)                  /* Взять код возврата */
*   WHEN (0) ;                      /* Ничего не делать; все в порядке */
*   WHEN (4) ;                      /* Предупреждение */
*   SELECT (REASCODE)                /* Взять код причины */
*   WHEN ('00C10823'X)              /* Несоответствие выпусков DB2 и CAF */
*         WRITE 'Обнаружено несоответствие между уровнями выпусков DB2 и CAF'
*   WHEN ('00C10824'X)              /* CAF готово к другим вызовам */
*         CONTROL = 'RESTART'        /* Начать сначала */
*   OTHERWISE
*     WRITE 'Неизвестное значение R0 при R15 = 4'
*     CONTROL = 'SHUTDOWN'
*     END INNER-SELECT
*     WHEN (8,12)                   /* Ошибка соединения */
*       SELECT (REASCODE)          /* Взять код причины */
*       WHEN ('00F30002'X,          /* Система DB2 не активна, но при */
*             '00F30012'X)          /* активизации передаст ECB запуска*/
*     DO
*       WRITE 'DB2 недоступна. Сообщу, когда она будет активна.'
*       WAIT SECB                 /* Ожидать запуска DB2 */
*       WRITE 'DB2 вновь доступна.'
*     END
*     /* Вставьте здесь проверки других ошибок соединения DB2. */
*     /* В спецификации CAF перечислены все коды, которые могут */
*     /* быть возвращены. Напишите обработку требуемых кодов. */
*     OTHERWISE                   /* Обнаружен непредвиденный код */
*       WRITE 'Предупреждение: Ошибка соединения DB2. Причина неизвестна'
*       CALL DSNALI ('TRANSLATE',SQLCA) /* Заполнить SQLCA */
*       WRITE SQLCODE и SQLERRM
*     END INNER-SELECT
*     WHEN (200)
*       WRITE 'CAF обнаружила пользовательскую ошибку. Смотрите DSNTRACE'
*     WHEN (204)
*       WRITE 'Системная ошибка CAF. Смотрите набор данных DSNTRACE'
*     OTHERWISE
*       CONTROL = 'SHUTDOWN'
*       WRITE 'Получен неизвестный код возврата'
*     END MAIN SELECT
*     IF (RETCODE > 4) THEN        /* Возникла ошибка соединения? */
*       CONTROL = 'SHUTDOWN'
*   END CHEKCODE

```

*Рисунок 192 (Часть 1 из 3). Подпрограмма на ассемблере для проверки кодов возврата CAF и DB2*

```

*****
* Процедура CHEKCODE проверяет коды возврата DB2 и CAF.
* Когда CHEKCODE получает управление, регистр R13 должен указывать на
* область сохранения вызывающей программы.
*****
CHEKCODE DS 0H
    STM R14,R12,12(R13) Пролог процедуры
    ST  R15,RETCODE Сохранить код возврата
    ST  R0,REASCODE Сохранить код причины
    LA  R15,SAVEAREA Получить адрес области сохранения
    ST  R13,4(,R15) Связать в цепочку области сохранения
    ST  R15,8(,R13) Связать в цепочку области сохранения
    LR  R13,R15 Поместить в R13 адрес области сохранения
*
***** ПРОВЕРКА НА FORCE ИЛИ ABTERM *****
    TM  TECB,POSTBIT Был передан ECB завершения?
    BZ  DOCHECKS Если нет - переход
    CLC  TECBCODE(3),QUIESCE Проверка STOP DB2 MODE=FORCE?
    BE  DOCHECKS Если не QUIESCE, а FORCE или ABTERM
    MVC  CONTROL,SHUTDOWN Завершение
    WRITE 'Обнаружен FORCE или ABTERM, завершаем работу'
    B  ENDCODE Переход в конец CHEKCODE
DOCHECKS DS 0H Проверить RETCODE и REASCODE
*
***** RETCODE = 0? *****
    CLC  RETCODE,ZERO Равно 0?
    BE  ENDCODE Ничего делать не нужно - в конец CHEKCODE
*
***** RETCODE = 4? *****
    CLC  RETCODE,FOUR Равно 4?
    BNE HUNT8 Если не равно 4 - переход на HUNT8
    CLC  REASCODE,C10823 Несоответствие уровней выпусков?
    BNE HUNT824 Если нет - переход на HUNT824
    WRITE 'Обнаружено несоответствие между уровнями выпусков DB2 и CAF'
    B  ENDCODE Все сделано. Переход в конец CHEKCODE
HUNT824 DS 0H Код причины - 'сброс CAF'?
    CLC  REASCODE,C10824 Проверяем код причины. Готов к перезапуску?
    BNE UNRECOG Если не 824, то это неизвестный код
    WRITE 'CAF готово получать новые требования'
    MVC  CONTROL,RESTART Необходимо заново выполнить CONNECT
    B  ENDCODE Все сделано. Переход в конец CHEKCODE
UNRECOG DS 0H
    WRITE 'Получен RETCODE = 4 и неизвестный код причины'
    MVC  CONTROL,SHUTDOWN Завершение, серьезная ошибка
    B  ENDCODE Все сделано. Переход в конец CHEKCODE
*
***** RETCODE = 8? *****
HUNT8 DS 0H
    CLC  RETCODE,EIGHT Код возврата равен 8?
    BE  GOT80R12
    CLC  RETCODE,TWELVE Код возврата равен 12?
    BNE HUNT200
GOT80R12 DS 0H Код возврата равен 8 или 12
    WRITE 'Получен RETCODE 8 или 12'
    CLC  REASCODE,F30002 Код причины равен X'00F30002'?
    BE  DB2DOWN

```

*Рисунок 192 (Часть 2 из 3). Подпрограмма на ассемблере для проверки кодов возврата CAF и DB2*

```

CLC REASCODE,F30012 Код причины равен X'00F30012'?
BE DB2DOWN
WRITE 'Ошибка соединения DB2 с неизвестным REASCODE'
CLC SQLCODE,ZERO Нужно ли выполнять TRANSLATE
BNE A4TRANS Если не пустое, пропустить TRANSLATE
* ***** Вызываем TRANSLATE для неизвестных кодов возврата ***
WRITE 'SQLCODE=0, а R15 нет - вызываем TRANSLATE, чтобы получить SQLCODE'
L R15,LALI Получить адрес языкового интерфейса
CALL (15),(TRANSLAT,SQLCA),VL,MF=(E,CAFSCALL)
C R0,C10205 Функция TRANSLATE выполнена успешно?
BNE A4TRANS Если не C10205, SQLERRM содержит информацию
WRITE 'Не удалось выполнить TRANSLATE для ошибки соединения'
B ENDCCODE Переход в конец CHEKCODE
A4TRANS DS 0H SQLERRM заполнена информацией
* Примечание: возможно, ваша программа должна удалить
* разделители X'FF' и отформатировать полученное
* содержимое области SQLERRM.
* Можно также использовать для форматирования сообщения
* пример прикладной программы DB2 DSNTIAR.
WRITE 'SQLERRM: ' SQLERRM
B ENDCCODE Все сделано. Переход в конец CHEKCODE
DB2DOWN DS 0H DB2 недоступна
WRITE 'DB2 не активна; сообщу, когда она станет активна'
WAIT ECB=SECB Ждать активации DB2
WRITE 'DB2 вновь активна'
MVC CONTROL,RESTART Необходимо заново выполнить CONNECT
B ENDCCODE
* ***** RETCODE = 200? *****
HUNT200 DS 0H Метка: Проверка RETCODE = 200
CLC RETCODE,NUM200 Проверяем RETCODE = 200?
BNE HUNT204
WRITE 'CAF обнаружила пользовательскую ошибку, смотрите DSNTRACE'
B ENDCCODE Все сделано. Переход в конец CHEKCODE
* ***** RETCODE = 204? *****
HUNT204 DS 0H Метка: Проверка RETCODE = 204
CLC RETCODE,NUM204 Проверяем RETCODE = 200?
BNE WASSAT Если не 204, это неизвестный код
WRITE 'Системная ошибка CAF, смотрите набор данных DSNTRACE'
B ENDCCODE Все сделано. Переход в конец CHEKCODE
* ***** НЕИЗВЕСТНЫЙ КОД ВОЗВРАТА *****
WASSAT DS 0H
WRITE 'Получен неизвестный RETCODE'
MVC CONTROL,SHUTDOWN Завершение
BE ENDCCODE Все сделано. Переход в конец CHEKCODE
ENDCCODE DS 0H Нужно ли завершить программу?
L R4,RETCODE Получить копию RETCODE
C R4,FOUR Проверить RETCODE
BNH BYEBYE Если RETCODE <= 4, выйти из CHEKCODE
MVC CONTROL,SHUTDOWN Завершение
BYEBYE DS 0H Выход из CHEKCODE
L R13,4(,R13) Указатель на область сохранения
* вызывающей программы)
RETURN (14,12) Возврат в вызывающую программу

```

Рисунок 192 (Часть 3 из 3). Подпрограмма на ассемблере для проверки кодов возврата CAF и DB2

## Использование фиктивной точки входа DSNHLI

Каждое средство подключения DB2 содержит точку входа с именем DSNHLI. Если используется CAF, но не задана опция прекомпилиатора ATTACH(CAF), для операторов SQL программы генерируются инструкции BALR вызовов DSNHLI. Чтобы задать правильную точку входа DSNHLI, не включая DSNALI в загрузочный модуль, напишите процедуру, имеющую точку входа с именем DSNHLI и передающую управление точке входа DSNHLI2 в модуле DSNALI. DSNHLI2 – это уникальное имя точки входа для DSNALI и эта точка входа располагается в DSNALI в том же месте, что и точка входа DSNHLI. DSNALI использует 31–битную адресацию. Если прикладная программа, вызывающая эту промежуточную процедуру, использует 24–битную адресацию, промежуточная процедура должна выполнять преобразования.

В следующем примере адрес LISQL доступен, поскольку в этом вызывающем CSECT используется тот же регистр 12, что и в CSECT DSNHLI. Аналогично при написании прикладной программы необходимо обеспечить доступность адреса LISQL.

```
*****
* Процедура DSNHLI перехватывает вызовы к точке входа DSNHLI языкового
* интерфейса
*****
DS      0D
DSNHLI  CSECT
        STM   R14,R12,12(R13)    Начало CSECT
        LA    R15,SAVEHLI       Пролог процедуры
        ST    R13,4(,R15)       Получить адрес области сохранения
        ST    R15,8(,R13)       Связать в цепочку области сохранения
        LR    R13,R15          Связать в цепочку области сохранения
        L     R15,LISQL         Поместить в R13 адрес области сохранения
        BASSM R14,R15          Получить адрес настоящей точки входа DSNHLI
                                Вызов DSNALI, выполняющего вызов SQL
                                DSNALI использует 31-битный режим,
                                поэтому используйте BASSM, чтобы
                                обеспечить предохранение режима адресации.
*
*                                         Восстановить R13 (адрес области сохранения
*                                         вызывающей программы)
*
*                                         Восстановить R14 (адрес возврата)
*
*                                         Восстановить R1-12, но НЕ R0 и R15 (коды)
        L     R13,4(,R13)
        L     R14,12(,R13)
        RETURN (1,12)
```

## Объявления переменных

На рис. 193 на стр. 805 показаны объявления некоторых переменных, используемых в предыдущих подпрограммах.

```

***** ПЕРЕМЕННЫЕ *****
SECB    DS   F           ECB запуска DB2
TECB    DS   F           ECB завершения DB2
LIALI   DS   F           Адрес точки входа DSNALI
LISQL   DS   F           Адрес точки входа DSNHLI2
SSID    DS   CL4          Имя подсистемы DB2. Параметр CONNECT
PLAN    DS   CL8          Имя плана DB2. Параметр OPEN
TRMOP   DS   CL4          Опция завершения для CLOSE (SYNC|ABRT)
FUNCTN  DS   CL12         Вызываемая функция CAF
RIBPTR  DS   F           Здесь DB2 помещает адрес блока информации
*           о выпуске
RETCODE  DS   F           Здесь CHEKCODE сохраняет R15
REASCODE DS   F           Здесь CHEKCODE сохраняет R0
CONTROL  DS   CL8          GO, SHUTDOWN или RESTART
SAVEAREA DS   18F          Область сохранения для CHEKCODE
***** КОНСТАНТЫ *****
SHUTDOWN DC  CL8'SHUTDOWN'  Значение CONTROL: Завершить работу
RESTART   DC  CL8'RESTART '  Значение CONTROL: Перезапуск
CONTINUE   DC  CL8'CONTINUE'  Значение CONTROL: Все хорошо, продолжать
CODE0     DC  F'0'          SQLCODE = 0
CODE100   DC  F'100'         SQLCODE = 100
QUIESCE   DC  XL3'000008'    Код ECB завершения: STOP DB2 MODE=QUIESCE
CONNECT   DC  CL12'CONNECT'  ' Имя службы CAF. Должно быть CL12!
OPEN      DC  CL12'OPEN'     ' Имя службы CAF. Должно быть CL12!
CLOSE     DC  CL12'CLOSE'    ' Имя службы CAF. Должно быть CL12!
DISCON    DC  CL12'DISCONNECT'  ' Имя службы CAF. Должно быть CL12!
TRANSLAT  DC  CL12'TRANSLATE'  ' Имя службы CAF. Должно быть CL12!
SYNC      DC  CL4'SYNC'      Опция завершения (COMMIT)
ABRT      DC  CL4'ABRT'      Опция завершения (ROLLBACK)
***** КОДЫ ВОЗВРАТА (R15) ОТ CAF *****
ZERO     DC  F'0'          0
FOUR     DC  F'4'          4
EIGHT    DC  F'8'          8
TWELVE   DC  F'12'         12 (Код возврата CAF в R15)
NUM200   DC  F'200'        200 (Пользовательская ошибка)
NUM204   DC  F'204'        204 (Системная ошибка CAF)
***** КОДЫ ПРИЧИНЫ (R00) ОТ CAF *****
C10205   DC  XL4'00C10205'  Ошибка TRANSLATE
C10823   DC  XL4'00C10823'  Несоответствие выпусков
C10824   DC  XL4'00C10824'  CAF готово продолжать работу
F30002   DC  XL4'00F30002'  Подсистема DB2 не активна
F30011   DC  XL4'00F30011'  Подсистема DB2 не активна
F30012   DC  XL4'00F30012'  Подсистема DB2 не активна
F30025   DC  XL4'00F30025'  DB2 завершает работу (REASCODE)
*
*           Вставьте здесь дополнительные коды, если они нужны для вашей
*           прикладной программы
*
***** SQLCA и RIB *****
EXEC SQL INCLUDE SQLCA
DSNDRIB          Получить блок информации о выпуске DB2
***** Список параметров макрокоманды CALL ****
CAFCALL  CALL  ,(*,*,*,*,*,*,*),VL,MF=L

```

Рисунок 193. Объявления переменных, используемых в предыдущих подпрограммах



---

## Глава 7–8. Программирование для утилиты подключения служб управления восстановимыми ресурсами (RRSAF)

Прикладная программа может использовать утилиту подключения служб управления восстановимыми ресурсами (RRSAF) для соединения с DB2 и использования DB2 для обработки операторов SQL, команд и вызовов интерфейса IFI (instrumentation facility interface – интерфейс инструментальных средств). RRSAF могут использовать программы, выполняющиеся как пакетные программы MVS, приоритетные программы TSO и фоновые программы TSO.

RRSAF использует службы управления транзакциями и менеджера восстановимых ресурсов OS/390 (OS/390 RRS). Используя RRSAF, можно согласовать обновления, производимые DB2, с обновлениями, производимыми другими менеджерами ресурсов, которые также используют OS/390 RRS в системе MVS.

**Необходимые знания:** Перед тем как рассматривать использование RRSAF, необходимо изучить следующие темы, касающиеся MVS:

- Макрокоманда CALL и стандартное соглашение о связях между модулями
- Адресация программ и опции резидентности (AMODE и RMODE)
- Создание заданий и управление заданиями; многозадачность
- Средства функционального восстановления, такие как ESTAE, ESTAI и FRR
- Методы синхронизации, такие как WAIT/POST.
- Функции OS/390 RRS, такие как SRRCMIT и SRRBACK.

---

### Возможности и ограничения RRSAF

Чтобы решить, нужно ли использовать RRSAF, рассмотрите следующие возможности и ограничения.

### Возможности прикладных программ RRSAF

Прикладная программа, использующая RRSAF, может:

- Использовать системные средства авторизации MVS и внешние продукты защиты (например, RACF) для регистрации в DB2 с ID авторизации конечного пользователя.
- Зарегистрироваться в DB2, используя новый ID авторизации и существующие соединение и план.
- Обращаться к DB2 из нескольких заданий MVS в одном адресном пространстве.
- Переключать поток DB2 между заданиями MVS в одном адресном пространстве.
- Обращаться к IFI DB2.
- Выполняться с программой монитора терминала TSO (TMP) или без нее.
- Выполняться, не являясь подзаданием командного процессора DSN (или какой-либо программы DB2).

- Выполняться в области памяти выше или ниже 16 Мбайт.
- Устанавливать явное соединение с DB2 через интерфейс вызовов и управлять точным состоянием соединения.
- Задавать блоки управления событиями (ECB), которые система DB2 передает при запуске или при завершении ее работы.
- Перехватывать коды возврата, коды причины и коды аварийного завершения от DB2 и преобразовывать их в текстовые сообщения.

## **Возможности заданий**

Любое задание в адресном пространстве может установить соединение с DB2 через RRSAF.

**Число соединений с DB2:** Каждый блок управления заданием (TCB) может установить только одно соединение с DB2. Требование к службам DB2, выданное программой, которая выполняется под данным заданием, использует соединение с DB2 этого задания. Это требование к службам выполняется независимо от любых других операций DB2 для любых других заданий.

При использовании нескольких одновременных соединений может возрасти вероятность тупиковых ситуаций при работе с ресурсами DB2. Учтите это при написании прикладных программ.

**Задание плана для задания:** Каждое использующее соединение задание может запускать план. Задания в одном адресном пространстве могут задавать один план, но каждый экземпляр плана будет выполняться независимо от других. Задание может завершить свой план и выполнять другой план, не прерывая полностью своего соединения с DB2.

**Обработчики ситуаций внимания и процедуры восстановления:** RRSAF не создает структуры задания и не содержит обработчиков ситуаций внимания и процедур функционального восстановления. Можно задать любые требуемые обработчики ситуаций внимания и процедуры функционального восстановления, но необходимо использовать только процедуры восстановления типа ESTAE/ESTAI.

## **Язык программирования**

Прикладная программа RRSAF может быть написана на языках ассемблер, С, COBOL, FORTRAN и PL/I. При выборе языка для написания прикладной программы учтите следующие ограничения:

- Если используются макрокоманды MVS (ATTACH, WAIT, POST и т.д.), необходимо выбрать язык программирования, который их поддерживает.
- Функция RRSAF TRANSLATE недоступна из языка FORTRAN. Чтобы использовать эту функцию, напишите вызывающую ее процедуру на другом языке и затем вызовите эту процедуру из языка FORTRAN.

## **Утилита трассировки**

Средство трассировки генерирует диагностические сообщения, помогающие при отладке программы и диагностике ошибок в программах RRSAF. Информация трассировки доступна только в виде дампа SYSABEND или SYSUDUMP.

## **Подготовка программ**

Подготовка прикладной программы для выполнения в RRSAF аналогична подготовке для выполнения в других средах, например, CICS, IMS и TSO. Подготовку прикладной программы RRSAF можно произвести в пакетной среде или через процесс подготовки программы DB2. Систему подготовки программы можно использовать или через DB2I, или через DSNH CLIST. Примеры и рекомендации по подготовке программ смотрите в разделе “Глава 6–1. Подготовка прикладной программы к запуску” на стр. 435.

## **Требования RRSAF**

При написании прикладных программ, использующих RRSAF, учтите следующие характеристики.

### **Размер программы**

Для кода RRSAF требуется около 10 Кбайт виртуальной памяти на каждое адресное пространство и дополнительные 10 Кбайт на каждый TCB, использующий RRSAF.

### **Использование макрокоманды LOAD**

RRSAF использует макрокоманду MVS SVC LOAD для загрузки модуля при инициализации после первого требования к службам. Модуль загружается в защищенную от выборки память, имеющую ключ защиты шага задания. Если локальная среда перехватывает и замещает вызов LOAD SVC, необходимо убедиться, что используемая версия LOAD работает с цепочками элементов списка загрузки (LLE) и записей каталога содержания (CDE) аналогично стандартной макрокоманде MVS LOAD.

### **Операции принятия и отката**

Для принятия единицы работы используйте в прикладной программе RRSAF функцию CPIC SRRCMIT или оператор DB2 COMMIT. Для отката единицы работы используйте функцию CPIC SRRBACK или оператор DB2 ROLLBACK. Информацию об использовании функций SRRCMIT и SRRBACK смотрите в руководстве *OS/390 MVS Programming: Callable Services for High-Level Languages*.

Для выбора использования для выполнения операций принятия и отката операторов DB2 или функций CPIC следуйте таким правилам:

- Используйте операторы DB2 COMMIT и ROLLBACK, если известно, что:
  - Восстановимые ресурсы, к которым обращается прикладная программа, представляют собой только данные DB2, управляемые одним экземпляром DB2.
  - Адресное пространство, из которого запускается обработка точки синхронизации – это то же самое адресное пространство, из которого выполнено соединение с DB2.

- Если прикладная программа обращается к другим восстановимым ресурсам или если обработка точек синхронизации и доступ к DB2 запускаются из разных адресных пространств, используйте функции SRRCMIT и SRRBACK.

## **Среда выполнения**

Прикладная программа, обращающаяся к службам DB2, должна удовлетворять некоторым требованиям среды выполнения. Эти требования должны выполняться вне зависимости от используемого средства подключения. Они относятся не только к RRSAF.

- Прикладная программа должна выполняться в режиме TCB.
- При обращениях прикладной программы к службам DB2 не должны быть активны EUT FRR. Если активен какой-либо EUT FRR, функциональное восстановление DB2 может вызвать ошибку и прикладная программа может получить непредсказуемый код аварийного завершения.
- Различные средства подключения не могут быть одновременно активны в одном адресном пространстве. Например:
  - Прикладная программа не должна использовать RRSAF в адресных пространствах CICS или IMS.
  - Прикладная программа, выполняющаяся в адресном пространстве, имеющем соединение с DB2 при помощи CAF, не может соединяться с DB2 при помощи RRSAF.
  - Прикладная программа, выполняющаяся в адресном пространстве, имеющем соединение с DB2 при помощи RRSAF, не может соединяться с DB2 при помощи CAF.
- Одно средство подключения не может запускать другое средство подключения. Это означает, что прикладная программа RRSAF не может использовать DSN, а подкоманда DSN RUN не может вызывать прикладную программу RRSAF.
- Модуль DSNRLI языкового интерфейса для RRSAF поставляется скомпонованным с атрибутами AMODE(31) и RMODE(ANY). Если прикладная программа загружает RRSAF в область ниже 16 Мбайт, необходимо заново скомпоновать модуль DSNRLI.

## **Как использовать RRSAF**

Для использования RRSAF необходимо сначала сделать доступным загрузочный модуль DSNRLI языкового интерфейса RRSAF. Информацию о загрузке или компоновке этого модуля смотрите в разделе “Обращение к языковому интерфейсу RRSAF” на стр. 811.

Программа использует RRSAF, выполняя операторы CALL DSNRLI с соответствующими опциями. Общий вид этих операторов смотрите в разделе “Описание функций RRSAF” на стр. 815.

Первый элемент каждого списка опций – **функция**, описывающая действие, которое должна выполнить RRSAF. Список возможных функций и описания их действий смотрите в разделе “Сводка функций соединения” на стр. 814. Результат выполнения любой функции в какой-то степени зависит от того, какие функции уже были выполнены программой. Перед использованием

какой-либо функции не забудьте прочитать описание ее использования. Прочтайте также раздел “Сводка функций соединения” на стр. 814, в котором описывается влияние выполненных ранее функций.

## Обращение к языковому интерфейсу RRSAF

На рис. 194 показана общая структура RRSAF и использующей его программы.

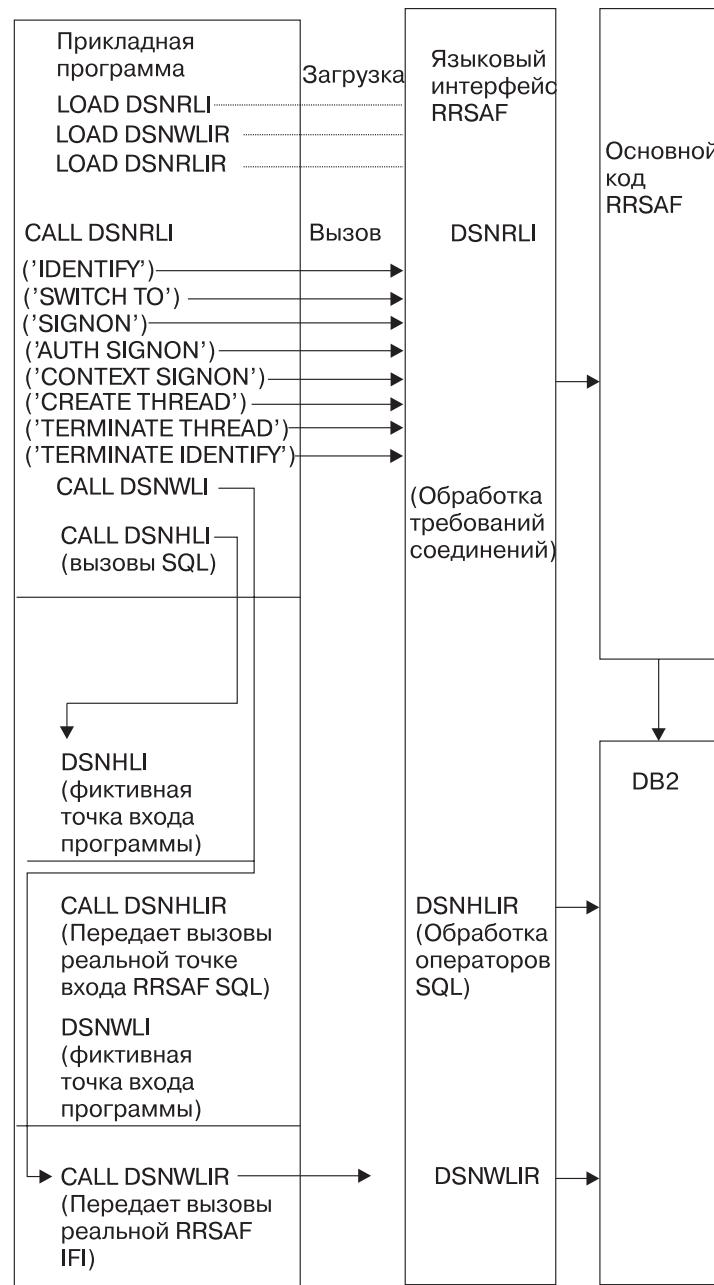


Рисунок 194. Пример конфигурации RRSAF

Часть RRSAF – загрузочный модуль DB2 DSNRLI, модуль языкового интерфейса RRSAF. Алиасы DSNRLI – DSNHLIR и DSNWLIR. Этот модуль имеет пять точек входа: DSNRLI, DSNHLI, DSNHLIR, DSNWLI и DSNWLIR:

- Точка входа DSNRLI используется для обработки явных требований на соединение DB2.
- DSNHLI и DSNHLIR используются для обработки вызовов SQL. DSNHLI применяется, если прикладная программа использует скомпонованный модуль RRSAF; DSNHLIR применяется, если прикладная программа использует загружаемый модуль RRSAF.
- DSNWLI и DSNWLIR используются для обработки вызовов IFI. DSNWLI применяется, если прикладная программа использует скомпонованный модуль RRSAF; DSNWLIR применяется, если прикладная программа использует загружаемый модуль RRSAF.

Для обращения к модулю DSNRLI можно использовать явные вызовы макрокоманды LOAD при выполнении программы или же включить модуль DSNRLI в ее загрузочный модуль в процессе компоновки программы. Каждый из этих способов имеет свои достоинства и недостатки.

### **Явная загрузка DSNRLI**

Чтобы загрузить DSNRLI, выполните макрокоманду MVS LOAD для точек входа DSNRLI и DSNHLIR. Если используются функции IFI, необходимо также загрузить DSNWLIR. Сохраните адрес точки входа, возвращаемый макрокомандой LOAD, и используйте его в макрокомандах CALL.

Явная загрузка модуля DSNRLI позволяет сделать обслуживание прикладной программы независимым от будущих изменений в языковом интерфейсе. Если языковый интерфейс будет изменен, это скорее всего не повлияет на работу вашего загрузочного модуля.

Необходимо указать DB2, какую точку входа следует использовать. Это можно сделать одним из двух способов:

- Задайте опцию прекомпилятора ATTACH(RRSAF).
- DB2 генерирует вызовы для точки входа DSNHLIR. Эту опцию нельзя использовать в прикладных программах на языке FORTRAN.
- Задайте в программе для вашего загрузочного модуля фиктивную точку входа с именем DSNHLI.

Если не задана опция прекомпилятора ATTACH, прекомпилятор DB2 генерирует для каждого оператора SQL вызовы для точки входа DSNHLI. Прекомпилятор не знает, какое будет использоваться средство подключения DB2, и его работа не зависит от этого. Когда вызовы, сгенерированные прекомпилятором DB2, передают управление точке входа DSNHLI, ваш код, соответствующий этой фиктивной точке входа, должен сохранить список опций, передаваемый в R1, и вызвать DSNHLIR с тем же списком опций. Пример программного кода для фиктивной точки входа DSNHLI смотрите в разделе “Использование фиктивной точки входа DSNHLI” на стр. 845.

## **Компоновка DSNRLI**

DSNRLI можно включить в ваш загрузочный модуль при компоновке. Например, в задании JCL можно использовать такой управляющий оператор компоновщика:

```
INCLUDE DB2LIB(DSNRLI).
```

Задание этого оператора предотвращает компоновку неправильного модуля языкового интерфейса.

Если DSNRLI включен во время компоновки, не нужно включать в программу фиктивную точку входа DSNHLI или задавать опцию прекомпилятора ATTACH. Модуль DSNRLI содержит точку входа для DSNHLI, которая идентична точке входа DSNHLIR, и точку входа DSNWLI, которая идентична точке входа DSNWLIR.

Недостаток компоновки DSNRLI в загрузочный модуль в том, что если IBM внесет в модуль DSNRLI какие-либо изменения, необходимо будет заново скомпоновать программу.

## **Основные свойства соединений RRSAF**

Некоторые основные свойства соединения RRSAF с DB2:

**Имя соединения и тип соединения:** Имя соединения и тип соединения имеют значение RRSAF. Можно использовать команду DISPLAY THREAD, чтобы получить список прикладных программ RRSAF, имеющих имя соединения RRSAF.

**ID авторизации:** Для каждого соединения DB2 есть набор ID авторизации. Соединение должно иметь первичный ID авторизации и может иметь один или несколько вторичных ID авторизации. Эти идентификаторы используются для:

- Проверки полномочий на доступ к DB2
- Проверки привилегий на операции с объектами DB2
- Задания владельцев объектов DB2
- Задания информации о пользователе соединения для трассировок контроля, производительности и учета.

Для проверки и задания полномочий для ID авторизации RRSAF использует системное средство авторизации MVS (SAF) и продукты защиты, например RACF. Прикладная программа, соединяющаяся с DB2 через RRSAF, должна послать свои идентификаторы средству SAF для проверки идентификаторов и их полномочий. RRSAF получает идентификаторы от SAF.

Локальная система может обеспечивать для соединения DB2 процедуру обработки, которая используется для изменения ID авторизации и сообщает, разрешено ли соединение. Реальные значения первичного и вторичных ID авторизации могут отличаться от значений, возвращаемых требованием SIGNON или AUTH SIGNON. Процедура обработки локальной системы для регистрации в DB2 может обращаться к первичному и вторичным ID авторизации и изменять эти ID, чтобы они удовлетворяли требованиям защиты этой системы. Эта процедура также может сообщать, нужно ли выполнять требование на регистрацию.

Информацию о ID авторизации и процедурах обработки для соединения и регистрациисмотрите в приложении В (том 2) руководства *DB2 Administration Guide*.

**Область видимости:** RRSAF обрабатывает соединения так, как будто каждое задание работает совершенно независимо. Когда задание вызывает какую-либо функцию, RRSAF передает этот вызов системе DB2 независимо от состояния соединений других заданий в это адресное пространство. Однако прикладная программа и подсистема DB2 имеют доступ к информации о состоянии соединений всех заданий в адресном пространстве.

В одном адресном пространстве нельзя совмещать соединения RRSAF с другими типами соединений. Первое соединение с DB2, сделанное из данного адресного пространства, задает тип разрешенных соединений.

### **Завершение задания**

Если прикладная программа, соединенная с DB2 через RRSAF завершается нормально до того, как функции TERMINATE THREAD или TERMINATE IDENTIFY освободят план, OS/390 RRS выполняет принятие всех изменений, сделанных после последней точки принятия.

Если прикладная программа, соединенная с DB2 через RRSAF, завершается ненормально до того, как функции TERMINATE THREAD или TERMINATE IDENTIFY освободят план, OS/390 RRS выполняет откат всех изменений, сделанных после последней точки принятия.

В обоих случаях DB2 освобождает план, если это необходимо, и завершает соединение для этой прикладной программы.

### **Аварийное завершение DB2**

Если в процессе выполнения прикладной программы происходит аварийное завершение DB2, система DB2 производит откат изменений к последней точке принятия. Если работа DB2 завершается во время обработки требования принятия, DB2 произведет принятие или откат всех изменений при последующем ее запуске. Выполняемая при этом операция зависит от состояния выполнения требования принятия в момент завершения работы системы DB2.

## **Сводка функций соединения**

С вызовами CALL DSNRLI можно использовать следующие функции:

### **IDENTIFY**

Задает задание как пользователя указанной подсистемы DB2. Когда первое задание в данном адресном пространстве выдает требование на соединение, это адресное пространство инициализируется как пользователь DB2. Смотрите раздел “Функция IDENTIFY: синтаксис и использование” на стр. 817.

### **SWITCH TO**

Направляет требования RRSAF, SQL или IFI к указанной подсистеме DB2. Смотрите раздел “Функция SWITCH TO: синтаксис и использование” на стр. 819.

### **SIGNON**

Передает системе DB2 ID пользователя и, возможно, один или несколько вторичных ID авторизации, для этого соединения. Смотрите раздел “Функция SIGNON: синтаксис и использование” на стр. 822.

### **AUTH SIGNON**

Передает системе DB2 ID пользователя, элемент Accessor Environment Element (ACEE) и, возможно, один или несколько вторичных ID авторизации для этого соединения. Смотрите раздел “Функция AUTH SIGNON: Синтаксис и использование” на стр. 825.

### **CONTEXT SIGNON**

Передает системе DB2 ID пользователя и, возможно, один или несколько вторичных ID авторизации, для этого соединения. Функцию CONTEXT SIGNON можно выполнять из неавторизованных программ. Смотрите раздел “Функция CONTEXT SIGNON: синтаксис и использование” на стр. 829.

### **CREATE THREAD**

Выделяет план или пакет DB2. Необходимо выполнить функцию CREATE THREAD, прежде чем можно будет выполнять операторы SQL. Смотрите раздел “Функция CREATE THREAD: синтаксис и использование” на стр. 833.

### **TERMINATE THREAD**

Освобождает план. Смотрите раздел “Функция TERMINATE THREAD: синтаксис и использование” на стр. 835.

### **TERMINATE IDENTIFY**

Удаляет задание из числа пользователей DB2 и, если это последнее или единственное задание в данном адресном пространстве, имеющая соединения с DB2, завершает соединение этого адресного пространства с DB2. Смотрите раздел “Функция TERMINATE IDENTIFY: синтаксис и использование” на стр. 836.

### **TRANSLATE**

Возвращает код SQL и текстовое описание (в SQLCA), описывающие код причины ошибки DB2. Функцию TRANSLATE нельзя вызывать из программ на языке FORTRAN. Смотрите раздел “Функция TRANSLATE: синтаксис и использование” на стр. 837.

## **Описание функций RRSAF**

При использовании функций RRSAF в программах на языках C, COBOL, FORTRAN или PL/I следуйте правилам конкретного языка для реализации вызовов процедур на ассемблере. Задавайте в списке параметров для каждого вызова RRSAF параметры кода возврата и кода причины.

В этом разделе представлена следующая информация:

- “Соглашение об использовании регистров” на стр. 816
- “Соглашение о передаче параметров для вызовов функций” на стр. 816
- “Функция IDENTIFY: синтаксис и использование” на стр. 817
- “Функция SWITCH TO: синтаксис и использование” на стр. 819
- “Функция SIGNON: синтаксис и использование” на стр. 822
- “Функция AUTH SIGNON: Синтаксис и использование” на стр. 825
- “Функция CONTEXT SIGNON: синтаксис и использование” на стр. 829

- “Функция CREATE THREAD: синтаксис и использование” на стр. 833
- “Функция TERMINATE THREAD: синтаксис и использование” на стр. 835
- “Функция TERMINATE IDENTIFY: синтаксис и использование” на стр. 836
- “Функция TRANSLATE: синтаксис и использование” на стр. 837

## **Соглашение об использовании регистров**

На Табл. 86 показано соглашение об использовании регистров для вызовов RRSAF.

Если в вызове RRSAF не заданы параметры кода возврата и кода причины, RRSAF помещает код возврата в регистр 15, а код причины – в регистр 0. Если заданы параметры кода возврата и кода причины, RRSAF помещает код возврата в регистр 15 и в параметр кода возврата, позволяя программе на языке высокого уровня использовать это значения для обработки кода возврата. RRSAF сохраняет содержимое регистров со 2 по 14.

*Таблица 86. Соглашение об использовании регистров для вызовов RRSAF*

<b>Регистр</b>	<b>применение</b>
R1	Указатель на список параметров
R13	Адрес области сохранения вызывающей программы
R14	Адрес возврата вызывающей программы
R15	Адрес точки входа RRSAF

## **Соглашение о передаче параметров для вызовов функций**

**Для языка ассемблер:** Используйте стандартный список параметров для вызовов MVS CALL. Это означает, что при вызове в регистре 1 должен находится адрес списка указателей на параметры. Каждый указатель представляет собой 4–байтный адрес. Последний адрес должен содержать в старшем бите значение 1.

Если нужно, чтобы в вызове на ассемблере для какого–либо параметра использовалось значение по умолчанию, задайте для него запятую в операторе CALL DSNRLI и затем задайте последующие параметры. Например, чтобы задать все необязательные параметры, кроме *кода возврата*, запишите такой вызов IDENTIFY:

```
CALL DSNRLI, (IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB,,REASCODE)
```

**Для всех языков:** При написании оператора CALL DSNRLI в любом языке задавайте все параметры, идущие перед *кодом возврата*. Нельзя пропускать какие–либо из этих параметров, задавая нули или пробелы. Для этих параметров нет значений по умолчанию.

Все параметры, начиная с *кода возврата*, являются необязательными.

**Для всех языков, кроме ассемблера:** Задайте 0 для необязательного параметра в операторе CALL DSNRLI, если нужно, чтобы для этого параметра использовалось значение по умолчанию, а после него задаются последующие параметры. Например, предположим, вы пишите вызов IDENTIFY в программе на языке COBOL. Нужно задать все параметры, кроме *кода возврата*. Запишите этот вызов следующим образом:

```
CALL 'DSNRLI' USING IDFYFN SSNM RIBPTR EIBPTR TERMCB STARTECB  
BY CONTENT ZERO BY REFERENCE REASCODE.
```

## Функция IDENTIFY: синтаксис и использование

IDENTIFY создает соединение с DB2.

```
►—CALL DSNRLI—(—фун, имя_п/с, adr_RIB, adr_EIB, ECB_зав, ECB_зап—  
                  , код_взвр—  
                  , код_прч—  
—)—————►
```

Рисунок 195. Функция DSNRLI IDENTIFY

Параметры указывают на следующие области:

### фун

18–байтная область, содержащая слово IDENTIFY, после которого идут десять пробелов.

### имя\_п/с

4–байтное имя подсистемы DB2 или имя группового подключения (если используется группа совместного использования данных) с которым образовано соединение. Если имя\_п/с короче четырех символов, дополните его справа пробелами до четырех символов.

### адр\_RIB

4–байтная область, в которую RRSAF помещает после вызова адрес блока информации о выпуске (RIB). Он может использоваться для определения уровня выпуска подсистемы DB2, с которым соединена прикладная программа. Если RIB недоступен (например, если указанная подсистема не существует), RRSAF заполняет эту 4–байтную область нулями.

Область, на которую указывает параметр *адр\_RIB*, находится в области памяти ниже 16 МБайт.

Этот параметр является обязательным, хотя прикладная программа может не использовать возвращенную информацию.

### адр\_EIB

4–байтная область, в которую RRSAF помещает после вызова адрес блока информации о среде (EIB). EIB содержит информацию о среде, например, имя группы совместного использования данных и имя члена этой группы для DB2, для которых выдано требование IDENTIFY. Если подсистема DB2 не является группой совместного использования данных, RRSAF заполняет пробелами имена группы совместного использования данных и члена группы. Если EIB недоступен (например, если в параметре *имя\_п/с* задано имя несуществующей подсистемы), RRSAF заполняет эту 4–байтную область нулями.

Область, на которую указывает параметр *адр\_EIB*, находится в области памяти выше 16 МБайт.

Этот параметр является обязательным, хотя прикладная программа может не использовать возвращенную информацию.

### ECB\_зав

Адрес блока управления событиями (ECB) для этой прикладной программы, используемого при завершении DB2. DB2 передает этот ECB, если оператор системы вводит команду STOP DB2 или если DB2

завершается ненормально. Задайте значение 0, если не нужно использовать ECB завершения.

RRSAF помещает в ECB код POST, указывающий тип завершения, как показано в Табл. 87.

Таблица 87. Коды POST для разных типов завершения DB2

Код POST	Тип завершения
8	QUIESCE
12	FORCE
16	ABTERM

#### *ECB\_запуска*

Адрес ECB запуска для этой прикладной программы. Если в момент, когда прикладная программа выдает вызов IDENTIFY, система DB2 не была запущена, DB2 передает этот ECB при завершении запуска DB2. Задайте значение 0, если не нужно использовать ECB запуска. DB2 передает максимум один ECB запуска для адресного пространства. Передаваемый ECB, связан с самым последним вызовом IDENTIFY из этого адресного пространства. Прикладная программа должна проверить все ненулевые коды причины RRSAF или DB2, прежде чем выдавать команду WAIT для этого ECB.

Если *имя\_подсистемы* – это имя группового подключения, DB2 игнорирует ECB запуска.

#### *код\_возврата*

4–байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

#### *код\_причины*

4–байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр *код\_возврата*.

**Использование:** Функция IDENTIFY задает вызывающее задание в качестве пользователя службы DB2. Если ни одно из других заданий в этом адресном пространстве в настоящее время не соединено с подсистемой с именем *имя\_подсистемы*, функция IDENTIFY также вызывает инициализацию этого адресного пространства для связи с адресными пространствами DB2. Функция IDENTIFY устанавливает полномочия для перекрестных обращений к памяти от данного адресного пространства к DB2 и создает управляющие блоки адресных пространств.

При выполнении функции IDENTIFY DB2 определяет, имеет ли адресное пространство пользователя полномочия на соединение с DB2. DB2 запускает MVS SAF и передает SAF первичный ID авторизации. Этот ID авторизации – это 7–байтный ID пользователя для этого адресного пространства, если только функция авторизации не создала ACEE для этого адресного пространства. Если функция авторизации создала ACEE для этого адресного пространства,

DB2 передает 8-байтный ID пользователя из этого ACEE. SAF вызывает внешний продукт защиты (например, RACF), чтобы определить, имеет ли задание права на использование:

- Класса ресурсов DB2 (CLASS=DSNR)
- Подсистемы DB2 (SUBSYS=*имя\_подсистемы*)
- Типа соединения RRSAF

Если эта проверка была успешной, DB2 вызывает обработчик соединений DB2, выполняющий дополнительную проверку и, возможно, изменяющий ID авторизации. Затем DB2 задает имя соединения для RRSAF и тип соединения для RRSAF.

В Табл. 88 показаны вызовы функции IDENTIFY для каждого языка.

*Таблица 88. Примеры вызовов RRSAF IDENTIFY*

Язык	Пример вызова
Ассемблер	CALL DSNRLI,(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE)
C	fret=dsnrl(&idfyfn[0],&ssnm[0], &ribptr, &eibptr, &termecb, &startecb, &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING IDFYFN SSNM RIBPTR EIBPTR TERMECB STARTECB RETCODE REASCODE.
FORTRAN	CALL DSNRLI(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE)
PL/I	CALL DSNRLI(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

```

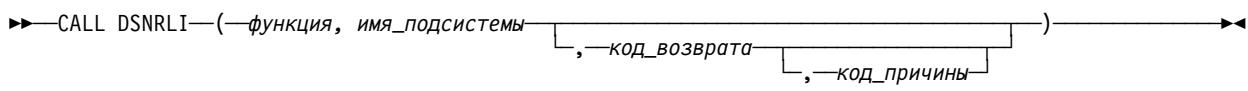
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
            int DSNALI(
                char * functn,
                ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);

```

### Функция SWITCH TO: синтаксис и использование

Функцию SWITCH TO можно использовать для направления требований RRSAF, SQL и/или IFI на указанную подсистему DB2.

Функцию SWITCH TO можно использовать только после успешного выполнения вызова IDENTIFY. Если имеется установленное соединение с одной подсистемой DB2, необходимо выдать вызов SWITCH TO перед вызовом IDENTIFY для соединения с другой подсистемой DB2.



*Рисунок 196. Функция DSNRLI SWITCH TO*

Параметры указывают на следующие области:

*функция*

18-байтная область, содержащая фразу SWITCH TO, после которой идут девять пробелов.

*имя\_подсистемы*

4-байтное имя подсистемы DB2 или имя группового подключения (если используется группа совместного использования данных) с которым образовано соединение. Если *имя\_подсистемы* короче четырех символов, дополните его справа пробелами до четырех символов.

*код\_возврата*

4-байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

*код\_причины*

4-байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр *код\_возврата*.

**Использование:** Используйте функцию SWITCH TO, чтобы установить соединения из одного задания с несколькими подсистемами DB2. Если выполняется вызов SWITCH TO для подсистемы DB2, для которой не был выполнен вызов IDENTIFY, DB2 возвращает код возврата 4 и код причины X'00C12205', предупреждая, что это задание еще не соединено ни с одной подсистемой DB2.

После установления соединения с подсистемой DB2, необходимо выполнить вызов SWITCH TO перед соединением с другой подсистемой DB2. Если перед выполнением вызова IDENTIFY для другой подсистемы DB2 не выполнен вызов SWITCH TO, DB2 возвращает код возврата = X'200' и код причины X'00C12201'.

В этом примере показано, как можно использовать функцию SWITCH TO для взаимодействия с тремя подсистемами DB2.

Вызовы RRSAF для подсистемы db21:  
 IDENTIFY  
 SIGNON  
 CREATE THREAD  
 Выполнение операторов SQL для подсистемы db21  
 SWITCH TO db22  
 Вызовы RRSAF для подсистемы db22:  
 IDENTIFY  
 SIGNON  
 CREATE THREAD  
 Выполнение операторов SQL для подсистемы db22  
 SWITCH TO db23  
 Вызовы RRSAF для подсистемы db23:  
 IDENTIFY  
 SIGNON  
 CREATE THREAD  
 Выполнение операторов SQL для подсистемы db23  
 SWITCH TO db21  
 Выполнение операторов SQL для подсистемы db21  
 SWITCH TO db22  
 Выполнение операторов SQL для подсистемы db22  
 SWITCH TO db21  
 Выполнение операторов SQL для подсистемы db21  
 SRRCMIT (чтобы выполнить принятие UR)  
 SWITCH TO db23  
 Выполнение операторов SQL для подсистемы db23  
 SWITCH TO db22  
 Выполнение операторов SQL для подсистемы db22  
 SWITCH TO db21  
 Выполнение операторов SQL для подсистемы db21  
 SRRCMIT (чтобы выполнить принятие UR)

В Табл. 89 показаны вызовы функции SWITCH TO для каждого языка.

*Таблица 89. Примеры вызовов RRSAF SWITCH TO*

<b>Язык</b>	<b>Пример вызова</b>
Ассемблер	CALL DSNRLI,(SWITCHFN,SSNM,RETCODE,REASCODE)
C	fnret=dsnrl(&switchfn[0], &ssnm[0], &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING SWITCHFN RETCODE REASCODE.
FORTRAN	CALL DSNRLI(SWITCHFN,RETCODE,REASCODE)
PL/I	CALL DSNRLI(SWITCHFN,RETCODE,REASCODE);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

```

C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
            int DSNALI(
                char * functn,
                ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
  
```

## Функция SIGNON: синтаксис и использование

Функция SIGNON задает для соединения первичный ID авторизации и также может задавать для него один или несколько вторичных ID авторизации.

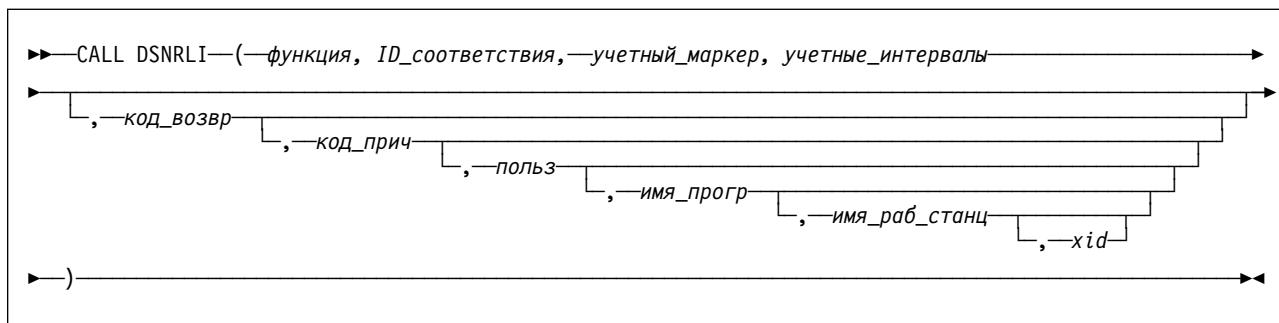


Рисунок 197. Функция DSNRLI SIGNON

Параметры указывают на следующие области:

### функция

18–байтная область, содержащая слово SIGNON, после которого идут двенадцать пробелов.

### ID\_соответствия

12–байтная область, в которую можно поместить ID соответствия DB2. ID соответствия выводится в трассировочных записях учета и статистики DB2. Этот ID можно использовать, чтобы установить соотношение между единицами работы. Он выводится в выходных данных команды DISPLAY THREAD. Если не требуется задавать ID соответствия, заполните эту 12–битную область пробелами.

### учетный\_маркер

22–байтная область, в которую можно поместить значение учетного маркера DB2. Это значение выводится в трассировочных записях учета и статистики DB2. Если не требуется задавать учетный маркер, заполните эту 22–битную область пробелами.

### учетный\_интервал

6–байтная область, позволяющая управлять моментами, когда DB2 записывает учетные записи. Если в этой области задано COMMIT, DB2 записывает учетные записи при каждом вызове прикладной программой функции SRRCMIT. Если задано какое–либо другое значение, DB2 записывает учетные записи при завершении прикладной программы или когда прикладная программа вызывает функцию SIGNON с новым ID авторизации.

### код\_возвр

4–байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

### код\_прич

4–байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр *код\_возврата*.

#### *польз*

16–байтная область, содержащая ID пользователя конечного пользователя клиента. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит этот ID пользователя в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *пользователь* короче 16 символов, необходимо дополнить его справа пробелами до 16 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возврата* и *код\_причины*. Если этот параметр не задан, ID пользователя для соединения считается не заданным. Можно пропустить этот параметр, задав значение 0.

#### *имя\_прогр*

32–байтная область, содержащая имя прикладной программы или транзакции для прикладной программы конечного пользователя. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит это имя прикладной программы в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *имя\_программы* короче 32 символов, необходимо дополнить его справа пробелами до 32 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возврата*, *код\_причины* и *пользователь*. Если этот параметр не задан, прикладная программа или транзакция для соединением считаются не заданными. Можно пропустить этот параметр, задав значение 0.

#### *имя\_раб\_станц*

18–байтная область, содержащая имя рабочей станции конечного пользователя клиента. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит это имя рабочей станции в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *имя\_рабочей\_станции* короче 18 символов, необходимо дополнить его справа пробелами до 18 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возвр*, *код\_прич*, *польз* и *имя\_прогр*. Если этот параметр не задан, никакого имени рабочей станции с соединением не связывается.

*xid* 4–байтная область, в которую надо записать одно из следующих значений:

- |   |   |
|---|---|
| 0 | Указывает, что поток не является частью глобальной транзакции.  |
| 1 | Указывает, что поток является частью глобальной транзакции. и что DB2 должна получить ID глобальной транзакции от RRS. Если ID глобальной транзакции для этого задания уже существует, поток становится частью соответствующей глобальной транзакции. В противном случае RRS генерирует новый ID глобальной транзакции. |

**адрес** 4-байтный адрес области, в которую вы заносите ID глобальной транзакции для потока. Если ID глобальной транзакции уже существует, поток становится частью соответствующей глобальной транзакции. В противном случае RRS создает новую глобальную транзакцию с указанным ID. Формат ID глобальной транзакции показан на Табл. 90.

*Таблица 90. Формат ID глобальной транзакции, задаваемого пользователем*

Описание поля	Длина в байтах	Тип данных
ID формата	4	Символьный
Длина ID глобальной транзакции	4	Целое
Длина спецификатора ветви	4	Целое
ID глобальной транзакции	от 1 до 64	Символьный
Спецификатор ветви	от 1 до 64	Символьный

Поток DB2, являющийся частью глобальной транзакции, и другие потоки DB2, которые являются частью той же глобальной транзакции, могут совместно использовать блокировки, обращаться к одним и тем же данным и модифицировать их. Глобальная транзакция существует, пока для одного из потоков, который является ее частью, не будет выполнено принятие или откат.

**Использование:** Функция SIGNON задает для соединения новый первичный ID авторизации и может также задавать для него вторичные ID авторизации. Чтобы выполнить вызов SIGNON, программа не обязательно должна быть авторизированной программой. Поэтому перед вызовом SIGNON нужно выполнить внешнюю макрокоманду интерфейса защиты RACROUTE REQUEST=VERIFY, чтобы:

- Определить и заполнить ACEE, определяющий пользователя программы.
- Связать этот ACEE с TCB пользователя.
- Проверить, определен ли этот пользователь для RACF и имеет ли он права на использование этой прикладной программы.

Дополнительную информацию о макрокоманде RACROUTE смотрите в руководстве *OS/390 Security Server (RACF) Macros and Interfaces*.

Обычно функция SIGNON вызывается после вызова функции IDENTIFY, но перед вызовом функции CREATE THREAD. Можно также вызвать функцию SIGNON, если прикладная программа находится в точке согласованности и

- Параметр *повторное\_использование* вызова CREATE THREAD имел значение RESET, или
- Параметр *повторное\_использование* вызова CREATE THREAD имел значение INITIAL, нет открытых сохраняемых указателей, пакет или план связан с опцией KEEPDYNAMIC(NO) и все специальные регистры содержат свои исходные значения. Если есть открытые сохраняемые указатели или пакет или план связаны с опцией KEEPDYNAMIC(YES), вызов SIGNON разрешается, только если первичный ID авторизации не изменен.

В Табл. 91 на стр. 825 показаны вызовы функции SIGNON для каждого языка.

Таблица 91. Примеры вызовов RRSAF SIGNON

<b>Язык</b>	<b>Пример вызова</b>
ассемблер	CALL DSNRLI,(SGNONFN,CORRID,ACCTTKN,ACCTINT, RETCODE,REASCODE,USERID,APPLNAME,WSNAME)
C	fnret=dsnrl(&sgnonfn[0], &corrid[0], &accttn[0], &acctint[0], &retcode, &reascode, &userid[0], &applname[0], &wsname[0]);
COBOL	CALL 'DSNRLI' USING SGNONFN CORRID ACCTTKN ACCTINT RETCODE REASCODE USERID APPLNAME WSNAME.
FORTRAN	CALL DSNRLI(SGNONFN,CORRID,ACCTTKN,ACCTINT, RETCODE,REASCODE,USERID,APPLNAME,WSNAME)
PL/I	CALL DSNRLI(SGNONFN,CORRID,ACCTTKN,ACCTINT, RETCODE,REASCODE,USERID,APPLNAME,WSNAME);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++     extern "OS" {
          int DSNALI(
              char * functn,
              ...); }

PL/I    DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

## Функция AUTH SIGNON: Синтаксис и использование

Функция AUTH SIGNON позволяет программе, авторизованной APF, передавать DB2:

- Первичный ID авторизации и, возможно, один или несколько вторичных ID авторизации.
  - ACEE, используемый для проверки авторизации

Функция AUTH SIGNON задает для соединения первичный ID авторизации и может также задавать для него один или несколько вторичных ID авторизации.

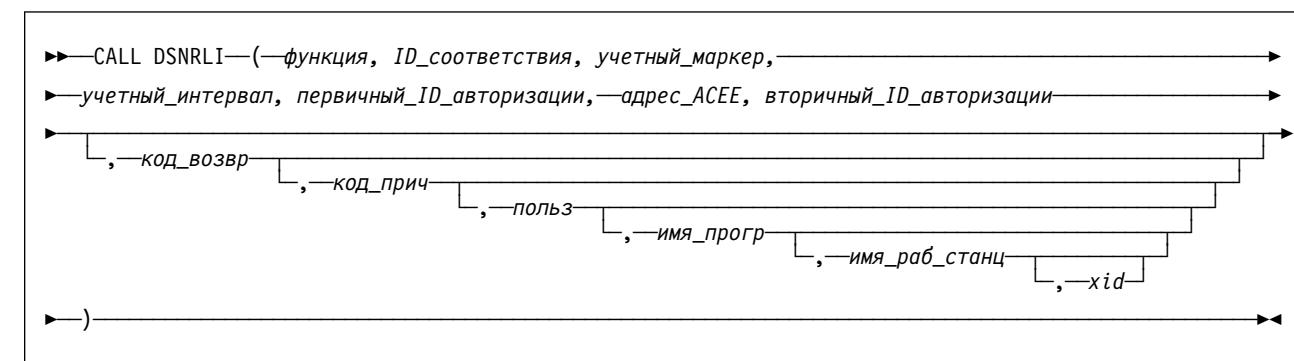


Рисунок 198. Функция DSNRLI AUTH SIGNON

Параметры указывают на следующие области:

## ФУНКЦИЯ

18-байтная область, содержащая фразу AUTH SIGNON, после которой идут семь пробелов.

#### *ID\_соответствия*

12–байтная область, в которую можно поместить ID соответствия DB2. ID соответствия выводится в трассировочных записях учета и статистики DB2. Этот ID можно использовать, чтобы установить соотношение между единицами работы. Он выводится в выходных данных команды DISPLAY THREAD. Если не требуется задавать ID соответствия, заполните эту 12–битную область пробелами.

#### *учетный\_маркер*

22–байтная область, в которую можно поместить значение учетного маркера DB2. Это значение выводится в трассировочных записях учета и статистики DB2. Если не требуется задавать учетный маркер, заполните эту 22–битную область пробелами.

#### *учетный\_интервал*

6–байтная область, позволяющая управлять моментами, когда DB2 записывает учетные записи. Если в этой области задано COMMIT, DB2 записывает учетные записи при каждом вызове прикладной программой функции SRRCMIT. Если задано какое–либо другое значение, DB2 записывает учетные записи при завершении прикладной программы или когда прикладная программа вызывает функцию SIGNON с новым ID авторизации.

#### *первичный\_ID\_авторизации*

8–байтная область, в которую можно поместить первичный ID авторизации. Если не нужно явно передавать DB2 первичный ID авторизации, поместите в первый байт этой области значение X'00' или пробел.

#### *адрес\_ACCE*

4–байтный адрес ACCE, передаваемого DB2. Если не нужно задавать ACCE, задайте в этом поле значение 0.

#### *вторичный\_ID\_авторизации*

8–байтная область, в которую можно поместить вторичный ID авторизации. Если не нужно явно передавать DB2 первичный ID авторизации, поместите в первый байт этой области значение X'00' или пробел. Если задается вторичный ID авторизации, необходимо также задать и первичный ID авторизации.

#### *код\_возвр*

4–байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

#### *код\_прич*

4–байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр *код\_возврата*.

#### *польз*

16–байтная область, содержащая ID пользователя конечного пользователя клиента. Этот параметр можно использовать, чтобы задать описание

конечного пользователя клиента для целей учета и контроля. DB2 выводит этот ID пользователя в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *пользователь* короче 16 символов, необходимо дополнить его справа пробелами до 16 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возврата* и *код\_причины*. Если этот параметр не задан, ID пользователя для соединения считается не заданным. Можно пропустить этот параметр, задав значение 0.

*имя\_прогр*

32-байтная область, содержащая имя прикладной программы или транзакции для прикладной программы конечного пользователя. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит это имя прикладной программы в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *имя\_программы* короче 32 символов, необходимо дополнить его справа пробелами до 32 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возврата*, *код\_причины* и *пользователь*. Если этот параметр не задан, прикладная программа или транзакция для соединением считаются не заданными. Можно пропустить этот параметр, задав значение 0.

*имя\_раб\_станц*

18-байтная область, содержащая имя рабочей станции конечного пользователя клиента. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит это имя рабочей станции в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *имя\_рабочей\_станции* короче 18 символов, необходимо дополнить его справа пробелами до 18 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возврата*, *код\_причины*, *пользователь* и *имя\_программы*. Если этот параметр не задан, никакого имени рабочей станции с соединением не связывается.

*xid* 4-байтная область, в которую надо записать одно из следующих значений:

- |              |   |
|--------------|---|
| <b>0</b>     | Указывает, что поток не является частью глобальной транзакции.  |
| <b>1</b>     | Указывает, что поток является частью глобальной транзакции, и что DB2 должна получить ID глобальной транзакции от RRS. Если ID глобальной транзакции для этого задания уже существует, поток становится частью соответствующей глобальной транзакции. В противном случае RRS генерирует новый ID глобальной транзакции. |
| <b>адрес</b> | 4-байтный адрес области, в которую вы заносите ID глобальной транзакции для потока. Если ID глобальной транзакции уже существует, поток становится частью соответствующей глобальной транзакции. В противном случае RRS создает новую глобальную транзакцию с   |

указанным ID. Формат ID глобальной транзакции показан на Табл. 90 на стр. 824.

Поток DB2, являющийся частью глобальной транзакции, и другие потоки DB2, которые являются частью той же глобальной транзакции, могут совместно использовать блокировки, обращаться к одним и тем же данным и модифицировать их. Глобальная транзакция существует, пока для одного из потоков, который является ее частью, не будет выполнено принятие или откат.

**Использование:** Функция AUTH SIGNON задает для соединения новый первичный ID авторизации и может также задавать для него вторичные ID авторизации.

Обычно функция AUTH SIGNON вызывается после вызова функции IDENTIFY, но перед вызовом функции CREATE THREAD. Также можно вызвать функцию AUTH SIGNON, если прикладная программа находится в точке согласованности и

- Параметр *повторное\_использование* вызова CREATE THREAD имел значение RESET, или
- Параметр *повторное\_использование* вызова CREATE THREAD имел значение INITIAL, нет открытых сохраняемых указателей, пакет или план связан с опцией KEEPDYNAMIC(NO) и все специальные регистры содержат свои исходные значения. Если есть открытые сохраняемые указатели или пакет или план связаны с опцией KEEPDYNAMIC(YES), вызов SIGNON разрешается, только если первичный ID авторизации не изменен.

В Табл. 92 показаны вызовы функции AUTH SIGNON для каждого языка.

Таблица 92. Примеры вызовов RRSAF AUTH SIGNON

Язык	Пример вызова
Ассемблер	CALL DSNRLI,(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPRTR, SAUTHID,RETCODE,REASCODE,USERID,APPLNAME,WSNAME)
C	fnret=dsnrl(&asgnonfn[0], &corrid[0], &accttkn[0], &acctint[0], &pauthid[0], &aceeprtr, &sauthid[0], &retcode, &reascode, &userid[0], &applname[0], &wsname[0]);
COBOL	CALL 'DSNRLI' USING ASGNONFN CORRID ACCTTKN ACCTINT PAUTHID ACEEPRTR SAUTHID RETCODE REASCODE USERID APPLNAME WSNAME.
FORTRAN	CALL DSNRLI(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPRTR, SAUTHID,RETCODE,REASCODE,USERID,APPLNAME,WSNAME)
PL/I	CALL DSNRLI(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPRTR, SAUTHID,RETCODE,REASCODE,USERID,APPLNAME,WSNAME);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
          int DSNALI(
            char * functn,
            ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

## Функция CONTEXT SIGNON: синтаксис и использование

Функция CONTEXT SIGNON задает для соединения первичный ID авторизации и один или несколько вторичных ID авторизации.

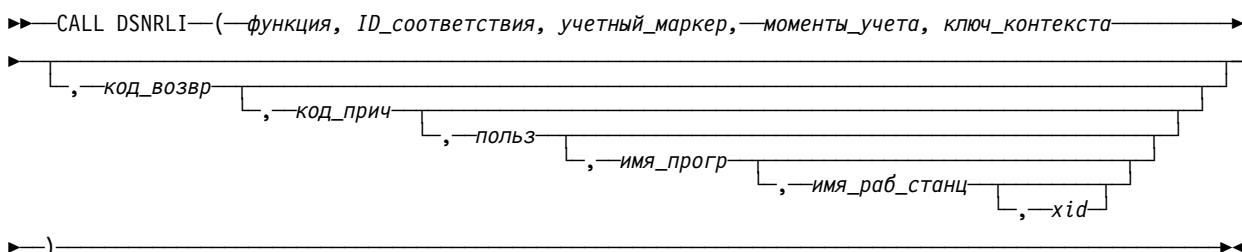


Рисунок 199. Функция DSNRLI CONTEXT SIGNON

Параметры указывают на следующие области:

### функция

18–байтная область, содержащая фразу CONTEXT SIGNON, после которой идут четыре пробела.

### ID\_соответствия

12–байтная область, в которую можно поместить ID соответствия DB2. ID соответствия выводится в трассировочных записях учета и статистики DB2. Этот ID можно использовать, чтобы установить соотношение между единицами работы. Он выводится в выходных данных команды DISPLAY THREAD. Если не требуется задавать ID соответствия, заполните эту 12–битную область пробелами.

### учетный\_маркер

22–байтная область, в которую можно поместить значение учетного маркера DB2. Это значение выводится в трассировочных записях учета и статистики DB2. Если не требуется задавать учетный маркер, заполните эту 22–битную область пробелами.

### учетный\_интервал

6–байтная область, позволяющая управлять моментами, когда DB2 записывает учетные записи. Если в этой области задано COMMIT, DB2 записывает учетные записи при каждом вызове прикладной программой функции SRRCMIT. Если задано какое–либо другое значение, DB2 записывает учетные записи при завершении прикладной программы или когда прикладная программа вызывает функцию SIGNON с новым ID авторизации.

### ключ\_контекста

32–байтная область, в которую помещается ключ контекста, заданный при вызове службы задания данных контекста RRS (CTXSDTA) для сохранения первичного ID авторизации и необязательного адреса ACEE.

### код\_возвр

4–байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

*код\_прич*

4–байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр *код\_возвр*.

*польз*

16–байтная область, содержащая ID пользователя конечного пользователя клиента. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит этот ID пользователя в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *польз* короче 16 символов, необходимо дополнить его справа пробелами до 16 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возвр* и *код\_прич*. Если этот параметр не задан, ID пользователя для соединения считается не заданным. Можно пропустить этот параметр, задав значение 0.

*имя\_прогр*

32–байтная область, содержащая имя прикладной программы или транзакции для прикладной программы конечного пользователя. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит это имя прикладной программы в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *имя\_прогр* короче 32 символов, необходимо дополнить его справа пробелами до 32 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возвр*, *код\_прич* и *польз*. Если этот параметр не задан, прикладная программа или транзакция для соединением считаются не заданными. Можно пропустить этот параметр, задав значение 0.

*имя\_раб\_станц*

18–байтная область, содержащая имя рабочей станции конечного пользователя клиента. Этот параметр можно использовать, чтобы задать описание конечного пользователя клиента для целей учета и контроля. DB2 выводит это имя рабочей станции в выходных данных команды DISPLAY THREAD и в трассировочных записях учета и статистики DB2. Если значение параметра *имя\_раб\_станц* короче 18 символов, необходимо дополнить его справа пробелами до 18 символов.

Это необязательный параметр. Если этот параметр задан, необходимо также задать параметры *код\_возвр*, *код\_прич*, *польз* и *имя\_прогр*. Если этот параметр не задан, никакого имени рабочей станции с соединением не связывается.

*xid* 4–байтная область, в которую надо записать одно из следующих значений:

- |          |  |
|----------|--|
| <b>0</b> | Указывает, что поток не является частью глобальной транзакции. |
|----------|--|

1	Указывает, что поток является частью глобальной транзакции, и что DB2 должна получить ID глобальной транзакции от RRS. Если ID глобальной транзакции для этого задания уже существует, поток становится частью соответствующей глобальной транзакции. В противном случае RRS генерирует новый ID глобальной транзакции.
адрес	4-байтный адрес области, в которую вы заносите ID глобальной транзакции для потока. Если ID глобальной транзакции уже существует, поток становится частью соответствующей глобальной транзакции. В противном случае RRS создает новую глобальную транзакцию с указанным ID. Формат ID глобальной транзакции показан на Табл. 90 на стр. 824.

Поток DB2, являющийся частью глобальной транзакции, и другие потоки DB2, которые являются частью той же глобальной транзакции, могут совместно использовать блокировки, обращаться к одним и тем же данным и модифицировать их. Глобальная транзакция существует, пока для одного из потоков, который является ее частью, не будет выполнено принятие или откат.

**Использование:** Функция CONTEXT SIGNON использует функции служб контекста RRS для задания данных контекста (CTXSDTA) и получения данных контекста (CTXRDTA). Перед вызовом CONTEXT SIGNON необходимо вызвать функцию CTXSDTA, чтобы сохранить первичный ID авторизации и, возможно, адрес ACEE в данных контекста, ключ контекста для которого задается во входном параметре функции CONTEXT SIGNON.

Функция CONTEXT SIGNON задает для соединения новый первичный ID авторизации и может также задавать для него один или несколько вторичных ID авторизации. Функция CONTEXT SIGNON использует ключ контекста для получения первичного ID авторизации из данных для текущего контекста RRS. DB2 использует функцию CTXRDTA служб контекста RRS, чтобы получить данные контекста, содержащие ID авторизации и адрес ACEE. Данные контекста должны иметь следующий формат:

*Номер версии* 4-байтная область, содержащая номер версии данных контекста. Задайте для этой области значение 1.

*Имя продукта сервера* 8-байтная область, содержащая имя продукта сервера, задавшего эти данные контекста.

*ALET* 4-байтная область, содержащая значение ALET. DB2 не использует эту область.

*Адрес ACEE* 4-байтная область, содержащая адрес ACEE или 0, если ACEE не задан. DB2 требует, чтобы этот ACEE находился в начальном адресном пространстве этого задания.

*первичный\_ID\_авторизации* 8-байтная область, содержащая используемый первичный ID авторизации. Если ID авторизации короче 8 символов, необходимо дополнить его справа пробелами до 8 символов.

Если новый первичный ID авторизации не отличается от текущего первичного ID авторизации (заданного функцией IDENTIFY или в предыдущем вызове SIGNON), DB2 вызывает только обработчик регистрации. Если это значение

изменено, DB2 устанавливает новый первичный ID авторизации и новый ID авторизации SQL и затем вызывает обработчик регистрации.

Если передается адрес ACEE, функция CONTEXT SIGNON использует значение поля ACEEGRPN в качестве вторичного ID авторизации, если длина имени группы (ACEEGRPL) не равна 0.

Обычно функция CONTEXT SIGNON вызывается после вызова функции IDENTIFY, но перед вызовом функции CREATE THREAD. Функцию CONTEXT SIGNON можно вызвать также, если прикладная программа находится в точке согласованности и

- Параметр *повторное\_использование* вызова CREATE THREAD имел значение RESET, или
- Параметр *повторное\_использование* вызова CREATE THREAD имел значение INITIAL, нет открытых сохраняемых указателей, пакет или план связан с опцией KEEPDYNAMIC(NO) и все специальные регистры содержат свои исходные значения. Если есть открытые сохраняемые указатели или пакет или план связаны с опцией KEEPDYNAMIC(YES), вызов SIGNON разрешается, только если первичный ID авторизации не изменен.

В Табл. 93 показаны вызовы функции CONTEXT SIGNON для каждого языка.

Таблица 93. Примеры вызовов RRSAF CONTEXT SIGNON

Язык	Пример вызова
Ассемблер	CALL DSNRLI(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXKEY, RETCODE,REASCODE,USERID,APPLNAME,WSNAME)
C	fnret=dsnrl(&csgnonfn[0], &corrid[0], &acctkn[0], &acctint[0], &ctxkey[0], &retcode, &reascode, &userid[0], &applname[0], &wsname[0]);
COBOL	CALL 'DSNRLI' USING CSGNONFN CORRID ACCTTKN ACCTINT CTXKEY RETCODE REASCODE USERID APPLNAME WSNAME.
FORTRAN	CALL DSNRLI(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXKEY, RETCODE,REASCODE, USERID,APPLNAME,WSNAME)
PL/I	CALL DSNRLI(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXKEY, RETCODE,REASCODE,USERID,APPLNAME,WSNAME);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

**C**       #pragma linkage(dsnali, OS)  
**C++**      extern "OS" {  
                int DSNALI(  
                    char \* functn,  
                    ...); }  
**PL/I**      DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);

## Функция CREATE THREAD: синтаксис и использование

Функция CREATE THREAD выделяет для прикладной программы ресурсы DB2.

