Раздел 5. Процедурные расширения языка SQL

Тема 5.2
Встраиваемый SQL

Вопросы лекции:

1. **Назначение встраиваемого SQL и его разновидности.**
2. Принципы использования статического SQL.
3. Принципы использования динамического SQL.
4. Курсоры и особенности их использования.

При создании базы данных и организации работы с ней возникают три основные проблемы:

- собственно создание базы данных (создание таблиц, индексов, ограничений целостности);
- обеспечение безопасности и разграничения доступа;
- организация пользователям удобного доступа к элементам таблицы (выборка, редактирование, удаление, добавление).

Первая проблема может быть решена посредством создания для СУБД некоторой утилиты, позволяющей в определенный момент осуществлять все необходимые действия по созданию базы данных. Однако это не полностью решает проблему. То же самое можно сказать и о решении второй проблемы.

Такая утилита не позволяет создать или изменить таблицу динамически во время работы прикладной программы.

Необходимы средства, дающие возможность формирования во время работы прикладной программы **запроса как на изменение содержания базы данных, так и ее структуры**, а возможно и управление безопасностью БД.

Встраиваемый (программный) SQL представляется операторами языка SQL, встроенными в прикладные программы, написанные на других языках программирования (в других программных средах).

Это дает возможность работы с базой данных с помощью прикладных программ, написанных на алгоритмических языках, но требует включения дополнительных средств, обеспечивающих **интерфейс** между операторами языка SQL и соответствующим языком программирования.

Компилятор с алгоритмического языка должен иметь возможность **выделения в тексте** прикладной программы последовательность операторов SQL, что позволяет **объединять возможности языка программирования** высокого уровня (переменные, ветвления, циклы) и возможности SQL (запросы на языке, близком к естественному).

Программный SQL позволяет:

- использовать операторы интерактивного SQL в тексте программы на языке программирования высокого уровня;
- наряду с операторами интерактивного SQL использовать новые специальные конструкции, дополняющие SQL и увеличивающие его возможности;
- для передачи параметров в запрос использовать в тексте запроса переменные, объявленные в программе;
- для возврата в программу результатов запроса использовать специальные конструкции, отсутствующие в интерактивном SQL;
- осуществлять компиляцию запросов совместно с программой, обеспечивая впоследствии согласованную работу программы и СУБД;
- заранее (на этапе компиляции) выполнять действия по анализу и оптимизации запросов, экономя время, затрачиваемое на этапе выполнения программы.

Варианты использования программного SQL - статический SQL и динамический SQL.

При **статическом** использовании языка (статический SQL) в текст прикладной программы **включаются конкретные операторы SQL**, и после компиляции исходной программы в выполняемый модуль **жестко включаются** соответствующие этим операторам функции реализации SQL-запросов. Изменения в вызываемых функциях могут здесь определяться только изменениями параметров операторов SQL, инициируемых с помощью переменных языка программирования.

При **динамическом** использовании языка (динамический SQL) соответствующие **SQL-функции** для обращения к базе данных **динамически формируются** в ходе выполнения программы (не на этапе компиляции) и далее осуществляются их передача в СУБД.

Раздел 5. Процедурные расширения языка SQL

Тема 5.2
Встраиваемый SQL

Вопросы лекции:

1. Назначение встраиваемого SQL и его разновидности.
2. **Принципы использования статического SQL.**
3. Принципы использования динамического SQL.
4. Курсоры и особенности их использования.

Существует два основных этапа, связанных с работой статического SQL – **компиляция** программы и **работа (выполнение)** программы.

Схема **компиляции и сборки** программы выглядит следующим образом:

- программа, включающая операторы языка программирования высокого уровня (ЯПВУ) вместе с операторами SQL, подается на вход специального **препроцессора**, который выделяет из нее части, связанные с SQL.

- вместо инструкций встроенного SQL препроцессор подставляет вызовы специальных функций языка программирования для СУБД.

Библиотеки таких функций для связи с языками программирования существуют для всех распространенных серверных СУБД. Стоит особо отметить, что эти библиотеки имеют "закрытый" интерфейс, т.е. разработчики библиотеки могут менять его по своему усмотрению, соответственно обновив препроцессор. Все это говорит о том, что программист не должен вмешиваться в этот процесс.

