1 Файл, файловая система. Особенности организации устройств внешней памяти на магнитных дисках. Структуры файлов на дисках. Классификация файловых систем. Основные подходы к защите файловых систем.

Файл — именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. **Файловая система** (ФС) — программная система, управляющая файлами и архивом файлов, хранящимся во внешней памяти.

Особенности организации устройств внешней памяти на магнитных дисках:

- **Жесткие магнитные диски** обеспечивают быстрый произвольный доступ к данным и большую емкость хранения за счет нескольких магнитных поверхностей.
- **Структура диска**:
  - Диск состоит из пакета магнитных пластин с радиально двигающимися головками.
  - Каждое положение головок соответствует цилиндру, содержащему дорожки на всех поверхностях.
- **Доступ к данным** включает:
  - *Подвод головок* ($t_{\text{пг}}$): время перемещения головок к нужному цилиндру (самое длительное).
  - *Поиск блока на дорожке* ($t_{\text{пб}}$): ожидание нужного блока при вращении диска.
  - *Обмен данными* ($t_{\text{об}}$): чтение или запись блока (самое короткое время).

Структуры файлов на дисках:

- Файлы состоят из последовательностей блоков, которые отображаются на физические блоки диска.
- Размер логических блоков обычно равен или кратен размеру физического блока диска.
- Два подхода к представлению файлов:
  - **Последовательность записей**: файлы как набор записей фиксированной или переменной длины.
  - **Последовательность байтов**: файлы как непрерывная последовательность байтов (как в UNIX).

Классификация файловых систем:

- По логической структуре и именованию файлов:

- **Изолированные ФС**: каждый архив файлов расположен на отдельном логическом диске; полные имена файлов начинаются с имени устройства.
- **Централизованные ФС**: единое дерево каталогов, физически распределенное по всем устройствам; имена файлов начинаются с корневого каталога.
- **Гибридные подходы** (например, UNIX): комбинация изолированных и централизованных систем через механизм монтирования.

Основные подходы к защите файловых систем:

- **Мандатный способ защиты**: каждому пользователю назначаются права на каждый файл (сложно и ресурсоемко).
- **Дискреционный способ защиты** (как в UNIX):
  - Пользователи имеют идентификаторы пользователя и группы.
  - Для каждого файла определяются права доступа для владельца, группы и остальных (чтение, запись, выполнение).
  - Обеспечивает компактность и быструю проверку прав доступа.

2 Возможности и области применения файловых систем. Сравнение ФС и СУБД по обеспечению требований со стороны информационных систем: согласованность данных, языки запросов, восстановление согласованного состояния после сбоев, реальный режим мультидоступа.

Возможности и области применения файловых систем:

- Хранение слабо структурированной информации, где структура данных определяется прикладными программами (текстовые редакторы, компиляторы).
- Предоставление базовых функций: распределение памяти, именование файлов, авторизация доступа, поддержка многопользовательского режима.
- Подходят для приложений с простыми структурами данных и последовательным доступом.

Сравнение ФС и СУБД по требованиям информационных систем:

- **Согласованность данных**:
  - **ФС**: не обеспечивают автоматическую согласованность данных между файлами; ответственность лежит на приложениях.
  - **СУБД**: поддерживают целостность данных через механизмы ограничений и транзакций, обеспечивая согласованное состояние базы данных.

- **Языки запросов**:
  - **ФС**: отсутствуют высокоуровневые языки запросов; доступ к данным осуществляется через низкоуровневые операции ввода-вывода.
  - **СУБД**: предоставляют декларативные языки запросов (например, SQL), позволяющие эффективно и удобно манипулировать данными.
- **Восстановление после сбоев**:
  - **ФС**: ограниченная поддержка; приложения должны самостоятельно обеспечивать восстановление, что может привести к рассогласованности данных.
  - **СУБД**: имеют встроенные механизмы журнализации и восстановления, позволяющие вернуть базу данных в согласованное состояние после сбоев.
- **Реальный режим мультидоступа**:
  - **ФС**: базовая синхронизация доступа через блокировки файлов; при записи файлы могут быть недоступны другим пользователям.
  - **СУБД**: реализуют тонкую синхронизацию на уровне записей или даже полей, позволяя множеству пользователей одновременно работать с данными без конфликтов.

3 СУБД. Основные функции СУБД. Типовая организация современной СУБД.

Основные функции СУБД:

- **Управление согласованными данными во внешней памяти**:
  - Поддержка структур хранения данных, метаданных и служебной информации (индексов).
  - Обеспечение логической целостности данных.
- **Управление буферами оперативной памяти**:
  - Кэширование данных для повышения скорости доступа.
  - Собственные алгоритмы замены буферов для эффективной работы.
- **Управление транзакциями**:
  - Поддержка свойств ACID (атомарность, согласованность, изоляция, долговечность).
  - Обеспечение сериализации транзакций для корректной работы в многопользовательском режиме.
- **Журнализация и восстановление**:
  - Ведение журнала изменений для возможности отката транзакций и восстановления после сбоев.

  - Использование протокола WAL (Write Ahead Log) для надежности данных.

- **Поддержка языков баз данных**:
  - Предоставление языка определения данных (DDL) и языка манипулирования данными (DML).
  - Стандартный язык SQL для работы с реляционными базами данных.
  - Возможность определения ограничений целостности и авторизации доступа на языковом уровне.

Типовая организация современной СУБД:

- **Ядро СУБД (Data Base Engine)**:
  - **Менеджер данных**: управление хранением и доступом к данным.
  - **Менеджер буферов**: работа с кэшированием данных в оперативной памяти.
  - **Менеджер транзакций**: обработка транзакций и обеспечение их свойств.
  - **Менеджер журнала**: ведение журнала изменений и поддержка восстановления.
- **Компилятор языка БД** (обычно SQL):
  - Компиляция и оптимизация запросов.
  - Преобразование декларативных запросов в исполняемый код или внутреннее представление.
- **Подсистема поддержки времени выполнения**:
  - Интерпретация или выполнение скомпилированных запросов.
  - Взаимодействие с ядром СУБД для доступа к данным.
- **Утилиты и инструменты**:
  - Загрузка и выгрузка данных.
  - Сбор статистики для оптимизации запросов.
  - Проверка целостности и другие административные функции.

4 Классификация СУБД. Файл-серверные, клиент-серверные и встраиваемые СУБД.

Файл-серверные СУБД:

- **Принцип работы**: база данных хранится на файловом сервере; СУБД запускается на каждом клиенте.
- **Особенности**:

  - Клиенты работают с файлами напрямую через сеть.
  - Эффективны только в локальных сетях с малым числом клиентов.

- **Недостатки**:
  - Высокая нагрузка на сеть.
  - Ограниченные возможности по синхронизации и управлению транзакциями.

Клиент-серверные СУБД:

- **Принцип работы**: СУБД работает на выделенном сервере; клиенты отправляют запросы серверу.
- **Особенности**:
  - Сервер обрабатывает запросы и управляет данными.
  - Клиенты получают только необходимые результаты, снижая трафик.
- **Преимущества**:
  - Масштабируемость и эффективная работа в многопользовательском режиме.
  - Централизованное управление безопасностью и данными.
- **Варианты**:
  - **Трехзвенная архитектура**: с сервером приложений между клиентом и сервером БД для дополнительной логики и безопасности.

Встраиваемые СУБД:

- **Принцип работы**: СУБД встраивается в приложение и работает в том же процессе.
- **Особенности**:
  - Обычно представляют собой библиотеку для работы с данными.
  - Подходят для мобильных и встроенных систем.
- **Недостатки**:
  - Ограниченные возможности по безопасности и многопользовательскому доступу.
  - Прямая зависимость от приложения, что может влиять на надежность.

5 Классификация СУБД. СУБД, хранящие данные во внешней памяти, и СУБД, сохраняющие данные в основной памяти (in-memory).

Классификация СУБД по месту хранения данных:
СУБД, хранящие данные во внешней памяти:

- **Особенности**:
  - Основной объем данных хранится на внешних носителях (жесткие диски).
  - Используют буферизацию для работы с данными в оперативной памяти.
- **Преимущества**:
  - Большой объем хранения.
  - Долговременное сохранение данных.
- **Недостатки**:
  - Более медленный доступ к данным по сравнению с оперативной памятью.

In-memory СУБД (СУБД в оперативной памяти):

- **Особенности**:
  - Все данные хранятся в оперативной памяти.
  - Внешняя память используется для журнализации и обеспечения долговечности транзакций.
- **Преимущества**:
  - Очень высокая скорость доступа к данным (до 4 порядков быстрее).
  - Подходят для приложений, требующих высокой производительности.
- **Подходы к организации**:
  - Без использования внешней памяти: надежность за счет кластерной репликации.
  - С журналом во внешней памяти: данные в ОП, журнал изменений на диске.
  - Данные в ОП и на диске: чтение из ОП, запись в ОП и на диск.
- **Недостатки**:
  - Ограничения по объему данных (размер ОП).
  - Необходимость обеспечения сохранности данных при сбоях питания.

6 Классификация СУБД. СУБД, хранящие данные во внешней памяти, и СУБД, сохраняющие данные в основной памяти (in-memory).

Классификация СУБД по месту хранения данных:
СУБД, хранящие данные во внешней памяти:

- **Особенности**:
  - Основной объем данных хранится на внешних носителях (жесткие диски).
  - Используют буферизацию для работы с данными в оперативной памяти.
- **Преимущества**:
  - Большой объем хранения.
  - Долговременное сохранение данных.
- **Недостатки**:
  - Более медленный доступ к данным по сравнению с оперативной памятью.

In-memory СУБД (СУБД в оперативной памяти):

- **Особенности**:
  - Все данные хранятся в оперативной памяти.
  - Внешняя память используется для журнализации и обеспечения долговечности транзакций.
- **Преимущества**:
  - Очень высокая скорость доступа к данным (до 4 порядков быстрее).
  - Подходят для приложений, требующих высокой производительности.
- **Подходы к организации**:
  - Без использования внешней памяти: надежность за счет кластерной репликации.
  - С журналом во внешней памяти: данные в ОП, журнал изменений на диске.
  - Данные в ОП и на диске: чтение из ОП, запись в ОП и на диск.
- **Недостатки**:
  - Ограничения по объему данных (размер ОП).
  - Необходимость обеспечения сохранности данных при сбоях питания.

7 Классификация СУБД. Однопроцессорные, параллельные с общей памятью, параллельные с общими дисками и параллельными без использования общих ресурсов СУБД.

Классификация СУБД по типу параллелизма:
Однопроцессорные СУБД:

- **Особенности**:
  - Работают на одном процессоре или ядре.
  - Не используют аппаратный параллелизм.
- **Ограничения**:
  - Низкая масштабируемость.
  - Ограниченная производительность для больших нагрузок.

Параллельные СУБД с общей памятью (Shared-Everything):

- **Особенности**:
  - Несколько процессоров или ядер имеют доступ к общей памяти и дискам.
  - Примеры: Oracle, DB2.
- **Преимущества**:
  - Высокая производительность за счет быстрого обмена данными.
- **Недостатки**:
  - Сложность синхронизации.
  - Ограниченная масштабируемость при добавлении ресурсов.

Параллельные СУБД с общими дисками (Shared-Disks):

- **Особенности**:
  - Узлы имеют собственную оперативную память и процессоры, но разделяют общую дисковую систему.
  - Примеры: Oracle Real Application Cluster.
- **Преимущества**:
  - Повышенная отказоустойчивость.
- **Недостатки**:
  - Необходимость синхронизации доступа к дискам.
  - Возможные узкие места при интенсивных обращениях к диску.

Параллельные СУБД без общих ресурсов (Shared-Nothing):

- **Особенности**:
  - Каждый узел имеет свои собственные ресурсы: процессоры, память, диски.
  - Узлы соединены сетью и обмениваются сообщениями.
  - Примеры: Greenplum, Vertica.
- **Преимущества**:
  - Лучшая масштабируемость при добавлении новых узлов.
  - Локализация сбоев.
- **Недостатки**:
  - Сложность распределения данных и запросов между узлами.
  - Необходимость эффективной сети для обмена данными.

8 Ранние дореляционные подходы к организации баз данных.

Ранние (дореляционные) подходы к организации баз данных:
Навигационные СУБД (1960-е годы):

- **Иерархические модели данных**:
  - Данные организованы в виде дерева.
  - Каждая запись имеет одну родительскую запись.
  - Пример: СУБД IMS (IBM).
- **Сетевые модели данных**:
  - Данные организованы в виде графа.
  - Записи могут иметь множественные связи "многие ко многим".
  - Пример: СУБД IDMS (Computer Associates).

Особенности ранних СУБД:

- **Отсутствие декларативных языков запросов**:
  - Доступ к данным осуществлялся через навигационные команды.
  - Программисты должны были явно указывать пути к данным.
- **Ограниченная поддержка целостности данных**:
  - Проверки и ограничения реализовывались на уровне приложений.
- **Сложность разработки и сопровождения приложений**:
  - Изменения в структуре данных требовали значительных изменений в коде приложений.

Переход к реляционному подходу:

- В конце 1960-х Эдгар Кодд предложил реляционную модель данных.
- Появление декларативных языков (SQL) и новых возможностей для работы с данными.

9 Базовые понятия реляционной модели данных. Ключи. Неопределенные значения. Ссылочная целостность в реляционной модели и способы ее поддержания.

Реляционная модель данных основывается на использовании нормализованных n-арных отношений, где данные представлены в виде кортежей (строк), а атрибуты (столбцы) имеют имена и домены (типы данных).

Ключи:

- **Первичный ключ**: минимальный набор атрибутов, который уникально идентифицирует каждый кортеж в отношении.
- **Возможные ключи**: другие минимальные наборы атрибутов, обладающие свойством уникальности.
- **Составной ключ**: ключ, состоящий из нескольких атрибутов.

Неопределенные значения (NULL):

- Используются, когда значение атрибута неизвестно или неприменимо.
- NULL не принадлежит ни одному типу данных и имеет особые правила при логических и арифметических операциях.

Фундаментальные свойства отношений:

- **Отсутствие кортежей-дубликатов**: тело отношения — это множество уникальных кортежей.
- **Отсутствие упорядоченности кортежей**: кортежи не имеют определенного порядка.
- **Отсутствие упорядоченности атрибутов**: атрибуты не упорядочены, доступ к ним осуществляется по именам.
- **Атомарность значений атрибутов**: каждое значение является неделимым (первая нормальная форма).

Ссылочная целостность в реляционной модели:
Требования целостности:

- **Целостность сущности**: каждый кортеж должен быть уникально идентифицируемым первичным ключом без NULL значений.

- **Ссылочная целостность**: значения внешнего ключа в одном отношении должны соответствовать существующим значениям первичного ключа в другом отношении или быть NULL.

Способы поддержания ссылочной целостности:

- **Ограничить (restrict)**: запретить операции, нарушающие целостность.
- **Установить в NULL (set NULL)**: при удалении связанной записи установить внешние ключи в NULL.
- **Каскадное удаление (cascade)**: автоматически удалить связанные записи.

10 Реляционная алгебра Кодда. Перечислить все операции. Приоритет операций.

Основная идея реляционной алгебры состоит в том, что поскольку отношения являются множествами, средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями, специфичными для реляционных баз данных.

Теоретико-множественные операции:

- **Объединение (UNION)**: включает все кортежи, присутствующие хотя бы в одном из отношений.
- **Пересечение (INTERSECT)**: включает кортежи, присутствующие в обоих отношениях.
- **Разность (MINUS)**: включает кортежи, присутствующие в первом отношении и отсутствующие во втором.
- **Декартово произведение (TIMES)**: создает новые кортежи путем конкатенации каждого кортежа первого отношения с каждым кортежем второго.

Специальные реляционные операции:

- **Ограничение (WHERE)**: выбирает кортежи, удовлетворяющие заданному условию.
- **Проекция (PROJECT)**: создает новое отношение, состоящее из указанных атрибутов исходного отношения, устраняя возможные дубликаты.
- **Соединение (JOIN)**: объединяет кортежи двух отношений на основе заданного условия, часто по совпадающим значениям атрибутов.
- **Деление (DIVIDE BY)**: находит значения атрибутов, связанные со всеми значениями другого отношения.

Дополнительные операции:

- **Переименование атрибутов (RENAME)**
- **Присваивание (:=)**

Приоритет операций (от высшего к низшему):

- RENAME
- WHERE, PROJECT
- TIMES, JOIN, INTERSECT, DIVIDE BY
- UNION, MINUS

11 Реляционная алгебра Кодда. Теоретико-множественные операции. Совместимость отношений по объединению и по расширенному декартовому произведению.

Реляционная модель данных основывается на использовании нормализованных n-арных отношений, где данные представлены в виде кортежей (строк), а атрибуты (столбцы) имеют имена и домены (типы данных).

Теоретико-множественные операции:

- **Объединение (UNION)**: включает все кортежи, присутствующие хотя бы в одном из отношений.
- **Пересечение (INTERSECT)**: включает кортежи, присутствующие в обоих отношениях.
- **Разность (MINUS)**: включает кортежи, присутствующие в первом отношении и отсутствующие во втором.
- **Декартово произведение (TIMES)**: создает новые кортежи путем конкатенации каждого кортежа первого отношения с каждым кортежем второго.

Совместимость отношений:

- **По объединению**: отношения совместимы, если имеют одинаковые схемы — одинаковые имена атрибутов и соответствующие домены.
- **По расширенному декартовому произведению**: отношения должны не иметь общих имен атрибутов; при наличии общих имен используется операция переименования (RENAME) для устранения конфликтов.

12 Реляционная алгебра Кодда. Специальные реляционные операции.

Специальные реляционные операции:

- **Ограничение (WHERE)**: выбирает кортежи, удовлетворяющие заданному условию.
- **Проекция (PROJECT)**: создает новое отношение, состоящее из указанных атрибутов исходного отношения, устраняя возможные дубликаты.
- **Соединение (JOIN)**: объединяет кортежи двух отношений на основе заданного условия, часто по совпадающим значениям атрибутов.
- **Деление (DIVIDE BY)**: находит значения атрибутов, связанные со всеми значениями другого отношения.

13 Реляционная алгебра A: базовые операции подробно с примерами

Алгебра A предложена Крисом Дейтом и Хью Дарвеном в конце 90-х годов. Ее базисом являются операции реляционного отрицания (дополнения), реляционной конъюнкции (и/или дизъюнкции) и удаления атрибута (проекции).

Реляционное дополнение (<NOT>):

Описание: создает отношение, содержащее все кортежи, соответствующие заголовку исходного отношения, но не входящие в его тело.

Пример: если отношение содержит проекты с номерами 1 и 2, то <NOT> этого отношения будет содержать проекты, которых нет в исходном отношении, но которые возможны в домене (например, проект 3).

Удаление атрибута (<REMOVE>):

Описание: удаляет указанный атрибут из отношения.

Пример: удаление атрибута ПРО_N из отношения СЛУЖАЩИЕ оставит отношение с атрибутами СЛУ_N, СЛУ_ИМЯ, СЛУ_ЗАРП.

Переименование атрибута (<RENAME>):

Описание: переименовывает атрибут в отношении.

Пример: переименование атрибута ПРО_N в СЛУ_ПРО_N в отношении СЛУЖАЩИЕ.

Реляционная конъюнкция (<AND>):

Описание: объединяет отношения по совпадающим атрибутам.

Пример: соединение отношений СЛУЖАЩИЕ и ПРОЕКТЫ по общим атрибутам дает результат, аналогичный естественному соединению.

Реляционная дизъюнкция (<OR>):

Описание: объединяет отношения, включающие кортежи из обоих исходных отношений.

Пример: объединение отношений СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 дает отношение, содержащее всех служащих из обоих проектов.