```
►—CALL DSNRLI—(—функция, план, собрание, повторное_использование—
  —, код_возврата —, код_причины—)
```

Рисунок 200. Функция DSNRLI CREATE THREAD

Параметры указывают на следующие области:

### функция

18–байтная область, содержащая фразу CREATE THREAD, после которой идут пять пробелов.

### план

8–байтное имя плана DB2. Если вместо имени плана задается имя собрания, задайте символ ? в первом байте этого поля. В этом случае DB2 выделяет специальный план с именем ?RRSAF и использует параметр собрание. Если в поле собрание не задано имя собрания, необходимо задать в этом поле правильное имя плана.

### собрание

18–байтная область, в которой вводится имя собрания. Когда вы задаете имя собрания и ставите символ ? в поле план, DB2 выделяет план с именем ?RRSAF и список пакетов из двух элементов:

- Имя этого собрания
- Запись, которая содержит \* для положения, имени собрания и имени пакета.

Если в поле план задано имя плана, DB2 игнорирует значение этого поля.

### повторное\_использование

8–байтная область, управляющая поведением DB2 в случаях, когда вызов SIGNON производится после вызова CREATE THREAD. Задайте в этом поле одно из следующих значений:

- RESET – чтобы освободить все заблокированные указатели и реинициализировать специальные регистры
- INITIAL – чтобы запретить выполнение функции SIGNON

Это обязательный параметр. Если эта 8–байтная область не содержит значения RESET или INITIAL, используется значение по умолчанию INITIAL.

### код\_возврата

4–байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

код\_причины

4-байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр код\_возврата.

**Использование:** Функция CREATE THREAD выделяет ресурсы DB2, необходимые для выполнения требований SQL или IFI. Если задано имя плана, RRSAF выделяет указанный план. Если в первом байте имени плана задан символ ? и задано имя собрания, DB2 выделяет специальный план с именем ?RRSAF и список пакетов, содержащий следующие записи:

- Имя собрания
- Запись, которая содержит \* для положения, имени собрания и имени пакета.

Это имя собрания используется для поиска пакета, связанного с первым оператором SQL программы. Запись, содержащая \*.\*.\*., позволяет программе обращаться к удаленным узлам и к пакетам в собраниях, отличающихся от собрания по умолчанию, которое было задано в момент создания потока.

Прикладная программа может использовать оператор SQL SET CURRENT PACKAGESET, чтобы изменить ID собрания, используемый системой DB2 для поиска пакета.

Выделив план с именем ?RRSAF, DB2 проверяет полномочия на выполнение пакета так же, как и при проверке полномочий на выполнение пакета для реквестера, отличающегося от DB2 for OS/390. Дополнительную информацию о проверке авторизации на выполнение пакетовсмотрите в разделе Раздел 3 (Том 1) *DB2 Administration Guide*.

В Табл. 94 показаны вызовы функции CREATE THREAD для каждого языка.

Таблица 94. Примеры вызовов RRSAF CREATE THREAD

Язык	Пример вызова
Ассемблер	CALL DSNRLI,(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE)
C	fnret=dsnrl(&crthrdn[0], &plan[0], &collid[0], &reuse[0], &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING CRTHRDFN PLAN COLLID REUSE RETCODE REASCODE.
FORTRAN	CALL DSNRLI(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE)
PL/I	CALL DSNRLI(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

**C** #pragma linkage(dsnali, OS)

**C++** extern "OS" {  
 int DSNALI(  
 char \* functn,  
 ...); }

**PL/I** DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);

## **Функция TERMINATE THREAD: синтаксис и использование**

Функция TERMINATE THREAD освобождает ресурсы DB2, которые были ранее выделены для прикладной программы функцией CREATE THREAD.

```
►—CALL DSNRLI—(—функция,—  
                  |,—код_возврата—  
                  |,—код_причины—  
                  )————►
```

Рисунок 201. Функция DSNRLI TERMINATE THREAD

Параметры указывают на следующие области:

### **функция**

18-байтная область, содержащая фразу TERMINATE THREAD, после которой идут два пробела.

### **код\_возврата**

4-байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

### **код\_причины**

4-байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр **код\_возврата**.

**Использование:** Функция TERMINATE THREAD освобождает связанные с планом ресурсы DB2. Эти ресурсы были ранее выделены функцией CREATE THREAD. Затем можно использовать функцию CREATE THREAD для выделения другого плана с использованием того же соединения.

Если вызывается функция TERMINATE THREAD, но прикладная программа не находится в точке согласованности, RRSAF возвращает код причины X'00C12211'.

В Табл. 95 на стр. 836 показаны вызовы функции TERMINATE THREAD для каждого языка.

Таблица 95. Примеры вызовов RRSAF TERMINATE THREAD

Язык	Пример вызова
Ассемблер	CALL DSNRLI,(TRMTHDFN,RETCODE,REASCODE)
C	fnret=dsnrl(&trmthdfn[0], &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING TRMTHDFN RETCODE REASCODE.
FORTRAN	CALL DSNRLI(TRMTHDFN,RETCODE,REASCODE)
PL/I	CALL DSNRLI(TRMTHDFN,RETCODE,REASCODE);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

```

C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
            int DSNALI(
                char * functn,
                ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);

```

## Функция TERMINATE IDENTIFY: синтаксис и использование

Функция TERMINATE IDENTIFY завершает соединение с DB2.

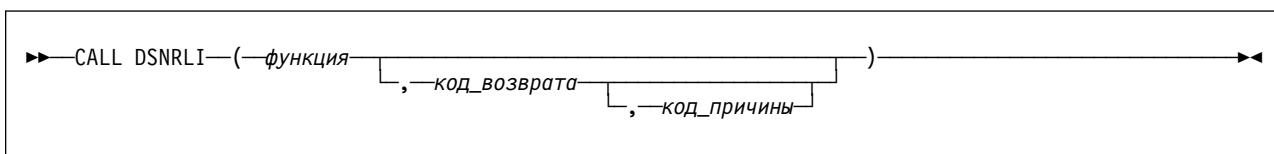


Рисунок 202. Функция DSNRLI TERMINATE IDENTIFY

Параметры указывают на следующие области:

### функция

18-байтная область, содержащая TERMINATE IDENTIFY.

### код\_возврата

4-байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

### код\_причины

4-байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр код\_возврата.

**Использование:** Функция TERMINATE IDENTIFY завершает соединение вызывающей программы с DB2. Если ни одно из других заданий в этом адресном пространстве не имеет активных соединений с DB2, DB2 также удаляет структуры управляющих блоков, созданные для этого адресного пространства, и отменяет полномочия для перекрестных обращений к памяти.

Если при вызове функции TERMINATE IDENTIFY прикладная программа не находится в точке согласованности, RRSAF возвращает код причины X'00C12211'.

Если прикладная программа выделила план и перед вызовом функции TERMINATE IDENTIFY не была вызвана функция TERMINATE THREAD, DB2 освобождает этот план перед завершением соединения.

Вызов функции TERMINATE IDENTIFY является необязательным. Если она не вызвана, DB2 выполняет те же функции при завершении задания.

Если завершается работа DB2, прикладная программа должна вызвать функцию TERMINATE IDENTIFY, чтобы сбросить управляющие блоки RRSAF. Это позволяет успешно выполнять последующие требования соединения от этого задания при перезапуске DB2.

В Табл. 96 показаны вызовы функции TERMINATE IDENTIFY для каждого языка.

Таблица 96. Примеры вызовов RRSAF TERMINATE IDENTIFY

Язык	Пример вызова
Ассемблер	CALL DSNRLI,(TMIDFYFN,RETCODE,REASCODE)
C	fnret=dsnrl(&tmidfyfn[0], &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING TMIDFYFN RETCODE REASCODE.
FORTRAN	CALL DSNRLI(TMIDFYFN,RETCODE,REASCODE)
PL/I	CALL DSNRLI(TMIDFYFN,RETCODE,REASCODE);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
        int DSNALI(
            char * functn,
            ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

### Функция TRANSLATE: синтаксис и использование

Функция TRANSLATE преобразует шестнадцатеричный код причины для ошибки DB2 в целое число со знаком SQLCODE и текстовое сообщение об ошибке. SQLCODE и текст сообщения помещаются в SQLCA вызывающей программы. Функцию TRANSLATE нельзя вызывать из программ на языке FORTRAN.

Вызывайте функцию TRANSLATE только после успешного выполнения операции IDENTIFY. Для ошибок, возникающих при выполнении требований SQL или IFI, функция TRANSLATE выполняется автоматически.

```

►►CALL DSNRLI—(—функция, sqlca
    [—код_возврата
     , —код_причины
    ]—)
    ►►

```

Рисунок 203. Функция DSNRLI TRANSLATE

Параметры указывают на следующие области:

**функция**

18-байтная область, содержащая слово TRANSLATE, после которой идут девять пробелов.

**sqlca**

Область связи SQL (SQLCA) программы.

**код\_возврата**

4-байтная область, в которую RRSAF помещает код возврата.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код возврата в регистр 15 и код причины в регистр 0.

**код\_причины**

4-байтная область, в которую RRSAF помещает код причины.

Это необязательный параметр. Если этот параметр не задан, RRSAF помещает код причины в регистр 0.

Если этот параметр задан, необходимо также задать параметр

**код\_возврата.**

**Использование:** Используйте функцию TRANSLATE, чтобы получить код ошибки SQL и текст сообщения, соответствующие коду причины ошибки DB2, который RRSAF возвращает в регистре 0 (после выполнения вызова CREATE THREAD). DB2 помещает эту информацию в переменные хоста SQLCODE и SQLSTATE или соответствующие поля SQLCA.

Функция TRANSLATE преобразует коды причины RRSAF, начинающиеся с X'00F3', но не преобразует коды, начинающиеся с X'00C1'. Если после выполнения требования OPEN получен код причины ошибки X'00F30040' (*ресурс недоступен*), функция TRANSLATE вернет имя недоступного объекта базы данных в последних 44 символах поля SQLERRM. Если функция DB2 TRANSLATE не распознала код причины ошибки, она возвращает SQLCODE -924 (SQLSTATE '58006') и помещает в поле SQLERRM копию текста исходной функции DB2 и коды возврата и причины ошибки. Содержание регистров 0 и 15 не меняется, за исключением случаев ошибок самой функции TRANSLATE, когда регистр 0 имеет значение X'00C12204', а регистр 15 – значение 200.

В Табл. 97 на стр. 839 показаны вызовы функции TRANSLATE для каждого языка.

Таблица 97. Примеры вызовов RRSAF TRANSLATE

Язык	Пример вызова
Ассемблер	CALL DSNRLI,(XLATFN,SQLCA,RETCODE,REASCODE)
C	fnret=dsnrl(&connfn[0], &sqlca, &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING XLATFN SQLCA RETCODE REASCODE.
PL/I	CALL DSNRLI(XLATFN,SQLCA,RETCODE,REASCODE);

**Примечание:** DSNRLI – это программа на ассемблере, поэтому необходимо включить в прикладные программы на языках C, C++ и PL/I следующие директивы компилятора:

```
C      #pragma linkage(dsnali, OS)
C++    extern "OS" {
        int DSNALI(
            char * functrn,
            ...); }
PL/I   DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);
```

## Сводка поведения RRSAF

В Табл. 98 на стр. 840 и Табл. 99 на стр. 841 приводится сводка поведения RRSAF после получения от прикладной программы различных требований. Возникающие ошибки описываются кодами причины DB2, возвращаемыми RRSAF. Список кодов причин смотрите в описании кодов причин X'C1.' в разделе 4 руководства *DB2 Сообщения и коды*. Используйте эти таблицы, чтобы уяснить порядок, в котором должны идти в прикладной программе вызовы функций RRSAF, операторы SQL и требования IFI.

В этих таблицах в первом столбце идут последние уже выполненные функции RRSAF или DB2. В первой строке перечислены следующие выполняющиеся функции. На пересечении строки и столбца описывается результат вызова функции из первой строки, вызываемой после функции из первого столбца. Например, если вызвана функция TERMINATE THREAD, а затем выполняется оператор SQL или вызов IFI, RRSAF вернет код причины X'00C12219'.

*Таблица 98. Результат последовательности вызовов, если следующий вызов – это вызов функции IDENTIFY, SIGNON, CREATE THREAD, SQL или IFI*

Следующий вызов ==> Предыдущая функция	IDENTIFY	SWITCH TO	SIGNON, AUTH SIGNON THREAD или CONTEXT SIGNON	CREATE THREAD	SQL или IFI
Отсутствует: это первый вызов	IDENTIFY	X'00C12205'	X'00C12204'	X'00C12204'	X'00C12204'
	IDENTIFY	X'00C12201'	Переключение на <i>имя_подсистемы</i>	Регистрация 1	X'00C12217'
	SWITCH TO	IDENTIFY	Переключение на <i>имя_подсистемы</i>	Регистрация 1	CREATE THREAD
	SIGNON, AUTH SIGNON или CONTEXT SIGNON	X'00C12201'	Переключение на <i>имя_подсистемы</i>	Регистрация 1	CREATE THREAD
	CREATE THREAD	X'00C12201'	Переключение на <i>имя_подсистемы</i>	Регистрация 1	X'00C12202'
	TERMINATE THREAD	X'00C12201'	Переключение на <i>имя_подсистемы</i>	Регистрация 1	CREATE THREAD
	IFI	X'00C12201'	Переключение на <i>имя_подсистемы</i>	Регистрация 1	X'00C12202'
	SQL	X'00C12201'	Переключение на <i>имя_подсистемы</i>	X'00F30092' 2	X'00C12202'
	SRRCMIT или SRRBACK	X'00C12201'	Переключение на <i>имя_подсистемы</i>	Регистрация 1	X'00C12202'

**Примечания:**

1. Регистрация означает регистрацию в DB2 при помощи одной из функций SIGNON, AUTH SIGNON или CONTEXT SIGNON.
2. Функции SIGNON, AUTH SIGNON или CONTEXT SIGNON нельзя вызывать, если после вызова функции CREATE THREAD или после последнего вызова функции SRRCMIT или SRRBACK выполнялись какие-либо операторы SQL.

**Таблица 99. Результат последовательности вызовов, если следующий вызов — это вызов функции TERMINATE THREAD, TERMINATE IDENTIFY или TRANSLATE**

Следующий вызов ==>	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
Предыдущая функция			
Отсутствует: это первый вызов	X'00C12204'	X'00C12204'	X'00C12204'
IDENTIFY	X'00C12203'	TERMINATE IDENTIFY	TRANSLATE
SWITCH TO	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
SIGNON, AUTH SIGNON или CONTEXT SIGNON	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
CREATE THREAD	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
TERMINATE THREAD	X'00C12203'	TERMINATE IDENTIFY	TRANSLATE
IFI	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE
SQL	X'00F30093' <sup>1</sup>	X'00F30093' <sup>2</sup>	TRANSLATE
SRRCMIT или SRRBACK	TERMINATE THREAD	TERMINATE IDENTIFY	TRANSLATE

**Примечания:**

- Функцию TERMINATE THREAD нельзя вызывать, если после вызова функции CREATE THREAD или после последнего вызова функции SRRCMIT или SRRBACK выполнялись какие-либо операторы SQL.
- Функцию TERMINATE IDENTIFY нельзя вызывать, если после вызова функции CREATE THREAD или после последнего вызова функции SRRCMIT или SRRBACK выполнялись какие-либо операторы SQL.

---

## Примеры сценариев

В этом разделе представлены примеры сценариев для соединения заданий с DB2.

### Отдельное задание

В этом примере показано отдельное задание, выполняющееся в некотором адресном пространстве. OS/390 RRS управляет операцией принятия, если это задание завершается нормально.

```
IDENTIFY
SIGNON
CREATE THREAD
SQL или IFI
:
TERMINATE IDENTIFY
```

### Несколько заданий

В этом примере показаны несколько заданий, выполняющихся в некотором адресном пространстве. Задание 1 не выполняет операторы SQL и не делает вызовы IFI. Ее назначение — ожидать ECB завершения и запуска DB2, а также проверять уровня выпуска DB2.

TASK 1	TASK 2	TASK 3	TASK n
IDENTIFY	IDENTIFY	IDENTIFY	IDENTIFY
SIGNON	SIGNON	SIGNON	SIGNON
CREATE THREAD	CREATE THREAD	CREATE THREAD	CREATE THREAD
SQL	SQL	SQL	SQL
...	...	...	...
SRRCMIT	SRRCMIT	SRRCMIT	SRRCMIT
SQL	SQL	SQL	SQL
...	...	...	...
SRRCMIT	SRRCMIT	SRRCMIT	SRRCMIT
...	...	...	...
TERMINATE IDENTIFY			

## Вызов функции SIGNON для повторного использования потока DB2

В этом примере показано повторное использование потока DB2 другим пользователем в точке согласованности. Эта прикладная программа вызывает функцию SIGNON для пользователя В при помощи плана DB2, выделенного функцией CREATE THREAD, которая вызвана для пользователя А.

```

IDENTIFY
SIGNON для пользователя А
CREATE THREAD
SQL
...
SRRCMIT
SIGNON для пользователя В
SQL
...
SRRCMIT

```

## Переключение потоков DB2 между задачами

В этом примере показано, как можно переключать потоки для четырех пользователей (А, В, С и D) между двумя заданиями (1 и 2). Прикладные программы выполняют следующие шаги:

- Задание 1 создает контекст а, выполняет переключение контекста, чтобы сделать активным контекст а для задания 1, затем соединяется с подсистемой. Задание всегда должно выполнить операцию соединения, прежде чем можно будет произвести переключение контекста. После выполнения операции соединения задание 1 выделяет поток для пользователя А и выполняет операции SQL.

В то же время задание 2 создает контекст b, выполняет переключение контекста, чтобы сделать активным контекст b для задания 2, соединяется с подсистемой и затем выделяет поток для пользователя В и также выполняет операции SQL.

Когда операции SQL выполнены, оба задания выполняют операции переключения контекста OS/390 RRS. Эти операции отсоединяют каждый поток DB2 от задания, для которого он выполнялся.

- Затем задание 1 создает контекст с, соединяется с подсистемой, выполняет переключение контекста, чтобы сделать активным контекст с для задания 1, затем выделяет поток для пользователя С и выполняет операции SQL для пользователя С.

Задание 2 выполняет те же действия для пользователя D.

Когда завершены операции SQL для пользователя C, задание 1 выполняет следующие операции переключения контекста:

- Отключает от задания 1 поток для пользователя C.
- Подключает к заданию 1 поток для пользователя B.

Для выполнения операции переключения контекста, связывающей задание с потоком DB2, требуется, чтобы перед этим была выполнена операция соединения с этим потоком DB2. Следовательно, прежде чем можно будет связать поток для пользователя B с заданием 1, задание 1 должно выполнить операцию соединения.

- Задание 2 выполняет две операции переключения контекста:
  - Отключает от задания 2 поток для пользователя D.
  - Подключает к заданию 2 поток для пользователя A.

Задание 1

```
CTXBEGC (создать контекст a)
CTXSWCH(a,0)
IDENTIFY
SIGNON для пользователя А
CREATE THREAD (план А)
SQL
...
CTXSWCH(0,a)
```

Задание 2

```
CTXBEGC (создать контекст b)
CTXSWCH(b,0)
IDENTIFY
SIGNON для пользователя В
CREATE THREAD (план В)
SQL
...
CTXSWCH(0,b)

CTXBEGC (создать контекст c)
CTXSWCH(c,0)
IDENTIFY
SIGNON для пользователя С
CREATE THREAD (план С)
SQL
...
CTXSWCH(b,c)
SQL (план В)
...
CTXSWCH(0,d)
...
CTXSWCH(a,0)
SQL (план А)
```

## Коды возврата и коды причин RRSAF

Если в вызове RRSAF заданы параметры кода возврата и кода причины, RRSAF помещает код возврата и код причины в эти параметры. В противном случае RRSAF помещает код возврата в регистр 15, а код причины — в регистр 0. Подробное описание кодов причин смотрите в разделе 4 руководства *DB2 Сообщения и коды*.

Если код причины начинается с X'00F3' (за исключением X'00F30006'), можно использовать функцию RRSAF TRANSLATE, чтобы получить текст сообщения об ошибке на печать или на экран.

Для вызовов SQL RRSAF возвращает в SQLCA стандартные коды возврата SQL. Список таких кодов возврата и их значений смотрите в разделе 2 руководства *DB2 Сообщения и коды*. RRSAF возвращает коды возврата и

коды причин IFI в области связи инструментального средства (IFCA). Список таких кодов возврата и их значений смотрите в разделе 4 руководства *DB2 Сообщения и коды*.

Таблица 100. Коды возврата RRSAF

Код возврата	Объяснение
0	Успешное завершение работы.
4	Информация о состоянии. Более подробная информация – в коде причины.
8	Несоответствие уровней выпусков DB2 и RRSAF.
200	Возможно, ошибка пользователя. <sup>1</sup>

**Примечание:**

1. Эта ошибка обычно вызывается одной из следующих причин:
  - Список параметров вызова содержит ошибки.
  - Вызовы функций RRSAF выполняются в неправильном порядке.

Этот тип ошибки не изменяет текущего состояния соединения с DB2. Прикладная программа может продолжать работу, выдав исправленное требование.

---

## Примеры программ

В этом разделе содержатся пример задания JCL для выполнения прикладной программы RRSAF и ассемблерный код для обращения к RRSAF.

### Пример задания JCL, использующего RRSAF

Используйте представленный ниже пример задания JCL как модель использования RRSAF в пакетной среде. Оператор DD для DSNRRSAF запускает трассировку RRSAF. Используйте этот оператор DD только для диагностики ошибок.

```
//имя_задания      JOB    данные_карты_job_MVS
//RRSJCL          EXEC   PGM=прикладная_программа_RRS
//STEPLIB          DD     DSN=загрузочная_библиотека_программы
//                  DD     DSN=загрузочная_библиотека_DB2
:
//SYSPRINT         DD     SYSOUT=*
//DSNRRSAF         DD     DUMMY
//SYSUDUMP         DD     SYSOUT=*
```

### Загрузка и удаление языкового интерфейса RRSAF

В следующем фрагменте кода показано, как прикладная программа загружает модули с точками входа DSNRLI и DSNHLIR языкового интерфейса RRSAF. Сохранение этих точек входа в переменных LIRLI и LISQL гарантирует, что прикладная программа загружает эти модули только один раз.

Удалите загруженные модули, когда прикладной программе более не требуется доступ к DB2.

```
***** ПОЛУЧИТЬ АДРЕСА ТОЧЕК ВХОДА ЯЗЫКОВОГО ИНТЕРФЕЙСА *****
LOAD EP=DSNRLI      Загрузить модуль с точкой входа функций RRSASF
ST   R0,LIRLI        Сохранить эту точку входа для вызовов RRSAF
LOAD EP=DSNHLIR     Загрузить модуль с точкой входа RRSAF для SQL
ST   R0,LISQL         Сохранить эту точку входа для вызовов SQL
*
*   .
*   .     Вставьте здесь вызовы функций соединения RRSAF и вызовы SQL
*
*   .
DELETE EP=DSNRLI    Для правильного значения счетчика использования
DELETE EP=DSNHLIR   Для правильного значения счетчика использования
```

## Использование фиктивной точки входа DSNHLI

Каждое средство подключения DB2 содержит точку входа с именем DSNHLI. Если используется RRSAF, но не задана опция препроцессора ATTACH(RRSAF), препроцессор генерирует инструкции BALR вызовов DSNHLI для операторов SQL программы. Чтобы задать правильную точку входа DSNHLI, не включая DSNRLI в загрузочный модуль, напишите процедуру, имеющую точку входа с именем DSNHLI и передающую управление точке входа DSNHLIR в модуле DSNRLI. DSNHLIR – это уникальное имя точки входа для DSNRLI; эта точка входа располагается в DSNRLI в том же месте, что и точка входа DSNHLI. DSNRLI использует 31-битную адресацию. Если прикладная программа,зывающая эту промежуточную процедуру, использует 24-битную адресацию, промежуточная процедура должна выполнять преобразования.

В следующем примере адрес LISQL доступен, поскольку в этом вызывающем CSECT используется тот же регистр 12, что и в CSECT DSNHLI. Аналогично при написании прикладной программы необходимо обеспечить доступность адреса LISQL.

```
*****
* Процедура DSNHLI перехватывает вызовы к точке входа DSNHLI языкового
* интерфейса
*****
DS   0D
DSNHLI CSECT          Начало CSECT
STM  R14,R12,12(R13)  Пролог процедуры
LA   R15,SAVEHLI      Получить адрес области сохранения
ST   R13,4(,R15)       Связать в цепочку области сохранения
ST   R15,8(,R13)       Связать в цепочку области сохранения
LR   R13,R15           Поместить в R13 адрес области сохранения
L    R15,LISQL         Получить адрес настоящей точки
                      входа DSNHLI
                      Вызов DSNRLI, выполняющей вызов SQL
BASSM R14,R15          DSNRLI использует 31-битный режим,
                        поэтому используйте BASSM, чтобы
                        обеспечить предохранение режима адресации.
L    R13,4(,R13)        Восстановить R13 (адрес области сохранения
                      вызывающей программы)
L    R14,12(,R13)        Восстановить R14 (адрес возврата)
RETURN (1,12)           Восстановить R1-12, но НЕ R0 и R15 (коды)
```

## Установление соединения с DB2

На рис. 204 показано, как выдаются требования для некоторых функций RRSAF (IDENTIFY, SIGNON, CREATE THREAD, TERMINATE THREAD и TERMINATE IDENTIFY).

В программах на рис. 204 не показано задание, ожидающее ECB завершения DB2. Можно написать такую программу, используя для ожидания ECB макрокоманду MVS WAIT. При получении ECB завершения задание, ожидающая этот ECB, должна завершить соединение для данного примера программы. Это задание также может ожидать ECB запуска DB2. Задание на рис. 204 ожидает ECB запуска на своем собственном уровне задания.

```
***** IDENTIFY *****
L      R15,LIRLI      Получить адрес языкового интерфейса
CALL (15),(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB),VL,MF=X
          (E,RRSAFCLL)
BAL    R14,CHEKCODE   Вызвать процедуру (не показана),
*           проверяющую коды возврата и причины
CLC    CONTROL,CONTINUE Все в порядке?
BNE    EXIT           Если нет (CONTROL не 'CONTINUE'), выход
USING  R8,RIB         Подготовка в обращению к RIB
L      R8,RIBPTR       Обращение к RIB, чтобы получить уровень
                      выпуска DB2
WRITE 'The current DB2 release level is' RIBREL
*****
***** SIGNON *****
L      R15,LIRLI      Получить адрес языкового интерфейса
CALL (15),(SGNONFN,CORRID,ACCTTKN,ACCTINT),VL,MF=(E,RRSAFCLL)
BAL    R14,CHEKCODE   Проверить коды возврата и причины
*****
***** CREATE THREAD *****
L      R15,LIRLI      Получить адрес языкового интерфейса
CALL (15),(CRTHRDFN,PLAN,COLLID,REUSE),VL,MF=(E,RRSAFCLL)
BAL    R14,CHEKCODE   Проверить коды возврата и причины
*****
***** SQL *****
*           Вставьте здесь свои вызовы SQL. Прекомпилятор DB2
*           сгенерирует вызовы точки входа DSNHLI. Нужно создать
*           фиктивную точку входа с этим именем для перехвата всех
*           вызовов SQL. Фиктивная точка входа DSNHLI показана ниже.
*****
***** TERMINATE THREAD *****
CLC    CONTROL,CONTINUE Все OK?
BNE    EXIT           Если нет (CONTROL не 'CONTINUE'), выход
L      R15,LIRLI      Получить адрес языкового интерфейса
CALL (15),(TRMTHDFN),VL,MF=(E,RRSAFCLL)
BAL    R14,CHEKCODE   Проверить коды возврата и причины
*****
***** TERMINATE IDENTIFY *****
CLC    CONTROL,CONTINUE Все в порядке?
BNE    EXIT           Если нет (CONTROL не 'CONTINUE'), выход
L      R15,LIRLI      Получить адрес языкового интерфейса
CALL (15),(TMIDFYFN),VL,MF=(E,RRSAFCLL)
BAL    R14,CHEKCODE   Проверить коды возврата и причины
```

Рисунок 204. Использование RRSAF для соединения с DB2

На рис. 205 на стр. 847 показаны объявления некоторых переменных, используемых в программах на рис. 204.

```

***** ПЕРЕМЕННЫЕ, ЗАДАВАЕМЫЕ ПРИКЛАДНОЙ ПРОГРАММОЙ *****
LIRLI   DS   F           Адрес точки входа DSNRLI
LISQL   DS   F           Адрес точки входа DSNHLIR
SSNM    DS   CL4          Имя подсистемы DB2 для IDENTIFY
CORRID  DS   CL12         Относительный ID для SIGNON
ACCTTKN DS   CL22         Фраза учета для SIGNON
ACCTINT DS   CL6          Моменты учета для SIGNON
PLAN    DS   CL8          Имя плана DB2 для CREATE THREAD
COLLID  DS   CL18         ID собрания для CREATE THREAD. Если PLAN
                          содержит имя плана, эта переменная
                          не используется.
*
*
REUSE   DS   CL8          Управляет SIGNON после CREATE THREAD
CONTROL  DS   CL8         Действия, выполняемые прикладной программой
                          на основе кода возврата RRSAF
*****
***** ПЕРЕМЕННЫЕ, ЗАДАВАЕМЫЕ DB2 *****
STARTECB DS  F           ECB запуска DB2
TERMECB  DS  F           ECB завершения DB2
EIBPTR   DS  F           Адрес блока информации о среде
RIBPTR   DS  F           Адрес блока информации о выпуске
*****
***** КОНСТАНТЫ *****
CONTINUE DC  CL8'CONTINUE'      Значение CONTROL: Все OK
IDFYFN   DC  CL18'IDENTIFY'     ' Имя функции RRSAF
SGNONFN  DC  CL18'SIGNON'       ' Имя функции RRSAF
CRTHRDFN DC  CL18'CREATE THREAD' ' Имя функции RRSAF
TRMTHDFN DC  CL18'TERMINATE THREAD' ' Имя функции RRSAF
TMIDFYFN DC  CL18'TERMINATE IDENTIFY' ' Имя функции RRSAF
*****
***** SQLCA и RIB *****
EXEC SQL INCLUDE SQLCA
DSNDRIB   DS             Отображение блока информации о выпуске DB2
*****
***** Список параметров для вызовов RRSAF *****
RRSAFCLL CALL ,(*,*,*,*,*),VL,MF=L

```

*Рисунок 205. Объявления переменных, используемых в подпрограмме соединения RRSAF*



---

## Глава 7–9. Особенности программирования для CICS

В этом разделе обсуждаются некоторые темы, важные для разработчиков прикладных программ CICS:

- Управление утилитой подключения CICS из прикладной программы
- Повышение показателя повторного использования потоков
- Проверка рабочего состояния утилиты подключения CICS

---

### Управление утилитой подключения CICS из прикладной программы

Утилите подключения CICS можно запустить и остановить из прикладной программы. Для запуска утилиты подключения надо включить в исходный текст следующий оператор:

```
EXEC CICS LINK PROGRAM('DSN2COM0')
```

Для остановки утилиты подключения применяется оператор:

```
EXEC CICS LINK PROGRAM('DSN2COM2')
```

При таком методе утилита подключения использует RCT по умолчанию. Имя RCT по умолчанию – это DSN2CT, к которому добавляется одно- или двухсимвольный суффикс. Этот суффикс задает системный администратор в подпараметре DSN2STRT параметра INITPARM процедуры запуска CICS. Если суффикс не задан, CICS использует имя RCT из DSN2CT00.

---

### Повышение показателя повторного использования потоков

В целом следует стремиться к тому, чтобы транзакции по возможности использовали потоки повторно, так как создание потока связано с высокими затратами процессорного времени. В Разделе 5 (Том 2) *DB2 Administration Guide* обсуждается, какие факторы влияют на повторное использование потока CICS и как писать ваши прикладные программы, чтобы управлять этими факторами.

Главное, что можно предпринять для повышения показателя повторного использования потоков – это перед каждой точкой синхронизации закрыть все указатели, объявленные с атрибутом WITH HOLD, так как DB2 не закрывает их автоматически. Поток для прикладной программы с открытым указателем нельзя использовать повторно. Полезно немедленно закрывать все указатели после того, как они перестали использоваться. Дополнительную информацию по влиянию объявления указателей WITH HOLD в прикладных программах CICS смотрите в разделе “Объявление указателя с опцией WITH HOLD” на стр. 129.

## Проверка рабочего состояния утилиты подключения CICS

Для проверки доступности подключения CICS в ваших прикладных программах можно воспользоваться командой INQUIRE EXITPROGRAM. Ниже показано, как это сделать:

```
STST      DS      F
ENTNAME   DS      CL8
EXITPROG  DS      CL8
:
MVC      ENTNAME,=CL8'DSNCSQL'
MVC      EXITPROG,=CL8'DSN2EXT1'
EXEC    CICS INQUIRE EXITPROGRAM(EXITPROG)          X
        ENTRYNAME(ENTNAME) STARTSTATUS(STST) NOHANDLE
CLC     EIBRESP,DFHRESP(NORMAL)
BNE     NOTREADY
CLC     STST,DFHVALUE(CONNECTED)
BNE     NOTREADY
UPNREADY DS      0H
        подключение доступно
NOTREADY DS      0H
        подключение еще не доступно
```

В этом примере команда INQUIRE EXITPROGRAM проверяет, запущен ли менеджер ресурсов для SQL и DSNCSQL. CICS возвращает результаты в поле EIBRESP блока интерфейса EXEC (EXEC interface block, EIB) и в поле, имя которое задается в аргументе параметра STARTSTATUS (в данном случае STST). Если значение EIBRESP указывает на нормальное выполнение команды, а значение STST указывает на доступность менеджера ресурсов, операторы SQL можно выполнять беспрепятственно. Дополнительную информацию о команде INQUIRE EXITPROGRAM смотрите в справочнике *CICS for MVS/ESA System Programming Reference*.

### Внимание

Может возникнуть ситуация, когда система продолжает получать работу, несмотря на то, что она закрыта; это называют эффектом *водостока*.

Такой эффект может быть возникнуть при выполнении двух следующих условий:

- Утилита подключения CICS закрыта.
- Вы используете INQUIRE EXITPROGRAM, чтобы избежать аварийных остановок AEY9.

Дополнительную информацию об эффекте водостока и способах избежать его смотрите в разделе Глава 3 *DB2 Data Sharing: Planning and Administration*.

Если вы используете выпуск CICS, более поздний, чем Версия 4, и задали STANDBY=SQLCODE и STRTWT=AUTO в макрокоманде DSNCRCT TYPE=INIT, проверять, работает ли утилита подключения CICS, до выполнения SQL не требуется. Если оператор SQL выполняется, когда утилита подключения CICS недоступна, DB2 выдает код SQLCODE -923 с кодом причины, который указывает недоступность утилиты подключения.

Информацию о макрокоманде DSNCRCT смотрите в Разделе 2 руководства *DB2 Installation Guide* а разъяснения по SQLCODE –923 – в книге *DB2 Сообщения и коды*.



---

## Глава 7–10. Приемы программирования: Вопросы и ответы

В этой главе даются ответы на некоторые часто задаваемые вопросы о приемах программирования баз данных.

---

### Задание уникального ключа для таблицы

**Вопрос:** Как задать уникальный идентификатор для таблицы, в которой нет уникального столбца?

**Ответ:** Добавьте столбец типа ROWID. В столбце типа ROWID для каждой строки таблицы содержится уникальное значение. Можно определить столбец ROWID с опцией GENERATED ALWAYS, тогда вы не сможете помещать в него какие-либо значения, или GENERATED BY DEFAULT, тогда DB2 будет генерировать для него значение, если вы не зададите его сами. Если вы задаете столбец ROWID с опцией GENERATED BY DEFAULT, у этого столбца должен быть уникальный индекс по одному столбцу.

---

### Просмотр ранее полученных данных

**Вопрос:** Когда программа получает данные из базы данных, оператор FETCH позволяет программе просматривать данные вперед. Как программа может обратиться к данным, уже прочитанным ранее?

**Ответ:** В DB2 нет оператора SQL выборки назад. Это значит, что у программы есть две возможности:

- Сохранять копию полученных данных и просматривать их какими-то собственными программными приемами.
- Воспользоваться SQL, чтобы опять получить эти данные, обычно – с помощью второго оператора SELECT. Метод здесь зависит от того, в каком порядке вы хотите просматривать данные: с начала, с середины или в обратном порядке.

Эти возможности более подробно рассмотрены ниже.

### Сохранение копии данных

QMF пользуется первым указанным вариантом: она сохраняет выбранные данные в виртуальной памяти. Если данные не помещаются в виртуальной памяти, QMF записывает их в набор данных BDAM под названием DSQSPILL. (Подробную информациюсмотрите в справочнике утилита управления запросами: *Reference*.)

Одно из следствий такого подхода – то, что при обратном просмотре вы будете видеть в точности ту же информацию, которую получили ранее, даже если после этого данные в базе изменились. Это может быть преимуществом, если вашим пользователям надо работать с постоянным набором данных. Но если пользователям нужно видеть изменения сразу, как только другие пользователи их внесут, это является недостатком.

Если вы пользуетесь этим приемом и вам не надо принимать изменения после получения данных, ваша программа может заблокировать доступ прочих пользователей к данным. (Подробная информация о блокировках в программе дается в разделе “Глава 5–2. Планирование одновременности” на стр. 351.)

## Получение с начала

Чтобы опять начать получать данные с начала, просто закройте активный указатель и опять откройте его. Указатель будет установлен на начало таблицы результатов. Если программа не сохраняет блокировки для всех данных, они могут быть изменены, и прежнее содержание первой строки таблицы результатов может уже не находиться в ней.

## Получение с середины

Чтобы еще раз получить данные откуда–нибудь из середины таблицы результатов, выполните второй оператор SELECT и объявите для него второй указатель. Допустим, например, что первый оператор SELECT имел следующий вид:

```
SELECT * FROM DSN8610.DEPT  
      WHERE LOCATION = 'CALIFORNIA'  
      ORDER BY DEPTNO;
```

Допустим также, что теперь вы хотите вернуться к строкам, начинающимся с DEPTNO = 'M95', и начать последовательное чтение с этой точки. Выполните:

```
SELECT * FROM DSN8610.DEPT  
      WHERE LOCATION = 'CALIFORNIA'  
      AND DEPTNO >= 'M95'  
      ORDER BY DEPTNO;
```

Этот оператор установит указатель на нужное вам место.

Но и в этом случае, если программа не заблокировала данные, другие пользователи могут вставить или удалить строки. Страна с DEPTNO = 'M95' может уже не существовать. Может также оказаться, что с DEPTNO от M95 и M99 теперь стало 20 строк, хотя раньше их было только 16.

**Порядок строк во второй таблице результатов:** Строки второй таблицы результатов могут оказаться в другом порядке. DB2 не учитывает порядок строк, если в операторе SELECT не указано ORDER BY. Таким образом, если есть несколько строк с одним и тем же значением DEPTNO, второй оператор SELECT может получить их в другом порядке, чем первый оператор SELECT. Гарантирано только то, что строки будут упорядочены по номеру отдела, как задано условием ORDER BY DEPTNO.

Порядок строк с одним и тем же значением DEPTNO может измениться, даже если вы выполните тот же оператор SQL с теми же переменными хоста второй раз. Например, между выполнениями могла быть изменена статистика в каталоге. Или же могли быть созданы или удалены какие–либо индексы, и PREPARE для оператора SELECT будет выполняться снова. (Описание оператора PREPARE дается в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.)

Скорее всего, порядок изменится, если у второго SELECT был предикат, которого не было у первого. DB2 может решить использовать индекс по новому предикату. Например, DB2 может выбрать индекс по LOCATION для первого оператора в нашем примере, и индекс по DEPTNO – для второго. Поскольку строки читаются в порядке, указанном ключом индекса, второй порядок не обязан совпадать с первым.

Выполнение PREPARE для двух похожих операторов SELECT может привести к разному порядку строк, даже если не изменена статистика и не созданы и не удалены никакие индексы. В нашем примере, если есть много различных значений LOCATION, DB2 может выбрать индекс по LOCATION для обоих операторов. Но если изменить значение DEPTNO во втором операторе, например:

```
SELECT * FROM DSN8610.DEPT  
WHERE LOCATION = 'CALIFORNIA'  
AND DEPTNO >= 'Z98'  
ORDER BY DEPTNO;
```

DB2 может выбрать индекс по DEPTNO. Поскольку форма оператора SQL и значения в нем связаны слабо, никогда не следует предполагать, что два различных оператора SQL будут возвращать строки в одном и том же порядке. Единственный способ, при котором можно гарантировать порядок (но не содержимое) возвращаемых строк – это использование условия ORDER BY, которое однозначно определяет порядок.

## Получение данных в обратном порядке

Если каждому значению DEPTNO соответствует ровно одна строка, следующий оператор задает однозначное упорядочивание строк:

```
SELECT * FROM DSN8610.DEPT  
WHERE LOCATION = 'CALIFORNIA'  
ORDER BY DEPTNO;
```

Чтобы получить те же строки в обратном порядке, необходимо просто указать, что они должны быть упорядочены по убыванию, как в следующем операторе:

```
SELECT * FROM DSN8610.DEPT  
WHERE LOCATION = 'CALIFORNIA'  
ORDER BY DEPTNO DESC;
```

Указатель во втором операторе получает строки в порядке, обратном указателю в первом операторе. Если первый оператор задает однозначное упорядочивание, второй оператор будет получать строки в *строго* обратном порядке.

Чтобы получать строки в обратном порядке, может оказаться полезным иметь два индекса по столбцу DEPTNO, один в порядке возрастания, а другой – в порядке убывания.

**Получение строк из таблицы со столбцом ROWID:** Если в таблице есть столбец ROWID, можно воспользоваться им, чтобы быстро получать строки в обратном порядке. После выполнения первичного SELECT можно запоминать значение ROWID для всех полученных строк. Затем, чтобы получать данные в обратном порядке, можно выполнять операторы SELECT с условием WHERE, который будет сравнивать значение столбца ROWID со всеми ранее сохраненными значениями ROWID.

Допустим, например, что вы добавили в таблицу DSN8610.DEPT столбец DEPTROWID типа ROWID. Чтобы выбрать все названия отделов, а затем получить эти названия в обратном порядке, можно воспользоваться следующей процедурой

```
*****  
/* Объявление переменных хоста */  
*****  
EXEC SQL BEGIN DECLARE SECTION;  
    SQL TYPE IS ROWID  hv_dept_rowid;  
    char[37] hv_deptname;  
EXEC SQL END DECLARE SECTION;  
*****  
/* Объявление прочих переменных */  
*****  
struct rowid_struct {  
    short int length;  
    char data[40]; /* Структура переменных ROWID */  
}  
struct rowid_struct rowid_array[200];  
    /* Массив для полученных */  
    /* ROWID. Предполагается, */  
    /* что будет получено не */  
    /* больше 200 строк. */  
short int i,j,n;  
*****  
/* Объявление указателя для получения названий отделов */  
*****  
        EXEC SQL DECLARE C1 CURSOR FOR  
            SELECT DEPTNAME, DEPTROWID FROM DSN8610.DEPT;  
:  
*****  
/* Получаем название отдела и ROWID из таблицы DEPT */  
/* и записываем ROWID в массив. */  
*****  
        EXEC SQL OPEN C1;  
i=0;  
while(SQLCODE==0) {  
    EXEC SQL FETCH C1 INTO :hv_deptname, :hv_dept_rowid;  
    rowid_array[i].length=hv_dept_rowid.length;  
    for(j=0;j<hv_dept_rowid.length;j++)  
        rowid_array[i].data[j]=hv_dept_rowid.data[j];  
    i++;  
}  
EXEC SQL CLOSE C1;  
n=i-1;           /* Получаем количество элементов массива */  
*****  
/* С помощью значений ROWID получим названия отделов */  
/* в обратном порядке. */  
*****  
for(i=n;i>=0;i--) {  
    hv_dept_rowid.length=rowid_array[i].length;  
    for(j=0;j<hv_dept_rowid.length;j++)  
        hv_dept_rowid.data[j]=rowid_array[i].data[j];  
    EXEC SQL SELECT DEPTNAME INTO :hv_deptname  
        FROM DSN8610.DEPT  
        WHERE DEPTROWID=:hv_dept_rowid;  
}
```

---

## Обновление ранее полученных данных

**Вопрос:** Как возвращаться к данным, прочитанным ранее, и изменять их?

**Ответ:** Одновременные просмотр и изменение данных может привести к непредсказуемым результатам. Выполнение операторов INSERT, UPDATE и DELETE из одного и того же процесса программы при открытом указателе может повлиять на таблицу результатов.

Предположим, например, что вы читаете строки из таблицы T с помощью указателя C, который определен так:

```
EXEC SQL DECLARE C CURSOR FOR SELECT * FROM T;
```

После чтения нескольких строк указатель C будет установлен на некоторую строку таблицы результатов. Если вы вставите в T строку, эффект будет непредсказуемым, так как строки таблицы результатов не упорядочены. Позже FETCH C может получить, а может и не получить новую строку таблицы T.

Если таблица результатов – не таблица только для чтения и вы изменяете уже имеющиеся в таблице строки, можно избежать этой проблемы с помощью позиционируемого оператора UPDATE (оператора UPDATE с условием WHERE CURRENT OF). Объявите два указателя, один для получения строк из таблицы, а другой для позиционируемого UPDATE. Информация о таблицах результатов только для чтения и позиционируемых операторах UPDATE приводится в разделе Глава 6 *DB2 SQL Reference*.

---

## Обновление данных во время получения их из базы данных

**Вопрос:** Как обновлять строки данных при получении?

**Ответ:** В операторе SELECT укажите FOR UPDATE OF, за которым следует список обновляемых столбцов (для увеличения быстродействия следует указывать только те столбцы, которые вы действительно собираетесь обновлять). Затем воспользуйтесь позиционируемым оператором UPDATE. В условии WHERE CURRENT OF задается указатель на строку, которую надо обновить.

---

## Обновление тысяч строк

**Вопрос:** Есть ли какой-нибудь специальный прием для обновления больших объемов данных?

**Ответ:** Да. При обновлении больших объемов данных с помощью указателя можно свести к минимуму время, в течение которого вы блокируете данные, объявив курсор с опцией HOLD и часто выполняя принятие

---

## Получение тысяч строк

**Вопрос:** Есть ли какой-нибудь специальный прием для получения и вывода больших объемов данных?

**Ответ:** Никакого специального приема нет; но для больших объемов данных скорость может быть чрезвычайно важна. В частности, следует иметь в виду возможности блокирования данных, включая возможность расширения блокировок.

Если ваша программа допускает ввод с терминала до того, как она выполняет принятие и тем самым снимает блокировки, возможна серьезная потеря синхронности результатов. При проектировании программы просмотрите описание блокировок в разделе “Опция ISOLATION” на стр. 373. Затем продумайте ожидаемое использование таблиц, чтобы понять, будут ли проблемы с блокировкой.

---

## Использование SELECT \*

**Вопрос:** Когда следует применять SELECT \* ?

**Ответ:** Как правило, нужно выбирать только необходимые столбцы, так как производительность DB2 зависит от количества выбираемых столбцов. Используйте SELECT \*, только когда вы уверены, что хотите выбрать все столбцы. Одной из альтернатив – использовать производные таблицы, определенные только с необходимыми столбцами, и применять оператор SELECT \* для доступа к этим производным таблицам. Избегайте SELECT \*, если все выбранные столбцы участвуют в операции сортировки (например, SELECT DISTINCT и SELECT...UNION).

---

## Оптимизация получения небольшого количества строк

**Вопрос:** Как сообщить DB2, что мне нужно только несколько из тысяч строк, удовлетворяющих запросу?

**Ответ:** Воспользуйтесь опцией OPTIMIZE FOR *n* ROWS.

DB2 обычно оптимизирует запросы для получения всех подходящих строк. Но иногда требуется получить только первые несколько строк. Например, чтобы получить первую строку со значением, которое не меньше заданного, введите:

```
SELECT список_столбцов FROM таблица  
WHERE ключ >= значение  
ORDER BY ключ ASC
```

Даже при указании ORDER BY DB2 может сначала получить все данные и затем отсортировать их, что может быть напрасной тратой времени. Вместо этого введите:

```
SELECT * FROM таблица  
WHERE ключ >= значение  
ORDER BY ключ ASC  
OPTIMIZE FOR 1 ROW
```

OPTIMIZE FOR 1 ROW влияет на путь доступа. OPTIMIZE FOR 1 ROW сообщает DB2, что надо выбрать такой путь доступа, который быстро возвратит первую подходящую строку.

Более подробная информация об условии OPTIMIZE FOR дана в разделе “Минимизация затрат при получении небольшого числа строк: OPTIMIZE FOR n ROWS” на стр. 694.

---

## Добавление данных в конец таблицы

**Вопрос:** Как добавляются данные в конец таблицы?

**Ответ:** Хотя этот вопрос задают достаточно часто, в реляционной базе данных он не имеет смысла. Строки таблицы базы данных не упорядочены; соответственно, у таблицы нет “конца.”

Чтобы получить эффект добавления данных в “конец” таблицы, определите уникальный индекс по столбцу TIMESTAMP при определении таблицы. Затем при получении данных из таблицы воспользуйтесь условием ORDER BY, где указан этот столбец. Последняя по времени вставка будет получена последней.

---

## Перевод требований конечных пользователей в операторы SQL

**Вопрос:** Программа перед выполнением переводит требования конечных пользователей в операторы SQL, и пользователи могут сохранить требование. Каким образом можно сохранить соответствующий оператор SQL?

**Ответ:** Соответствующие операторы SQL можно сохранять в таблице со столбцом с типом данных VARCHAR(*n*), где *n* – максимальная длина любого оператора SQL. Сохранять надо исходные операторы SQL, а не подготовленные версии. Это означает, что надо будет получить, а затем подготовить каждый оператор перед тем как выполнять версию, сохраненную в таблице. По существу, программа будет преобразовывать символьную строку в оператор SQL и динамически выполнять его. (Описание динамического SQL дается в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.)

---

## Изменение определения таблицы

**Вопрос:** Как написать приложение SQL, которое позволит пользователям создавать новые таблицы, добавлять в них столбцы, увеличивать длину символьных столбцов, менять столбцы местами и уничтожать столбцы?

**Ответ:** Ваша программа может динамически выполнять введенные пользователями операторы CREATE TABLE и ALTER TABLE, чтобы создавать новые таблицы, добавлять столбцы в уже имеющиеся таблицы или увеличивать длину столбцов VARCHAR. Добавляемые столбцы изначально заполняются пустым значением или значением по умолчанию. Оба оператора, как и все операторы определения данных, сравнительно дороги в выполнении; учтите еще и эффекты блокировок.

Менять местами или уничтожать столбцы в имеющейся таблице невозможно без уничтожения всей таблицы. Вместо этого можно создать производную таблицу, которая будет включать только нужные столбцы в нужном порядке. Это даст тот же эффект, что и переопределение таблицы.

Динамическое выполнение SQL описано в разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543.

---

## Хранение данных не в табличном формате

**Вопрос:** Как хранить большой объем данных, которые не определены как набор столбцов таблицы?

**Ответ:** Эти данные можно сохранить в базе данных в качестве одного столбца типа VARCHAR.

---

## Поиск нарушенного реляционного или проверочного ограничения

**Вопрос:** Когда нарушено реляционное или проверочное ограничение, как определить, какое именно?

**Ответ:** Получив ошибку SQL из-за нарушения ограничения, выведите SQLCA. Чтобы отформатировать SQLCA, можно воспользоваться процедурой DSNTIAR, описанной в разделе “Обработка кодов возврата ошибок SQL” на стр. 119. Посмотрите название ограничения в тексте вставки сообщения SQL об ошибке (SQLERRM). Сведения о возможных нарушенияхсмотрите в описании кодов SQLCODE от –530 до –548 в разделе Раздел 2 *DB2 Сообщения и коды*.

---

## **Приложения**



## Приложение А. Таблицы примеров DB2

Большинство примеров в этой книге используют таблицы, описанные в данном приложении. Эта группа таблиц содержит информацию о сотрудниках, отделах, проектах и действиях; они используются в программах примеров, которые иллюстрируют большинство возможностей DB2. Хранилища, базы данных, табличные пространства, таблицы и производные таблицы примеров создаются, когда вы запускаете задания установки примеров DSNTEJ1 и DSNTEJ7. В число объектов примеров DB2 входят большие объекты, которые создаются заданием DSNTEJ7. Все прочие объекты примеров создает задание DSNTEJ1. Операторы CREATE INDEX для таблиц примеров здесь не показаны; индексы также создаются заданиями DSNTEJ1 и DSNTEJ7.

Для всех объектов примеров задана авторизация PUBLIC, чтобы программы примеров было проще запускать. Содержимое каждой таблицы легко получить, выполнить оператор SQL, например, SELECT \* FROM DSN8610.PROJ. Для удобства понимания примеров таблицы отделов и сотрудников приводятся здесь полностью.

### Таблица работ (DSN8610.ACT)

Таблица действий описывает работы, которые могут быть выполнены в рамках определенных проектов. Эта таблица находится в базе данных DSN8D61A и создается оператором:

```
CREATE TABLE DSN8610.ACT
  (ACTNO  SMALLINT      NOT NULL,
   ACTKWD  CHAR(6)       NOT NULL,
   ACTDESC VARCHAR(20)   NOT NULL,
   PRIMARY KEY (ACTNO) )
IN DSN8D61A.DSN8S61P
CCSID EBCDIC;
```

### Содержимое

В Табл. 101 показано содержимое столбцов.

Таблица 101. Столбцы таблицы работ

Столбец	Имя столбца	Описание
1	ACTNO	Идентификатор работы (первичный ключ)
2	ACTKWD	Обозначение работы (до 6 символов)
3	ACTDESC	Описание работы

У таблицы работ есть следующие индексы:

Таблица 102. Индексы таблицы работ

Имя	По столбцу	Тип индекса
DSN8610.XACT1	ACTNO	Первичный, восходящий
DSN8610.XACT2	ACTKWD	Индекс уникальности, восходящий

## Отношения с другими таблицами

Таблицы работ является родительской для таблицы работ по проектам через внешний ключ по столбцу ACTNO.

### Таблица отделов (DSN8610.DEPT)

Таблица отделов описывает каждый отдел предприятия и указывает его руководителя и подчиненность.

Это таблица, показанная на Табл. 105 на стр. 865, находится в табличном пространстве DSN8D61A.DSN8S61D и создается оператором:

```
CREATE TABLE DSN8610.DEPT
  (DEPTNO    CHAR(3)          NOT NULL,
   DEPTNAME  VARCHAR(36)      NOT NULL,
   MGRNO     CHAR(6)          ,
   ADMRDEPT  CHAR(3)          NOT NULL,
   LOCATION   CHAR(16)         ,
   PRIMARY KEY (DEPTNO)      )
IN DSN8D61A.DSN8S61D
CCSID EBCDIC;
```

Поскольку эта таблица автореферентна, а также входит в цикл зависимостей, ее внешние ключи надо добавить позже при помощи операторов:

```
ALTER TABLE DSN8610.DEPT
  FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8610.DEPT
    ON DELETE CASCADE;

ALTER TABLE DSN8610.DEPT
  FOREIGN KEY RDE (MGRNO) REFERENCES DSN8610.EMP
    ON DELETE SET NULL;
```

## Содержимое

В Табл. 103 показано содержимое столбцов.

Таблица 103. Столбцы таблицы отделов

Столбец	Имя столбца	Описание
1	DEPTNO	Идентификатор отдела, первичный ключ
2	DEPTNAME	Название отдела, отражающее его функции
3	MGRNO	Номер сотрудника (EMPNO) – руководителя отдела
4	ADMRDEPT	Идентификатор отдела, перед которым данный отдел отчитывается; отдел высшего уровня отчитывается перед самим собой.
5	LOCATION	Местоположение отдела

У таблицы отделов есть следующие индексы:

Таблица 104. Индексы таблицы отделов

Имя	По столбцу	Тип индекса
DSN8610.XDEPT1	DEPTNO	Первичный, восходящий
DSN8610.XDEPT2	MGRNO	Восходящий
DSN8610.XDEPT3	ADMRDEPT	Восходящий

## Отношения с другими таблицами

Это автореферентная таблица: значение столбца вышестоящего отдела должно быть идентификатором отдела.

Эта таблица является родительской для:

- Таблицы сотрудников, через внешний ключ по столбцу WORKDEPT
- Таблицы проектов, через внешний ключ по столбцу DEPTNO

Она зависит от таблицы сотрудников через ее внешний ключ по столбцу MGRNO.

Таблица 105. DSN8610.DEPT: таблица отделов

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	_____
B01	PLANNING	000020	A00	_____
C01	INFORMATION CENTER	000030	A00	_____
D01	DEVELOPMENT CENTER	_____	A00	_____
E01	SUPPORT SERVICES	000050	A00	_____
D11	MANUFACTURING SYSTEMS	000060	D01	_____
D21	ADMINISTRATION SYSTEMS	000070	D01	_____
E11	OPERATIONS	000090	E01	_____
E21	SOFTWARE SUPPORT	000100	E01	_____
F22	BRANCH OFFICE F2	_____	E01	_____
G22	BRANCH OFFICE G2	_____	E01	_____
H22	BRANCH OFFICE H2	_____	E01	_____
I22	BRANCH OFFICE I2	_____	E01	_____
J22	BRANCH OFFICE J2	_____	E01	_____

Столбец LOCATION содержит пустые значения, пока программа примера DSNTEJ6 не заполнит его местоположениями отделов.

---

## Таблица сотрудников (DSN8610.EMP)

В таблице сотрудников для каждого сотрудника указан номер сотрудника и основные сведения о нем.

Эта таблица, показанная в Табл. 108 на стр. 868 и Табл. 109 на стр. 869, находится в многораздельном табличном пространстве DSN8D61A.DSN8S61E. Поскольку у этой таблицы есть внешний ключ, ссылающийся на DEPT, эту таблицу и ее индекс по первичному ключу следует создать до таблицы сотрудников. Затем создается таблица EMP:

```
CREATE TABLE DSN8610.EMP
  (EMPNO      CHAR(6)          NOT NULL,
   FIRSTMNAME VARCHAR(12)      NOT NULL,
   MIDINIT    CHAR(1)          NOT NULL,
   LASTNAME   VARCHAR(15)      NOT NULL,
   WORKDEPT   CHAR(3),
   PHONENO    CHAR(4)          CONSTRAINT NUMBER CHECK
                               (PHONENO >= '0000' AND
                                PHONENO <= '9999'),
   HIREDATE   DATE            ,
   JOB        CHAR(8)          ,
   EDLEVEL    SMALLINT        ,
   SEX        CHAR(1)          ,
   BIRTHDATE  DATE            ,
   SALARY     DECIMAL(9,2)      ,
   BONUS      DECIMAL(9,2)      ,
   COMM       DECIMAL(9,2)      ,
   PRIMARY KEY (EMPNO)
   FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8610.DEPT
                                     ON DELETE SET NULL
   )
EDITPROC DSN8EAE1
IN DSN8D61A.DSN8S61E
CCSID EBCDIC;
```

## Содержимое

В Табл. 106 на стр. 867 показано содержимое столбцов. У этой таблицы есть проверочное ограничение NUMBER, которое проверяет, что телефонный номер находится в пределах от 0000 до 9999.

Таблица 106. Столбцы таблицы сотрудников

Столбец	Имя столбца	Описание
1	EMPNO	Номер сотрудника (первичный ключ)
2	FIRSTNAME	Имя сотрудника
3	MINIT	Средний инициал сотрудника
4	LASTNAME	Фамилия сотрудника
5	WORKDEPT	Идентификатор отдела, где работает сотрудник
6	PHONE NO	Телефонный номер сотрудника
7	HIREDATE	Дата приема на работу
8	JOB	Должность сотрудника
9	EDLEVEL	Уровень образования (число лет)
10	SEX	Пол сотрудника (M – мужской, F – женский)
11	BIRTHDATE	Дата рождения
12	SALARY	Годовой оклад в долларах
13	BONUS	Годовая премия в долларах
14	COMM	Годовые комиссионные в долларах

У этой таблицы есть следующие индексы:

Таблица 107. Индексы таблицы сотрудников

Имя	По столбцу	Тип индекса
DSN8610.XEMP1	EMPNO	Первичный, индекс разделения, восходящий
DSN8610.XEMP2	WORKDEPT	Восходящий

## Отношения с другими таблицами

Эта таблицы является родительской для:

- Таблицы отделов, через внешний ключ по столбцу MGRNO
- Таблицы проектов, через внешний ключ по столбцу RESPEMP

Она зависит от таблицы отделов через ее внешний ключ по столбцу WORKDEPT.

Таблица 108. Левая половина таблицы сотрудников DSN8610.EMP:. Обратите внимание на то, что пропуск в столбце MIDINIT – это значение ' ' (один символ пробел), а не пустое значение.

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10
000030	SALLY	A	KWAN	C01	4738	1975-04-05
000050	JOHN	B	GEYER	E01	6789	1949-08-17
000060	IRVING	F	STERN	D11	6423	1973-09-14
000070	EVA	D	PULASKI	D21	7831	1980-09-30
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19
000110	VINCENZO	G	LUCCHESI	A00	3490	1958-05-16
000120	SEAN		O'CONNELL	A00	2167	1963-12-05
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15
000150	BRUCE		ADAMSON	D11	4510	1972-02-12
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07
000190	JAMES	H	WALKER	D11	2986	1974-07-26
000200	DAVID		BROWN	D11	4501	1966-03-03
000210	WILLIAM	T	JONES	D11	0942	1979-04-11
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1969-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5698	1947-05-05
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01
200120	GREG		ORLANDO	A00	2167	1972-05-05
200140	KIM	H	NATZ	C01	1793	1976-12-15
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15
200220	REBA	K	JOHN	D11	0672	1968-08-29
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12
200330	HELENA		WONG	E21	2103	1976-02-23
200340	ROY	R	ALONZO	E21	5698	1947-05-05

Таблица 109. Правая половина таблицы сотрудников DSN8610.EMP:

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000010)	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
(000020)	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
(000030)	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
(000050)	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
(000060)	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
(000070)	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
(000090)	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
(000100)	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
(000110)	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
(000120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(000130)	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
(000140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(000150)	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
(000160)	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
(000170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(000180)	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
(000190)	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
(000200)	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
(000210)	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
(000220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(000230)	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
(000240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(000250)	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
(000260)	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
(000270)	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
(000280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(000290)	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
(000300)	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00
(000310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
(000320)	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
(000330)	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
(000340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00
(200010)	SALESREP	18	F	1933-08-14	46500.00	1000.00	4220.00
(200120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(200140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(200170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(200220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(200240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(200280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(200310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
(200330)	FIELDREP	14	F	1941-07-18	25370.00	500.00	2030.00
(200340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

## Таблица фотографий и резюме сотрудников (DSN8610.EMP\_PHOTO\_RESUME)

Фотографии и резюме сотрудников дополняют таблицу сотрудников. Каждая строка таблицы фотографий и резюме содержит фотографию сотрудника в двух форматах и резюме сотрудника. Таблица фотографий и резюме находится в табличном пространстве DSN8D61A.DSN8S61E. Эту таблицу создает следующий оператор:

```
CREATE TABLE DSN8610.EMP_PHOTO_RESUME
  (EMPNO      CHAR(06) NOT NULL,
   EMP_ROWID  ROWID GENERATED ALWAYS,
   PSEG_PHOTO BLOB(100K),
   BMP_PHOTO  BLOB(100K),
   RESUME     CLOB(5K))
  PRIMARY KEY EMPNO
  IN DSN8D61L.DSN8S61B
  CCSID EBCDIC;
```

Для каждого столбца большого объекта в таблице DB2 требуется дополнительная таблица. Следующие операторы определяют дополнительные таблицы для трех столбцов больших объектов в DSN8610.EMP\_PHOTO\_RESUME:

```
CREATE AUX TABLE DSN8610.AUX_BMP_PHOTO
  IN DSN8D61L.DSN8S61M
  STORES DSN8610.EMP_PHOTO_RESUME
  COLUMN BMP_PHOTO;

CREATE AUX TABLE DSN8610.AUX_PSEG_PHOTO
  IN DSN8D61L.DSN8S61L
  STORES DSN8610.EMP_PHOTO_RESUME
  COLUMN PSEG_PHOTO;
```

```
CREATE AUX TABLE DSN8610.AUX_EMP_RESUME
  IN DSN8D61L.DSN8S61N
  STORES DSN8610.EMP_PHOTO_RESUME
  COLUMN RESUME;
```

## Содержимое

В Табл. 110 показано содержимое столбцов.

Таблица 110. Столбцы таблицы фотографий и резюме сотрудников

Столбец	Имя столбца	Описание
1	EMPNO	Идентификатор сотрудника (первичный ключ)
2	EMP_ROWID	ID строки, однозначно определяющий каждую строку таблицы. Значения в этот столбец заносит DB2.
3	PSEG_PHOTO	Фотография сотрудника в формате PSEG
4	BMP_PHOTO	Фотография сотрудника в формате BMP
5	RESUME	Резюме сотрудника

У таблицы фотографий и резюме сотрудников есть индексы:

Таблица 111. Индексы таблицы фотографий и резюме сотрудников

Имя	По столбцу	Тип индекса
DSN8610.XEMP_PHOTO_RESUME	EMPNO	Первичный, восходящий

У дополнительных таблиц для таблицы фотографий и резюме сотрудников есть индексы:

Таблица 112. Индексы дополнительных таблицы для таблицы фотографий и резюме сотрудников

Имя	По таблице	Тип индекса
DSN8610.XAUX_BMP_PHOTO	DSN8610.AUX_BMP_PHOTO	Индекс уникальности
DSN8610.XAUX_PSEG_PHOTO	DSN8610.AUX_PSEG_PHOTO	Индекс уникальности
DSN8610.XAUX_EMP_RESUME	DSN8610.AUX_EMP_RESUME	Индекс уникальности

## Отношения с другими таблицами

Эта таблица является родительской для таблицы проектов через внешний ключ по столбцу RESPEMP.

## Таблица проектов (DSN8610.PROJ)

Таблица проектов описывает каждый текущий проект предприятия. Данные в каждой строке содержат номер проекта, название, ответственного и плановые даты.

Эта таблица находится в базе данных DSN8D61A. Поскольку у нее есть внешние ключи, ссылающиеся на DEPT и EMP, эти таблицы и индексы их первичных ключей должны быть созданы раньше. Затем создается таблица PROJ:

```
CREATE TABLE DSN8610.PROJ
  (PROJNO  CHAR(6) PRIMARY KEY NOT NULL,
   PROJNAME VARCHAR(24)      NOT NULL WITH DEFAULT
     'PROJECT NAME UNDEFINED',
   DEPTNO  CHAR(3)           NOT NULL REFERENCES
     DSN8610.DEPT ON DELETE RESTRICT,
   RESPEMP  CHAR(6)           NOT NULL REFERENCES
     DSN8610.EMP ON DELETE RESTRICT,
   PRSTAFF  DECIMAL(5, 2)      ,
   PRSTDATE DATE              ,
   PRENDATE DATE              ,
   MAJPROJ  CHAR(6))
  IN DSN8D61A.DSN8S61P
  CCSID EBCDIC;
```

Поскольку эта таблица автореферентна, позже должен быть добавлен внешний ключ для реляционной связи:

```
ALTER TABLE DSN8610.PROJ
    FOREIGN KEY RPP (MAJPROJ) REFERENCES DSN8610.PROJ
        ON DELETE CASCADE;
```

## Содержимое

В Табл. 113 показано содержимое столбцов.

*Таблица 113. Столбцы таблицы проектов*

Столбец	Имя столбца	Описание
1	PROJNO	Идентификатор проекта (первичный ключ)
2	PROJNAME	Название проекта
3	DEPTNO	Идентификатор отдела, ответственного за проект
4	RESPEMP	Идентификатор сотрудника, ответственного за проект
5	PRSTAFF	Оценка среднего числа занятых от PRSTDATE до PRENDATE для выполнения проекта в целом со всеми его подпроектами
6	PRSTDATE	Планируемая дата начала проекта
7	PRENDATE	Планируемая дата завершения проекта
8	MAJPROJ	Идентификатор проекта, в который входит данный проект

У таблицы проектов есть следующие индексы:

*Таблица 114. Индексы таблицы проектов*

Имя	По столбцу	Тип индекса
DSN8610.XPROJ1	PROJNO	Первичный, восходящий
DSN8610.XPROJ2	RESPEMP	Восходящий

## Отношения с другими таблицами

Это автореферентная таблица: непустое значение MAJPROJ должно быть номером проекта. Эта таблица является родительской для таблицы работ по проектам через внешний ключ по столбцу PROJNO. Она зависит от:

- Таблицы отделов, через ее внешний ключ по DEPTNO
- Таблицы сотрудников, через ее внешний ключ по RESPEMP

---

## Таблица работ по проектам (DSN8610.PROJACT)

В таблице работ по проектам перечисляются работы, выполняемые по каждому проекту. Эта таблица находится в базе данных DSN8D61A.

Поскольку у нее есть внешние ключи, ссылающиеся на PROJ и ACT, эти таблицы и индексы их первичных ключей должны быть созданы раньше. Затем создается таблица PROJACT:

```

CREATE TABLE DSN8610.PROJACT
  (PROJNO      CHAR(6)                      NOT NULL,
   ACTNO       SMALLINT                     NOT NULL,
   ACSTAFF     DECIMAL(5,2)                  ,
   ACSTDATE    DATE                         NOT NULL,
   ACENDATE    DATE                         ,
   PRIMARY KEY (PROJNO, ACTNO, ACSTDATE),
   FOREIGN KEY RPAP (PROJNO) REFERENCES DSN8610.PROJ
                                         ON DELETE RESTRICT,
   FOREIGN KEY RPAA (ACTNO) REFERENCES DSN8610.ACT
                                         ON DELETE RESTRICT)
IN DSN8D61A.DSN8S61P
CCSID EBCDIC;

```

## Содержимое

В Табл. 115 показано содержимое столбцов.

*Таблица 115. Столбцы таблицы работ по проектам*

Столбец	Имя столбца	Описание
1	PROJNO	Идентификатор проекта
2	ACTNO	Идентификатор работы
3	ACSTAFF	Оценка среднего число сотрудников проекта
4	ACSTDATE	Планируемая дата начала работы
5	ACENDATE	Планируемая дата завершения работы

У таблицы работ по проектам есть индекс:

*Таблица 116. Индекс таблицы работ по проектам*

Имя	По столбцам	Тип индекса
DSN8610.XPROJAC1	PROJNO, ACTNO, ACSTDATE	Первичный, восходящий

## Отношения с другими таблицами

Эта таблица является родительской для таблицы сотрудников работ по проектам через внешний ключ по столбцам PROJNO, ACTNO и EMSTDATE. Она зависит от:

- Таблицы работ, через ее внешний ключ по столбцу ACTNO
- Таблицы проектов, через ее внешний ключ по столбцу PROJNO

---

## Таблица сотрудников работ по проектам (DSN8610.EMPPROJACT)

Таблица сотрудников работ по проектам указывает сотрудников, которые выполняют работы по проектам, доли рабочего времени для каждого сотрудника и плановые даты работ.

Эта таблица находится в базе данных DSN8D61A. Поскольку у нее есть внешние ключи, ссылающиеся на EMP и PROJACT, эти таблицы и индексы их первичных ключей должны быть созданы раньше. Затем создается таблица EMPPROJACT:

```

CREATE TABLE DSN8610.EMPPROJACT
  (EMPNO      CHAR(6)          NOT NULL,
   PROJNO     CHAR(6)          NOT NULL,
   ACTNO      SMALLINT        NOT NULL,
   EMPTIME    DECIMAL(5,2)      ,
   EMSTDATE   DATE            ,
   EMENDATE   DATE            ,
   FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
   REFERENCES DSN8610.PROJACT
   ON DELETE RESTRICT,
   FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8610.EMP
   ON DELETE RESTRICT)
IN DSN8D61A.DSN8S61P
CCSID EBCDIC;

```

## Содержимое

В Табл. 117 показано содержимое столбцов.

*Таблица 117. Столбцы таблицы сотрудников работ по проектам примера*

Столбец	Имя столбца	Описание
1	EMPNO	Номер сотрудника
2	PROJNO	Идентификатор проекта
3	ACTNO	Идентификатор работы по проекту
4	EMPTIME	Доля полного рабочего времени (от 0,00 до 1,00) сотрудника на данную работу
5	EMSTDATE	Дата начала работы
6	EMENDATE	Дата завершения работы

У этой таблицы есть следующие индексы:

*Таблица 118. Индексы таблицы сотрудников работ по проектам примера*

Имя	По столбцам	Тип индекса
DSN8610.XEMPPROJECT1	PROJNO, ACTNO, EMSTDATE, EMPNO	Индекс уникальности, восходящий
DSN8610.XEMPPROJECT2	EMPNO	Восходящий

## Отношения с другими таблицами

Эта таблица зависит от:

- Таблицы сотрудников, через ее внешний ключ по столбцу EMPNO
- Таблицы работ по проектам, через ее внешний ключ по столбцам PROJNO, ACTNO и EMSTDATE.

## Отношения между таблицами

На рис. 206 показаны отношения между таблицами. Они устанавливаются внешними ключами в зависимых таблицах, которые ссылаются на первичные ключи в родительских таблицах. Описания столбцов можно найти в описаниях таблиц.

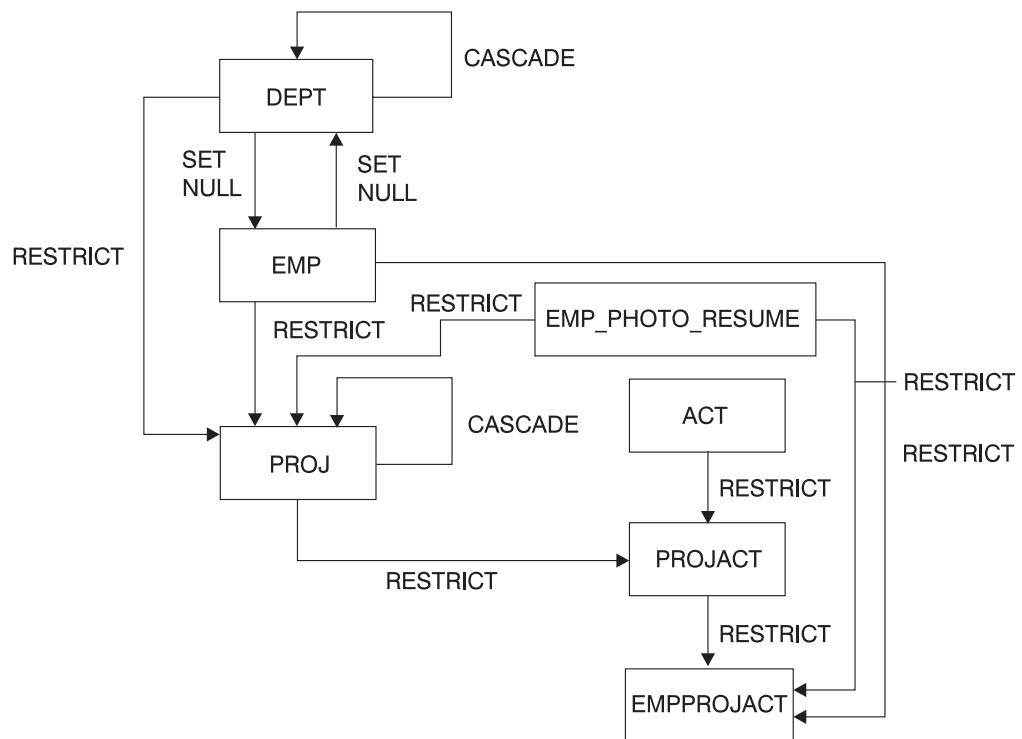


Рисунок 206. Отношения между таблицами в программе примера. Стрелки идут от родительских таблиц к зависимым.

## Производные таблицы для таблиц примеров

DB2 создает для таблиц примера множество производных таблиц, которые используются программами примера. В Табл. 119 на стр. 876 показаны таблицы, для которых определяется каждая производная таблица, и программы примера, где производные таблицы используются. Для всех названий производных таблиц используется спецификатор DSN8610.

Таблица 119. Производные таблицы для таблиц примеров

Имя производной таблицы	Для таблицы или производной таблицы	Используется в программе
VDEPT	DEPT	organization project
VHDEPT	DEPT	Distributed organization
VEMP	EMP	Distributed organization organization project
VPROJ	PROJ	project
VACT	ACT	project
VEMPPROJECT	EMPPROJECT	project
VDEPMG1	DEPT EMP	organization
VEMPDPT1	DEPT EMP	organization
VASTRDE1	DEPT	
VASTRDE2	VDEPMG1 EMP	organization
VPROJRE1	PROJ EMP	project
VPSTRDE1	VPROJRE1 VPROJRE2	project
VPSTRDE2	VPROJRE1	project
VSTAFAC1	PROJECT ACT	project
VSTAFAC2	EMPPROJECT ACT EMP	project
VPHONE	EMP DEPT	phone
VEMPLP	EMP	phone

Операторы SQL, создающие производные таблицы, приводятся ниже.

```

CREATE VIEW DSN8610.VDEPT
    AS SELECT ALL      DEPTNO  ,
                  DEPTNAME,
                  MGRNO   ,
ADMRACT
    FROM DSN8610.DEPT;

CREATE VIEW DSN8610.VHDEPT
    AS SELECT ALL      DEPTNO  ,
                  DEPTNAME,
                  MGRNO   ,
ADMRACT,
LOCATION
    FROM DSN8610.DEPT;

```

```

CREATE VIEW DSN8610.VEMP
AS SELECT ALL      EMPNO   ,
                  FIRSTNME,
                  MIDINIT ,
                  LASTNAME,
                  WORKDEPT
FROM DSN8610.EMP;

CREATE VIEW DSN8610.VPROJ
AS SELECT ALL
          PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,
          PRSTDATE, PRENDATE, MAJPROJ
FROM DSN8610.PROJ;

CREATE VIEW DSN8610.VACT
AS SELECT ALL      ACTNO   ,
                  ACTKWD ,
                  ACTDESC
FROM DSN8610.ACT ;

CREATE VIEW DSN8610.VPROJECT
AS SELECT ALL
          PROJNO, ACTNO, ACSTAFF, ACSTDATE, ACENDATE
FROM DSN8610.PROJECT ;

CREATE VIEW DSN8610.VEMPPROJECT
AS SELECT ALL
          EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE
FROM DSN8610.EMPPROJECT ;

CREATE VIEW DSN8610.VDEPMG1
(DEPTNO, DEPTNAME, MGRNO, FIRSTNME, MIDINIT, LASTNAME, ADMRDEPT)
AS SELECT ALL
          DEPTNO, DEPTNAME, EMPNO, FIRSTNME, MIDINIT, LASTNAME, ADMRDEPT
          FROM DSN8610.DEPT LEFT OUTER JOIN DSN8610.EMP
          ON MGRNO = EMPNO ;

CREATE VIEW DSN8610.VEMPDPT1
(DEPTNO, DEPTNAME, EMPNO, FRSTINIT, MIDINIT,
LASTNAME, WORKDEPT)
AS SELECT ALL
          DEPTNO, DEPTNAME, EMPNO, SUBSTR(FIRSTNME, 1, 1), MIDINIT,
          LASTNAME, WORKDEPT
          FROM DSN8610.DEPT RIGHT OUTER JOIN DSN8610.EMP
          ON WORKDEPT = DEPTNO ;

CREATE VIEW DSN8610.VASTRDE1
(DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
AS SELECT ALL
          D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
          D1.LASTNAME, '1',
          D2.DEPTNO,D2.DEPTNAME,D2.MGRNO,D2.FIRSTNME,D2.MIDINIT,
          D2.LASTNAME
          FROM DSN8610.VDEPMG1 D1, DSN8610.VDEPMG1 D2
          WHERE D1.DEPTNO = D2.ADMRDEPT ;

```

```

CREATE VIEW DSN8610.VASTRDE2
  (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
   DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
AS SELECT ALL
  D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
  D1.LASTNAME,'2',
  D1.DEPTNO,D1.DEPTNAME,E2.EMPNO,E2.FIRSTNME,E2.MIDINIT,
  E2.LASTNAME
  FROM DSN8610.VDEPMG1 D1, DSN8610.EMP E2
  WHERE D1.DEPTNO = E2.WORKDEPT;

CREATE VIEW DSN8610.VPROJRE1
  (PROJNO,PROJNAME,PROJDEP,RESPEMP,FIRSTNME,MIDINIT,
   LASTNAME,MAJPROJ)
AS SELECT ALL
  PROJNO,PROJNAME,DEPTNO,EMPNO,FIRSTNME,MIDINIT,LASTNAME,MAJPROJ
  FROM DSN8610.PROJ, DSN8610.EMP
  WHERE RESPEMP = EMPNO ;

CREATE VIEW DSN8610.VPSTRDE1
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
AS SELECT ALL
  P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
  P1.LASTNAME,
  P2.PROJNO,P2.PROJNAME,P2.RESPEMP,P2.FIRSTNME,P2.MIDINIT,
  P2.LASTNAME
  FROM DSN8610.VPROJRE1 P1,
       DSN8610.VPROJRE1 P2
  WHERE P1.PROJNO = P2.MAJPROJ ;

CREATE VIEW DSN8610.VPSTRDE2
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
AS SELECT ALL
  P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
  P1.LASTNAME,
  P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
  P1.LASTNAME
  FROM DSN8610.VPROJRE1 P1
WHERE NOT EXISTS
  (SELECT * FROM DSN8610.VPROJRE1 P2
   WHERE P1.PROJNO = P2.MAJPROJ) ;

CREATE VIEW DSN8610.VFORPLA
  (PROJNO,PROJNAME,RESPEMP,PROJDEP,FRSTINIT,MIDINIT,LASTNAME)
AS SELECT ALL
  F1.PROJNO,PROJNAME,RESPEMP,PROJDEP, SUBSTR(FIRSTNME, 1, 1),
  MIDINIT, LASTNAME
  FROM DSN8610.VPROJRE1 F1 LEFT OUTER JOIN DSN8610.EMPPROJECT F2
  ON F1.PROJNO = F2.PROJNO;

```

```

CREATE VIEW DSN8610.VSTAFAC1
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDAT,EENDATE, TYPE)
 AS SELECT ALL
    PA.PROJNO, PA.ACTNO, AC.ACTDESC, ' ', ' ', ' ', ' ',
    PA.ACSTAFF, PA.ACSTDAT,
    PA.ACENDATE,'1'
   FROM DSN8610.PROJECT PA, DSN8610.ACT AC
  WHERE PA.ACTNO = AC.ACTNO ;

CREATE VIEW DSN8610.VSTAFAC2
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDAT,EENDATE, TYPE)
 AS SELECT ALL
    EP.PROJNO, EP.ACTNO, AC.ACTDESC, EP.EMPNO,EM.FIRSTNME,
    EM.MIDINIT, EM.LASTNAME, EP.EMPTIME, EP.EMSTDAT,
    EP.EMENDATE,'2'
   FROM DSN8610.EMPPROJECT EP, DSN8610.ACT AC, DSN8610.EMP EM
  WHERE EP.ACTNO = AC.ACTNO AND EP.EMPNO = EM.EMPNO ;

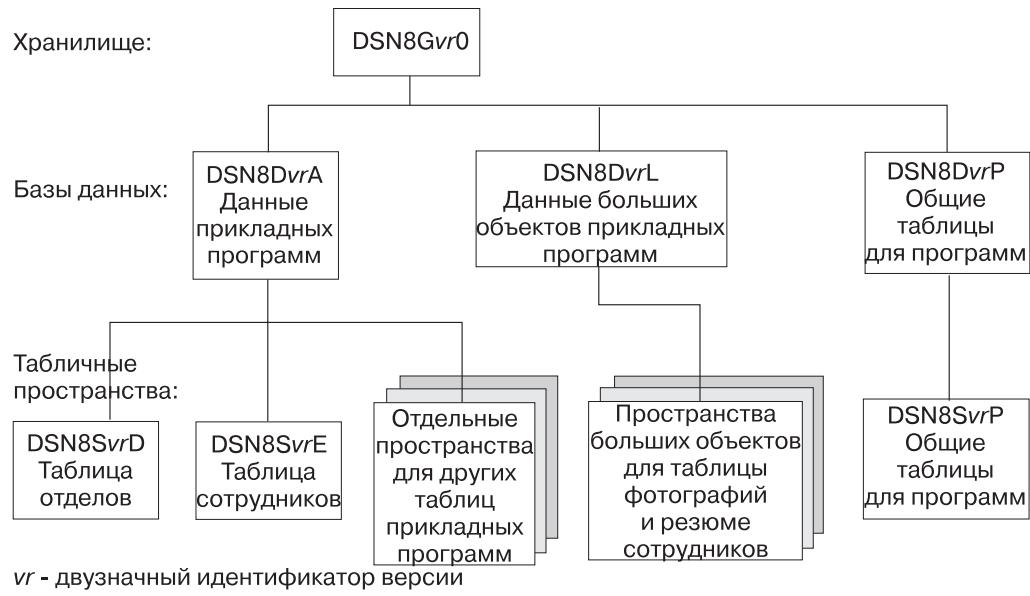
CREATE VIEW DSN8610.VPHONE
  (LASTNAME,
   FIRSTNAME,
   MIDDLEINITIAL,
   PHONENUMBER,
   EMPLOYEENUMBER,
   DEPTNUMBER,
   DEPTNAME)
 AS SELECT ALL      LASTNAME,
                   FIRSTNME,
                   MIDINIT ,
                   VALUE(PHONENO,'      '),
                   EMPNO,
                   DEPTNO,
                   DEPTNAME
   FROM DSN8610.EMP, DSN8610.DEPT
  WHERE WORKDEPT = DEPTNO;

CREATE VIEW DSN8610.VEMPLP
  (EMPLOYEENUMBER,
   PHONENUMBER)
 AS SELECT ALL      EMPNO      ,
                   PHONENO
   FROM DSN8610.EMP ;

```

## Хранение таблиц программ примеров

На рис. 207 на стр. 880 показано, как таблицы примеров связаны с базами данных и хранилищами. Две базы данных вместо одной используются, чтобы проиллюстрировать такую возможность. Обычно взаимосвязанные данные хранятся в одной базе данных.



*Рисунок 207. Отношения между базами данных и табличными пространствами примера*

Кроме хранилища и баз данных, показанных на рис. 207, при запуске DSNEJ2A создается хранилище DSN8G61U и база данных DSN8D61U.

## Хранилище

Хранилище по умолчанию, SYSDEFLT, создаваемое при установке DB2, не используется для данных программ примера. Хранилище, используемое для этих данных, создается оператором:

```
CREATE STOGROUP DSN8G610
  VOLUMES (DSNV01)
  VCAT DSNC610;
```

## Базы данных

База данных по умолчанию, создаваемая при установке DB2, не используется для данных программ примера. Для них используются две базы данных: одна для таблиц, связанных с прикладными программами, другая для таблиц, связанных с системными программами. Они определяются следующими операторами:

```
CREATE DATABASE DSN8D61A
  STOGROUP DSN8G610
  BUFFERPOOL BP0
  CCSID EBCDIC;
```

```
CREATE DATABASE DSN8D61P
  STOGROUP DSN8G610
  BUFFERPOOL BP0
  CCSID EBCDIC;
```

```
CREATE DATABASE DSN8D61L
  STOGROUP DSN8G610
  BUFFERPOOL BP0
  CCSID EBCDIC;
```

## Табличные пространства

Следующие табличные пространства явно определяются показанными ниже операторами. Табличные пространства, не определенные явно, неявно создаются в базе данных DSN8D61A с использованием атрибутов пространства по умолчанию.

```
CREATE TABLESPACE DSN8S61D
    IN DSN8D61A
    USING STOGROUP DSN8G610
        PRIQTY 20
        SECQTY 20
        ERASE NO
    LOCKSIZE PAGE LOCKMAX SYSTEM
    BUFFERPOOL BP0
    CLOSE NO
    CCSID EBCDIC;

CREATE TABLESPACE DSN8S61E
    IN DSN8D61A
    USING STOGROUP DSN8G610
        PRIQTY 20
        SECQTY 20
        ERASE NO
    NUMPARTS 4
        (PART 1 USING STOGROUP DSN8G610
            PRIQTY 12
            SECQTY 12,
        PART 3 USING STOGROUP DSN8G610
            PRIQTY 12
            SECQTY 12)
    LOCKSIZE PAGE LOCKMAX SYSTEM
    BUFFERPOOL BP0
    CLOSE NO
    COMPRESS YES
    CCSID EBCDIC;

CREATE TABLESPACE DSN8S61B
    IN DSN8D61L
    USING STOGROUP DSN8G610
        PRIQTY 20
        SECQTY 20
        ERASE NO
    LOCKSIZE PAGE
    LOCKMAX SYSTEM
    BUFFERPOOL BP0
    CLOSE NO
    CCSID EBCDIC;
```

```
| CREATE LOB TABLESPACE DSN8S61M  
|   IN DSN8D61L  
|   LOG NO;  
  
| CREATE LOB TABLESPACE DSN8S61L  
|   IN DSN8D61L  
|   LOG NO;  
  
| CREATE LOB TABLESPACE DSN8S61N  
|   IN DSN8D61L  
|   LOG NO;  
  
CREATE TABLESPACE DSN8S61C  
  IN DSN8D61P  
    USING STOGROUP DSN8G610  
      PRIQTY 160  
      SECQTY 80  
    SEGSIZE 4  
    LOCKSIZE TABLE  
    BUFFERPOOL BP0  
    CLOSE NO  
    CCSID EBCDIC;  
  
CREATE TABLESPACE DSN8S61P  
  IN DSN8D61A  
    USING STOGROUP DSN8G610  
      PRIQTY 160  
      SECQTY 80  
    SEGSIZE 4  
    LOCKSIZE ROW  
    BUFFERPOOL BP0  
    CLOSE NO  
    CCSID EBCDIC;  
  
CREATE TABLESPACE DSN8S61R  
  IN DSN8D61A  
    USING STOGROUP DSN8G610  
      PRIQTY 20  
      SECQTY 20  
      ERASE NO  
    LOCKSIZE PAGE LOCKMAX SYSTEM  
    BUFFERPOOL BP0  
    CLOSE NO  
    CCSID EBCDIC;  
  
CREATE TABLESPACE DSN8S61S  
  IN DSN8D61A  
    USING STOGROUP DSN8G610  
      PRIQTY 20  
      SECQTY 20  
      ERASE NO  
    LOCKSIZE PAGE LOCKMAX SYSTEM  
    BUFFERPOOL BP0  
    CLOSE NO  
    CCSID EBCDIC;
```

---

## Приложение В. Примеры прикладных программ

В этом приложении описываются примеры прикладных программ DB2 и операционные среды, в которых работает каждая из этих программы. Здесь также приводится информация об использовании этих прикладных программ и о том, как напечатать их исходные тексты.

Несколько примеров прикладных программ поставляются вместе с DB2 для овладения приемами программирования DB2 и практики в создании программ для каждой из следующих сред: пакетной, TSO, IMS и CICS. В эти примеры входят различные прикладные программы, которые могли бы применяться для управления компанией.

Исходный текст этих примеров прикладных программ можно посмотреть в электронной библиотеке примеров, поставляемой вместе с программным продуктом DB2. Эта библиотека примеров называется *префикс.SDSNSAMP*.

---

### Типы примеров прикладных программ

**Прикладная программа organization:** Прикладная программа organization управляет следующей информацией в компании:

- Структурой управления отделами
- Информацией об отделах
- Информацией об отдельных служащих.

Обработка информации о структурах управления отделами включает в себя взаимосвязь между разными отделами. Можно просматривать или изменять организационную структуру каждого отдела и информацию о конкретных сотрудниках любого из отделов. Прикладная программа масштаба организации работает интерактивно в средах ISPF/TSO, IMS и CICS и доступна на языках PL/I и COBOL.

**Прикладная программа project:** Прикладная программа project обрабатывает информацию о работах компаний по проектам, в том числе:

- Структуру проектов
- Списки работ по проектам
- Обработку отдельных проектов
- Обработку оценок работ по отдельным проектам
- Обработку персонала по отдельным проектам.

Каждый из отделов работает над проектами, содержащими наборы связанных между собой работ. Информация об этих работах включает в себя кадровые вопросы, оценки времени завершения всего проекта и отдельных работ по проекту. Прикладная программа project работает интерактивно в IMS и CICS и доступна только на PL/I.

**Прикладная программа phone:** Прикладная программа phone позволяет просматривать или изменять телефонные номера отдельных служащих. Есть различные версии этой прикладной программы для ISPF/TSO, CICS, IMS и пакетной среды:

- Прикладные программы для ISPF/TSO используют COBOL и PL/I.

- Прикладные программы для CICS и IMS используют PL/I.
- Прикладные программы для пакетной среды используют C, C++, COBOL, FORTRAN и PL/I.

**Прикладные программы хранимых процедур:** Есть три набора прикладных программ хранимых процедур:

- Прикладные программы IFI

Эти прикладные программы позволяют передавать команды DB2 от программы клиента хранимой процедуре, запускающей команды на сервере DB2 с использованием интерфейса IFI (instrumentation facility interface – интерфейс инструментальных средств). Есть два набора программ клиентов и хранимых процедур. Один из наборов содержит клиент и хранимую процедуру на языке PL/I; другой – клиент и хранимую процедуру на языке С.

- Прикладная программа ODBA

Эта прикладная программа демонстрирует возможность использования интерфейса ODBA IMS для доступа к базам данных IMS из хранимых процедур. Хранимая процедура обращается к базе данных примера DL/I IMS. Программа клиента и хранимая процедура написаны на языке COBOL.

- Программы хранимых процедур утилит

Эта программа показывает, как вызывать хранимые процедуры утилит. Дополнительную информацию об этих процедурахсмотрите в разделе Приложение В *DB2 Utility Guide and Reference*.

Все прикладные программы хранимых процедур работают в среде TSO batch.

**Прикладные программы пользовательских функций:** Прикладные программы пользовательских функций состоят из программ клиентов, вызывающих примеры пользовательских функций, и набора пользовательских функций, выполняющих следующие функции:

- Преобразование текущей даты в заданный пользователем формат
- Преобразование времени из одного формата в другой
- Преобразование текущего времени в заданный пользователем формат
- Преобразование времени из одного формата в другой
- Возврат дня недели для заданной пользователем даты
- Возврат месяца для заданной пользователем даты
- Форматирование числа с плавающей запятой в виде денежной суммы
- Возврат имени таблицы для таблицы, производной таблицы или алиаса
- Возврат спецификатора для таблицы, производной таблицы или алиаса
- Возврат положения для таблицы, производной таблицы или алиаса
- Возврат таблицы информации о погоде

Все программы написаны на С или С++ и работают в среде TSO.

**Прикладная программа LOB:** Прикладная программа LOB демонстрирует выполнение следующих задач:

- Определения объектов DB2 для хранения данных большого объекта
- Занесение в таблицы DB2 данных больших объектов с использованием утилиты LOAD или же при помощи операторов INSERT и UPDATE, когда данные слишком велики для использования с утилитой LOAD

- Работа с данными больших объектов с использованием локаторов больших объектов

Программы, создающие и наполняющие большие объекты, используют DSNTIAD и работают в среде TSO. Программа, работающая с данными больших объектов, написана на языке С и работает в ISPF/TSO.

## Использование прикладных программ

Эти прикладные программы можно использовать интерактивно, обращаясь к данным в примерах таблиц на экранах дисплея (панелях). К таблицам примеров можно также обращаться из пакетной среды при использовании прикладных программ phone. В разделе 2 *DB2 Installation Guide* содержится подробная информация об использовании каждой из прикладных программ. Для всех программ примеров задана авторизация PUBLIC, что облегчает их запуск.

**Языки и среды прикладных программ:** В Табл. 120 показаны среды, в которых работают прикладные программы и языки, которые они используют в каждой из сред.

Таблица 120. Языки и среды прикладных программ

Программы	ISPF/TSO	IMS	CICS	Пакетная	SPUFI
Программы динамического SQL				Ассемблер PL/I	
Подпрограммы—обработчики	Ассемблер	Ассемблер	Ассемблер	Ассемблер	Ассемблер
organization	COBOL <sup>1</sup>	COBOL PL/I	COBOL PL/I		
phone	COBOL PL/I Ассемблер <sup>2</sup>	PL/I	PL/I	COBOL FORTRAN PL/I C C++	
project		PL/I	PL/I		
Подпрограммы форматирования SQLCA		Ассемблер	Ассемблер	Ассемблер	Ассемблер
Хранимые процедуры				PL/I C COBOL	
Пользовательские функции				C C++	
LOB	C				

**Примечание:**

1. Везде, где в таблице указан COBOL, программу можно скомпилировать с использованием OS/VS COBOL, VS/COBOL II или IBM COBOL for MVS & VM.
2. Подпрограмма ассемблера DSN8CA.

**Прикладные программы:** В таблицах 121 – 123 на страницах 886 – 888 приводятся имена программ, имена компонентов JCL и краткое описание отдельных программ для каждой из трех следующих сред: TSO, IMS и CICS.

## TSO

Таблица 121 (Стр. 1 из 2). Примеры прикладных программ DB2 для TSO

Программа	Имя подпрограммы подготовки	Имя компонента JCL	Утилита подключения	Описание
phone	DSN8BC3	DSNTEJ2C	DSNELI	Эта пакетная программа на языке COBOL выводит список сотрудников и обновляет его по запросу.
phone	DSN8BD3	DSNTEJ2D	DSNELI	Эта пакетная программа на С выводит список номеров телефонов сотрудников и обновляет их по запросу.
phone	DSN8BE3	DSNTEJ2E	DSNELI	Эта пакетная программа на C++ выводит список номеров телефонов сотрудников и обновляет их по запросу.
phone	DSN8BP3	DSNTEJ2P	DSNELI	Эта пакетная программа на языке PL/I выводит список номеров телефонов сотрудников и обновляет их по запросу.
phone	DSN8BF3	DSNTEJ2F	DSNELI	Эта пакетная программа на языке FORTRAN выводит список номеров телефонов сотрудников и обновляет их по запросу.
organization	DSN8HC3	DSNTEJ3C DSNTEJ6	DSNALI	Эта программа ISPF на языке COBOL показывает и обновляет информацию о местном отделе. Она также может показывать и обновлять информацию о сотруднике в местном или удаленном отделе.
phone	DSN8SC3	DSNTEJ3C	DSNALI	Эта программа ISPF на языке COBOL выводит список телефонных номеров сотрудников и обновляет его по запросу.
phone	DSN8SP3	DSNTEJ3P	DSNALI	Эта программа ISPF на языке PL/I выводит список телефонных номеров сотрудников и обновляет его по запросу.
UNLOAD	DSNTIAUL	DSNTEJ2A	DSNELI	Эта программа на языке ассемблера позволяет выгружать данные из таблицы или производной таблицы и генерирует для этих данных управляющие операторы утилиты LOAD.
Динам. SQL	DSNTIAD	DSNTIJTM	DSNELI	Эта программа на языке ассемблера позволяет динамически выполнять операторы (кроме SELECT), читаемые из SYSIN, то есть использует динамический SQL для выполнения операторов SQL не—SELECT.

Таблица 121 (Стр. 2 из 2). Примеры прикладных программ DB2 для TSO

Программа	Имя подпрограммы	Имя компонента JCL	Утилита подключения	Описание
Динам. SQL	DSNTEP2	DSNTEJ1P or DSNTEJ1L	DSNELI	Эта программа на языке PL/I динамически выполняет операторы SQL, читаемые из SYSIN. В отличие от DSNTIAD, эта программа может выполнять и операторы SELECT.
Хранимые процедуры	DSN8EP1	DSNTEJ6P	DSNELI	Эти прикладные программы состоят из вызывающей программы и из программы хранимой процедуры. Примеры, подготовленные заданиями DSNTEJ6P, DSNTEJ6S, DSNTEJ6D и DSNTEJ6T, выполняют команды DB2 с использованием интерфейса IFI. DSNTEJ6P и DSNTEJ6S готовят версию прикладной программы на языке PL/I. DSNTEJ6D и DSNTEJ6T готовят версию на языке С. Хранимая процедура С использует наборы результатов для возврата команд клиенту. Пример, подготовленный DSNTEJ61 и DSNTEJ62, демонстрирует хранимую процедуру, обращающуюся к базам данных IMS через интерфейс ODBA. Пример, подготовленный DSNTEJ6U, вызывает хранимую процедуру утилит.
	DSN8EP2	DSNTEJ6S	DSNALI	
	DSN8EPU	DSNTEJ6U	DSNELI	
	DSN8ED1	DSNTEJ6D	DSNELI	
	DSN8ED2	DSNTEJ6T	DSNALI	
	DSN8EC1	DSNTEJ61	DSNRLI	
	DSN8EC2	DSNTEJ62	DSNELI	
Пользоват. функции	DSN8DUAD	DSNTEJ2U	DSNELI	Эти прикладные программы состоят из набора скалярных пользовательских функций, которые могут быть вызваны через SPUFI или DSNTEP2, и одной табличной пользовательской функции, DSN8DUWF, которая может быть вызвана программой клиента DSN8DUWC. DSN8EUDN и DSN8EUMN написаны на языке C++. Все остальные программы написаны на языке С.
	DSN8DUAT	DSNTEJ2U	DSNELI	
	DSN8DUCD	DSNTEJ2U	DSNELI	
	DSN8DUCT	DSNTEJ2U	DSNELI	
	DSN8DUCY	DSNTEJ2U	DSNELI	
	DSN8DUTI	DSNTEJ2U	DSNELI	
	DSN8DUWC	DSNTEJ2U	DSNELI	
	DSN8DUWF	DSNTEJ2U	DSNELI	
	DSN8EUDN	DSNTEJ2U	DSNELI	
	DSN8EUMN	DSNTEJ2U	DSNELI	
LOB	DSN8DLPL	DSNTEJ71	DSNELI	Эти прикладные программы демонстрируют заполнение столбца большого объекта размером больше 32 Кбайт, обработку данных с использованием встроенных функций POSSTR и SUBSTR и отображение данных в ISPF с использованием GDDM®.
	DSN8DLCT	DSNTEJ71	DSNELI	
	DSN8DLRV	DSNTEJ73	DSNELI	
	DSN8DLPV	DSNTEJ75	DSNELI	

## IMS

Таблица 122 (Стр. 1 из 2). Примеры прикладных программ DB2 для IMS

Программа	Имя программы	Имя компонента JCL	Описание
organization	DSN8IC0 DSN8IC1 DSN8IC2	DSNTEJ4C	Программа organization IMS COBOL

Таблица 122 (Стр. 2 из 2). Примеры прикладных программ DB2 для IMS

Программа	Имя программы	Имя компонента JCL	Описание
organization	DSN8IP0 DSN8IP1 DSN8IP2	DSNTEJ4P	Программа organization IMS PL/I
project	DSN8IP6 DSN8IP7 DSN8IP8	DSNTEJ4P	Программа project IMS PL/I
phone	DSN8IP3	DSNTEJ4P	Программа phone IMS на языке PL/I. Эта программа выводит список телефонных номеров сотрудников и обновляет его по запросу.

## CICS

Таблица 123. Примеры прикладных программ DB2 для CICS

Программа	Имя программы	Имя компонента JCL	Описание
organization	DSN8CC0 DSN8CC1 DSN8CC2	DSNTEJ5C	Программа organization CICS COBOL
organization	DSN8CP0 DSN8CP1 DSN8CP2	DSNTEJ5P	Программа organization CICS PL/I
project	DSN8CP6 DSN8CP7 DSN8CP8	DSNTEJ5P	Программа project CICS PL/I
phone	DSN8CP3	DSNTEJ5P	Программа phone CICS на языке PL/I. Эта программа выводит список телефонных номеров сотрудников и обновляет его по запросу.

---

## Приложение С. Как запускать примеры программ DSNTIAUL, DSNTIAD и DSNTEP2

С DB2 поставляются три примера программ, которые могут помочь многим пользователям в работе. Эти программы поставляются в виде исходных текстов, и их можно изменять в соответствии со своими потребностями. Это программы:

- DSNTIAUL** Пример программы выгрузки. Эта программа на ассемблере выгружает часть строк или все строки из таблиц DB2 (обрабатываются до 100 таблиц за раз). При помощи DSNTIAUL можно выгрузить данные любого встроенного типа данных DB2 или пользовательского типа. Из столбца большого объекта можно выгрузить до 32 Кбайт данных. DSNTIAUL выгружает строки в формате, совместимом с утилитой LOAD, и генерирует управляющие операторы для утилиты LOAD. DSNTIAUL позволяет также выполнить любой допускающий динамическое выполнение оператор SQL, кроме SELECT.
- DSNTIAD** Пример программирования динамического SQL на ассемблере. С помощью этой программы можно выполнить любой допускающий динамическое выполнение оператор SQL, кроме SELECT.
- DSNTEP2** Пример программирования динамического SQL на языке PL/I. С помощью этой программы можно выполнить любой оператор SQL, который можно выполнять динамически. Можно воспользоваться исходным текстом DSNTEP2 и изменить его в соответствии со своими потребностями, если же у вас не установлен компилятор PL/I, можно взять объектный код программы DSNTEP2.

Поскольку эти три программы воспринимают еще и операторы статического SQL CONNECT, SET CONNECTION и RELEASE, с их помощью можно также обращаться к удаленным таблицам DB2.

DSNTIAUL и DSNTIAD поставляются только в виде исходного текста, так что до того, как их можно будет использовать, необходимо обработать их препроцессором, ассемблировать, скомпоновать и связать. Чтобы использовать исходный текст DSNTEP2, необходимо обработать его препроцессором, ассемблировать, скомпоновать и связать. Объектный код DSNTEP2 перед использованием необходимо связать. Обычно эти программы подготавливают системный администратор в процессе установки. В Табл. 124 указано, какое задание установки подготавливает каждый из примеров программ. Все задания установки находятся в наборе данных DSN610.SDSNSAMP.

Таблица 124 (Стр. 1 из 2). Задания, подготавливающие DSNTIAUL, DSNTIAD и DSNTEP2

Имя программы	Задание подготовки
DSNTIAUL	DSNTEJ2A
DSNTIAD	DSNTIJTM

Таблица 124 (Стр. 2 из 2). Задания, подготавливающие DSNTIAUL, DSNTIAD и DSNTEP2

Имя программы	Задание подготовки
DSNTEP2 (исходный текст)	DSNTEJ1P
DSNTEP2 (объектный код)	DSNTEJ1L

Для запуска примеров программ следует воспользоваться командой DSN RUN, подробно описанной в Главе 2 справочника *DB2 Command Reference*. В Табл. 125 перечислены загрузочные модули и планы, которые надо задать, а также параметры запуска каждой программы. В последующих разделах описываются значения всех параметров.

Таблица 125. Значения параметров DSN RUN для DSNTIAUL, DSNTIAD и DSNTEP2

Имя программы	Модуль загрузки	План	Параметры
DSNTIAUL	DSNTIAUL	DSNTIB61	SQL
DSNTIAD	DSNTIAD	DSNTIA61	RC0 SQLTERM(символ–огранич.)
DSNTEP2	DSNTEP2	DSNTEP61	ALIGN(MID) или ALIGN(LHS) MAXSEL( <i>n</i> ) NOMIXED или MIXED SQLTERM(символ–огранич.)

Далее в приложении содержится следующая информация о запуске каждой из программ:

- Описание входных параметров
- Наборы данных, которые необходимо разместить перед запуском программы
- Коды возврата из программы
- Примеры запуска

В Табл. 124 на стр. 889 показаны работающие примеры заданий для каждой из программ.

## Запуск DSNTIAUL

В этом разделе содержится информация, необходимая для запуска DSNTIAUL, включая параметры, наборы данных, коды возврата и примеры вызова.

**Параметры DSNTIAUL:** DSNTIAUL воспринимает один параметр – SQL. Если задан этот параметр, то входной набор данных содержит один или несколько законченных операторов SQL, каждый из которых заканчивается точкой с запятой. Во входной набор данных можно включить любой оператор SQL, который можно выполнять динамически. Кроме того, можно использовать операторы статического SQL CONNECT, SET CONNECTION или RELEASE. Максимальная длина оператора – 32765 байт. DSNTIAUL определяет, какие таблицы выгрузить, с помощью операторов SELECT и динамически выполняет все остальные операторы, кроме CONNECT, SET CONNECTION и RELEASE. Чтобы соединиться с удаленными таблицами, DSNTIAUL выполняет CONNECT, SET CONNECTION и RELEASE статически.

Если параметр SQL не указан, то входной набор данных должен содержать один или несколько односторонних операторов со следующим синтаксисом:  
имя таблицы или производной таблицы [WHERE условия]  
[ORDER BY столбцы]

Каждый входной оператор должен быть правильным оператором SQL SELECT с опущенным условием SELECT \* FROM и без закрывающей точки с запятой. DSNTIAUL генерирует оператор SELECT для каждого входного оператора, добавляя входную строку к SELECT \* FROM, и по полученному оператору определяет, какие таблицы следует выгрузить. В этом входном формате текст каждой спецификации таблицы может быть максимальной длины 72 байта и должен находиться на одной строке.

В обоих входных форматах можно указывать операторы SELECT, объединяющие две или несколько таблиц или выбирающие из таблицы отдельные столбцы. Если вы указываете столбцы, необходимо изменить генерируемый DSNTIAUL оператор LOAD.

#### **Наборы данных DSNTIAUL:**

##### **Набор данных Описание**

<b>SYSIN</b>	Входной набор данных. Информация о содержимом входных данных приводится в разделе <i>Параметры DSNTIAUL</i> .  На вход DSNTIAUL нельзя подавать комментарии.  Длина записи для входного набора данных должна быть не меньше 72 байтов. DSNTIAUL читает только первые 72 байта каждой записи.
<b>SYSPRINT</b>	Выходной набор данных. В этот набор данных DSNTIAUL записывает информационные сообщения и сообщения об ошибках.  Длина записи для набора данных SYSPRINT – 121 байт.
<b>SYSPUNCH</b>	Выходной набор данных. В этот набор данных DSNTIAUL записывает управляющие операторы утилиты LOAD.
<b>SYSREC<sup>n</sup></b>	Выходные наборы данных. Значение <i>п</i> принимает значения от 00 до 99. За один запуск DSNTIAUL может создать не более 100 выходных наборов данных. В каждый набор данных помещаются данные, выгружаемые при обработке DSNTIAUL оператора SELECT из входного набора данных. Таким образом, количество выходных наборов данных должно совпадать с количеством операторов SELECT (если задан параметр SQL) или спецификаций таблиц во входном наборе данных.

Все наборы данных должны быть определены как последовательные наборы данных. Можно указать длину записи и размер блока наборов данных SYSPUNCH и SYSREC<sup>n</sup>. Максимальная длина записи для наборов данных SYSPUNCH и SYSREC<sup>n</sup> – 32760 байтов.

#### **Коды возврата DSNTIAUL:**

**Код возврата Значение**

- 0** Успешное завершение работы.
- 4** Оператор SQL получил код предупреждения. Если оператор SQL был оператором SELECT, DB2 не производила соответствующей операции выгрузки.
- 8** Оператор SQL получил код ошибки. Если оператор SQL был оператором SELECT, DB2 не производила соответствующей операции выгрузки.
- 12** DSNTIAUL не смогла открыть набор данных, оператор SQL возвратил код серьезной ошибки (-8nn или -9nn) или произошла ошибка в подпрограмме форматирования сообщений SQL.

**Примеры вызова DSNTIAUL:** Предположим, что требуется выгрузить из таблицы проектов строки для отдела D01. Спецификацию таблицы можно уместить в одной строке, и никаких операторов, кроме SELECT, выполнять не надо, так что параметр SQL не нужен. Программа вызывается так:

```
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DSN)
  RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB61) -
    LIB('DSN610.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
//           UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//           VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//           UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//           VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN    DD *
  DSN8610.PROJ WHERE DEPTNO='D01'
```

*Рисунок 208. Вызов DSNTIAUL без параметра SQL*

Если требуется получить управляющие операторы утилиты LOAD для загрузки строк в таблицу, но выгружать строки не нужно, можно для имен наборов данных SYSRECnn задать DUMMY. Например, чтобы получить управляющие операторы утилиты для загрузки строк в таблицу отделов, DSNTIAUL вызывается так:

```

//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPPRT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DSN)
  RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB61) -
    LIB('DSN610.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DUMMY
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//           UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//           VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN   DD *
DSN8610.DEPT

```

*Рисунок 209. Вызов DSNTIAUL для получения управляемых операторов LOAD*

Теперь предположим, что DSNTIAUL должна делать еще и следующее:

- Выгрузить все строки из таблицы проектов
- Выгрузить из таблицы сотрудников строки сотрудников тех отделов, обозначения которых начинаются на D, и упорядочить выгруженные строки по номерам сотрудников
- Заблокировать обе таблицы в режиме совместного доступа перед тем, как выгружать их

Для выполнения этих действий надо указать при запуске DSNTIAUL параметр SQL. DSNTIAUL вызывается так:

```

//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPPRT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DSN)
  RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB61) PARMS('SQL') -
    LIB('DSN610.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
//           UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//           VOL=SER=SCR03
//SYSREC01 DD DSN=DSN8UNLD.SYSREC01,
//           UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
//           VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
//           UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
//           VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN   DD *
LOCK TABLE DSN8610.EMP IN SHARE MODE;
LOCK TABLE DSN8610.PROJ IN SHARE MODE;
SELECT * FROM DSN8610.PROJ;
SELECT * FROM DSN8610.EMP
  WHERE WORKDEPT LIKE 'D%'
  ORDER BY EMPNO;

```

*Рисунок 210. Вызов DSNTIAUL с параметром SQL*

## Запуск DSNTIAD

В этом разделе содержится информация, необходимая для запуска DSNTIAD, включая параметры, наборы данных, коды возврата и примеры вызова.

### Параметры DSNTIAD:

#### RC0

Если задан этот параметр, DSNTIAD закончит работу с кодом возврата 0, даже если программа обнаружит ошибки SQL. Если не указать RC0, DSNTIAD закончит работу с кодом возврата, который будет отражать серьезность возникших ошибок. Без RC0 DSNTIAD прекращает работу, если будет обнаружено более 10 ошибок SQL.

#### SQLTERM(символ–ограничитель)

Этот параметр задает символ, которым будет завершаться каждый оператор SQL. Можно указать любой специальный символ, кроме символов, перечисленных в Табл. 126. По умолчанию выбирается SQLTERM(:).

Таблица 126. Некорректные специальные символы для символа–ограничителя SQL

Имя	Символ	Шестнадцатеричное представление
пробел		X'40'
запятая	,	X'5E'
двойная кавычка	"	X'7F'
левая круглая скобка	(	X'4D'
правая круглая скобка	)	X'5D'
одинарная кавычка	'	X'7D'
подчеркивание	_	X'6D'

Точку с запятой нельзя использовать, если будет выполняться оператор, сам содержащий точку с запятой. Например, предположим, что задан параметр SQLTERM(#), чтобы сделать символом–ограничителем оператора символ #. Тогда оператор CREATE TRIGGER с вложенной точкой с запятой будет выглядеть следующим образом:

```
CREATE TRIGGER NEW_HIRe
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

При выборе символа–ограничителя оператора необходимо, чтобы этот символ не использовался в операторе.

### Наборы данных DSNTIAD:

## **Набор данных Описание**

<b>SYSIN</b>	Входной набор данных. В этом наборе данных может содержаться любое количество операторов SQL, кроме SELECT, каждый из которых должен заканчиваться точкой с запятой. Оператор можно перенести на следующую строку, но DSNTIAD читает только первые 72 байта каждой строки.  На вход DSNTIAD нельзя подавать комментарии.
<b>SYSPRINT</b>	Выходной набор данных. В этот набор данных DSNTIAD записывает информационные сообщения и сообщения об ошибках. Длина записи этого набора данных – 121 байт, размер блока – 1210 байтов.

Все наборы данных должны быть определены как последовательные наборы данных.

### **Коды возврата DSNTIAD:**

#### **Код возврата Значение**

- |           |   |
|-----------|---|
| <b>0</b>  | Успешное завершение работы, или же пользователь указал параметр RC0.  |
| <b>4</b>  | Оператор SQL получил код предупреждения.  |
| <b>8</b>  | Оператор SQL получил код ошибки.  |
| <b>12</b> | DSNTIAD не смогла открыть набор данных, длина оператора SQL превысила 32760 байтов, оператор SQL возвратил код серьезной ошибки (-8pp или -9pp) или произошла ошибка в подпрограмме форматирования сообщений SQL. |

**Пример вызова DSNTIAD:** Предположим, что требуется выполнить 20 операторов UPDATE и нужно, чтобы DSNTIAD продолжала работу даже после возникновения 10 ошибок. Программа вызывается так:

```
//RUNTIAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DSN)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA61) PARMS('RC0') -
    LIB('DSN610.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
  UPDATE DSN8610.PROJ SET DEPTNO='J01' WHERE DEPTNO='A01';
  UPDATE DSN8610.PROJ SET DEPTNO='J02' WHERE DEPTNO='A02';
  :
  UPDATE DSN8610.PROJ SET DEPTNO='J20' WHERE DEPTNO='A20';
```

*Рисунок 211. Вызов DSNTIAD с параметром RC0*

## Запуск DSNTEP2

В этом разделе содержится информация, необходимая для запуска DSNTEP2, включая параметры, наборы данных, коды возврата и примеры вызова.

### Параметры DSNTEP2:

Параметр	Описание
----------	----------

#### ALIGN(MID) или ALIGN(LHS)

Чтобы DSNTEP2 центрировала информацию при выводе, нужно задать ALIGN(MID). Чтобы выровнять вывод по левому краю, нужно задать ALIGN(LHS). По умолчанию выбирается ALIGN(MID).

#### MAXSEL(*n*)

Параметр MAXSEL(*n*) ограничивает количество строк, которое возвращает DSNTEP2 для оператора SELECT. *n* должно быть целым числом от 0 до 32768. Если не указать MAXSEL(*n*), DSNTEP2 возвратит все строки таблицы результатов.

#### NOMIXED или MIXED

Если входные данные DSNTEP2 содержат какие-либо символы DBCS, следует указать MIXED. Если во входных данных нет символов DBCS, следует указать NOMIXED. По умолчанию выбирается NOMIXED.

#### SQLTERM(символ-ограничитель)

Этот параметр задает символ, которым будет завершаться каждый оператор SQL. Можно указать любой символ, *кроме* символов, перечисленных в Табл. 126 на стр. 894. По умолчанию выбирается SQLTERM(;).

Точку с запятой нельзя использовать, если будет выполняться оператор, сам содержащий точку с запятой. Например, предположим, что задан параметр SQLTERM(#), чтобы сделать символом-ограничителем оператора символ #. Тогда оператор CREATE TRIGGER с вложенной точкой с запятой будет выглядеть следующим образом:

```
CREATE TRIGGER NEW_HIRe
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#
```

При выборе символа-ограничителя оператора необходимо, чтобы этот символ не использовался в операторе.

Если вы хотите заменить символ-ограничитель SQL в ряде операторов SQL, можно использовать управляющий оператор —#SET TERMINATOR. Предположим, например, что у вас есть существующий набор операторов SQL, к которому вы хотите добавить оператор CREATE TRIGGER, в который входит точка с запятой. Для всех существующих операторов SQL можно использовать значение SQLTERM по умолчанию (точка с запятой). Перед выполнением оператора CREATE TRIGGER включите управляющий оператор —#SET TERMINATOR #, чтобы заменить символ-ограничитель SQL на символ #:

```

SELECT * FROM DEPT;
SELECT * FROM ACT;
SELECT * FROM EMPPROJECT;
SELECT * FROM PROJ;
SELECT * FROM PROJECT;
--#SET TERMINATOR #
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END#

```

Дополнительная информация об управляющем операторе —#SET приводится в описании набора данных SYSIN.

### **Наборы данных DSNTEP2:**

#### **Набор данных Описание**

**SYSIN** Входной набор данных. В этом наборе данных может содержаться любое количество операторов SQL, каждый из которых должен заканчиваться точкой с запятой. Оператор может быть перенесен на следующую строку, но DSNTEP2 читает только первые 72 байта каждой строки.

На входе DSNTEP2 можно вводить комментарии с помощью звездочки (\*) в столбце 1 или двух дефисов (—) в любом месте строки. Текст после звездочки рассматривается как комментарий. Текст после двух дефисов может быть комментарием или управляющим оператором. Комментарии или управляющие операторы нельзя продолжать на следующей строке.

Во входном наборе данных DSNTEP2 можно ввести много управляющих операторов. Эти управляющие операторы имеют вид

```
--#SET управляемая-опция значение
```

Допустимы следующие управляющие опции:

#### **TERMINATOR**

Символ—ограничитель операторов SQL. Значением может быть любой однобайтный символ, кроме перечисленных в Табл. 126 на стр. 894. По умолчанию используется значение параметра SQLTERM.

#### **ROWS\_FETCH**

Число строк, выбираемых из таблицы результатов. Значением может быть числовая литерал от —1 до числа строк в таблице результатов. —1 означает, что выбираются все строки. По умолчанию предполагается —1.

#### **ROWS\_OUT**

Число выбранных строк, посыпаемых в выходной набор данных. Значением может быть числовая литерал от —1 до числа выбранных строк. —1 означает, что все выбранные строки посыпаются в выходной набор данных. По умолчанию предполагается —1.

**SYSPRINT** Выходной набор данных. В этот набор данных DSNTEP2 записывает информационные сообщения и сообщения об ошибках. DSNTEP2 выводит выходные записи длиной не более 133 байтов.

Все наборы данных должны быть определены как последовательные наборы данных.