Схема **компиляции и сборки** программы выглядит следующим образом: (продолжение)

- сами инструкции SQL препроцессор выделяет в отдельный файл.
- программа поступает на вход обычного компилятора языка программирования, после чего получаются объектные модули, которые вместе с библиотеками для работы с СУБД собираются в один исполняемый модуль – приложение.
- наряду с этими операциями происходит работа с файлом, содержащим SQL-инструкции и формируется модуль, который в литературе часто носит название "**модуль запросов к базе данных**" (Database Request Module, DBRM).

Обработку этого модуля осуществляет специальная утилита, которая обычно носит название BIND.

Схема **компиляции и сборки** программы выглядит следующим образом: (продолжение)

Для каждой инструкции SQL утилита, которая обычно носит название BIND, выполняет следующие действия:

- осуществляет синтаксический анализ запроса (проверяет, является ли запрос корректным);
- проверяет, существуют ли в базе данных те объекты, на которые ссылается запрос;
- выбирает, каким образом осуществлять выполнение запроса – формирует план выполнения запроса;

- Все планы выполнения запросов сохраняются в СУБД для последующего использования.

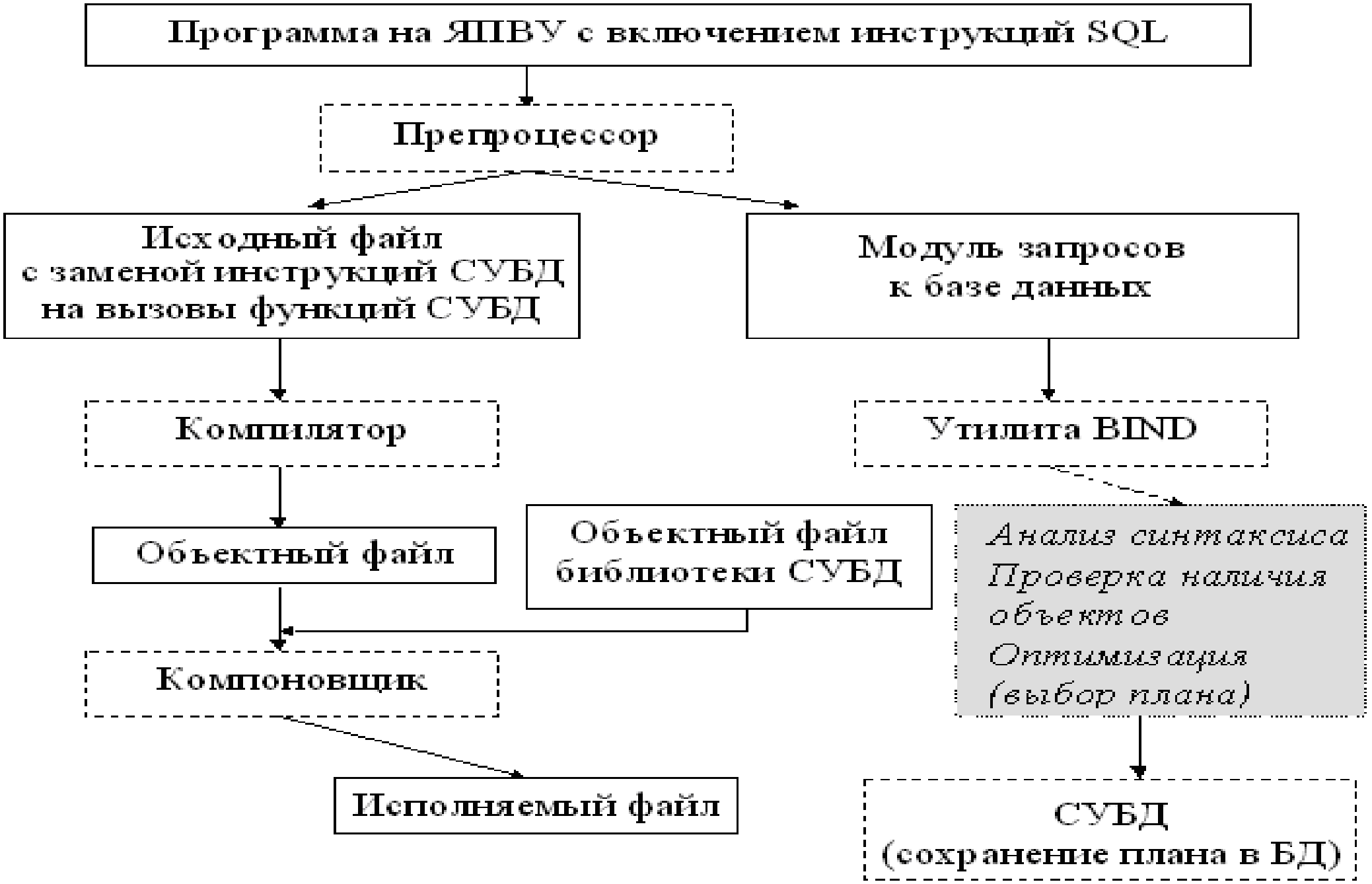


Схема **компиляции** программы с встроенными **инструкциями статического SQL**

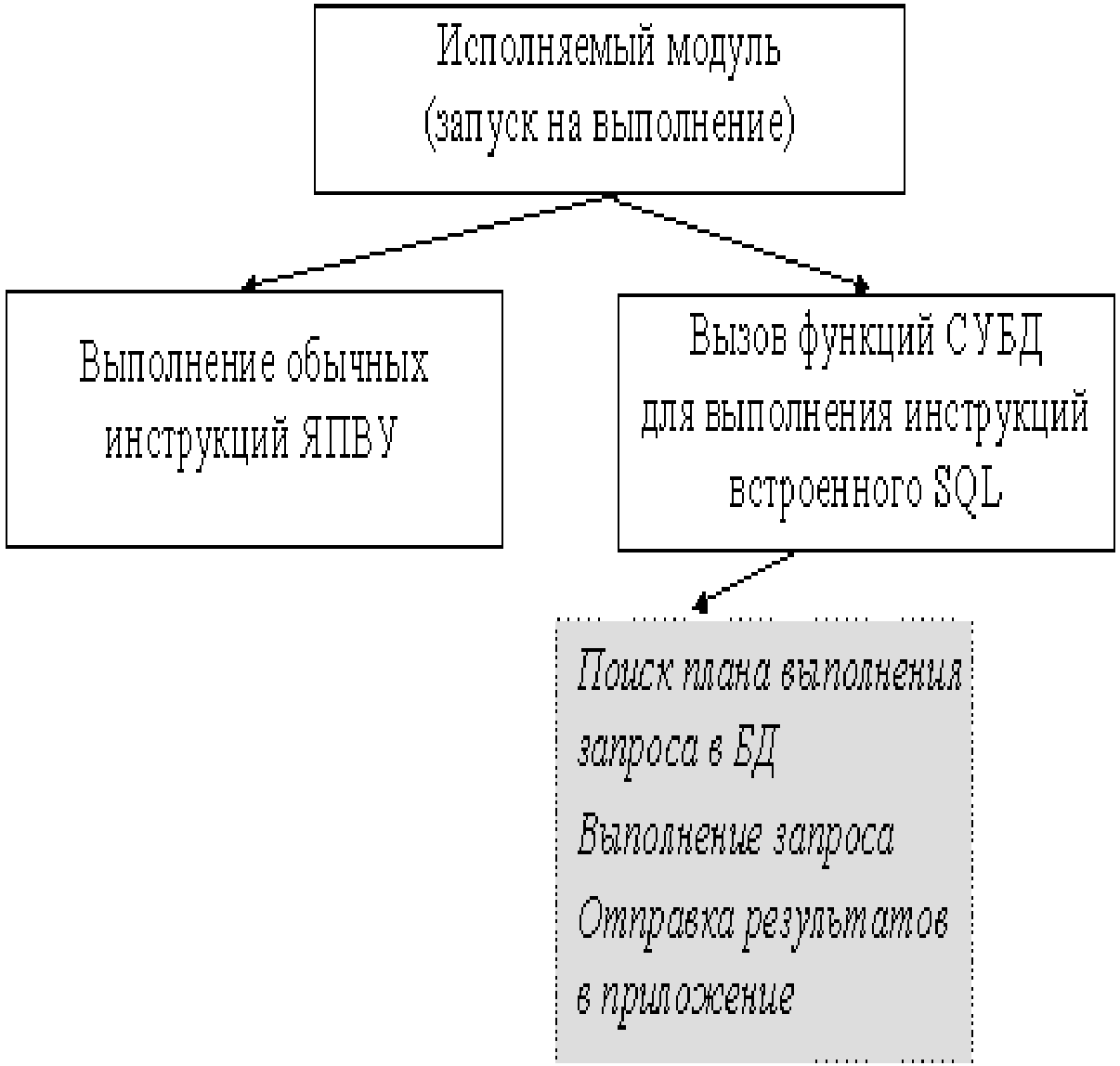


Схема выполнения программы с **встроенными** инструкциями **статического** SQL

Схема выполнения программы выглядит следующим образом:

- Программа запускается на выполнение обычным образом.
- При необходимости выполнить запрос программой осуществляется вызов специальной функции СУБД, которая отыскивает уже сформированный ранее план выполнения запроса.
- СУБД выполняет запрос в соответствии с выбранным планом.
- Результат выполнения запроса поступает в приложение.

Для реализации вышеуказанных схем статический SQL должен содержать дополнительные операторы (по сравнению с интерактивным SQL), позволяющие компилятору:

- выделить в тексте программы SQL-запросы,
- объявлять используемые в этих запросах таблицы,
- объявлять переменные для обработки ошибок, как результатов реализации запросов и т. п.

Основные команды статического SQL приводятся в таблице на следующем слайде.

EXEC SQL	Спецификатор, указывающий, что следующая за ним инструкция является инструкцией встроенного SQL
DECLARE TABLE	Объявляет таблицу, которая потом будет использоваться в инструкциях встроенного SQL
SQLCODE	Переменная для обработки ошибок
SQLSTATE	Переменная для обработки ошибок
GET DIAGNOSTICS	Инструкция для обработки ошибок
WHENEVER SQLERROR SQLWARNING NOT FOUND GOTO CONTINUE	Набор совместно используемых инструкций для упрощения обработки ошибок
BEGIN DECLARE SECTION END DECLARE SECTION	Инструкции для определения области, в которой будут объявлены переменные, впоследствии используемые в запросах SQL
INTO	Используется в операторе SELECT для указания переменной, в которую необходимо поместить результат выполнения запроса