14 Полнота алгебры A. Определение операций алгебры Кодда через алгебру A.

Алгебра A считается полной, так как позволяет выразить все операции алгебры Кодда:

- **UNION**: соответствует операции `<OR>`.
- **INTERSECT**: соответствует операции `<AND>` при совпадающих схемах.
- **TIMES**: соответствует операции `<AND>` при непересекающихся схемах.
- **PROJECT**: реализуется путем последовательного применения `<REMOVE>` к ненужным атрибутам.
- **RENAME**: соответствует операции `<RENAME>`.
- **MINUS**: выражается как `r1 MINUS r2 = r1 <AND> (<NOT> r2)`.
- **JOIN**: реализуется через комбинацию `<AND>` после переименования атрибутов и применения ограничений.
- **DIVIDE BY**: выражается через комбинацию операций `<AND>`, `<NOT>`, `<REMOVE>`, `<PROJECT>`.

15 Реляционная алгебра A. Перечислить базовые операции. Избыточность алгебры A. Сокращение набора операций алгебры A.

Базовые операции алгебры A:

- Реляционное дополнение `<NOT>`
- Реляционная конъюнкция `<AND>`
- Реляционная дизъюнкция `<OR>`
- Удаление атрибута `<REMOVE>`
- Переименование атрибута `<RENAME>`
- Присваивание `<:=>`

Избыточность алгебры A:

- Операции `<AND>` и `<OR>` являются избыточными, так как одну можно выразить через другую с использованием `<NOT>` (аналогично законам де Моргана).
- Операцию `<RENAME>` можно выразить через комбинацию `<AND>`, `<NOT>`, `<REMOVE>`, но это менее практично.

Сокращение набора операций алгебры A:

- Набор базовых операций можно сократить до трех:
  - <NOT>
  - <AND> или <OR> (одна из них)
  - <REMOVE>
- Введя аналоги штриха Шеффера (<sh>) или стрелки Пирса (<pi>), можно свести набор операций к двум:
  - <sh> (A, B) = <NOT> A <OR> <NOT> B
  - <pi> (A, B) = <NOT> A <AND> <NOT> B

  Это позволяет реализовать все операции, используя только одну из этих функций и <REMOVE>.

16 Реляционное исчисление: исчисление кортежей и доменов. Сравнение механизмов реляционной алгебры и реляционного исчисления на примере формулирования запроса.

Реляционное исчисление — это прикладная ветвь исчисления предикатов первого порядка, основанная на переменных, предикатах и кванторах. Различают два вида реляционного исчисления:

- **Исчисление кортежей**: переменные принимают значения кортежей из отношений базы данных.
- **Исчисление доменов**: переменные принимают значения из доменов атрибутов.

Сравнение с реляционной алгеброй на примере запроса:
Задание: Получить имена и номера служащих, которые являются руководителями проектов со средней зарплатой более 100000.
Реляционная алгебра (процедурный подход):

```
((СЛУЖАЩИЕ JOIN ПРОЕКТЫ WHERE (СЛУ_N = ПРО_РУК_N))
 WHERE (ПРО_ЗАРП > 100000))
 PROJECT (СЛУ_ИМЯ, СЛУ_N)
```

Шаги:

- Выполнить соединение отношений СЛУЖАЩИЕ и ПРОЕКТЫ по условию СЛУ_N = ПРО_РУК_N.
- Отфильтровать результаты по условию ПРО_ЗАРП > 100000.
- Спроецировать атрибуты СЛУ_ИМЯ и СЛУ_N.

Реляционное исчисление (декларативный подход):

```
RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ
RANGE ПРОЕКТ IS ПРОЕКТЫ
СЛУЖАЩИЙ.СЛУ_ИМЯ, СЛУЖАЩИЙ.СЛУ_N
WHERE EXISTS ПРОЕКТ
  (СЛУЖАЩИЙ.СЛУ_N = ПРОЕКТ.ПРО_РУК_N AND ПРОЕКТ.ПРО_ЗАРП > 100000)
```

Описание: Выбрать значения СЛУ_ИМЯ и СЛУ_N из СЛУЖАЩИЕ, для которых существует кортеж в ПРОЕКТЫ, удовлетворяющий условиям.

Вывод: Реляционная алгебра задает последовательность операций (процедурно), тогда как реляционное исчисление описывает свойства требуемого результата (декларативно).

17 Исчисление кортежей. Кортежная переменная. Правильно построенная формула. Пример. Способ реализации

Кортежная переменная обозначается с помощью оператора RANGE и принимает значения кортежей из указанного отношения.

```
RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ
```

Правильно построенная формула (WFF) используется для выражения условий на кортежные переменные с помощью простых условий и логических операторов (NOT, AND, OR, IF...THEN).

Пример WFF:

```
IF (СЛУЖАЩИЙ.СЛУ_ИМЯ = 'Иванов') THEN (СЛУЖАЩИЙ.ПРО_N = 1)
```

Эта формула проверяет, что если имя служащего "Иванов то его номер проекта равен 1.

Способ реализации - Метод вложенных циклов:

- Просмотреть все кортежи в СЛУЖАЩИЕ.
- Применить WFF к каждому кортежу.
- Выбрать те кортежи, для которых WFF истинна.

18 Исчисление кортежей. Кванторы, свободные и связанные переменные. Целевые списки. Выражения реляционного исчисления.

Кванторы:

- **EXISTS**: проверяет существование кортежа, для которого формула истинна.
- **FORALL**: проверяет, что формула истинна для всех кортежей.

Свободные и связанные переменные:

- **Свободные переменные**: не связаны кванторами; определяют результирующее отношение.
- **Связанные переменные**: связаны кванторами внутри WFF; используются для формулировки условий.

Целевые списки (target list) определяют, какие атрибуты включить в результат.

- `var.attr` (например, `СЛУЖАЩИЙ.СЛУ_ИМЯ`)
- `var` (все атрибуты переменной)
- `new_name = var.attr` (переименование атрибута)

Выражение реляционного исчисления:

```
target_list WHERE WFF
```

Пример:

```
RANGE СЛУ1 IS СЛУЖАЩИЕ
RANGE СЛУ2 IS СЛУЖАЩИЕ
СЛУ1.СЛУ_N, СЛУ1.СЛУ_ИМЯ
WHERE EXISTS СЛУ2
  (СЛУ1.СЛУ_ЗАРП > СЛУ2.СЛУ_ЗАРП)
```

Это выражение выбирает номера и имена служащих, зарплата которых выше, чем у какого-либо другого служащего.

19 Исчисление доменов. Основные отличия от исчисления кортежей.

Основные отличия:

- **Область определения переменных**: в исчислении доменов переменные принимают значения из доменов (типов данных атрибутов), а не из кортежей отношений.
- **Предикат "Условие членства"**: используется для привязки переменных к отношениям.

Формат условия членства:

```
R (атрибут1: переменная1, атрибут2: переменная2, ...)
```

Истинно, если существует кортеж в отношении `R` с указанными значениями.

- В исчислении доменов необходимо явно указать связи между переменными и атрибутами отношений через условия членства.
- Переменные представляют отдельные значения, а не целые кортежи.

Пример:

```
СЛУ_N1, СЛУ_ИМЯ1 WHERE EXISTS СЛУ_ЗАРП2
СЛУЖАЩИЕ (СЛУ_ЗАРП: СЛУ_ЗАРП2) AND
СЛУЖАЩИЕ (СЛУ_N: СЛУ_N1, СЛУ_ИМЯ: СЛУ_ИМЯ1, СЛУ_ЗАРП: СЛУ_ЗАРП1) AND
СЛУ_ЗАРП1 > СЛУ_ЗАРП2
```

Это выражение выбирает номера и имена служащих, чья зарплата не минимальна.

20 Классический подход к проектированию баз данных на основе нормализации. Нормальная форма. Общие свойства нормальных форм. Полный список нормальных форм. Нормализация в OLAP и OLTP системах.

Классический подход включает процесс нормализации, который представляет собой последовательные декомпозиции исходных отношений для достижения удовлетворительных нормальных форм. Начальная точка — представление предметной области в виде одного или нескольких отношений с большим количеством атрибутов. Каждый шаг проектирования предполагает модификацию схем отношений (декомпозицию) для улучшения их свойств, таких как уменьшение избыточности и устранение аномалий обновления.

20.1 Нормальная форма

Нормальная форма — это определенный набор ограничений, которым должна удовлетворять схема отношения. Если отношение соответствует нормальной форме, оно удовлетворяет всем её ограничениям. Каждая нормальная форма направлена на устранение определенных видов аномалий и избыточности.

20.2 Общие свойства нормальных форм

- **Последовательность**: Каждая последующая нормальная форма включает требования предыдущих.
- **Улучшение структуры данных**: Повышение нормальной формы ведет к лучшей организации данных.
- **Сохранение целостности**: Нормализация способствует поддержанию целостности данных и уменьшению избыточности.

20.3 Полный список нормальных форм

1. Первая нормальная форма (1NF)
2. Вторая нормальная форма (2NF)
3. Третья нормальная форма (3NF)

4. Нормальная форма Бойса-Кодда (BCNF)
5. Четвёртая нормальная форма (4NF)
6. Пятая нормальная форма (5NF)

20.4 Нормализация в OLAP и OLTP системах

20.4.1 OLTP (On-Line Transaction Processing)

- **Цель**: Высокий уровень нормализации для обеспечения надежности и уменьшения избыточности.
- **Преимущества**: Повышенная скорость и надежность транзакций, снижение риска аномалий обновления.
- **Особенности**: Большая часть запросов известна заранее, транзакции преимущественно на вставку/удаление/модификацию.

20.4.2 OLAP (On-Line Analytical Processing)

- **Цель**: Часто применяется денормализация для оптимизации выполнения сложных аналитических запросов.
- **Преимущества**: Ускорение обработки больших объемов данных, повышение эффективности выполнения нерегламентированных и сложных запросов.
- **Особенности**: Добавление данных происходит редко крупными блоками, данные не удаляются, запросы нерегламентированные и сложные.

21 Функциональная зависимость. Пример отношения и его функциональных зависимостей. Связь функциональных зависимостей и ограничений целостности. Тривиальная FD. Транзитивная FD.

21.1 Функциональная зависимость

Функциональная зависимость (FD) — отношение, при котором значение одного или нескольких атрибутов (детерминант) однозначно определяет значение другого атрибута.

21.2 Пример отношения и его функциональных зависимостей

Рассмотрим отношение **СЛУЖ_ПРО_ЗАДАН** с атрибутами {СЛУ_N, ПРО_N, СЛУ_ЗАДАН}.

- **Ключ отношения**: {СЛУ_N, ПРО_N, СЛУ_ЗАДАН}.

- В данном отношении отсутствуют нетривиальные FD, что указывает на соблюдение нормальной формы BCNF.

21.3 Связь функциональных зависимостей и ограничений целостности

Функциональные зависимости обеспечивают целостность данных, гарантируя правильные зависимости между атрибутами. Они помогают предотвратить избыточность и аномалии обновления, поддерживая согласованность данных в базе.

21.4 Тривиальная функциональная зависимость

Тривиальная FD — зависимость, в которой зависимые атрибуты входят в детерминант или равны всему множеству атрибутов.

- **Пример**: $A \to A$.

Тривиальная FD всегда выполняется и не представляет интереса с практической точки зрения. Однако в теоретических рассуждениях их наличие необходимо учитывать.

21.5 Транзитивная функциональная зависимость

Транзитивная FD возникает, когда $A \to B$ и $B \to C$, следовательно, $A \to C$.

- **Пример**: Если $\_N \to \_$ и $\_ \to \_$, тогда $\_N \to \_$.
- **Последствия**: Создает косвенные зависимости между атрибутами, что может приводить к избыточности данных и аномалиям обновления.

22 Замыкание множества функциональных зависимостей. Аксиомы Армстронга (с доказательством). Расширенный набор правил вывода Дейта (с выводом).

22.1 Тривиальная функциональная зависимость

Определение 22.1. *FD $A \to B$ называется тривиальной, если $B \subseteq A$, то есть множество атрибутов A включает множество B или совпадает с множеством B.*

Очевидно, что любая тривиальная FD всегда выполняется. Например, в отношении **СЛУЖ_ПРО_ЗАДАН_1** всегда выполняется FD {СЛУ_ЗАРП, ПРО_НОМ} → {СЛУ_ЗАРП}.

Частным случаем тривиальной FD является $A \to A$. Поскольку тривиальные FD выполняются всегда, их нельзя трактовать как ограничения целостности, и поэтому они не представляют интереса с практической точки зрения. Однако в теоретических рассуждениях их наличие необходимо учитывать.

22.2 Транзитивная функциональная зависимость

Определение 22.2. *FD $A \to C$ называется транзитивной, если существует такой атрибут B, что имеются функциональные зависимости $A \to B$ и $B \to C$, и отсутствует функциональная зависимость $C \to A$.*

22.3 Аксиомы Армстронга

Пусть A, B и C являются (в общем случае, составными) атрибутами переменной отношения r. Множества A, B и C могут иметь непустое пересечение. Для краткости будем обозначать через AB объединение $A \cup B$.

1. Если $B \subseteq A$, то выполняется FD $A \to B$ (аксиома рефлексивности).
2. Если выполняется FD $A \to B$, то выполняется и FD $AC \to BC$ (аксиома пополнения).
3. Если выполняются FD $A \to B$ и $B \to C$, то выполняется и FD $A \to C$ (аксиома транзитивности).

22.3.1 Доказательство аксиом Армстронга

Доказательство первой аксиомы (рефлексивности): Если $B \subseteq A$, то FD $A \to B$ является тривиальной и, следовательно, выполняется по определению тривиальной FD.

Доказательство второй аксиомы (пополнения): Пусть выполняется FD $A \to B$. Необходимо показать, что выполняется FD $AC \to BC$.

Доказательство. Предположим, что FD $AC \to BC$ не выполняется. Это означает, что существует кортеж t_1 и t_2 в r, такие что $t_1(AC) = t_2(AC)$, но $t_1(BC) \neq t_2(BC)$. Из $t_1(AC) = t_2(AC)$ следует, что $t_1(A) = t_2(A)$ и $t_1(C) = t_2(C)$. Поскольку $A \to B$, из $t_1(A) = t_2(A)$ следует, что $t_1(B) = t_2(B)$. Таким образом, $t_1(BC) = t_2(BC)$, что противоречит предположению. Следовательно, FD $AC \to BC$ выполняется. □

Доказательство третьей аксиомы (транзитивности): Пусть выполняются FD $A \to B$ и $B \to C$. Необходимо показать, что выполняется FD $A \to C$.

Доказательство. Предположим, что FD $A \to C$ не выполняется. Это означает, что существует кортежи t_1 и t_2 в r, такие что $t_1(A) = t_2(A)$, но $t_1(C) \neq t_2(C)$. Из $A \to B$ следует, что $t_1(B) = t_2(B)$. Из $B \to C$ следует, что $t_1(C) = t_2(C)$, что противоречит предположению. Следовательно, FD $A \to C$ выполняется. □

Таким образом, система правил вывода Армстронга полна и совершенна в том смысле, что для любого множества FD S любая FD, потенциально выводимая из S, может быть выведена на основе аксиом Армстронга, и применение этих аксиом не может привести к выводу лишней FD.

22.4 Расширенный набор правил вывода Дейта

Дейт предложил расширить базовый набор правил вывода Армстронга следующими правилами:

1. **Самодетерминированность**: Для любого атрибута A выполняется FD $A \to A$. Это следует из аксиомы рефлексивности.

2. **Декомпозиция**: Если выполняется FD $A \to BC$, то выполняются FD $A \to B$ и $A \to C$. Это прямо следует из аксиомы декомпозиции.

3. **Декомпозиция при наличии цепочки зависимостей**: Если выполняются FD $A \to B$ и $C \to D$, то выполняется FD $AC \to BD$. Это следует из аксиомы пополнения и транзитивности.

4. **Объединение**: Если выполняются FD $A \to B$ и $A \to C$, то выполняется FD $A \to BC$. Это следует из аксиомы пополнения и транзитивности.

5. **Композиция**: Если выполняются FD $A \to B$ и $B \to C$, то выполняется FD $A \to C$. Это следует из аксиомы транзитивности.

6. **Накопление**: Если выполняются FD $A \to BC$ и $B \to D$, то выполняется FD $A \to BCD$. Это следует из аксиомы пополнения и транзитивности.

Эти правила позволяют более гибко выводить функциональные зависимости из заданного множества FD.

23 Замыкание множества атрибутов на множестве FD. Алгоритм построения. Пример. Польза. Суперключ отношения, его связь с замыканием и FD.

Определение 23.1. ***Замыкание множества атрибутов****. Пусть заданы переменная отношения r, множество Z атрибутов этого отношения (подмножество Hr, или составной атрибут r) и некоторое множество FD S, выполняемых для r. Тогда замыканием Z над S называется наибольшее множество Z^+ таких атрибутов Y отношения r, что FD $Z \to Y$ выводится из S^+.*

23.1 Алгоритм построения замыкания

$k \leftarrow 0$ $Z[0] \leftarrow Z$ $k \leftarrow k+1$ $Z[k] \leftarrow Z[k-1]$ каждую FD $A \to B$ в S $A \subseteq Z[k]$ $Z[k] \leftarrow Z[k] \cup B$ $Z[k] = Z[k-1]$ $Z^+ \leftarrow Z[k]$

23.2 Пример

Пусть имеется отношение с заголовком {A, B, C, D, E, F} и заданное множество FD $S = \{A \to D, AB \to E, BF \to E, CD \to F, E \to C\}$. Требуется найти замыкание множества атрибутов $\{A, E\}$.

- **Первый проход**:

$$Z[0] = \{A, E\}$$
$$A \to D \Rightarrow Z[1] = \{A, E, D\}$$
$$E \to C \Rightarrow Z[1] = \{A, E, D, C\}$$

- **Второй проход**:

$$Z[1] = \{A, E, D, C\}$$
$$CD \to F \Rightarrow Z[2] = \{A, E, D, C, F\}$$

- **Третий проход**:

$$Z[2] = \{A, E, D, C, F\}$$

Никаких новых FD не добавляется $\Rightarrow Z^+ = \{A, E, D, C, F\}$

23.3 Польза замыкания атрибутов

Замыкание множества атрибутов позволяет определить, какие атрибуты функционально зависят от данного множества. Это полезно для определения суперключей и для проверки, является ли определенная функциональная зависимость следствием заданного множества FD.

23.4 Суперключ отношения

Определение 23.2. ***Суперключ*** *отношения r — любое подмножество K заголовка Hr, включающее, по меньшей мере, хотя бы один возможный ключ r.*

Одно из следствий этого определения состоит в том, что подмножество K заголовка Hr является суперключом тогда и только тогда, когда для любого атрибута A (возможно, составного) из заголовка отношения r выполняется FD $K \to A$. В терминах замыкания множества атрибутов K является суперключом тогда и только тогда, когда $K^+ = Hr$.

24 Покрытие множества FD, эквивалентные покрытия, минимальное множество FD. Примеры. Алгоритм построения минимального эквивалентного множества. Минимальное покрытие множества функциональных зависимостей.

24.1 Покрытие множества FD

Множество FD S_2 называется **покрытием** множества FD S_1, если любая FD, выводимая из S_1, выводится также и из S_2. Легко заметить, что S_2 является покрытием S_1 тогда и только тогда, когда $S_1^+ \subseteq S_2^+$.

Два множества FD S_1 и S_2 называются **эквивалентными покрытиями**, если каждое из них является покрытием другого, то есть $S_1^+ = S_2^+$.