**Коды возврата DSNTEP2:**

**Код возврата Значение**

- |           |  |
|-----------|--|
| <b>0</b>  | Успешное завершение работы.  |
| <b>4</b>  | Оператор SQL получил код предупреждения.   |
| <b>8</b>  | Оператор SQL получил код ошибки.   |
| <b>12</b> | Длина оператора SQL превысила 32 760 байтов, оператор SQL возвратил код серьезной ошибки (-8пп или -9пп) или произошла ошибка в подпрограмме форматирования сообщений SQL. |

**Пример вызова DSNTEP2:** Предположим, что требуется, чтобы DSNTEP2 выполняла операторы SQL SELECT, которые могут содержать символы DBCS. Требуется также выравнивать вывод по левому краю. Программа вызывается так:

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DSN)
  RUN PROGRAM(DSNTEP2) PLAN(DSNTEP61) PARMS('/ALIGN(LHS) MIXED') -
    LIB('DSN610.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
  SELECT * FROM DSN8610.PROJ;
```

*Рисунок 212. Вызов DSNTEP2 с параметрами ALIGN(LHS) и MIXED*

---

## Приложение D. Примеры программирования

В этом приложении приводятся следующие примеры программирования:

- Пример программы на языке COBOL с динамическим SQL
- “Пример динамического и статического SQL в программе на языке C” на стр. 913
- “Пример программы на языке COBOL с использованием доступа DRDA” на стр. 917
- “Пример программы на языке COBOL, использующей доступ по собственному протоколу DB2” на стр. 925
- “Примеры использования хранимых процедур” на стр. 932

Для подготовки и запуска этих примеров используйте в качестве модели ваших JCL задания JCL из DSN610.SDSNSAMP. В Приложение B, “Примеры прикладных программ” на стр. 883 приводится список процедур JCL для подготовки программ примеров. Сведения о требуемых опциях компилятора для каждого языка приводятся в разделе Раздел 2 *DB2 Installation Guide*.

---

### Пример программы на языке COBOL с динамическим SQL

В разделе “Глава 7–1. Кодирование динамического SQL в прикладных программах” на стр. 543 описывается три вида операторов динамического SQL:

- Все операторы, кроме операторов SELECT
- Операторы SELECT с постоянным списком

В этом случае при написании программы известно число возвращаемых столбцов и их типы данных.

- Операторы SELECT с переменным списком.

В этом случае при написании программы **не** известно число возвращаемых столбцов и их типы данных.

В этом приложении изложены методы кодирования операторов SELECT с переменным списком в языках VS COBOL II, COBOL/370 и IBM COBOL for MVS & VM. Термин *COBOL* далее относится только к указанным версиям.

| Программа примера не поддерживает типы данных BLOB, CLOB и DBCLOB.

### Указатели и базированные переменные

В языке COBOL существуют тип POINTER и оператор SET, которые позволяют использовать указатели и базированные переменные.

Оператор SET задает указатель по адресу области в разделе компоновки или по другому указателю строит указатель; оператор может также задать адрес области в разделе компоновки. Эта операция показана на рис. 214 на стр. 903. В операторе SET нельзя использовать адрес из раздела WORKING-STORAGE.

## **Выделение памяти**

В языке COBOL нет средств для выделения основной памяти в самой программе. Для этого можно использовать некоторую начальную программу, которая выделяет память, а затем вызывает вторую программу, выполняющую операции с указателем. (COBOL не позволяет непосредственно работать с указателем из-за высокой вероятности ошибок и аварийных остановок.)

Эта начальная программа очень проста. Она содержит раздел размещения рабочей памяти, который выделяет максимальное количество необходимой памяти. Затем программа вызывает вторую программу, передавая область или области в операторе CALL. Вторая программа определяет область в разделе компоновки и может затем использовать указатели внутри этой области.

Если требуется выделение отдельных частей памяти, лучше всего использовать индексы или массивы. Элементы массивов можно использовать в арифметических операциях или операциях сравнения.

## **Пример**

На рис. 213 на стр. 901 показан пример начальной программы, DSN8BCU1, выделяющей память и вызывающей вторую программу DSN8BCU2 (смотрите рис. 214 на стр. 903). DSN8BCU2 определяет затем переданные области памяти в своем разделе компоновки и использует в операторе PROCEDURE DIVISION условие USING.

Для некоторых манипуляций с указателями, которые нельзя выполнить непосредственно, можно, определив указатель, затем переопределить его как число. Например, длину столбца к указателю записи добавлять нельзя, но можно добавить эту величину к числовому значению, которое получается переопределением данного указателя.

```

* ПРИМЕР - ПАКЕТНАЯ ПРОГРАММА ВЫГРУЗКИ НА COBOL DSN8BCU1- DB2 ***
*
* ИМЯ МОДУЛЯ = DSN8BCU1
*
* ОПИСАНИЕ = ПРИМЕР ПРОГРАММЫ DB2
*             ПАКЕТНАЯ
*             ПРОГРАММА ВЫГРУЗКИ
*             VS COBOL II, COBOL/370 ИЛИ
*             IBM COBOL FOR MVS & VM
*
* ФУНКЦИЯ = ДАННЫЙ МОДУЛЬ ПРЕДОСТАВЛЯЕТ ПАМЯТЬ, КОТОРАЯ
*             ТРЕБУЕТСЯ DSN8BCU2, И ВЫЗЫВАЕТ ЭТУ ПРОГРАММУ
*
* ПРИМЕЧАНИЯ =
*             ЗАВИСИМОСТИ = ТРЕБУЕТСЯ VS COBOL II. ИСПОЛЬЗОВАНО
*                         НЕСКОЛЬКО НОВЫХ СРЕДСТВ.
*
* ОГРАНИЧЕНИЯ =
*             МАКСИМАЛЬНОЕ ЧИСЛО СТОЛБЦОВ - 750,
*             ОГРАНИЧЕНИЕ SQL.
*
*             ЗАПИСЬ ДАННЫХ ОГРАНИЧЕНА ОБЪЕМОМ 32700 БАЙТ,
*             ВКЛЮЧАЯ САМИ ДАННЫЕ, ДЛИНУ ДЛЯ ДАННЫХ VARCHAR
*             И ПРОБЕЛ ДЛЯ ИНДИКАТОРОВ NULL.
*
* ТИП МОДУЛЯ = ПРОГРАММА НА COBOL
*             ОБРАБОТКА = VS COBOL II, COBOL/370 ИЛИ
*                         IBM COBOL FOR MVS & VM
*             РАЗМЕР МОДУЛЯ = СМОТРИТЕ РЕДАКТИРОВАНИЕ СВЯЗЕЙ
*             АТРИБУТЫ = ПОВТОРНО-ВХОДИМАЯ
*
* ТОЧКА ВХОДА = DSN8BCU1
*             ЦЕЛЬ = СМОТРИТЕ ФУНКЦИЮ
*             КОМПОНОВКА = ВЫЗЫВАЕТСЯ ЗАПУСКОМ DSN
*             ВВОД = НЕТ
*             ВЫВОД = НЕТ
*
* EXIT-NORMAL = КОД ВОЗВРАТА 0 НОРМАЛЬНОЕ ВЫПОЛНЕНИЕ
*
* EXIT-ERROR =
*             КОД ВОЗВРАТА = НЕТ
*             КОДЫ АВАРИЙНОЙ ОСТАНОВКИ = НЕТ
*             СООБЩЕНИЯ ОБ ОШИБКЕ = НЕТ
*
* ВНЕШНИЕ ССЫЛКИ =
*             ПОДПРОГРАММ/СЛУЖБЫ =
*                         DSN8BCU2 - САМА ПРОГРАММА ВЫГРУЗКИ
*
*             ОБЛАСТИ ДАННЫХ           =     НЕТ
*             БЛОКИ УПРАВЛЕНИЯ         =     НЕТ
*
* ТАБЛИЦЫ = НЕТ
* ИЗМЕНЕНИЕ АКТИВНОСТИ = НЕТ

```

*Рисунок 213 (Часть 1 из 2). Начальная программа, выделяющая память*

```

*
*   *ПСЕВДОКОД*
*
*   ПРОЦЕДУРА
*   ВЫЗЫВАЕМ DSN8BCU2.
*   КОНЕЦ.
*-----
/
IDENTIFICATION DIVISION.
*-----
PROGRAM-ID.      DSN8BCU1
*
ENVIRONMENT DIVISION.
*
CONFIGURATION SECTION.
DATA DIVISION.
*
WORKING-STORAGE SECTION.
*
01 WORKAREA-IND.
    02 WORKIND PIC S9(4) COMP OCCURS 750 TIMES.
01 RECWORK.
    02 RECWORK-LEN PIC S9(8) COMP VALUE 32700.
    02 RECWORK-CHAR PIC X(1) OCCURS 32700 TIMES.
*
PROCEDURE DIVISION.
*
    CALL 'DSN8BCU2' USING WORKAREA-IND RECWORK.
    GOBACK.

```

*Рисунок 213 (Часть 2 из 2). Начальная программа, выделяющая память*

```

*** ПРИМЕР - ПАКЕТНАЯ ПРОГРАММА ВЫГРУЗКИ НА COBOL DSN8BCU2- DB2 *
*
* ИМЯ МОДУЛЯ = DSN8BCU2
*
* ОПИСАНИЕ      = ПРИКЛАДНАЯ ПРОГРАММА ПРИМЕРА DB2
*                  ПАКЕТНАЯ
*                  ПРОГРАММА ВЫГРУЗКИ
*                  VS COBOL II, COBOL/370 ИЛИ
*                  IBM COBOL FOR MVS & VM
*
* ФУНКЦИЯ = ДАННЫЙ МОДУЛЬ ВОСПРИНИМАЕТ ИМЯ ТАБЛИЦЫ
*           ИЛИ ПРОИЗВОДНОЙ ТАБЛИЦЫ
*           И ВЫГРУЖАЕТ ДАННЫЕ ИЗ ЭТОЙ ТАБЛИЦЫ
*           ИЛИ ПРОИЗВОДНОЙ ТАБЛИЦЫ.
* ЧИТАЕМ ИМЯ ТАБЛИЦЫ ИЗ SYSIN.
* ПОМЕЩАЕМ ДАННЫЕ ТАБЛИЦЫ В DD SYSREC01.
* ЗАПИСЫВАЕМ РЕЗУЛЬТАТЫ В SYSPRINT.
*
* ЗАМЕЧАНИЯ =
* ЗАВИСИМОСТИ = НЕЛЬЗЯ ИСПОЛЬЗОВАТЬ COBOL OS/VS.
*
* ОГРАНИЧЕНИЯ =
* РАЗМЕР SQLDA ОГРАНИЧЕН 33016 БАЙТАМИ.
* ЭТО НЕ ПОЗВОЛЯЕТ ИМЕТЬ ДЛЯ DB2 БОЛЕЕ
* 750 СТОЛБЦОВ.
*
* ЗАПИСЬ ДАННЫХ ОГРАНИЧЕНА ОБЪЕМОМ 32700 БАЙТ,
* ВКЛЮЧАЯ САМИ ДАННЫЕ, ДЛИНУ ДЛЯ ДАННЫХ VARCHAR
* И ПРОБЕЛ ДЛЯ ИНДИКАТОРОВ NULL.
*
* ПРОГРАММА ПРИНИМАЕТ ИМENA ТАБЛИЦ ИЛИ
* ПРОИЗВОДНЫХ ТАБЛИЦ, ПРИ КАЖДОМ ЗАПУСКЕ МОЖНО
* ЗАДАТЬ ОДНО ИМЯ.
*
* ТИП МОДУЛЯ = ПРОГРАММА НА ЯЗЫКЕ COBOL
* ОБРАБОТКА = ПРЕКОМПИЛЯТОР DB2
*             VS/COBOL II, COBOL/370, ИЛИ
*             IBM COBOL FOR MVS & VM
* РАЗМЕР МОДУЛЯ = СМОТРИТЕ РЕДАКТИРОВАНИЕ СВЯЗЕЙ
* АТРИБУТЫ = ПОВТОРНО-ВХОДИМАЯ
*
* ТОЧКА ВХОДА = DSN8BCU2
* ЦЕЛЬ = СМОТРИТЕ ФУНКЦИЮ
* КОМПОНОВКА =
*             ВЫЗОВ 'DSN8BCU2' С ПОМОЩЬЮ WORKAREA-IND RECWORK.
*
* ВВОД = ОБОЗНАЧЕНИЕ = WORKAREA-IND
*       ОПИСАНИЕ = МАССИВ ПЕРЕМЕННЫХ-ИНДИКАТОРОВ
*       01 WORKAREA-IND.
*             02 WORKIND PIC S9(4) COMP OCCURS 750 TIMES.
*
* ОБОЗНАЧЕНИЕ = RECWORK
* ОПИСАНИЕ = РАБОЧАЯ ОБЛАСТЬ ДЛЯ ВЫХОДНЫХ ЗАПИСЕЙ
* 01 RECWORK.
*             02 RECWORK-LEN PIC S9(8) COMP.
*
* ОБОЗНАЧЕНИЕ = SYSIN
* ОПИСАНИЕ = ВХОДНЫЕ ТРЕБОВАНИЯ - ТАБЛИЦА ИЛИ
*             ПРОИЗВОДНАЯ ТАБЛИЦА
*

```

*Рисунок 214 (Часть 1 из 10). Вызываемая программа, выполняющая действия с указателем*

```

*      ВЫВОД = ОБОЗНАЧЕНИЕ = SYSPRINT
*          ОПИСАНИЕ = РЕЗУЛЬТАТЫ НА ПЕЧАТЬ
*
*          ОБОЗНАЧЕНИЕ = SYSREC01
*          ОПИСАНИЕ = ВЫГРУЖЕННЫЕ ТАБЛИЧНЫЕ ДАННЫЕ
*
*      EXIT-NORMAL = КОД ВОЗВРАТА О НОРМАЛЬНОЕ ВЫПОЛНЕНИЕ
*      EXIT-ERROR =
*          КОД ВОЗВРАТА = НЕТ
*          КОДЫ АВАРИЙНОЙ ОСТАНОВКИ = НЕТ
*          СООБЩЕНИЯ ОБ ОШИБКЕ =
*              DSNT490I ПРОГРАММА ПРИМЕРА ВЫГРУЗКИ ДАННЫХ
*                  - НА COBOL, ВЕРСИЯ 3.0
*                  - ЭТО СООБЩЕНИЕ О НОРМАЛЬНОМ
*                  - ЗАПУСКЕ ДАННОЙ ПРОГРАММЫ.
*              DSNT493I ОШИБКА SQL, SQLCODE = NNNNNNNN
*                  - ОБНАРУЖЕНА ОШИБКА ИЛИ ПРЕДУПРЕЖДЕНИЕ SQL
*                  - ЗА ЭТИМ СООБЩЕНИЕМ СЛЕДУЕТ
*                  - ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ ИЗ DSNTIAR
*              DSNT495I УСПЕШНАЯ ВЫГРУЗКА XXXXXXXX СТРОК
*                  ТАБЛИЦЫ TTTTTTTT
*                      - ВЫГРУЗКА БЫЛА УСПЕШНОЙ. XXXXXXXX -
*                      - ЧИСЛО ВЫГРУЖЕННЫХ СТРОК. TTTTTTTT -
*                      - ИМЯ ТАБЛИЦЫ ИЛИ ПРОИЗВОДНОЙ ТАБЛИЦЫ,
*                      - ИЗ КОТОРОЙ ПРОИЗВЕДЕНА ВЫГРУЗКА.
*              DSNT496I НЕОПОЗНАННЫЙ КОД ТИПА ДАННЫХ NNNNN
*                  - ОПЕРАТОР PREPARE ВОЗВРАТИЛ КОД НЕВЕРНОГО
*                  - ТИПА ДАННЫХ. NNNNN - КОД
*                  - В ДЕСЯТИЧНОМ ФОРМАТЕ. ОБЫЧНО ОЗНАЧАЕТ
*                  - ОШИБКУ В ЭТОЙ ПОДПРОГРАММЕ
*                  - ИЛИ НОВЫЙ ТИП ДАННЫХ.
*              DSNT497I КОД ВОЗВРАТА ИЗ ПОДПРОГРАММЫ
*                  - СООБЩЕНИЙ DSNTIAR
*                  - ПОДПРОГРАММА ФОРМАТИРОВАНИЯ СООБЩЕНИЙ
*                  - ОБНАРУЖИЛА ОШИБКУ.
*                  - ПОСМОТРИТЕ ИНФОРМАЦИЮ
*                  - О КОДЕ ВОЗВРАТА,
*                  - СОДЕРЖАЩУЮСЬ В ПОДПРОГРАММЕ.
*                  - ОБЫЧНО ОЗНАЧАЕТ ОШИБКУ
*                  - В ЭТОЙ ПОДПРОГРАММЕ.
*              DSNT498I ОШИБКА, НЕ НАЙДЕНО ПРАВИЛЬНЫХ СТОЛБЦОВ
*                  - ОПЕРАТОР PREPARE ВОЗВРАТИЛ ДАННЫЕ,
*                  - КОТОРЫЕ НЕ ОБРАЗУЮТ
*                  - ПРАВИЛЬНОЙ ВЫХОДНОЙ ЗАПИСИ.
*                  - ОБЫЧНО ОЗНАЧАЕТ ОШИБКУ
*                  - В ЭТОЙ ПОДПРОГРАММЕ.
*              DSNT499I В ТАБЛИЦЕ ИЛИ ПРОИЗВОДНОЙ ТАБЛИЦЕ
*                  - НЕ ОБНАРУЖЕНО СТРОК
*                  - ВЫБРАННЫЕ ТАБЛИЦА
*                  - ИЛИ ПРОИЗВОДНЫЕ ТАБЛИЦЫ
*                  - НЕ ВОЗВРАТИЛИ СТРОК.
*          СООБЩЕНИЯ ОБ ОШИБКАХ ОТ МОДУЛЯ DSNTIAR
*              - ПРИ ПОЯВЛЕНИИ ОШИБКИ ДАННЫЙ МОДУЛЬ
*              - ВЫДАЕТ СООБЩЕНИЯ ОШИБКИ.
*
*          ВНЕШНИЕ ССЫЛКИ =
*          ПОДПРОГРАММЫ/СЛУЖБЫ =
*              DSNTIAR - ПРЕОБРАЗУЕТ SQLCA В СООБЩЕНИЯ
*          DATA-AREAS = НЕТ
*          БЛОКИ-УПРАВЛЕНИЯ =
*              SQLCA - ОБЛАСТЬ СВЯЗИ SQL
*
*          ТАБЛИЦЫ = НЕТ
*          ИЗМЕНЕНИЕ АКТИВНОСТИ = НЕТ

```

*Рисунок 214 (Часть 2 из 10). Вызываемая программа, выполняющая действия с указателем*

```

* *ПСЕВДОКОД*
* ПРОЦЕДУРА
* EXEC SQL DECLARE DT CURSOR FOR SEL END-EXEC.
* EXEC SQL DECLARE SEL STATEMENT END-EXEC.
* ИНИЦИАЛИЗИРУЕМ ДАННЫЕ, ОТКРЫВАЕМ ФАЙЛЫ.
* ПОЛУЧАЕМ ПАМЯТЬ ИЗ SQLDA И ЗАПИСЕЙ ДАННЫХ.
* ЧИТАЕМ ИМЯ ТАБЛИЦЫ.
* ОТКРЫВАЕМ SYSREC01.
* СТРОИМ ОПЕРАТОР SQL, КОТОРЫЙ НЕОБХОДИМО ВЫПОЛНИТЬ
* EXEC SQL PREPARE SQL STATEMENT INTO SQLDA END-EXEC.
* ЗАДАЕМ АДРЕСА В SQLDA ДЛЯ ДАННЫХ.
* ИНИЦИАЛИЗИРУЕМ СЧЕТЧИК ЗАПИСЕЙ ДАННЫХ НУЛЕМ.
* EXEC SQL OPEN DT END-EXEC.
* DO WHILE SQLCODE IS 0.
* EXEC SQL FETCH DT USING DESCRIPTOR SQLDA END-EXEC.
* ДОБАВЛЯЕМ МАРКЕРЫ ДЛЯ ОБОЗНАЧЕНИЯ ПУСТЫХ ЗАПИСЕЙ.
* ЗАПИСЫВАЕМ ДАННЫЕ В SYSREC01.
* УВЕЛИЧИВАЕМ СЧЕТЧИК ЗАПИСЕЙ ДАННЫХ.
* КОНЕЦ.
* EXEC SQL CLOSE DT END-EXEC.
* УКАЗЫВАЕМ РЕЗУЛЬТАТЫ ОПЕРАЦИИ ВЫГРУЗКИ.
* ЗАКРЫВАЕМ ФАЙЛЫ SYSIN, SYSPRINT И SYSREC01.
* КОНЕЦ.
*-----*/
/
IDENTIFICATION DIVISION.
*-----
PROGRAM-ID.      DSN8BCU2
*-----ENVIRONMENT DIVISION.*-----CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SYSIN
        ASSIGN TO DA-S-SYSIN.
    SELECT SYSPRINT
        ASSIGN TO UT-S-SYSPRINT.
    SELECT SYSREC01
        ASSIGN TO DA-S-SYSREC01.
*-----DATA DIVISION.*-----FILE SECTION.
FD      SYSIN
    RECORD CONTAINS 80 CHARACTERS
    BLOCK CONTAINS 0 RECORDS
    LABEL RECORDS ARE OMITTED
    RECORDING MODE IS F.
01 CARDREC          PIC X(80).
*-----FD SYSPRINT
    RECORD CONTAINS 120 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS MSGREC
    RECORDING MODE IS F.
01 MSGREC          PIC X(120).

```

*Рисунок 214 (Часть 3 из 10). Вызываемая программа, выполняющая действия с указателем*

```

*
FD  SYSREC01
    RECORD CONTAINS 5 TO 32704 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS REC01
    RECORDING MODE IS V.
01  REC01.
    02  REC01-LEN PIC S9(8) COMP.
    02  REC01-CHAR PIC X(1) OCCURS 1 TO 32700 TIMES
        DEPENDING ON REC01-LEN.
/
WORKING-STORAGE SECTION.
*
*****  

* СТРУКТУРА ДЛЯ ВВОДА *  

*****  

01  IOAREA.
    02  TNAME      PIC X(72).
    02  FILLER     PIC X(08).
01  STMTBUF.
    49  STMTLEN    PIC S9(4) COMP VALUE 92.
    49  STMTCHAR   PIC X(92).
01  STMTBLD.
    02  FILLER     PIC X(20) VALUE 'SELECT * FROM'.
    02  STMTTAB    PIC X(72).
*
*****  

* СТРУКТУРА ЗАГОЛОВКА ОТЧЕТА *  

*****  

01  HEADER.
    02  FILLER PIC X(35)
        VALUE ' DSNT490I SAMPLE COBOL DATA UNLOAD '.
    02  FILLER PIC X(85) VALUE 'PROGRAM RELEASE 3.0'.
01  MSG-SQLERR.
    02  FILLER PIC X(31)
        VALUE ' DSNT493I SQL ERROR, SQLCODE = '.
    02  MSG-MINUS   PIC X(1).
    02  MSG-PRINT-CODE PIC 9(8).
    02  FILLER PIC X(81) VALUE ' '.
01  UNLOADED.
    02  FILLER PIC X(28)
        VALUE ' DSNT495I SUCCESSFUL UNLOAD '.
    02  ROWS      PIC 9(8).
    02  FILLER PIC X(15) VALUE ' ROWS OF TABLE '.
    02  TABLENAM   PIC X(72) VALUE ' '.
01  BADTYPE.
    02  FILLER PIC X(42)
        VALUE ' DSNT496I UNRECOGNIZED DATA TYPE CODE OF '.
    02  TYPcod    PIC 9(8).
    02  FILLER PIC X(71) VALUE ' '.
01  MSGRETCD.
    02  FILLER PIC X(42)
        VALUE ' DSNT497I RETURN CODE FROM MESSAGE ROUTINE'.
    02  FILLER PIC X(9) VALUE 'DSNTIAR '.
    02  RETCODE    PIC 9(8).
    02  FILLER PIC X(62) VALUE ' '.

```

*Рисунок 214 (Часть 4 из 10). Вызываемая программа, выполняющая действия с указателем*

```

01 MSGNOCOL.
    02 FILLER PIC X(120)
        VALUE ' DSNT498I ERROR, NO VALID COLUMNS FOUND'.
01 MSG-NOROW.
    02 FILLER PIC X(120)
        VALUE ' DSNT499I NO ROWS FOUND IN TABLE OR VIEW'.
*****
* РАБОЧИЕ ОБЛАСТИ *
*****
77 NOT-FOUND      PIC S9(8) COMP VALUE +100.
*****
* ПЕРЕМЕННЫЕ ДЛЯ ФОРМАТИРОВАНИЯ СООБЩЕНИЙ ОБ ОШИБКАХ *
00
*****
01 ERROR-MESSAGE.
    02 ERROR-LEN  PIC S9(4)  COMP VALUE +960.
    02 ERROR-TEXT  PIC X(120) OCCURS 8 TIMES
        INDEXED BY ERROR-INDEX.
    77 ERROR-TEXT-LEN  PIC S9(8)  COMP VALUE +120.
*****
* ОБЛАСТЬ ДЕСКРИПТОРОВ SQL *
*****
EXEC SQL INCLUDE SQLDA END-EXEC.
*
* ТИПЫ ДАННЫХ ИЗ SQLTYPE ПОСЛЕ УДАЛЕНИЯ ПУСТОГО БИТА
*
77 VARCTYPE        PIC S9(4)  COMP VALUE +448.
77 CHARTYPE        PIC S9(4)  COMP VALUE +452.
77 VARLTYPE        PIC S9(4)  COMP VALUE +456.
77 VARGTYPE        PIC S9(4)  COMP VALUE +464.
77 GTYPE            PIC S9(4)  COMP VALUE +468.
77 LVARGTYP        PIC S9(4)  COMP VALUE +472.
77 FLOATTYPE       PIC S9(4)  COMP VALUE +480.
77 DECTYPE          PIC S9(4)  COMP VALUE +484.
77 INTTYPE          PIC S9(4)  COMP VALUE +496.
77 HWTYPE           PIC S9(4)  COMP VALUE +500.
77 DATETYP          PIC S9(4)  COMP VALUE +384.
77 TIMETYP          PIC S9(4)  COMP VALUE +388.
77 TIMESTMP         PIC S9(4)  COMP VALUE +392.
*

```

*Рисунок 214 (Часть 5 из 10). Вызываемая программа, выполняющая действия с указателем*

```

01 RECPTR POINTER.
01 RECNUM REDEFINES RECPTR PICTURE S9(8) COMPUTATIONAL.
01 IRECPTR POINTER.
01 IRECNUM REDEFINES IRECPTR PICTURE S9(8) COMPUTATIONAL.
01 I      PICTURE S9(4) COMPUTATIONAL.
01 J      PICTURE S9(4) COMPUTATIONAL.
01 DUMMY PICTURE S9(4) COMPUTATIONAL.
01 MYTYPE PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-IND PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-LEN PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-PREC PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-SCALE PICTURE S9(4) COMPUTATIONAL.
01 INDCOUNT          PIC S9(4) COMPUTATIONAL.
01 ROWCOUNT          PIC S9(4) COMPUTATIONAL.
01 WORKAREA2.

        02 WORKINDPTR POINTER OCCURS 750 TIMES.
*****
* ОБЪЯВЛЕНИЕ УКАЗАТЕЛЯ И ОПЕРАТОРА ДЛЯ ДИНАМИЧЕСКОГО SQL
*****
*
        EXEC SQL DECLARE DT CURSOR FOR SEL END-EXEC.
        EXEC SQL DECLARE SEL STATEMENT END-EXEC.

*
*****
* ВСТАВКА SQL ДЛЯ SQLCA *
*****
*
        EXEC SQL INCLUDE SQLCA END-EXEC.

*
77 ONE          PIC S9(4) COMP VALUE +1.
77 TWO          PIC S9(4) COMP VALUE +2.
77 FOUR         PIC S9(4) COMP VALUE +4.
77 QMARK        PIC X(1)      VALUE '?'.

*
LINKAGE SECTION.
01 LINKAREA-IND.
        02 IND PIC S9(4) COMP OCCURS 750 TIMES.
01 LINKAREA-REC.
        02 REC1-LEN PIC S9(8) COMP.
        02 REC1-CHAR PIC X(1) OCCURS 1 TO 32700 TIMES
            DEPENDING ON REC1-LEN.
01 LINKAREA-QMARK.
        02 INDREC PIC X(1).
/

```

*Рисунок 214 (Часть 6 из 10). Вызываемая программа, выполняющая действия с указателем*

```

PROCEDURE DIVISION USING LINKAREA-IND LINKAREA-REC.
*
***** *****
* ОБРАБОТКА КОДА ВОЗВРАТА SQL *
*****
EXEC SQL WHENEVER SQLERROR GOTO DBERROR END-EXEC.
EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
*
*****
* ОСНОВНАЯ ПОДПРОГРАММА *
*****
SET IRECPTR TO ADDRESS OF REC1-CHAR(1).
*                                     **ОТКРЫВАЕМ ФАЙЛЫ
OPEN INPUT SYSIN
      OUTPUT SYSPRINT
      OUTPUT SYSREC01.
*                                     **ЗАПИСЫВАЕМ ЗАГОЛОВОК
WRITE MSGREC FROM HEADER
      AFTER ADVANCING 2 LINES.
*                                     **ПОЛУЧАЕМ ПЕРВУЮ ВХОДНУЮ ЗАПИСЬ
READ SYSIN RECORD INTO IOAREA.
*                                     **ОСНОВНАЯ ПРОЦЕДУРА
PERFORM PROCESS-INPUT THROUGH IND-RESULT.
*
PROG-END.
*                                     **ЗАКРЫВАЕМ ФАЙЛЫ
CLOSE SYSIN
      SYSPRINT
      SYSREC01.
GOBACK.
/
*****
*      РАЗДЕЛ ВЫПОЛНЕНИЯ:
*      ОБРАБОТКА ТОЛЬКО ЧТО ПРОЧИТАННОЙ ТАБЛИЦЫ
*      ИЛИ ПРОИЗВОДНОЙ ТАБЛИЦЫ
*
*****
PROCESS-INPUT.
*
MOVE TNAME TO STMTTAB.
MOVE STMTBLD TO STMTCHAR.
EXEC SQL PREPARE SEL INTO :SQLDA FROM :STMTBUF END-EXEC.
*****
*                                     *
*      ЗАДАЕМ АДРЕСА ДАННЫХ В SQLDA. *
*
*****
IF SQLD = ZERO THEN
    WRITE MSGREC FROM MSGNOCOL
        AFTER ADVANCING 2 LINES
    GO TO IND-RESULT.
MOVE ZERO TO ROWCOUNT.
MOVE ZERO TO REC1-LEN.
SET RECPTR TO IRECPTR.
MOVE ONE TO I.
PERFORM COLADDR UNTIL I > SQLD.

```

*Рисунок 214 (Часть 7 из 10). Вызываемая программа, выполняющая действия с указателем*

```

*****
*
* ЗАДАЕМ ДЛИНУ ЗАПИСИ ВЫВОДА. *
* EXEC SQL OPEN DT END-EXEC. *
* DO WHILE SQLCODE IS 0. *
*   EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC. *
*     ВВОДИМ МАРКЕРЫ ДЛЯ ОБОЗНАЧЕНИЯ ПУСТЫХ ЗАПИСЕЙ. *
*     ЗАПИСЫВАЕМ ДАННЫЕ В SYSREC01. *
*     УВЕЛИЧИВАЕМ СЧЕТЧИК ЗАПИСЕЙ ДАННЫХ. *
*   КОНЕЦ. *
*
*****                                     **ОТКРЫВАЕМ УКАЗАТЕЛЬ
*
* EXEC SQL OPEN DT END-EXEC.
* PERFORM BLANK-REC.
* EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC.
*                                         **СТРОК НЕ НАЙДЕНО
*                                         **ВЫВОДИМ СООБЩЕНИЕ ОБ ОШИБКЕ
*
* IF SQLCODE = NOT-FOUND
*   WRITE MSGREC FROM MSG-NOROW
*     AFTER ADVANCING 2 LINES
* ELSE
*   **ЗАПИСЫВАЕМ СТРОКУ И
*   **ПРОДОЛЖАЕМ, ПОКА
*   **СТРОКИ НЕ КОНЧАТСЯ
*   PERFORM WRITE-AND-FETCH
*     UNTIL SQLCODE IS NOT EQUAL TO ZERO.
*
* EXEC SQL WHENEVER NOT FOUND GOTO CLOSEDT    END-EXEC.
*
CLOSEDT.
* EXEC SQL CLOSE DT END-EXEC.
*
*****                                     * УКАЗЫВАЕМ РЕЗУЛЬТАТЫ ОПЕРАЦИИ ВЫГРУЗКИ. *
*
*****                                     IND-RESULT.
MOVE TNAME TO TABLENAM.
MOVE ROWCOUNT TO ROWS.
WRITE MSGREC FROM UNLOADED
  AFTER ADVANCING 2 LINES.
GO TO PROG-END.
*
WRITE-AND-FETCH.
* ADD IN MARKERS TO DENOTE NULLS.
MOVE ONE TO INDCOUNT.
PERFORM NULLCHK UNTIL INDCOUNT = SQLD.
MOVE REC1-LEN TO REC01-LEN.
WRITE REC01 FROM LINKAREA-REC.
ADD ONE TO ROWCOUNT.
PERFORM BLANK-REC.
EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC.
*
NULLCHK.
IF IND(INDCOUNT) < 0 THEN
  SET ADDRESS OF LINKAREA-QMARK TO WORKINDPTR(INDCOUNT)
  MOVE QMARK TO INDREC.
  ADD ONE TO INDCOUNT.

```

*Рисунок 214 (Часть 8 из 10). Вызываемая программа, выполняющая действия с указателем*

```

*****
*   СНАЧАЛА ЗАНОСИМ В ЗАПИСЬ ПРОБЕЛЫ
*****
BLANK-REC.
    MOVE ONE TO J.
    PERFORM BLANK-MORE UNTIL J > REC1-LEN.
BLANK-MORE.
    MOVE ' ' TO REC1-CHAR(J).
    ADD ONE TO J.
*
COLADDR.
    SET SQLDATA(I) TO RECPTR.
*****
*
*   ОПРЕДЕЛЯЕМ ДЛИНУ ЭТОГО СТОЛБЦА (COLUMN-LEN)
*   ЭТО ЗАВИСИТ ОТ ТИПА ДАННЫХ. В БОЛЬШИНСТВЕ ТИПОВ ДАННЫХ
*   ДЛИНА ЗАДАНА, НО ДЛЯ ДАННЫХ VARCHAR, GRAPHIC, VARGRAPHIC
*   И DECIMAL ТРЕБУЕТСЯ РАСЧЕТ.
*   ДЛЯ УПРОЩЕНИЯ АТРИБУТ NULL СЛЕДУЕТ ОТДЕЛИТЬ.
*
MOVE SQLLEN(I) TO COLUMN-LEN.
*   COLUMN-IND - 0 ДЛЯ NO NULLS И 1 ДЛЯ NULLS
DIVIDE SQLTYPE(I) BY TWO GIVING DUMMY REMAINDER COLUMN-IND.
*   MYTYPE СОВПАДАЕТ С SQLTYPE, НО НЕ СОДЕРЖИТ ПУСТОГО БИТА
MOVE SQLTYPE(I) TO MYTYPE.
SUBTRACT COLUMN-IND FROM MYTYPE.
*   ЗАДАЕМ ДЛИНУ СТОЛБЦА В ЗАВИСИМОСТИ ОТ ТИПА ДАННЫХ
EVALUATE MYTYPE
    WHEN CHARTYPE CONTINUE,
    WHEN DATETYP CONTINUE,
    WHEN TIMETYP CONTINUE,
    WHEN TIMESTAMP CONTINUE,
    WHEN FLOATYPE CONTINUE,
    WHEN VARCTYPE
        ADD TWO TO COLUMN-LEN,
        VARLTYPE
        ADD TWO TO COLUMN-LEN,
        GTYPE
        MULTIPLY COLUMN-LEN BY TWO GIVING COLUMN-LEN,
        VARGTYPE
        PERFORM CALC-VARG-LEN,
        LVARGTYP
        PERFORM CALC-VARG-LEN,
        WHEN HWTYP
        MOVE TWO TO COLUMN-LEN,
        INTTYPE
        MOVE FOUR TO COLUMN-LEN,
        DECTYPE
        PERFORM CALC-DECIMAL-LEN,
        OTHER
        PERFORM UNRECOGNIZED-ERROR,
END-EVALUATE.
ADD COLUMN-LEN TO RECNUM.
ADD COLUMN-LEN TO REC1-LEN.

```

*Рисунок 214 (Часть 9 из 10). Вызываемая программа, выполняющая действия с указателем*

```

*****
*      ЕСЛИ ЭТОТ СТОЛБЕЦ МОЖЕТ БЫТЬ ПУСТЫМ, ТРЕБУЕТСЯ      *
*      ПЕРЕМЕННАЯ-ИНДИКАТОР. В ВЫХОДНОЙ ЗАПИСИ ПРЕДУСМОТРЕНО   *
*      СПЕЦИАЛЬНОЕ МЕСТО, ЧТОБЫ ОТМЕТИТЬ ПУСТОЕ ЗНАЧЕНИЕ      *
*****
MOVE ZERO TO IND(I).
IF COLUMN-IND = ONE THEN
    SET SQLIND(I) TO ADDRESS OF IND(I)
    SET WORKINDPTR(I) TO RECPTR
    ADD ONE TO RECNUM
    ADD ONE TO REC1-LEN.
*
    ADD ONE TO I.
*      РАСЧЕТ ДЛИНЫ СТОЛБЦА
*      ДЛЯ СТОЛБЦА С ДАННЫМИ ДЕСЯТИЧНОГО ТИПА
CALC-DECIMAL-LEN.
    DIVIDE COLUMN-LEN BY 256 GIVING COLUMN-PREC
        REMAINDER COLUMN-SCALE.

    MOVE COLUMN-PREC TO COLUMN-LEN.
    ADD ONE TO COLUMN-LEN.
    DIVIDE COLUMN-LEN BY TWO GIVING COLUMN-LEN.
*      РАСЧЕТ ДЛИНЫ СТОЛБЦА
*      ДАННЫХ ТИПА VARGRAPHIC
CALC-VARG-LEN.
    MULTIPLY COLUMN-LEN BY TWO GIVING COLUMN-LEN.
    ADD TWO TO COLUMN-LEN.
*      СТОЛБЕЦ ДАННЫХ
*      НЕИЗВЕСТНОГО ТИПА
UNRECOGNIZED-ERROR.
*
*      СООБЩЕНИЕ ОБ ОШИБКЕ ДЛЯ ДАННЫХ НЕИЗВЕСТНОГО ТИПА
*
MOVE SQLTYPE(I) TO TYPSCOD.
    WRITE MSGREC FROM BADTYPE
        AFTER ADVANCING 2 LINES.
    GO TO IND-RESULT.
*
*****
* ОШИБКА SQL - ПОЛУЧАЕМ СООБЩЕНИЕ      *
*****
DBERROR.
*
    MOVE SQLCODE TO MSG-PRINT-CODE.
    IF SQLCODE < 0 THEN MOVE '-' TO MSG-MINUS.
    WRITE MSGREC FROM MSG-SQLERR
        AFTER ADVANCING 2 LINES.
    CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
    IF RETURN-CODE = ZERO
        PERFORM ERROR-PRINT VARYING ERROR-INDEX
            FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 8
    ELSE
        MOVE RETURN-CODE TO RETCODE
        WRITE MSGREC FROM MSGRETCD
            AFTER ADVANCING 2 LINES.
    GO TO PROG-END.
*
*****
*      ВЫВОДИМ ТЕКСТ СООБЩЕНИЯ      *
*****
ERROR-PRINT.
    WRITE MSGREC FROM ERROR-TEXT (ERROR-INDEX)
        AFTER ADVANCING 1 LINE.

```

*Рисунок 214 (Часть 10 из 10). Вызываемая программа, выполняющая действия с указателем*

---

## Пример динамического и статического SQL в программе на языке С

На рис. 215 показан пример использования динамического и статического SQL в программе на языке С. Каждый из разделов программы обозначен комментарием. В разделе 1 программы показан статический SQL; в разделах 2, 3 и 4 показан динамический SQL. Функция каждого из разделов подробно объясняется во введении к программе.

```
/********************************************/  
/* Описательное имя = Пример динамического SQL с использованием */  
/* языка С */  
/*  
/* Функция = Показать примеры использования динамического и */  
/* статического SQL. */  
/*  
/* Примечания = В этом примере предполагается, что определены */  
/* таблицы EMP и DEPT. Это не обязательно те же */  
/* таблицы, что в примере таблиц DB2. */  
/*  
/* Тип модуля      = программа на языке С */  
/* Обработка       = препроцессор DB2, компилятор С */  
/* Размер модуля   = смотрите компоновку-связывание */  
/* Атрибуты        = не повторно-входимая, не повторно-используемая */  
/*  
/* Ввод      = */  
/*  
/*          обозначение/имя = DEPT */  
/*          описание = произвольная таблица */  
/*          обозначение/имя = EMP */  
/*          описание = произвольная таблица */  
/*  
/* Выход      = */  
/*  
/*          обозначение/имя = SYSPRINT */  
/*          описание = печать результатов через printf */  
/*  
/* Нормальный выход = код возврата 0 при нормальном завершении */  
/*  
/* Выход по ошибке = */  
/*  
/* Код возврата   = SQLCA */  
/*  
/* Аварийные коды = нет */  
/*  
/* Внешние ссылки = нет */  
/*  
/* Управляющие блоки = */  
/*          SQLCA - область связи sq1 */  
/*
```

Рисунок 215 (Часть 1 из 4). Пример SQL в программе на С

```

/* Описание логической структуры: */  

/* */  

/* Есть четыре раздела SQL. */  

/* */  

/* 1) Статический SQL 1: использование статического указателя */  

/* с оператором SELECT. Две выходных переменных хоста. */  

/* 2) Динамический SQL 2: SELECT с фиксированным списком, */  

/* использование того же самого оператора SELECT, что и */  

/* в SQL 1, чтобы показать разницу. */  

/* Подготовленная строка :iptstr может быть назначена вместе */  

/* с другими операторами SQL, пригодными для динамического */  

/* выполнения. */  

/* 3) Динамический SQL 3: Вставка с маркерами параметров. */  

/* Используются четыре маркера параметров, представляющие */  

/* четыре выходных переменных хоста в пределах структуры хоста */  

/* 4) Динамический SQL 4: EXECUTE IMMEDIATE */  

/* Оператор GRANT выполняется немедленно путем передачи его DB2 */  

/* через переменную хоста - строку переменной длины. Этот пример */  

/* показывает, как задавать переменную хоста перед ее передачей. */  

/* */  

/*******/  

#include "stdio.h"  

#include "stdefs.h"  

EXEC SQL INCLUDE SQLCA;  

EXEC SQL INCLUDE SQLDA;  

EXEC SQL BEGIN DECLARE SECTION;  

    short edlevel;  

        struct { short len;  

            char x1[56];  

        } stmtbf1, stmtbf2, inpstr;  

        struct { short len;  

            char x1[15];  

        } lname;  

    short hv1;  

    struct { char deptno[4];  

        struct { short len;  

            char x[36];  

        } deptname;  

        char mgrno[7];  

        char admrdept[4];  

    } hv2;  

    short ind[4];  

EXEC SQL END DECLARE SECTION;  

EXEC SQL DECLARE EMP TABLE  

    (EMPNO      CHAR(6)      ,  

     FIRSTNAME   VARCHAR(12)  ,  

     MIDINIT    CHAR(1)      ,  

     LASTNAME   VARCHAR(15)  ,  

     WORKDEPT   CHAR(3)      ,  

     PHONENO    CHAR(4)      ,  

     HIREDATE   DECIMAL(6)   ,  

     JOBCODE    DECIMAL(3)   ,  

     EDLEVEL    SMALLINT    ,  

     SEX        CHAR(1)      ,  

     BIRTHDATE   DECIMAL(6)   ,  

     SALARY     DECIMAL(8,2)  ,  

     FORFNAME   VARGRAPHIC(12),  

     FORMNAME   GRAPHIC(1)   ,  

     FORLNAME   VARGRAPHIC(15),  

     FORADDR    VARGRAPHIC(256) );

```

Рисунок 215 (Часть 2 из 4). Пример SQL в программе на C

```

EXEC SQL DECLARE DEPT TABLE
(
    DEPTNO      CHAR(3)      ,
    DEPTNAME    VARCHAR(36)   ,
    MGRNO       CHAR(6)      ,
    ADMRDEPT   CHAR(3)      );
main ()
{
printf("??/n***      начало программы                      ***");
EXEC SQL WHENEVER SQLERROR GO TO HANDLERR;
EXEC SQL WHENEVER SQLWARNING GO TO HANDWARN;
EXEC SQL WHENEVER NOT FOUND GO TO NOTFOUND;
/*****************************************/
/* Присвоение значений переменным хоста,          */
/* которые будут входными для DB2                  */
/*****************************************/
strcpy(hv2.deptno,"M92");
strcpy(hv2.deptname.x,"DDL");
hv2.deptname.len = strlen(hv2.deptname.x);
strcpy(hv2.mgrno,"123456");
strcpy(hv2.admrdept,"abc");
/*****************************************/
/* Статический SQL 1:  DECLARE CURSOR, OPEN, FETCH, CLOSE      */
/* Выбор в :edlevel, :lname                                */
/*****************************************/
printf("??/n***      начало объявления                      ***");
EXEC SQL DECLARE C1 CURSOR FOR SELECT EDLEVEL, LASTNAME FROM EMP
    WHERE EMPNO = '000010';
printf("??/n***      начало открытия                      ***");
EXEC SQL OPEN C1;

printf("??/n***      начало чтения                          ***");
EXEC SQL FETCH C1 INTO :edlevel, :lname;
printf("??/n***  возвращенные значения                      ***");
printf("??/n?/?nedlevel = %d",edlevel);
printf("??/nlname = %s\n",lname.x1);

printf("??/n***      начало закрытия                      ***");
EXEC SQL CLOSE C1;
/*****************************************/
/* Динамический SQL 2: PREPARE, DECLARE CURSOR, OPEN, FETCH, CLOSE */
/* Выбор в :edlevel, :lname                                */
/*****************************************/
sprintf (inpstr.x1,
    "SELECT EDLEVEL, LASTNAME FROM EMP WHERE EMPNO = '000010'");
inpstr.len = strlen(inpstr.x1);
printf("??/n***      начало подготовки                      ***");
EXEC SQL PREPARE STAT1 FROM :inpstr;
printf("??/n***      начало объявления                      ***");
EXEC SQL DECLARE C2 CURSOR FOR STAT1;
printf("??/n***      начало открытия                      ***");
EXEC SQL OPEN C2;

printf("??/n***      начало чтения                          ***");
EXEC SQL FETCH C2 INTO :edlevel, :lname;
printf("??/n***  возвращенные значения                      ***");
printf("??/n?/?nedlevel = %d",edlevel);
printf("??/nlname = %s\n",lname.x1);

printf("??/n***      начало закрытия                      ***");
EXEC SQL CLOSE C2;

```

*Рисунок 215 (Часть 3 из 4). Пример SQL в программе на C*

```

/*****
/* Динамический SQL 3: PREPARE с маркерами параметров      */
/* Вставка четырех значений.                                */
/*****
sprintf (stmtbf1.x1,
        "INSERT INTO DEPT VALUES (?,?,?,?,?)");
stmtbf1.len = strlen(stmtbf1.x1);
printf("??/n***      начало подготовки          ***");
EXEC SQL PREPARE s1 FROM :stmtbf1;
printf("??/n***      начало выполнения          ***");
EXEC SQL EXECUTE s1 USING :hv2:ind;
printf("??/n***      далее предполагается вставка результатов ***");
printf("??/n  hv2.deptno = %s",hv2.deptno);
printf("??/n  hv2.deptname.len = %d",hv2.deptname.len);
printf("??/n  hv2.deptname.x = %s",hv2.deptname.x);
printf("??/n  hv2.mgrno = %s",hv2.mgrno);
printf("??/n  hv2.admrdept = %s",hv2.admrdept);
EXEC SQL COMMIT;
/*****
/* Динамический SQL 4: EXECUTE IMMEDIATE           */
/* оператора GRANT SELECT                         */
/*****
sprintf (stmtbf2.x1,
        "GRANT SELECT ON EMP TO USERX");
stmtbf2.len = strlen(stmtbf2.x1);
printf("??/n***      начало немедленного выполнения      ***");
EXEC SQL EXECUTE IMMEDIATE :stmtbf2;
printf("??/n***      конец программы          ***");
goto progend;
HANDWARN: HANDLERR: NOTFOUND: ;
printf("??/n  SQLCODE   = %d",SQLCODE);
printf("??/n  SQLWARN0 = %c",SQLWARN0);
printf("??/n  SQLWARN1 = %c",SQLWARN1);
printf("??/n  SQLWARN2 = %c",SQLWARN2);
printf("??/n  SQLWARN3 = %c",SQLWARN3);
printf("??/n  SQLWARN4 = %c",SQLWARN4);
printf("??/n  SQLWARN5 = %c",SQLWARN5);
printf("??/n  SQLWARN6 = %c",SQLWARN6);
printf("??/n  SQLWARN7 = %c",SQLWARN7);
progend: ;
}

```

*Рисунок 215 (Часть 4 из 4). Пример SQL в программе на C*

---

## Пример программы на языке COBOL с использованием доступа DRDA

В этом примере показан доступ к распределенным данным с использование DRDA.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. TWOPHASE.  
AUTHOR.  
REMARKS.  
*****  
* *  
* ИМЯ МОДУЛЯ = TWOPHASE *  
* *  
* ОПИСАНИЕ = ПРИМЕР ПРОГРАММЫ DB2, ИСПОЛЬЗУЮЩЕЙ *  
* ДВУХФАЗНОЕ ПРИНЯТИЕ И МЕТОД *  
* РАСПРЕДЕЛЕННОГО ДОСТУПА DRDA *  
* *  
* COPYRIGHT = 5665-DB2 (C) АВТОРСКИЕ ПРАВА IBM CORP 1982, 1989 *  
* СМОТРИТЕ ИНСТРУКЦИЮ ОБ АВТОРСКИХ ПРАВАХ, ФОРМА НОМЕР G120-2083*  
* *  
* СОСТОЯНИЕ = ВЕРСИЯ 5 *  
* *  
* ФУНКЦИЯ = ЭТΟТ МОДУЛЬ ИЛЛЮСТРИРУЕТ РАСПРЕДЕЛЕННЫЙ ДОСТУП К *  
* ДАННЫМ С ИСПОЛЬЗОВАНИЕМ ДВУХФАЗНОГО ПРИНЯТИЯ, *  
* ПЕРЕВОДЯ СОТРУДНИКА С ОДНОГО МЕСТА НА ДРУГОЕ. *  
* *  
* ПРИМЕЧАНИЕ: ЭТА ПРОГРАММА ПРЕДПОЛАГАЕТ, ЧТО ТАБЛИЦА*  
* SYSADM.EMP РАСПОЛОЖЕНА В STLEC1 И В STLEC2. *  
* *  
* ТИП МОДУЛЯ = ПРОГРАММА НА ЯЗЫКЕ COBOL *  
* ОБРАБОТКА = ПРЕКОМПИЛЯТОР DB2, VS COBOL II *  
* РАЗМЕР МОДУЛЯ = СМОТРИТЕ РЕДАКТИРОВАНИЕ СВЯЗЕЙ *  
* АТРИБУТЫ = НЕ ПОВТОРНО-ВХОДИМАЯ, НЕ ПОВТОРНО-ИСПОЛЬЗУЕМАЯ *  
* *  
* ТОЧКА ВХОДА = *  
* ЦЕЛЬ = ПРОИЛЛЮСТРИРОВАТЬ ДВУХФАЗНОЕ ПРИНЯТИЕ *  
* КОМПОНОВКА = ВЫЗОВ ИЗ DSN RUN *  
* ВВОД = НЕТ *  
* ВЫВОД = *  
* ОБОЗНАЧЕНИЕ/ИМЯ = SYSPRINT *  
* ОПИСАНИЕ = ПЕЧАТАЕТ ОПИСАНИЕ КАЖДОГО ШАГА И *  
* ПОЛУЧЕННЮЮ SQLCA *  
* *  
* НОРМАЛЬНЫЙ ВЫХОД = КОД ВОЗВРАТА 0 ПРИ НОРМАЛЬНОМ ЗАВЕРШЕНИИ *  
* *  
* АВАРИЙНЫЙ ВЫХОД = НЕТ *  
* *  
* ВНЕШНИЕ ССЫЛКИ = *  
* ПРОГРАММНЫЕ СЛУЖБЫ = НЕТ *  
* ОБЛАСТИ ДАННЫХ = НЕТ *  
* УПРАВЛЯЮЩИЕ БЛОКИ = *  
* SQLCA - ОБЛАСТЬ СВЯЗИ SQL *  
* *  
* ТАБЛИЦЫ = НЕТ *  
* *  
* ИЗМЕНЕНИЕ АКТИВНОСТИ = НЕТ *  
* *  
* *
```

Рисунок 216 (Часть 1 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA

```

* ПСЕВДОКОД
*
*      MAINLINE.
*          Выполнить CONNECT-TO-SITE-1, чтобы установить
*          соединение с локальным узлом.
*          If предыдущая операция выполнена успешно, Then
*              Do.
*                  Выполнить PROCESS-CURSOR-SITE-1 чтобы
*                  получить информацию о сотруднике, который
*                  переводится в другое место.
*                  If информация о сотруднике успешно получена, Then
*                      Do.
*                          Выполнить UPDATE-ADDRESS, чтобы изменить
*                          текущую информацию об этом
*                          сотруднике.
*                          Выполнить CONNECT-TO-SITE-2, чтобы
*                          установить соединение с узлом, куда
*                          переводится сотрудник.
*                          Выполнить CONNECT-TO-SITE-2, чтобы
*                          установить соединение с узлом, куда
*                          переводится сотрудник.
*                          If соединение успешно установлено, Then
*                              Do
*                                  Выполнить PROCESS-SITE-2, чтобы
*                                  записать информацию о сотруднике
*                                  в место, куда он переводится.
*                              End If соединение успешно установлено.
*                          End If информация о сотруднике успешно получена.
*                      End If предыдущая операция выполнена успешно.
*                  Выполнить COMMIT-WORK, чтобы ПРИНЯТЬ изменения,
*                  внесенные в STLEC1 и в STLEC2.
*
*      PROG-END.
*          Закрыть принтер.
*          Return.
*
*      CONNECT-TO-SITE-1.
*          Дать текстовое описание следующего шага.
*          Установить соединение с местом, откуда переводится
*          сотрудник.
*          Напечатать выходную SQLCA.
*
*      PROCESS-CURSOR-SITE-1.
*          Дать текстовое описание следующего шага.
*          Дать текстовое описание следующего шага.
*          получения информации о сотруднике, переводящемся с
*          этого места.
*          Напечатать выходную SQLCA.
*          If указатель успешно открыт, Then
*              Do
*                  Do.
*                      удалить информацию о сотруднике, переводящемся с
*                      этого места.
*                      Выполнить CLOSE-CURSOR-SITE-1, чтобы закрыть
*                      указатель.
*                  End If указатель успешно открыт.
*

```

*Рисунок 216 (Часть 2 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA*

```

*      FETCH-DELETE-SITE-1. *
*      Дать текстовое описание следующего шага. *
*      Взять информацию о переводимом сотруднике. *
*      Напечатать выходную SQLCA. *
*      If информация успешно получена, Then *
*          Do *
*              | Выполнить DELETE-SITE-1, чтобы удалить сотрудника с *
*              | этого места. *
*          End If информация успешно получена. *
*      *
*      DELETE-SITE-1. *
*      Дать текстовое описание следующего шага. *
*      Дать текстовое описание следующего шага. *
*      Удалить информацию о переводимом сотруднике с этого места*
*      Напечатать выходную SQLCA. *
*      *
*      CLOSE-CURSOR-SITE-1. *
*      Дать текстовое описание следующего шага. *
*      Закрыть указатель, использовавшийся для получения *
*          информации о переводимом сотруднике. *
*      Напечатать выходную SQLCA. *
*      *
*      UPDATE-ADDRESS. *
*      Обновить адрес сотрудника. *
*      Обновить город сотрудника. *
*      Обновить местоположение сотрудника. *
*      *
*      CONNECT-TO-SITE-2. *
*      Дать текстовое описание следующего шага. *
*      Установить соединение с местом, куда переводится *
*          сотрудник. *
*      Напечатать выходную SQLCA. *
*      *
*      PROCESS-SITE-2. *
*      Дать текстовое описание следующего шага. *
*      Вставить информацию о сотруднике в место, куда он *
*          переводится. *
*      Напечатать выходную SQLCA. *
*      *
*      COMMIT-WORK. *
*      ПРИНЯТЬ все изменения, сделанные в STLEC1 и в STLEC2. *
*      *
*****
```

```

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PRINTER, ASSIGN TO S-OUT1.

DATA DIVISION.
FILE SECTION.
FD  PRINTER
    RECORD CONTAINS 120 CHARACTERS
    DATA RECORD IS PRT-TC-RESULTS
    LABEL RECORD IS OMITTED.
01  PRT-TC-RESULTS.
    03  PRT-BLANK           PIC X(120).
```

*Рисунок 216 (Часть 3 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA*

WORKING-STORAGE SECTION.

```
*****
* Объявление переменных *
*****
```

```
01 H-EMPTBL.  
 05 H-EMPNO  PIC X(6).  
 05 H-NAME.  
    49 H-NAME-LN  PIC S9(4) COMP-4.  
    49 H-NAME-DA  PIC X(32).  
 05 H-ADDRESS.  
    49 H-ADDRESS-LN  PIC S9(4) COMP-4.  
    49 H-ADDRESS-DA  PIC X(36).  
 05 H-CITY.  
    49 H-CITY-LN  PIC S9(4) COMP-4.  
    49 H-CITY-DA  PIC X(36).  
 05 H-EMPLLOC  PIC X(4).  
 05 H-SSNO    PIC X(11).  
 05 H-BORN    PIC X(10).  
 05 H-SEX     PIC X(1).  
 05 H-HIRED   PIC X(10).  
 05 H-DEPTNO  PIC X(3).  
 05 H-JOBCODE  PIC S9(3)V COMP-3.  
 05 H-SRATE   PIC S9(5) COMP.  
 05 H-EDUC    PIC S9(5) COMP.  
 05 H-SAL     PIC S9(6)V9(2) COMP-3.  
 05 H-VALIDCHK PIC S9(6)V COMP-3.
```

```
01 H-EMPTBL-IND-TABLE.  
 02 H-EMPTBL-IND      PIC S9(4) COMP OCCURS 15 TIMES.
```

```
*****
* Включаемые файлы с переменными, используемыми в стандартных  *
* процедурах COBOL и в SQLCA.                                     *
*****
```

```
EXEC SQL INCLUDE COBSVAR END-EXEC.  
EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
*****
* Объявление таблицы, содержащей информацию о служащем          *
*****
```

```
EXEC SQL DECLARE SYSADM.EMP TABLE  
  (EMPNO  CHAR(6) NOT NULL,  
   NAME    VARCHAR(32),  
   ADDRESS VARCHAR(36) ,  
   CITY    VARCHAR(36) ,  
   EMPLLOC CHAR(4) NOT NULL,  
   SSNO    CHAR(11),  
   BORN    DATE,  
   SEX     CHAR(1),  
   HIRED   CHAR(10),  
   DEPTNO  CHAR(3) NOT NULL,  
   JOBCODE  DECIMAL(3),  
   SRATE   SMALLINT,  
   EDUC    SMALLINT,
```

Рисунок 216 (Часть 4 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA

```

        SAL      DECIMAL(8,2) NOT NULL,
        VALCHK  DECIMAL(6))
END-EXEC.

*****
* Константы
*****
77 SITE-1                  PIC X(16) VALUE 'STLEC1'.
77 SITE-2                  PIC X(16) VALUE 'STLEC2'.
77 TEMP-EMPNO               PIC X(6)  VALUE '080000'.
77 TEMP-ADDRESS-LN          PIC 99   VALUE 15.
77 TEMP-CITY-LN             PIC 99   VALUE 18.

*****
* Объявление указателя, который будет использоваться для      *
* получения информации о переведшемся сотруднике           *
*****
EXEC SQL DECLARE C1 CURSOR FOR
    SELECT EMPNO, NAME, ADDRESS, CITY, EMPLOC,
           SSNO, BORN, SEX, HIRED, DEPTNO, JOBCODE,
           SRATE, EDUC, SAL, VALCHK
    FROM   SYSADM.EMP
    WHERE EMPNO = :TEMP-EMPNO
END-EXEC.

PROCEDURE DIVISION.
A101-HOUSE-KEEPING.
OPEN OUTPUT PRINTER.

*****
* Сотрудник переводится из STLEC1 в STLEC2.                  *
* Получить информацию о сотруднике из STLEC1, удалить информацию* *
* о сотруднике из STLEC1 и вставить сотрудника в STLEC2,          *
* используя информацию, полученную из STLEC1.                  *
*****
MAINLINE.
PERFORM CONNECT-TO-SITE-1
IF SQLCODE IS EQUAL TO 0
    PERFORM PROCESS-CURSOR-SITE-1
    IF SQLCODE IS EQUAL TO 0
        PERFORM UPDATE-ADDRESS
        PERFORM CONNECT-TO-SITE-2
        IF SQLCODE IS EQUAL TO 0
            PERFORM PROCESS-SITE-2.
        PERFORM COMMIT-WORK.

```

*Рисунок 216 (Часть 5 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA*

```

PROG-END.

CLOSE PRINTER.
GOBACK.

*****
* Установить подключение к STLEC1 *
*****


CONNECT-TO-SITE-1.

MOVE 'CONNECT TO STLEC1' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    CONNECT TO :SITE-1
END-EXEC.
PERFORM PTSQLCA.

*****
* Как только соединение с STLEC1 было успешно установлено,      *
* открыть указатель, который будет использоваться для получения *
* информации о переводящемся сотруднике.                      *
*****


PROCESS-CURSOR-SITE-1.

MOVE 'OPEN CURSOR C1' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    OPEN C1
END-EXEC.
PERFORM PTSQLCA.
IF SQLCODE IS EQUAL TO ZERO
    PERFORM FETCH-DELETE-SITE-1
    PERFORM CLOSE-CURSOR-SITE-1.

*****
* Получить информацию о переводящемся сотруднике.          *
* Если сотрудник существует, выполнить                      *
* DELETE-SITE-1, чтобы удалить сотрудника из STLEC1.       *
*****


FETCH-DELETE-SITE-1.

MOVE 'FETCH C1' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    FETCH C1 INTO :H-EMPTBL:H-EMPTBL-IND
END-EXEC.
PERFORM PTSQLCA.
IF SQLCODE IS EQUAL TO ZERO
    PERFORM DELETE-SITE-1.

```

*Рисунок 216 (Часть 6 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA*

```

*****
* Удалить сотрудника из STLEC1. *
*****
DELETE-SITE-1.

MOVE 'DELETE EMPLOYEE      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
MOVE 'DELETE EMPLOYEE      ' TO STNAME
EXEC SQL
    DELETE FROM SYSADM.EMP
        WHERE EMPNO = :TEMP-EMPNO
END-EXEC.
PERFORM PTSQLCA.

*****
* Закрыть указатель, использованный для получения информации о *
* переводимом сотруднике. *
*****
CLOSE-CURSOR-SITE-1.

MOVE 'CLOSE CURSOR C1      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    CLOSE C1
END-EXEC.
PERFORM PTSQLCA.

*****
* Обновить информацию о данном сотруднике, чтобы сделать      *
* ее актуальной *
*****
UPDATE-ADDRESS.
MOVE TEMP-ADDRESS-LN      TO H-ADDRESS-LN.
MOVE '1500 NEW STREET'    TO H-ADDRESS-DA.
MOVE TEMP-CITY-LN         TO H-CITY-LN.
MOVE 'NEW CITY, CA 97804' TO H-CITY-DA.
MOVE 'SJCA'               TO H-EMPLC.

*****
* Установить соединение с STLEC2 *
*****
CONNECT-TO-SITE-2.

MOVE 'CONNECT TO STLEC2    ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    CONNECT TO :SITE-2
END-EXEC.
PERFORM PTSQLCA.

```

*Рисунок 216 (Часть 7 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA*

```

*****
* Используя информацию, которая была получена из STLEC1, и      *
* обновлена выше, вставить сотрудника в STLEC2.                  *
*****
PROCESS-SITE-2.

MOVE 'INSERT EMPLOYEE      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    INSERT INTO SYSADM.EMP VALUES
    (:H-EMPNO,
     :H-NAME,
     :H-ADDRESS,
     :H-CITY,
     :H-EMPLOC,
     :H-SSNO,
     :H-BORN,
     :H-SEX,
     :H-HIRED,
     :H-DEPTNO,
     :H-JOBCODE,
     :H-SRATE,
     :H-EDUC,
     :H-SAL,
     :H-VALIDCHK)
END-EXEC.
PERFORM PTSQLCA.

*****
* ПРИНЯТЬ все изменения, сделанные в STLEC1 и в STLEC2.          *
*****
COMMIT-WORK.

MOVE 'COMMIT WORK      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    COMMIT
END-EXEC.
PERFORM PTSQLCA.

*****
* Включить стандартные процедуры языка COBOL                   *
*****
INCLUDE-SUBS.
EXEC SQL INCLUDE COBSSUB END-EXEC.