DECLARE CURSOR	**Курсор** – специальный инструмент, предназначенный для обработки результатов запроса, содержащих более одной строки. Работа с курсором похожа на работу с файлами. Данная инструкция служит для создания курсора и связывания его с конкретным запросом
OPEN	Команда, открывающая курсор и побуждающая СУБД начать выполнение запроса. Устанавливает курсор перед первой строкой результата запроса
FETCH	Команда, перемещающая указатель текущей строки (курсор) на следующую строку. В некоторых СУБД и стандарте SQL-92 реализованы разные формы команды FETCH, перемещающие курсор на произвольную строку результатов запроса
CLOSE	Закрывает курсор и прекращает доступ к результатам запроса

Основные команды **статического** программного SQL

Использование описанной выше схемы компиляции/сборки/выполнения программы позволяет:

- использовать SQL совместно с программой на языке программирования высокого уровня;
- **заранее** осуществлять проверку синтаксиса запросов и оптимизацию их выполнения (выбор плана).

Понятно, что проверка синтаксиса выполняется быстро, но выбор плана выполнения – весьма трудоемкая процедура.

Тот факт, что она выполняется один раз на этапе компиляции, позволяет говорить о существенном уменьшении накладных расходов.

Однако статическая разновидность программного SQL имеет некоторые существенные ограничения:

- Переменные в запросах могут использоваться только в тех местах, где в запросах обычно стоят константы. Например, нельзя задавать имя таблицы, из которой производится выборка, а также названия столбцов, как параметр (переменная).