24.2 Минимальное множество FD

Определение 24.1. ***Минимальное множество FD*** *— множество FD S, которое удовлетворяет следующим условиям:*

1. *Правая часть каждой FD из S содержит только один атрибут.*
2. *Детерминант каждой FD из S минимален, т.е. удаление любого атрибута из детерминанта приводит к утрате зависимости.*
3. *Удаление любой FD из S приводит к изменению замыкания, т.е. множество FD перестает быть эквивалентным исходному.*

24.3 Пример минимального множества FD

Пусть задано отношение **СЛУЖ** {СЛУ_N, СЛУ_УРОВ, СЛУ_ЗАРП} с множеством FD:

$$\{\text{СЛУ\_Н} \to \text{СЛУ\_УРОВ},\ \text{СЛУ\_Н} \to \text{СЛУ\_ЗАРП},$$

$$\text{СЛУ\_Н} \to \text{ПРО\_Н},\ \text{ПРО\_Н} \to \text{ПРОЕКТ\_РУК}\}$$

Это множество FD является минимальным, так как:

- Правая часть каждой FD содержит только один атрибут.
- Детерминанты минимальны — удаление любого атрибута из детерминанта нарушает зависимость.
- Удаление любой FD приводит к изменению замыкания.

24.4 Алгоритм построения минимального эквивалентного множества

1. **Декомпозиция правых частей FD**: Привести все FD к форме, где правая часть содержит только один атрибут.
2. **Минимизация детерминантов**: Для каждой FD проверить, можно ли удалить атрибуты из детерминанта без потери зависимости.
3. **Удаление избыточных FD**: Проверить каждую FD на избыточность и удалить ее, если она выводится из остальных FD.

24.5 Пример построения минимального покрытия

Рассмотрим множество FD $S = \{A \to B, A \to BC, AB \to C, AC \to D, B \to C\}$.

1. **Декомпозиция правых частей FD**:

$$S_1 = \{A \to B, A \to B, A \to C, AB \to C, AC \to D, B \to C\}$$

2. **Минимизация детерминантов**:
   - Для FD $AC \to D$ можно удалить атрибут C, получив $A \to D$.

Теперь множество FD:

$$S_2 = \{A \to B, A \to C, AB \to C, A \to D, B \to C\}$$

3. **Удаление избыточных FD**:

   - FD $AB \to C$ выводится из $A \to C$, поэтому ее можно удалить.

Минимальное покрытие:

$$S_{\min} = \{A \to B, A \to C, A \to D, B \to C\}$$

Таким образом, полученное множество $S_{\min}$ является минимальным покрытием, эквивалентным исходному множеству S.

25 Корректные и некорректные декомпозиции отношений. Теорема Хита (с доказательством). Минимально зависимые атрибуты.

25.1 Корректные и некорректные декомпозиции отношений

Корректная декомпозиция — это разбиение отношения на подотношения без потери данных и устранения аномалий обновления.

Некорректная декомпозиция — приводит к потере данных или возникновению аномалий обновления.

25.2 Теорема Хита

Теорема 25.1 (Теорема Хита)**.** *Пусть задано отношение r с заголовком $\{A, B, C\}$, и выполняется FD $A \to B$. Тогда декомпозиция:*

$$r = (r\ PROJECT\ \{A, B\})\ NATURAL\ JOIN\ (r\ PROJECT\ \{A, C\})$$

является декомпозицией без потерь.

Доказательство. Пусть R — некоторое допустимое значение переменной r. Обозначим результат операции R PROJECT $\{A, B\}$ как R_1, результат операции R PROJECT $\{A, C\}$ как R_2, а результат R_1 NATURAL JOIN R_2 как R_3.

Докажем, что R_3 содержит все кортежи, содержащиеся в R.

Пусть кортеж $\langle A, B, C\rangle \in R$. Тогда по определению операции проекции $\langle A, B\rangle \in R_1$ и $\langle A, C\rangle \in R_2$. Следовательно, $\langle A, B, C\rangle \in R_3$. □

25.3 Минимально зависимые атрибуты

Определение 25.1. ***Минимально зависимые атрибуты*** *— атрибуты, которые функционально зависят от ключа, и для которых невозможно удалить любой атрибут из детерминанта без утраты зависимости.*

26 Минимальные функциональные зависимости. Аномалии, возникающие из-за наличия неминимальных FD. Пример декомпозиции, решающей проблему. 2НФ

26.1 Минимальные функциональные зависимости

Атрибут **B** минимально зависит от атрибута **A**, если выполняется минимальная слева функциональная зависимость $\mathbf{A} \to \mathbf{B}$.

26.2 Аномалии из-за неминимальных FD

Наличие неминимальных функциональных зависимостей приводит к аномалиям обновления:

- **Добавление:** Невозможно добавить служащего без участия в проекте.
- **Удаление:** Удаление последнего проекта служащего приводит к потере информации о его зарплате.
- **Модификация:** Изменение зарплаты требует обновления всех кортежей с этим служащим.

26.3 Пример декомпозиции

Исходное отношение **СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ** декомпозируется на:

- **СЛУЖ** {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП}
- **СЛУЖ_ПРО_ЗАДАН** {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}

Эта декомпозиция устраняет аномалии обновления и приводит отношения к **Второй нормальной форме (2НФ)**.

2НФ (Вторая нормальная форма):

Переменная отношения находится во второй нормальной форме (2NF) тогда и только тогда, когда она находится в первой нормальной форме, и каждый ее неключевой атрибут минимально функционально зависит от первичного ключа.

27 Транзитивные функциональные зависимости. Аномалии, возникающие из-за наличия транзитивных FD. Пример декомпозиции, решающей проблему. 3НФ

27.1 Транзитивные функциональные зависимости

Функциональная зависимость $\mathbf{A} \to \mathbf{C}$ называется транзитивной, если существует атрибут **B**, такой что $\mathbf{A} \to \mathbf{B}$ и $\mathbf{B} \to \mathbf{C}$, при отсутствии $\mathbf{C} \to \mathbf{A}$.

27.2 Аномалии из-за транзитивных FD

Наличие транзитивных FD вызывает аномалии обновления:

- **Добавление:** Невозможно добавить новый разряд и зарплату без существования служащего.
- **Удаление:** Удаление последнего служащего с разрядом приводит к потере информации о разряде и зарплате.
- **Модификация:** Изменение зарплаты требует обновления всех соответствующих кортежей.

27.3 Пример декомпозиции

Отношение **СЛУЖ** делится на:

- **СЛУЖ1** {СЛУ_НОМ, СЛУ_УРОВ}
- **УРОВ** {СЛУ_УРОВ, СЛУ_ЗАРП}

Эта декомпозиция устраняет транзитивные зависимости и приводит отношения к **Третьей нормальной форме (3НФ)**.

3НФ (Третья нормальная форма):

Переменная отношения находится в третьей нормальной форме (3NF) тогда и только тогда, когда она находится во второй нормальной форме, и каждый неключевой атрибут нетранзитивно функционально зависит от первичного ключа.

28 Независимые проекции отношений. Теорема Риссанена (без доказательства). Атомарные отношения.

28.1 Независимые проекции отношений

Проекции отношений называются независимыми, если они могут обновляться независимо без нарушения целостности данных.

28.2 Теорема Риссанена

Проекции r_1 и r_2 отношения r являются независимыми тогда и только тогда, когда:

1. Каждая функциональная зависимость (FD) в отношении r выводится из FD в r_1 и r_2.
2. Общие атрибуты r_1 и r_2 образуют возможный ключ хотя бы для одного из этих отношений.

28.3 Атомарные отношения

Атомарной переменной отношения называется такое отношение, которое невозможно декомпозировать на независимые проекции без потери информации или появления аномалий.

29 Перекрывающиеся возможные ключи, аномалии обновления, возникающие из-за их наличия. Нормальная форма Бойса-Кодда.

29.1 Перекрывающиеся возможные ключи

Отношение имеет несколько возможных ключей, которые перекрываются (имеют общие атрибуты). Это приводит к аномалиям обновления, например, необходимость обновления атрибутов в нескольких кортежах.

29.2 Аномалии из-за перекрывающихся ключей

- **Модификация:** Изменение атрибута, входящего в несколько ключей, требует обновления всех соответствующих кортежей.
- **Удаление:** Удаление кортежа может привести к потере информации, если этот атрибут является частью нескольких ключей.

29.3 Нормальная форма Бойса-Кодда (BCNF)

Отношение находится в BCNF, если для любой нетривиальной и минимальной FD детерминант является некоторым возможным ключом.

29.4 Пример и декомпозиция в BCNF

Отношение **СЛУЖ_ПРО_ЗАДАН1** с перекрывающимися ключами {**СЛУ_НОМ, ПРО_НОМ**} и {**СЛУ_ИМЯ, ПРО_НОМ**} декомпозируется на:

- **СЛУЖ_НОМ_ИМЯ** {СЛУ_НОМ, СЛУ_ИМЯ}
- **СЛУЖ_ИМЯ_ПРО_ЗАДАН** {СЛУ_ИМЯ, ПРО_НОМ, СЛУ_ЗАДАН}

Эта декомпозиция устраняет аномалии обновления и приводит отношения к **BCNF**.

29.5 Всегда ли следует стремиться к BCNF?

Нет. В некоторых случаях декомпозиция до BCNF может привести к дополнительным сложностям, таким как необходимость введения ограничений целостности, что может увеличить технические накладные расходы. Поэтому при проектировании необходимо оценивать преимущества и недостатки нормализации до BCNF.

30 Многозначные зависимости. Двойственность многозначной зависимости. Лемма Фейджина. Теорема Фейджина (с доказательством).

30.1 Многозначные зависимости (MVD)

Многозначная зависимость атрибута **A** от атрибута **B** обозначается как $\mathbf{A} \to\to \mathbf{B}$ и означает, что множеству значений **B** соответствует множество значений **C**,

независимо от **A**.

30.2 Двойственность MVD

Многозначные зависимости обладают свойством двойственности. Согласно лемме Фейджина, в отношении $\{A, B, C\}$ выполняется $\mathbf{A} \to\to \mathbf{B}$ тогда и только тогда, когда $\mathbf{A} \to\to \mathbf{C}$.

30.3 Лемма Фейджина

Если выполняется $\mathbf{A} \to\to \mathbf{B}$, то автоматически выполняется $\mathbf{A} \to\to \mathbf{C}$, и наоборот.

30.4 Теорема Фейджина

Отношение **r** с атрибутами **A, B, C** удовлетворяет

$$\mathbf{r} = (\mathbf{r} \text{ PROJECT } \{\mathbf{A}, \mathbf{B}\}) \text{ NATURAL JOIN } (\mathbf{r} \text{ PROJECT } \{\mathbf{A}, \mathbf{C}\})$$

тогда и только тогда, когда выполняется $\mathbf{A} \to\to \mathbf{B}|\mathbf{C}$.

30.4.1 Доказательство

Достаточность: Если $\mathbf{A} \to\to \mathbf{B}|\mathbf{C}$, то любое значение **r** можно восстановить через естественное соединение проекций $\{A, B\}$ и $\{A, C\}$. Это следует из определения многозначной зависимости и свойств естественного соединения.

Необходимость: Если **r** равно соединению проекций $\{A, B\}$ и $\{A, C\}$, то должны выполняться $\mathbf{A} \to\to \mathbf{B}$ и $\mathbf{A} \to\to \mathbf{C}$, что подтверждает $\mathbf{A} \to\to \mathbf{B}|\mathbf{C}$. Это противоречие возникает, если хотя бы один из необходимых кортежей отсутствует, что невозможно при выполнении равенства $\mathbf{r} = \mathbf{r_1}$ NATURAL JOIN $\mathbf{r_2}$.

31 Многозначные зависимости. Аномалии, возникающие из-за наличия MVD. Пример декомпозиции, решающей проблему (на чем основывается). 4НФ. Нетривиальная и тривиальная многозначные зависимости.

31.1 Нетривиальная и тривиальная MVD

- **Тривиальная MVD:** Выполняется, если **A** включает **B** или $\mathbf{A} \cup \mathbf{B}$ совпадает с заголовком отношения.
- **Нетривиальная MVD:** Не является тривиальной и приводит к аномалиям обновления.

31.2 Аномалии из-за MVD

- **Добавление:** Требуется добавлять несколько кортежей одновременно.
- **Удаление:** Потеря информации о заданиях при удалении служащего.

- **Модификация:** Необходимость изменения атрибута в нескольких кортежах.

31.3 Пример декомпозиции

Отношение **СЛУЖ_ПРО_ЗАДАН** декомпозируется на:

- **СЛУЖ_ПРО_НОМ** {СЛУ_НОМ, ПРО_НОМ}
- **СЛУЖ_ЗАДАНИЕ** {СЛУ_НОМ, СЛУ_ЗАДАН}

31.4 Основание декомпозиции

Декомпозиция основана на устранении многозначных зависимостей СЛУ_НОМ $\rightarrow$ ПРО_НОМ и СЛУ_НОМ $\rightarrow$ СЛУ_ЗАДАН, приводящих отношение к **Четвертой нормальной форме (4НФ)**.

4НФ (Четвертая нормальная форма):

Переменная отношения `r` находится в четвертой нормальной форме (4NF) тогда и только тогда, когда она находится в BCNF, и все MVD `r` являются FD с детерминантами — возможными ключами отношения `r`.

32 N-декомпозируемые отношения. Пример декомпозиций. Зависимость проекции/соединения.

32.1 N-декомпозируемые отношения

Отношение называется **N-декомпозируемым**, если его можно декомпозировать без потерь на **N** проекций.

32.2 Пример декомпозиции

Отношение **СЛУЖ_ПРО_ЗАДАН** декомпозируется на три проекции:

- **СЛУЖ_ПРО_НОМ** {СЛУ_НОМ, ПРО_НОМ}
- **СЛУЖ_ЗАДАНИЕ** {СЛУ_НОМ, СЛУ_ЗАДАН}
- **ПРО_НОМ_ЗАДАН** {ПРО_НОМ, СЛУ_ЗАДАН}

32.3 Зависимость проекции/соединения

После декомпозиции отношения выполняется зависимость проекции/соединения, обеспечивая восстановление исходного отношения через естественное соединение всех проекций.

33 Аномалии, возникающие из-за наличия зависимости проекции/соединения. Пример декомпозиции, решающей проблему. 5НФ.

33.1 Аномалии из-за зависимости проекции/соединения (PJD)

- **Добавление:** Некорректное добавление кортежей без соблюдения ограничений целостности.
- **Удаление:** Потеря связанных кортежей при удалении.
- **Модификация:** Нарушение целостности данных при изменении атрибутов.

33.2 Пример декомпозиции

Отношение **СЛУЖ_ПРО_ЗАДАН** декомпозируется на три проекции:

- **СЛУЖ_ПРО_НОМ** {СЛУ_НОМ, ПРО_НОМ}
- **СЛУЖ_ЗАДАНИЕ** {СЛУ_НОМ, СЛУ_ЗАДАН}
- **ПРО_НОМ_ЗАДАН** {ПРО_НОМ, СЛУ_ЗАДАН}

33.3 Решение проблемы

Эта декомпозиция устраняет аномалии обновления, связанные с зависимостью проекции/соединения, и приводит отношение к **Пятой нормальной форме (5НФ)**, обеспечивая восстановление исходного отношения без потерь.

33.4 Пятая нормальная форма (5НФ)

Отношение находится в 5НФ, если каждая нетривиальная зависимость проекции/соединения подразумевается возможными ключами. Это гарантирует отсутствие аномалий обновления, которые можно было бы устранить декомпозицией.

34 Организация внешней памяти в SQL-ориентированных базах данных, хранение таблиц по строкам и столбцам. Понятие tid-а.

В современных SQL-ориентированных базах данных есть несколько особенностей, влияющих на организацию внешней памяти. Во-первых, есть двухуровневая система: на одном уровне производится непосредственное управление данными во внешней памяти, буферами оперативной памяти, транзакциями и журнализацией, на другом — реализация языка SQL. Во-вторых, поддерживаются таблицы-каталоги, содержащие метаданные, описывающие все объекты базы данных и при этом ими же являющиеся. В-третьих, основной набор объектов внешней памяти имеет простую регулярную структуру, но при этом во внешней памяти поддерживаются дополнительные управляющие структуры —

индексы, позволяющие эффективно выполнять операции SQL языкового уровня. В-четвертых, поддерживается избыточность хранения данных для выполнения требований надежности. Соответственно, во внешней памяти содержатся: строки таблиц — основная часть БД, большей частью видимая пользователю, управляющие структуры, служебная и журнальная информация.

Существуют два принципиальных подхода к физическому хранению таблиц. Наиболее распространенным является построчное хранение, при котором единицей физического хранения является строка таблицы. Требуется, чтобы строка целиком хранилась в одном блоке внешней памяти, что обеспечивает быстрый к ней доступ, но ведет к дублированию общих значений разных строк и замедлению работы, если строки длинные, а нужна только их часть (частый случай в аналитических базах данных).

Страница данных — это блок данных во внешней памяти, в котором хранятся строки каких-то таблиц. У каждой строки есть уникальный идентификатор

$$\text{tid} = \langle N, i \rangle,$$

где N – номер страницы, i – смещение от начала блока до описателя. В описателе хранится ссылка на начало строки в этой же таблице или (если в процессе работы строка перестала помещаться на странице) ссылка на другую страницу. Число строк в странице не ограничено, для реализации динамического числа строк в странице используют метод двух указателей: область описателей растет сверху вниз, область хранения — снизу вверх.

Для хранения очень больших строк самым простым решением является использование отдельных файлов, другим возможным решением — хранение длинных строк в отдельном наборе страниц внешней памяти, связанном физическими ссылками. Оба этих способа ставят под вопрос как надежность хранения, так и скорость работы с длинными данными. Еще одним методом является хранение длинных данных на основе B-деревьев, а в сегодняшнее время поддерживается внутренняя файловая система внутри SQL-сервера.

Обычно в одной странице хранятся строки одной таблицы, но возможен и обратный случай, повышающий эффективность некоторых операций, но требующий хранения дополнительной служебной информации.

При добавлении нового столбца в таблицу физической реорганизации таблицы не происходит, но в описатель строки добавляется новый столбец. Строки расширяются только при занесении информации в новое поле.

Еще один нетривиальный момент — распределение памяти в страницах данных. К примеру, если в ходе транзакции блок диска освободился, то он не может быть использован другой транзакцией до подтверждения первой из-за возможности отката.

Еще одним способом повышения эффективности СУБД является кластеризация таблицы — указание, что в одном блоке внешней памяти надо хранить те строки таблицы, у которых значения столбца кластеризации близки. Тогда некоторые запросы выполняются быстрее, но со временем кластеризация ухудшается и требует физической реструктуризации таблицы. Также возможна совместная кластеризация. Обратным подходом является декластеризация, используемая с целью использования возможностей распараллеливания обменов с внешней памятью.

Альтернативным подходом является хранение таблицы по столбцам, что экономит память и дает возможность оптимизировать операции соединения и

быстрее считать значения агрегатных функций, но, очевидно, требует дополнительных действий для сборки целой строки. Для каждой строки таблицы здесь хранится строка, состоящая из ссылок на места расположения соответствующих значений столбцов.

35 Индексы в базе данных. Индексы на основе B-деревьев.

Основным назначением индексов является обеспечение эффективного прямого доступа к строке таблицы по ключу. Обычно индекс определяется для одной таблицы, и ключом является значение ее поля. Если ключом индекса является возможный ключ таблицы, то индекс должен обладать свойством уникальности. Полезным свойством индекса является обеспечение последовательного просмотра строк таблицы в заданном диапазоне значений ключа в порядке возрастания или убывания его значения. Общей идеей любой организации такого индекса является хранение упорядоченного списка значений ключа с привязкой к каждому значению списка идентификаторов строк.