```

*Рисунок 216 (Часть 8 из 8). Пример программы с двухфазным принятием на языке COBOL для доступа DRDA*

---

## Пример программы на языке COBOL, использующей доступ по собственному протоколу DB2

Этот пример показывает доступ к распределенным данным, использующие собственный протокол DB2, с двухфазным принятием.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. TWOPHASE.  
AUTHOR.  
REMARKS.  
*****  
* *  
* ИМЯ МОДУЛЯ = TWOPHASE *  
* *  
* ОПИСАНИЕ = ПРИМЕР ПРОГРАММЫ DB2, ИСПОЛЬЗУЮЩЕЙ *  
* ДВУХФАЗНОЕ ПРИНЯТИЕ И МЕТОД *  
* РАСПРЕДЕЛЕННОГО ДОСТУПА ПО СОБСТВЕННОМУ *  
* ПРОТОКОЛУ DB2 *  
* *  
* COPYRIGHT = 5665-DB2 (C) АВТОРСКИЕ ПРАВА IBM CORP 1982, 1989 *  
* СМОТРИТЕ ИНСТРУКЦИЮ ОБ АВТОРСКИХ ПРАВАХ, ФОРМА НОМЕР G120-2083*  
* *  
* СОСТОЯНИЕ = ВЕРСИЯ 5 *  
* *  
* ФУНКЦИЯ = ЭТΟТ МОДУЛЬ ИЛЛЮСТРИРУЕТ РАСПРЕДЕЛЕННЫЙ ДОСТУП К *  
* ДАННЫМ С ИСПОЛЬЗОВАНИЕМ ДВУХФАЗНОГО ПРИНЯТИЯ, *  
* ПЕРЕВОДЯ СОТРУДНИКА С ОДНОГО МЕСТА НА ДРУГОЕ. *  
* *  
* ПРИМЕЧАНИЕ: ЭТА ПРОГРАММА ПРЕДПОЛАГАЕТ, ЧТО ТАБЛИЦА*  
* SYSADM.EMP РАСПОЛОЖЕНА В STLEC1 И В STLEC2. *  
* *  
* ТИП МОДУЛЯ = ПРОГРАММА НА ЯЗЫКЕ COBOL *  
* ОБРАБОТКА = ПРЕКОМПИЛЯТОР DB2, VS COBOL II *  
* РАЗМЕР МОДУЛЯ = СМОТРИТЕ РЕДАКТИРОВАНИЕ СВЯЗЕЙ *  
* АТРИБУТЫ = НЕ ПОВТОРНО-ВХОДИМАЯ, НЕ ПОВТОРНО-ИСПОЛЬЗУЕМАЯ *  
* *  
* ТОЧКА ВХОДА = *  
* ЦЕЛЬ = ПРОИЛЛЮСТРИРОВАТЬ ДВУХФАЗНОЕ ПРИНЯТИЕ *  
* КОМПОНОВКА = ВЫЗОВ ИЗ DSN RUN *  
* ВВОД = НЕТ *  
* ВЫВОД = *  
* ОБОЗНАЧЕНИЕ/ИМЯ = SYSPRINT *  
* ОПИСАНИЕ = ПЕЧАТАЕТ ОПИСАНИЕ КАЖДОГО ШАГА И *  
* ПОЛУЧЕННЮЮ SQLCA *  
* *  
* НОРМАЛЬНЫЙ ВЫХОД = КОД ВОЗВРАТА 0 ПРИ НОРМАЛЬНОМ ЗАВЕРШЕНИИ *  
* *  
* АВАРИЙНЫЙ ВЫХОД = НЕТ *  
* *  
* ВНЕШНИЕ ССЫЛКИ = *  
* ПРОГРАММНЫЕ СЛУЖБЫ = НЕТ *  
* ОБЛАСТИ ДАННЫХ = НЕТ *  
* УПРАВЛЯЮЩИЕ БЛОКИ = *  
* SQLCA - ОБЛАСТЬ СВЯЗИ SQL *  
* *  
* ТАБЛИЦЫ = НЕТ *  
* *  
* ИЗМЕНЕНИЕ АКТИВНОСТИ = НЕТ *  
* *  
*
```

Рисунок 217 (Часть 1 из 7). Пример программы двухфазного принятия на языке COBOL для собственного протокола DB2

```

*
* ПСЕВДОКОД
*
* MAINLINE.
*   Выполнить PROCESS-CURSOR-SITE-1, чтобы получить
*   информацию о сотруднике, который переводится на новое
*   место.
*   If информация о сотруднике успешно получена, Then
*     Do.
*       | Выполнить UPDATE-ADDRESS, чтобы обновить информацию
*         так, чтобы она отражала современное состояние.
*       | Выполнить PROCESS-SITE-2, чтобы вставить информацию
*         о сотруднике в место, куда он переводится.
*     End If информация о сотруднике успешно получена.
*   Выполнить COMMIT-WORK, чтобы ПРИНЯТЬ изменения, внесенные
*   в STLEC1 и в STLEC2.
*
* PROG-END.
*   Закрыть принтер.
*   Return.
*
* PROCESS-CURSOR-SITE-1.
*   Дать текстовое описание следующего шага.
*   Дать текстовое описание следующего шага.
*   получения информации о переводимом с этого места
*   сотруднике.
*   Напечатать выходную SQLCA.
*   If указатель успешно получен, Then
*     Do.
*     Do.
*       | удалить информацию о переводимом сотруднике с
*         этого места.
*       | Выполнить CLOSE-CURSOR-SITE-1, чтобы закрыть
*         указатель.
*     End If указатель успешно получен.
*
* FETCH-DELETE-SITE-1.
*   Дать текстовое описание следующего шага.
*   Получить информацию о переводящемся сотруднике.
*   Напечатать выходную SQLCA.
*   If информация получена успешно, Then
*     Do.
*       | Выполнить DELETE-SITE-1, чтобы удалить служащего с
*         этого места.
*     End If информация получена успешно.
*
* DELETE-SITE-1.
*   Дать текстовое описание следующего шага.
*   Удалить информацию о переводящемся сотруднике с этого
*   места.
*   Напечатать выходную SQLCA.
*
* CLOSE-CURSOR-SITE-1.
*   Дать текстовое описание следующего шага.
*   Закрыть указатель, использованный для получения
*   информации о переводящемся сотруднике.
*   Напечатать выходную SQLCA.
*

```

*Рисунок 217 (Часть 2 из 7). Пример программы двухфазного принятия на языке COBOL для собственного протокола DB2*

```

*      UPDATE-ADDRESS.          *
*      Обновить адрес сотрудника.    *
*      Обновить город сотрудника.    *
*      Обновить местоположение сотрудника.    *
*      *
*      PROCESS-SITE-2.          *
*      Дать текстовое описание следующего шага.    *
*      Вставить информацию о сотруднике в место, куда он    *
*      переводится.          *
*      Напечатать выходную SQLCA.    *
*      *
*      COMMIT-WORK.          *
*      ПРИНЯТЬ все изменения, внесенные в STLEC1 и в STLEC2.    *
*      *
*****
```

```

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT PRINTER, ASSIGN TO S-OUT1.

DATA DIVISION.
FILE SECTION.
FD  PRINTER
   RECORD CONTAINS 120 CHARACTERS
   DATA RECORD IS PRT-TC-RESULTS
   LABEL RECORD IS OMITTED.
01  PRT-TC-RESULTS.
   03  PRT-BLANK           PIC X(120).

WORKING-STORAGE SECTION.

*****  

* Объявление переменных          *
*****
```

```

01  H-EMPTBL.
05  H-EMPNO   PIC X(6).
05  H-NAME.
   49 H-NAME-LN   PIC S9(4) COMP-4.
   49 H-NAME-DA   PIC X(32).
05  H-ADDRESS.
   49 H-ADDRESS-LN   PIC S9(4) COMP-4.
   49 H-ADDRESS-DA   PIC X(36).
05  H-CITY.
   49 H-CITY-LN   PIC S9(4) COMP-4.
   49 H-CITY-DA   PIC X(36).
05  H-EMPLOC   PIC X(4).
05  H-SSNO     PIC X(11).
05  H-BORN     PIC X(10).
05  H-SEX      PIC X(1).
05  H-HIRED    PIC X(10).
05  H-DEPTNO   PIC X(3).
05  H-JOBCODE  PIC S9(3)V COMP-3.
05  H-SRATE    PIC S9(5) COMP.
05  H-EDUC     PIC S9(5) COMP.
05  H-SAL      PIC S9(6)V9(2) COMP-3.
05  H-VALIDCHK PIC S9(6)V COMP-3.
```

*Рисунок 217 (Часть 3 из 7). Пример программы двухфазного принятия на языке COBOL для собственного протокола DB2*

```

01 H-EMPTBL-IND-TABLE.
02 H-EMPTBL-IND      PIC S9(4) COMP OCCURS 15 TIMES.

*****
* Включаемые файлы с переменными, используемыми в стандартных *
* процедурах COBOL и в SQLCA.                                     *
*****


EXEC SQL INCLUDE COBSVAR END-EXEC.
EXEC SQL INCLUDE SQLCA END-EXEC.

*****
* Объявление таблицы, содержащей информацию о сотруднике          *
*****


EXEC SQL DECLARE SYSADM.EMP TABLE
(EMPNO  CHAR(6) NOT NULL,
 NAME    VARCHAR(32),
 ADDRESS VARCHAR(36) ,
 CITY    VARCHAR(36) ,
 EMPLOC  CHAR(4) NOT NULL,
 SSNO    CHAR(11),
 BORN    DATE,
 SEX     CHAR(1),
 HIRED   CHAR(10),
 DEPTNO  CHAR(3) NOT NULL,
 JOBCODE  DECIMAL(3),
 SRATE   SMALLINT,
 EDUC    SMALLINT,
 SAL     DECIMAL(8,2) NOT NULL,
 VALCHK  DECIMAL(6))
END-EXEC.

*****
* Константы
*****


77 TEMP-EMPNO           PIC X(6) VALUE '080000'.
77 TEMP-ADDRESS-LN       PIC 99    VALUE 15.
77 TEMP-CITY-LN          PIC 99    VALUE 18.

*****
* Объявление указателя, который будет использоваться для          *
* получения информации о переведящемся сотруднике                  *
*****


EXEC SQL DECLARE C1 CURSOR FOR
      SELECT EMPNO, NAME, ADDRESS, CITY, EMPLOC,
             SSNO, BORN, SEX, HIRED, DEPTNO, JOBCODE,
             SRATE, EDUC, SAL, VALCHK
        FROM STLEC1.SYSADM.EMP
       WHERE EMPNO = :TEMP-EMPNO
END-EXEC.

```

*Рисунок 217 (Часть 4 из 7). Пример программы двухфазного принятия на языке COBOL для собственного протокола DB2*

```

PROCEDURE DIVISION.
A101-HOUSE-KEEPING.
OPEN OUTPUT PRINTER.

*****
* Сотрудник переводится из STLEC1 в STLEC2. *
* Получить информацию о сотруднике из STLEC1, удалить информацию*
* о сотруднике из STLEC1 и вставить сотрудника в STLEC2,      *
* используя информацию, полученную из STLEC1.                  *      *
*****


MAINLINE.
    PERFORM PROCESS-CURSOR-SITE-1
    IF SQLCODE IS EQUAL TO 0
        PERFORM UPDATE-ADDRESS
        PERFORM PROCESS-SITE-2.
    PERFORM COMMIT-WORK.

PROG-END.
    CLOSE PRINTER.
    GOBACK.

*****
* Открыть указатель, который будет использован для получения      *
* информации о переводимом сотруднике.                           *
*****


PROCESS-CURSOR-SITE-1.

    MOVE 'OPEN CURSOR C1      ' TO STNAME
    WRITE PRT-TC-RESULTS FROM STNAME
    EXEC SQL
        OPEN C1
    END-EXEC.
    PERFORM PTSQLCA.
    IF SQLCODE IS EQUAL TO ZERO
        PERFORM FETCH-DELETE-SITE-1
        PERFORM CLOSE-CURSOR-SITE-1.

*****
* Получить информацию о переводящемся сотруднике.          *
* Если этот служащий существует, выполнить                   *
* DELETE-SITE-1, чтобы удалить сотрудника из STLEC1.       *
*****


FETCH-DELETE-SITE-1.

    MOVE 'FETCH C1      ' TO STNAME
    WRITE PRT-TC-RESULTS FROM STNAME
    EXEC SQL
        FETCH C1 INTO :H-EMPTBL:H-EMPTBL-IND
    END-EXEC.

```

*Рисунок 217 (Часть 5 из 7). Пример программы двухфазного принятия на языке COBOL для собственного протокола DB2*

```

        PERFORM PTSQLCA.
        IF SQLCODE IS EQUAL TO ZERO
            PERFORM DELETE-SITE-1.

*****
* Удалить сотрудника из STLEC1. *
*****


        DELETE-SITE-1.

        MOVE 'DELETE EMPLOYEE'      TO STNAME
        WRITE PRT-TC-RESULTS FROM STNAME
        MOVE 'DELETE EMPLOYEE'      TO STNAME
        EXEC SQL
            DELETE FROM STLEC1.SYSADM.EMP
                WHERE EMPNO = :TEMP-EMPNO
        END-EXEC.
        PERFORM PTSQLCA.

*****
* Закрыть указатель, использованный для получения информации о *
* переведимом сотруднике. *
*****


        CLOSE-CURSOR-SITE-1.

        MOVE 'CLOSE CURSOR C1'      TO STNAME
        WRITE PRT-TC-RESULTS FROM STNAME
        EXEC SQL
            CLOSE C1
        END-EXEC.
        PERFORM PTSQLCA.

*****
* Обновить информацию о данном сотруднике, чтобы сделать ее      *
* актуальной. *
*****


        UPDATE-ADDRESS.
        MOVE TEMP-ADDRESS-LN      TO H-ADDRESS-LN.
        MOVE '1500 NEW STREET'    TO H-ADDRESS-DA.
        MOVE TEMP-CITY-LN        TO H-CITY-LN.
        MOVE 'NEW CITY, CA 97804' TO H-CITY-DA.
        MOVE 'SJCA'              TO H-EMPLOC.

```

*Рисунок 217 (Часть 6 из 7). Пример программы двухфазного принятия на языке COBOL для собственного протокола DB2*

```
*****
* Используя информацию, которая была получена из STLEC1, и      *
* только что исправлена, вставить сотрудника в STLEC2.          *
*****
```

PROCESS-SITE-2.

```
MOVE 'INSERT EMPLOYEE      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    INSERT INTO STLEC2.SYSADM.EMP VALUES
    (:H-EMPNO,
     :H-NAME,
     :H-ADDRESS,
     :H-CITY,
     :H-EMPLOC,
     :H-SSNO,
     :H-BORN,
     :H-SEX,
     :H-HIRED,
     :H-DEPTNO,
     :H-JOBCODE,
     :H-SRATE,
     :H-EDUC,
     :H-SAL,
     :H-VALIDCHK)
END-EXEC.
PERFORM PTSQLCA.
```

```
*****
* ПРИНЯТЬ все изменения, внесенные в STLEC1 и в STLEC2.      *
*****
```

COMMIT-WORK.

```
MOVE 'COMMIT WORK      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    COMMIT
END-EXEC.
PERFORM PTSQLCA.
```

```
*****
* Включить стандартные процедуры языка COBOL           *
*****
```

```
INCLUDE-SUBS.
EXEC SQL INCLUDE COBSSUB END-EXEC.
```

Рисунок 217 (Часть 7 из 7). Пример программы двухфазного принятия на языке COBOL для собственного протокола DB2

---

## **Примеры использования хранимых процедур**

В этом разделе показаны примеры программ, которые можно использовать при написании собственных хранимых процедур. В DSN8610.SDSNSAMP содержатся примеры заданий DSNTEJ6P и DSNTEJ6S и программы DSN8EP1 и DSN8EP2, которые вы можете запустить.

### **Вызов хранимой процедуры из программы на языке С**

В этом примере показано, как вызывать версию на языке С хранимой процедуры GETPRML, использующую метод передачи параметров GENERAL WITH NULLS. Поскольку эту хранимая процедура возвращает наборы результатов, данная программа проверяет наборы результатов и получает их содержимое.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
    /* Включение SQLCA и SQLDA */
    EXEC SQL INCLUDE SQLCA;
    EXEC SQL INCLUDE SQLDA;
    /* Объявления переменных, не относящихся к SQL. */
    short int i; /* Счетчик цикла */
    /* Обявление:
     * - Параметров, используемых для вызова хранимой процедуры */
    /* GETPRML */
    /* - SQLDA для оператора DESCRIBE PROCEDURE */
    /* - SQLDA для оператора DESCRIBE CURSOR */
    /* - Локаторов наборов результатов для получения до трех */
    /* наборов результатов */
    EXEC SQL BEGIN DECLARE SECTION;
    char procnm[19]; /* Входной параметр -- имя процедуры */
    char schema[9]; /* Входной параметр -- пользовательская схема */
    long int out_code; /* Выходное значение -- SQLCODE */
                        /* от оператора SELECT. */
    struct {
        short int parmlen;
        char parmtxt[254];
    } parmlst; /* Выходные значения -- RUNOPTS */
                /* для соответствующей строки в */
                /* таблице каталога SYSROUTINES */
    short int procnm_ind;
    short int schema_ind;
    short int out_code_ind;
    short int parmlst_ind;
} parmind; /* Структура переменных-индикаторов */

struct sqlda *proc_da; /* SQLDA для DESCRIBE PROCEDURE */
struct sqlda *res_da; /* SQLDA для DESCRIBE CURSOR */
static volatile
SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2, *loc3;
/* Локаторы */
EXEC SQL END DECLARE SECTION;

```

*Рисунок 218 (Часть 1 из 4). Вызов хранимой процедуры из программы на языке C*

```

/*
 ****
 /* Выделение SQLDA, используемых для операторов DESCRIBE      */
 /* PROCEDURE и DESCRIBE CURSOR. Предполагается, что          */
 /* возвращается не более трех указателей и что каждый набор   */
 /* результатов содержит не более пяти столбцов.           */
 /****

proc_da = (struct sqlda *)malloc(SQLDASIZE(3));
res_da = (struct sqlda *)malloc(SQLDASIZE(5));

/*
 ****
 /* Вызов хранимой процедуры GETPRML, возвращающей значения   */
 /* RUNOPTS для этой хранимой процедуры. В этом примере        */
 /* запрашиваются определения PARMLIST для хранимой          */
 /* с именем DSN8EP2.                                         */
 /*
 /* Этот вызов должен завершаться с SQLCODE +466,            */
 /* так как хранимая процедура GETPRML возвращает наборы    */
 /* результатов.                                              */
 /****

strcpy(procnm,"dsn8ep2");
/* Входной параметр -- имя процедуры, которую нужно найти */
strcpy(schema,"");
/* Входной параметр -- имя схемы для этой процедуры */

parmind.procnam_ind=0;
parmind.schema_ind=0;
parmind.out_code_ind=0;
/* Указывает, что ни один из входных параметров */
/* не содержит пустого значения */
parmind.parmlst_ind=-1;
/* PARMLST -- это выходной параметр. */
/* Задаем, что параметр PARMLST имеет пустое */
/* значение, то есть реквестер DB2 может */
/* не посыпать на сервер всю переменную */
/* PARMLST. Это экономит время на операции */
/* ввода-вывода в сети, так как */
/* довольно большой размер. */

EXEC SQL
CALL GETPRML(:procnam INDICATOR :parmind.procnam_ind,
             :schema INDICATOR :parmind.schema_ind,
             :out_code INDICATOR :parmind.out_code_ind,
             :parm1st INDICATOR :parmind.parm1st_ind);
if(SQLCODE!=+466)          /* При ошибке SQL CALL */
{
    /* напечатать SQLCODE и текст */
    /* сообщения об ошибке */
    printf("SQL CALL failed due to SQLCODE = %d\n",SQLCODE);
    printf("sqlca.sqlerrmc = ");
    for(i=0;i<sqlca.sqlerrml;i++)
        printf("%c",sqlca.sqlerrmc[i]);
    printf("\n");
}

```

*Рисунок 218 (Часть 2 из 4). Вызов хранимой процедуры из программы на языке C*

```

else          /* При успешном выполнении      */
/* оператора CALL                */
if(out_code!=0)    /* Если возникли ошибки при   */
/* выполнении процедуры GETPRML */
printf("GETPRML failed due to RC = %d\n",out_code);
/*****************************************/
/* Если ошибок не было:          */
/* - Напечатать возвращенные значения параметров. */
/* - Получить возвращенные наборы результатов. */
/*****************************************/
else
{
    printf("RUNOPTS = %s\n",parm1st.parmtxt);
    /* Напечатать список RUNOPTS */

/*****************************************/
/* Использовать оператор DESCRIBE PROCEDURE,        */
/* возвращающий информацию о наборах результатов   */
/* в SQLDA, на которую указывает proc_da:          */
/* - SQLD содержит число наборов результатов,     */
/* возвращенных хранимой процедурой.               */
/* - Каждый из элементов SQLVAR содержит следующую */
/* информацию об одном наборе результатов:         */
/* - SQLNAME содержит имя указателя, используемого  */
/* хранимой процедурой для возврата этого набора   */
/* результатов.                                     */
/* - SQLIND содержит оценку числа строк в этом наборе */
/* результатов.                                     */
/* - SQLDATA содержит локатор результата для этого   */
/* набора результатов.                            */
/*****************************************/
EXEC SQL DESCRIBE PROCEDURE INTO :proc_da;
/*****************************************/
/* Предполагается, что значение SQLD было проверено и   */
/* обнаружено, что имеется один набор результатов.       */
/* Используется оператор ASSOCIATE LOCATORS для задания */
/* локатора результата для этого набора результатов. */
/*****************************************/
EXEC SQL ASSOCIATE LOCATORS (:loc1) WITH PROCEDURE GETPRML;

/*****************************************/
/* При помощи оператора ALLOCATE CURSOR           */
/* указатель связывается с набором результатов.    */
/*****************************************/
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
/*****************************************/
/* Оператор DESRIBE CURSOR                         */
/* определяет столбцы в этом наборе результатов. */
/*****************************************/
EXEC SQL DESCRIBE CURSOR C1 INTO :res_da;

```

*Рисунок 218 (Часть 3 из 4). Вызов хранимой процедуры из программы на языке С*

```
/********************************************/  
/* Вызвать подпрограмму (не показанную здесь), которая: */  
/* - Выделяет буфер для значений данных и индикаторов, */  
/* выбираемых из таблицы результатов. */  
/* - В поля SQLDATA и SQLIND в каждой SQLVAR в *res_da */  
/* записывает адреса, куда должны быть помещены */  
/* выбранные значения данных и переменных-индикаторов.*/  
/********************************************/  
alloc_outbuff(res_da);  
  
/********************************************/  
/* Выбрать данные из таблицы результатов. */  
/********************************************/  
while(SQLCODE==0)  
    EXEC SQL FETCH C1 USING DESCRIPTOR :*res_da;  
}  
return;  
}
```

Рисунок 218 (Часть 4 из 4). Вызов хранимой процедуры из программы на языке С

## Вызов хранимой процедуры из программы на языке COBOL

В этом примере показано, как вызвать версию хранимой процедуры GETPRML, использующей метод передачи параметров GENERAL, из программы на языке COBOL в системе MVS. Поскольку эту хранимую процедуру возвращает наборы результатов, данная программа проверяет наборы результатов и получает их содержимое.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. CALPRML.  
  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
SELECT REPOUT  
    ASSIGN TO UT-S-SYSPRINT.  
  
DATA DIVISION.  
FILE SECTION.  
FD REPOUT  
    RECORD CONTAINS 127 CHARACTERS  
    LABEL RECORDS ARE OMITTED  
    DATA RECORD IS REPREC.  
01 REPREC          PIC X(127).  
  
WORKING-STORAGE SECTION.  
*****  
* СООБЩЕНИЯ ДЛЯ ОПЕРАТОРА SQL CALL *  
*****  
01 SQLREC.  
    02 BADMSG    PIC X(34) VALUE  
        ' SQL CALL FAILED DUE TO SQLCODE = '.  
    02 BADCODE   PIC +9(5) USAGE DISPLAY.  
    02 FILLER    PIC X(80) VALUE SPACES.  
01 ERMREC.  
    02 ERRMSG    PIC X(12) VALUE ' SQLERRMC = '.  
    02 ERMCODE   PIC X(70).  
    02 FILLER    PIC X(38) VALUE SPACES.  
01 CALLREC.  
    02 CALLMSG   PIC X(28) VALUE  
        ' GETPRML FAILED DUE TO RC = '.  
    02 CALLCODE   PIC +9(5) USAGE DISPLAY.  
    02 FILLER    PIC X(42) VALUE SPACES.  
01 RSLTREC.  
    02 RSLTMSG   PIC X(15) VALUE  
        ' TABLE NAME IS '.  
    02 TBLNAME   PIC X(18) VALUE SPACES.  
    02 FILLER    PIC X(87) VALUE SPACES.
```

Рисунок 219 (Часть 1 из 3). Вызов хранимой процедуры из программы на языке COBOL

```

*****
* РАБОЧИЕ ОБЛАСТИ *
*****
01 PROCNM PIC X(18).
01 SCHEMA PIC X(8).
01 OUT-CODE          PIC S9(9) USAGE COMP.
01 PARMST.
 49 PARMLEN      PIC S9(4) USAGE COMP.
 49 PARMTXT     PIC X(254).
01 PARMBUF REDEFINES PARMST.
 49 PARBLEN      PIC S9(4) USAGE COMP.
 49 PARMARRY    PIC X(127) OCCURS 2 TIMES.
01 NAME.
 49 NAMELEN      PIC S9(4) USAGE COMP.
 49 NAMETXT     PIC X(18).
77 PARMIND      PIC S9(4) COMP.
77 I             PIC S9(4) COMP.
77 NUMLINES     PIC S9(4) COMP.
*****
* ОБЪЯВЛЕНИЕ ЛОКАТОРА НАБОРА РЕЗУЛЬТАТОВ ДЛЯ      *
* ВОЗВРАЩЕННОГО НАБОРА РЕЗУЛЬТАТОВ.                  *
*****
01 LOC           USAGE SQL TYPE IS
                 RESULT-SET-LOCATOR VARYING.

*****
* ВКЛЮЧЕНИЕ SQL ДЛЯ SQLCA                          *
*****
EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
-----
PROG-START.
  OPEN OUTPUT REPOUT.
*          ОТКРЫТЬ ВЫХОДНОЙ ФАЙЛ
  MOVE 'DSN8EP2'          ' TO PROCNM.
*          ВХОДНОЙ ПАРАМЕТР -- ПРОЦЕДУРА, КОТОРУЮ НУЖНО НАЙТИ
  MOVE SPACES TO SCHEMA.
*          ВХОДНОЙ ПАРАМЕТР -- СХЕМА В SYSROUTINES
  MOVE -1 TO PARMIND.
*          ПАРАМЕТР PARMLST -- ЭТО ВЫХОДНОЙ ПАРАМЕТР.
*          ЗАДАЕМ, ЧТО ПАРАМЕТР PARMLST ИМЕЕТ ПУСТОЕ
*          ЗНАЧЕНИЕ, А ЗНАЧИТ, РЕКВЕСТЕРУ DB2 НЕ НУЖНО
*          ПОСЫПАТЬ НА СЕРВЕР ВСЮ ПЕРЕМЕННУЮ PARMLST.
*          ЭТО ЭКОНОМИТ ВРЕМЯ ОПЕРАЦИЙ ВВОДА-ВЫВОДА
*          В СЕТИ, ТАК КАК ПЕРЕМЕННАЯ PARMLST
*          ИМЕЕТ ДОВОЛЬНО БОЛЬШОЙ РАЗМЕР.
  EXEC SQL
  CALL GETPRML(:PROCNM,
               :SCHEMA,
               :OUT-CODE,
               :PARMLST INDICATOR :PARMINDEX)
END-EXEC.

```

*Рисунок 219 (Часть 2 из 3). Вызов хранимой процедуры из программы на языке COBOL*

```

*           ВЫПОЛНЯЕМ ВЫЗОВ ХРАНИМОЙ ПРОЦЕДУРЫ
IF SQLCODE NOT EQUAL TO +466 THEN
*           ЕСЛИ ВОЗВРАЩЕН SQLCODE ОШИБКИ
MOVE SQLCODE TO BADCODE
WRITE REPREC FROM SQLREC
MOVE SQLERRMC TO ERRMCODE
WRITE REPREC FROM ERRMREC
ELSE
    PERFORM GET-PARMS
    PERFORM GET-RESULT-SET.
PROG-END.
CLOSE REPOUT.
*           ЗАКРЫТЬ ВЫХОДНОЙ ФАЙЛ
GOBACK.
PARMPRT.
    MOVE SPACES TO REPREC.
    WRITE REPREC FROM PARMARRY(I)
    AFTER ADVANCING 1 LINE.
GET-PARMS.
*           ЕСЛИ ВЫЗОВ ЗАВЕРШЕН УСПЕШНО
IF OUT-CODE NOT EQUAL TO 0 THEN
*           ЕСЛИ ПРИ ВЫПОЛНЕНИИ GETPRML ВОЗНИКЛИ ОШИБКИ
MOVE OUT-CODE TO CALLCODE
WRITE REPREC FROM CALLREC
ELSE
*           ЕСЛИ ОШИБОК НЕ БЫЛО
DIVIDE 127 INTO PARMLEN GIVING NUMLINES ROUNDED
*           ОПРЕДЕЛЯЕМ, СКОЛЬКО СТРОК НАПЕЧАТАТЬ
PERFORM PARMPRT VARYING I
    FROM 1 BY 1 UNTIL I GREATER THAN NUMLINES.
GET-RESULT-SET.
*****
* ПРЕДПОЛАГАЕТСЯ, ЧТО ИЗВЕСТНО, ЧТО ВОЗВРАЩАЕТСЯ      *
* ТОЛЬКО ОДИН НАБОР РЕЗУЛЬТАТОВ И ИЗВЕСТЕН ФОРМАТ      *
* ЭТОГО НАБОРА РЕЗУЛЬТАТОВ. ВЫДЕЛЯЕМ УКАЗАТЕЛЬ ДЛЯ      *
* НАБОРА РЕЗУЛЬТАТОВ И ВЫБИРАЕМ СОДЕРЖИМОЕ            *
* НАБОРА РЕЗУЛЬТАТОВ.                                     *
*****
EXEC SQL ASSOCIATE LOCATORS (:LOC)
    WITH PROCEDURE GETPRML
    END-EXEC.
*           СВЯЗАТЬ НАБОР РЕЗУЛЬТАТОВ С ЛОКАТОРОМ
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC
    END-EXEC.
*           СВЯЗАТЬ УКАЗАТЕЛЬ С НАБОРОМ РЕЗУЛЬТАТОВ
PERFORM GET-ROWS VARYING I
    FROM 1 BY 1 UNTIL SQLCODE EQUAL TO +100.
GET-ROWS.
    EXEC SQL FETCH C1 INTO :NAME
    END-EXEC.
    MOVE NAME TO TBLNAME.
    WRITE REPREC FROM RSLTREC
    AFTER ADVANCING 1 LINE.

```

*Рисунок 219 (Часть 3 из 3). Вызов хранимой процедуры из программы на языке COBOL*

## Вызов хранимой процедуры из программы на языке PL/I

В этом примере показано, как вызвать версию хранимой процедуры GETPRML, использующей метод передачи параметров GENERAL, из программы на языке PL/I в системе MVS.

```
*PROCESS SYSTEM(MVS);
CALPRML:
PROC OPTIONS(MAIN);

/************************************************
/* Объявление параметров, используемых для вызова хранимой */
/* процедуры GETPRML.                                         */
/************************************************
DECLARE PROCNM CHAR(18),      /* Входной параметр -- имя процедуры */
        SCHEMA CHAR(8),      /* Входной параметр -- пользовательская схема */
        OUT_CODE FIXED BIN(31),
                           /* Выходное значение -- SQLCODE */
                           /* от оператора SELECT.          */
        PARMLST CHAR(254)    /* Выходные значения -- RUNOPTS */
                           /* VARYING,           /* для соответствующих строк в */
                           /*                   /* таблице каталога SYSROUTINES */
        PARMIND FIXED BIN(15);
                           /* Переменная-индикатор PARMLST */
/************************************************
/* Включение SQLCA                                         */
/************************************************
EXEC SQL INCLUDE SQLCA;
/************************************************
/* Вызов хранимой процедуры GETPRML, возвращающей значения */
/* RUNOPTS для этой хранимой процедуры. В этом примере       */
/* запрашиваются значения RUNOPTS для хранимой процедуры   */
/* с именем DSN8EP2.                                         */
/************************************************
PROCNM = 'DSN8EP2';
                           /* Входной параметр -- имя процедуры, которую нужно найти */
SCHEMA = ' ';
SCHEMA = ' '; /* Входной параметр -- имя схемы в SYSROUTINES */
PARMIND = -1; /* Параметр PARMLST -- это выходной параметр.
                /* Задаем, что параметр PARMLST имеет пустое
                /* значение, то есть реквестеру DB2 не нужно
                /* посыпать на сервер всю переменную PARMLST.
                /* Это экономит время на операции ввода-вывода
                /* в сети, так как переменная PARMLST имеет
                /* довольно большой размер. */
EXEC SQL
CALL GETPRML(:PROCNM,
             :SCHEMA,
             :OUT_CODE,
             :PARMLST INDICATOR :PARMIND);
```

Рисунок 220 (Часть 1 из 2). Вызов хранимой процедуры из программы на языке PL/I

```

IF SQLCODE=-0 THEN          /* При ошибке SQL CALL      */
DO;
  PUT SKIP EDIT('SQL CALL failed due to SQLCODE = ',
    SQLCODE) (A(34),A(14));
  PUT SKIP EDIT('SQLERRM = ',
    SQLERRM) (A(10),A(70));
END;
ELSE                      /* Если CALL выполнен успешно */
  IF OUT_CODE=-0 THEN     /* Была ли ошибка GETPRML? */
    PUT SKIP EDIT('GETPRML failed due to RC = ',
      OUT_CODE) (A(33),A(14));
  ELSE                    /* Если нет ошибок           */
    PUT SKIP EDIT('RUNOPTS = ', PARMST) (A(11),A(200));
RETURN;
END CALPRML;

```

*Рисунок 220 (Часть 2 из 2). Вызов хранимой процедуры из программы на языке PL/I*

## Хранимая процедура на языке C: GENERAL

Эта хранимая процедура примера выполняет следующие действия:

- Ищет строки в таблице SYSROUTINES каталога DB2, значения в которых совпадают с входными параметрами, переданными из программы клиента. Эти два входных параметра содержат значения для столбцов NAME и SCHEMA.
- Ищет в таблице SYSTABLES каталога DB2 все таблицы, у которых значения столбца CREATOR совпадает со значением входного параметра SCHEMA. Хранимая процедура использует указатель, чтобы возвратить имена этих таблиц.

Для этой хранимой процедуры используется метод передачи параметров GENERAL.

Выходные параметры этой хранимой процедуры содержат значение SQLCODE для оператора SELECT и значение столбца RUNOPTS из таблицы SYSROUTINES.

Оператор CREATE PROCEDURE для этой хранимой процедуры может выглядеть так:

```

CREATE PROCEDURE GETPRML(
  PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
  OUTCODE INTEGER OUT, PARMST VARCHAR(254) OUT)
LANGUAGE C
DETERMINISTIC
READS SQL
EXTERNAL NAME 'GETPRML'
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
EXTERNAL SECURITY DB2
RESULT SETS 2
COMMIT ON RETURN NO;

```

```

#pragma runopts(plist(os))
#include <stdlib.h>

EXEC SQL INCLUDE SQLCA;

/*********************************************************************
/* Объявление переменных С для операций SQL с параметрами.      */
/* Это локальные переменные для этой программы на языке С,      */
/* и значения параметров для хранимой процедуры нужно           */
/* скопировать в эти локальные переменные и обратно.           */
/********************************************************************/

EXEC SQL BEGIN DECLARE SECTION;
char PROCNM[19];
char SCHEMA[9];
char PARMLST[255];
EXEC SQL END DECLARE SECTION;

/*********************************************************************
/* Объявление указателей для возвращения наборов результатов   */
/* вызывающей программе.                                         */
/********************************************************************/

EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
SELECT NAME
FROM SYSIBM.SYSTABLES
WHERE CREATOR=:SCHEMA;

main(argc,argv)
int argc;
char *argv[];
{
    /*********************************************************************
    /* Копирование значений входных параметров в область,      */
    /* выделенную в программе для работы SQL.                      */
    /********************************************************************/
    strcpy(PROCNM, argv[1]);
    strcpy(SCHEMA, argv[2]);

    /*********************************************************************
    /* Выдается SQL SELECT для таблицы SYSROUTINES                 */
    /* каталога DB2.                                                 */
    /********************************************************************/
    strcpy(PARMLST, "");          /* Очистить PARMLST        */
    EXEC SQL
        SELECT RUNOPTS INTO :PARMLST
        FROM SYSIBM.ROUTINES
        WHERE NAME=:PROCNM AND
              SCHEMA=:SCHEMA;
}

```

*Рисунок 221 (Часть 1 из 2). Хранимая процедура на языке С, использующая метод передачи параметров GENERAL*

```

/*
 * Копирование SQLCODE в список выходных параметров.
 */
*(int *) argv[3] = SQLCODE;

/*
 * Копируем значение PARMLST, возвращенное SELECT,
 * обратно в список параметров хранимой процедуры.
 */
strcpy(argv[4], PARMLST);

/*
 * Открыть указатель C1, чтобы DB2 вернула вызывающей
 * программе набор результатов.
 */
EXEC SQL OPEN C1;
}

```

*Рисунок 221 (Часть 2 из 2). Хранимая процедура на языке C, использующая метод передачи параметров GENERAL*

## Хранимая процедура на языке C: GENERAL WITH NULLS

Эта хранимая процедура примера выполняет следующие действия:

- Ищет в таблице SYSROUTINES каталога DB2 строки, значения в которых совпадают с входными параметрами, переданными из программы клиента. Эти два входных параметра содержат значения для столбцов NAME и SCHEMA.
- Ищет в таблице SYSTABLES каталога DB2 все таблицы, у которых значения столбца CREATOR совпадает со значением входного параметра SCHEMA. Хранимая процедура использует указатель, чтобы возвратить имена этих таблиц.

Для этой хранимой процедуры используется метод передачи параметров GENERAL WITH NULLS.

Выходные параметры этой хранимой процедуры содержат значение SQLCODE для оператора SELECT и значение столбца RUNOPTS из таблицы SYSROUTINES.

Оператор CREATE PROCEDURE для этой хранимой процедуры может выглядеть так:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
                           OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
LANGUAGE C
DETERMINISTIC
READS SQL
EXTERNAL NAME 'GETPRML'
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
EXTERNAL SECURITY DB2
RESULT SETS 2
COMMIT ON RETURN NO;
```

```

#pragma runopts(plist(os))
#include <stdlib.h>

EXEC SQL INCLUDE SQLCA;

/*********************************************************************
/* Объявление переменных С для операций SQL с параметрами.      */
/* Это локальные переменные для этой программы на языке С,      */
/* и значения параметров хранимой процедуры нужно скопировать   */
/* в эти локальные переменные и обратно.                           */
/********************************************************************

EXEC SQL BEGIN DECLARE SECTION;
char PROCNM[19];
char SCHEMA[9];
char PARMLST[255];
struct INDICATORS {
    short int PROCNM_IND;
    short int SCHEMA_IND;
    short int OUT_CODE_IND;
    short int PARMLST_IND;
} PARM_IND;
EXEC SQL END DECLARE SECTION;

/*********************************************************************
/* Объявление указателей для возвращения наборов результатов  */
/* вызывающей программе.                                         */
/********************************************************************

EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
SELECT NAME
FROM SYSIBM.SYSTABLES
WHERE CREATOR=:SCHEMA;

main(argc,argv)
int argc;
char *argv[];
{

/*********************************************************************
/* Копирование значений входных параметров в область,      */
/* выделенную в программе для работы SQL.                      */
/********************************************************************

strcpy(PROCNM, argv[1]);
strcpy(SCHEMA, argv[2]);

/*********************************************************************
/* Копирование переменных-индикаторов для списка параметров.*/
/********************************************************************

memcpy(&PARM_IND,(struct INDICATORS *) argv[5],
sizeof(PARM_IND));

```

*Рисунок 222 (Часть 1 из 2). Хранимая процедура на языке С, использующая метод передачи параметров GENERAL WITH NULLS*

```

/*
 * Если какой-либо из входных параметров содержит      */
/* пустое значение, возвращается код ошибки и PARMLST   */
/* присваивается пустое значение.                      */
*/
if (PARM_IND.PROCNM_IND<0 ||          /* задать выходной код возврата */
    PARM_IND.SCHEMA_IND<0 ||          /* непустое значение */
    *(int *) argv[3] = 9999;           /* PARMLST имеет пустое значение */
    PARM_IND.OUT_CODE_IND = 0;         /* SYSIBM.SYSROUTINES. */
    PARM_IND.PARMLST_IND = -1;        /* */
}
else {
    /*
     * Если входные параметры имеют непустые значения,      */
     * выдается оператор SQL SELECT для таблицы каталога   */
     * SYSIBM.SYSROUTINES.                                */
    strcpy(PARMLST, "");             /* Очистить PARMLST */
    EXEC SQL
        SELECT RUNOPTS INTO :PARMLST
        FROM SYSIBM.SYSROUTINES
        WHERE NAME=:PROCNM AND
              SCHEMA=:SCHEMA;
    /*
     * Копирование SQLCODE в список выходных параметров.   */
    *(int *) argv[3] = SQLCODE;
    PARM_IND.OUT_CODE_IND = 0;         /* OUT_CODE имеет непустое значение */
}
/*
 * Копирование значения RUNOPTS в область выходных      */
/* параметров.                                         */
strcpy(argv[4], PARMLST);

/*
 * Копирование индикаторов пустых значений в область      */
/* выходных параметров.                                */
memcpuy((struct INDICATORS *) argv[5],&PARM_IND,
        sizeof(PARM_IND));

/*
 * Открыть указатель C1, чтобы DB2 вернула вызывающей   */
/* программе набор результатов.                         */
EXEC SQL OPEN C1;
}

```

*Рисунок 222 (Часть 2 из 2). Хранимая процедура на языке C, использующая метод передачи параметров GENERAL WITH NULLS*

## Хранимая процедура на языке COBOL: GENERAL

Эта хранимая процедура примера выполняет следующие действия:

- Ищет в таблице SYSROUTINES каталога DB2 строки, значения в которых совпадают с входными параметрами, переданными из программы клиента. Эти два входных параметра содержат значения для столбцов NAME и SCHEMA.

- Ищет в таблице SYSTABLES каталога DB2 все таблицы, у которых значения столбца CREATOR совпадает со значением входного параметра SCHEMA. Хранимая процедура использует указатель, чтобы возвратить имена этих таблиц.

Эта процедура может возвращать пустые значения для выходных переменных хоста.

Для этой хранимой процедуры используется метод передачи параметров GENERAL.

Выходные параметры этой хранимой процедуры содержат значение SQLCODE для оператора SELECT и значение столбца RUNOPTS из таблицы SYSROUTINES.

Оператор CREATE PROCEDURE для этой хранимой процедуры может выглядеть так:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
                           OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE COBOL
  DETERMINISTIC
  READS SQL
  EXTERNAL NAME 'GETPRML'
  COLLID GETPRML
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL
  STAY RESIDENT NO
  RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
  WLM ENVIRONMENT SAMPPROG
  PROGRAM TYPE MAIN
  EXTERNAL SECURITY DB2
  RESULT SETS 2
  COMMIT ON RETURN NO;
```

```

CBL RENT
    IDENTIFICATION DIVISION.
        PROGRAM-ID. GETPRML.
        AUTHOR. EXAMPLE.
        DATE-WRITTEN. 03/25/98.

    ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
            FILE-CONTROL.
        DATA DIVISION.
            FILE SECTION.

    WORKING-STORAGE SECTION.

        EXEC SQL INCLUDE SQLCA END-EXEC.
*****
*   ОБЪЯВЛЕНИЕ ПЕРЕМЕННОЙ ХОСТА ДЛЯ ХРАНЕНИЯ
*   ВХОДНОЙ СХЕМЫ
*****
01 INSCHEMA PIC X(8).

*****
*   ОБЪЯВЛЕНИЕ УКАЗАТЕЛЯ ДЛЯ ВОЗВРАЩЕНИЯ НАБОРОВ
*   РЕЗУЛЬТАТОВ
*****
*
    EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
        SELECT NAME FROM SYSIBM.SYSTABLES WHERE CREATOR=:INSCHEMA
    END-EXEC.

*
    LINKAGE SECTION.
*****
*   ОБЪЯВЛЕНИЕ ВХОДНЫХ ПАРАМЕТРОВ ДЛЯ ЭТОЙ ПРОЦЕДУРЫ
*****
01 PROCNM PIC X(18).
01 SCHEMA PIC X(8).

*****
*   ОБЪЯВЛЕНИЕ ВЫХОДНЫХ ПАРАМЕТРОВ ДЛЯ ЭТОЙ ПРОЦЕДУРЫ
*****
01 OUT-CODE PIC S9(9) USAGE BINARY.
01 PARMLST.
    49 PARMLST-LEN PIC S9(4) USAGE BINARY.
    49 PARMLST-TEXT PIC X(254).

PROCEDURE DIVISION USING PROCNM, SCHEMA,
    OUT-CODE, PARMLST.

```

*Рисунок 223 (Часть 1 из 2). Хранимая процедура на языке COBOL, использующая метод передачи параметров GENERAL*

```

*****
* ВЫДАЕТСЯ SQL SELECT ДЛЯ ТАБЛИЦЫ SYSIBM.SYSPROCEDURES
* КАТАЛОГА DB2.
*****
EXEC SQL
  SELECT RUNOPTS INTO :PARMLST
    FROM SYSIBM.ROUTINES
      WHERE NAME=:PROCNM AND
SCHEMA=:SCHEMA
  END-EXEC.

*****
* КОПИРУЕМ SQLCODE В ОБЛАСТЬ ВЫХОДНЫХ ПАРАМЕТРОВ
*****
MOVE SQLCODE TO OUT-CODE.
*****
* ОТКРЫВАЕМ УКАЗАТЕЛЬ C1, ЧТОБЫ DB2 ВЕРНУЛА ВЫЗЫВАЮЩЕЙ
* ПРОГРАММЕ НАБОР РЕЗУЛЬТАТОВ.
*****
EXEC SQL OPEN C1
  END-EXEC.

PROG-END.
GOBACK.

```

*Рисунок 223 (Часть 2 из 2). Хранимая процедура на языке COBOL, использующая метод передачи параметров GENERAL*

## Хранимая процедура на языке COBOL: метод передачи параметров GENERAL WITH NULLS

Эта хранимая процедура примера выполняет следующие действия:

- Ищет в таблице SYSIBM.SYSROUTINES каталога DB2 строки, значения в которых совпадают с входными параметрами, переданными из программы клиента. Эти два входных параметра содержат значения для столбцов NAME и SCHEMA.
- Ищет в таблице SYSTABLES каталога DB2 все таблицы, у которых значения столбца CREATOR совпадает со значением входного параметра SCHEMA. Хранимая процедура использует указатель, чтобы возвратить имена этих таблиц.

Для этой хранимой процедуры используется метод передачи параметров GENERAL WITH NULLS.

Выходные параметры этой хранимой процедуры содержат значение SQLCODE для оператора SELECT и значение столбца RUNOPTS из таблицы SYSIBM.SYSROUTINES.

Оператор CREATE PROCEDURE для этой хранимой процедуры может выглядеть так:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
                           OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
LANGUAGE COBOL
DETERMINISTIC
READS SQL
EXTERNAL NAME 'GETPRML'
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
EXTERNAL SECURITY DB2
RESULT SETS 2
COMMIT ON RETURN NO;
```

```

CBL RENT
    IDENTIFICATION DIVISION.
    PROGRAM-ID. GETPRML.
    AUTHOR. EXAMPLE.
    DATE-WRITTEN. 03/25/98.

    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
        FILE-CONTROL.
        DATA DIVISION.
            FILE SECTION.

/*
    WORKING-STORAGE SECTION.
*/
    EXEC SQL INCLUDE SQLCA END-EXEC.

/*
*****  

*   ОБЪЯВЛЕНИЕ ПЕРЕМЕННОЙ ХОСТА ДЛЯ ХРАНЕНИЯ  

*   ВХОДНОЙ СХЕМЫ  

*****  

01 INSCHEMA PIC X(8).
*****  

*   ОБЪЯВЛЕНИЕ УКАЗАТЕЛЯ ДЛЯ ВОЗВРАЩЕНИЯ НАБОРОВ  

*   РЕЗУЛЬТАТОВ  

*****  

*
    EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
        SELECT NAME FROM SYSIBM.SYSTABLES WHERE CREATOR=:INSCHEMA
        END-EXEC.

/*
    LINKAGE SECTION.
*****  

*   ОБЪЯВЛЕНИЕ ВХОДНЫХ ПАРАМЕТРОВ ДЛЯ ЭТОЙ ПРОЦЕДУРЫ  

*****  

01 PROCNM PIC X(18).
01 SCHEMA PIC X(8).
*****  

*   ОБЪЯВЛЕНИЕ ВЫХОДНЫХ ПАРАМЕТРОВ ДЛЯ ЭТОЙ ПРОЦЕДУРЫ  

*****  

01 OUT-CODE PIC S9(9) USAGE BINARY.
01 PARMST.
    49 PARMLST-LEN PIC S9(4) USAGE BINARY.
    49 PARMLST-TEXT PIC X(254).
*****  

*   ОБЪЯВЛЕНИЕ СТРУКТУРЫ, СОДЕРЖАЩЕЙ ИНДИКАТОРЫ ПУСТЫХ  

*   ЗНАЧЕНИЙ ДЛЯ ВХОДНЫХ И ВЫХОДНЫХ ПАРАМЕТРОВ.  

*****  

01 IND-PARM.
    03 PROCNM-IND PIC S9(4) USAGE BINARY.
    03 SCHEMA-IND PIC S9(4) USAGE BINARY.
    03 OUT-CODE-IND PIC S9(4) USAGE BINARY.
    03 PARMLST-IND PIC S9(4) USAGE BINARY.

```

*Рисунок 224 (Часть 1 из 2). Хранимая процедура на языке COBOL, использующая метод передачи параметров GENERAL WITH NULLS*

```

PROCEDURE DIVISION USING PROCNM, SCHEMA,
OUT-CODE, PARMLST, IND-PARM.
*****
* ЕСЛИ КАКОЙ-ЛИБО ИЗ ВХОДНЫХ ПАРАМЕТРОВ ИМЕЕТ ПУСТОЕ
* ЗНАЧЕНИЕ, ВОЗВРАЩАЕТСЯ ПУСТОЕ ЗНАЧЕНИЕ ДЛЯ PARMLST
* И ДЛЯ ВЫХОДНОГО КОДА ВОЗВРАТА ЗАДАЕТСЯ ЗНАЧЕНИЕ 9999.
*****
IF PROCNM-IND < 0 OR
SCHEMA-IND < 0
MOVE 9999 TO OUT-CODE
MOVE 0 TO OUT-CODE-IND
MOVE -1 TO PARMLST-IND
ELSE
*****
* ВЫДАЕТСЯ SQL SELECT ДЛЯ ТАБЛИЦЫ SYSIBM.SYSROUTINES
* КАТАЛОГА DB2.
*****
EXEC SQL
SELECT RUNOPTS INTO :PARMLST
FROM SYSIBM.SYSROUTINES
WHERE NAME=::PROCNM AND
SCHEMA=::SCHEMA
END-EXEC
MOVE 0 TO PARMLST-IND
*****
* КОПИРУЕМ SQLCODE В ОБЛАСТЬ ВЫХОДНЫХ ПАРАМЕТРОВ
*****
MOVE SQLCODE TO OUT-CODE
MOVE 0 TO OUT-CODE-IND.
*
*****
* ОТКРЫВАЕМ УКАЗАТЕЛЬ C1, ЧТОБЫ DB2 ВЕРНУЛА ВЫЗЫВАЮЩЕЙ
* ПРОГРАММЕ НАБОР РЕЗУЛЬТАТОВ.
*****
EXEC SQL OPEN C1
END-EXEC.
PROG-END.
GOBACK.

```

*Рисунок 224 (Часть 2 из 2). Хранимая процедура на языке COBOL, использующая метод передачи параметров GENERAL WITH NULLS*

## Хранимая процедура на языке PL/I: GENERAL

Этот пример хранимой процедуры ищет в таблице SYSIBM.SYSROUTINES каталога DB2 строки, значения в которых совпадают с входными параметрами, переданными из программы клиента. Эти два входных параметра содержат значения для столбцов NAME и SCHEMA.

Для этой хранимой процедуры используется метод передачи параметров GENERAL.

Выходные параметры этой хранимой процедуры содержат значение SQLCODE для оператора SELECT и значение столбца RUNOPTS из таблицы SYSIBM.SYSROUTINES.

Оператор CREATE PROCEDURE для этой хранимой процедуры может выглядеть так:

```

CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
                        OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
LANGUAGE PLI
DETERMINISTIC
READS SQL
EXTERNAL NAME 'GETPRML'
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
EXTERNAL SECURITY DB2
RESULT SETS 0
COMMIT ON RETURN NO;

*PROCESS SYSTEM(MVS);

GETPRML:
PROC(PROCNM, SCHEMA, OUT_CODE, PARMLST)
OPTIONS(MAIN NOEXECOPS REENTRANT);

DECLARE PROCNM CHAR(18),      /* Входной параметр -- имя процедуры */
       SCHEMA CHAR(8),      /* Входной параметр -- пользовательская схема */

       OUT_CODE FIXED BIN(31), /* Выходное значение -- SQLCODE */
                           /* от оператора SELECT. */
       PARMLST CHAR(254)    /* Выходные значения -- RUNOPTS */
                           /* VARYING;           для поиска строк в таблице */
                           /*                   SYSIBM.SYSROUTINES */
                           /* */

EXEC SQL INCLUDE SQLCA;

/*****************************************/
/* Выполнение оператора SELECT для таблицы каталога          */
/* SYSIBM.SYSROUTINES.                                         */
/*****************************************/
EXEC SQL
SELECT RUNOPTS INTO :PARMLST
   FROM SYSIBM.SYSROUTINES
 WHERE NAME=:PROCNM AND
       SCHEMA=:SCHEMA;

OUT_CODE = SQLCODE; /* Вернуть SQLCODE вызывающей программе */
RETURN;
END GETPRML;

```

*Рисунок 225. Хранимая процедура на языке PL/I, использующая метод передачи параметров GENERAL*

## **Хранимая процедура на языке PL/I: метод передачи параметров GENERAL WITH NULLS**

Этот пример хранимой процедуры ищет в таблице SYSIBM.SYSROUTINES каталога DB2 строки, значения в которых совпадают с входными параметрами, переданными из программы клиента. Эти два входных параметра содержат значения для столбцов NAME и SCHEMA.

Для этой хранимой процедуры используется метод передачи параметров GENERAL WITH NULLS.

Выходные параметры этой хранимой процедуры содержат значение SQLCODE для оператора SELECT и значение столбца RUNOPTS из таблицы SYSIBM.SYSROUTINES.

Оператор CREATE PROCEDURE для этой хранимой процедуры может выглядеть так:

```
CREATE PROCEDURE GETPRML(PROCNM CHAR(18) IN, SCHEMA CHAR(8) IN,
                           OUTCODE INTEGER OUT, PARMLST VARCHAR(254) OUT)
  LANGUAGE PLI
  DETERMINISTIC
  READS SQL
  EXTERNAL NAME 'GETPRML'
  COLLID GETPRML
  ASUTIME NO LIMIT
  PARAMETER STYLE GENERAL WITH NULLS
  STAY RESIDENT NO
  RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
  WLM ENVIRONMENT SAMPPROG
  PROGRAM TYPE MAIN
  EXTERNAL SECURITY DB2
  RESULT SETS 0
  COMMIT ON RETURN NO;
```

```

*PROCESS SYSTEM(MVS);

GETPRML:
  PROC(PROCNM, SCHEMA, OUT_CODE, PARMLST, INDICATORS)
    OPTIONS(MAIN NOEXECOPS REENTRANT);

  DECLARE PROCNM CHAR(18),      /* Входной параметр -- имя процедуры */
         SCHEMA CHAR(8),      /* Входной параметр -- пользовательская схема */

         OUT_CODE FIXED BIN(31), /* Выходное значение -- SQLCODE */
                           /* от оператора SELECT. */
         PARMLST CHAR(254)      /* Выходные значения -- RUNOPTS */
                               /* VARYING;           для поиска строк в таблице */
                               /*                   SYSIBM.SYSPROCEDURES */
  DECLARE 1 INDICATORS,        /* Объявление индикаторов пустых */
          /* значений для входных и выходных */
          /* параметров. */
  3 PROCNM_IND   FIXED BIN(15),
  3 SCHEMA_IND   FIXED BIN(15),
  3 OUT_CODE_IND FIXED BIN(15),
  3 PARMLST_IND  FIXED BIN(15);

  EXEC SQL INCLUDE SQLCA;

  IF PROCNM_IND<0 |
     SCHEMA_IND<0 THEN
    DO;                      /* Если какой-либо из входных параметров */
                           /* имеет пустое значение: */
     OUT_CODE = 9999;        /* Задать выходной код возврата. */
     OUT_CODE_IND = 0;       /* Выходной код возврата имеет непустое */
                           /* значение. */
     PARMLST_IND = -1;      /* Задать пустое значение для PARMLST. */
  END;
  ELSE                     /* Если входные параметры имеют непустые */
                           /* значения: */
    DO;                      /* */
  /*************************************************************************/
  /* Выдать оператор SQL SELECT для таблицы SYSIBM.SYSROUTINES */
  /* каталога DB2. */
  /*************************************************************************/
  EXEC SQL
    SELECT RUNOPTS INTO :PARMLST
      FROM SYSIBM.SYSROUTINES
      WHERE NAME=:PROCNM AND
            SCHEMA=:SCHEMA;
    PARMLST_IND = 0;        /* Задать, что PARMLST имеет непустое */
                           /* значение. */
                           /* Вернуть SQLCODE вызывающей программе */
    OUT_CODE = SQLCODE;    /* */
    OUT_CODE_IND = 0;       /* Выходной код возврата имеет непустое */
                           /* значение. */
  END;
  RETURN;

END GETPRML;

```

Рисунок 226. Хранимая процедура на языке PL/I, использующая метод передачи параметров GENERAL WITH NULLS



## Приложение Е. Подкоманды REBIND для списков планов или пакетов

Если список пакетов или планов, который требуется повторно связать, сложно задать с помощью звездочек, возможно, вам удастся создать необходимые подкоманды REBIND автоматически с помощью программы примера DSNTIAUL.

В частности, этот метод может оказаться полезным, если во время повторного связывания большого числа планов или пакетов стал недоступным какой-либо ресурс. Обычно DB2 прекращает повторное связывание и не связывает оставшиеся планы или пакеты. После этого, однако, имеет смысл выполнить повторное связывание только для оставшихся несвязанными объектов. Можно построить подкоманды REBIND для оставшихся планов или пакетов, с помощью программы DSNTIAUL выбирая планы или пакеты из каталога DB2 и создавая подкоманды REBIND. Затем можно, как обычно, передать подкоманды на выполнение командному процессору DSN.

Возможно, прежде чем полученные от DSNTIAUL данные можно будет передать на вход DSN, их придется отредактировать. Большую часть работы для вас может выполнить команда CLIST DSNTEdit.

В этом разделе рассматриваются следующие темы:

- Обзор процедуры генерации списка команд REBIND
- “Примеры операторов SELECT для генерации команд REBIND” на стр. 958
- “Пример JCL для запуска списков команд REBIND” на стр. 961

### Обзор процедуры генерации списка команд REBIND

На рис. 227 показана последовательность действий при использовании команд REBIND PLAN и REBIND PACKAGE.

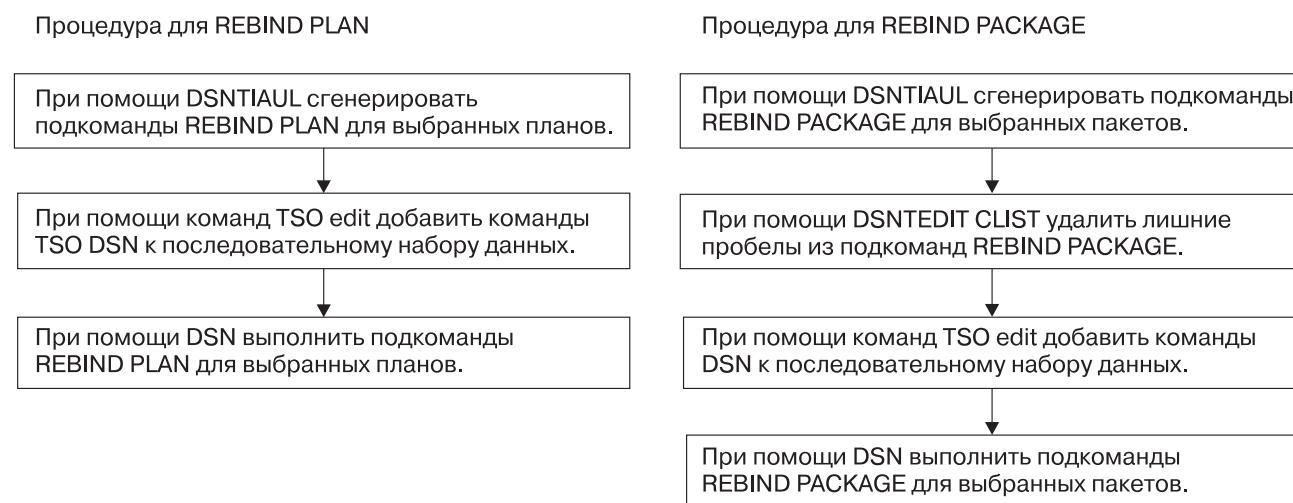


Рисунок 227. Процедура выполнения списков команд REBIND

---

## Примеры операторов SELECT для генерации команд REBIND

**Построение подкоманд REBIND:** В приводимых ниже примерах показаны следующие приемы:

- Использование оператора SELECT для выбора отдельных пакетов или планов, подлежащих повторному связыванию
- Использование операции CONCAT для соединения команды REBIND с несколькими именами планов или пакетов
- Использование функции SUBSTR для преобразования строки переменной длины в строку постоянной длины
- Добавление пустых символов к подкомандам REBIND PLAN и REBIND PACKAGE с целью получения длины записи, приемлемой для командного процессора DSN

**Если оператор SELECT возвращает строки,** DSNTIAUL формирует подкоманды REBIND для планов и пакетов, указанных в возвращенных строках. Поместите эти подкоманды в последовательный набор данных, где их можно будет редактировать.

Для подкоманд REBIND PACKAGE удалите все лишние пробелы из имени пакета либо с помощью команды редактирования TSO, либо с помощью команды DB2 CLIST DSNTEDIT.

И для подкоманды REBIND PLAN, и для подкоманды REBIND PACKAGE добавьте с помощью команд редактирования TSO команду DSN как первую строку последовательного набора данных оператора и команду END как последнюю строку. После редактирования последовательный набор данных можно запустить для повторного связывания выбранных планов или пакетов.

**Если оператор SELECT не возвращает подходящих строк,** DSNTIAUL не формирует подкоманды REBIND.

В примерах данного раздела формируются подкоманды REBIND для DB2 for OS/390 Версия 6. Возможно, для более ранних выпусков DB2 синтаксис будет несколько другим, и в примеры придется внести изменения.

**Пример 1:** Выполнить повторное связывание (REBIND) всех планов, игнорируя недоступность ресурсов.

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME  
          CONCAT')' ,1,45)  
FROM SYSIBM.SYSPLAN;
```

**Пример 2:** Выполнить повторное связывание (REBIND) всех версий пакетов, игнорируя недоступность ресурсов.

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.  
          CONCAT NAME CONCAT'.*)' ,1,55)  
FROM SYSIBM.SYSPACKAGE;
```

**Пример 3:** Выполнить повторное связывание (REBIND) всех планов, временная отметка которых раньше заданной.

```

SELECT SUBSTR('REBIND PLAN('CONCAT NAME
               CONCAT')
               ',1,45)
FROM SYSIBM.SYSPLAN
WHERE BINDDATE <= 'ггггммдд' AND
      BINDTIME <= 'ччммссдс';

```

где *ггггммдд* – дата, а *ччммссдс* – время из строки отметки времени.

В значении даты 'гггг' означает четыре цифры года из строки отметки времени. В отметке времени 'дс' – необязательная часть, соответствующая долям секунды, где 'д' – десятые секунды, а 'с' – сотые секунды.

**Пример 4:** Выполнить повторное связывание (REBIND) всех пакетов, связанных ранее определенной даты и времени.

```

SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.
               CONCAT NAME CONCAT').(*)',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME <= 'отметка_времени';

```

где *отметка\_времени* – строка отметки времени по стандарту ISO.

**Пример 5:** Выполнить повторное связывание (REBIND) всех планов, начиная с определенной даты и времени.

```

SELECT SUBSTR('REBIND PLAN('CONCAT NAME
               CONCAT')
               ',1,45)
FROM SYSIBM.SYSPLAN
WHERE BINDDATE >= 'ггггммдд' AND
      BINDTIME <= 'ччммссдс';

```

где *ггггммдд* – дата, а *ччммссдс* – время из строки отметки времени.

В значении даты 'гггг' означает четыре цифры года из строки отметки времени. В отметке времени 'дс' – необязательная часть, соответствующая долям секунды, где 'д' – десятые секунды, а 'с' – сотые секунды.