- В связи с этим при использовании статического варианта вложенного (программного) SQL необходимо на этапе написания программы точно знать состав запросов, которые необходимо будет выполнять в прикладной программе.

Во многих случаях эти ограничения являются существенными. Для их устранения была введена новая разновидность программного SQL – **динамический** SQL.

Раздел 5. Процедурные расширения языка SQL

Тема 5.2
Встраиваемый SQL

Вопросы лекции:

1. Назначение встраиваемого SQL и его разновидности.
2. Принципы использования статического SQL.
3. **Принципы использования динамического SQL.**
4. Курсоры и особенности их использования.

Достоинство **динамического** SQL в том, что он позволяет формировать запрос к базе данных **во время работы программы**, реагируя на те или иные произошедшие события.

Такая возможность является жизненно важной для клиент-серверной и трехзвенной архитектур, в которых структура базы данных и деловые правила имеют тенденцию к изменению, что требует определенной гибкости при организации процесса обработки данных.

История возникновения динамического SQL во многом связана с компанией **IBM**, внедрившей этот мощный инструмент в свою СУБД **DB2**.

Стандарты SQL, в частности SQL-1, не поддерживали динамического SQL. Лишь в 1992 году в стандарт SQL-2 были включены спецификации динамического SQL.

Основной концепцией динамического SQL является следующее утверждение:

- встроенная инструкция SQL не записывается в исходный текст программы,
- вместо этого программа формирует текст инструкции во время выполнения («на лету») в одной из своих областей данных,
- а затем передает сформированную инструкцию в СУБД для динамического выполнения.

Очевидно, что двухэтапную схему применения статического SQL нельзя реализовать для динамического SQL, так как на этапе компиляции программы запрос неизвестен.

Поэтому **проверку, разборку и оптимизацию** запросов здесь приходится выполнять непосредственно во время работы программы.

Таким образом, если эти операции в статическом SQL выполнялись во время компиляции один раз, то в динамическом SQL они будут выполняться столько раз для одного запроса, сколько раз он будет сформирован в процессе работы прикладной программы.

Это определяет **существенный недостаток динамического SQL**– низкую производительность по сравнению со статическим.

Учитывая относительно низкую производительность динамического SQL, представляется правильным, там, где только возможно, **рекомендовать использование статической разновидности** SQL, применяя аппарат динамического SQL где это действительно необходимо.