Наиболее популярным методом организации индексов в базе данных является использование B-деревьев. B-дерево — это сбалансированное сильно ветвистое дерево во внешней памяти, сбалансированность значит, что длина пути от корня дерева к любому его листу одна и та же. Ветвистость — это свойство каждого узла дерева ссылаться на большое число узлов-потомков. С точки зрения физической организации B-дерево представляется как мультисписочная структура страниц внешенй памяти, то есть каждому узлу дерева соответствует блок внешней памяти. Недостаток B-деревьев состоит в трудности балансировки.

Модификацией B-дерева является B+-дерево, которое достаточно просто балансировать. В B+-дереве внутренние и листовые страницы имеют разную структуру. Типовая структура внутренней страницы выглядит как упорядоченная последовательность ключей и ссылок на узлы более низкого уровня, причем узел, заключенный между двух ключей, содержит только ключи, заключенные между этих ключей. Листовая страница выглядит как упорядоченная последовательность ключей и списков идентификаторов строк, причем в n-м списке содержатся строки с n-м ключом.

Поиск в B+ дереве есть прохождение от корня к листу в соответствии с заданным значением ключа. Поскольку B+ деревья сильно ветвисты и сбалансированы, для поиска требуется одно и то же обычно небольшое число обменов с внешней памятью. Более точно, если во внутренней странице помещается n ключей, то для хранения m записей требуется дерево глубиной $\log_n(m)$.

Основным плюсом B+-деревьев является автоматическое поддержание свойства сбалансированности. Если при вставке новой записи закончилось место в дереве, выполняется расщепление страницы, куда производилась запись, и результат примерно пополам записывается в две страницы. Если не осталось места для записи ключа (чтобы к новой странице можно было добраться), производится аналогичная операция уровнем выше, и так может дойти до корневой страницы (но на самом деле такое происходит очень редко). С удалением аналогично, но вместо расщепления производится слияние. Стоит заметить, что для повышения эффективности следует производить расщепление и слияние не в тот момент, когда место закончилось совсем, а так, чтобы иметь некоторый запас.

С B+-деревьями из-за автоматической балансировки сложно работать в режиме мультидоступа, так как блокировать все дерево затратно (потому что оно сильно ветвистое и по факту проблемы могут и не возникнуть), а для неполной блокировки надо знать, насколько вверх поднимется расщепление.

36 Индексы в базе данных. Индексы на основе хэширования: расширяемое хэширование.

Основным назначением индексов является обеспечение эффективного прямого доступа к строке таблицы по ключу. Обычно индекс определяется для одной таблицы, и ключом является значение ее поля. Если ключом индекса является возможный ключ таблицы, то индекс должен обладать свойством уникальности.

Альтернативным по отношению к использованию B-деревьев подходом организации индексов является использование техники хеширования. Общей идеей является применение к значению ключа некоторой функции свёртки, вырабатывающей значение меньшего размера. Значение хэш-функции затем используется для доступа к записи. Основным требованием к хэш-функции является равномерное распространение значений свертки. При возникновении коллизий (одна и та же свертка для разных ключей) образуются цепочки переполнения. Главным ограничением здесь является фиксированный размер таблицы. Если таблица слишком сильно заполнена или переполнена, то время поиска по цепочке переполнения сведет на нет главное преимущество хэширования — доступ к записи почти всегда за одно обращение. Расширение таблицы требует ее полной переделки на основе новой хэш-функции.

Двумя наиболее часто используемыми методами хэширования являются расширяемое и линейное хэширования.

В основе подхода расширяемого хэширования лежит принцип использования деревьев цифрового поиска в оперативной памяти. В ОП поддерживается справочник, организованный на основе бинарного дерева цифрового поиска, ключами которого являются значения хэш-функции, а в листовых вершинах хранятся номера страниц записей во внешней памяти. Здесь под коллизией понимается переполнение страницы внешней памяти, и в этом случае страница расщепляется на две, и дерево цифрового поиска переформируется. При этом может потребоваться расширение справочника.

Метод расширяемого хеширования заключается в том, что хеш-таблица представлена как каталог, а каждая ячейка будет указывать на бакет, который имеет определенную вместимость. Сама хеш-таблица будет иметь глобальную глубину G, а каждая из емкостей имеет локальную глубину l_i. Глобальная глубина G показывает сколько последних бит будут использоваться для того чтобы определить в какую емкость следует заносить значения. А из разницы локальной глубины и глобальной глубины можно понять, сколько ячеек каталога ссылаются на емкость. Алгоритм для ключа k с значением свёртки $h(k)$: 1) По первым G битам свёртки $h(k)$ решаем в какой бакет отправить ключ. 2) Если в бакете есть свободное место, то помещаем туда ключ, если бакет переполнен, смотрим на локальную глубину бакета: 2.1) Если локальная глубина меньше, чем глобальная глубина, то создаём новый бакет и заново перераспределяем все ключи в текущем бакете с учётом новой длины бит. Не забыть увеличить глубину обоих бакетов на 1. Возвращаемся к шагу 1. 2.2) Если локальная глубина

равна глобальной, то мы увеличиваем глобальную глубину на 1, удваивая при этом количество ячеек, количество указателей на бакеты, а также увеличиваем количество последних бит, по которым мы распределяем значения.

37 Индексы в базе данных. Индексы на основе хэширования: линейное хэширование.

Вопрос 35 + : Основой метода линейного хэширования является то, что для адресации страницы внешней памяти всегда используются младшие биты значения хэш-функции. Если возникает потребность в расщеплении, то записи перераспределяются по страницам так, чтобы адресация осталась правильной.

38 Этапы проектирования баз данных. Ограниченность реляционной модели при использовании в проектировании. Понятие концептуальной (семантической, инфологической) модели. Достоинства концептуальных моделей. Средства автоматизации проектирования баз данных.

Этапы проектирования БД

Концептуальное, Логическое, Физическое проектирование БД — процесс создания модели информации в виде описания основных понятий предметной области (объектов и связей между ними).

Концептуальное проектирование БД– процесс создания модели информации, не зависящий от любых физических аспектов ее представления.

Логическое проектирование БД– процесс создания модели информации с учетом выбранной модели организации данных, но независимо от типа целевой СУБД и других физических аспектов реализации.

Физическое проектирование БД– процесс описания реализации БД для выбранной целевой СУБД с учетом ее особенностей, а также указание способов хранения данных на внешних ЗУ и методов эффективного доступа к данным, включая определение индексов.

Логическое проектирование реляционных баз данных:

• При выборе реляционной модели данных на этапе логического проектирования создается схема базы данных как набор именованных отношений с определенными заголовками, ключами. На этом же этапе осуществляется нормализация данных отношений.

• Широкое распространение реляционных (SQL-ориентированных) СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель данных достаточна для моделирования разнообразных предметных областей.

• Предметная область– часть реального мира, рассматриваемая в определенном смысле как единое целое, сведения о которой представляют собой информационные ресурсы в аспектах создания и использования БД.

• Однако проектирование реляционной базы данных в терминах отношений на основе рассмотренного в курсе лекций механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс.

• При использовании в проектировании ограниченность реляционной модели проявляется в нескольких аспектах.

Ограниченность реляционной модели данных при использовании в проектировании:

1. Неудобство для проектировщиков: а) На ранних стадиях проектирования, как правило, требуется участие специалистов, хорошо знающих предметную область, но не владеющих теорией БД.

б) Во многих предметных областях трудно осуществлять моделирование информации на основе плоских таблиц.

2. Отсутствие наглядности: а) Реляционная модель не предлагает какого-либо механизма для разделения объектов предметной области и связей между ними.

б) Реляционная модель не обеспечивает достаточных средств для представления семантики (смысла) данных. Это касается, прежде всего, представления ограничений целостности, выходящих за пределы ограничений первичного и внешнего ключей.

3. Невозможность автоматизации процесса проектирования: Реляционная модель не предоставляет какие-либо формализованные средства для представления функциональных и других зависимостей, на основе которых осуществляется процесс проектирования.

Концептуальное проектирование баз данных:

Потребность проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области привела к появлению концептуальных или семантических моделей данных.

Концептуальная (семантическая, инфологическая) модель – способ представления понятий или объектов предметной области, описывающий существенные для данного представления совокупности объектов, их параметры, поведение и отношения между ними. Как правило, такие модели представляются графическими нотациями (т.е. в виде диаграмм).

Семантическое моделирование используется на первой (самой ранней) стадии проектирования базы данных. В терминах той или иной семантической модели производится концептуальная схема базы данных, которая затем преобразуется к реляционной (или какой-либо другой) схеме. Этот процесс выполняется под управлением методик, в которых достаточно четко оговорены все этапы такого преобразования.

Концептуальное проектирование баз данных:

От проектировщиков требуется только знание основ выбранной семантической модели и правил преобразования концептуальной схемы в реляционную.

Нельзя воспринимать концептуальное проектирование как дополнительную и излишнюю работу. Данную ошибку часто совершают начинающие проектировщики. На самом деле, концептуальное проектирование предоставляет несколько неоспоримых преимуществ даже при проектировании вручную небольших баз данных.

Скорее всего, без концептуального проектирования можно обойтись, если число таблиц не превышает десяти, но оно совершенно необходимо, если БД включает более сотни таблиц.

В последнем случае затруднительно ручное проектирование реляционной

схемы и требуется применение средств автоматизации проектирования.

Достоинства концептуальных моделей:

• Построение наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на ранних стадиях проектирования.

• Концептуальная модель (даже в виде вручную нарисованной диаграммы) является важной документацией, полезной не только на стадии проектирования БД, но и при ее дальнейшей эксплуатации, сопровождении и развитии.

• На рынке представлены CASE-системы, обеспечивающие автоматизированное преобразование концептуальных схем (диаграмм) в реляционные (язык SQL).

Автоматизация проектирования баз данных:

CASE-система (Computer-Aided Software Engineering) – система поддержки технологий автоматизированного проектирования, реализации и сопровождения сложных программных систем на всех этапах их жизненного цикла.

Ранние CASE-средства проектирования БД (начало 1990-х г.г.) в основном обеспечивали рисование диаграмм, проверку их формальной корректности и их долговременное хранение.

Подавляющее большинство современных CASE-систем обеспечивает автоматизированное преобразование диаграммных концептуальных схем баз данных в реляционные схемы, специфицированные чаще всего на языке SQL, на основе имеющейся четкой методики такого преобразования.

Как правило, на основе концептуальной схемы невозможно автоматически сгенерировать триггеры, хранимые процедуры, ограничения целостности общего вида, поэтому на завершающем этапе проектирования реляционной схемы требуется ручная работа проектировщика.

Автоматизация проектирования баз данных:

Как правило, CASE-средства, автоматизирующие преобразование концептуальной схемы БД в реляционную, производят реляционную схему базы данных в 3NF. Нормализация более высокого уровня усложняет программную реализацию и редко требуется на практике.

Вопрос: Если создатели семантической модели данных предоставляют язык (как правило, диаграммный), используя который проектировщики БД на основе исходной информации о предметной области могут сформировать концептуальную схему БД, то почему бы не реализовать программу, которая сама генерирует концептуальную схему БД в соответствующей семантической модели, используя исходную информацию о предметной области?

Существовали экспериментальные (исследовательские) интегрированные системы проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области и последующим преобразованием концептуальной схемы в реляционную схему.

Семантическая модель Entity Relationship:

Альтернативные названия: ER-модель, модель «Сущность-Связь», диаграммы Чена, реляционная информационная модель.

Предложена: 1976 г., Питер Чен (Peter Pin-Shan Chen. The Entity Relationship Model - Toward a Unified View of Data // ACM Transactions on Database Systems, Vol. 1, N 1, 1976, https://citforum.ru/database/classics/chen/)

Назначение: описание моделей предметных областей с целью последующего проектирования БД.

Стандарт: отсутствует.

Применяемая нотация: графическая (диаграммы), множество альтернативных нотаций.

Изучаемая нотация: применяется в CASE-системе Oracle.

39 ER-модель. Основные понятия. Представление на диаграммах сущностей, атрибутов и связей. Примеры. Уникальные идентификаторы типов сущностей. Нормальные формы ER-моделей.

Основные понятия ER-модели:

Сущность (тип сущности)– реальный или представляемый объект, информация о котором должна сохраняться и быть доступной.

Атрибут сущности– любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности.

Некоторые атрибуты могут помечаться как необязательные. Значения таких атрибутов не обязаны присутствовать во всех экземплярах данного типа сущности (допускается значение NULL).

ЧЕЛОВЕК: фио, например, Иванов И.И.; дата рождения, например, 30.06.1980; пол, например, М или Ж.

Основные понятия ER-модели:

Атрибут в реляционной модели: <имя атрибута, имя типа данных/домена>

В ER-модели указание типа атрибута не является обязательным.

Несмотря на то, что в настоящее время типовые возможности РСУБД в основном стандартизованы (на основе стандарта языка SQL), детали базового набора типов данных и средств определения доменов в разных СУБД могут различаться. Поскольку производители CASE-средств проектирования реляционных БД стремятся не связывать обеспечиваемые ими возможности семантического моделирования с конкретной реализацией СУБД, они стимулируют откладывание строгого определения типов атрибутов до стадии полного определения реляционной схемы и выбора целевой СУБД.

Пониманию предполагаемой сути типов данных или доменов способствует возможность указания примеров значений атрибутов или выбор имени атрибута, подсказывающего его тип или домен.

Основные понятия ER-модели:

Связь (тип связи)– графически изображаемая ассоциация, устанавливаемая между двумя сущностями.

Графическое отображение: ненаправленная линия.

В ER-модели допускаются только бинарные связи, то есть, соединяющие две сущности или сущность саму с собою (рекурсивная связь).

Конец связи называют ролью.

Характеристики роли: • Имя роли связи в данной сущности (указывается над линией связи вблизи соответствующего конца) • Степень роли (допустимое количество экземпляров соответствующей сущности в данной связи): - Один экземпляр – одноточечный вход - Много экземпляров – трехточечный вход • Обязательность роли: - Обязательная (каждый экземпляр данной сущности ДОЛЖЕН участвовать в связи) - Необязательная (каждый экземпляр данной сущности МОЖЕТ участвовать в связи)

Связь «Один-к-Одному»:

ЧЕЛОВЕК — ИМЕЕТ — ПАСПОРТ — ПРИНАДЛЕЖИТ

• Каждый человек имеет один и только один паспорт (здесь под человеком будем понимать обычных взрослых людей).

• Каждый паспорт может принадлежать только одному человеку (допускается наличие пустых не выписанных бланков паспортов).

Связь «Один-ко-многим»:

МУЖЧИНА — СЫН / ОТЕЦ

• Каждый мужчина является сыном одного и только одного мужчины (биологический отец обязан существовать).

• Каждый мужчина может являться отцом одного или более мужчин (может не иметь детей либо у него могут быть только дочери).

Связь «Многие-ко-Многим»:

АВТОР — НАПИСАЛ / НАПИСАНА — КНИГА

• Каждая книга может быть написана одним или более авторами (авторы книги могут быть неизвестны или не определены).

• Каждый автор принимал участие в написании одной или более книг (должен написать хотя бы одну книгу).

Нормальные формы ER-моделей:

Как и в случае схем реляционных баз данных, для ER-диаграмм вводится понятие нормальных форм, причем их смысл очень близок смыслу нормальных форм отношений.

В 1NF устраняются атрибуты, содержащие множественные значения.

Во 2NF устраняются атрибуты, зависящие только от части уникального идентификатора.

В 3NF устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор.

Нормальные формы ER-моделей: 1NF

пользуется услугами АЭРОДРОМ длина ВПП, число ангаров, . . . , самолеты производит ремонт АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ

• Сущность «Аэродром» помимо атрибутов, отражающих собственные характеристики, содержит атрибут, множественное значение которого характеризует самолеты, приписанные к этому аэродрому.

• Искажается смысл связи: авиаремонтные предприятия ремонтируют самолеты, а не аэродромы.

• Диаграмма не удовлетворяет требованию 1NF.

Нормальные формы ER-моделей: 1NF (после исправления)

обслуживает производит ремонт АЭРОДРОМ (длина ВПП, число ангаров, . . .) приписан АВИАРЕМОНТНОЕ ПРЕДПРИЯТИЕ обслуживается САМОЛЕТ

• Выделена сущность «Самолет».

• Связь между «Аэродром» и «Самолет»: к одному аэродрому приписывается несколько самолетов. • Связь «Самолет»–«Авиаремонтное предприятие»: предприятие обслуживает самолеты.

Нормальные формы ER-моделей: 2NF

Пример с элементом расписания: УИД элемента расписания – пара {, -}.

Однако не все атрибуты полностью зависят от этой пары.

FD:

$$\{\text{номер рейса, дата-время вылета}\} \to \text{бортовой номер самолета;}$$

номер рейса → аэропорт вылета;

номер рейса → аэропорт назначения;

бортовой номер самолета → тип самолета.

Так как некоторые атрибуты зависят от части УИД, диаграмма не в 2NF.

Решение: выделить «Рейс» (номер рейса, аэропорт вылета, аэропорт назначения) в отдельную сущность. Связь «Город» – «Рейс». В элементе расписания оставить дату-время вылета, бортовой номер самолета, тип самолета. Теперь нет атрибутов, зависящих только от части УИД.

Нормальные формы ER-моделей: 3NF

Каждый день каждый рейс выполняется одним самолетом. Бортовой номер самолета зависит от {дата-время вылета, рейс}, но он сам является уникальным идентификатором самолета, от которого зависит тип самолета.

FD:

{КОГДА, НА ЧЕМ, дата-время вылета} → бортовой номер самолета;

{КОГДА, НА ЧЕМ, дата-время вылета} → тип самолета;

бортовой номер самолета → тип самолета.

Выделяем «Самолет» отдельно.

Теперь диаграмма в 3NF.

40 Получение реляционной схемы из ER-диаграммы. Пошаговый алгоритм (без учета наследования и взаимно исключающих связей)

40.1 Получение реляционной схемы из ER-модели

1. **Преобразование сущностей:** Каждый простой тип сущности превращается в отношение. Имя сущности становится именем отношения. Экземплярам сущности соответствуют кортежи данного отношения.

2. **Преобразование атрибутов:** Каждый атрибут сущности становится атрибутом соответствующего отношения, при этом выбирается тип для представления соответствующих данных. Атрибуты, помеченные как необязательные, могут содержать неопределенные значения, обязательные атрибуты — не могут.

3. **Определение первичных ключей:** Компоненты уникального идентификатора сущности становятся первичным ключом отношения. Если имеется несколько возможных уникальных идентификаторов, для первичного ключа выбирается наиболее характерный УИД. Если в состав уникального идентификатора входят связи, к числу атрибутов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (копия первичного ключа соответствующего отношения). В этом случае при возникновении конфликта имен можно использовать имена концов связей и/или имена парных типов сущностей. Следует учитывать, что при неудачном выборе связей в качестве компонента УИД может произойти зацикливание.