**Пример 6:** Выполнить повторное связывание (REBIND) всех пакетов, связанных после определенной даты и времени.

```

SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID
               CONCAT'.'CONCAT NAME
               CONCAT').(*)',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME <= 'отметка_времени';

```

где *отметка\_времени* – строка отметки времени по стандарту ISO.

**Пример 7:** Выполнить повторное связывание (REBIND) всех планов в определенном диапазоне дат и времени.

```

        SELECT SUBSTR('REBIND PLAN('CONCAT NAME
                      CONCAT')',1,45)
        FROM SYSIBM.SYSPLAN
       WHERE
          (BINDDATE >= 'ггггммдд' AND
           BINDTIME >= 'ччммссдс') AND
          BINDDATE <= 'ггггммдд' AND
          BINDTIME <= 'ччммссдс');

```

где *ггггммдд* – дата, а *ччммссдс* – время из строки отметки времени.

В значении даты 'гггг' означает четыре цифры года из строки отметки времени. В отметке времени 'дс' – необязательная часть, соответствующая долям секунды, где 'д' – десятые секунды, а 'с' – сотые секунды.

**Пример 8:** Выполнить повторное связывание (REBIND) всех пакетов, связанных в определенном диапазоне дат и времени.

```

        SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
                      CONCAT NAME CONCAT').(*)',1,55)
        FROM SYSIBM.SYSPACKAGE
       WHERE BINDTIME >= 'timestamp1' AND
             BINDTIME <= 'timestamp2';

```

где *отметка\_времени1* и *отметка\_времени2* – строки отметок времени по стандарту ISO.

**Пример 9:** Выполнить повторное связывание (REBIND) всех дефектных планов.

```

        SELECT SUBSTR('REBIND PLAN('CONCAT NAME
                      CONCAT')',1,45)
        FROM SYSIBM.SYSPLAN
       WHERE VALID = 'N';

```

**Пример 10:** Выполнить повторное связывание (REBIND) всех дефектных версий пакетов.

```

        SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
                      CONCAT NAME CONCAT').(*)',1,55)
        FROM SYSIBM.SYSPACKAGE
       WHERE VALID = 'N';

```

**Пример 11:** Выполнить повторное связывание (REBIND) всех планов с уровнем изоляции "стабильность на уровне указателя".

```

        SELECT SUBSTR('REBIND PLAN('CONCAT NAME
                      CONCAT')',1,45)
        FROM SYSIBM.SYSPLAN
       WHERE ISOLATION = 'S';

```

**Пример 12:** Выполнить повторное связывание (REBIND) всех версий пакетов, допускающих параллелизм использования процессора и ввода–вывода.

```

        SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
                      CONCAT NAME CONCAT').(*)',1,55)
        FROM SYSIBM.SYSPACKAGE
       WHERE DEGREE='ANY';

```

---

## Пример JCL для запуска списков команд REBIND

На рис. 228 приводится задание JCL, выполняющее повторное связывание всех версий всех пакетов, связанных в 1994 году.

На рис. 229 на стр. 963 приводится пример задания JCL, выполняющего повторное связывание с DEGREE(ANY) всех планов, связанных без указания ключевого слова DEGREE.

---

```
//REBINDS JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,USER=SYSADM,
//                      REGION=1024K
//*****                                                 *****/
//SETUP   EXEC PGM=IKJEFT01
//SYSTSPPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSNТИAUL) PLAN(DSNТИB61) PARM('SQL') -
      LIB('DSN610.RUNLIB.LOAD')
END
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSREC00 DD DSN=SYSADM.SYSTSIN.DATA,
//                      UNIT=SYSDA,DISP=SHR
```

---

*Рисунок 228 (Часть 1 из 2). Пример JCL: Повторное связывание всех пакетов, связанных в 1994 году.*

---

```

//*****
//*
//** GENER= '<ПОДКОМАНДЫ ПОВТОРНОГО СВЯЗЫВАНИЯ ВСЕХ ПАКЕТОВ, СВЯЗАННЫХ В 1994 ГОДУ
//*
//*****
//SYSIN DD *
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.'
    CONCAT NAME CONCAT'.*)')           ',1,55)
    FROM SYSIBM.SYSPACKAGE
    WHERE BINDTIME >= '1994-01-01-00.00.00.000000' AND
        BINDTIME <= '1994-12-31-23.59.59.999999';
/*
//*****
//*
//** ИСКЛЮЧАЕМ ПРОБЕЛЫ ИЗ ПОДКОМАНД ПОВТОРНОГО СВЯЗЫВАНИЯ
//*
//*****
//STRIP    EXEC PGM=IKJEFT01
//SYSPROC  DD DSN=SYSADM.DSNCLIST,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN DD *
DSNTEdit SYSADM.SYSTSIN.DATA
//SYSIN   DD DUMMY
/*
//*****
//*
//** ВСТАВКА ОПЕРАТОРОВ КОМАНДЫ DSN
//*
//*****
//EDIT    EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
    EDIT 'SYSADM.SYSTSIN.DATA' DATA NONUM
    TOP
    INSERT DSN SYSTEM(DSN)
    BOTTOM
    INSERT END
    TOP
    LIST * 99999
    END SAVE
/*
//*****
//*
//** ВЫПОЛНЯЕМ ПОДКОМАНДЫ ПОВТОРНОГО СВЯЗЫВАНИЯ ПАКЕТОВ ЧЕРЕЗ DSN
//*
//*****
//LOCAL    EXEC PGM=IKJEFT01
//DBRMLIB  DD DSN=DSN610.DBRLIB.DATA,
//          DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD DSN=SYSADM.SYSTSIN.DATA,
//          UNIT=SYSDA,DISP=SHR
/*

```

---

*Рисунок 228 (Часть 2 из 2). Пример JCL: Повторное связывание всех пакетов, связанных в 1994 году.*

---

```

//REBINDS JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,USER=SYSADM,
//          REGION=1024K
//*****
//SETUP EXEC TSOBATCH
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSREC00 DD DSN=SYSADM.SYSTSIN.DATA,
//          UNIT=SYSDA,DISP=SHR
//*****
//*
//** ПОВТОРНОЕ СВЯЗЫВАНИЕ С ОПЦИЕЙ DEGREE(ANY) ВСЕХ ПЛАНОВ, КОТОРЫЕ БЫЛИ
//** СВЯЗАНЫ БЕЗ УКАЗАНИЯ КЛЮЧЕВОГО СЛОВА DEGREE
//*
//*****
//SYSTSIN DD *
  DSN S(DSN)
  RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB61) PARM('SQL')
  END
//SYSIN DD *
  SELECT SUBSTR('REBIND PLAN('CONCAT NAME
                CONCAT')' DEGREE(ANY)      ',1,45)
    FROM SYSIBM.SYSPLAN
    WHERE DEGREE = ' ';
/*
//*****
//*
//** ВСТАВКА ОПЕРАТОРОВ КОМАНДЫ DSN
//*
//*****
//EDIT EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  EDIT 'SYSADM.SYSTSIN.DATA' DATA NONUM
  TOP
  INSERT DSN S(DSN)
  BOTTOM
  INSERT END
  TOP
  LIST * 99999
  END SAVE
/*
//*****
//*
//** ВЫПОЛНЯЕМ ПОДКОМАНДЫ ПОВТОРНОГО СВЯЗЫВАНИЯ ЧЕРЕЗ DSN
//*
//*****
//REBIND EXEC PGM=IKJEFT01
//STEPLIB  DD DSN=SYSADM.TESTLIB,DISP=SHR
//          DD DSN=DSN610.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=SYSADM.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN  DD DSN=SYSADM.SYSTSIN.DATA,DISP=SHR
//SYSIN   DD DUMMY
/*

```

---

*Рисунок 229. Пример JCL: Повторное связывание выбранных планов с другой опцией связывания*



---

## Приложение F. Зарезервированные слова SQL

В Табл. 127 на стр. 966 перечислены слова, которые нельзя использовать как обычные идентификаторы в некоторых контекстах, поскольку они могут быть проинтерпретированы как ключевые слова SQL. Например, нельзя использовать ALL как имя столбца в операторе SELECT. Однако каждое слово можно использовать как обычный идентификатор в контексте, где его нельзя ставить непосредственно, если заключить его в символы—ограничители. Например, если как выделенный символ, который начинает и завершает ограниченные идентификаторы, используются кавычки ("), "ALL" можно использовать как имя столбца в операторе SELECT. Кроме того, в некоторых разделах данной книги могут быть указаны слова, которые нельзя использовать в определенных описанных контекстах.

Таблица 127. Зарезервированные слова SQL

ADD	CURRENT_TIMESTAMP	HAVING	OF	SIMPLE
AFTER	CURSOR	HOUR	ON	SOME
ALL	DATA	HOURS	OPTIMIZATION	SOURCE
ALLOW	DATABASE	IMMEDIATE	OPTIMIZE	SPECIFIC
ALTER	DAY	IN	OR	STANDARD
AND	DAYS	INDEX	ORDER	STAY
ANY	DBINFO	INNER	OUT	STOGROUP
AS	DB2SQL	INOUT	OUTER	STORES
ASUTIME	DEFAULT	INSERT	PACKAGE	STYLE
AUDIT	DELETE	INTO	PARAMETER	SUBPAGES
AUX	DESCRIPTOR	IS	PART	SYNONYM
AUXILIARY	DETERMINISTIC	ISOBid	PATH	SYSFUN
BEFORE	DISALLOW	JOIN	PIECESIZE	SYSIBM
BEGIN	DISTINCT	KEY	PLAN	SYSPROC
BETWEEN	DOUBLE	LANGUAGE	PRECISION	SYSTEM
BUFFERPOOL	DROP	LC_CTYPE	PRIQTY	TABLE
BY	DSSIZE	LEFT	PRIVILEGES	TABLESPACE
CALL	EDITPROC	LIKE	PROCEDURE	THEN
CAPTURE	ELSE	LOCAL	PROGRAM	TO
CASCADED	END	LOCALE	PSID	TRIGGER
CASE	END-EXEC1	LOCATOR	QUERYNO	TYPE
CAST	ERASE	LOCATORS	READS	UNION
CCSID	ESCAPE	LOCKMAX	REFERENCES	UNIQUE
CHAR	EXCEPT	LOCKSIZE	RENAME	UPDATE
CHARACTER	EXISTS	LONG	RESTRICT	USER
CHECK	EXTERNAL	MICROSECOND	RESULT	USING
CLUSTER	FENCED	MICROSECONDS	RETURN	VALIDPROC
COLLECTION	FIELDPROC	MINUTE	RETURNS	VALUES
COLLID	FINAL	MINUTES	RIGHT	VARIANT
COLUMN	FOR	MODIFIES	RUN	VCAT
CONCAT	FROM	MONTH	SCHEMA	VIEW
CONNECTION	FULL	MONTHS	SCRATCHPAD	VOLUMES
CONSTRAINT	FUNCTION	NAME	SECOND	WHEN
CONTAINS	GENERAL	NO	SECONDS	WHERE
CURRENT	GENERATED	NOT	SEQQTY	WITH
CURRENT_DATE	GO	NULL	SECURITY	WLM
CURRENT_LC_CTYPE	GOTO	NULLS	SELECT	YEAR
CURRENT_PATH	GRANT	NUMPARTS	SET	YEARS
CURRENT_TIME	GROUP	OBID		

Примечание: <sup>1</sup>Только для языка COBOL

В IBM SQL есть дополнительные зарезервированные слова, использование которых в DB2 for OS/390 не ограничивается. Мы рекомендуем не использовать эти дополнительные зарезервированные слова как обычные идентификаторы в именах, которые вы собираетесь использовать далее. Список этих слов приводится в справочнике *IBM SQL Reference*.

## Приложение G. Характеристики операторов SQL в DB2 for OS/390

В этом приложении приводится сводка действий, разрешенных для операторов SQL в DB2 for OS/390. В нем приводится также список операторов SQL, которые можно выполнять во внешних пользовательских функциях и хранимых процедурах.

### Разрешенные действия для операторов SQL

В Табл. 128 показано, можно ли конкретный оператор DB2 выполнить, подготовить интерактивно или динамически, или обработать реквестером прикладных программ или препроцессором. Буква **Д** в таблице означает да.

Таблица 128 (Стр. 1 из 3). Разрешенные действия для операторов SQL в DB2 for OS/390

Оператор SQL	Выполнение	Интеракт. или динамич. подготовка		Обработка	
		системой реквестера	сервером	препроцессором	препроцессором
ALLOCATE CURSOR <sup>1</sup>	Д	Д	Д		
ALTER <sup>2</sup>	Д	Д		Д	
ASSOCIATE LOCATORS <sup>1</sup>	Д	Д	Д		
BEGIN DECLARE SECTION					Д
CALL <sup>1</sup>	Д			Д	
CLOSE	Д			Д	
COMMENT ON	Д	Д		Д	
COMMIT	Д	Д		Д	
CONNECT (Type 1 and Type 2)	Д		Д		
CREATE <sup>2</sup>	Д	Д		Д	
DECLARE CURSOR					Д
DECLARE STATEMENT					Д
DECLARE TABLE					Д
DELETE	Д	Д		Д	
DESCRIBE	Д			Д	
DESCRIBE CURSOR	Д		Д		
DESCRIBE INPUT	Д			Д	
DESCRIBE PROCEDURE	Д		Д		
DROP <sup>2</sup>	Д	Д		Д	
END DECLARE SECTION					Д
EXECUTE	Д			Д	
EXECUTE IMMEDIATE	Д			Д	
EXPLAIN	Д	Д		Д	
FETCH	Д			Д	

Таблица 128 (Стр. 2 из 3). Разрешенные действия для операторов SQL в DB2 for OS/390

Оператор SQL	Выполнение	Интеракт. или динамич. подготовка	системой реквестера	Обработка	
				сервером	прекомпил.
FREE LOCATOR <sup>1</sup>	Д	Д		Д	
GRANT <sup>2</sup>	Д	Д		Д	
HOLD LOCATOR <sup>1</sup>	Д	Д		Д	
INCLUDE					Д
INSERT	Д	Д		Д	
LABEL ON	Д	Д		Д	
LOCK TABLE	Д	Д		Д	
OPEN	Д			Д	
PREPARE	Д			Д <sup>4</sup>	
RELEASE	Д		Д		
RENAME <sup>2</sup>	Д	Д		Д	
REVOKE <sup>2</sup>	Д	Д		Д	
ROLLBACK	Д	Д		Д	
SELECT INTO	Д			Д	
SET CONNECTION	Д		Д		
SET CURRENT DEGREE	Д	Д		Д	
SET CURRENT LC_CTYPE	Д	Д		Д	
SET CURRENT OPTIMIZATION	Д	Д		Д	
HINT					
SET CURRENT PACKAGESET	Д		Д		
SET CURRENT PATH	Д	Д		Д	
SET CURRENT PRECISION	Д	Д		Д	
SET CURRENT RULES	Д	Д		Д	
SET CURRENT SQLID <sup>5</sup>	Д	Д		Д	
SET переменная_хоста = CURRENT DATE	Д			Д	
SET переменная_хоста = CURRENT DEGREE	Д			Д	
SET переменная_хоста = CURRENT PACKAGESET	Д		Д		
SET переменная_хоста = CURRENT PATH	Д			Д	
SET переменная_хоста = CURRENT QUERY	Д			Д	
OPTIMIZATION LEVEL					
SET переменная_хоста = CURRENT SERVER	Д		Д		
SET переменная_хоста = CURRENT SQLID	Д			Д	

Таблица 128 (Стр. 3 из 3). Разрешенные действия для операторов SQL в DB2 for OS/390

Оператор SQL	Выполнение	Интеракт. или динамич. подготовка	Обработка	
		системой реквестера	сервером	прекомпил.
SET переменная_хоста = CURRENT TIME	Д		Д	
SET переменная_хоста = CURRENT TIMESTAMP	Д		Д	
SET переменная_хоста = CURRENT TIMEZONE	Д		Д	
SIGNAL SQLSTATE <sup>6</sup>	Д		Д	
UPDATE	Д	Д	Д	
VALUES <sup>6</sup>	Д		Д	
VALUES INTO	Д		Д	
WHENEVER				

**Примечания:**

1. Этот оператор можно подготовить динамически. Однако его нельзя выполнить динамически.
2. Этот оператор можно подготовить динамически, только если явно или неявно задано поведение выполнения DYNAMICRULES.
3. Этот оператор можно подготовить динамически, но только из драйвера ODBC или CLI, который поддерживает динамические операторы CALL.
4. Система реквестера обрабатывает оператор PREPARE, когда подготавливаемый оператор – ALLOCATE CURSOR или ASSOCIATE LOCATORS.
5. Значение, заданное для специального регистра CURRENT SQLID, используется как ID авторизации SQL и неявный спецификатор для динамических операторов SQL, только когда действует поведение выполнения DYNAMICRULES. При других поведениях DYNAMICRULES значение CURRENT SQLID игнорируется.
6. Этот оператор может использоваться только в действии триггера.

## Операторы SQL, допустимые в функциях и в хранимых процедурах

В Табл. 129 показано, какие операторы SQL можно выполнять в хранимой процедуре или во внешней пользовательской функции. Операторы, которые можно выполнить, зависят от уровня доступа к данным SQL, с которым определена хранимая процедура или внешняя функция (NO SQL, CONTAINS SQL, READS SQL DATA или MODIFIES SQL DATA). Буква **Д** в таблице означает да.

Обычно если исполняемый оператор SQL встречается в хранимой процедуре или функции, определенной как NO SQL, возвращается SQLSTATE 38001. Если для программы определен некоторый уровень доступа SQL, операторы SQL, не поддерживаемые ни в каком контексте, возвращают SQLSTATE 38003. Операторы SQL, не разрешенные для программ, определенных как CONTAINS SQL, возвращают SQLSTATE 38004, а операторы SQL, не разрешенные для READS SQL DATA, возвращают SQL STATE 38002.

Таблица 129 (Стр. 1 из 3). Операторы SQL во внешних пользовательских функциях и хранимых процедурах

Оператор SQL	Уровень доступа SQL			
	NO SQL	CONTAINS SQL	READS SQL DATA	MODIFIES SQL DATA
ALLOCATE CURSOR			Д	Д
ALTER				Д
ASSOCIATE LOCATORS			Д	Д
BEGIN DECLARE SECTION	Д <sup>1</sup>	Д	Д	Д
CALL		Д <sup>2</sup>	Д <sup>2</sup>	Д <sup>2</sup>
CLOSE			Д	Д
COMMENT ON				Д
COMMIT				
CONNECT (Тип 1 и Тип 2)				
CREATE				Д
DECLARE CURSOR	Д <sup>1</sup>	Д	Д	Д
DECLARE GLOBAL TEMPORARY TABLE		Д	Д	Д
DECLARE STATEMENT	Д <sup>1</sup>	Д	Д	Д
DECLARE TABLE	Д <sup>1</sup>	Д	Д	Д
DELETE				Д
DESCRIBE			Д	Д
DESCRIBE CURSOR			Д	Д
DESCRIBE INPUT			Д	Д
DESCRIBE PROCEDURE			Д	Д
DROP				Д
END DECLARE SECTION	Д <sup>1</sup>	Д	Д	Д
EXECUTE		Д <sup>3</sup>	Д <sup>3</sup>	Д
EXECUTE IMMEDIATE		Д <sup>3</sup>	Д <sup>3</sup>	Д

Таблица 129 (Стр. 2 из 3). Операторы SQL во внешних пользовательских функциях и хранимых процедурах

Оператор SQL	Уровень доступа SQL			
	NO SQL	CONTAINS SQL	READS SQL DATA	MODIFIES SQL DATA
EXPLAIN			д	
FETCH			д	д
FREE LOCATOR		д	д	д
GRANT				д
HOLD LOCATOR		д	д	д
INCLUDE	д <sup>1</sup>	д	д	д
INSERT				д
LABEL ON				д
LOCK TABLE		д	д	д
OPEN			д	д
PREPARE		д	д	д
RELEASE				
RENAME				д
REVOKE				д
ROLLBACK				
SELECT			д	д
SELECT INTO			д	д
SET Назначение		д <sup>4</sup>	д	д
SET CONNECTION				
SET специальный регистр		д	д	д
SIGNAL SQLSTATE		д	д	д
UPDATE				д
VALUES			д	д
VALUES INTO		д <sup>4</sup>	д	д
WHENEVER	д <sup>1</sup>	д	д	д

*Таблица 129 (Стр. 3 из 3). Операторы SQL во внешних пользовательских функциях и хранимых процедурах*

Оператор SQL	Уровень доступа SQL		
	NO SQL	CONTAINS SQL	READS SQL DATA

**Примечания:**

- Хотя опция SQL предполагает, что операторы SQL не заданы, использование невыполнимых операторов не ограничивается.
- Вызываемая хранимая процедура должна использовать тот же уровень доступа к данным SQL, что и текущий действующий уровень, или более строгий. Например, программа, определенная как MODIFIES SQL DATA, может вызвать хранимую процедуру, определенную как MODIFIES SQL DATA, READS SQL DATA или CONTAINS SQL. Программа, определенная как CONTAINS SQL, может вызвать только процедуру, определенную как CONTAINS SQL.
- Оператор, заданный в операторе EXECUTE, должен быть оператором, который разрешен для действующего уровня доступа к данным SQL. Например, если действующий уровень – READS SQL DATA, не допускаются операторы INSERT, UPDATE или DELETE.
- Этот оператор поддерживается, только если он не содержит подзапроса или выражения запроса.

## Приложение Н. Опции подготовки программ для удаленных пакетов

В приводимой ниже таблице даны общие описания опций подготовки программ, перечислены эквивалентные опции DB2 для каждой из них и, если это применимо, указано, являются ли они опциями связывания пакета (С) или опциями препроцессора (П). Кроме того, в таблице указано, поддерживает ли сервер DB2 данную опцию.

Таблица 130 (Стр. 1 из 3). Опции подготовки программ для пакетов

Общее описание опции	Эквивалент для DB2 реввестера	Опция связывания или препр.ил.	Поддержка сервером DB2
Замена пакета: защита существующих пакетов	ACTION(ADD)	С	Поддерживается
Замена пакета: замена существующих пакетов	ACTION(REPLACE)	С	Поддерживается
Замена пакетов: имя версии	ACTION(REPLACE REPLVER (идентификатор версии))	С	Поддерживается
Ограничитель строки оператора	APOSTSQL/QUOTESQL	П	Поддерживается
Доступ DRDA: SQL CONNECT (тип 1)	CONNECT(1)	П	Поддерживается
Доступ DRDA: SQL CONNECT (тип 2)	CONNECT(2)	П	Поддерживается
Протокол блокировки: Не блокировать данные для неоднозначного указателя	CURRENTDATA(YES)	С	Поддерживается
Протокол блокировки: Блокировать данные при возможности	CURRENTDATA(NO)	С	Поддерживается
Протокол блокировки: Никогда не блокировать данные	(Недоступна)		Не поддерживается
Имя удаленной базы данных	CURRENTSERVER(имя положения)	С	Поддерживается как опция BIND PLAN
Формат даты оператора	DATE	П	Поддерживается
Протокол для удаленного доступа	DBPROTOCOL	С	Не поддерживается
Максимальное число десятичных разрядов: 15	DEC(15)	П	Поддерживается
Максимальное число десятичных разрядов: 31	DEC(31)	П	Поддерживается
Отложить подготовку динамического SQL	DEFER(PREPARE)	С	Поддерживается
Не откладывать подготовку динамического SQL	NODEFER(PREPARE)	С	Поддерживается
Авторизация динамического SQL	DYNAMICRULES	С	Поддерживается

Таблица 130 (Стр. 2 из 3). Опции подготовки программ для пакетов

Общее описание опции	Эквивалент для DB2 реквестера	Опция связывания или прекомпил.	Поддержка сервером DB2
Опция объяснения	EXPLAIN	C	Поддерживается
Немедленно записать разделы или наборы независимых от пула буферов страниц групп в среде совместного использования данных.	IMMEDWRITE	C	Поддерживается
Уровень изоляции пакета: CS	ISOLATION(CS)	C	Поддерживается
Уровень изоляции пакета: RR	ISOLATION(RR)	C	Поддерживается
Уровень изоляции пакета: RS	ISOLATION(RS)	C	Поддерживается
Уровень изоляции пакета: UR	ISOLATION(UR)	C	Поддерживается
Сохранять подготовленные операторы после точек принятия	KEEPDYNAMIC	C	Поддерживается
Элемент согласованности	LEVEL	П	Поддерживается
Имя пакета	MEMBER	C	Поддерживается
Владелец пакета	OWNER	C	Поддерживается
Список схематических имен для пользовательских функций, определенных типов и хранимых процедур	PATH	C	Поддерживается
Разделитель целой и дробной частей	PERIOD/COMMA	П	Поддерживается
Спецификатор по умолчанию	QUALIFIER	C	Поддерживается
Использовать подсказки для путей доступа	OPTHINT	C	Поддерживается
Опция снятия блокировки	RELEASE	C	Поддерживается
Выбирать путь доступа во время выполнения	REOPT(VARS)	C	Поддерживается
Выбирать путь доступа только во время связывания	NOREOPT(VARS)	C	Поддерживается
Управление созданием: создавать пакет, несмотря на ошибки	SQLERROR(CONTINUE)	C	Поддерживается
Управление созданием: не создавать пакет при наличии ошибок	SQLERROR(NO PACKAGE)	C	Поддерживается
Управление созданием: не создавать пакет	(Недоступна)		Поддерживается
Формат времени оператора	TIME	П	Поддерживается
Проверка существования: полная	VALIDATE(BIND)	C	Поддерживается
Проверка существования: отложенная	VALIDATE(RUN)	C	Поддерживается

Таблица 130 (Стр. 3 из 3). Опции подготовки программ для пакетов

Общее описание опции	Эквивалент для DB2 реквестера	Опция связывания или прекомпил.	Поддержка сервером DB2
Версия пакета	VERSION	П	Поддерживается
Подтип символов по умолчанию: используемый системой по умолчанию	(Недоступна)		Поддерживается
Подтип символов по умолчанию: BIT	(Недоступна)		Не поддерживается
Подтип символов по умолчанию: SBCS	(Недоступна)		Не поддерживается
Подтип символов по умолчанию: DBCS	(Недоступна)		Не поддерживается
CCSID символов по умолчанию: SBCS	(Недоступна)		Не поддерживается
CCSID символов по умолчанию: Смешанный	(Недоступна)		Не поддерживается
CCSID символов по умолчанию: Графический	(Недоступна)		Не поддерживается
Метка пакета	(Недоступна)		Игнорируется при получении; ошибка не возвращается
Наследование привилегий: сохранять	по умолчанию		Поддерживается
Наследование привилегий: аннулировать	(Недоступна)		Не поддерживается



---

## Приложение I. Замечания

Эта информация описывает продукты и услуги, доступные в США. IBM может не предлагать описанные продукты, услуги и возможности в других странах. Сведения о продуктах и услугах, доступных в настоящее время в вашей стране, можно получить в местном представительстве IBM. Любые ссылки на продукты, программы или услуги IBM не означают явным или неявным образом, что можно использовать только продукты, программы или услуги IBM. Разрешается использовать любые функционально эквивалентные продукты, программы или услуги, если при этом не нарушаются права IBM на интеллектуальную собственность. Однако ответственность за оценку и проверку работы любых продуктов, программ и услуг других фирм лежит на пользователе.

Фирма IBM может располагать патентами или рассматриваемыми заявками на патенты, относящимися к предмету данного документа. Получение этого документа не означает предоставления каких-либо лицензий на эти патенты. Запросы по поводу лицензий следует направлять в письменной форме по адресу:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

По поводу лицензий, связанных с использованием наборов двухбайтных символов (DBCS), обращайтесь в отдел интеллектуальной собственности IBM в вашей стране или направьте запрос в письменной форме по адресу:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**Следующий абзац не применяется в Великобритании или в любой другой стране, где подобные заявления противоречат местным законам:** КОРПОРАЦИЯ INTERNATIONAL BUSINESS MACHINES ПРЕДСТАВЛЯЕТ ДАННУЮ ПУБЛИКАЦИЮ "КАК ЕСТЬ" БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ ПРЕДПОЛАГАЕМЫЕ ГАРАНТИИ СОВМЕСТИМОСТИ, РЫНОЧНОЙ ПРИГОДНОСТИ И СООТВЕТСТВИЯ ОПРЕДЕЛЕННОЙ ЦЕЛИ, НО НЕ ОГРАНИЧИВАЯСЬ ИМИ. В некоторых странах для определенных сделок подобные оговорки не допускаются, таким образом, это утверждение может не относиться к вам.

Данная информация может содержать технические неточности и типографские опечатки. Периодически в информацию вносятся изменения, они будут включены в новые издания этой публикации. Фирма IBM может в любое время без уведомления вносить изменения и усовершенствования в продукты и программы, описанные в этой публикации.

Любые ссылки в данной публикации на Web—сайты, не принадлежащие IBM, приводятся только для удобства и никоим образом не означают поддержки IBM этих Web—сайтов. Материалы этих Web—сайтов не являются частью данного продукта IBM и вы можете использовать их только на собственную ответственность.

Если обладателю лицензии на данную программу понадобятся сведения о возможности: (i) обмена данными между независимо разработанными программами и другими программами (включая данную) и (ii) совместного использования таких данных, он может обратиться по адресу:

IBM Corporation  
J74/G4  
555 Bailey Avenue  
P.O. Box 49023  
San Jose, CA 95161–9023  
U.S.A.

Такая информация может быть предоставлена на определенных условиях (в некоторых случаях к таким условиям может относиться оплата).

Лицензированная программа, описанная в данной публикации, и все лицензированные материалы, доступные с ней, предоставляются IBM на условиях Соглашения IBM с заказчиком, Лицензионного соглашения международной программы IBM или эквивалентного соглашения.

Эта информация может содержать примеры данных и отчетов, иллюстрирующие типичные деловые операции. Чтобы эти примеры были правдоподобны, в них включены имена лиц, названия компаний и товаров. Все эти имена и названия вымышлены и любое их сходство с реальными именами и адресами полностью случайно.

#### ЛИЦЕНЗИЯ НА КОПИРОВАНИЕ:

Эта информация содержит примеры прикладных программ на языках программирования, иллюстрирующих приемы программирования для различных операционных платформ. Разрешается копировать, изменять и распространять эти примеры программ в любой форме без оплаты фирме IBM для целей разработки, использования, сбыта или распространения прикладных программ, соответствующих интерфейсу прикладного программирования операционных платформ, для которых эти примеры программ написаны. Эти примеры не были всесторонне проверены во всех возможных условиях. Поэтому IBM не может гарантировать их надежность, пригодность и функционирование.

---

## Информация об интерфейсе программирования

Эта книга предназначена, чтобы помочь вам в написании программ, содержащих операторы SQL. В ней в основном описаны интерфейс программирования общего назначения и сопутствующая руководящая информация для системы IBM DATABASE 2 Universal Database Server for OS/390 (DB2 for OS/390).

Интерфейсы программирования общего назначения позволяют писать программы, которые обращаются к службам DB2 for OS/390.

Однако в этой книге описываются также продуктозависимый интерфейс программирования и сопутствующая руководящая информация.

Продуктозависимые интерфейсы программирования позволяют выполнять при пользовательской установке такие задачи, как диагностику, модификацию, мониторинг, исправление, оптимизацию или настройку продуктов программного обеспечения IBM. Использование таких интерфейсов приводит к зависимости от детальной разработки и реализации продуктов программного обеспечения IBM. Продуктозависимые интерфейсы программирования должны использоваться только для этих специальных целей. Возможно, что, будучи зависимыми от деталей разработки и реализации, программы, написанные для таких интерфейсов, могут потребовать изменений при переходе к работе с новыми выпусками или версиями продукта, или в результате сервисных операций.

Продуктозависимый интерфейс программирования и сопутствующая руководящая информация обозначаются в соответствующих местах глав или разделов с помощью вводного текста или следующей маркировки:

Продуктозависимый интерфейс программирования

Интерфейс программирования общего назначения и Associated Guidance Information ...

Конец Продуктозависимый интерфейс программирования

---

## Товарные знаки

Следующие термины являются товарными знаками корпорации International Business Machines в Соединенных Штатах и/или других странах:

AD/Cycle	Enterprise System/3090
AIX	Enterprise System/9000
APL2	ESA/390
AS/400	GDDM
BookManager	IBM
CICS	IMS
CICS/ESA	IMS/ESA
CICS/MVS	Language Environment
COBOL/370	MVS/DFP
C/370	MVS/ESA
DATABASE 2	Net.Data
DataHub	OS/2
DataPropagator	OS/390
DataRefresher	OS/400
DB2 Connect	Parallel Sysplex
DB2 Universal Database	QMF
DFSMS	RACF
DFSMSdfp	RMF
DFSMSdss	RS/6000
DFSMSShsm	SQL/DS
DFSMS/MVS	System/370
DFSORT	System/390
DB2	SystemView
Distributed Relational Database Architecture	VSE/ESA
DRDA	VTAM
DXT	3090
eNetwork	

Tivoli™ и NetView® – товарные знаки Tivoli Systems Inc. в Соединенных Штатах и/или других странах.

Java и все торговые марки на основе Java – логотипы, товарные знаки или зарегистрированные товарные знаки Sun Microsystems, Inc. в Соединенных Штатах и/или других странах.

Microsoft™, Windows™, Windows NT™ и логотип Windows – товарные знаки Microsoft Corporation в Соединенных Штатах и в других странах.

UNIX – зарегистрированный товарный знак, ее использование в Соединенных Штатах и в других странах лицензируется исключительно фирмой X/Open Company Limited.

Названия других компаний, продуктов и услуг могут быть товарными знаками или марками сервиса других фирм.

## Глоссарий

Термины и сокращения ниже определены так, как они используются в библиотеке DB2. Если вы не нашли нужного вам термина здесь, посмотрите указатель или книгу *IBM Dictionary of Computing*.

### A

**API.** Application programming interface – интерфейс прикладного программирования.

**AR.** Application requester – реквестер прикладных программ. Смотрите *реквестер*.

**AS.** Application server – сервер прикладных программ. Смотрите *сервер*.

**ASCII.** Кодировка, используемая для символьных строк во многих средах, обычно на PC и рабочих станциях. Сравните с *EBCDIC*.

### B

**BLOB.** Последовательность байтов длиной от 0 байтов до 2 Гбайта – 1. Такая строка не зависит от используемой CCSID. Размер значения двоичного большого объекта может быть любым до 2 Гбайта – 1.

**BMP.** Batch Message Processing – пакетная обработка сообщений (в IMS).

### C

**CAF.** Call attachment facility – утилита подключения по вызову.

**CCSID.** Coded character set identifier – идентификатор набора кодовых символов.

**CDB.** Смотрите *база данных связей*.

**CDRA.** Character data representation architecture – структура представления символьных данных.

**CICS.** В данной публикации относится к CICS/MVS и к CICS/ESA.

**CICS/MVS:** Customer Information Control System/Multiple Virtual Storage.

**CICS/ESA:** Customer Information Control System/Enterprise Systems Architecture.

**CLIST.** Сокращение от "Command list". Язык для выполнения заданий TSO.

**CLOB.** Последовательность байтов, представляющая однобайтные символы или смесь однобайтных и двухбайтных символов, длиной до 2 Гбайта – 1. Хотя размер значений символьного большого объекта может быть любым до 2 Гбайта – 1, обычно они используются, когда строка байтов по длине может превзойти пределы для типа VARCHAR.

**CP.** Смотрите *центральный процессор (CP)*.

### D

**DASD.** Direct access storage device – устройство хранения прямого доступа.

**DATABASE 2 Interactive (DB2I).** Утилита DB2, которая обеспечивает выполнение операторов SQL, команд (операций) DB2, команд программиста и вызовов утилит.

**Data Language/I (DL/I).** Язык манипулирования данными IMS; общий интерфейс высокого уровня между пользовательской программой и IMS.

**DBA.** Database administrator – администратор базы данных.

**DBCLOB.** Последовательность байтов, представляющая двухбайтные символы, длиной до 2 гигабайт. Хотя размер значений двухбайтного двоичного большого объекта может быть любым до 2 Гигабайт, обычно этот тип используется, когда строка двухбайтных символов по длине может превзойти пределы для типа VARGRAPHIC.

**DBCS.** Double-byte character set – набор двухбайтных символов.

**DBD.** Database descriptor – дескриптор базы данных.

**DB2I.** DATABASE 2 Interactive.

**DBMS.** Database management system – система управления базами данных.

**DBRM.** Database request module – модуль требований базы данных.

**DCLGEN.** Declarations generator – генератор объявлений.

**DDF.** Distributed data facility – утилита распределенных данных.

**Distributed Relational Database Architecture (DRDA).** Протокол соединения для обработки распределенных реляционных баз данных, используемый в продуктах фирмы IBM для работы с реляционными базами данных. DRDA включает протоколы связи между прикладными программами и удаленными системами управления реляционными базами данных, а также для связи между системами управления реляционными базами данных.

**DL/I.** Data Language/I. Язык манипулирования данными IMS; общий интерфейс высокого уровня между пользовательской программой и IMS.

**DRDA.** Distributed relational database architecture – архитектура распределенных реляционных баз данных.

**DSN.** (1) Имя подсистемы DB2 по умолчанию.  
(2) Имя командного процессора TSO для DB2.  
(3) Три первых символа имен модулей и макрокоманд DB2.

## E

**EBCDIC.** Extended binary coded decimal interchange code – расширенный двоично–десятичный код обмена. Схема кодирования, применяемая для представления символьных данных в средах MVS, VM, VSE и OS/400®. Сравните с *ASCII*.

**EUR.** Стандарты IBM для Европы.

## I

**ID авторизации (authorization ID).** Стока, которую можно проверить при соединении с DB2 и для которой задается набор привилегий. Она может соответствовать отдельному пользователю, группе или функции, однако DB2 не определяет это соответствие.

**ID авторизации SQL (SQL authorization ID, SQL ID).** ID авторизации, который используется для проверки динамических операторов SQL в некоторых ситуациях.

**IFP.** IMS Fast Path – быстрый путь IMS.

**IMS.** Information Management System – система управления информацией.

**Interactive System Productivity Facility (ISPF).** Лицензированная программа IBM, поддерживающая интерактивные диалоговые службы.

**IRLM.** Internal resource lock manager – менеджер блокировок внутренних ресурсов.

**ISO.** International Standards Organization – Международная организация по стандартизации.

**ISPF.** Interactive System Productivity Facility – интерактивная утилита производительности системы.

**ISPF/PDF.** Interactive System Productivity Facility/Program Development Facility – интерактивная утилита производительности системы/утилита разработки программ.

## J

**JCL.** Job control language – язык управления заданиями.

**JIS.** Japanese Industrial Standard – японский промышленный стандарт.

## L

**L-lock.** Смотрите *логическая блокировка*.

**LOB.** Последовательность байтов, представляющая битовые данные, однобайтные символы, двухбайтные символы или смесь однобайтных и двухбайтных символов. Размер большого объекта может доходить до 2 Гбайт без одного байта. Смотрите также *BLOB*, *CLOB* и *DBCLOB*.

**LUW.** Logical unit of work – логическая единица работы.

## M

**MPP.** Message processing program – программа обработки сообщений (IMS).

**MVS.** Multiple Virtual Storage.

**MVS/ESA.** Multiple Virtual Storage/Enterprise Systems Architecture.

## N

**NUL.** В C – символ–ограничитель конца строки.

**null.** Специальное значение, указывающее на отсутствие информации.

**P**

**PCT.** Program control table – таблица управления программами (CICS).

**P-lock.** Смотрите *физическая блокировка*.

**PPT.** (1) Processing program table – таблица обрабатываемых программ (CICS). (2) Program properties table – таблица свойств программ (MVS).

**Q**

**QMF.** Query Management Facility – утилита управления запросами.

**R**

**RACF.** OS/VS2 MVS Resource Access Control Facility – утилита управления доступом к ресурсам.

**RCT.** Resource control table – таблица управления ресурсами (утилита подключения CICS).

**RDB.** Смотрите *реляционная база данных*.

**RDBMS.** Relational database management system – система управления реляционными базами данных.

**RDBNAM.** Смотрите *имя реляционной базы данных*.

**RLF.** Resource limit facility – утилита ограничения ресурсов.

**ROWID.** Смотрите *идентификатор строки*.

**RRSAF.** Recoverable Resource Manager Services attachment facility – утилита подключения служб менеджера восстановимых ресурсов. Подкомпонент DB2, который использует службы управления транзакциями OS/390 и менеджера восстановимых ресурсов для координации принятия ресурсов между DB2 и остальными менеджерами ресурсов, которые также используют OS/390 RRS в системе OS/390.

**S**

**SPUI.** SQL Processor Using File Input – процессор SQL, использующий файловый ввод. Утилита подкомпонента подключения TSO, которая позволяет пользователю DB2I выполнять операторы SQL, не встраивая их в прикладную программу.

**SQL.** Structured Query Language – язык структурированных запросов.

**SQLCA.** область связи SQL.

**SQLDA.** область дескриптора SQL.

**SQL/DS.** SQL/Data System. Другое название – *DB2 for VSE & VM*.

**SQL ID.** ID авторизации SQL.

**Structured Query Language (SQL).**

Стандартизованный язык определения данных в реляционных базах данных и работы с ними.

**T**

**TCB.** Task control block – блок управления заданием MVS.

**TMP.** Terminal Monitor Program – программа монитора терминала.

**TSO.** Time-sharing option – опция разделения времени.

**V**

**Virtual Telecommunications Access Method (VTAM).**

Лицензированная программа IBM, которая управляет связью и потоком данных в сети SNA.

**VSAM.** Virtual storage access method – виртуальный метод доступа.

**VTAM.** Virtual telecommunication access method – виртуальный телекоммуникационный метод доступа MVS.

**A**

**аварийная остановка.** (abend) Аварийное завершение задания.

**аварийное завершение задания (аварийная остановка).** (abnormal end of task, abend)

Прекращение выполнения задания или работы подсистемы из-за возникшей ситуации ошибки, которую невозможно устранить во время выполнения средствами восстановления.

**агент.** (agent) В DB2 – структура, связывающая все процессы в единице работы DB2. Термин *внешний агент* обычно используется как синоним *внешнего потока*. *Системные агенты* – единицы работы, выполняемые независимо от внешнего агента, такие как выполнение предварительных выборок, отложенной записи и служебных задач.

**администратор базы данных.** (database administrator, DBA) Человек, отвечающий за

проектирование, разработку, создание, сохранность, обслуживание и использование базы данных.

**адресное пространство.** (address space) Диапазон страниц виртуальной памяти, определяемый номером (ASID), собранием сегментов и таблицами страниц, которые отображают виртуальные страницы на реальные страницы в памяти компьютера.

**активация триггера.** (trigger activation) Процесс, происходящий, когда случается событие триггера, указанное в его определении. Активация триггера состоит из оценки условия действия триггера и условного выполнения операторов SQL триггера.

**алиас.** (alias) Альтернативное имя, которое можно использовать в операторах SQL для ссылок на таблицы или производные таблицы в той же подсистеме DB2 или в удаленной подсистеме.

**атрибут.** (attribute) Характеристика элемента. Например, при проектировании баз данных номер телефона сотрудника – один из его атрибутов.

## Б

**база данных.** (database) Собрание таблиц, или собрание табличных пространств и индексных пространств.

**база данных связей.** (communications database, CDB) Набор таблиц в каталоге DB2, используемых для установки диалогов с удаленными СУБД.

**базовая таблица.** (base table) (1) Таблица, созданная оператором SQL CREATE TABLE и используемая для хранения постоянных данных. Сравните с *таблицей результатов и временной таблицей*.

(2) Таблица, содержащая определение столбца большого объекта. Сами данные столбца большого объекта не хранятся в базовой таблице. Базовая таблица содержит идентификатор строки для каждой строки и столбец индикатора для каждого из столбцов больших объектов. Сравните с *дополнительной таблицей*.

**базовое табличное пространство.** (base table space) Табличное пространство, содержащее базовые таблицы.

**битовые данные.** (bit data) Данные символьных типов CHAR или VARCHAR, не зависящие от используемой кодировки.

**блок запроса.** (query block) Часть запроса, состоящая из одного из условий FROM. В условии FROM может быть несколько блоков запроса в

зависимости от внутренней обработки запроса в DB2.

**блокировка.** (lock) Средство управления одновременными событиями или доступом к данным. В DB2 блокировки выполняются IRLM.

**блокировка строки.** (row lock) Блокировка одной строки данных.

**блокировка транзакции.** (transaction lock) Блокировка, используемая для управления одновременным выполнением операторов SQL.

**блок управления заданием.** (task control block, TCB) Блок управления используется для информации о связи в адресном пространстве, соединенном с DB2. Адресное пространство может поддерживать несколько соединений заданий (по одному на задание), но только одно соединение адресного пространства. Смотрите *соединение адресного пространства*.

**большая блокировка.** (gross lock) Блокировка таблицы, раздела или табличного пространства в совместном, модификационном, или монопольном режиме.

**большой объект.** (large object, LOB) Смотрите *LOB*.

**бронирующая блокировка.** (drain lock) Блокировка заявочного класса, запрещающая выполнение заявок.

## В

**вариативная функция.** (variant function) Смотрите *недетерминированная функция*.

**версия.** (version) Член комплекта подобных программ, DBRM, пакетов или больших объектов.

**Версия программы** – это исходный текст, полученный при препроцессинге. Версия программы идентифицируется именем программы и отметкой времени (маркером соответствия).

**Версия DBRM** – это DBRM, полученный при препроцессинге программы. Версия DBRM идентифицируется тем же именем программы и отметкой времени, что и соответствующая версия программы.

**Версия пакета** – это результат связывания DBRM с определенной системой баз данных. Версия пакета идентифицируется тем же именем программы и маркером соответствия, что и DRBM.

**Версия большого объекта** – это копия значения большого объекта на некоторый

момент времени. Номер версии большого объекта хранится в записи дополнительного индекса для большого объекта.

**вложенное табличное выражение.** (nested table expression) Подвыбор в условии FROM в скобках.

**внешнее адресное пространство.** (allied address space) Область памяти, внешняя по отношению к DB2, соединенная с DB2 и тем самым возможная для использования службами DB2.

**внешнее объединение.** (outer join) Результат операции объединения, в который включаются все соответствующие строки обеих таблиц и некоторые или все их не имеющие соответствия строки. Смотрите также *объединение*.

**внешний ключ.** (foreign key) Ключ, указанный в определении реляционной связи. Таблица, для которой задан внешний ключ, называется зависимой. Этот ключ должен включать в себя столько же столбцов с теми же описаниями, что и первичный ключ родительской таблицы.

**внешний поток.** (allied thread) Поток, запущенный на локальной подсистеме DB2, который может обращаться к данным на удаленной подсистеме DB2.

**внешняя функция.** (external function) Функция, исходный текст которой написан на языке программирования, принимающая скалярные значения аргументов и выдающая при каждом вызове скалярный результат. Сравните с *функцией с источником и встроенной функцией*.

**внутреннее объединение.** (inner join) Результат операции объединения, в который включаются только соответствующие строки объединяемых таблиц. Смотрите также *объединение*.

**внутриоператорное имя.** (correlation name) Идентификатор, обозначающий таблицу, производную таблицу или отдельные строки таблицы или производной таблицы внутри одного оператора SQL. Может быть определено в любом условии FROM или в первом условии оператора UPDATE или DELETE.

**восстановление.** (recovery) Процесс воссоздания баз данных после системной ошибки.

**временная таблица.** (temporary table) Таблица, созданная оператором SQL CREATE GLOBAL TEMPORARY TABLE и используемая для хранения

временных данных. Сравните с *таблицей результатов*.

**время.** (time) Значение из трех частей, содержащее время дня в часах, минутах и секундах.

**время активации триггера.** (trigger activation time) В определении триггера – указание, должен триггер активироваться перед указанным событием или после него.

**встроенная функция.** (built-in function) Функция, входящая в состав DB2. Сравните с *пользовательской функцией*.

**встроенный SQL.** (embedded SQL) Операторы SQL, входящие в код прикладной программы. Смотрите *статический SQL*.

**выбор функции.** (function selection) Смотрите *разрешение функции (function resolution)*.

**выражение.** (expression) Операнд или операнды, соединенные знаками операций, при вычислении которых получается одно значение.

## Г

**генератор объявлений.** (declarations generator, DCLGEN) Подкомпонент DB2, который генерирует объявления таблиц SQL и объявления структур данных COBOL, С или PL/I, соответствуют этим таблицам. Объявления генерируются по информации системного каталога DB2. DCLGEN вызывается также подкомандой DSN.

**глобальная блокировка.** (global lock) Блокировка, которая обеспечивает и внутрисистемное, и межсистемное управление одновременной работой DB2, то есть область блокировки охватывает все подсистемы DB2 группы совместного использования данных.

**графическая строка.** (graphic string) Последовательность символов DBCS.

**группа совместного использования данных.** (data sharing group) Собрание из одной или нескольких подсистем DB2, которые прямо обращаются к одним и тем же данным и модифицируют их, не нарушая их целостности.

**группа хранения.** (storage group) Именованный набор томов DASD, где могут храниться данные DB2.

## Д

**дата.** (date) Значение из трех составных частей — числа, месяца и года.

**двоичная строка.** (binary string)

Последовательность битов, не зависящая от CCSID. Данные типа BLOB представляют собой двоичные строки.

**двоичное целое.** (binary integer) Основной тип данных, далее подразделяемый на короткое целое и длинное целое.

**двоичный большой объект.** (binary large object, BLOB) Смотрите *BLOB*.

**двухбайтный символьный большой объект  
(double-byte character large object).** (DBCLOB)  
Смотрите *DBCLOB*.

**действие триггера.** (triggered action) Программа на языке SQL, выполняемая при активации триггера. Действие триггера состоит из необязательной оценки условия триггера и набора операторов SQL триггера, которые выполняются, если условие триггера справедливо.

**дескриптор базы данных.** (database descriptor, DBD) Внутреннее представление определения базы данных DB2, которое отражает определение данных из каталога DB2. В дескрипторе определяются следующие объекты: табличные пространства, таблицы, индексы, индексные пространства и отношения.

**детерминированная функция.** (deterministic function) Пользовательская функция, результат которой зависит только от входных аргументов. Это значит, что при повторном вызове функции с теми же значениями аргументов будет получен тот же результат. Иногда называется *невариативной функцией*, которая не обязательно дает один и тот же результат при вызове с теми же входными значениями.

**дефектный пакет.** (invalid package) Пакет считается дефектным, если он зависит от отброшенного объекта, который не является пользовательской функцией. Такие пакеты неявно повторно связываются при вызове. Сравните с *неработоспособным пакетом*.

**динамический SQL.** (dynamic SQL) Операторы SQL, подготавливаемые и выполняемые в прикладной программе во время ее выполнения. В динамическом SQL исходный текст SQL

записывается в переменные хоста, а не кодируется в тексте программы. Оператор SQL во время выполнения прикладной программы может изменяться.

**динамическое определение ресурсов.** (resource definition online) Возможность CICS, позволяющая оперативно определять ресурсы CICS без составления таблиц.

**длительность.** (duration) Число, представляющее интервал времени. Смотрите *длительность для дат, длительность для временных отметок и длительность для времени*.

**длительность блокировки.** (lock duration) Промежуток времени, в течение которого сохраняется блокировка DB2.

**длительность для времени.** (time duration) Десятичное целое, представляющее число часов, минут и секунд.

**длительность для временных отметок.** (labeled duration) Число, обозначающее длительность в годах, месяцах, днях, часах, минутах, секундах и микросекундах.

**длительность для дат.** (date duration) Десятичное целое, представляющее число лет, месяцев и дней.

**доопределенная функция.** (overloaded function) Общее для нескольких экземпляров имя функции.

**дополнительная таблица.** (auxiliary table) Таблица, где хранятся столбцы, определенные в другой таблице.

**дополнительный индекс.** (auxiliary index) Индекс дополнительной таблицы, в котором каждая запись индекса ссылается на большой объект.

**доступ DRDA.** (DRDA access) Метод доступа к распределенным данным, при котором вы можете соединяться с удаленным положением при помощи операторов SQL, чтобы выполнять пакеты, ранее связанные с этим положением. Для идентификации серверов используется оператор SQL CONNECT или трехчастное имя; операторы SQL выполняются с использованием пакетов, ранее связанных с этими серверами. Сравните с *доступом по собственному протоколу DB2*.

**доступ по собственному протоколу DB2.** (собственный протокол DB2 access) Метод доступа к распределенным данным, при помощи которого можно направить запрос другой системе DB2. Сравните с *доступом DRDA*.

## 3

**забронировать.** (drain) Оставить за собой блокированный ресурс, запретив дальнейшие обращения к этому объекту.

**зависимый.** (dependent) Объект (строка, таблица или табличное пространство) считается зависимым, если у него есть хотя бы один родитель. Этот объект называют также зависимым от своего родителя. Смотрите *родительская строка*, *родительская таблица*, *родительское табличное пространство*.

**запись.** (record) Представление строки или иных данных в памяти.

**заявить.** (claim) Зарегистрировать на DB2, что к объекту производится обращение. Такая регистрация называется заявкой. Заявка производится, чтобы предотвратить бронирование объекта до момента принятия. Сравните с *забронировать*.

**заявочный класс.** (claim class) Класс для следующих типов доступа к объекту:

стабильность на уровне указателя (CS)  
многократное чтение (RR)  
запись

**значение.** (value) Минимальная единица данных, с которой работает SQL.

**значение даты–времени.** (datetime value) Значения типов даты DATE, TIME или TIMESTAMP.

**значение по умолчанию.** (default value) Предопределенное значение, атрибут или опция, которые предполагаются, если явно не указано других.

## И

**идентификатор набора кодовых символов.** (coded character set identifier, CCSID) 16-битное число, однозначно идентифицирующее кодовое представление графических символов. Обозначает идентификатор схемы кодирования и одну или несколько пар из идентификатора набора символов и соответствующего идентификатора кодовой страницы.

**идентификатор с ограничителями.** (delimited identifier) Последовательность символов, заключенная в кавычки (""). Эта последовательность должна начинаться с латинской буквы, за которой могут идти латинские буквы, цифры или символ подчеркивания (\_).

**идентификатор строки.** (row identifier, ROWID) Значение, однозначно определяющее строку. Это значение хранится вместе со строкой и никогда не меняется.

**идентификатор хоста.** (host identifier) Имя, объявленной в программе хоста.

**имя DD.** (DD name) Имя определения данных.

**имя LU.** (LU name) Имя логического устройства, то есть имя, по которому VTAM обращается к узлу в сети. Сравните с *именем положения*.

**имя группы.** (group name) Идентификатор XCF MVS для группы совместного использования данных.

**имя определения данных.** (data definition name, DD name) Имя оператора определения данных (DD), соответствующее блоку управления данными с тем же именем.

**имя плана.** (plan name) Имя плана прикладной программы.

**имя положения.** (location name) Имя, при помощи которого DB2 ссылается на конкретную подсистему DB2 в сети, где имеется много подсистем. Сравните с *именем LU*.

**имя реляционной базы данных.** (relational database name, RDBNAM) Уникальный идентификатор RDBMS в сети. В DB2 это значение столбца LOCATION таблицы SYSIBM.LOCATIONS в CDB. В публикациях по DB2 имя RDBMS называют значением LOCATION или значением положения.

**индекс.** (index) Набор указателей, логически упорядоченных по значениям ключа. Индексы обеспечивают быстрый доступ к данным и могут обеспечивать уникальность для строк в таблице.

**индекс кластеризации.** (clustering index) Индекс, который определяет, как строки физически упорядочены в табличном пространстве.

**индексное пространство.** (index space) Набор страниц, используемый для хранения записей одного индекса.

**индексный раздел.** (index partition) Набор данных VSAM, который содержится в многораздельном индексном пространстве.

**индексы типа 2.** (type 2 indexes) Индексы, созданные DB2 после Версии 5, или определенные как индексы типа 2 в DB2 Версия 4 или Версия 5.

**индексы типа 1.** (type 1 indexes) Индексы, созданные в выпуске DB2 до DB2 Версия 4 или определенные как индексы типа 1 в Версии 4.

Сравните с **индексами типа 2**. Версия 6 более не поддерживает индексы типа 1.

**интерфейс прикладного программирования**. (application program interface, API) Функциональный интерфейс, предоставляемый операционной системой или отдельно приобретаемой лицензированной программой, который позволяет прикладной программе, написанной на языке высокого уровня, использовать особые данные или функции операционной системы или лицензированной программы.

**исходная программа**. (source program) Набор операторов языка хоста и операторов SQL, которые обрабатываются прекомпилятором SQL.

**исходный тип**. (source type) Существующий тип, используемый для внутреннего представления пользовательского типа.

## К

**Кандзи DB2I**. (DB2I Kanji Feature) Лента с панелями и заданиями, которая позволяет выводить панели DB2I японскими иероглифами (канджи).

**каскадное срабатывание триггеров**. (trigger cascading) Процесс, происходящий, когда действие триггера вызывает активацию другого триггера.

**каталог DB2**. (catalog) Собрание таблиц DB2, содержащих описания таких объектов, как таблицы, производные таблицы и индексы.

**каталог DB2**. (DB2 catalog) Таблицы, поддерживаемые DB2 и содержащие описания объектов DB2 – таблиц, производных таблиц и индексов.

**Кбайт**. (KB) Килобайт (1024 байт).

**клиент**. (client) Смотрите *реквестер*.

**ключ**. (key) Столбец или упорядоченный набор столбцов, указанный в описании таблицы, индекса или реляционной связи.

**ключ индекса**. (index key) Набор столбцов в таблице, используемый для определения порядка записей индекса.

**код**. (code point) В CDRA – уникальный битовый образ символа из кодовой страницы.

**код возврата SQL**. (SQL return code) SQLCODE или SQLSTATE.

**кодовая страница**. (code page) Набор соответствий символов и их кодов.

**код причины аварийной остановки**. (abend reason code) 4–байтный шестнадцатеричный код, который однозначно определяет ошибку DB2. Полный список кодов причин аварийных остановок DB2 с объяснениями приводится в книге *DB2 Сообщения и коды*.

**команда**. (command) Команда операции DB2 или подкоманда DSN. Не следует путать с оператором SQL.

**команда DB2**. (DB2 command) Указание подсистеме DB2, позволяющее пользователю запускать или останавливать DB2, выводить информацию о текущих пользователях, запускать или останавливать базы данных, выводить информацию об их состоянии и т.п.

**компоновка**. (link-edit) Создание загружаемой компьютерной программы при помощи компоновщика.

**константа**. (constant) Элемент языка, представляющий постоянное значение. Константы делятся на строковые и числовые. Сравните с *переменной*.

**конфликт глобальной блокировки**. (global lock contention) Конфликт требований блокировки между различными членами группы совместного использования данных DB2 при попытке установить порядок совместного использования ресурсов.

**конфликт физических блокировок**. (physical lock contention) Конфликтное состояние претендентов на физическую блокировку. Смотрите *согласуемая блокировка*.

**кратность триггера**. (trigger granularity) Указание в определении триггера, сколько раз он должен выполняться: один раз при событии триггера или же по одному разу для каждой строки, затронутой этим событием.

**критерий поиска**. (search condition) Критерий выбора строк из таблицы. Критерий поиска состоит из одного или нескольких предикатов.

## Л

**левое внешнее объединение**. (left outer join) Результат операции объединения, в который включаются все соответствующие строки обеих таблиц и строки из первой таблицы, не имеющие соответствия. Смотрите также *объединение*.

**логическая блокировка**. (logical lock) Тип блокировки, используемый транзакциями для

управление внутри- и межсистемной одновременной работой транзакций DB2 с данными.

**логическая единица работы.** (logical unit of work, LUW) В IMS – работа, выполняемая программой от одной точки синхронизации до другой.

**логический раздел индекса.** (logical index partition) Набор всех ключей, ссылающихся на один и тот же раздел данных.

**логическое устройство.** (logical unit) Точка доступа прикладной программы к сети SNA для связи с другой прикладной программой.

**логическое устройство партнера.** (partner logical unit) Точка доступа к сети SNA, которая соединена с локальной DB2 диалогом VTAM.

**ложный конфликт глобальных блокировок.** (false global lock contention) Сигнал о конфликте от coupling facility, когда хеш-функция дает одно и то же значение для нескольких имен блокировок и реального конфликта нет.

**локальная блокировка.** (local lock) Блокировка, обеспечивающая управление одновременностью внутри системы DB2, но не между различными системами DB2, то есть область этой блокировки – одна система DB2.

**локальная подсистема.** (local subsystem) Отдельная RDBMS, к которой пользователь или прикладная программа подключены непосредственно (в случае DB2 – при помощи возможностей подключения DB2).

**локальный.** (local) Относится к любому объекту из локальной подсистемы DB2. Например, **локальная таблица** – эта таблица в локальной подсистеме DB2. Сравните с **удаленным**.

**локатор большого объекта.** (LOB locator) Механизм, который позволяет прикладной программе работать со значением большого объекта в системе баз данных. Локатор большого объекта – целое значение, занимающее полное слово и представляющее значение одного большого объекта. Прикладная программа записывает локатор большого объекта в переменную хоста; затем она может применять операции SQL к значению большого объекта при помощи его локатора.

**локатор набора результатов.** (result set locator) 4-байтное значение, используемое DB2 для однозначной идентификации набора результатов запроса, возвращенного хранимой процедурой.

**локатор таблицы.** (table locator) Механизм, который позволяет обращаться к таблицам

переходов триггеров в условии FROM из операторов SELECT, подвыборов операторов INSERT или из пользовательских функций. Локатор таблицы – целое значение, занимающее полное слово и представляющее таблицу переходов.

## M

**маркер параметра.** (parameter marker)

Вопросительный знак (?) в строке динамического оператора SQL. Вопросительный знак может стоять там, где может стоять переменная хоста в строке соответствующего статического оператора SQL.

**маркер согласованности.** (consistency token)

Временная отметка, используемая для генерации идентификатора версии прикладной программы. Смотрите также **версия**.

**маркер типизированного параметра.** (typed parameter marker) Маркер параметра, который задается вместе с типом данных назначения. Он имеет следующий общий вид:

CAST(?) AS тип-данных)

**масштаб.** (scale) В SQL – число цифр справа от десятичной точки (то, что в языке C называется точностью). В библиотеках DB2 используется определение SQL.

**материализация.** (materialize) (1) Процесс переноса строк из производной таблицы или вложенного табличного выражения в рабочий файл для дальнейшей обработки в запросе.

(2) Перемещение значения большого объекта в непрерывную память. Поскольку значение большого объекта может быть очень большим, DB2 не материализует данные большого объекта, пока это не становится абсолютно необходимым.

**менеджер блокировок внутренних ресурсов.**

(internal resource lock manager, IRLM) Подсистема MVS, используемая DB2 для управления блокировками связи и баз данных.

**меню.** (menu) Список доступных функций, выводимый на экран для выбора оператором. Может называться также **панелью меню**.

**многократное чтение.** (repeatable read, RR)

Уровень изоляции, обеспечивающий максимальный уровень защиты от других выполняемых программ. Когда программа выполняется с защитой такого типа, строки, к которым она обращается, не могут быть изменены другими программами, пока программа не достигнет точки принятия.

**многораздельное табличное пространство.**

(partitioned table space) Табличное пространство,

разделенное на части (по диапазонам индексного ключа), каждый из которых может обрабатываться утилитами независимо.

**многосистемная доступность ч/з DB2.** (inter-DB2 R/W interest) Свойство данных в табличном пространстве, индексе или разделе, которые открыты несколькими членами группы совместного использования данных, причем хотя бы одним из них – для записи.

**многоузловое изменение.** (multi-site update) Распределенная обработка реляционных баз данных, при которой в одной единице работы производится изменение данных в нескольких подсистемах.

**модифицируемые блокировки.** (modify locks) Физические или логические блокировки, для которых задан атрибут MODIFY. Список активных блокировок такого рода постоянно поддерживается в структуре блокировок раздела обеспечения взаимодействия. Если затребованная DB2 завершает работу из-за ошибки, модифицируемые блокировки DB2 преобразуются в *поддерживаемые блокировки*.

**модуль загрузки.** (load module) Блок программы, который можно загружать в основную память для выполнения. Получается на выходе компоновщика.

**модуль требований базы данных.** (database request module, DBRM) Член набора данных, созданный прекомпилятором DB2, который содержит информацию об операторах SQL. Модули DBRM используются в процессе связывания.

## H

**набор двухбайтных символов.** (double-byte character set, DBCS) Набор символов, используемых в национальных языках типа японского или китайского, где число символов больше, чем можно представить одним байтом. При этом каждый символ представляется двумя байтами, что требует особого оборудования для вывода и печати.

**набор кодовых символов.** (coded character set) Набор однозначных правил, задающий набор символов и взаимно-однозначное соответствие между символами набора и их кодами.

**набор результатов.** (result set) Набор строк, возвращаемый клиентской программе хранимой процедурой.

**набор символов.** (character set) Определенный набор символов.

**набор страниц.** (page set) Способ обращения к табличному или индексному пространству. Каждый набор страниц формируется из собрания наборов данных VSAM.

**наследование.** (inheritance) Способность одного объекта передавать свои атрибуты другому объекту.

**начальное задание.** (originating task) В параллельной группе – первичный агент, который получает данные от других единиц выполнения (*параллельных заданий*), которые обрабатывают части запроса параллельно.