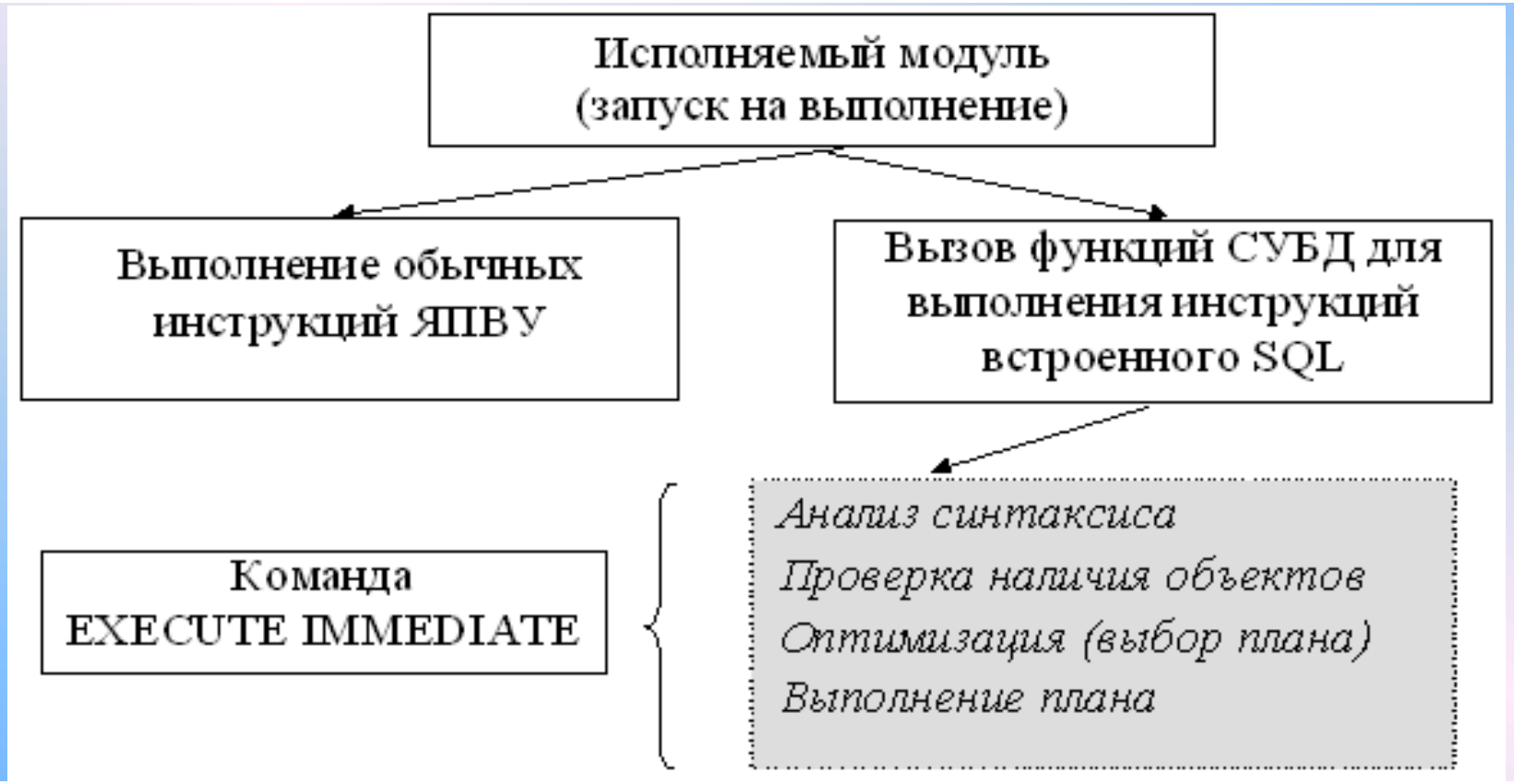
Динамический SQL также должен содержать дополнительные операторы (по сравнению с интерактивным и статическим SQL).

EXECUTE IMMEDIATE	Немедленное выполнение инструкции
PREPARE	Подготовка инструкции к выполнению
EXECUTE	Выполнение подготовленной ранее инструкции
DESCRIBE	Специальная команда, участвующая при возврате результата выполнения инструкций динамического SQL
DECLARE CURSOR	Разновидность инструкции DECLARE CURSOR, применявшейся также в рамках статического SQL, содержащая вместо запроса его имя (связанное с запросом при помощи инструкции PREPARE)
OPEN FETCH CLOSE	Разновидности инструкций для работы с курсором в динамическом SQL

Основные инструкции **динамического** программного SQL

Схема функционирования динамического SQL предусматривает одноэтапное и двухэтапное выполнение инструкций.

Одноэтапное выполнение инструкций осуществляется командой EXECUTE IMMEDIATE.



Порядок выполнения программы со встроенными инструкциями динамического SQL с применением одноэтапной схемы

Одноэтапная схема выполнения инструкции подразумевает:

- динамическое формирование команды SQL в строковом виде во время работы программы;
- передачу строкового вида инструкции в СУБД при помощи команды EXECUTE IMMEDIATE;
- выполнение инструкции системой управления БД, включающее синтаксический анализ, проверку параметров, оптимизацию (выбор плана) и выполнение этого плана.

Основные проблемы одноэтапной схемы заключаются в том, что

- она не позволяет выполнять инструкции SELECT (ибо нет средств для возврата в приложение результатов запроса) и
- приводит к нерациональному расходованию вычислительных ресурсов (т.к. при повторном выполнении той же инструкции вновь будет затрачено время на все те же действия по ее интерпретации и выполнению).

Двухэтапное выполнение инструкций основано на следующем соображении: скорее всего, команда динамического SQL в таком виде, как она поступает на выполнение, будет выполняться неоднократно.