A	a_1	a_2	$\cdots$	a_n
UIDA	UIDB	**A**	a_1	a_2
PRIMARY KEY	(a1, a2)			

B	b_1	b_2	$\cdots$	b_k
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
PRIMARY KEY	(b1)			

AB	a_1	a_2	b_1
PRIMARY KEY	(a1, a2, b1)		
FOREIGN KEY	(a1, a2)	REFERENCES	A(a1, a2)
FOREIGN KEY	(b1)	REFERENCES	B(b1)

4. **Представление подтипов и супертайпов:** Подтипы и супертайпы могут быть представлены в реляционной модели двумя способами:

[label=6.]**Общее отношение для всех подтипов (рекомендуется для иерархий с глубоким уровнем наследования)**

(a)

tc	a_1	$\cdots$	am	$b_{11} \cdots b_{1k1}$
B1	X	X	X	N N N N
Bn	X	X	X	X X X

Доступ к экземплярам подтипов:

PROJECT A$\{a_1, \ldots, a_m, b_{i1}, \ldots, b_{ik}\}$ (A WHERE tc = 'Bi')

Достоинства:

i. Соответствие логике супертипов
ii. Простой способ доступа к экземплярам супертипов и не слишком сложный — к экземплярам подтипов
iii. Сокращение количества отношений

Недостатки:

i. Единственное отношение — узкое место при многопользовательском доступе
ii. Усложнение логики приложений БД (необходимость анализа кода типа)
iii. Непроизводительный расход внешней памяти (хранение NULL)

(b) **Отдельное отношение для каждого подтипа**

B1	a_1	$\cdots$	am	$b_{11} \cdots b_{1k1}$
Bn	a_1	$\cdots$	am	$b_{n1} \cdots b_{nk_n}$

Воссоздание супертипа:

PROJECT B1$\{a_1, \ldots, a_m\}$ UNION ... UNION PROJECT Bn$\{a_1, \ldots, a_m\}$

Достоинства:

i. Более понятные правила работы с подтипами
ii. Упрощение логики приложений

Недостатки:

i. Увеличение количества отношений

ii. Усложнение доступа к экземплярам супертипа

iii. В общем случае невозможность модификации экземпляров супертипа

5. **Представление взаимно исключающих связей:**

[label=7.]**Преобразование модели со взаимно исключающими связями в модель с наследованием** (далее см. пункт 6). Для моделей, где связи «один-ко-многим» являются взаимоисключающими со стороны сущности со степенью роли «многие», возможно применение еще двух способов преобразования.

Общее хранение внешних ключей

(b)

A	...	rid
fk	...	X
Bi	Y	...

Достоинства:

- Добавление небольшого количества атрибутов для представления связей

Недостатки:

- Применим при условии, что все первичные ключи сущностей B определены на одном домене
- Сложная операция соединения: (A WHERE rid='Bi') JOIN Bi WHERE A.fk=Bi.pki

(c) **Раздельное хранение внешних ключей**

A	...	pk1
⋮	...	pkn
B1	X	Y
NULL	NULL	X
pk1	...	Y

Достоинства:

- Применим независимо от доменов первичных ключей в B
- Операция естественного соединения: `A NATURAL JOIN Bi`

Недостатки:

- Увеличение количества атрибутов для представления связей
- Непроизводительный расход внешней памяти (хранение NULL)

41 Наследование сущностей в ER-модели. Примеры. Отображение диаграммы с наследованием в реляционную схему

41.1 Наследование в ER-модели

Тип сущности (A) может быть расщеплен на несколько взаимно исключающих подтипов (B1, B2, ..., Bn). Тип сущности, на основе которого определяются

подтипы, называется супертипом. Подтипы наследуют атрибуты и связи супертипа и могут определять собственные атрибуты и/или связи. Простым типом сущности называется тип, не являющийся подтипом и не имеющий подтипов.

41.1.1 Правила наследования в ER-модели

1. Включение: $\forall b \in B_i \to b \in A, \quad i = 1, \ldots, n$
2. Отсутствие собственных экземпляров у супертипа: $\forall a \in A \to a \in B_i, \quad i = 1, \ldots, n$
3. Разъединенность подтипов: $\forall b \in B_i \to b \notin B_j, \quad i \neq j$

41.2 Пример ER-модели с наследованием

В супертипе “Печатное издание” определяются два атрибута (название издания и год издания) и обязательная связь “один-ко-многим” с типом сущности “Издательство”. Эти атрибуты и связь наследуются всеми подтипами этого супертипа.

41.2.1 Описание подтипов

Подтип “Книга” Дополнительно определяется атрибут ISBN (International Standard Book Number) и связь “многие-ко-многим” с авторами. В совокупности у данной сущности имеются три атрибута (два унаследованных от супертипа) и две связи (одна унаследованная).

Подтип “Журнал” Дополнительно определяются три атрибута: том, номер и ISSN (International Standard Serial Number). Аналогично, еще два атрибута и связь унаследованы от супертипа “Печатное издание”.

Поскольку супертип “Печатное издание” в ER-модели является абстрактным, для возможности описания печатных изданий, не являющихся книгами или журналами (брошюры, листовки и т.п.), определяется дополнительный подтип “Прочие”.

Подтипизация может продолжаться на более низких уровнях (например, можно определить подтипы сущности “Книга” или “Журнал”), но опыт использования ER-модели при проектировании баз данных показывает, что в большинстве случаев оказывается достаточно двух-трех уровней.

42 Взаимно исключающие связи в ER-модели. Примеры. Отображение диаграммы со взаимно исключающими связями в реляционную схему

42.1 Взаимно исключающие связи

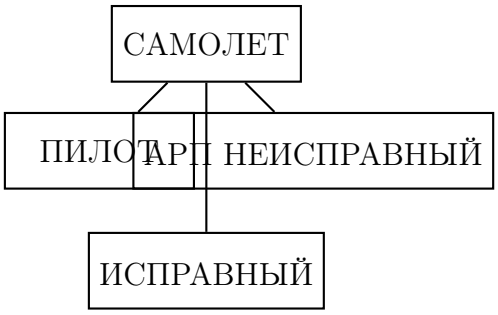
Взаимно исключающими связями называется такой набор связей одной сущности с другими, что для каждого экземпляра сущности может или должен существовать экземпляр только одной связи из данного набора.

42.1.1 Примеры взаимно исключающих связей

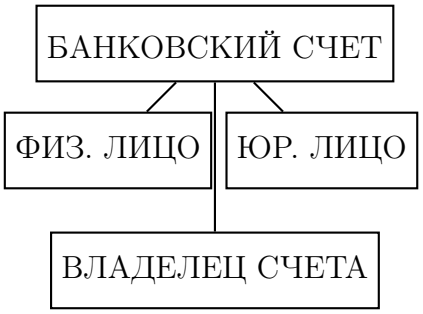
- **Самолет:** Самолет может быть исправным (тогда у него есть один пилот) или неисправным (тогда он ремонтируется авиаремонтным предприятием).
- **Банковский счет:** Банковский счет может принадлежать либо физическому, либо юридическому лицу.

42.2 Преобразования диаграмм со взаимно исключающими связями

1. **Введение подтипов**



2. **Введение общего супертипа**



42.3 Уникальные идентификаторы экземпляров сущностей

При определении сущности необходимо гарантировать, что каждый ее экземпляр является отличимым от любого другого экземпляра этой же сущности. Это достигается путем введения уникальных идентификаторов.

В ER-модели у экземпляра типа сущности не может быть назначаемого пользователем имени или назначаемого системой внешнего уникального идентификатора. Экземпляр сущности может идентифицироваться только своими индивидуальными характеристиками: значениями атрибутов и экземплярами связей.

Поэтому в качестве уникального идентификатора сущности проектировщик может выбрать (и сообщить об этом CASE-системе):

- Атрибут
- Комбинацию атрибутов
- Связь
- Комбинацию связей
- Комбинацию атрибутов и связей

42.3.1 Выбор уникального идентификатора экземпляров сущности «Человек»

ЧЕЛОВЕК	фио	дата рождения	пол	**ИМЕЕТ**
ПРИНАДЛЕЖИТ				
ПАСПОРТ	серия	номер	дата выдачи	кем выдан

- **Для БД небольших предприятий:** либо фио, либо фио + дата рождения.
- **Для БД крупных предприятий, городов, регионов:** связь с паспортом, тогда в качестве УИД человека будет использоваться УИД его паспорта (серия + номер).
- **Замечание:** связь с человеком не может использоваться в качестве УИД паспорта, поскольку могут существовать еще не выданные людям бланки паспортов.

42.3.2 Выбор уникального идентификатора экземпляров сущности «Книга»

- **Книжный склад:** прообразом типа сущности будет набор одноименных книг одного автора, вышедших в одном издательстве (уникально характеризуемый ISBN, который и является УИД). Дополнительно вводится атрибут, определяющий доступное количество экземпляров книг в этом наборе.
- **Библиотека:** требуется различать индивидуальные экземпляры книг (даже одинаковых), поэтому в каждой библиотеке экземплярам книг присваивается уникальный библиотечный номер, который и может использоваться в качестве УИД. ISBN в данном случае не уникален.

42.3.3 Пример выбора уникального идентификатора для сущности «Курс»

- Профессора обладают знаниями в нескольких учебных дисциплинах.
- Преподавание каждой дисциплины доступно нескольким профессорам.
- Каждый профессор может готовить курсы по любой доступной ему дисциплине.
- По каждой дисциплине может преподаваться несколько учебных курсов.
- Но каждый курс должен готовиться только одним профессором и быть посвящен только одной дисциплине.

Таким образом, каждый экземпляр сущности “Курс” уникально идентифицируется парой связей с именами концов “Готовится” и “Посвящен” или же уникальными идентификаторами экземпляров соответствующих сущностей “Профессор” и “Дисциплина”. Заметим, что сущности “Профессор” и “Дисциплина” связями не идентифицируются (соответствующие роли не являются уникальными).

43 Диаграммы классов языка UML. Основные понятия. Отображение классов, стереотипов, комментариев и ограничений на диаграммах. Примеры.

Диаграммой классов в UML называется диаграмма, на которой показан набор классов и некоторых других сущностей, а также связи между этими классами, и, возможно, комментарии и ограничения. Ограничения могут задаваться как на естественном языке, так и на языке объектных ограничений OCL.

43.1 Класс

Класс — это именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. Изображается в виде прямоугольника. У класса должно быть уникальное имя, которое рекомендуется сделать коротким, осмысленным и читаемым.

43.1.1 Атрибут

Атрибутом класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Атрибут является абстракцией состояния объекта. Класс может иметь любое число атрибутов. Любой атрибут должен иметь некоторое значение. Имена атрибута записываются под именем класса, и рекомендации для них такие же, как и для имени класса.

43.1.2 Операция

Операция класса — это именованная услуга, которую можно запросить у любого объекта этого класса. Класс может содержать любое число операций. Операции записываются под атрибутами и могут содержать сигнатуру и тип значения операции.

43.2 Связи

В диаграмме классов могут участвовать связи трех разных категорий: зависимости, обобщения и ассоциации.

43.2.1 Зависимость

Зависимость называется связью по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый. Изображается пунктирной линией со стрелкой, направленной к классу, от которого имеется зависимость.

Наиболее частое применение связей-зависимостей: при использовании стереотипа в качестве типа атрибута или в сигнатуре операции зависимого класса либо использование одного класса в сигнатуре операции другого класса.

43.2.2 Обобщение

Обобщением называется связь между общим классом (суперклассом) и более специализированной его разновидностью (подклассом). Подклассы могут использоваться везде, где могут использоваться суперклассы — это называется полиморфизм по включению. Связь обобщения изображается сплошной линией с большой незакрашенной стрелкой, направленной к суперклассу. Допускается множественное наследование, в котором есть ряд проблем, в том числе проблема именования атрибутов.

43.2.3 Ассоциация

Ассоциацией называется структурная связь между объектами одного класса и объектами другого или того же самого класса. В UML допускается создание n-арных ассоциаций, связывающих сразу несколько классов. Изображается как сплошная линия. Дополнительными параметрами ассоциации являются имя, роль каждого класса, участвующего в ассоциации, и кратность, показывающая, сколько объектов класса может участвовать в каждом экземпляре ассоциации.

Обычная ассоциация характеризует связь между равноправными классами. Если связь имеет вид часть-целое, то такая ассоциация называется агрегатной и изображается дополнительным ромбом на стороне класса-целого, закрашенным, если часть не может существовать без целого.

43.3 Стереотипы

Стереотип — механизм расширения семантики UML, позволяющий создавать новые элементы UML на основе существующих (например, классов) с учётом особенностей решаемой задачи.

43.4 Комментарии и ограничения

Комментарии и ограничения могут быть добавлены на диаграмму классов для уточнения дополнительных свойств или правил, которые должны соблюдаться при проектировании системы. Ограничения могут задаваться как на естественном языке, так и на языке объектных ограничений OCL.

44 Диаграммы классов языка UML. Категории связей и их отображение на диаграмме. Примеры.

Категории связей UML включают зависимости, обобщения и ассоциации.

44.1 Зависимости

Зависимостью называется связь по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый. Изображается пунктирной линией, направленной от зависимого класса.

Наиболее частое применение связей-зависимостей: использование стереотипа в качестве типа атрибута или в сигнатуре операции зависимого класса либо использование одного класса в сигнатуре операции другого класса.

При проектировании реляционных БД непонятно, как использовать информацию о наличии связей-зависимостей. При использовании диаграммы классов для генерации программного кода наличие связи-зависимости является сигналом для генерации директив `include` или `import` в исходном коде зависимого класса.

44.2 Обобщения в UML

Обобщением называется связь между общим классом (суперклассом) и более специализированной его разновидностью (подклассом). Класс-потомок наследует все атрибуты и операции класса-предка, но в нем могут быть определены дополнительные атрибуты и операции. Изображается сплошной линией с незакрашенным треугольником, направленным к суперклассу (родителю).

44.2.1 Правила наследования в ER-модели (см. предыдущую лекцию)

1. **Включение**: $\forall b \in B_i \to b \in A, \quad i = 1, \ldots, n$ — выполняется в UML.
2. **Отсутствие собственных экземпляров у супертипа**: $\forall a \in A \to a \in B_i, \quad i = 1, \ldots, n$ — не выполняется в UML по умолчанию (исключение: объявление суперкласса абстрактным, это внутреннее свойство класса, не отображаемое на диаграмме).
3. **Разъединенность подтипов**: $\forall b \in B_i \to b \notin B_j, \quad i \neq j$ — не выполняется в UML по умолчанию (исключение: ограничение обобщения {`disjoint`}).

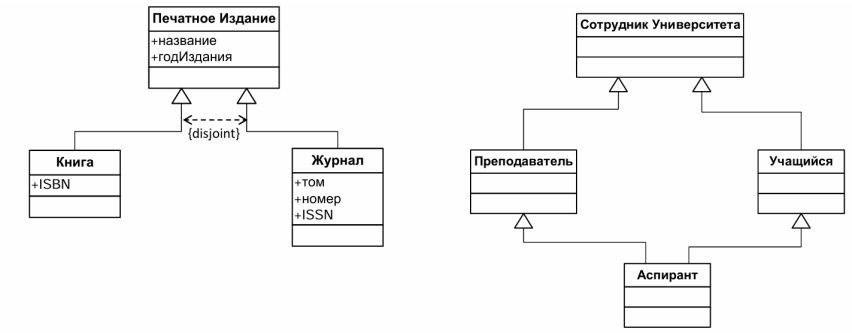


Рис. 1: 43.2.3

44.2.2 Пример с использованием связей обобщений UML

Класс “Прочие” в UML не требуется, так как суперклассы по умолчанию не являются абстрактными и могут иметь собственные экземпляры.

44.2.3 Проблемы использования множественного наследования

Одиночное наследование является достаточным в большинстве случаев применения связи-обобщения в диаграмме классов, предназначенной для проектирования реляционной БД.

Множественное наследование, помимо того, что не слишком часто требуется на практике, порождает ряд проблем, из которых одной из наиболее известных является проблема именования атрибутов и операций в подклассе, полученном путем множественного наследования.

- **Пример**: в классах “Учащийся” и “Преподаватель” может быть определён атрибут “номерКомнаты”, где значениями этого атрибута будут номера комнат в студенческом общежитии для учащихся и номера служебных кабинетов для преподавателей. Аспирант может проживать в общежитии и при этом иметь рабочий кабинет, предоставленный кафедрой.

44.3 Ассоциации в UML

Ассоциацией называется структурная связь между объектами одного класса и объектами другого или того же самого класса. В UML допускается создание n-арных ассоциаций, связывающих сразу несколько классов. Изображается сплошной линией (в общем случае ненаправленной).

44.4 Агрегатные ассоциации в UML

Агрегатные ассоциации используются для отображения отношений “часть-целое”, где класс “целое” имеет более высокий концептуальный уровень, чем часть. В агрегатных ассоциациях часть может одновременно принадлежать нескольким целым. Уничтожение целого не приводит к автоматическому уничтожению всех его частей.

44.5 Навигация между ассоциированными объектами

При наличии ассоциации предполагается возможность навигации между объектами, входящими в один экземпляр ассоциации. По умолчанию в UML навига-

Рис. 2: 43.3

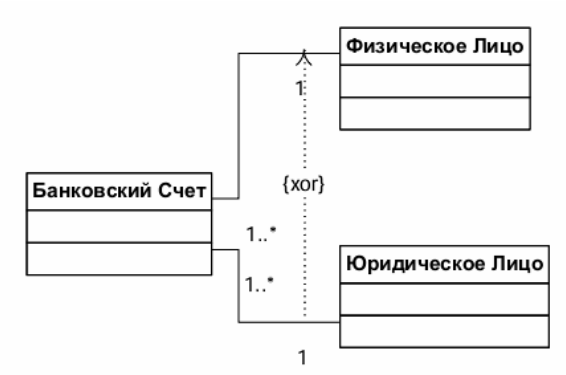


Рис. 3: 43.6

ция может проводиться в обоих направлениях. Для ограничения направления навигации на линии ассоциации ставится стрелка, указывающая требуемое направление.

44.6 Взаимно исключающие связи в UML

Для организации взаимно исключающих связей используется ограничение ассоциации {`xor`}.

45 Язык OCL. Инварианты классов. Основные типы данных и выражения OCL.

OCL — язык объектных ограничений, на котором могут быть написаны ограничения классов в диаграммах классов языка UML. Из UML в OCL заимствованы следующие понятия: класс, атрибут, операция, объект (экземпляр класса), ассоциация, тип данных, значение (экземпляр типа данных).

45.1 Инварианты класса

Под инвариантом класса в OCL понимается условие, которому должны удовлетворять все объекты данного класса при создании и в течение всего времени своего существования.

Синтаксис инварианта следующий:

$$\text{context} \ < \text{class\_name} > \ \text{inv:} \ < \text{OCL-выражение} >$$

Где:

- **<class_name>** — имя класса, для которого определяется инвариант.
- **inv** — ключевое слово, указывающее, что определяется инвариант.
- **<OCL-выражение>** — логическое выражение, которое должно быть истинным для всех объектов данного класса.

45.2 Типы данных OCL

45.2.1 Скалярные типы

- **Integer**
- **Real**
- **Boolean**
- **String**

45.2.2 Коллекции

- **Set** — неупорядоченная коллекция, не содержащая одинаковых элементов.
- **Bag** — неупорядоченная коллекция, которая может содержать одинаковые элементы.
- **Sequence** — упорядоченная коллекция, которая может содержать одинаковые элементы.
- **OrderedSet** — упорядоченная коллекция, не содержащая одинаковых элементов.

45.2.3 Объектные типы

Классы, определённые в UML.

45.3 Операции над значениями скалярных типов OCL

- **Boolean**: `and`, `or`, `xor`, `not`, `implies`, `if-then-else-endif`
- **Integer**: `*`, `+`, `-`, `/`, `abs()`, `div()`, `mod()`, `min()`, `max()`, операции сравнения
- **Real**: `*`, `+`, `-`, `/`, `abs()`, `floor()`, `round()`, `min()`, `max()`, операции сравнения
- **String**: `concat()`, `size()`, `substring()`, `toLower()`, `toUpper()`

45.4 Операции над объектами

- **Получение значения атрибута**: `<объект>.<имя_атрибута>`
- **Переход по экземпляру ассоциации**: `<объект>.<имя_роли>`
- **Вызов операции класса**: `<объект>.<имя_операции>(<список_фактических_параметров>`

45.5 Операции над коллекциями OCL

45.5.1 Конструкторы коллекций

- **select(<логическое выражение>)** — конструирует новую коллекцию, состоящую из тех элементов исходной коллекции, для которых результатом вычисления логического выражения является `true`.
- **reject(<логическое выражение>)** — конструирует новую коллекцию, состоящую из тех элементов исходной коллекции, для которых результатом вычисления логического выражения является `false`.
- **collect(<выражение>)** — конструирует новую коллекцию, состоящую из новых элементов, значения которых вычисляются путем применения заданного выражения к каждому элементу исходной коллекции.