**невариативная функция.** (not-variant function) Смотрите *детерминированная функция*.

**недетерминированная функция.** (not-deterministic function) Пользовательская функция, результат которой зависит не только от входных аргументов. Это значит, что при повторном вызове функции с теми же значениями аргументов может быть получен другой результат. Иногда называется *вариативной функцией*. Сравните с *детерминированной функцией*, которая всегда дает один и тот же результат при вызове с теми же входными значениями.

**неоднозначная.** (indoubt) Состояние единицы восстановления. Если ошибка DB2 происходит после завершения фазы 1 обработки принятия, но до начала фазы 2, только координатор принятия знает, должна эта единица восстановления быть принята или же откачена. Если при аварийном перезапуске DB2 не имеет необходимой для принятия информации, данная единица восстановления считается *неоднозначной*, пока DB2 не получит нужную информацию от координатора.

**неоднозначный указатель.** (ambiguous cursor) Указатель базы данных, при определении которого не задано условие FOR FETCH ONLY или FOR UPDATE OF, не определенный на таблице результатов только для чтения, не являющийся назначением условия WHERE CURRENT операторов SQL UPDATE или DELETE, и находящийся в плане или пакете, содержащем операторы SQL PREPARE или EXECUTE IMMEDIATE.

**неработоспособный пакет.** (inoperative package) Пакет, который нельзя использовать, поскольку одна или несколько пользовательских функций, от которых пакет зависит, были отброшены. Такой пакет должен быть явным образом связан повторно. Сравните с *дефектным пакетом*.

**нетипизированный маркер параметра.** (untyped parameter marker) Маркер параметра, который задается без типа данных назначения. Задается в виде вопросительного знака.

**O**

**область дескриптора SQL.** (SQL Descriptor Area, SQLDA) Структура, которая описывает входные и выходные переменные или столбцы таблицы результатов.

**область связи SQL.** (SQL Communication Area, SQLCA) Структура, используемая для передаче прикладной программе информации о выполнении ее операторов SQL.

**объединение.** (join) Операция получения данных из нескольких таблиц на основе совпадения значений в столбцах. Смотрите также *полное внешнее объединение, внутреннее объединение, левое внешнее объединение, внешнее объединение, правое внешнее объединение и объединение с уравнением*.

**объединение с уравнением.** (equi-join) Операция объединения, условие которого имеет вид *выражение = выражение*.

**объект блокировки.** (lock object) Ресурс, доступ к которому регулируется блокировкой DB2.

**ограничение.** (constraint) Правило, которое накладывает определенные условия на значения в таблице при вставке, изменении и удалении. Смотрите *реляционные ограничения, ограничения уникальности и проверочные ограничения таблиц*.

**одновременность.** (concurrency) Совместное использование ресурсов несколькими прикладными процессами одновременно.

**однораздельный индекс.** (nonpartitioned index) Индекс, не являющийся многораздельным индексом многораздельного табличного пространства.

**операторы SQL триггера.** (triggered SQL statements) Набор операторов SQL, которые выполняются при активации триггера, если условие действия триггера справедливо. Называются также *телом триггера*.

**операция SQL триггера.** (triggering SQL operation) Операция SQL, которая при применении к таблице триггера вызывает активацию этого триггера.

**операция сравнения.** (comparison operator) Знак (например,  $=$ ,  $>$ ,  $<$ ), задающий отношение между двумя значениями.

**определитель функции.** (function definer) Идентификатор авторизации владельца схемы функции, указанный в операторе CREATE FUNCTION.

**опция разделения времени.** (time-sharing option, TSO) Опция, обеспечивающая разделение времени для удаленных терминалов.

**особое имя функции.** (specific function name) Определенная пользовательская функция может быть известна менеджеру базы данных под своим особым именем. У нескольких пользовательских функций имени могут совпадать. Когда для базы данных определяется пользовательская функция, каждой функции назначается особое имя, уникальное в ее схеме. Это имя может быть задано пользователем или же назначено по умолчанию.

**откат.** (rollback) Процесс восстановления данных, измененных операторами SQL, к состоянию последней точки принятия. Все блокировки после этого снимаются. Противоположное понятие – *принятие*.

**отложенная проверка.** (check pending) Состояние табличного пространства или раздела, которое не дает использовать его некоторым утилитам и операторам SQL, поскольку в нем могут быть строки, нарушающие реляционные или проверочные ограничения.

**отметка времени.** (timestamp) Значение из семи частей, состоящее из даты и времени и выражаемое в годах, месяцах, днях, часах, минутах, секундах и микросекундах.

**П**

**пакет.** (package) Называется также *прикладным пакетом*. Объект, содержащий набор операторов SQL, связанных статически и доступных для обработки.

**пакет триггера.** (trigger package) Пакет, создаваемый при выполнении оператора CREATE TRIGGER. Этот пакет выполняется при активации триггера.

**пакет функции.** (function package) Пакет, получаемый в результате связывания DBRM для программы функции.

**панель.** (panel) Предопределенный вид экрана, где указаны положения и характеристики полей на поверхности дисплея (например, *панель меню*).

**панель справки.** (help panel) Экран информации, содержащий указания для помощи пользователю за терминалом.

**параллелизм ввода–вывода запроса.** (query I/O parallelism) Параллельный доступ к данным, который достигается переключением между несколькими требованиями ввода–вывода в одном запросе.

**параллелизм запросов СР.** (query CP parallelism) Параллельная обработка одного запроса, достигаемая при использовании нескольких заданий. Смотрите также *параллелизм запросов Sysplex*.

**параллелизм запросов Sysplex.** (Sysplex query parallelism) Параллельная обработка одного запроса, достигаемая при использовании нескольких заданий на нескольких подсистемах DB2. Смотрите также *параллелизм запросов СР*.

**параллельное задание.** (parallel task) Выполняемая единица, динамически создаваемая для параллельной обработки запроса. Реализуется при помощи блока служебного требования MVS.

**первичный индекс.** (primary index) Индекс, который обеспечивает уникальность значений первичного ключа.

**первичный ключ.** (primary key) Уникальный непустой ключ, составляющий часть определения таблицы. Таблицу нельзя определить как родительскую, если у нее нет уникального или первичного ключа.

**перезапуск группы.** (group restart) Перезапуск по крайней мере одного из членов группы совместного использования данных после потери области блокировок или совместной области связей.

**переменная.** (variable) Элемент данных, значение которого может быть изменено. Пример переменной – элементарные данные в языке COBOL. Противоположное понятие *константа*.

**переменная–индикатор.** (indicator variable) Переменная, используемая для представления пустого (null) значения в прикладной программе. Если значение в выбранном столбце – null, в переменную–индикатор заносится отрицательное значение.

**переменная перехода.** (transition variable) Переменная, которая содержит значение столбца затронутой строки таблицы триггера в его состоянии перед или после события этого триггера. Операторы SQL в определении триггера смогут обращаться к набору старых или новых значений.

**переменная хоста.** (host variable) В прикладной программе – переменная, на которую ссылаются встроенные операторы SQL.

**переменная хоста с символом–ограничителем.** (NUL-terminated host variable) Переменная хоста переменной длины, конец данных в значении которой обозначается символом–ограничителем.

**плавающее число с одинарной точностью.** (single-precision floating point number) 64-битное

приближенное представление действительного числа.

**план.** (plan) Смотрите *план прикладной программы*.

**план программы.** (application plan) Управляющая структура, формируемая в процессе связывания и используемая DB2 для обработки операторов SQL, встреченных во время выполнения оператора.

**повторное связывание.** (rebind) Создание нового плана программы для прикладной программы, которая ранее уже связывалась. Если вы, например, добавили индекс к таблице, созданной вашей программой, необходимо повторно связать эту программу, чтобы можно было воспользоваться преимуществами, которые может дать этот индекс.

**повышение уровня блокировки.** (lock promotion) Процесс изменения размера или режима блокировки DB2 на более высокий уровень.

**подвыбор.** (subselect) Форма запроса, не включающая условия ORDER BY, условия UPDATE или оператора UNION.

**подготовленный оператор SQL.** (prepared SQL statement) Именованный объект, представляющий собой исполняемую форму оператора SQL, которая обработана оператором PREPARE.

**подзапрос.** (subquery) Оператор SELECT в условии WHERE или HAVING другого оператора SQL; вложенный оператор SQL.

**подсистема.** (subsystem) Отдельный экземпляр RDBMS.

**показатель фильтра.** (filter factor) Число от нуля до единицы, оценка доли строк, для которой предикат истинен.

**полное внешнее объединение.** (full outer join) Результат операции объединения, в который включаются все соответствующие строки обеих таблиц и несоответствующие строки из каждой из этих таблиц. Смотрите также *объединение*.

**пользовательская функция.** (user-defined function, UDF) Функция, определенная в DB2 при помощи оператора CREATE FUNCTION; она может использоваться далее в операторах SQL. Пользовательская функция может быть либо *внешней функцией*, либо *функцией с источником*. Сравните со *встроенной функцией*.

**пользовательский тип.** Пользовательский тип данных, внутреннее представление которого совпадает с внутренним представлением существующего типа (исходного для него), однако

рассматриваемый как особый и несовместимый по семантическим соображениям.

**последовательный набор данных.** (sequential data set) Набор данных (не системы DB2), записи которого организованы на основе их последовательных физических позиций, как например, на магнитной ленте. Некоторые утилиты баз данных DB2 работают с последовательными наборами данных.

**поток.** (thread) Структура DB2, которая описывает соединение программы, прослеживает ход его выполнения, обрабатывает функции доступа к ресурсам и ограничивает доступ к ресурсам и службам DB2. Большинство функций DB2 выполняются в структуре потока. Смотрите также [внешний поток](#) и [поток доступа к базе данных](#).

**поток доступа к базе данных.** (database access thread) Поток доступа к данным локальной подсистемы для удаленной подсистемы.

**правое внешнее объединение.** (right outer join) Результат операции объединения, в который включаются все соответствующие строки обеих таблиц и строки из второй таблицы, не имеющие соответствия. Смотрите также [объединение](#).

**предикат.** (predicate) Элемент критерия поиска, который представляет собой операцию сравнения выражений.

**прекомпиляция.** (precompilation) Обработка прикладных программ с операторами SQL, которая проводится перед компиляцией. Операторы SQL заменяются на операторы, которые распознает компилятор языка хоста. Вывод этапа прекомпиляции включает исходный текст, который передается компилятору, и модуль требований баз данных (DBRM), который используется в процессе связывания.

**прикладная программа.** (application) Программа или система программ, выполняющая задание для пользователя, например, программа расчета заработной платы.

**принятие.** (commit) Операция, которая завершает единицу работы, снимая блокировки; внесенные в базы данных единицей работы изменения становятся доступными другим процессам.

**пробелы.** (space) Последовательность из одного или нескольких символов пробела.

**проверочная целостность.** Условие, означающее, что каждая строка таблицы удовлетворяет проверочным ограничениям, определенным для этой таблицы. Поддержание проверочной целостности

требует проверки соблюдения проверочных ограничений для операция, которые добавляют или изменяют данные.

**проверочное ограничение.** Смотрите [проверочное ограничение таблицы](#)

**проверочное ограничение таблицы.** (table check constraint) Определяемое пользователем ограничение, которое задает, какие значения могут содержать столбцы базовой таблицы.

**программа хоста.** (host program) Прикладная программа на языке высокого уровня, которая содержит встроенные операторы SQL.

**производная таблица.** (view) Способ представления данных одной или нескольких таблиц. Производная таблица может включать все или некоторые столбцы из таблиц, на которых она определена.

**простой идентификатор.** (ordinary identifier) Латинская буква в верхнем регистре, за которой могут идти латинские буквы в верхнем регистре, цифры или символы подчеркивания (\_). Обычный идентификатор не должен совпадать с зарезервированным словом.

**простой элемент.** (ordinary token) Числовая константа, простой идентификатор, идентификатор хоста или ключевое слово.

**пространство данных.** (data space) Непрерывная область адресов виртуальной памяти размером до 2 Гбайт, которой программа может манипулировать непосредственно. В отличие от адресного пространства пространство данных может содержать только данные, оно не содержит общие области, системную информацию или программы.

**процесс блокировки.** (locking) Процесс, посредством которого обеспечивается целостность данных. Блокировка не дает одновременно работающим пользователям обращаться к изменяемым данным.

**процесс прикладной программы.** (application process) Единица, для которой отводятся ресурсы и выполняются блокировки. Процесс прикладной программы включает в себя выполнение одной или нескольких подпрограмм.

**путь.** (path) Смотрите [путь SQL](#).

**путь SQL.** (SQL path) Упорядоченный список имен схем, используемый при разрешении неспецифицированных ссылок на пользовательские функции, пользовательские типы и хранимые процедуры. В динамическом SQL текущий путь берется из специального регистра CURRENT PATH.

В статическом SQL он определяется опцией связывания PATH.

**путь доступа.** (access path) Путь, используемый для получения данных, указанных в операторе SQL. Путь доступа может включать в себя индексный или последовательный поиск.

## P

**раздел данных.** (data partition) Набор данных VSAM, который содержится в многораздельном табличном пространстве.

**размер.** (size) В языке C – общее число цифр десятичного числа (в SQL этот показатель называется *точностью* числа). В библиотеках DB2 используется определение SQL.

**размер блокировки.** (lock size) Количество данных, доступ к которым регулируется блокировкой DB2; это может быть строка, страница, большой объект, раздел, таблица или табличное пространство.

**размещение плана.** (plan allocation) Процесс размещения ресурсов DB2 для плана при подготовке к его выполнению.

**разрешение неоднозначности.** (indoubt resolution) Процесс разрешения статуса неоднозначной логической единицы работы для выполнения принятия или отката.

**разрешение функции.** (function resolution) Внутренний для DBMS процесс, при котором вызов функции связывается с определенным экземпляром функции. В этом процессе для выбора используются имя функции, типы данных ее аргументов и список применяемых имен схем (называемый *путем SQL*). Процесс называют также *выбором функции*.

**расширение блокировок.** (lock escalation) Перенос блокировок со строк, страниц или больших объектов на все табличное пространство, когда число блокировок страниц для данного ресурса превышает установленный предел.

**реализатор функции.** (function implementer) Идентификатор авторизации владельца программы функции или пакета функции.

**режим блокировки.** (lock mode) Обозначение типа доступа одновременно работающим программам к ресурсам, защищенным блокировкой DB2.

**реквестер.** (requester) Называется также *реквестером прикладных программ (AR)*. Источник требований к удаленной RDBMS, система, которая запрашивает данные.

**реквестер прикладных программ.** (application requester, AR) Смотрите *реквестер*.

**реляционная база данных.** (relational database) База данных, которую можно представить как набор таблиц, обрабатываемый в соответствии с реляционной моделью данных.

**реляционная целостность.** (referential integrity) Условие, которое соблюдается, если все ссылки от данных одного столбца таблицы на данные другого столбца той же или другой таблицы правильны. Поддержание реляционной целостности налагает реляционные ограничения на все операции LOAD, RECOVER, INSERT, UPDATE и DELETE.

**реляционное ограничение.** (referential constraint) Требование, чтобы допустимые непустые значения указанного внешнего ключа соответствовали значениям первичного ключа указанной таблицы.

**реоптимизация.** (reoptimization) Процесс DB2 пересмотра пути доступа для оператора SQL во время выполнения с использованием значений переменных хоста, маркеров параметров или специальных регистров.

**родительская строка.** (parent row) Стока, значение первичного ключа которой является внешним ключом для зависимой строки.

**родительская таблица.** (parent table) Таблица, на первичный ключ которой ссылается внешний ключ зависимой таблицы.

**родительский ресурс блокировки.** (lock parent) При явной иерархической блокировке – ресурс, дочерние блокировки которого расположены ниже в иерархии; обычно родительские блокировки – это блокировки табличного пространства или раздела.

**родительское табличное пространство.** Табличное пространство, содержащее родительскую таблицу. Табличное пространство, содержащее таблицу, которая зависит от данной, называется зависимым табличным пространством.

## C

**связанные столбцы.** (correlated columns) Столбцы, где значение в одном из них связано со значением в другом.

**связанный подзапрос.** (correlated subquery) подзапрос (часть условия WHERE или HAVING), примененный к строке или группе строк таблицы или производной таблицы, названной во внешнем операторе подвыбора.

**связывание.** (bind) Процесс, при котором вывод прокомпилиатора DB2 преобразуется в управляющую структуру, которую может вызывать пакет или план прикладной программы. В ходе этого процесса выбираются пути доступа к данным и выполняются некоторые проверки авторизации.

**автоматическое связывание (automatic bind).**

(Более точно – *автоматическое повторное связывание*). Процесс, при котором операторы SQL связываются автоматически (а не по команде BIND от пользователя), когда процесс прикладной программы начинает выполняться, а связанный пакет или план прикладной программы оказывается недействительным.

**динамическое связывание (dynamic bind).**

Процесс связывания операторов SQL по мере их ввода.

**пошаговое связывание (incremental bind).**

Процесс, при котором операторы SQL связываются при выполнении процесса прикладной программы, поскольку они не могли быть связаны ранее, на стадии связывания, и задано VALIDATE(RUN).

**статическое связывание (static bind).** Процесс, при котором операторы SQL связываются после прокомпиляции. Все статические операторы SQL при этом подготавливаются к выполнению одновременно. Сравните с *динамическим связыванием*.

**секционированный набор страниц.** (partitioned page set) Секционированное табличное пространство или индексное пространство. Страницы заголовков, страницы карт пространства, страницы данных и индексные страницы ссылаются только на данные в пределах раздела.

**сервер.** (server) Называется также *сервером прикладных программ* (*application server, AS*). Система RDBMS, которая принимает требования от удаленной RDBMS и предоставляет им данные.

**сервер прикладных программ.** (*application server, AS*) Смотрите *сервер*.

**символ включения.** (shift-in character) Специальный управляющий символ (X'0F') в системах EBCDIC, означающий, что следующие байты содержат однобайтные символы. Смотрите *символ отключения*.

**символ выделения.** (escape character) Символ–ограничитель идентификатора SQL. Символ выделения – это кавычки (""), кроме программ на языке COBOL, где символ выделения (кавычки или апостроф) может назначать пользователь.

**символ выделения SQL.** (SQL escape character) Символ–ограничитель идентификатора SQL. В

качестве этого символа используются кавычки (""). Смотрите *символ выделения*.

**символ–ограничитель.** (NUL terminator) В С – значение, обозначающее конец строки. Для символьных строк символ–ограничитель – это символ X'00'.

**символ отключения.** (shift-out character)

Специальный управляющий символ (X'0E') в системах EBCDIC, означающий, что следующие байты до ближайшего символа включения содержат двухбайтные символы.

**символ подстановки.** (substitution character)

Определенный символ, который подставляется при преобразовании символов вместо любого символа исходной программы, для которого нет соответствия в представлении кодировки назначения.

**символьная строка.** (character string)

Последовательность байтов, представляющая битовые данные, однобайтные символы или смесь однобайтных и двухбайтных символов.

**символьный большой объект.** (character large object, CLOB) Смотрите *CLOB*.

**сионим.** (synonym) В SQL – альтернативное имя таблицы или производной таблицы. Синоним можно использовать для ссылки на объект только в той подсистеме, где он определен.

**система управления базами данных.** (database management system, DBMS) Программная система, управляющая созданием, структурой, модификацией базы данных и доступом к данным, хранимым в ней.

**система управления реляционными базами данных.** (relational database management system, RDBMS) Менеджер реляционных баз данных, правильно работающий на поддерживаемых системах IBM.

**системный администратор.** (system administrator) Человек со вторым по значимости уровнем полномочий в DB2. Системные администраторы принимают решения о способах использования DB2 и реализуют эти решения, выбирай системные параметры. Они следят за работой системы и меняют ее характеристики в соответствии с изменением требований и целей обработки данных.

**системный диалог.** (system conversation) Диалог, который устанавливают между собой две DB2 для обработки системных сообщений перед началом любой распределенной обработки.

**скалярная функция.** (scalar function) Операция SQL, которая получает единственное значение из другого значения, и обозначается именем функции

со списком ее аргументов в скобках). Смотрите также *функция столбца*.

**собрание.** (collection) Группа пакетов с одним и тем же спецификатором.

**событие триггера.** (triggering event) Указанная в определении триггера операция, которая вызывает активацию триггера. Событие триггера состоит из операции триггера (INSERT, UPDATE или DELETE) и таблицы триггера, над которой эта операция производится.

**совместное использование данных.** (data sharing) Возможность нескольких подсистем DB2 напрямую обращаться к одному и тому же набору данных и модифицировать его.

**совместно используемая блокировка.** (shared lock) Блокировка, которая не дает одновременно выполняемым прикладным процессам изменять данные, но не мешает читать их.

**согласованность данных.** (data currency) Состояние, при котором данные переменных хоста в вашей программе являются копией данных в базовой таблице.

**согласованные данные.** (current data) Данные в хост-структуре, идентичные данным в базовой таблице в базовой таблице.

**согласуемая блокировка.** (negotiable lock) Блокировка, режим которой может быть изменен по соглашению между претендентами, чтобы быть совместимым со всеми участвующими пользователями. Пример согласуемой блокировки – физическая блокировка.

**соединение.** (connection) Существование пути связи между двумя партнерскими LU, позволяющего им обмениваться информацией (например, две системы DB2 соединены и связываются путем диалога).

**соединение адресного пространства.** (address space connection) Результат присоединения внешнего адресного пространства к DB2. У каждого адресного пространства, содержащего связанное с DB2 задание, есть ровно одно соединение адресного пространства, даже если могут присутствовать несколько блоков управления заданиями (TCB). Смотрите *внешнее адресное пространство и блок управления заданием*.

**соединение по собственному протоколу DB2.** (собственный протокол DB2 connection) Собственное соединение DB2 прикладного процесса. Смотрите также *собственное соединение*.

**соединение по собственному протоколу.** (private connection) Соединение связи, характерное для DB2.

**составной ключ.** (composite key) Упорядоченный набор ключевых столбцов в одной таблице.

**сохраняемая блокировка.** (retained lock) Блокировка MODIFY, сохраняемая DB2 при ошибке системы. Эта блокировка сохраняется в структуре блокировок раздела обеспечения взаимодействия при ошибке DB2.

**стабильность на уровне указателя.** (cursor stability, CS) Уровень изоляции, обеспечивающий максимальный уровень параллельности без возможности чтения непринятых данных. При стабильности на уровне указателя единица работы поддерживает блокировки только для непринятых изменений и для текущей строки каждого из своих указателей.

**стабильность чтения.** (read stability, RS) Уровень изоляции, подобные многократному чтению, но не полностью изолирующий прикладной процесс от всех прочих одновременно выполняющихся прикладных процессов. При уровне RS программа, которая посылает один и тот же запрос несколько раз, может читать дополнительные строки, называемые *фантомными строками*, которые вставлены и приняты одновременно выполняющимся прикладным процессом.

**статический SQL.** (static SQL) Операторы SQL, встроенные в программу и подготавливаемые в процессе подготовки программы (до ее выполнения). После подготовки оператор SQL не меняется (хотя могут измениться значения переменных хоста, указанных в нем).

**степень параллелизма.** (degree of parallelism) Число одновременно выполняемых операций, запускаемых для обработки запроса.

**столбец индикатора.** (indicator column) 4-байтное значение, хранящееся в базовой таблице вместо столбца большого объекта.

**страница.** (page) Единица памяти в табличном пространстве (4, 8, 16 или 32 Кбайта) или в индексном пространстве (4 Кбайта). Страница табличного пространства содержит одну или несколько строк таблицы. Для страниц табличного пространства большого объекта значение большого объекта может занимать несколько страниц, но на каждой хранится не более одного значения большого объекта.

**строгая типизация.** (strong typing) Строгая типизация гарантирует, что к пользовательскому типу смогут обращаться только пользовательские

функции и операции, определенные для этого типа. Например, вы не можете непосредственно сравнить суммы в различных валютах, например, в канадских и американских долларах. Но вы можете создать функцию, которая преобразует сумму в одной валюте в сумму в другой и выполнит такое сравнение.

**строка.** (row) Горизонтальный ряд в таблице. Стока состоит из последовательности значений, по одному на каждый столбец в таблице.

**строка.** (string) Смотрите *символьная строка* или *графическая строка*.

**строка оператора.** (statement string) Для динамического оператора SQL – символьная строка, образующая этот оператор.

**строка переменной длины.** (varying-length string) Символьная или графическая строка, длина которой может меняться в заданных пределах. Противоположное понятие *строка фиксированной длины*.

**строка смешанных данных.** (mixed data string) Символьная строка, которая может содержать как однобайтные, так и двухбайтные символы.

**строка фиксированной длины.** (fixed-length string) Символьная или графическая строка, длина которой задана и не может быть изменена. Сравните со *строкой переменной длины*.

**структура блокировки.** (lock structure) Структура данных раздела обеспечения взаимодействия, состоящая из ряда записей блокировок и поддерживающая совместные и монопольные блокировки логических ресурсов.

**структура представления символьных данных.** (character data representation architecture, CDRA) Структура, обеспечивающая правильное представление, обработку и пересылку строковых данных.

**структура хоста.** (host structure) В прикладной программе – структура, на которую ссылаются встроенные операторы SQL.

**схема.** (schema) Логическая группировка пользовательских функций, пользовательских типов, триггеров и хранимых процедур. Когда создается объект одного из этих типов, он помещается в одну из схем, которая определяется по его имени. Например, следующий оператор создает пользовательский тип T в схеме C:

```
CREATE DISTINCT TYPE C.T ...
```

**счетчик заявок.** (claim count) Счетчик числа агентов, обращающихся к объекту.

## T

**таблица.** (table) Именованный объект данных, содержащий определенное число столбцов и некоторое количество неупорядоченных строк. Может быть базовой таблицей или временной таблицей.

**таблица каталога.** (catalog table) Любая таблица в каталоге DB2.

**таблица переходов.** (transition table) Временная таблица, которая содержит все затронутые строки таблицы триггера в их состоянии перед или после события этого триггера. Операторы SQL в определении триггера смогут обращаться к таблице изменяемых строки в старом или в новом состоянии.

**таблица результатов.** (result table) Набор строк, заданный в операторе SELECT.

**таблица триггера.** (triggering table) Таблица, для которой создается триггер. Триггер активируется, когда для этой таблицы происходит событие триггера.

**таблица управления ресурсами.** (resource control table, RCT) Структура утилиты подключения CICS, создаваемая по параметрам макрокоманды от подсистемы, которая определяет атрибуты авторизации и доступа для транзакций или групп транзакций.

**табличная функция.** (table function) Функция, которая получает набор аргументов и возвращает оператору SQL, который к ней обращается, таблицу. К табличной функции можно обращаться только в условии FROM подвыбора.

**табличное пространство.** (table space) Набор страниц, используемый для хранения записей одной или нескольких таблиц.

**табличное пространство большого объекта.** (LOB table space) Аналогично обычному табличному пространству. Табличное пространство большого объекта содержит все данные для определенного столбца большого объекта в соответствующей базовой таблице.

**тело триггера.** (trigger body) Набор операторов SQL, которые выполняются при активации триггера, если условие действия триггера справедливо.

**тип данный, определенный пользователем.** (user-defined data type, UDT) Смотрите *пользовательский тип*.

**типа данных.** (data type) Атрибут столбцов, литералов, переменных хоста, специальных регистров и результатов функций и выражений.

**точка принятия.** (commit point) Момент времени, когда изменения в данных считаются окончательными.

**точка синхронизации.** (sync point) Смотрите *точка принятия*.

**точка согласованности.** (point of consistency) Момент, когда все восстановимые данные, к которым обращается программа, согласованы с другими данными. Синоним *точки синхронизации* или *точки принятия*.

**точность.** (precision) В SQL – общее число цифр десятичного числа (в языке C этот показатель называется *размером числа*). В языке C – число цифр справа от десятичной точки (в SQL этот показатель называется *масштабом числа*). В библиотеках DB2 термин используется в смысле SQL.

**требование принятия.** (request commit) Сигнал на фазе подготовки о том, что участник модифицировал данные и готов к принятию или откату.

**трехчастное имя.** (three-part name) Полное имя таблицы, производной таблицы или алиаса. Состоит из имени положения, ID авторизации и имени объекта, разделяемых точками.

**триггер AFTER.** (after trigger) Триггер, определенный со временем активации AFTER.

**триггер BEFORE.** (before trigger) Триггер, определенный со временем активации BEFORE.

**триггер insert.** (insert trigger) Триггер, определенный с операцией SQL INSERT.

**триггер update.** (update trigger) Триггер, определенный с операцией SQL UPDATE.

**триггер оператора.** (statement trigger) Триггер, определенный с условием триггера FOR EACH STATEMENT.

**триггер строки.** (row trigger) Триггер, определенный с условием триггера FOR EACH ROW.

**триггер удаления.** (delete trigger) Триггер, определенный с операцией SQL DELETE.

**тупиковая ситуация.** (deadlock) Неразрешимый конфликт при использовании такого ресурса как таблица или индекс.

## У

**удаленная подсистема.** (remote subsystem) Любая RDBMS, не являющаяся локальной подсистемой, с которой может связываться пользователь или прикладная программа. Эта подсистема не обязательно является удаленной в физическом смысле, и реально может работать на том же процессоре в той же системе MVS.

**удаленный.** (remote) Относится к любому объекту, поддерживаемому удаленной подсистемой DB2, то есть любой подсистемой DB2, отличной от локальной. Например, *удаленная производная таблица* – это таблица, поддерживаемая удаленной подсистемой DB2. Противоположное понятие – *локальный*.

**уровень изоляции.** (isolation level) Степень изоляции единицы работы от операций модификации данных другими единицами работы. Смотрите также *стабильность на уровне указателя*, *многократное чтение*, *чтение непринятого*, и *стабильность чтения*.

**условие.** (clause) В SQL – отдельная часть оператора, например, условие SELECT или условие WHERE.

**условие действия триггера.** (triggered action condition) Необязательная часть действия триггера. Этот логический критерий выглядит как условие WHEN и задает критерий, который DB2 оценивает, чтобы определить, надо ли выполнять действие триггера.

**условие проверки.** (check clause) Расширение операторов SQL CREATE TABLE и SQL ALTER TABLE, определяющее проверочные ограничения таблицы.

**устройство хранения прямого доступа.** (direct access storage device, DASD) Устройство, для которого время доступа не зависит от положения данных.

**утилита ограничения ресурсов.** (resource limit facility, RLF) Часть кода DB2, которая не дает динамическим операторам SQL превышать заданные ограничения по времени.

**утилита подключения CICS.** (CICS attachment facility) Подкомпонент DB2, который использует интерфейс MVS SSI (Subsystem Interface) и межпространственные связи для обработки требований от CICS к DB2 и для координации принятия ресурсов.

**утилита подключения IMS.** (IMS attachment facility) Подкомпонент DB2, который использует протоколы интерфейса MVS SSI (Subsystem Interface) и

межпространственные связи для обработки требований от IMS к DB2 и для координации принятия ресурсов.

**утилита подключения TSO.** (TSO attachment facility) Утилита DB2, состоящая из процессора команд DSN и DB2I. Те программы, которые не написаны для сред CICS или IMS, могут выполняться при помощи утилиты подключения TSO.

**утилита подключения по вызову.** (call attachment facility, CAF) Способ подключения DB2 для прикладных программ, выполняемых в TSO или MVS Batch. CAF – альтернатива процессору команд DSN, допускает больше возможностей управления средой выполнения.

**утилита распределенных данных.** (distributed data facility, DDF) Набор компонентов DB2, при помощи которых DB2 связывается с другими RDBMS.

## Ф

**фаза принятия.** (committed phase) Вторая фаза процесса изменения на нескольких узлах, когда все участники принимают изменения, внесенные логической единицей работы.

**физическая блокировка.** (physical lock, P-lock) Тип блокировки, используемый только для совместного использования данных, который применяется DB2, чтобы обеспечить согласованность данных кешей в разных подсистемах DB2.

**физическая согласованность.** (physical consistency) Состояние страницы, которая не находится в состоянии частичного изменения.

**фрагмент.** (piece) Набор данных в однораздельном наборе страниц.

**функция.** (function) Вычисление или действие определенного назначения, например, функция столбца или скалярная функция. (Смотрите *функция столбца* и *скалярная функция*.)

Кроме того, функции делятся на пользовательские, встроенные и генерируемые DB2. (Смотрите *встроенная функция*, *функция изменения типа*, *пользовательская функция*, *внешняя функция*, *функция с источником*.)

**функция изменения типа.** (cast function) Функция, преобразующая данные исходного типа в данные другого типа (типа назначения). Обычно функция изменения типа носит то же имя, что и тип данных назначения. Имеет один простой аргумент исходного типа данных и возвращает данные типа назначения.

**функция с источником.** (sourced function) Функция, которая реализуется при помощи другой встроенной или определенной пользователем функции, уже известной менеджеру баз данных. Это может быть скалярная функция или функция столбца, принимающая набор значений и возвращающая одно значение (например, MAX или AVG). Сравните с *внешней функцией* и *встроенной функцией*.

**функция столбца.** (column function) Операция SQL, которая получает результат на основе набора значений в одной или нескольких строках. Сравните со *скалярной функцией*.

## Х

**хранимая процедура.** (stored procedure)

Написанная пользователем прикладная программа, которую можно вызвать при помощи оператора SQL CALL.

## Ц

**центральный процессор.** (central processor, CP)

Часть компьютера, служащая для выполнения программ, начальной загрузки и других операций компьютера.

## Ч

**член плана.** (plan member) Связанная копия DBRM, указанная в условии member.

**член совместного использования данных.** (data sharing member) Подсистема DB2, отнесенная службами XCF к группе совместного использования данных.

## Э

**элемент разграничения.** (delimiter token)

Строковая константа, идентификатор с ограничителями, символ операции или любой специальный символ, показанный на синтаксических диаграммах.

## Я

**явная иерархическая блокировка.** (explicit hierarchical locking)

Блокировка, используемая для установления отношений родитель/потомок между известными IRLM ресурсами. Такая блокировка выполняется, чтобы избежать издержек, связанных с глобальной блокировкой, когда ресурс не используется несколькими подсистемами DB2.

## **язык хоста**

**язык хоста.** (host language) Язык программирования, в который можно вставлять операторы SQL.

# Библиография

**Библиотеки продукта DB2 Universal Database Server for OS/390 Версия 6:**

## **DB2 Universal Database for OS/390**

- *DB2 Administration Guide, SC26-9003*
- Руководство по прикладному программированию и языку SQL для DB2, SH43-0083
- *DB2 Application Programming Guide and Reference for Java™, SC26-9018*
- *DB2 ODBC Guide and Reference, SC26-9005*
- *DB2 Command Reference, SC26-9006*
- *DB2 Data Sharing: Planning and Administration, SC26-9007*
- *DB2 Data Sharing Quick Reference Card, SX26-3843*
- *DB2 Diagnosis Guide and Reference, LY36-3736*
- *DB2 Diagnostic Quick Reference Card, LY36-3737*
- *DB2 Image, Audio, and Video Extenders Administration and Programming, SC26-9650*
- *DB2 Installation Guide, GC26-9008*
- *DB2 Licensed Program Specifications, GC26-9009*
- *DB2 Messages and Codes, GC26-9011*
- *DB2 Master Index, SC26-9010*
- *DB2 Reference for Remote DRDA Requesters and Servers, SC26-9012*
- *DB2 Reference Summary, SX26-3844*
- *DB2 Release Planning Guide, SC26-9013*
- *DB2 SQL Reference, SC26-9014*
- *DB2 Text Extender Administration and Programming, SC26-9651*
- *DB2 Utility Guide and Reference, SC26-9015*
- *DB2 Что нового? GH43-0082*
- *DB2 Program Directory, GI10-8182*

## **DB2 Administration Tool**

- *DB2 Administration Tool for OS/390 User's Guide, SC26-9847*

## **DB2 Buffer Pool Tool**

- *DB2 Buffer Pool Tool for OS/390 User's Guide and Reference, SC26-9306*

## **DB2 DataPropagator**

- *DB2 Replication Guide and Reference, SC26-9642*

## **Net.Data for OS/390**

Следующие книги доступны по адресу <http://www.software.ibm.com/data/net.data/library.html>:

- *Net.Data Library: Administration and Programming Guide for OS/390*
- *Net.Data Library: Language Environment Interface Reference*
- *Net.Data Library: Messages and Codes*
- *Net.Data Library: Reference*

## **DB2 PM for OS/390**

- *DB2 PM for OS/390 Batch User's Guide, SC26-9167*
- *DB2 PM for OS/390 Command Reference, SC26-9166*
- *DB2 PM for OS/390 General Information, GC26-9172*
- *DB2 PM for OS/390 Installation and Customization, SC26-9171*
- *DB2 PM for OS/390 Messages, SC26-9169*
- *DB2 PM for OS/390 Online Monitor User's Guide, SC26-9168*
- *DB2 PM for OS/390 Report Reference Volume 1, SC26-9164*
- *DB2 PM for OS/390 Report Reference Volume 2, SC26-9165*
- *DB2 PM for OS/390 Using the Workstation Online Monitor, SC26-9170*
- *DB2 PM for OS/390 Program Directory, GI10-8183*

## **утилита управления запросами**

- утилита управления запросами: *Developing QMF Applications, SC26-9579*
- утилита управления запросами: *Getting Started with QMF on Windows, SC26-9582*
- утилита управления запросами: *High Performance Option User's Guide for OS/390, SC26-9581*
- утилита управления запросами: *Installing and Managing QMF on OS/390, GC26-9575*
- утилита управления запросами: *Installing and Managing QMF on Windows, GC26-9583*
- утилита управления запросами: *Introducing QMF, GC26-9576*
- утилита управления запросами: *Messages and Codes, GC26-9580*
- утилита управления запросами: *Reference, SC26-9577*
- утилита управления запросами: *Using QMF, SC26-9578*

## **Ada/370**

- *IBM Ada/370 Language Reference, SC09–1297*
- *IBM Ada/370 Programmer's Guide, SC09–1414*
- *IBM Ada/370 SQL Module Processor for DB2 Database Manager User's Guide, SC09–1450*

## **APL2**

- *APL2 Programming Guide, SH21–1072*
- *APL2 Programming: Language Reference, SH21–1061*
- *APL2 Programming: Using Structured Query Language (SQL), SH21–1057*

## **AS/400**

- *DB2 for OS/400 SQL Programming, SC41–4611*
- *DB2 for OS/400 SQL Reference, SC41–4612*

## **BASIC**

- *IBM BASIC/MVS Language Reference, GC26–4026*
- *IBM BASIC/MVS Programming Guide, SC26–4027*

## **BookManager READ/MVS**

- *BookManager READ/MVS V1R3: Installation Planning & Customization, SC38–2035*

## **C/370**

- *IBM SAA AD/Cycle C/370 Programming Guide, SC09–1841*
- *IBM SAA AD/Cycle C/370 Programming Guide for Language Environment/370, SC09–1840*
- *IBM SAA AD/Cycle C/370 User's Guide, SC09–1763*
- *SAA CPI C Reference, SC09–1308*

## **Character Data Representation Architecture**

- *Character Data Representation Architecture Overview, GC09–2207*
- *Character Data Representation Architecture Reference and Registry, SC09–2190*

## **CICS/ESA**

- *CICS/ESA Application Programming Guide, SC33–1169*
- *CICS for MVS/ESA Application Programming Reference, SC33–1170*
- *CICS for MVS/ESA CICS–RACF Security Guide, SC33–1185*
- *CICS for MVS/ESA CICS–Supplied Transactions, SC33–1168*
- *CICS for MVS/ESA Customization Guide, SC33–1165*
- *CICS for MVS/ESA Data Areas, LY33–6083*
- *CICS for MVS/ESA Installation Guide, SC33–1163*
- *CICS for MVS/ESA Intercommunication Guide, SC33–1181*

- *CICS for MVS/ESA Messages and Codes, GC33–1177*
- *CICS for MVS/ESA Operations and Utilities Guide, SC33–1167*
- *CICS/ESA Performance Guide, SC33–1183*
- *CICS/ESA Problem Determination Guide, SC33–1176*
- *CICS for MVS/ESA Resource Definition Guide, SC33–1166*
- *CICS for MVS/ESA System Definition Guide, SC33–1164*
- *CICS for MVS/ESA System Programming Reference, GC33–1171*

## **CICS/MVS**

- *CICS/MVS Application Programmer's Reference, SC33–0512*
- *CICS/MVS Facilities and Planning Guide, SC33–0504*
- *CICS/MVS Installation Guide, SC33–0506*
- *CICS/MVS Operations Guide, SC33–0510*
- *CICS/MVS Problem Determination Guide, SC33–0516*
- *CICS/MVS Resource Definition (Macro), SC33–0509*
- *CICS/MVS Resource Definition (Online), SC33–0508*

## **IBM C/C++ for MVS/ESA**

- *IBM C/C++ for MVS/ESA Library Reference, SC09–1995*
- *IBM C/C++ for MVS/ESA Programming Guide, SC09–1994*

## **IBM COBOL**

- *IBM COBOL Language Reference, SC26–4769*
- *IBM COBOL for MVS & VM Programming Guide, SC26–4767*

## **Conversion Guide**

- *IMS–DB and DB2 Migration and Coexistence Guide, GH21–1083*

## **Cooperative Development Environment**

- *CoOperative Development Environment/370: Debug Tool, SC09–1623*

## **Data Extract (DXT)**

- *Data Extract Version 2: General Information, GC26–4666*
- *Data Extract Version 2: Planning and Administration Guide, SC26–4631*

## **DataPropagator NonRelational**

- *DataPropagator NonRelational MVS/ESA Administration Guide, SH19–5036*

- *DataPropagator NonRelational MVS/ESA Reference, SH19–5039*

## **Data Facility Data Set Services**

- *Data Facility Data Set Services: User's Guide and Reference, SC26–4388*

## **Database Design**

- *DB2 Design and Development Guide, Gabrielle Wiorkowski and David Kull, Addison Wesley, ISBN 0–20158–049–8*
- *Handbook of Relational Database Design, C. Fleming and B. Von Halle, Addison Wesley, ISBN 0–20111–434–8*

## **DataHub**

- *IBM DataHub General Information, GC26–4874*

## **DB2 Connect**

- *DB2 Connect Enterprise Edition for OS/2 and Windows NT: Quick Beginnings, GC09–2828*
- *DB2 Connect Personal Edition Quick Beginnings, GC09–2830*
- *DB2 Connect User's Guide, SC09–2838*

## **DB2 Server for VSE & VM**

- *DB2 Server for VM: DBS Utility, SC09–2394*
- *DB2 Server for VSE: DBS Utility, SC09–2395*

## **DB2 Universal Database (UDB)**

- *DB2 UDB Administration Guide Volume 1: Design and Implementation, SC09–2839*
- *DB2 UDB Administration Guide Volume 2: Performance, SC09–2840*
- *DB2 UDB Administrative API Reference, SC09–2841*
- *DB2 UDB Application Building Guide, SC09–2842*
- *DB2 UDB Call Level Interface Guide and Reference, SC09–2843*
- *DB2 UDB SQL Getting Started, SC09–2856*
- *DB2 UDB SQL Reference Volume 1, SC09–2847*
- *DB2 UDB SQL Reference Volume 2, SC09–2848*

## **Device Support Facilities**

- *Device Support Facilities User's Guide and Reference, GC35–0033*

## **DFSMS/MVS**

- *DFSMS/MVS: Access Method Services for the Integrated Catalog, SC26–4906*
- *DFSMS/MVS: Access Method Services for VSAM Catalogs, SC26–4905*
- *DFSMS/MVS: Administration Reference for DFSMSdss, SC26–4929*
- *DFSMS/MVS: DFSMSshm Managing Your Own Data, SH21–1077*

- *DFSMS/MVS: Diagnosis Reference for DFSMSdfp, LY27–9606*
- *DFSMS/MVS: Macro Instructions for Data Sets, SC26–4913*
- *DFSMS/MVS: Managing Catalogs, SC26–4914*
- *DFSMS/MVS: Program Management, SC26–4916*
- *DFSMS/MVS: Storage Administration Reference for DFSMSdfp, SC26–4920*
- *DFSMS/MVS: Using Advanced Services, SC26–4921*
- *DFSMS/MVS: Utilities, SC26–4926*
- *MVS/DFP: Using Data Sets, SC26–4749*

## **DFSORT**

- *DFSORT Application Programming: Guide, SC33–4035*

## **Distributed Relational Database**

- *Data Stream and OPA Reference, SC31–6806*
- *Distributed Relational Database Architecture: Application Programming Guide, SC26–4773*
- *Distributed Relational Database Architecture: Connectivity Guide, SC26–4783*
- *Distributed Relational Database Architecture: Evaluation and Planning Guide, SC26–4650*
- *Distributed Relational Database Architecture: Problem Determination Guide, SC26–4782*
- *Distributed Relational Database: Every Manager's Guide, GC26–3195*
- *IBM SQL Reference, SC26–8416*
- *Open Group Technical Standard (Open Group в настоящее время сделала следующие книги доступными на своем сайте по адресу [www.opengroup.org](http://www.opengroup.org)):*
  - *DRDA Volume 1: Distributed Relational Database Architecture (DRDA), ISBN 1–85912–295–7*
  - *DRDA Volume 3: Distributed Database Management (DDM) Architecture, ISBN 1–85912–206–X*

## **Domain Name System**

- *DNS and BIND, Третье издание, Paul Albitz and Cricket Liu, O'Reilly, SR23–8771*

## **Education**

- *IBM Dictionary of Computing, McGraw–Hill, ISBN 0–07031–489–6*
- *1999 IBM All-in-One Education and Training Catalog, GR23–8105*

## **Enterprise System/9000 and Enterprise System/3090**

- *Enterprise System/9000 и Enterprise System/3090 Processor Resource/System Manager Planning Guide, GA22–7123*

## **High Level Assembler**

- *High Level Assembler for MVS and VM and VSE Language Reference, SC26-4940*
- *High Level Assembler for MVS and VM and VSE Programmer's Guide, SC26-4941*

## **Parallel Sysplex Library**

- *OS/390 Parallel Sysplex Application Migration, GC28-1863*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1862*
- *OS/390 Parallel Sysplex Overview: An Introduction to Data Sharing and Parallelism, GC28-1860*
- *OS/390 Parallel Sysplex Systems Management, GC28-1861*
- *OS/390 Parallel Sysplex Test Report, GC28-1963*
- *System/390 9672/9674 System Overview, GA22-7148*

## **ICSF/MVS**

- *ICSF/MVS General Information, GC23-0093*

## **IMS/ESA**

- *IMS Batch Terminal Simulator General Information, GH20-5522*
- *IMS/ESA Administration Guide: System, SC26-8013*
- *IMS/ESA Administration Guide: Transaction Manager, SC26-8731*
- *IMS/ESA Application Programming: Database Manager, SC26-8727*
- *IMS/ESA Application Programming: Design Guide, SC26-8016*
- *IMS/ESA Application Programming: Transaction Manager, SC26-8729*
- *IMS/ESA Customization Guide, SC26-8020*
- *IMS/ESA Installation Volume 1: Installation and Verification, SC26-8023*
- *IMS/ESA Installation Volume 2: System Definition and Tailoring, SC26-8024*
- *IMS/ESA Messages and Codes, SC26-8028*
- *IMS/ESA Operator's Reference, SC26-8030*
- *IMS/ESA Utilities Reference: System, SC26-8035*

## **ISPF**

- *ISPF V4 Dialog Developer's Guide and Reference, SC34-4486*
- *ISPF V4 Messages and Codes, SC34-4450*
- *ISPF V4 Planning and Customizing, SC34-4443*
- *ISPF V4 User's Guide, SC34-4484*

## **Language Environment**

- *Debug Tool User's Guide and Reference, SC09-2137*

## **National Language Support**

- *National Language Support Reference Volume 2, SE09-8002*

## **NetView**

- *NetView Installation and Administration Guide, SC31-8043*
- *NetView User's Guide, SC31-8056*

## **ODBC**

- *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide, Microsoft Press, ISBN 1-55615-658-8*

## **OS/390**

- *OS/390 C/C++ Programming Guide, SC09-2362*
- *OS/390 C/C++ Run-Time Library Reference, SC28-1663*
- *OS/390 eNetwork Communications Server: IP Configuration, SC31-8513*
- *OS/390 Hardware Configuration Definition Planning, GC28-1750*
- *OS/390 Information Roadmap, GC28-1727*
- *OS/390 Introduction and Release Guide, GC28-1725*
- *OS/390 JES2 Initialization and Tuning Guide, SC28-1791*
- *OS/390 JES3 Initialization and Tuning Guide, SC28-1802*
- *OS/390 Language Environment for OS/390 & VM Concepts Guide, GC28-1945*
- *OS/390 Language Environment for OS/390 & VM Customization, SC28-1941*
- *OS/390 Language Environment for OS/390 & VM Debugging Guide, SC28-1942*
- *OS/390 Language Environment for OS/390 & VM Programming Guide, SC28-1939*
- *OS/390 Language Environment for OS/390 & VM Programming Reference, SC28-1940*
- *OS/390 MVS Diagnosis: Procedures, LY28-1082*
- *OS/390 MVS Diagnosis: Tools and Service Aids, LY28-1085*
- *OS/390 MVS Initialization and Tuning Guide, SC28-1751*
- *OS/390 MVS Initialization and Tuning Reference, SC28-1752*
- *OS/390 MVS Installation Exits, SC28-1753*
- *OS/390 MVS JCL Reference, GC28-1757*
- *OS/390 MVS JCL User's Guide, GC28-1758*
- *OS/390 MVS Planning: Global Resource Serialization, GC28-1759*
- *OS/390 MVS Planning: Operations, GC28-1760*
- *OS/390 MVS Planning: Workload Management, GC28-1761*
- *OS/390 MVS Programming: Assembler Services Guide, GC28-1762*
- *OS/390 MVS Programming: Assembler Services Reference, GC28-1910*
- *OS/390 MVS Programming: Authorized Assembler Services Guide, GC28-1763*

- *OS/390 MVS Programming: Authorized Assembler Services Reference, Тома 1–4, GC28–1764, GC28–1765, GC28–1766, GC28–1767*
- *OS/390 MVS Programming: Callable Services for High-Level Languages, GC28–1768*
- *OS/390 MVS Programming: Extended Addressability Guide, GC28–1769*
- *OS/390 MVS Programming: Sysplex Services Guide, GC28–1771*
- *OS/390 MVS Programming: Sysplex Services Reference, GC28–1772*
- *OS/390 MVS Programming: Workload Management Services, GC28–1773*
- *OS/390 MVS Routing and Descriptor Codes, GC28–1778*
- *OS/390 MVS Setting Up a Sysplex, GC28–1779*
- *OS/390 MVS System Codes, GC28–1780*
- *OS/390 MVS System Commands, GC28–1781*
- *OS/390 MVS System Messages Volume 1, GC28–1784*
- *OS/390 MVS System Messages Volume 2, GC28–1785*
- *OS/390 MVS System Messages Volume 3, GC28–1786*
- *OS/390 MVS System Messages Volume 4, GC28–1787*
- *OS/390 MVS System Messages Volume 5, GC28–1788*
- *OS/390 MVS Using the Subsystem Interface, SC28–1789*
- *OS/390 Security Server (RACF) Auditor's Guide, SC28–1916*
- *OS/390 Security Server (RACF) Command Language Reference, SC28–1919*
- *OS/390 Security Server (RACF) General User's Guide, SC28–1917*
- *OS/390 Security Server (RACF) Introduction, GC28–1912*
- *OS/390 Security Server (RACF) Macros and Interfaces, SK2T–6700 (OS/390 Collection Kit), SK27–2180 (OS/390 Security Server Information Package)*
- *OS/390 Security Server (RACF) Security Administrator's Guide, SC28–1915*
- *OS/390 Security Server (RACF) System Programmer's Guide, SC28–1913*
- *OS/390 SMP/E Reference, SC28–1806*
- *OS/390 SMP/E User's Guide, SC28–1740*
- *OS/390 RMF User's Guide, SC28–1949*
- *OS/390 TSO/E CLISTS, SC28–1973*
- *OS/390 TSO/E Command Reference, SC28–1969*
- *OS/390 TSO/E Customization, SC28–1965*
- *OS/390 TSO/E Messages, GC28–1978*
- *OS/390 TSO/E Programming Guide, SC28–1970*
- *OS/390 TSO/E Programming Services, SC28–1971*
- *OS/390 TSO/E User's Guide, SC28–1968*
- *OS/390 DCE Administration Guide, SC28–1584*

- *OS/390 DCE Introduction, GC28–1581*
- *OS/390 DCE Messages and Codes, SC28–1591*
- *OS/390 OS/390 UNIX System Services Command Reference, SC28–1892*
- *OS/390 OS/390 UNIX System Services Planning, SC28–1890*
- *OS/390 OS/390 UNIX System Services User's Guide, SC28–1891*

## **PL/I for MVS & VM**

- *IBM PL/I MVS & VM Language Reference, SC26–3114*
- *IBM PL/I MVS & VM Programming Guide, SC26–3113*

## **OS PL/I**

- *OS PL/I Programming Language Reference, SC26–4308*
- *OS PL/I Programming Guide, SC26–4307*

## **Prolog**

- *IBM SAA AD/Cycle Prolog/MVS & VM Programmer's Guide, SH19–6892*

## **Remote Recovery Data Facility**

- *Remote Recovery Data Facility Program Description and Operations, LY37–3710*

## **Storage Management**

- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26–3123*
- *MVS/ESA Storage Management Library: Leading a Storage Administration Group, SC26–3126*
- *MVS/ESA Storage Management Library: Managing Data, SC26–3124*
- *MVS/ESA Storage Management Library: Managing Storage Groups, SC26–3125*
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide, SC26–4659*

## **System/370 and System/390**

- *ESA/370 Principles of Operation, SA22–7200*
- *ESA/390 Principles of Operation, SA22–7201*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28–1210*

## **System Network Architecture (SNA)**

- *SNA Formats, GA27–3136*
- *SNA LU 6.2 Peer Protocols Reference, SC31–6808*
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2, GC30–3084*
- *SNA/Management Services Alert Implementation Guide, GC31–6809*

## TCP/IP

- *IBM TCP/IP for MVS: Customization & Administration Guide, SC31-7134*
- *IBM TCP/IP for MVS: Diagnosis Guide, LY43-0105*
- *IBM TCP/IP for MVS: Messages and Codes, SC31-7132*
- *IBM TCP/IP for MVS: Planning and Migration Guide, SC31-7189*

## VS COBOL II

- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, GC26-4047*
- *VS COBOL II Installation and Customization for MVS, SC26-4048*

## VS FORTRAN

- *VS FORTRAN Version 2: Language and Library Reference, SC26-4221*
- *VS FORTRAN Version 2: Programming Guide for CMS and MVS, SC26-4222*

## VTAM

- *Planning for NetView, NCP, and VTAM, SC31-8063*
- *VTAM for MVS/ESA Diagnosis, LY43-0069*
- *VTAM for MVS/ESA Messages and Codes, SC31-6546*
- *VTAM for MVS/ESA Network Implementation Guide, SC31-6548*
- *VTAM for MVS/ESA Operation, SC31-6549*
- *VTAM for MVS/ESA Programming, SC31-6550*
- *VTAM for MVS/ESA Programming for LU 6.2, SC31-6551*
- *VTAM for MVS/ESA Resource Definition Reference, SC31-6552*

# Индекс

## Спецсимволы

(подчеркивание)

- переменная хоста в ассемблере 148
- предикат LIKE 30
- символьная 30
- : (двоеточие)
  - COBOL 188
  - FORTRAN 208
  - перед переменной хоста 111
  - переменная хоста FORTRAN 208
  - переменная хоста PL/I 220
  - переменная хоста ассемблера 150
  - переменная хоста в С 164
  - переменная хоста языка COBOL 188
  - программа С 176
- "грязное" чтение 375
  - См.также UR (чтение непринятого)
- % (знак процента)
- предикат LIKE 29

## A

- ABRT, параметр CAF (утилиты подключения по вызову) 787, 799
- ACQUIRE
  - опция подкоманды BIND PLAN
  - блокировка таблиц и табличных пространств 369
- ALL
  - уточненный предикат 87
- AND
  - операция в условии WHERE 31
- ANY
  - уточненный предикат 87
- ASCII
  - данные, чтение из DB2 for OS/390 570
- ATTACH, опция
  - CAF 777
  - RRSAF 812
  - прекомпилятор 777, 812
- AUTH SIGNON
  - функция соединения RRSAF
  - использование 825
  - синтаксис 825

## B

- BIND PACKAGE, подкоманда DSN
  - опции
    - ISOLATION 373
    - RELEASE 369
  - опции, связанные с доступом по протоколу
    - DRDA 413, 415

BIND PACKAGE, подкоманда DSN (*продолжение*)

параметры

REOPT(VARS) 682

SQLERROR 413

имя-положения 413

удаленная 451

BIND PLAN, подкоманда DSN

опции

ACQUIRE 369

ISOLATION 373

RELEASE 369

опции, связанные с доступом по протоколу

DRDA 414

параметры

CACHESIZE 459

DISCONNECT 414

REOPT(VARS) 682

SQLRULES 414, 460

удаленная 451

BTS (batch terminal simulator – пакетный симулятор терминала) 513

## C

CACHESIZE

опция подкоманды BIND PLAN 459

подкоманда REBIND 459

CAF (утилита подключения по вызову)

выполнение прикладных программ 772

коды возврата

CLOSE 787

CONNECT 781

DISCONNECT 789

OPEN 785

TRANSLATE 791

проверка 800

ограничения 769

описание 769

описания функций 779

параметры 779

прикладные программы

подготовка 771

примеры 798

соглашение об использовании регистров 779

соединение с DB2 799

среда выполнения 772

структур загружаемого модуля 773

язык программирования 770

CALL DSNALI, оператор 779, 793

CALL DSNRLI, оператор 815

CCSID (идентификатор кодового набора символов)

SQLDA 569

CD-ROM, книги на 10  
CDSSRDEF, параметр подсистемы 757  
CICS  
  единица работы 394  
  логическая единица работы 394  
  модуль языкового интерфейса (DSNCLI)  
    использование при компоновке прикладной  
      программы 464  
  обработка памяти  
    С 181  
    COBOL 203  
    ассемблер 160  
  планирование  
    среда 469  
  подпрограмма DSNTIAC  
    С 181  
    COBOL 203  
    PL/I 232  
    ассемблер 160  
 поток  
  повторное использование 849  
программирование  
  команда SYNCPOINT 394  
  макрокоманда DFHEIENT 149  
  примеры прикладных программ 885, 888  
 работа  
  выполнение программ 507  
  системный сбой 395  
 средства  
  EDF (execution diagnostic facility – утилита  
    диагностики выполнения) 514  
  управляющие области 507  
 управление памятью  
  PL/I 232  
 утилита подключения  
  замечания по программированию 849  
  управление из прикладных программ 849  
 утилиты  
  транслятор языка команд 449  
CLOSE  
  оператор  
    описание 129  
    условие WHENEVER NOT FOUND 562, 574  
 функция соединения CAF  
  использование 787  
  описание 774  
  пример программы 799  
  синтаксис 787  
COMMA  
  опция препроцессора 442  
CONNECT  
  оператор  
    SPUFI 96  
    тип 1 418  
  опция препроцессора 442  
  функция соединения CAF (утилиты подключения  
    по вызову)  
CONNECT (продолжение)  
  функция соединения CAF (утилиты подключения  
    по вызову) (продолжение)  
    использование 781, 785  
    описание 774  
    пример программы 799  
    синтаксис 781, 785  
CONTINUE  
  условие оператора WHENEVER 118  
CREATE THREAD  
  функция соединения RRSAF  
    пример программы 846  
CS (стабильность на уровне указателя)  
  блокировка страниц и строк 374  
CURRENTDATA, опция команды BIND  
  разные опции для плана и для пакета 380  
CURRENT DEGREE, поле панели DSNTIP4 757  
CURRENT DEGREE, специальный регистр  
  изменение значения по умолчанию  
  подсистемы 757  
CURRENT RULES специальный регистр  
  использование 460  
CURRENT SQLID, специальный регистр  
  использование при тестировании 507

## D

DATE  
  опция препроцессора 442  
DB2, аварийное завершение 778, 814  
DB2I (DB2 Interactive)  
  SPUFI 93  
  выбор  
    DCLGEN (генератор объявлений) 138  
    SPUFI 93  
  меню 93  
  обработка EDITJCL  
    библиотеки времени выполнения 484  
  обработка в фоновом режиме  
    библиотеки времени выполнения 484  
панели  
  BIND PACKAGE 489  
  BIND PLAN 492  
  Current SPUFI Defaults (текущие умолчания  
    SPUFI) 96  
  DCLGEN 134, 141  
  компиляция, компоновка и запуск 503  
  меню основных опций DB2I 93, 476  
  параметры по умолчанию для BIND PLAN 496  
  Подготовка программ 478  
  прекомпиляция 486  
    типы соединений системы 500  
  подготовка программ 475  
  прерывание 100  
  пример подготовки программы 478

**DBCS** (двуихбайтный набор символов)  
 константы 219  
**DBCS** (набор двухбайтных символов)  
 имена таблиц 133  
 использование меток с DCLGEN 136  
 трансляция в CICS 449  
**DBPROTOCOL(DRDA)**  
 повышает производительность для  
 распределенных систем 421  
**DBRM** (модуль требований базы данных)  
 как связывать 343  
 описание 440  
**DEC31**  
 опция прекомпилятора 442  
**DECIMAL**  
 константы 176  
 тип данных  
     совместимость с С 174  
 функция  
     использование 43  
     язык С 175  
**DEFER(PREPARE)**  
 повышает производительность для  
 распределенных систем 421  
**DELETE**  
 оператор  
     когда избегать 74  
     описание 73  
     опция CASCADE 73  
     подзапросы 87  
     правила 73  
     проверка кодов возврата 116  
     связанные подзапросы 91  
     условие WHERE CURRENT 129  
**DFSLI000** (модуль языкового интерфейса IMS) 464  
**DISCONNECT**  
 функция соединения CAF  
 использование 789  
 описание 774  
 пример программы 799  
 синтаксис 789  
**DISTINCT**  
 условие оператора SELECT 24  
**DL/I**  
 batch  
     ID контрольной точки 535  
     модуль DSNMTV01 531  
     набор входных данных DDITV02 528  
     параметр SSM= 531  
     передача программы на выполнение 531  
     разработка программ 526  
     свойства 525  
     вызов TERM 395  
     пакетная среда  
     требования к DB2 526  
**DSN\_FUNCTION\_TABLE** 322  
**DSN**, команда TSO  
 командный процессор  
     службы, утрачиваемые под CAF 772  
**DSN**, команда в среде TSO  
 обработка кода возврата 466  
 подкоманды  
     См.также individual subcommands  
     RUN 465  
**DSN**, прикладные программы, выполнение с  
 использованием CAF 772  
**DSNALI** (модуль языкового интерфейса CAF)  
 загрузка 798  
 удаление 798  
**DSNCLI** (CICS модуль языкового интерфейса)  
 включение в компоновку 464  
**DSNELI** (модуль языкового интерфейса TSO) 772  
**DSNH**, команда TSO 520  
 См.также прекомпилятор  
 получение выходных данных SYSTEM 520  
**DSNHDECP**  
 неявное соединение CAF 775  
**DSNHLI**, точка входа в DSNALI  
 неявные вызовы 775  
**DSNHLI**, точка входа для DSNALI  
 пример программы 804  
**DSNHLI**, точка входа для DSNRLI  
 пример программы 845  
**DSNHLI2**, точка входа для DSNALI 803  
**DSNRLI** (модуль языкового интерфейса RRSAF)  
 загрузка 844  
 удаление 844  
**DSNTEDIT CLIST** 957  
**DSNTIAUL**, пример программы 510  
**DSNTRACE**, набор данных 796  
**DYNAMICRULES**  
 влияние на прикладные программы 457

## E

**ECB** (блок управления событиями)  
**CONNECT**, функция соединения CAF 781, 785  
**CONNECT**, функция соединения RRSAF 817  
 адрес в списке параметров CALL DSNALI 779  
 пример программы 799, 803  
 создание программ для CAF (утилиты  
     подключения по вызову) 799  
**EDIT panel, SPUFI**  
 операторы SQL 99  
 пустая 99  
**ESTAE**, процедура в CAF (средстве подключения  
 вызовов) 795  
**EXCLUSIVE**  
 режим блокировки  
     большого объекта 390  
     влияние на ресурсы 365  
     страница 364

**EXCLUSIVE** (*продолжение*)  
режим блокировки (*продолжение*)  
строки 364  
таблицы, раздела и табличного  
пространства 364

**EXPLAIN**  
оператор  
исследование работы SQL 701  
объяснение данных 711  
описание 701  
просмотр индекса 713  
отчет о внешнем объединении 730  
параметр  
использование при автоматическом  
связывании 349

## F

**FLOAT**  
опция препроцессора 442

**FOLD**  
значение для С и CPP 443  
значение опции препроцессора HOST 443  
**FRR** (процедура функционального  
восстановления) 795, 796  
**FULL OUTER JOIN** 78  
См.также операция объединения  
пример 78

## G

**GRANT**, оператор  
полномочия 509  
**GRAPHIC**  
опция препроцессора 442

**H**  
**HOST**

значение FOLD для С и CPP 443  
опция препроцессора 443

**I**  
**IDENTIFY**

функция соединения RRSAF (средства  
подключений службы управления  
восстановимыми ресурсами)  
использование 817  
пример программы 846  
синтаксис 817