При этом могут меняться какие-то конкретные детали.

А это значит, что инструкцию можно **параметризовать**.

Использование параметризованных инструкций позволяет сделать схему выполнения двухэтапной, разделив процесс на

- "**подготовку** инструкции" и
- "**выполнение** инструкции".

Схема выполнения программы со встроенными инструкциями **динамического** SQL с применением **двухэтапной** схемы

На этапе подготовки через СУБД можно осуществить синтаксический анализ инструкции, интерпретировать ее и подготовиться к выполнению, выбрав план выполнения.

На этапе выполнения СУБД подставляет в полученную на подготовительном этапе SQL-инструкцию значения параметров (полученные из программы) и использует сформированный ранее план выполнения для достижения результата.

При этом реализуется идея однократного выполнения тех действий, которые можно выполнить один раз (подготовка запроса и выбор плана его выполнения).

Далее, подготовленная один раз инструкция **может быть выполнена десятки раз** с разными параметрами.

Недостатки использования программного SQL

Программный SQL отличается от обычной, **интерактивной** формы наличием некоторых специальных инструкций, а также **механизмом трансляции и выполнения** запросов.

Таким образом, для применения программного SQL в тексте своих программ программистам необходимо ознакомиться с некоторым специфическим набором инструкций и методикой их использования.

Стоит заметить, что в разных СУБД эти наборы инструкций, вообще говоря, могут несколько отличаться друг от друга.

В результате возникает некоторая проблема, связанная с **непереносимостью** программы, содержащей инструкции программного SQL.

Раздел 5. Процедурные расширения языка SQL

Тема 5.2
Встраиваемый SQL

Вопросы лекции:

1. Назначение встраиваемого SQL и его разновидности.
2. Принципы использования статического SQL.
3. Принципы использования динамического SQL.
4. **Курсоры и особенности их использования.**

При использовании результатов запросов в клиентских приложениях существует проблема передачи данных в переменные языка программирования для их дальнейшего использования и обработки. Программный SQL позволяет для этих целей использовать специальный объект, называемый **курсором**.

Курсор базы данных позволяет выбрать из базы данных группу строк, перемещаться от строки к строке и проводить анализ строки, на которой в текущий момент установлен курсор.

Другим вариантом использования курсора является сохранение результатов выполнения запроса для последующего использования.

По сути курсоры представляют область памяти, содержащую **множество строк результата запроса SQL** (результирующего набора).

Курсоры позволяют усовершенствовать обработку результатов:

- позиционируясь на отдельные строки результирующего набора;
- получая одну или несколько строк от текущей позиции в результирующем наборе;
- поддерживая изменение данных в строках в текущей позиции результирующего набора;
- поддерживая разные уровни видимости изменений, сделанных другими пользователями для данных, представленных в результирующем наборе;
- предоставляя инструкциям SQL в сценариях, хранимых процедурах и триггерах доступ к данным результирующего набора.

Результирующее множество строк курсора формируется на основании выполнения предложения SELECT.

Если приложение или процедура **требуют повторного использования набора строк**, лучше всего создать один раз курсор и многократно его использовать вместо многократного выполнения предложения SELECT по отношению к базе данных.

Для использования курсора следует выполнить **следующие шаги**.

1. Определить курсор.
2. Открыть курсор.
3. Обработать строки курсора по одной за раз.
4. Закрыть курсор.
5. Уничтожить курсор.

Определение курсора

Курсор определяется с помощью предложения DECLARE CURSOR, которое **имеет следующий синтаксис**:

DECLARE имя_курсора [чувствительность]
[прокручиваемость]
CURSOR
[сохраняемость] [возвращаемость]
FOR спецификация_курсора

Чувствительность курсора определяет, **являются ли видимыми через курсор изменения данных**, произведенные в той транзакции SQL, в которой курсор был открыт, до того момента, когда курсор будет закрыт.

Параметр принимает одно из следующих значений:
SENSITIVE | INSENSITIVE | ASENSITIVE

Значение SENSITIVE указывает, что такие изменения являются видимыми,

значение INSENSITIVE - что эти изменения оказываются невидимыми.