45.5.2 Кванторы

- **exists(<логическое выражение>)** — возвращает `true`, если результат вычисления заданного логического выражения равен `true` хотя бы для одного из элементов коллекции.
- **forAll(<логическое выражение>)** — возвращает `true`, если результат вычисления заданного логического выражения равен `true` для всех элементов коллекции.

45.5.3 Теоретико-множественные операции

- **union(<коллекция>)** — конструирует новую коллекцию, включающую все элементы из исходной и заданной коллекций.
- **intersection(<коллекция>)** — конструирует новую коллекцию, включающую элементы, которые присутствуют одновременно как в исходной, так и в заданной коллекциях.
- **difference(<коллекция>)** — конструирует новую коллекцию, включающую те элементы исходной, которые отсутствуют в заданной коллекции.
- **symmetricDifference(<коллекция>)** — конструирует новую коллекцию, включающую элементы, которые присутствуют либо в исходной, либо в заданной коллекциях, но не в обеих из них.

45.5.4 Прочие операции

- **count(<элемент>)** — возвращает число вхождений заданного элемента в коллекцию.
- **includes(<элемент>)** — возвращает `true`, если коллекция содержит заданный элемент.
- **excludes(<элемент>)** — возвращает `true`, если коллекция не содержит заданный элемент.
- **includesAll(<коллекция>)** — возвращает `true`, если исходная коллекция содержит все элементы заданной коллекции.

- **excludesAll(<коллекция>)** — возвращает `true`, если коллекция не содержит ни одного элемента заданной коллекции.
- **size()** — возвращает число элементов в коллекции.
- **at(<индекс>)** — возвращает значение элемента в заданной позиции, применимо к `sequence` и `orderedSet`.
- **min()**, **max()**, **sum()** — для коллекций чисел определяются минимальное, максимальное значения и сумма значений всех элементов соответственно.

45.6 Инвариант класса

Инвариант класса — логическое выражение, при вычислении которого для любого объекта данного класса должно получаться значение `true` в течение всего времени существования этого объекта.

45.7 Пример инварианта класса

Класс Студент
Инвариант
`self.стипендия>=Университет.minСтипендия и self.стипендия<=Университет.maxСтипендия`

46 Получение реляционной схемы из диаграммы классов UML. Основные проблемы и рекомендации.

Если не обращать внимания на различия в терминологии, то здесь выполняются практически те же шаги, что и в случае преобразования в схему реляционной БД ER-диаграммы Поэтому ограничимся только некоторыми рекомендациями, специфичными для диаграмм классов.
Рекомендация 1.: Прежде чем определять в классах операции, подумайте, что вы будете делать с этими определениями в среде целевой РСУБД Если в этой среде поддерживаются хранимые процедуры, то, возможно, некоторые операции могут быть реализованы именно с помощью такого механизма Но если в среде РСУБД поддерживается механизм определяемых пользователями функций, возможно, он окажется более подходящим
Рекомендация 2.: Помните, что сравнительно эффективно в РСУБД реализуются только ассоциации видов «один ко многим» и «многие ко многим» Если в созданной диаграмме классов имеются ассоциации «один к одному», следует задуматься о целесообразности такого проектного решения Реализация в среде РСУБД ассоциаций с точно заданными кратностями ролей возможна, но требует определения дополнительных триггеров, выполнение которых понизит эффективность
Рекомендация 3.: В спецификации UML говорится о том, что, определяя однонаправленные связи, вы можете способствовать эффективности доступа к некоторым объектам Для технологии реляционных баз данных поддержка такого объявления вызовет дополнительные накладные расходы и тем самым снизит эффективность

Рекомендация 4.: Не злоупотребляйте возможностями OCL.
Диаграммы классов UML – это мощный инструмент для создания концептуальных схем баз данных, но, как известно, все хорошо в меру

47 Язык баз данных SQL. Возможности и структура языка SQL. Основные черты модели данных SQL и ее отличия от реляционной модели. Критика SQL.

Язык баз данных SQL Предложен: 1974 г., исследовательский проект IBM System R. Первоначальное название: `SEQUEL (Structured English QUEry Language)`. По юридическим соображениям в 1977 г. название языка было сокращено до `SQL (Structured Query Language)`. Еще одна интерпретация аббревиатуры `SQL`: `Standard Query Language`. Однако до сих пор многие произносят аббревиатуру `SQL` как "sequel". Стандарт: ANSI (первая версия, 1986 г.), ISO (`ISO-9075`)

Возможности языка SQL

- Формулирование запросов к БД
- Манипулирование данными (`CREATE`, `MODIFY`, `DELETE`)
- Определение и манипулирование схемой БД
- Определение ограничений целостности данных
- Определение представлений (виртуальных таблиц)
- Определение структур физического уровня, поддерживающих эффективное исполнение запросов
- Авторизация доступа к данным
- Управление транзакциями, сессиями, подключениями

Структура языка SQL С точки зрения разработчиков СУБД

- Базовый
- Промежуточный
- Полный

Базовый SQL – минимальное подмножество языка, реализация которого является обязательным условием соответствия стандарту. Промежуточный SQL – подмножество языка, реализация которого желательна в СУБД (именно этот уровень и поддерживается в большинстве реализаций). Полный SQL – реализация данного уровня является целью, к которой следует стремиться.

Критерий отнесения конструкции языка к определенному уровню – сложность ее реализации.

Структура языка SQL С точки зрения программиста приложений БД

- Прямой

- Встраиваемый
- Динамический

Прямой SQL – конструкции языка, которые можно использовать при прямом взаимодействии пользователя с СУБД в интерактивном режиме (например, через консольное приложение).

Встраиваемый SQL – включает конструкции, которые позволяют использовать возможности прямого SQL в программах, написанных на традиционных языках программирования.

Динамический SQL – добавляются конструкции, которые позволяют приложениям обращаться к СУБД с использованием конструкций прямого SQL, динамически образуемыми во время выполнения программы.

Структура языка SQL `SQL DDL DML DQL (условное DAL)`

Разделение на подъязыки:

- `Data Definition Language` – язык определения данных (определение схемы базы данных: таблиц, доменов, типов данных, ограничений целостности данных и т.п.)
- `Data Manipulation Language` – язык манипулирования данными (создание, модификация, удаление данных)
- `Data Query Language` – язык запросов (выборки данных)
- `Data Administration Language (DCL)` – язык администрирования или язык управления: определение прав доступа к данным, управление транзакциями, сессиями, подключениями

Подразделение на подъязыки действительно является условным: конструкции одного подъязыка могут включать конструкции другого.

Наиболее известные реализации SQL-ориентированных СУБД SQL-ориентированные коммерческие СУБД: 1979 г. `Oracle` 1981 г. `IBM SQL/DS` на коде System R (до конца 90-х) 1983 г. `IBM DB2` 1985 г. `INFORMIX SQL` (в 2001 г. приобретена IBM, продукт продолжает поддерживаться IBM) 1987 г. `Sybase SQL Server` (в 2010 г. компания Sybase приобретена SAP, SAP использует SQL Server в составе своих интегрированных решений, но не продает как отдельный продукт) 1989 г. `Microsoft SQL Server` (по технологии Sybase)

Бесплатные дистрибутивы коммерческих СУБД `Oracle XE (Express Edition)` `IBM DB2 Community Edition` `Microsoft SQL Server Express (ограничения функционала)`

Наиболее известные реализации SQL-ориентированных СУБД Наиболее распространенные реализации с открытым кодом: 1996 г. `PostgreSQL` (Berkley University, М. Стоунбрейкер, сообщество PostgreSQL)

- Лицензия `PostgreSQL` позволяет на его основе создавать различные, в том числе коммерческие, ответвления
- `Postgres Plus (EnterpriseDB)` – наиболее известный продукт с двойной лицензией (свободная без поддержки и коммерческая с официальной поддержкой)
- `Postgres Pro (Postgres Professional, Россия)` – сертификат ФСТЭК для использования в государственных организациях

1998 г. MySQL (М. Видениус, MySQL AB, с 2008 г. Sun Microsystems, с 2010 г. принадлежит Oracle) – двойная лицензия: GPL и коммерческая 2009 г. MariaDB (М. Видениус, MariaDB Foundation) – ответвление MySQL, лицензия GPL 2000 г. FirebirdSQL (Firebird Foundation) – Mozilla Public License

- 2007 г. Ред База Данных (Ред Софт, Россия), свободная и коммерческая лицензия, сертификат ФСТЭК

Наиболее известные реализации SQL-ориентированных СУБД Все вышеперечисленные СУБД относятся к универсальным клиент-серверным СУБД Из файл-серверных SQL-ориентированных СУБД следует выделить Informix Standard Engine (SE), которая в настоящее время не поддерживается, и OpenOffice/LibreOffi Base

Встраиваемые SQL-ориентированные СУБД:

- SQLite – написан на C, исходный код передан автором (Р. Хипп) в общественное достояние
- HSQLDB – написан на Java, входит в состав OpenOffice Base, лицензия BSD
- Firebird Embedded – движок FirebirdSQL в одной библиотеке, написан на C++, входит в состав LibreOffice Base, лицензия MPL
- Microsoft SQL Server Compact Edition (не выпускается с 2013 г.)

Стандартизация языка SQL

Поколение	Год	Стандарт & Изменения
1	1986	SQL-86 Первоначальная версия
	1989	SQL-89 Четкая стандартизация синтаксиса
2	1992	SQL-92 Манипулирование схемой, транзакц
	1995	SQL/CLI (Call-Level Interface)
	1996	SQL/PSM (Persistent Stored Modules)
3	1999	SQL:1999 OO расширения, регулярные выр
	2003	SQL:2003 Java, XML, поддержка OLAP
	2006	Переработано один том: поддержка языка
	2008	SQL:2008 Улучшена поддержка OLAP, XQ
	2011	SQL:2011 Улучшена поддержка темпоральн
	2016	SQL:2016 Работа с JSON, полиморфные таб
	2019	Добавлен новый том: поддержка многомер
	2023	SQL:2023 Тип данных JSON, работа с граф

Структура стандарта SQL (2003-2023)

- Том 1: Framework – Концептуальная структура стандарта
- Том 2: Foundation – Синтаксис, семантика, правила связывания для процедурных языков программирования
- Том 3: Call-Level Interface – Интерфейс уровня вызовов (основа SQL-ориентированных API)
- Том 4: Persistent Stored Modules – Описание языка SQL/PSM
- Том 9: Management of External Data – Языковые средства взаимодействия с внешними данными

- Том 10: Object Language Bindings – Правила связывания для объектно-ориентированных языков программирования
- Том 11: Information and Definition Schemas – Описание информационной схемы (хранение описателей данных)
- Том 13: SQL Routines and Types Using the Java Programming Language – Использование SQL совместно с языком программирования Java
- Том 14: XML-Related Specifications – Языковые средства работы с XML документами
- Том 15 (2019): Multi-dimensional Arrays – Поддержка работы с многомерными массивами
- Том 16 (2023): Property Graph Queries – Мэппинг графов в SQL и язык запросов для доступа к графам, хранящимся в таблицах SQL (основан на GQL)

Отличие модели данных SQL от классической реляционной модели SQL-ориентированная база данных представляет собой набор таблиц, каждая из которых в любой момент времени содержит некоторое мультимножество строк, соответствующих заголовку таблицы. Заголовок таблицы представляет собой список столбцов таблицы: поддерживается порядок столбцов, соответствующий порядку их определения при создании таблицы (при изменении таблицы новые столбцы добавляются всегда в конец заголовка). Во временных таблицах, порождаемых запросами SQL, допускаются безымянные столбцы, а также дублирование имен столбцов.

1. Отношение РМД – множество кортежей; таблица SQL – мультимножество строк:
   - В SQL допускаются дубликаты строк в таблицах
   - Ограничение первичного ключа для таблицы SQL не является обязательным
2. В РМД заголовок отношения – множество атрибутов; в SQL заголовок таблицы – список столбцов
3. В РМД `NULL` не является допустимым значением для атрибутов, составляющих первичный или возможный ключ отношения; в SQL это правило действует только для первичных ключей
4. В РМД существует единственный способ сопоставления внешнего ключа с первичным/возможным; в SQL таких способов три, только один из которых соответствует РМД
5. Разная терминология (**РМД** – отношение, атрибут, кортеж; **SQL** – таблица, столбец, строка); таблица SQL – это вовсе не отношение, хотя во многом они похожи

Критика языка SQL

1. Противоречие принципам реляционной модели данных

2. Сложность языка (противоречие первоначальной идее авторов сделать язык доступным для понимания не квалифицированными пользователями)

3. Чрезмерная избыточность языка (например, имеется несколько способов выразить один и тот же запрос)

4. Отступления от стандарта в реализациях (многочисленные диалекты SQL, особенно это касается языка SQL/PSM)

5. Трехзначная логика (`False`, `True`, `Unknown`)

6. Неоднозначность использования и интерпретации `NULL`

Значение `NULL` в SQL

1. Означает "отсутствие значения", но на самом деле используется в трех разных по смыслу ситуациях (т.е. требуется три различных `NULL`, но это приведет к усложнению логики):

   - значение неизвестно;
   - значение временно не определено;
   - значение недопустимо в данном контексте.

2. Для логического типа `NULL` и `UNKNOWN` трактуются как одно и то же значение, хотя `NULL` — это "отсутствие значения", а `UNKNOWN` — вполне определенное логическое значение.

3. По правилам SQL сравнение значения с `NULL` дает в результате `UNKNOWN` (`a comp_op NULL = NULL comp_op a = NULL comp_op NULL = UNKNOWN`), но для операции равенства это не всегда так:

   - Равенство как операция сравнения (`a = NULL`) = (`NULL = a`) = (`NULL = NULL`) = `UNKNOWN`;
   - Поиск дубликатов: (`NULL = NULL`) = `TRUE`, (`a = NULL`) = (`NULL = a`) = `FALSE`.

48 Основные типы данных языка SQL (без учета объектных расширений). Преобразования типов данных.

Типы данных SQL

- Данные, хранящиеся в таблицах SQL, являются типизированными.
- СУБД должна отслеживать, чтобы в каждом столбце каждой строки таблицы присутствовали только допустимые для соответствующих типов данных значения.
- `NULL` является допустимым значением для любого типа данных SQL.

Булевский тип: BOOLEAN Литералы: FALSE, TRUE, UNKNOWN SQL:2003: “В этой спецификации не проводится различие между NULL значением булевского типа данных и истинностным значением UNKNOWN, являющимся результатом вычисления предиката, условия поиска или булевского выражения. Они могут использоваться взаимозаменяемо и означают в точности одно и то же”.

Точные числовые типы Целые числа: SMALLINT, INTEGER, BIGINT Точность представления определяется в реализации, стандарт лишь определяет, что точность SMALLINT $\leq$ INTEGER $\leq$ BIGINT.

Литералы: десятичные целые числа со знаками ‘+’ или ‘-’, отсутствие знака – ‘+’.

Числа с фиксированной точкой: NUMERIC(p, s), DECIMAL(p, s) p – точность (число сохраняемых десятичных цифр, включая дробную часть): максимально допустимая для NUMERIC, минимальная для DECIMAL (пример: литерал 123.4 недопустим для NUMERIC(3,1), но допустим для DECIMAL(3,1)). s – масштаб (число десятичных цифр в дробной части). Допустимые значения p, s определяются в реализации (p $\geq$ s). Значения по умолчанию: s=0, p – определяется в реализации.

Литералы: целые или вещественные числа в формате с фиксированной точкой (точка отделяет целую часть от дробной), со знаком или без. В большинстве реализаций NUMERIC и DECIMAL трактуются как один и тот же тип.

Приближенные числовые типы REAL, DOUBLE PRECISION, FLOAT(p) p – число бит, требуемых для хранения мантиссы (точность представления). Точность представления для первых двух типов и максимальное значение p определяется в реализации. Литералы: целые числа, или вещественные числа в формате с фиксированной точкой, или вещественные числа в формате с плавающей точкой: xEy, где x – мантисса, y – десятичный порядок в виде целого числа ($x \times 10^y$). Арифметика для приближенных числовых типов работает быстрее, чем для чисел с фиксированной точкой, но возможна потеря точности за счет погрешности вычислений. Числа с фиксированной точностью надо использовать там, где требуется соблюдать точность вычисления (финансы).

Типы символьных и битовых строк CHAR(x), VARCHAR(x), CLOB(z) x, z – количество символов в строке: фиксированное для CHAR (допустимо записывать литералы меньшей длины, которые будут дополняться справа пробелами), максимально допустимое для VARCHAR и CLOB. Допустимые значения x, z определяются в реализации (z значительно больше x), z может указываться в виде nK, nM, nG. Вид литералов: ‘abCdef’ или ‘‘FbcDE’’. Набор допустимых символов определяется в реализации (ASCII как минимум).

BIT(x), BIT VARYING(x), BLOB(z) В отличие от строк состоят из произвольных байтов, не обязательно кодирующих символы. Вид литералов: B’01001101101’ или X’79CA83’.

Типы даты, времени DATE, TIME(p), TIMESTAMP(p), TIME WITH TIMEZONE(p), TIMESTAMP WITH TIMEZONE(p) p – точность представления долей секунды, максимальное значение определяется в реализации (p $\geq$ 6), значение по умолчанию p=0 для TIME, p=6 для TIMESTAMP. Допустимое значение секунд варьируется от 00 до 61.

Форматы литералов: DATE ’yyyy-mm-dd’, TIME ’hh:mm:ss:ff...f’, TIMESTAMP ’yyyy-mm-dd hh:mm:ss:ff...f’. Временная зона задается в виде +hh:mm или -hh:mm.

Функции, возвращающие текущие дату и время: CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP.

Типы временных интервалов INTERVAL start(p) [TO end(q)] start, end: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND. Поддерживаемые интервалы:

- YEAR, YEAR TO MONTH, MONTH
- DAY, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND
- HOUR, HOUR TO MINUTE, HOUR TO SECOND
- MINUTE, MINUTE TO SECOND, SECOND

p – точность представления лидирующего поля, q – точность долей секунды (применяется, если end=SECOND), максимальные значения определяются в реализации, значения по умолчанию p=2, q=6. Формат литералов: INTERVAL '1:35' HOUR TO MINUTE.

Преобразование типов данных в SQL

- Поддерживается явное и неявное преобразование значений одного типа данных к другому.
- Правила неявного преобразования не сильно отличаются от тех, что существуют в языках программирования.
- Точный числовой тип приводится к другому точному числовому типу с большей точностью или к приближенному числовому типу.
- Приближенный числовой тип приводится к другому приближенному числовому типу с большей точностью.
- Строковый тип фиксированной длины приводится к другому строковому типу фиксированной или переменной длины с большим допустимым количеством символов.
- Строковый тип переменной длины приводится к другому строковому типу переменной длины с большим допустимым количеством символов.
- Явное преобразование осуществляется с помощью оператора CAST(expression AS datatype).
- Подробные правила выполнения оператора CAST см. в учебной литературе.

49 Средства работы с доменами в SQL.