**IKJEFT01**, программа монитора терминала в  
TSO 467

**IMS**

восстановление 395  
вызов CHKP 395  
вызов ROLB 395, 402

**IMS** (*продолжение*)  
вызов ROLL 395, 402  
вызов SYNC 395  
вызовы контрольной точки 396  
единица работы 395, 396  
модуль языкового интерфейса DFSLI000  
компонентовка 464  
обработка ошибок 398  
ограничения 397  
пакет 403  
планирование  
среда 468  
программы 399  
точка принятия 396

**IN**

предикат 33  
условие в подзапросах 88  
**INNER JOIN** 76  
См.также операция объединения  
пример 76  
**INSERT**, выполнение  
влияние опции MEMBER CLUSTER команды  
CREATE TABLESPACE 357  
**INTENT EXCLUSIVE**, режим блокировки 365, 390  
**INTENT SHARE**, режим блокировки 365, 390  
табличного пространства большого объекта 390  
**Interactive System Productivity Facility (ISPF)** 93  
См.также ISPF (Interactive System Productivity  
Facility)

**IRLM** (internal resource lock manager – менеджер  
блокировки внутренних ресурсов) 532  
См.также IRLM (internal resource lock manager)  
описание 532

**ISOLATION**  
опция подкоманды BIND PLAN  
влияние на блокировки 373

**ISPF**  
Меню DB2I 476  
подготовка  
панель подготовки программ 478  
прекомпиляция под 476

**ISPF** (Interactive System Productivity Facility –  
интерактивная утилита производительности  
системы)  
программирование 768  
**ISPF** (Interactive System Productivity Facility)  
DB2 использует диалоговое управление 93  
команда прокрутки 102  
просмотр 96, 100

## J

**JCL** (job control language – язык управления  
заданием)  
пример пакетного восстановления 533

JCL (язык управления заданиями)  
запуск пакетной программы TSO 467  
процедуры прекомпиляции 469

## K

KEEPDYNAMIC  
опция подкоманды BIND 550

## L

LEFT OUTER JOIN 79  
См. также операция объединения  
пример 79  
LOAD, используемая RRSAF макрокоманда  
MVS 809  
LOAD, макрокоманда MVS, используемая CAF 771  
LOCK TABLE, оператор  
влияние на блокировки 383  
LOCKPART, условие команд CREATE и ALTER  
TABLESPACE  
влияние на блокировку 362  
LOCKSIZE, условие  
рекомендации 358

## M

MEMBER CLUSTER, опция CREATE  
TABLESPACE 357  
MIXED DATA  
передача в удаленное положение 429  
MVS  
31-битная адресация 504  
31-битовая адресация 464

## N

NULL  
атрибут оператора UPDATE 71  
опция условия WHERE 27  
указатель в С 163

## O

OPEN  
оператор  
без маркеров параметров 573  
открытие указателя 126  
подготовленный SELECT 561  
производительность 744  
условие USING DESCRIPTOR 576  
функция соединения CAF  
использование 785  
описание 774  
пример программы 799  
синтаксис 785

## Q

QUOTE  
опция прекомпилятора 444

## R

RCT (таблица управления ресурсами)  
определение DB2 в CICS 450  
прикладные программы 469  
тестирование программ 507  
трансляция программы 450  
REBIND PACKAGE, подкоманда DSN  
опции  
ISOLATION 373  
RELEASE 369  
удаленная 451  
REBIND PLAN, подкоманда DSN  
опции  
ACQUIRE 369  
ISOLATION 373  
RELEASE 369  
RELEASE  
опция подкоманды BIND PLAN  
сочетание с другими опциями 369  
RIB (блок информации о выпуске)  
CONNECT, функция соединения CAF 781  
CONNECT, функция соединения RRSAF 817  
адрес в списке параметров CALL DSNALI 779  
пример программы 799  
RIGHT OUTER JOIN 79  
См. также операция объединения  
пример 79  
ROWID  
вставка в таблицу 853  
доступ только через индекс 714  
пример программного кода 717  
RR (многократное чтение)  
блокировка страниц и строк 373  
как сохраняются блокировки (рисунок) 373  
RRSAF (утилита подключения служб управления  
восстановимыми ресурсами)  
коды возврата  
AUTH SIGNON 825  
CONNECT 817  
SIGNON 822  
TERMINATE IDENTIFY 836  
TERMINATE THREAD 835  
TRANSLATE 837  
ограничения 807  
описание 807  
описания функций 816  
прикладные программы  
подготовка 809  
примеры 844  
соглашение об использовании регистров 816

RRSAF (утилита подключения служб управления восстановимыми ресурсами) (продолжение)  
соединение с DB2 846  
среда выполнения 810  
структура загрузочного модуля 811  
транзакции  
использование глобальных транзакций 360  
язык программирования 808

RS (стабильность чтения)  
блокировка страниц и строк (рисунок) 374

RUN  
подкоманда DSN  
запуск программы в основном режиме в среде TSO 465  
обработка кода возврата 466  
ограничение CICS 450

## S

SET CURRENT DEGREE, оператор 757

SHARE  
INTENT EXCLUSIVE, режим блокировки 365, 390  
режим блокировки  
большого объекта 390  
страница 364  
строки 364  
таблицы, раздела и табличного пространства 364

SIGNON  
функция соединения RRSAF  
использование 822  
синтаксис 822  
функция соединения RRSAF (средства подключений службы управления восстановимыми ресурсами)  
пример программы 846

SOURCE  
опция препроцессора 445

SPUFI  
возвращаемый SQLCODE 100  
завершение операторов SQL 99  
задание ограничителя SQL 96  
значения по умолчанию 96  
изменение ширины столбцов 101  
обработка операторов SQL 93, 100  
панели  
выбор в меню DB2I 93  
заполнение 94  
предыдущие значения выводятся на панели 93  
размещение выходного набора данных 95  
форматирование и показ вывода 100  
поле CONNECT LOCATION 96  
просмотр вывода 100  
создание заголовков столбцов 102

SQL  
динамический  
пример программы на языке С 913  
опция препроцессора 445  
статический  
пример программы на языке С 913

SQL (Structured Query Language – язык структурированных запросов)  
переменные хоста 110  
структуры 110

SQL (Structured Query Language – язык структурированных запросов), зарезервированные 965  
динамический  
разрешенные операторы 967

SQL (Structured Query Language – язык структурированных запросов)  
коды возврата  
обработка 119

SQL (Structured Query Language — язык структурированных запросов)  
коды возврата  
проверка 116

SQL (Structured Query Language)  
выражение CASE 46  
динамический  
кодирование 543  
кодирование  
C 160  
C++ 160  
COBOL 181  
PL/I 216  
ассемблер 145  
динамический 577  
основы 107  
программа на языке FORTRAN 206

написание программ  
объектные расширения 251  
ограничитель строк 484  
проверка синтаксиса 412  
с переменным списком 562, 576  
символ выделения 441  
указатели 123

SQL область связи (SQLCA) 119  
См. также SQLCA (область связи SQL)

SQL-INIT-FLAG, переустановка 187

SQLCA (SQL communication area – область связи SQL)  
C 161  
COBOL 181  
FORTRAN 204  
ассемблер 145  
подпрограмма DSNTIAC  
C 181  
COBOL 203  
ассемблер 160

SQLCA (SQL communication area – область связи SQL) (*продолжение*)  
подпрограмма DSNTIAR  
    C 179  
    COBOL 202  
    FORTRAN 215  
    ассемблер 159  
    пример программы на языке C 913  
SQLCA (SQL communication area – область связи с SQL)  
    PL/I 216  
    подпрограмма DSNTIAC  
        PL/I 232  
    подпрограмма DSNTIAR  
        PL/I 231  
SQLCA (SQL область связи)  
    код причины для истечения срока ожидания 353  
    код причины для тупиковой ситуации 355  
SQLCA (область связи SQL)  
    описание 116  
SQLCODE  
    +004 788, 790  
    +100 118  
    +256 795, 796  
    +802 118  
    -510 380  
    -923 529  
    -925 402, 525  
    -926 402, 525  
SQLDA (SQL descriptor area – область дескрипторов SQL)  
    C 161, 563  
    COBOL 183  
    FORTRAN 205  
    PL/I 216, 563  
    ассемблер 146  
    без вхождений SQLVAR 565  
    выделение памяти 566  
    для типов большой объект (LOB) и пользовательских типов 571  
    маркеры параметров 575  
    оператор OPEN 561  
    оператор SELECT с переменным списком 563  
    параметр функции CAF TRANSLATE 791  
    параметр функции RRSAF TRANSLATE 837  
    пример динамического SELECT 569  
    программа на ассемблере 563  
    требует адреса памяти 569  
SQLERROR  
    условие оператора WHENEVER 118  
SQLRULES 460  
SQLSTATE  
    '01519' 118  
    '2D521' 402, 525  
    '57015' 529

SSID (идентификатор подсистемы), задание 483  
SSN (имя подсистемы)  
вызовы SQL через CAF (утилита подключения по вызову) 775  
параметр в функции RRSAF CONNECT 817  
параметр функции CAF CONNECT 781  
параметр функции CAF OPEN 785  
список параметров CALL DSNALI 779  
STOP DATABASE, команда  
    превышение срока ожидания 354  
SYNC, параметр CAF (утилиты подключения по вызову) 787, 799  
SYSPRINT  
    выходные данные прекомпилятора  
        использование для анализа ошибок 521  
    раздел исходного текста операторов,  
        пример 522  
    раздел опций 521  
    раздел сводки, пример 523  
    раздел символических перекрестных ссылок 522  
SYSTEM, выходные данные для анализа ошибок 520

## T

TCB (блок управления заданием)  
    возможности при использовании CAF 770  
    возможности при использовании RRSAF 808  
    вызов функции CAF CLOSE 788  
    вызов функции CAF OPEN 786  
TERMINATE IDENTIFY  
    функция соединения RRSAF  
        использование 836  
        пример программы 846  
        синтаксис 836  
TERMINATE THREAD  
    функция соединения RRSAF  
        использование 835  
        пример программы 846  
        синтаксис 835  
TEST, команда TSO 512  
TMP (программа монитора терминала)  
    запуск в среде TSO 467  
    командный процессор DSN 465  
TRANSLATE, функция CAF  
    использование 791  
    пример программы 803  
    синтаксис 791  
TRANSLATE, функция RRSAF  
    использование 837  
    синтаксис 837  
TRANSLATE, функция соединения CAF  
    описание 774  
TSO  
    DSNALI, модуль языкового интерфейса 772

**TSO** (*продолжение*)  
TEST, команда 512  
единица работы, завершение 395  
процедуры CLIST  
вызов прикладных программ 468  
запуск в основном режиме 468  
**TWOPASS**  
опция препроцессора 446

## U

**UPDATE**  
оператор  
описание 71  
подзапросы 87  
связанные подзапросы 91  
условие SET 71  
условие WHERE CURRENT 128  
режим блокировки  
страница 364  
строки 364  
таблицы, раздела и табличного  
пространства 364  
**UR** (чтение непринятого)  
блокировка страниц и строк 375  
влияет на чтение больших объектов 389  
ограничения на одновременный доступ 377  
рекомендации 359  
**USER**  
значение в операторе UPDATE 71  
специальный регистр 71

## V

**VALUES**  
условие оператора INSERT 66  
**VERSION**  
опция препроцессора 446, 456  
Visual Explain 693, 701

## W

**WITH**, условие  
задает уровень изоляции 382

## A

аварийное завершение  
DB2 778, 814  
IMS  
U0102 534  
U0775 400  
U0778 402  
действие на положение указателя 130  
для вызовов синхронизации 528  
код возврата, передаваемый RRSAF  
CONNECT 817

аварийное завершение (*продолжение*)  
код возврата, посылаемый CAF CONNECT 781  
коды причины 796  
обработчики 795  
программа 394  
программа одиночного режима 396  
программы множественного режима 396  
система  
Х'04E' 525  
аварийные завершения вызова синхронизации 528  
автоматический  
повторное связывание  
работа EXPLAIN 709  
автоматическое  
повторное связывание  
неверный план или пакет 348  
адресное пространство  
завершение  
команда CAF CLOSE 788  
команда CAF DISCONNECT 789  
инициализация  
команда CAF CONNECT 783  
команда CAF OPEN 785  
отдельные задания 770, 808  
примеры сценариев 794, 841  
апостроф  
опция 441  
опция препроцессора STRING DELIMITER 441  
арифметические выражения в операторе  
UPDATE 71

## Б

база данных  
пример прикладной программы 880  
библиотека  
электронная 10  
библиотеки времени выполнения, DB2I  
обработка EDITJCL 484  
обработка в фоновом режиме 484  
блок информации о выпуске (RIB) 779  
См.также RIB (release information block)  
блок управления заданием (TCB) 770, 808  
См.также TCB (task control block)  
См.также TCB (блок управления заданием)  
блокировка  
блокировка больших объектов 387  
блокировки страниц  
описание 361  
сравнение CS, RS и RR 373  
влияние опций  
многократное чтение 373  
программы 368  
путь доступа 385  
связывания 369  
стабильность на уровне указателя 374  
стабильность чтения 374

блокировка (*продолжение*)  
влияние опций (*продолжение*)  
чтение непринятого 375  
влияние указателя WITH HOLD 381  
длительность  
больших объектов 391  
описание 363  
управление 369  
единица работы 394, 395  
единицы работы 393, 395  
иерархия  
описание 361  
избежание 379  
класс  
транзакции 351  
объект  
индексы 368  
описание 367  
описание 351  
преимущества 352  
размер  
раздел 361  
страница 361  
таблица 361  
табличное пространство 361  
расширение  
при получении большого количества строк 858  
режим 364  
рекомендации для одновременности 357  
сводная информация 386  
совместимость 366  
эффекты  
превышение срока ожидания 353  
приостановка 353  
тупиковая ситуация 354  
блокировка транзакций  
описание 351  
блочная выборка  
использование 424  
предотвращение 429  
стабильность на уровне указателя 429  
большого объекта  
блокировка  
одновременность с читающими процессами с изоляцией UR 377  
описание 387  
большой объект  
блокировка 387  
длительность блокировки 391  
команды LOCK TABLE 392  
режимы блокировки табличного пространства 390  
режимы блокировок больших объектов 390  
большой объект (LOB)  
локатор 262

большой объект (LOB) (*продолжение*)  
материализация 262  
объявление локаторов больших объектов 257  
объявление переменных хоста 257  
описание 253  
пространство данных 262  
с переменными-индикаторами 266

## B

ввод набора данных DDITV02 528  
версия пакета  
идентификация 456  
вложенное табличное выражение 82  
обработка 745  
внешнее объединение 78  
*См.также* операция объединения  
FULL OUTER JOIN  
пример 78  
LEFT OUTER JOIN  
пример 79  
RIGHT OUTER JOIN  
пример 79  
материализация 731  
отчет EXPLAIN 730  
восстановление 394  
*См.также* единица работы  
завершение 393  
определение требований к программе 400  
планирование 393  
прикладная программа IMS 395  
программа IMS 401  
временная таблица 62  
временные таблицы  
просмотр табличного пространства 722  
вспомогательные таблицы  
команды LOCK TABLE 392  
вставка нескольких строк 69  
выбор  
всех столбцов 22  
некоторых столбцов 23  
нескольких строк 112  
по критериям 29  
столбцы без имен 23  
строк 26  
указанных столбцов 23  
вывод  
вычисление значений 34  
справки  
столбцы таблицы 55  
таблиц 54  
вызов CHKP, IMS 395  
вызов ROLB, IMS  
в пакетных программах 402  
заканчивает единицу работы 395  
преимущества перед ROLL 402

вызов ROLL, IMS  
в пакетных программах 402  
заканчивает единицу работы 395

вызов SYNC 396

вызов SYNC, IMS 395

вызов TERM в DL/I 395

вызов XRST, программа IMS 398

выполнение прикладной программы  
ошибки 518

выполнение слияния  
производные таблицы или вложенные табличные выражения 745

выражение  
результаты 34  
столбцы 34

выражение CASE 46

вычисление значений  
вывод 34  
группировка с критериями 51  
получение значений по группам 51  
результаты 34

## Г

гибридное объединение  
описание 736

глобальная транзакция  
поддержка RRSAF 823, 827, 830

глобальная транзакция RRS  
поддержка RRSAF 823, 827, 830

графические переменные хоста  
С 166  
PL/I 222  
ассемблер 152

групповые программы IMS 399

## Д

данные  
влияние блокировок на целостность 352  
добавление в конец таблицы 859  
защита и целостность 393  
не табличное хранение 860  
обновление во время получения 857  
обновление ранее полученных данных 857  
обратный просмотр 853  
получение больших объемов 858  
получение набора строк 127  
получение с помощью SELECT \* 858  
понимание доступа 701  
связанные с условием WHERE 26  
согласованность 429  
спорное состояние 397  
улучшение доступа 701

дата–время  
арифметика 34

двоеточие  
перед переменной хоста 111  
переменная хоста FORTRAN 208  
переменная хоста PL/I 220  
переменная хоста ассемблера 150  
переменная хоста в С 164  
переменная хоста языка COBOL 188

двуфазное принятие  
координация изменений 416

декартово объединение 733

деление на ноль 118

десятичная арифметика 35

диагностика ошибок  
основные правила 518

диапазон значений, поиск 32

динамические операторы SQL 577  
PL/I 563  
PREPARE и EXECUTE 556, 558  
без операторов SELECT 555, 558  
в программе на языке FORTRAN 207  
влияние опции связывания REOPT(VARS) 576  
влияние указателя WITH HOLD 558  
использование DESCRIBE INPUT 559  
кэширование  
влияние опции связывания RELEASE 370  
кэширование подготовленных операторов 547  
ограничения 545  
оператор EXECUTE IMMEDIATE 555  
операторы SELECT с переменным списком 562, 576  
операторы SELECT с фиксированным списком 559, 562  
описание 543  
преимущества и недостатки 544  
пример программы на языке С 913  
программа на ассемблере 563  
программа на языке С 563  
программа на языке COBOL 185, 577  
программирование 543, 577  
разрешенные операторы 967  
требования 545  
языки хостов 554

динамический выбор плана  
использование пакетов 462  
ограничения для специального регистра CURRENT PACKAGESET 462

длительность блокировок  
описание 363  
управление 369

доступ DRDA  
в сравнении с доступом по собственному протоколу DB2 407  
использование 410  
кодирование программы 408  
опции прекомпилятора 413  
опции связывания 413, 414

доступ DRDA (*продолжение*)  
освобождение соединений 411  
планирование 405, 407  
подготовка программ 413  
пример 407, 410  
пример программы 917  
смешанная среда 967  
советы по программированию 412  
доступ по собственному протоколу DB2  
в сравнении с доступом по протоколу DRDA 407  
кодирование программы 408  
планирование 405, 407  
пример 406  
пример программы 925  
смешанная среда 967

## E

единица восстановления  
неоднозначная  
восстановление CICS 395  
спорные  
восстановление IMS 397  
единица работы  
DL/I batch 401  
IMS  
конец 395  
начальная точка 395  
пакет 401  
точка принятия 396  
TSO  
выполнение 393  
оператор ROLLBACK 394  
блокировки доступа других пользователей к  
данным 393  
длительность 393  
завершение  
TSO 393, 395  
откатка 394  
открытые указатели 129  
принятие 394  
начало 393  
описание 393  
описание в среде CICS 394

## 3

завершение  
плана, используя функцию CAF CLOSE 788  
зависимая  
таблица 72  
загрузка  
данные  
DSNTIAUL 510  
замечания, об авторских правах 977

запуск прикладных программ  
CICS 469  
IMS 468  
зарезервированные ключевые слова 965  
защита и целостность данных 393  
заявка  
влияние указателя WITH HOLD 382  
знак процента 29  
значение  
null 27  
в диапазоне значений 32  
подобное символьной строке 29  
список 33

## I

изменение  
больших объемов 857  
во время получения 857  
значения из переменных хоста 112  
имя подсистемы (SSN) 775  
*См. также SSN (subsystem name)*  
индекс  
блокировки 368  
способы доступа  
выбор пути доступа 723  
несколько 725  
по несогласованному индексу 725  
просмотр индекса с одной выборкой 727  
просмотр индекса с IN-списком 725  
согласованное с индексом описание 723  
согласованные с индексом столбцы 713

## K

ключ  
уникальный 853  
ключевые слова, зарезервированные 965  
книги DB2, электронные 10  
код возврата  
SQL 788  
*См. также SQLCODE*  
команда DSN 466  
код причины  
CAF  
X'00C10824' 788, 790  
X'00F30050' 796  
X'00F30083' 795  
преобразование 796, 803  
X'00C90088' 355  
X'00C9008E' 353  
X'00D44057' 525  
кодирование  
операторы SQL  
C 160  
C++ 160  
COBOL 181

кодирование (*продолжение*)  
операторы SQL (*продолжение*)  
FORTRAN 204  
PL/I 215  
ассемблер 145  
динамический 543  
команды LOCK TABLE  
действие на вспомогательные таблицы 392  
компоновка  
опция AMODE 504  
опция RMODE 504  
прикладные программы 463  
конец данных 127  
конец указателя 127  
конкатенация  
данных из нескольких столбцов 34  
ключевое слово (CONCAT) 34  
операция 34  
константа  
COBOL 187  
ассемблер 149  
синтаксис  
C 176  
FORTRAN 213  
контрольная точка  
вызовы 396, 398  
частота 401  
копирование  
таблиц из удаленных положений 429  
критерий поиска  
операторы SELECT 85  
операции сравнения 27  
с использованием условия WHERE 27  
кэширование  
динамические операторы SQL  
влияние опции связывания  
RELEASE(DEALLOCATE) 370  
кэшировать операторы динамического SQL 547

## Л

лицензированная программа DataPropagator  
Relational 429  
логическая единица работы  
описание в среде CICS 394  
локатор набора результатов  
C 168  
COBOL 192  
FORTRAN 209  
PL/I 222  
ассемблер 152  
использование 639  
пример 639

## М

макрокоманда DFHEIENT 149  
маркер параметра  
динамические операторы SQL 557  
несколько 558  
предоставляемые OPEN значения 561  
с произвольными операторами 574, 576  
маркеры параметров  
преобразование 324  
массовое удаление  
конфликты с процессом UR 377  
материализация  
LOB 262  
внешнее объединение 731  
производные таблицы и вложенные табличные выражения 746  
метка, столбец 571  
модули языкового интерфейса  
DSNCLI  
опция компоновки AMODE 464  
модуль DSNMTV01 531

## Н

набор входных данных DDITV02 528  
набор выходных данных DDOTV02 530  
наборы данных SYSLIB 470  
настройка  
DB2  
запросы, содержащие переменные хоста 682  
начальное задание 757  
неоднозначный указатель 380  
несвязанные подзапросы 688  
См.также subquery  
несегментированное табличное пространство  
просмотр 722

## О

область действия блокировки 361  
область связи SQL (SQLCA) 116  
обработка  
операторы SQL 100  
обработка ввода–вывода  
параллельная  
запросы 756  
обработка исключительных ситуаций 117  
обработка пакетной среды  
совместный доступ к DB2 и DL/I  
вызовы контрольных точек 527  
обработка пакетов  
пакетная программа DB2  
запуск 467  
запуск при помощи CLIST 468

обработка ситуаций внимания 770, 795, 808  
 объект блокировки 367  
 объектно–ориентированная программа  
     подготовка 475  
 объявление  
     в прикладной программе 138  
 генератор (DCLGEN) 133  
     См.также подкоманда DCLGEN команды DSN  
 переменные в примере программы для CAF 804  
 ограничение 61  
     См.также проверочное ограничение таблицы  
 ограниченный просмотр разделов 718  
 ограничитель  
     строки 441  
 ограничитель SQL  
     задание в SPUFI 96  
 одновременность  
     влияние  
         опций ISOLATION 373, 374, 377  
         размера блокировки 363  
          чтение непринятого 375  
     описание 351  
     рекомендации 357  
     управление с помощью блокировок 352  
 оператор  
     метки  
         FORTRAN 207  
         PL/I 218  
 оператор ALLOCATE CURSOR  
     использование 638  
 оператор ASSOCIATE LOCATORS  
     использование 638  
 оператор COMMIT  
     завершение единицы работы 393  
     когда выполнять 394  
     описание 95  
 оператор CREATE TABLE  
     использование 57  
 оператор DECLARE CURSOR  
     опция WITH RETURN 603  
     подготовленный оператор 561, 564  
 оператор DECLARE TABLE  
     описание 109  
     описание таблицы 133  
 оператор DECLARE в языке COBOL 185  
 оператор DESCRIBE  
     метки столбцов 571  
     условия INTO 565, 567  
 оператор DESCRIBE CURSOR  
     использование 639  
 оператор DESCRIBE INPUT  
     использование 559  
 оператор DESCRIBE PROCEDURE  
     использование 637  
 оператор EXECUTE  
     динамическое выполнение 558  
 оператор EXECUTE (*продолжение*)  
     типы параметров 574  
     условие USING DESCRIPTOR 576  
 оператор EXECUTE IMMEDIATE  
     динамическое выполнение 555  
 оператор FETCH  
     переменные хоста 561  
     просмотр данных 853  
     условие USING DESCRIPTOR 574  
 оператор INCLUDE  
     вывод DCLGEN 138  
 оператор INSERT  
     для столбца ROWID 70  
     несколько строк 69  
     описание 66  
     подзапросы 87  
     условие VALUES 66  
 оператор PREPARE  
     динамическое выполнение 557  
     переменная хоста 561  
     условие INTO 565  
 Оператор RELEASE  
     оператор 411  
 оператор ROLLBACK  
     единица работы в TSO 394  
     описание 95  
     ошибка в IMS 525  
 оператор SELECT  
     выбор набора строк 123  
     изменение формата результатов 101  
     использование с  
         \* (для выбора всех столбцов) 22  
         оператором DECLARE CURSOR 125  
         списком имен столбцов 23  
     критерий поиска 85  
     маркеры параметров 574  
     подзапросы 85  
     с переменным списком 562, 576  
     с фиксированным списком 559, 562  
     столбцы без имен 23  
     указанные столбцы 23  
     условия  
         DISTINCT 24  
         FROM 22  
         GROUP BY 51  
         HAVING 51  
         ORDER BY 47  
         UNION 52  
         WHERE 26  
 оператор SET CURRENT PACKAGESET 454  
 оператор SYNCPOINT в среде CICS 394, 395  
 оператор WHENEVER  
     C 164  
     COBOL 186  
     FORTRAN 207  
     PL/I 218

оператор WHENEVER (*продолжение*)  
    ассемблер 148  
    коды ошибок SQL 117

оператор динамического SQL  
    кэширование 547

оператором DECLARE CURSOR  
    описание 124, 125  
    опция WITH HOLD 130

операторы SQL  
    ALLOCATE CURSOR 638  
    ASSOCIATE LOCATORS 638  
    CALL  
        ограничения на 599  
    CLOSE 129, 562  
    CONNECT (типа 1) 418  
    CONNECT (типа 2) 418  
    DECLARE CURSOR  
        описание 125  
        пример 561, 564  
    DECLARE TABLE 109, 133  
    DELETE  
        описание 129  
        пример 73  
    DESCRIBE 565  
    DESCRIBE CURSOR 639  
    DESCRIBE PROCEDURE 637  
    EXECUTE 558  
    EXECUTE IMMEDIATE 555  
    EXPLAIN  
        текущий контроль пути доступа 701  
    FETCH  
        описание 127  
        пример 561  
    INSERT  
        строки 66  
    OPEN  
        описание 126  
        пример 561  
    PREPARE 557  
    SELECT  
        %, \_ 29  
        несколько критериев 31  
        объединение таблиц 75  
        объединение таблицы с этой же таблицей 77  
        операция AND 31  
        операция NOT 28  
        операция OR 31  
        описание 26  
        предикат BETWEEN 32  
        предикат IN 33  
        предикат LIKE 29  
        скобки 31  
    SET CURRENT DEGREE 757  
    UPDATE  
        описание 128  
        пример 71

операторы SQL (*продолжение*)  
    WHENEVER 117  
    встроенные 439  
    коды возврата ошибок 119  
    обозначения set 149  
    оператор RELEASE  
        с доступом по протоколу DRDA 411

перенос на следующую строку  
    COBOL 184  
    FORTRAN 206  
    PL/I 217  
    ассемблер 147  
    язык С 163

операция NOT в условии WHERE 28

операция OR в условии WHERE 31

операция объединения  
    FULL OUTER JOIN  
        пример 78  
    INNER JOIN  
        пример 76  
    LEFT OUTER JOIN  
        пример 79  
    RIGHT OUTER JOIN  
        пример 79  
    вложенное табличное выражение 82  
    вложенный цикл 731  
    гибридное  
        описание 736  
    декартово 733  
    объединение таблиц 75  
    объединение таблицы с этой же таблицей 77  
    описание 728  
    последовательность объединения 737  
    просмотр слияния 733  
    семантика SQL 80  
    табличные пользовательские функции 82

опция APOST  
    прекомпилятор 441

опция APOSTSQL  
    прекомпилятор 441

опция ATTACH  
    прекомпилятор 441

опция DYNAM языка COBOL 186

опция FLAG  
    прекомпилятор 442

опция FLOAT  
    прекомпилятор 442

опция KEEP UPDATE LOCKS условия WITH 382

опция LINECOUNT  
    прекомпилятор 443

опция NODYNAM языка COBOL 186

опция NOFOR  
    прекомпилятор 444

опция NOOPTIONS  
    прекомпилятор 444

опция OPTIONS  
прекомпилятор 444  
опция PARMs  
запуск в основном режиме 466  
опция PERIOD  
прекомпилятор 444  
опция QUOTESQL  
прекомпилятор 445  
опция SQLFLAG  
прекомпилятор 445  
опция STDSQL  
прекомпилятор 446  
опция TIME  
прекомпилятор 446  
опция XREF  
прекомпилятор 446  
опция компоновки AMODE 464, 504  
опция компоновки–редактирования RMODE 504  
опция прекомпилиатора DEC15 442  
опция прекомпилиатора LEVEL 443  
опция прекомпилиатора MARGINS 443  
опция прекомпилиатора NOGRAPHIC 444  
опция прекомпилиатора NOSOURCE 444  
опция прекомпилиатора NOXREF 444  
опция прекомпилиатора ONEPASS 444  
отбрасывание  
таблиц 64  
откат  
использование RRSAF 809  
откатка  
опция оператора CICS SYNCPOINT 395  
отладка прикладных программ 511  
отношение кластеризации  
влияние на просмотр табличного  
пространства 722  
с предварительной выборкой списка 740  
отображение макрокоманд  
программы на ассемблере 160  
ошибка  
арифметическое выражение 118  
выполнение 518  
коды возврата 116  
обработка 117  
сообщения, генерируемые  
прекомпилятором 519, 520

## П

пакет  
связывание  
PLAN\_TABLE 703  
пакетная обработка  
совместный доступ к DB2 и DL/I  
загрузка 531  
запуск 531  
компоновка плана 530  
прекомпиляция 530

пакетная обработка (*продолжение*)  
совместный доступ к DB2 и DL/I (*продолжение*)  
принятие 527  
пакеты  
выбор 454  
идентификация версии 456  
как использовать 343  
неработоспособные  
условия 348  
повторное связывание с подстановками 346  
подсистема 454  
преимущества 343  
связывание  
EXPLAIN при удаленном связывании 710  
модуля DBRM с пакетом 450  
с планами 453  
удаленная 451  
список  
связывание плана 453  
указание при запуске 454  
память  
адреса в SQLDA 569  
запрос  
SQLDA 566  
полученная строка 568  
панель  
Current SPUFI Defaults (текущие умолчания  
SPUFI) 96  
DCLGEN 134, 141  
DSNEDP01 134, 141  
DSNEPRI 93  
DSNESP01 93  
DSNESP02 96  
EDIT (для входного набора данных SPUFI) 99  
SPUFI 93  
меню основных опций DB2I 93  
параллелизм запросов 753  
параллелизм запросов Sysplex  
распределение обработки больших запросов  
между подсистемами DB2 753  
параллельная обработка  
настройка 762  
описание 753  
разрешение 757  
соответствующие столбцы PLAN\_TABLE 720  
перевод  
требований конечных пользователей в операторы  
SQL 859  
перезапуск  
программы DL/I batch, использующие JCL 534  
переменная  
объявление  
C 174  
COBOL 198  
FORTRAN 212  
PL/I 227  
программа на ассемблере 156

переменная (*продолжение*)  
 хоста  
     C 165  
     COBOL 188  
     FORTRAN 209  
     PL/I 221  
     ассемблер 151  
 переменная перехода  
     триггер 237  
 переменная хоста  
     C 164, 165  
     COBOL 188  
     FORTRAN 208, 209  
     PL/I 220  
     SELECT  
         условие программы на COBOL 112  
     ассемблер 150  
     в предикате равенства 684  
     влияние на выбор путь доступа 682  
     вставка в таблицы 112  
     гибкость статического SQL 544  
     графическая  
         PL/I 222  
         ассемблер 152  
     графический  
         C 166  
     именование структур  
         программа на языке PL/I 223  
     именование структуры  
         программа C 169  
     настройка запросов 682  
     оператор EXECUTE IMMEDIATE 556  
     оператор FETCH 561  
     оператор PREPARE 561  
     оператор UPDATE 71  
     описание 110  
     пример запрос 682  
     пример использования в программе на  
         COBOL 111  
     символьная  
         C 165  
         COBOL 190  
         FORTRAN 209  
         PL/I 222  
         ассемблер 151  
     условие WHERE в программе на языке  
         COBOL 113  
 переменная–индикатор  
     C 178  
     COBOL 201  
     FORTRAN 214  
     PL/I 160, 229  
     задание пустых значений в программе на языке  
         COBOL 114  
     объявление массива в DCLGEN 137  
     описание 113

переменная–индикатор (*продолжение*)  
     программа на ассемблере 158  
     структуры в программе на COBOL 115  
 переполнение 118  
 перечень ошибок программы  
     информация об ошибочных ситуациях 511  
     сообщения об ошибках 512  
 план программы  
     динамический выбор плана для прикладных  
         программ CICS 462  
     использование пакетов 343  
     неработоспособный  
         условия 348  
     повторное связывание  
         изменение пакетов 347  
     связывание 453  
     создание 450  
     создание списка пакетов 453  
 планирование  
     восстановления 393  
     доступ к распределенным данным 405, 429  
     одновременность 349, 392  
     прекомпиляция 342  
     связывание 342, 349  
 повторная оптимизация пути доступа 682  
 повторное связывание 347, 451  
     См.также REBIND PACKAGE subcommand of DSN  
     См.также подкоманда DSN REBIND PLAN  
     автоматически  
         работа EXPLAIN 709  
     автоматическое  
         условия 348  
     используемые пакеты или планы 342  
     наборы пакетов с подстановками 346  
     параметры 342  
     планирование 349  
     планы 347  
     после изменений 345  
     списки планов и пакетов 957  
     списки планов или пакетов 957  
 подвыбор  
     оператор INSERT 70  
 подготовка программы 435  
     См.также прикладные программы  
 подготовленный оператор SQL  
     кэширование 550  
     разрешенные операторы 967  
 подзапрос  
     использование с UPDATE, DELETE и INSERT 87  
     настройка 687  
     несвязанный 688  
     ограничения для DELETE 92  
     оператор DELETE 91  
     оператор UPDATE 91  
     описание 85  
     преобразование в объединение 690

подзапрос (*продолжение*)  
примеры настройки 691  
реляционные связи 92  
связанный  
настройка 687  
оператор DELETE 91  
оператор UPDATE 91  
подзапрос 89  
пример 89  
подкоманда BIND PACKAGE DSN  
параметры  
KEEPDYNAMIC 550  
подкоманда BIND PLAN DSN  
параметры  
KEEPDYNAMIC 550  
подкоманда DCLGEN DSN  
построение объявлений данных 133  
подкоманда DCLGEN команды DSN  
включение объявлений в программу 138  
запуск 133  
идентификация таблиц 133  
использование 133  
обявление массива  
переменных–индикаторов 137  
оператор INCLUDE 138  
построение объявлений данных 133  
пример 140  
формирование имен переменных хоста 137  
подкоманда DSN REBIND PACKAGE  
повторное связывание с подстановками 346  
формирование списка 957  
подкоманда DSN REBIND PLAN  
параметры  
NOPKLIST 347  
PKLIST 347  
удаленная 451  
формирование списка 957  
подпрограмма DSNTIAC  
C 181  
COBOL 203  
PL/I 232  
ассемблер 160  
подпрограмма DSNTIAR  
C 179  
COBOL 202  
FORTRAN 215  
PL/I 231  
ассемблер 159  
описание 119  
подпрограмма DSNTIR 215  
подсистема  
идентификатор (SSID), задание 483  
показатель фильтрации  
предикат 669  
поле AUTOCOMMIT на панели SPUFI 95  
поле CONNECT LOCATION панели SPUFI 96  
поле EXPLAIN PROCESSING на панели DSNTIPO  
издержки 709  
поле RELEASE LOCKS на панели DSNTIP4 381  
поле SQLN SQLDA  
DESCRIBE 565  
поле SQLVAR SQLDA 568  
полномочия  
создание тестовых таблиц 509  
получение  
больших объемов данных 858  
в диапазоне значений 32  
данные в ASCII из DB2 for OS/390 570  
данные, изменение CCSID 570  
данных с помощью SELECT \* 858  
получение значений по группам 51  
пользовательская функция  
вызов из триггера 240  
разрешенные операторы 970  
пользовательская функция (UDF)  
DSN\_FUNCTION\_TABLE 322  
аварийное завершение работы 325  
автор вызывающей программы 268  
вызов  
синтаксис 317  
вызов из предиката 325  
главная программа 275  
доступ к таблицам переходов 307  
задание выходных значений 283  
использование временной памяти 306  
использование специальных регистров 304  
как вызвать 317  
локаторы таблиц в языке COBOL 310  
локаторы таблиц в языке PL/I 311  
локаторы таблиц в языке ассемблер 308  
локаторы таблиц на языках C или C++ 310  
обзор 268  
ограничения 274, 275  
определение 270  
определитель 268  
особенности параллелизма 276  
параметры 278  
передача параметров на языке C 292  
передача параметров на языке COBOL 299  
передача параметров на языке PL/I 302  
подготовка 312  
подпрограмма 275  
преобразование аргументов 324  
преобразование типов данных 321  
пример 268  
пример определения 272  
разрешение функции 318  
реализатор 268  
реализация 274  
соглашения о параметрах для ассемблера 291  
тестирование 315

пользовательская функция (UDF) (*продолжение*)  
    типы данных хоста 278  
    упрощение разрешения функции 322  
Пользовательские функции  
    описание 267  
пользовательский тип  
    описание 327  
последовательная предварительная выборка  
    время связывания 739  
    описание 738  
последовательное обнаружение 741, 742  
последовательные номера  
    FORTRAN 207  
    PL/I 218  
    программа на языке COBOL 185  
поток  
    завершение  
        функция CLOSE 774  
    создание  
        функция OPEN 774  
права доступа  
    ID авторизации 467  
    таблица SYSIBM.SYSTABAUTH 54  
превышение срока ожидания  
    диагностика в IMS 354  
    код причины X'00C9008E' в SQLCA 353  
    описание 353  
предварительная выборка списка 739  
    См. также list prefetch  
    описание 739  
    пределевые значения 740  
предикат  
    1 этапа 661  
    2 этап  
        влияние на создание 697  
        проверенный 661  
    влияние на путь доступа 658, 692  
    генерация 674  
    индексируемый 660  
    локальный 659  
    модификация 674  
    общие правила 662  
        условие WHERE 26  
    объединение 659  
    описание 658  
    подзапрос 659  
    показатель фильтрации 669  
    свойства 659  
    уточненный 85  
предикат BETWEEN 32  
предикат EXISTS 88  
предикат LIKE 29  
прекомпилятор  
    диагностика 440  
    запуск  
        динамический 471  
        язык JCL для процедур 469  
прекомпилятор (*продолжение*)  
    запуск заданий  
        панели ISPF 478  
    инициализация заданий  
        панели DB2I 486  
        панели ISPF 476  
    максимальный размер на входе 438  
    описание 438  
    описание опций 441  
    параметры  
        CONNECT 413  
        SQL 413  
        доступ DRDA 413  
        Значения по умолчанию 446  
    планирование 342  
    прекомпиляция программ 438  
    программа на входе 439  
    программа на выходе 439  
    связывание на другой системе 441  
    символ выделения 441  
        функции 438  
преобразование  
    в вызове пользовательской функции 324  
прикладная программа  
    приостановка  
        описание 353  
    прикладная программа organization  
        примеры 883  
    прикладная программа phone  
        описание 883  
    прикладная программа project 883  
        описание 883  
    прикладная программа на COBOL  
        переменные в SQL 110  
    прикладная программа на ассемблере  
        ассемблирование 463  
    прикладная программа на языке APL2 108, 438  
    прикладная программа на языке BASIC 108, 438  
    прикладная программа на языке C  
        графические переменные хоста 166  
        кодирование операторов SQL 160, 179  
        константы 176  
        объявление переменной  
            правила 174  
        переменные-индикаторы 178  
        пример прикладной программы 883  
        примеры 913  
        символьные переменные хоста 165  
        соглашения об именовании 163  
        строка переменной длины 178  
        строка фиксированной длины 178  
    прикладная программа на языке C++  
        кодирование операторов SQL 160  
        опция прекомпилятора, значения по  
            умолчанию 447  
        особенности 181

прикладная программа на языке C++ (*продолжение*)  
с классами  
подготовка 475

прикладная программа на языке COBOL  
динамические операторы SQL 577  
имя записи FILLER 199  
кодирование операторов SQL 107, 181  
компиляция 463  
объявление переменной 198  
объявления данных 133  
оператор DECLARE 185  
оператор WHENEVER 186  
описание записи от DCLGEN 138  
опция прекомпилятора DB2, значения по умолчанию 447  
параметры 186  
переменная хоста  
использование дефисов 187  
переменные-индикаторы 201  
переустановка SQL-INIT-FLAG 187  
подготовка 463  
пример программы 899  
с классами  
подготовка 475  
с объектно-ориентированными расширениями  
особенности 204  
символьные переменные хоста  
строки переменной длины 190  
строки фиксированной длины 190  
совместимость типа данных 200  
соглашения об именовании 185

прикладная программа на языке FORTRAN  
байтовый тип данных 208  
включение кода 207  
кодирование операторов SQL 204  
константы, различия в синтаксисе 213  
метки операторов 207  
объявление  
переменных 212  
производных таблиц 207  
таблиц 207  
оператор @PROCESS 208  
оператор SQL INCLUDE 208  
описание SQLCA 204  
параллелизм 208  
переменная хоста 208  
переменные-индикаторы 214  
поля для операторов SQL 207  
последовательные номера 207  
правила присваивания значений, числовых 212  
символьная переменная хоста 208, 209  
соглашения об именовании 207  
строки комментариев 206  
типы данных 210

прикладная программа на языке PL/I  
SQLCA, определение 216

прикладная программа на языке PL/I (*продолжение*)  
SQLDA, определение 216  
графические переменные хоста 222  
кодирование операторов SQL 215  
комментарии 217  
метки операторов 218  
объявление производных таблиц 218  
объявление таблиц 218  
оператор WHENEVER 218  
особенности 219  
переменная хоста  
использование 220  
объявление 221  
числовая 221  
переменная, объявление 227  
переменная-индикатор 229  
последовательные номера 218  
символьные переменные хоста 222  
соглашения об именовании 218  
типы данных 224, 229

прикладное программирование  
особенности разработки  
RRSAF 808

прикладные программы  
вопросы разработки  
аварийные завершения вызова  
синхронизации 528  
выполнение  
CAF (утилита подключения по вызову) 772  
запуск  
CICS 469  
IMS 468  
TSO 465  
TSO CLIST 468  
синхронизация программы в DL/I batch 527

кодирование операторов SQL  
COBOL 181  
FORTRAN 204  
PL/I 216  
ассемблер 145  
ввод данных 68  
выбор строк с использованием указателя 123  
динамические операторы SQL 543, 577  
объявления данных 133  
описание 107  
переменные хоста 110  
обработка ошибок 397  
объектные расширения 251  
особенности разработки  
вызов XRST 528  
вызовы IMS 527  
использование ISPF 475  
контрольная точка 528  
операторы SQL 527  
планирование изменений 344  
прекомпиляция 341  
разработка программ для DL/I batch 526

прикладные программы (*продолжение*)  
    особенности разработки (*продолжение*)  
        связывание 341  
        символическая контрольная точка 527  
        структура 765  
        хранимые процедуры 579  
подготовка  
    ассемблирование 463  
    доступ DRDA 413  
    использование DB2I (DB2 Interactive) 475  
    компиляция 463  
    компоновка 463  
    определение в CICS 450  
    опции компиляции по умолчанию 438  
    опция прекомпилиатора DB2, значения по  
        умолчанию 447  
    панель подготовки программ 475  
    подготовка к запуску 435  
    прекомпиляция 342  
    пример 478  
        связывание 342, 450  
среда тестирования 507  
тестирование 507  
пример прикладной программы  
    LOB 884  
    organization 883  
    phone 883  
    динамические операторы SQL 913  
    доступ DRDA 917  
    доступ по собственному протоколу DB2 925  
    использование 885  
    пользовательская функция 884  
    программы 886  
    проект 883  
    среды 885  
    статические операторы SQL 913  
    строковая 879  
    утилита подключения по вызову 770  
    утилита подключения служб менеджера  
        восстановимых ресурсов 808  
    хранимая процедура 884  
    языки 885  
пример программы  
    DSN8BC3 203  
    DSN8BD3 180  
    DSN8BE3 180  
    DSN8BF3 215  
    DSN8BP3 231  
    DSNTIAD 160  
пример программы DSN8BC3 203  
пример программы DSN8BF3 215  
пример программы DSN8BP3 231  
пример программы DSNTIAR  
    как запустить 889  
    параметры 890  
    подготовка программы 889  
    подпрограмма вызовов DSNTIAR 160  
    указание символа-ограничителя SQL 894  
пример программы DSNTIAUL  
    как запустить 889  
    параметры 890  
    подготовка программы 889  
принятие  
    использование RRSAF 809  
    координация отката 401  
присваивание значений  
    числа 212  
проверка  
    действие на DELETE 73  
проверка синтаксиса SQL 412  
проверка целостности данных  
    оператор CREATE 61  
    оператор INSERT 68  
    оператор UPDATE 72  
прогностическое ограничение  
    в распределенной среде 553  
    обработка в прикладной программе 553  
    с DEFER(PREPARE) 553  
программа BMP (с пакетной обработкой сообщений)  
    контрольные точки 399, 400  
    транзакционно-ориентированные 399  
программа монитора терминала (TMP) 465, 467  
    См. также TMP (terminal monitor program)  
программа на ассемблере  
    графические переменные хоста 152  
    кодирование операторов SQL 145  
    макрокоманды DB2 160  
    переменная хоста  
        объявление 150  
        соглашения об именовании 148  
    переменная-индикатор 158  
    символьная строка переменной длины 151  
    символьная строка фиксированной длины 151  
    символьные переменные хоста 151  
    совместимость типа данных 157  
программа на языке C  
    опция прекомпилиатора, значения по  
        умолчанию 447  
программа на языке COBOL  
    пустые значения 114  
    строковая хоста 115  
программа на языке FORTRAN  
    опция прекомпилиатора, значения по  
        умолчанию 447  
программа примера DSN8BD3 180  
программа примера DSN8BE3 180  
программа примера DSNTIAUL 957

программы IMS одиночного режима 399  
 производительность  
     влияние  
         DEFER(PREPARE) 421  
         NODEFER (PREPARE) 421  
         размера блокировки 363  
         строки прикладной программы 768  
         удаленные запросы 418, 421  
     текущий контроль  
         с помощью EXPLAIN 701  
 производная таблица  
     EXPLAIN 748  
     данные сводки 66  
     использование  
         вставка строк 66  
         выбор строк с использованием указателя 123  
         изменение строк 71  
         удаление строк 73  
     обработка  
         материализация производной таблицы в PLAN\_TABLE 718  
         описание материализации производной таблицы 746  
         слияние производных таблиц 745  
         описание 65  
         содержимое 66  
     создание  
         объявление производной таблицы в языке COBOL 185  
         описание 109  
 просмотр  
     ISPF (interactive system productivity facility) 102  
     данных, обратный 853, 854  
     обратный, с помощью ROWID 855  
 просмотр раздела  
     ограниченный 718  
 простое табличное пространство  
     блокировки 362  
 пространство данных  
     материализация большого объекта 262  
 процедура DSNHASM 469  
 процедура DSNHC 469  
 процедура DSNHCOB 469  
 процедура DSNHCOB2 469  
 процедура DSNHCPP 469  
 процедура DSNHCPP2 469  
 процедура DSNHFOR 469  
 процедура DSNHCB2 469  
 процедура DSNHICOB 469  
 процедура DSNHPLI 469  
 процедура обработчика  
     восстановление при аварийном завершении при использовании CAF 795  
     обработка ситуаций внимания при использовании CAF 795  
 процедура полей  
     изменение последовательности сортировки 50  
 процедура функционального восстановления (FRR) 796  
 процедура, хранимая 579  
     См. также хранимая процедура  
 прямой доступ к строке 714  
 пул RID (идентификаторов записи)  
     использование в предварительной выборке списка 739  
 пустое значение  
     описание 27  
 программы на языке COBOL 114  
 путь доступа  
     влияние на атрибуты блокировки 385  
     выбор  
         Visual Explain 693, 701  
         влияние SQL 693  
         запросы, содержащие переменные хоста 682  
         проблемы 655  
     доступ только через индекс 714  
     индекс уникальности с соответствием значений 728  
     многондексный доступ  
         PLAN\_TABLE 712  
         описание 725  
     низкое отношение кластеризации  
         предлагает просмотр табличного пространства 722  
         с предварительной выборкой списка 740  
 просмотр табличного пространства 722  
 прямой доступ к строке 714

## P

разделитель  
     операторы SQL 109  
 разделитель END-EXEC 109  
 разделитель EXEC SQL 109  
 разрешение функции 318  
     пользовательская функция (UDF) 318  
 распределенное табличное пространство  
     блокировки 362  
 распределенные данные  
     вопросы производительности 421  
     выбор метода доступа 407  
     идентификация сервера во время выполнения 429  
     копирование из удаленных положений 429  
     передача смешанных данных 429  
     переход от доступа по собственному протоколу DB2 к доступу по протоколу DRDA 430  
 планирование  
     программный доступ 405, 429  
     повышение эффективности 418  
     подготовка программы 415

распределенные данные (*продолжение*)  
поддержка согласованности данных 429  
получение таблиц ASCII в DB2 for OS/390 429  
программирование  
    программирование с использованием доступа по протоколу DRDA 408  
    программирование с использованием собственного протокола доступа DB2 408  
производительность для больших объектов 419  
терминология 405  
режим блокировки 364  
реляционная целостность  
    действие на  
        CREATE 60  
        DELETE 73  
        INSERT 68  
        UPDATE 72  
    подзапросы 92  
особенности программирования 860  
реляционное ограничение  
    действие на CREATE 60  
    определение нарушений 860  
ресурс недоступен, условие 837  
родительская таблица 72

## C

сброс блоков управления 789  
сброс управляющих блоков 836  
связанная ссылка  
    корреляционное имя  
        пример 90  
связанные подзапросы 687  
    См. также subquery  
связывание  
    задание правил SQL 460  
    модули DBRM, прекомпилированные на другой подсистеме 441  
    опции проверки 415  
    опции, связанные с доступом по протоколу DRDA 413  
пакеты  
    используемые 342  
    как использовать 343  
    удаленная 451  
параметры 342  
планирование 342, 349  
планы  
    включение пакетов 453  
    используемые 342  
    параметры 453  
    содержащие модули DBRM 453  
планы прикладных программ 450  
после изменений 345  
удаленный пакет  
    требования 451

сегментированное табличное пространство  
    блокировки 362  
    просмотр 722  
секционированный набор данных (PDS) 133  
символ NUL в C 163  
символ выделения  
    SQL 441  
символ–ограничитель SQL  
    указание в DSNTIAD 894  
символы подстановки 29  
символьная строка  
    ширина столбца результатов 98, 101  
символьная строка переменной длины  
    C 178  
    COBOL 190  
    ассемблер 151  
символьная строка фиксированной длины  
    C 178  
    ассемблер 151  
    значение в операторе CREATE TABLE 58  
символьные переменные хоста  
    C 165  
    COBOL 190  
    FORTRAN 209  
    PL/I 222  
    ассемблер 151  
синтаксические диаграммы, как читать 6  
скалярная функция  
    вложенная 44  
    описание 38  
службы ISPLINK SELECT 768  
смешанные данные  
    описание 20  
собрание, пакет  
    идентификация 454  
    оператор SET CURRENT PACKAGESET 454  
совместимость  
    блокировки 366  
    правила 20  
    типы данных 20  
соглашение об использовании регистров для CAF  
    (утилиты подключения по вызову) 779  
соглашение об использовании регистров для RRSAF  
    (средства подключений службы управления восстановимыми ресурсами) 816  
соглашения об именовании  
    C 163  
    COBOL 185  
    FORTRAN 207  
    PL/I 218  
    ассемблер 148  
    создаваемых таблиц 58  
соединение  
    DB2  
        соединение из задач 765  
    функция CAF  
        CLOSE 787, 799

соединение (*продолжение*)  
функция CAF (*продолжение*)  
CONNECT 781, 785, 799  
DISCONNECT 789, 799  
OPEN 785, 799  
TRANSLATE 791, 803  
краткое описание поведения 792, 793  
описание 773  
примеры сценариев 794, 795  
функция RRSAF  
AUTH SIGNON 825  
CREATE THREAD 846  
IDENTIFY 817, 846  
SIGNON 822, 846  
TERMINATE IDENTIFY 836, 846  
TERMINATE THREAD 835, 846  
TRANSLATE 837  
краткое описание поведения 839  
описание 810  
примеры сценариев 841  
создание прикладных программ  
особенности проектирования  
CAF 770  
хранимые процедуры 593  
сообщение  
анализ 519  
ошибки CAF 792  
ошибки RRSAF 839  
получение текста  
C 179  
COBOL 202  
FORTRAN 215  
PL/I 231  
ассемблер 159  
описание 119  
сортировка  
показано в PLAN\_TABLE 743  
программа  
RID (идентификаторы записей) 744  
когда выполняется 744  
удаление повторяющихся значений 743  
состояние  
блокировки 364  
специальный регистр  
CURRENT DEGREE 757  
CURRENT PACKAGESET 71  
CURRENT RULES 460  
CURRENT SERVER 71  
CURRENT SQLID 71  
CURRENT TIME 71  
CURRENT TIMESTAMP 71  
CURRENT TIMEZONE 71  
USER 71  
определение 54  
поведение в пользовательских функциях 304  
поведение в хранимых процедурах 599

специальный регистр CURRENT PACKAGESET  
динамическое переключение плана 462  
идентификация собрания пакетов 454  
специальный регистр CURRENT SERVER  
в прикладной программе 429  
описание 454  
специальный регистр CURRENT SQLID  
описание 71  
сравнение  
правила совместимости 20  
сравнения  
операции с подзапросами 87  
среда тестирования, создание 507  
статистика каталога  
влияние на путь доступа 699  
статические операторы SQL  
описание 543  
пример программы на языке C 913  
статический SQL  
переменные хоста 544  
столбец  
вывод  
список столбцов 55  
выражения 34  
заголовок, создаваемый SPUFI 102  
заданный в CREATE TABLE 57  
значение по умолчанию  
пользовательское 58  
предопределенное системой 58  
значения, отсутствие 27  
имя  
как переменная хоста 137  
оператор UPDATE 71  
метки  
использование 571  
надписи  
DCLGEN 136  
получение  
при помощи SELECT 22  
типы данных 20  
упорядочивание 47  
ширина результатов 98, 101  
столбец PAGE\_RANGE таблицы PLAN\_TABLE 718  
столбец PRIMARY\_ACCESSTYPE таблицы  
PLAN\_TABLE 714  
столбец результата  
именование при помощи условия AS 24  
страница  
блокировки  
описание 361  
строка  
ограничитель  
апостроф 441  
переменной длины  
C 178  
COBOL 190  
ассемблер 151

строка (*продолжение*)  
  произвольной длины  
    PL/I 229  
  фиксированной длины  
    C 178  
    COBOL 190  
    PL/I 229  
  ассемблер 151  
  значение в операторе CREATE TABLE 58

строка символов  
  литералы 108  
  операторы сравнения 28  
  предикат LIKE в условии WHERE 29  
  смешанные данные 20

строки  
  выбор по условию WHERE 26  
  изменение 71  
  изменение текущей 128  
  обновление больших объемов 857

строковые переменные хоста в C 175

структура загружаемого модуля CAF (утилиты подключения по вызову) 773

структура загрузочного модуля RRSAF (средства подключений службы управления восстановимыми ресурсами) 811

структура хоста  
  C 169  
  COBOL 115, 193  
  PL/I 223  
  описание 115

## Т

таблица  
  блокировки 361  
  временная 62  
  вывод списка 54  
  выражение, вложенное  
    обработка 745  
  зависимая  
    изменение 72  
  заполнение  
    вставка строк 66  
    тестовыми данными 510  
  изменение  
    изменение определений 859  
  изменение строк 71  
  копирование из удаленных положений 429  
  объявление 109, 133  
  отбрасывание  
    оператор DROP 64  
  получение  
    с использованием указателя 123  
  родительская  
    изменение 72  
  требования к доступу 109

таблица (*продолжение*)  
  требования при обращении из программы на языке COBOL 185  
  удаление строк 73

таблица DSN\_STATEMNT\_TABLE  
  описания столбцов 749

таблица PLAN\_TABLE  
  описание столбцов 703  
  отчет о внешнем объединении 730

таблица SYSIBM.SYSDUMMY1  
  использование 23

таблица операторов  
  описания столбцов 749

таблица отделов примера  
  описание 864  
  создание 59

таблица переходов  
  триггер 237

таблица примера  
  DSN8610.ACT (работы) 863  
  DSN8610.DEPT (отделы) 864  
  DSN8610.EMP (сотрудники) 866  
  DSN8610.EMP\_PHOTO\_RESUME (фотографии и резюме сотрудников) 870  
  DSN8610.EMPPROJACT (сотрудники работ по проектам) 873  
  DSN8610.PROJ (проекты) 871  
  PROJACT (работы по проектам) 872  
  производные таблицы 875

таблица проектов примера 871

таблица работ по проектам примера 872

таблица работ примера 863

таблица сотрудников примера 866

таблица сотрудников работ по проектам примера 873

таблица фотографий и резюме сотрудников примера 870

таблицы каталога  
  доступ 54

таблицы результатов  
  пример 19

табличное проверочное ограничение  
  описание 61  
  определение нарушений 860  
  особенности программирования 860  
  пример 61

табличное пространство  
  блокировки  
    описание 361  
  для программ примеров 881  
  просмотр  
    определенный утилитой EXPLAIN 702  
    путь доступа 722

табличные выражения, вложенные  
  материализация 746

тип данных  
локатор набора результатов 639  
совместимость  
  C и SQL 171  
  COBOL и SQL 195, 200  
  FORTRAN 213  
  FORTRAN и SQL 212  
  PL/I и SQL 229  
  ассемблер и SQL 156  
  программа на ассемблере 157  
эквивалентный  
  FORTRAN 210  
  PL/I 224  
только для чтения  
  таблицы результатов 126  
точка принятия  
  единица работы IMS 396  
  описание 394  
  снятие блокировки 396  
точка принятия DB2 396  
транзакции  
  IMS  
    использование глобальных транзакций 360  
  транзакционно-ориентированная BMP, контрольные  
    точки в ней 399  
трехчастные имена таблиц  
  использование 408  
  пример 408  
триггер  
  взаимодействие с реляционными связями 244  
  каскадное срабатывание 242  
  кодирование 235  
  обзор 233  
  описание 62, 233  
  переменная перехода 237  
  порядок активации 243  
  пример 233  
  таблица переходов 237  
  элементы 235  
тупиковая ситуация  
  диагностика  
    в CICS 357  
    в IMS 356  
    в TSO 356  
  код причины X'00C90088' в SQLCA 355  
  описание 354  
  пример 354  
  рекомендации, как избежать 358  
  с RELEASE(DEALLOCATE) 359

## У

удаление  
  данные 73  
  каждой строки таблицы 74  
  родительский ключ 73

удаление (*продолжение*)  
  строк из таблицы 73  
  текущие строки 129  
указатель  
  WITH HOLD 130  
    блокировки 381  
    заявки 382  
  действие аварийного завершения на  
    положение 130  
  закрытие  
    оператор CLOSE 129  
  изменение текущей строки 128  
  конец данных 127  
  неоднозначный 380  
  объявление 125  
  описание 123  
  открытие  
    оператор OPEN 126  
  открытое состояние 129  
  получение строки данных 127  
  пример 124  
  сохранение положения 130  
  удаление текущей строки 129  
указатель WITH HOLD 558  
  влияние на блокировки и заявки 381  
уникальный индекс  
  создание с помощью системного таймера 853  
уровень блокировки 361  
уровень изоляции  
  задание в операторе SQL  
  пример 382  
  рекомендации 359  
условие "ресурс недоступен" 791  
условие AS  
  именование столбцов результата 24  
условие FOR FETCH ONLY 424  
условие FOR READ ONLY 424  
  См. также условие FOR FETCH ONLY  
условие FOR UPDATE OF  
  используемое для изменения столбца 125  
  пример 126  
условие FROM  
  объединение таблиц 75  
  оператор SELECT 22  
условие GO TO оператора WHENEVER 118  
условие GROUP BY  
  влияние на условие OPTIMIZE 695  
подвыбор  
  описание 51  
  примеры 51  
условие HAVING для подвыбора  
  выбор групп с критериями 51  
условие NOT FOUND оператора WHENEVER 118  
условие NOT NULL  
  оператор CREATE TABLE  
  использование 58

условие ON  
объединение таблиц 75  
условие OPTIMIZE FOR n ROWS 694  
условие ORDER BY  
    влияние на условие OPTIMIZE 695  
    оператор SELECT 47  
условие SEGSIZE оператора CREATE TABLESPACE  
    рекомендации 722  
условие SET оператора UPDATE 71  
условие SQLWARNING  
    оператор WHENEVER в программе на языке  
        COBOL 118  
условие UNION  
    влияние на условие OPTIMIZE 695  
    оператор SELECT 52  
    удаление повторяющихся значений с помощью  
        сортировки 743  
условие USING DESCRIPTOR  
    оператор EXECUTE 576  
    оператор FETCH 574  
    оператор OPEN 576  
условие WHERE  
    оператор SELECT  
        %, \_ 29  
        объединение таблиц 75  
        объединение таблицы с этой же таблицей 77  
    операция AND 31  
    операция NOT 28  
    операция OR 31  
    описание 26  
    предикат BETWEEN ... AND 32  
    предикат IN (...) 33  
    предикат LIKE 29  
    скобки 31  
    опция NULL 27  
условие WITH HOLD оператора DECLARE  
    CURSOR 130  
утилита ISPF  
    программирование 765  
утилита PRELINK 481  
утилита ограничения ресурсов 552  
    См.также утилиты ограничения ресурсов  
утилита ограничения ресурсов 552  
    обработка прогностического ограничения в  
        прикладной программе 553  
    описание 552  
утилита подключения CICS 849  
    См.также CICS  
утилита подключения по вызову (CAF) 769  
    См.также утилита CAF (call attachment facility)  
утилита подключения служб управления  
    восстановимыми ресурсами (RRSAF) 807  
    См.также RRSAF (утилита подключения служб  
        менеджера восстановимых ресурсов)  
уточненный предикат SOME 87

## Ф

флаги, переустановка 187  
формат  
    SQL во входном наборе данных 99  
    результатов оператора SELECT 101  
функция  
    пользовательская 45  
    скалярная  
        вложенная 44  
        пример 38  
    столбец  
        AVG 37  
        COUNT 37  
        MAX 37  
        MIN 37  
        SUM 37  
        вложенная 44  
        когда выполняется 721  
        описания 37  
    функция AVG 37  
    функция COALESCE 78  
    функция COUNT 37  
    функция MAX 37  
    функция MIN 37  
    функция STDDEV  
        когда выполняется 721  
    функция SUM 37  
    функция VARIANCE  
        когда выполняется 721  
    функция столбца 44  
        См.также функция

## Х

хранение не табличных данных 860  
хранилище, DB2  
    пример прикладной программы 880  
хранимая процедура 317, 654  
    возвращение набора результатов 603  
    возвращение нереляционных данных 604  
    вызов из триггера 240  
    выполнение в качестве авторизованной  
        программы 605  
    запрещенные операторы SQL 599  
    использование 579  
    использование временных таблиц в 604  
    использование переменных хоста 583  
    использование специальных регистров 599  
    методы передачи параметров 609  
    написание 593  
    оператор CALL  
        ограничения на 599  
        описание 607  
    определение для DB2 585  
    определение списков параметров 613

*хранимая процедура (продолжение)*

поддерживаемые языки 593

пример 581

разрешенные операторы 970

связывание 606

тестирование 649

## Ч

числа

присваивание значений 212

числовая

данные

не используются с LIKE 29

числовые

данные

ширина столбца результатов 98, 101

чтение в обход блокировок 375

*См. также UR (чтение непринятого)*

чтение непринятого (изоляция UR) 375

*См. также UR (чтение непринятого)*

## Э

электронные книги 10

электронные публикации 10

эффект водостока 850

## Я

язык Ada 108

язык хоста

встроенные операторы SQL в языке хоста 108

динамические операторы SQL 554

объявления в DB2I (DB2 Interactive) 134



Номер программы: 5645-DB2



Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

SH43-0083-00