Наконец, значение ASENSITIVE указывает, что видимость изменений определяется конкретной реализацией.

Прокручиваемость курсора свидетельствует о возможности перемещения курсора вверх и вниз по таблице, специфицирующей курсор.

Его синтаксис весьма прост: **[NO] SCROLL**

Значение SCROLL свидетельствует, что прокручиваемость допускается, значение N0 SCROLL - нет.

Сохраняемость курсора определяет **свойство курсора оставаться открытым после завершения транзакции**, в которой курсор был создан.

Параметр принимает одно из следующих **двух значений**:

{WITH | WITHOUT} HOLD

Значение WITH HOLD указывает, что курсор сохраняемый, то есть **не закрывается** (если находится в открытом состоянии) **при завершении транзакции предложением COMMIT** и закрывается, если транзакция завершается предложением ROLLBACK.

Значение WITHOUT HOLD свидетельствует, что курсор несохраняемый, то есть закрывается по завершении транзакции, в которой он был создан.

Возвращаемостъ — это способность курсора возвращать таблицу, которая его специфицирует.

Параметр принимает одно из следующих двух значений:

{WITH | WITHOUT} RETURN

Значение **WITH RETURN** указывает, что если к моменту выхода из процедуры курсор был открытым, **он возвращает во внешнюю программу таблицу**, которая его специфицирует.

Спецификация курсора имеет следующий синтаксис:

запрос
[ORDER BY список_упорядочения]
[FOR {READ ONLY | UPDATE [OF список_столбцов]}]

Основу спецификации курсора составляет запрос, который определяет таблицу курсора.

Упорядоченность строк таблицы **указывается фразой ORDER BY**, которая совпадает по своим возможностям с аналогичной фразой в предложении SELECT.

Работа с курсором

Курсор может находиться **в открытом или закрытом состоянии**.

После объявления курсор оказывается **в закрытом** состоянии.

Чтобы можно было работать с курсором, его следует **открыть командой OPEN**:

OPEN имя_курсора

В результате открытия курсора создается таблица, которая его специфицирует, устанавливается упорядоченность строк и **указатель курсора** располагается перед первой строкой этой таблицы.

Работа с курсором

Чтобы **переместить курсор на необходимую строку таблицы** и запомнить значения столбцов найденной строки, используется **предложение FETCH** со следующим синтаксисом:

FETCH [[ориентация]
FROM] имя_курсора
INTO целевой_список

Ориентация определяет способ получения необходимой строки и принимает одно из следующих значений:

NEXT | PRIOR | FIRST | LAST | {ABSOLUTE | RELATIVE} значение

Работа с курсором

Эти значения указывают, какая строка будет выбрана:
- NEXT - следующая;
- PRIOR — предыдущая;
- FIRST — первая;
- LAST — последняя;
- ABSOLUTE значение — с указанным номером;
- RELATIVE значение — отстоящая на указанное значение (оно может быть положительным и отрицательным).

Целевой список в простейшем случае — это список переменных, в которых запоминаются столбцы найденной строки таблицы.

Запомненные в переменных значения используются далее в процедуре или прикладной программе.

Работа с курсором

Если курсор является обновляемым, можно изменять и/или удалять строки целевой (исходной) таблицы курсора.

Для этого используются разновидности предложений UPDATE и DELETE, которые получили название **позиционного обновления** и **позиционного удаления**.

Эти разновидности команд имеют следующий синтаксис:
UPDATE имя_таблицы [[AS] псевдоним]
SET список_обновления
WHERE CURRENT OF имя_курсора

DELETE FROM имя_таблицы [[AS] псевдоним]
WHERE CURRENT OF имя_курсора
При обновлении **изменяется та строка указанной таблицы, которая является текущей в указанном курсоре**. То же самое имеет место и при удалении.

Работа с курсором

В **закрытое состояние** курсор возвращается **командой CLOSE**:

CLOSE имя_курсора

или **предложением ROLLBACK** той транзакции, которая его содержит.

Открытый курсор, который не был определен как сохраняемый, закрывается также
предложением COMMIT.

В Oracle имеются некоторые специфические особенности работы с курсорами.