Домены в SQL
Домен – базовый тип данных + ограничение области определения значений
Домен является долговременно хранимым именованным объектом схемы БД
Оператор определения домена: CREATE DOMAIN domain_name AS datatype [DEFAULT value] [constraint_list] DEFAULT value – определение значения по умолчанию (литерал базового типа, либо значение NULL, либо вызов предопределенной функции, возвращающей литеральное значение базового типа) constraint_list – список ограничений области определения значений, все ограничения в списке связываются логической функцией AND

Пример:

```
CREATE DOMAIN SALARY AS NUMERIC(10, 2)
```

DEFAULT 10000.00
CHECK(VALUE BETWEEN 10000.00 AND 10000000.00)
CONSTRAINT SAL_NOT_NULL CHECK(VALUE IS NOT NULL);

Домены в SQL
Оператор изменения определения домена:
ALTER DOMAIN domain_name action Допустимы 4 действия:
SET DEFAULT value, DROP DEFAULT, ADD CONSTRAINT [constraint_name] constraint_expressi
DROP CONSTRAINT constraint_name Если к моменту исполнения ALTER DOMAIN ADD CONSTRAINT существуют столбцы таблиц, определенные на данном домене, текущие значения которых противоречат новому ограничению, то СУБД должна отвергнуть этот оператор. Новое ограничение связывается с существующими ограничениями домена логической функцией AND.

Пример: ALTER DOMAIN SALARY SET DEFAULT 20000.00; ALTER DOMAIN SALARY DROP CONSTRAINT SAL_NOT_NULL;

Домены в SQL Оператор отмены определения домена: DROP DOMAIN domain_name { RESTRICT | CASCADE } DROP DOMAIN RESTRICT отвергается, если домен использован в определении некоторого столбца таблицы (базовой или виртуальной) или в определении ограничения целостности. DROP DOMAIN CASCADE выполняется всегда по следующим правилам:

- Уничтожаются все ограничения целостности и виртуальные таблицы, в определениях которых задействован данный домен
- Столбцы базовых таблиц, определенные на данном домене, преобразуются к его базовому типу и наследуют значение по умолчанию и все ограничения уничтожаемого домена

50 Средства определения, изменения и отмены определения базовых таблиц в SQL

Таблицы в SQL

- Базовые – реально хранимые в БД таблицы (подразделяются на традиционные и типизированные)
- Порождаемые – таблицы, формируемые и существующие во время выполнения запросов
- Представления (виртуальные) – таблицы, существование которых поддерживается алгоритмически

Оператор определения базовой (традиционной) таблицы:

```
CREATE TABLE table_name (
    column_list [constraint_list]
)
column_definition ::= column_name { datatype | domain } [
   DEFAULT value] [constraint_list]
```

Данный оператор создает соответствующие описатели в схеме БД и выделяет область во внешней памяти для хранения данных.

Изменение определения базовой таблицы

```
ALTER TABLE table_name action
```

Возможны следующие действия:

- `ADD COLUMN column_name { datatype | domain } [DEFAULT value] [constraint_list]`
- `ALTER COLUMN column_name SET DEFAULT value`
- `ALTER COLUMN column_name DROP DEFAULT`
- `DROP COLUMN column_name { RESTRICT | CASCADE }`
- `ADD CONSTRAINT [constraint_name] constraint_expression`
- `DROP CONSTRAINT constraint_name [ { RESTRICT | CASCADE } ]`

Отмена определения базовой таблицы

```
DROP TABLE table_name { RESTRICT | CASCADE }
```

`DROP TABLE RESTRICT` отвергается, если таблица задействована в определении представлений или ограничений целостности (не считая собственных табличных ограничений, не ссылающихся на другие базовые таблицы).

`DROP TABLE CASCADE` выполняется всегда по следующим правилам:

- Уничтожаются все ограничения целостности и представления, в определениях которых задействована данная таблица.
- Уничтожаются все строки, хранящиеся в данной таблице, а также определения ее столбцов и табличных ограничений (таблица перестает существовать).

51 Иерархия ограничений в SQL. Средства определения и отмены общих ограничений (ограничений БД).

Иерархия ограничений целостности языка SQL:

- Ограничения БД
- Ограничения таблиц
- Ограничения столбцов
- Ограничения доменов

Дополнительные ограничения базы данных – общие ограничения целостности

Определение:

```
CREATE ASSERTION constraint_name
CHECK( constraint_expression )
```

Отмена:

```
DROP ASSERTION constraint_name
```

Пример:

```
CREATE ASSERTION DEPT_MNG_CONSTR
CHECK( NOT EXISTS(
    SELECT * FROM EMPLOYEE EMP, DEPARTMENT DEPT
    WHERE EMP.EMP_ID = DEPT.DEPT_MANAGER
      AND CURRENT_DATE - EMP.EMP_BDATE < INTERVAL '30' YEAR
));
```

52 Базовые средства манипулирования данными в языке SQL

Базовые средства манипулирования данными

- Средства манипулирования данными присутствуют в прямом, встраиваемом и динамическом SQL
- К базовым средствам манипулирования данными относятся:
  - Оператор вставки строк (`INSERT`)
  - Оператор модификации строк (`UPDATE`)
  - Оператор удаления строк (`DELETE`)

Оператор вставки строк

```
INSERT INTO table_name [(column_list)] VALUES
   row_constructor_list
```

- Если список столбцов пропущен, то предполагается, что используется список из имен всех столбцов таблицы, указанных в том порядке, в котором они были описаны в операторе `CREATE TABLE`
- В списке столбцов должны быть обязательно указаны те столбцы, для которых не задано значение по умолчанию (явно или неявно)
- Порядок перечисления значений столбцов в конструкторе строки должен совпадать с порядком перечисления столбцов в соответствующем списке
- В качестве значений столбцов могут использоваться литералы, `DEFAULT`, `NULL`, вызовы предопределенных функций, возвращающих литеральное значение соответствующего типа, а также скалярные запросы, результат выполнения которых состоит из единственной строки, включающей единственный столбец

Пример:

```
INSERT INTO EMPLOYEE VALUES
ROW( 319, 'Ivanov I.I.', '1980-07-19', DEFAULT, NULL, NULL ),
ROW( 320, 'Petrov P.P.', '1983-05-09', (SELECT EMP_SALARY FROM
    EMPLOYEE WHERE EMP_ID = 200), 12, NULL );
```

```
INSERT INTO EMPLOYEE( EMP_ID, EMP_NAME, EMP_BDATE ) VALUES
ROW(321, 'Sidorov S.S.', '1962-12-10' );
```

Оператор вставки строк – вставка результата запроса Определим таблицу, хранящую информацию о временных сотрудниках:

```
CREATE TABLE EMPLOYEE_TEMP (
    EMPT_ID INTEGER PRIMARY KEY,
    EMPT_NAME VARCHAR(200) NOT NULL,
    EMPT_BDATE DATE NOT NULL,
    EMPT_ENTRY_DATE DATE NOT NULL
);
```

Переведем временных сотрудников, чей испытательный срок превысил 3 месяца, в основной штат:

```
INSERT INTO EMPLOYEE( EMP_ID, EMP_NAME, EMP_BDATE )
( SELECT EMPT_ID, EMPT_NAME, EMPT_BDATE FROM EMPLOYEE_TEMP
  WHERE CURRENT_DATE - EMPT_ENTRY_DATE >= INTERVAL '3' MONTH )
    ;
```

Оператор модификации строк

```
UPDATE table_name SET column_value_assignment_list WHERE
   logical_expression
column_value_assignment := column_name = value_expression
```

- Для всех строк таблицы с указанным именем вычисляется логическое условие. Строки, для которых значением этого условия является `TRUE`, считаются подлежащими модификации (`S_m` – множество модифицируемых строк)
- Каждая строка `s` `S_m` подвергается модификации таким образом, что значение каждого столбца этой строки, указанного в списке модификации, заменяется значением, указанным в правой части соответствующего элемента списка. Если новое значение задается оператором запроса, в который входят имена модифицируемых столбцов, то под значениями этих столбцов в запросе понимаются их значения до модификации

Пример. Перевести всех служащих, выполняющих проект с номером 5, в отдел 12 и повысить им заработную плату на 5000 руб.

```
UPDATE EMPLOYEE SET DEPT_ID = 12, EMP_SALARY = EMP_SALARY +
   5000.00 WHERE PRO_ID = 5;
```

Оператор удаления строк

```
DELETE FROM table_name WHERE logical_expression
```

- Для всех строк таблицы с указанным именем вычисляется логическое условие. Строки, для которых значением этого условия является `TRUE`, считаются подлежащими удалению (`S_d` – множество удаляемых строк)
- Каждая строка `s` `S_d` удаляется из указанной таблицы.

Пример. Уволить всех служащих, размер заработной платы которых превышает размер заработной платы начальников их отделов.

```
DELETE FROM EMPLOYEE WHERE EMP_SALARY >
(SELECT EMP.EMP_SALARY
 FROM EMPLOYEE EMP, DEPARTMENT DEPT
 WHERE EMPLOYEE.DEPT_ID = DEPT.DEPT_ID
   AND DEPT.DEPT_MANAGER = EMP.EMP_ID);
```

Учебный пример

```
CREATE TABLE EMPLOYEE (
    ...
    DEPT_ID INTEGER DEFAULT NULL REFERENCES DEPARTMENT ON
       DELETE SET NULL,
    ...
);
CREATE TABLE DEPARTMENT (
    DEPT_ID INTEGER PRIMARY KEY,
    DEPT_EMP_NUM INTEGER NOT NULL CHECK ( DEPT_EMP_NUM = (
       SELECT COUNT(*) FROM EMPLOYEE WHERE DEPT_ID = EMPLOYEE.
       DEPT_ID )),
    CHECK( VALUE <= 100 ),
    ...
);
```

Операторы модификации таблицы `EMPLOYEE` вида:

```
INSERT INTO EMPLOYEE ( ..., DEPT_ID, ...) VALUES ROW ( ..., X,
    ...);
UPDATE EMPLOYEE SET DEPT_ID = X WHERE ...;
DELETE FROM EMPLOYEE WHERE ...;
```

где `X` – значение `DEPT_ID`, отличное от `NULL`, а во множество удаляемых строк включается хотя бы одна строка, в которой значение столбца `DEPT_ID` отличается от `NULL`, нарушают выделенное ограничение целостности.

53 Понятие триггера. Механизм триггеров в SQL. Типы триггеров и их выполнение.

Триггеры в SQL
Триггер – хранимая в БД процедура, автоматически вызываемая СУБД при возникновении определенных условий. В языке обеспечиваются возможности определения триггеров, которые вызываются при модификации указанной базовой таблицы (определение триггеров над представлениями не допускается).

Предметная таблица – базовая таблица, с которой связывается определение триггера.
Инициирующий оператор – оператор SQL, выполнение которого приводит к срабатыванию триггера.

Основные области применения триггеров:

- Согласование и очистка данных
- Журнализация и аудит
- Операции, не связанные с изменением БД (генерация отчетов, печать документов, рассылка почты)

53.1 Определение триггера

Оператор `CREATE TRIGGER` (отмена – `DROP TRIGGER`). В операторе указываются:

1. Имя триггера

2. Момент срабатывания: `BEFORE` или `AFTER`
3. Тип инициирующего оператора: `INSERT`, `UPDATE [OF column_list]` или `DELETE`
4. Имя предметной таблицы: `ON table_name [REFERENCING OLD {ROW | TABLE} AS name NEW {ROW | TABLE} AS name]`
5. Количество срабатываний триггера: `FOR EACH ROW` или `FOR EACH STATEMENT`
6. Дополнительное условие применимости триггера: `WHEN (logical_expression)`
7. Тело триггера: одиночный оператор или процедура на языке `SQL/PSM`

53.2 Триггеры для учебного примера

```
CREATE TRIGGER CHANGE_DEPT_I AFTER INSERT ON EMPLOYEE FOR EACH
    ROW
WHEN( EMPLOYEE.DEPT_ID IS NOT NULL )
UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM + 1 WHERE
   DEPARTMENT.DEPT_ID = EMPLOYEE.DEPT_ID;
```

```
CREATE TRIGGER CHANGE_DEPT_D AFTER DELETE ON EMPLOYEE FOR EACH
    ROW
WHEN( EMPLOYEE.DEPT_ID IS NOT NULL )
UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM - 1 WHERE
   DEPARTMENT.DEPT_ID = EMPLOYEE.DEPT_ID;
```

```
CREATE TRIGGER CHANGE_DEPT_U AFTER UPDATE OF DEPT_ID ON
   EMPLOYEE
REFERENCING OLD ROW AS OLD_EMP NEW ROW AS NEW_EMP FOR EACH ROW
BEGIN ATOMIC
    UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM - 1
       WHERE DEPARTMENT.DEPT_ID = OLD_EMP.DEPT_ID;
    UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM + 1
       WHERE DEPARTMENT.DEPT_ID = NEW_EMP.DEPT_ID;
END;
```

53.3 Выполнение триггеров

При выполнении каждого триггера система устанавливает контекст выполнения триггера. Контекст выполнения триггера всегда является атомарным. Контекст выполнения триггера включает:

- триггерное событие (`INSERT`, `UPDATE` или `DELETE`)
- имя предметной таблицы триггера
- имена столбцов предметной таблицы, специфицированных в определении триггера (для триггеров по `UPDATE`)
- набор переходов (представление всех вставляемых, модифицируемых или удаляемых строк, а также список уже выполненных триггеров и представлений строк, над которыми эти триггеры выполнялись)

Отслеживание уже выполненных триггеров ведется для предотвращения зацикливания выполнения системы триггеров.

54 Общая структура оператора выборки в SQL и схема его выполнения.

Оператор выборки данных в SQL

```
SELECT [ ALL | DISTINCT ] select_item_list
FROM table_reference_list
[ WHERE logical_expression ]
[ GROUP BY column_name_list ]
[ HAVING logical_expression ]
[ ORDER BY order_item_list ]
```

Выполнение запроса состоит из нескольких шагов, соответствующих разделам оператора выборки. СУБД обычно не придерживаются данной схемы выполнения, но результат должен получиться таким, как если бы он получался при точном следовании данной схеме.

54.1 Семантика оператора выборки

Шаг 1: `FROM table_reference_list` Выполняется операция расширенного декартова произведения таблиц, указанных в списке `FROM`: `T = T1 TIMES T2 TIMES ... TIMES Tn` Аналогом операции переименования `RENAME` алгебры Кодда в SQL служат псевдонимы таблиц (пример: `EMPLOYEE E`) и квалифицированные имена столбцов (`E.EMP_BDATE`).

Шаг 2: `[ WHERE logical_expression ]` Выполняется операция ограничения таблицы `T`, сформированной на предыдущем шаге, по заданному логическому условию: `T1 = T WHERE logical_expression` `T1` содержит только те строки таблицы `T`, для которых результатом вычисления логического выражения является `TRUE`. Отсутствие `WHERE` означает `WHERE TRUE` (`T1` $\equiv$ `T`).

Шаг 3: `[ GROUP BY column_name_list ]` На основе таблицы `T1`, полученной на предыдущем шаге, формируется сгруппированная таблица `T2`.

Шаг 4: `[ HAVING logical_expression ]` Строится таблица `T3`, содержащая только те группы строк таблицы `T2`, для которых результатом вычисления логического выражения является `TRUE`. Отсутствие `HAVING` означает `HAVING TRUE` (`T3` $\equiv$ `T2`).

Шаг 5: `SELECT [ ALL | DISTINCT ] select_item_list` Результирующая таблица `T4` содержит столько строк, сколько `T1` (без группировки) или сколько групп `T3` (с группировкой). `DISTINCT` – на завершающей стадии из `T4` удаляются строки-дубликаты.

Шаг 6: `[ ORDER BY order_item_list ]` На завершающей стадии выполнения выборки данных производится сортировка строк результирующей таблицы. `ORDER BY` не приводит к появлению таблицы, а к появлению упорядоченного списка строк.

Конструкция `TABLE table_name` является сокращенной формой `SELECT * FROM table_name`.

54.2 Ссылки на таблицы раздела FROM

- Имя базовой таблицы (CREATE TABLE)
- Имя представления (CREATE VIEW)
- Порождаемая таблица (выражение запроса в круглых скобках)
- Имя запроса, присоединенного с помощью WITH

54.3 Представления (виртуальные таблицы)

Определение представления: CREATE VIEW view_name [column_name_list] AS query_expression Явное указание имен столбцов представляемой таблицы требуется, если их невозможно вывести из выражения запроса.

```
CREATE VIEW DEPARTMENT_MANAGER AS
SELECT E.EMP_ID, E.EMP_NAME, E.EMP_BDATE, E.EMP_SALARY, D.
   DEPT_ID, D.DEPT_NAME
FROM EMPLOYEE E, DEPARTMENT D WHERE E.EMP_ID = D.DEPT_MANAGER;
```

Отмена представления: DROP VIEW view_name { RESTRICT | CASCADE }

Из представлений можно выполнять SELECT. В общем случае они не модифицируемы (без рассмотрения WITH CHECK OPTION).

55 Представляемые и порождаемые таблицы в SQL. Агрегатные и кванторные функции.

Порождаемые таблицы и присоединенные запросы
Семантически порождаемые таблицы соответствуют временным представлениям, которые существуют лишь в момент исполнения запроса. Явное указание псевдонима порождаемой таблицы и имен ее столбцов требуется в том случае, когда эти имена невозможно вывести из соответствующего выражения запроса Пример (Найти общее число служащих и максимальный размер зарплаты в отделах с одинаковым максимальным размером зарплаты.):

```
WITH DEPT_MAX_SAL(MAX_SAL, TOTAL_EMP) AS
(
    SELECT MAX(EMP_SALARY), COUNT(*)
    FROM EMPLOYEE
    WHERE DEPT_ID IS NOT NULL
    GROUP BY DEPT_ID
)
SELECT SUM(TOTAL_EMP), MAX_SAL
FROM DEPT_MAX_SAL
GROUP BY MAX_SAL;
```

Агрегатные и кванторные функции
Function([DISTINCT | ALL] value_expression)
Исходное мультимножество строк – вся таблица или группа строк в случае сгруппированной таблицы. На основании аргумента-выражения производится

мультимножество значений путем вычисления данного выражения для каждой строки в мультимножестве. Из мультимножества значений удаляются NULL, а при наличии DISTINCT – и дубликаты. Затем производится вычисление.

- `COUNT` - число строк (если аргумент *) или значений, 0 для пустого множества,
- `MAX` –максимальное значение, NULL для пустого множества,
- `MIN` –минимальное значение, NULL для пустого множества,
- `AVG` –среднее значение, NULL для пустого множества,
- `SUM` –суммарное значение, NULL для пустого множества,
- `EVERY` – квантор всеобщности (TRUE, если вычисление аргумента равно TRUE для каждой строки исходного мультимножества, TRUE для пустого множества),
- `SOME/ANY` –квантор существования (TRUE, если вычисление аргумента равно TRUE хотя бы для одной строки исходного мультимножества, FALSE для пустого множества).
- Важный частный случай: `COUNT(*)` – подсчет строк в мультимножестве (при этом все строки считаются различными)

56 Предикаты в логических выражениях языка SQL

Логические выражения в языке SQL
Синтаксически логическое выражение SQL определяется как булевское выражение, которое строится на основе предикатов с использованием логических операций `AND`, `OR` и `NOT`, а также круглых скобок.
В дальнейшем будем использовать следующие обозначения:

- s – скалярная величина
- R – строковое значение
- $|R|$– степень строки (количество скалярных значений в ней)
- T– табличное значение
- $|T|$ - количество строк в таблице

Для предикатов выполняются следующие правила:

1. Совместимость типов операндов
2. Равенство степеней строк-операндов ($|RX| = |RY|$ или $|RX| = |RT|$, $RT \in T$)
3. Существование отрицательной формы предиката: `NOT pred = NOT(pred)` – имеются исключения из данного правила

56.1 Предикат сравнения и предикат BETWEEN

Предикат сравнения:
s_X op s_Y или R_X op R_Y
op ::= = | <> | < | > | <= | >=
NULL op s = UNKNOWN, s op NULL = UNKNOWN, NULL op NULL = UNKNOWN

Предикат BETWEEN (проверка вхождения в диапазон значений):
s_X BETWEEN s_Y AND s_Z или R_X BETWEEN R_Y AND R_Z, $|R_X| = |R_Y| = |R_Z|$
Эквивалентные формы (выражение через предикаты сравнения):
$s_X \geq s_Y$ AND $s_X \leq s_Z$ или $R_X \geq R_Y$ AND $R_X \leq R_Z$

Пример:
Найти номера, имена и размер зарплаты служащих, получающих зарплату, размер которой не меньше средней зарплаты служащих своего отдела и не больше зарплаты начальника отдела:

```
SELECT E1.EMP_ID, E1.EMP_NAME, E1.EMP_SALARY
FROM EMPLOYEE E1
WHERE E1.EMP_SALARY BETWEEN
    (SELECT AVG(E2.EMP_SALARY)
     FROM EMPLOYEE E2
     WHERE E2.DEPT_ID = E1.DEPT_ID)
    AND
    (SELECT MNG.EMP_SALARY
     FROM DEPARTMENT_MANAGER MNG
     WHERE MNG.DEPT_ID = E1.DEPT_ID);
```

56.2 Предикат сравнения с квантором

Квантифицированное сравнение строчного значения с табличным запросом:

$$\texttt{FALSE} \leftrightarrow \exists RT \in T : (R_X \ \texttt{op} \ RT) = \texttt{FALSE}$$
$$R_X \ \texttt{op ALL} \ T = \texttt{TRUE} \leftrightarrow \forall RT \in T \Rightarrow (R_X \ \texttt{op} \ RT) = \texttt{TRUE} \vee T \equiv \emptyset$$
$$\texttt{UNKNOWN} \leftrightarrow \forall RT \in T \Rightarrow (R_X \ \texttt{op} \ RT) \neq \texttt{FALSE} \wedge \exists RT \in T : (R_X \ \texttt{op} \ RT) = \texttt{UNKNOWN}$$
$$\texttt{TRUE} \leftrightarrow \exists RT \in T : (R_X \ \texttt{op} \ RT) = \texttt{TRUE}$$
$$R_X \ \texttt{op SOME} \ T = \texttt{FALSE} \leftrightarrow \forall RT \in T \Rightarrow (R_X \ \texttt{op} \ RT) = \texttt{FALSE} \vee T \equiv \emptyset$$
$$\texttt{UNKNOWN} \leftrightarrow \forall RT \in T \Rightarrow (R_X \ \texttt{op} \ RT) \neq \texttt{TRUE} \wedge \exists RT \in T : (R_X \ \texttt{op} \ RT) = \texttt{UNKNOWN}$$
$$R_X \ \texttt{op ANY} \ T \equiv R_X \ \texttt{op SOME} \ T$$

Пример:
Найти номера сотрудников отдела 12, зарплата которых в этом отделе не является минимальной:

```
SELECT EMP_ID
FROM EMPLOYEE
WHERE DEPT_ID = 12
  AND EMP_SALARY >
    SOME (
        SELECT E1.EMP_SALARY
        FROM EMPLOYEE E1
        WHERE EMPLOYEE.DEPT_ID = E1.DEPT_ID
    );
```

56.3 Предикат IS NULL

Проверяет, являются ли неопределенными значения всех элементов строки операнда: `RX IS NULL`.

RX IS NULL	RX IS NOT NULL
$\|RX\| = 1, s = \texttt{NULL}$	TRUE
FALSE	FALSE
TRUE	
$\|RX\| = 1, s \neq \texttt{NULL}$	FALSE
TRUE	TRUE
FALSE	
$\|RX\| > 1, \forall s \in RX \Rightarrow s = \texttt{NULL}$	TRUE
FALSE	FALSE
TRUE	
$\|RX\| > 1, \forall i \neq j \exists s_i \in RX : s_i = \texttt{NULL} \wedge \exists s_j \in RX : s_j \neq \texttt{NULL}$	FALSE
FALSE	TRUE
TRUE	
$\|RX\| > 1, \forall s \in RX \Rightarrow s \neq \texttt{NULL}$	FALSE
TRUE	TRUE
FALSE	

Пример:

Найти номера и имена служащих, не задействованных в проектах:

```
SELECT EMP_ID, EMP_NAME
FROM EMPLOYEE
WHERE PRO_ID IS NULL
```

56.4 Предикат IN

Проверяет факт вхождения скалярного значения или строки в указанное множество: `s IN (s1, s2, ..., sn)` или `RX IN T`

$$\texttt{TRUE} \leftrightarrow \exists RT \in T : (RX = RT) = \texttt{TRUE}$$

$$RX \texttt{ IN } T = \begin{cases} \texttt{FALSE} & \leftrightarrow \forall RT \in T \Rightarrow (RX = RT) = \texttt{FALSE} \vee T \equiv \emptyset \\ \texttt{UNKNOWN} & \leftrightarrow \forall RT \in T \Rightarrow (RX = RT) \neq \texttt{TRUE} \wedge \exists RT \in T : (RX = RT) = \texttt{UNKNOWN} \end{cases}$$

Пример:

Найти номера сотрудников, не являющихся начальниками отделов и получающих зарплату, размер которой равен размеру зарплаты какого-либо начальника отдела:

```
SELECT EMP_ID
FROM EMPLOYEE
WHERE EMP_ID NOT IN (
    SELECT DEPT_MANAGER
    FROM DEPARTMENT
)
AND EMP_SALARY IN (
    SELECT EMP_SALARY
    FROM DEPARTMENT_MANAGER
);
```

56.5 Предикаты LIKE и SIMILAR

Сопоставление символьных и битовых строк с заданным шаблоном: `source_string LIKE pattern_string [ESCAPE symbol]`

В шаблоне могут использоваться два специальных символа:

- `_` (подчеркивание) – интерпретируется как произвольный одиночный символ
- `%` (процент) – интерпретируется как произвольная подстрока произвольной длины

Для битовых строк используются коды соответствующих символов (X'5F' и X'25').

Раздел `ESCAPE` специфицирует одиночный символ (например, '\'), используемый для изменения способа интерпретации '_' и '%'. Пары символов '_' и '\%' будут интерпретироваться как одиночные символы '_' и '%' соответственно.

Основное отличие `SIMILAR` от `LIKE` состоит в возможности использования регулярных выражений внутри шаблона (в курсе лекций не рассматривается).

Пример:

Подсчитать количество проектов, в названии которых присутствует слово 'software':

```
SELECT COUNT(*)
FROM PROJECT
WHERE PRO_TITLE LIKE '%software%'
   OR PRO_TITLE LIKE '%Software%';
```

56.6 Предикат OVERLAPS

Служит для проверки перекрытия по времени двух событий: `RX OVERLAPS RY`, $|RX| = |RY| = 2$, строки задаются в виде: (tS, tF) или (tS, `interval`), где tF = tS + `interval`

Эквивалентное представление в виде предикатов сравнения:

$$\begin{aligned}&(t_{SX} = t_{SY}) \vee \\ &(t_{SX} > t_{SY}) \wedge (t_{SX} \leq t_{FY} \vee t_{FX} \leq t_{FY}) \vee \\ &(t_{SY} > t_{SX}) \wedge (t_{SY} \leq t_{FX} \vee t_{FY} \leq t_{FX})\end{aligned}$$

Пример:

Найти названия проектов, которые будут выполняться в организации в одно и то же время:

```
SELECT P1.PRO_ID, P2.PRO_ID
FROM PROJECT P1, PROJECT P2
WHERE P1.PRO_ID <> P2.PRO_ID
  AND NOT (
    (P1.PRO_SDATE, P1.PRO_DURATION) IS DISTINCT FROM
    (P2.PRO_SDATE, P2.PRO_DURATION)
  );
```

56.7 Предикат EXISTS

Проверяет наличие строк в результате запроса:

$$\texttt{TRUE} \leftrightarrow |T| > 0$$
$$\texttt{EXISTS}(T) = \texttt{FALSE} \leftrightarrow |T| = 0$$

Запросы с предикатом EXISTS можно переформулировать в виде запросов с предикатом сравнения и агрегатной функцией COUNT(*).

Пример:

Найти номера отделов, среди служащих которых есть руководители проектов:

```
SELECT D.DEPT_ID
FROM DEPARTMENT D
WHERE EXISTS (
    SELECT E.EMP_ID
    FROM EMPLOYEE E
    WHERE E.DEPT_ID = D.DEPT_ID
      AND EXISTS (
          SELECT P.PRO_MANAGER
          FROM PROJECT P
          WHERE P.PRO_MANAGER = E.EMP_ID
      )
);
```

56.8 Предикат IS DISTINCT FROM

Позволяет проверить, являются ли две строки дубликатами.

$$\texttt{RX IS DISTINCT FROM RY} = \begin{cases} \texttt{FALSE} & \leftrightarrow \forall i = 1, |RX| \Rightarrow (s_{Xi} = s_{Yi}) = \texttt{TRUE} \vee s_{Xi} = \texttt{NULL} \wedge s_{Yi} = \\ \texttt{TRUE} & \leftrightarrow \exists i : (s_{Xi} = s_{Yi}) = \texttt{FALSE} \vee s_{Xi} = \texttt{NULL} \wedge s_{Yi} \neq \texttt{NULL} \vee s_{Xi} \end{cases}$$

Отрицательная форма предиката (IS NOT DISTINCT FROM) в языке SQL отсутствует.

Пример:

Найти пары номеров проектов, выполняющихся в организации в одно и то же время:

```
SELECT P1.PRO_ID, P2.PRO_ID
FROM PROJECT P1, PROJECT P2
WHERE P1.PRO_ID <> P2.PRO_ID
  AND NOT (
    (P1.PRO_SDATE, P1.PRO_DURATION) IS DISTINCT FROM
    (P2.PRO_SDATE, P2.PRO_DURATION)
  );
```

56.9 Предикат UNIQUE

Служит для проверки факта отсутствия строк-дубликатов в результате запроса.

UNIQUE(T) = TRUE тогда и только тогда, когда в таблице T отсутствуют строки-дубликаты, в п

Пример:
Найти названия отделов, служащих которых можно различить по размеру получаемой зарплаты:

```
SELECT DEPT_NAME
FROM DEPARTMENT
WHERE UNIQUE( SELECT EMP_SALARY
              FROM EMPLOYEE
              WHERE EMPLOYEE.DEPT_ID = DEPARTMENT.DEPT_ID );
```

56.10 Предикат MATCH

Условие соответствия строчного значения результату подзапроса:

$$RX \texttt{ MATCH } [\texttt{UNIQUE}] \texttt{ SIMPLE } T = \texttt{TRUE} \leftrightarrow \exists s \in RX : s = \texttt{NULL} \vee \exists RT \in T : (RX = RT) = \texttt{TRUE}$$

$$RX \texttt{ MATCH } [\texttt{UNIQUE}] \texttt{ PARTIAL } T = \texttt{TRUE} \leftrightarrow \forall s \in RX \Rightarrow s = \texttt{NULL} \vee \exists RT \in T : \forall s_i \neq \texttt{NULL} \Rightarrow (s_i =$$

$$RX \texttt{ MATCH } [\texttt{UNIQUE}] \texttt{ FULL } T = \texttt{TRUE} \leftrightarrow \forall s \in RX \Rightarrow s = \texttt{NULL} \vee \forall s \in RX \Rightarrow s \neq \texttt{NULL} \wedge \exists RT \in T : ($$

Если указано UNIQUE, то дополнительно проверяется, является ли найденная строка RT уникальной в T (FALSE, если неуникальная).

Пример:
Найти номера служащих, зачисленных в один из отделов, для которых в их отделах работают служащие с той же самой датой рождения:

```
SELECT EMP_ID
FROM EMPLOYEE
WHERE ( DEPT_ID, EMP_BDATE ) MATCH FULL (
    SELECT E1.DEPT_ID, E1.EMP_BDATE
    FROM EMPLOYEE E1
    WHERE E1.EMP_ID <> EMPLOYEE.EMP_ID
);
```

57 Поддержка авторизации доступа к данным в SQL. Объекты и привилегии. Пользователи и роли.

Метод авторизации доступа, используемый в SQL, относится к мандатным видам защиты данных: с каждым зарегистрированным в СУБД пользователем или ролью (субъектом) и каждым защищаемым объектом БД связывается мандат (или привилегия), определяющий действия, которые может выполнять данный субъект над данным объектом.

Принцип сокрытия информации: В SQL поддерживается принцип сокрытия информации об объектах, содержащихся в схеме БД, от субъектов, которые лишены доступа к этим объектам. Если некоторый субъект не обладает

привилегиями доступа к некоторой таблице, то при попытке выполнить какое-либо действие над ней он получит такое же диагностическое сообщение, как если бы данная таблица не существовала.

Владельцы объектов: Создатель объекта базы данных автоматически становится владельцем этого объекта, который обладает полным набором привилегий для выполнения действий над объектом, в том числе привилегией на передачу всех или части своих привилегий другим субъектам.

57.1 Привилегии доступа к объектам

Действие	**Привилегия**	**Объекты**
Просмотр	SELECT	Таблицы, столбцы, хранимые процедуры
Вставка	INSERT	Таблицы, столбцы
Модификация	UPDATE	Таблицы, столбцы
Удаление	DELETE	Таблицы
Ссылка	REFERENCES	Таблицы, столбцы
Использование	USAGE	Домены, типы и прочие определения
Инициирование	TRIGGER	Таблицы
Выполнение	EXECUTE	Хранимые процедуры
Типизация	UNDER	Определяемые пользователем типы
Передача	GRANT/ADMIN	Привилегии/роли

Таблица 1: Привилегии доступа к объектам

Привилегии над представлениями основываются на привилегиях по отношению к базовым таблицам данных представлений.

57.2 Пользователи и роли

Привилегии доступа к объектам предоставляются пользователям, а также ролям, выполнение которых, в свою очередь, может предоставляться пользователям или другим ролям. С каждым пользователем и каждой ролью связывается уникальный идентификатор авторизации (`authID`).

Роль: Динамически образуемая группа пользователей СУБД, каждый из которых обладает привилегией на исполнение данной роли, а также всеми привилегиями данной роли для доступа к объектам БД. Роли упрощают построение и администрирование системы авторизации доступа.

В стандарте SQL не определяются средства создания и ликвидации идентификаторов пользователей. Для создания и ликвидации ролей поддерживаются операторы `CREATE ROLE`/`DROP ROLE`. В стандарте также поддерживается концепция идентификатора псевдопользователя `PUBLIC`, который соответствует любому пользователю, зарегистрированному в СУБД.

Пользователю `PUBLIC` могут предоставляться привилегии доступа к объектам базы данных, как и любому другому пользователю, при этом они будут распространяться автоматически и на всех вновь создаваемых пользователей.

58 Передача и аннулирование привилегий и ролей в SQL.

58.1 Создание и ликвидация роли

58.1.1 Создание роли

Синтаксис:

```
CREATE ROLE role_name [ WITH ADMIN { CURRENT_USER | CURRENT_ROLE } ]
```

Имя роли должно отличаться от любого идентификатора авторизации (`authID`), уже определенного в СУБД. При наличии раздела `WITH ADMIN` привилегию на исполнение данной роли вместе с правом передачи привилегии получает либо текущий пользователь, либо текущая роль SQL сессии. По умолчанию привилегия на исполнение создаваемой роли передается текущему пользователю. Если SQL сессия не имеет пользователя, то текущей роли. Привилегии, требуемые для выполнения `CREATE ROLE`, определяются в реализациях (как правило, выполнение разрешается только администраторам).

58.1.2 Ликвидация роли

Синтаксис:

```
DROP ROLE role_name
```

Для выполнения `DROP ROLE` текущий `authID` SQL сессии (пользователь или роль) должен являться владельцем данной роли. При ликвидации роли автоматически ликвидируются привилегии на ее исполнение у всех пользователей и ролей, которым эта привилегия ранее передавалась.

58.2 Передача привилегий

Синтаксис:

```
GRANT { ALL PRIVILEGES | privilege_list } ON object_name TO { PUBLIC | authID_list
    [ WITH GRANT OPTION ]
    [ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
```

Привилегии передаются от текущего `authID` сессии (пользователя или роли) к указанным в списке `authID` (пользователям или ролям). Для этого он должен обладать привилегией на передачу всех или части привилегий из списка, указанного в операторе `GRANT`.

Если текущий `authID` обладает правом на передачу только части привилегий из этого списка, то передается именно эта часть и выдается предупреждение, а если он не имеет прав на передачу ни одной из перечисленных привилегий, то фиксируется ошибка.

Если указан раздел `WITH GRANT OPTION`, то привилегии из списка передаются с правом дальнейшей передачи.

Раздел `GRANTED BY` позволяет явно указать, от какого `authID` (текущего пользователя или текущей роли) передаются привилегии.

Избыточная дублирующая передача привилегий от имени одного и того же `authID` (authID1) другому (тому же самому) `authID` (authID2) игнорируется

(при этом дублирующая передача привилегии с правом передачи и без права передачи независимо от порядка действий будет означать, что authID2 обладает данной привилегией вместе с правом ее дальнейшей передачи).

58.3 Передача ролей

Синтаксис:

```
GRANT role_name_list TO { PUBLIC | authID_list }
    [ WITH ADMIN OPTION ]
    [ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
```

Оператор позволяет передавать произвольное число ролей произвольному числу пользователей или ролей.

Если текущий authID обладает правом на передачу только части ролей из этого списка, то передается именно эта часть и выдается предупреждение, а если он не имеет прав на передачу ни одной из перечисленных ролей, то фиксируется ошибка.

Если указан раздел WITH ADMIN OPTION, то привилегии на исполнение ролей из списка передаются с правом дальнейшей передачи.

Раздел GRANTED BY позволяет явно указать, от какого authID (текущего пользователя или текущей роли) передаются привилегии на исполнение ролей.

58.4 Аннулирование привилегий

Синтаксис:

```
REVOKE [ GRANT OPTION FOR ] privilege_list ON object_name FROM { PUBLIC | authID_li
    [ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
    { RESTRICT | CASCADE }
```

- RESTRICT – действие оператора отвергается, если хотя бы одна из указанных в списке привилегий была передана другому authID, у которого привилегия должна быть аннулирована.
- CASCADE – указанные привилегии аннулируются у всех authID, прямо или косвенно (через промежуточные authID) получивших привилегии от текущего authID, выполняющего данную операцию.

При наличии GRANT OPTION FOR аннулируется привилегия на передачу привилегий из указанного списка, сами привилегии при этом остаются, но при каскадном аннулировании отзыв привилегии на передачу у какого-либо authID означает и аннулирование самих привилегий у всех authID, прямо или косвенно получивших ее от данного authID.

Для успешного выполнения REVOKE необходимо, чтобы текущий authID обладал всеми или частью привилегий из указанного списка. В последнем случае аннулируется именно эта часть (с выдачей предупреждения).

Если текущий authID не обладает привилегиями на передачу ни одной из перечисленных привилегий, операция REVOKE завершится ошибкой.

58.5 Аннулирование ролей

Синтаксис:

```
REVOKE [ ADMIN OPTION FOR ] role_name_list FROM { PUBLIC | authID_list }
    [ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
    { RESTRICT | CASCADE }
```

Действие операции аннулирования ролей очень похоже на действие операции аннулирования привилегий. Отличия состоят в том, что аннулируются не привилегии, а роли, а также в том, что для аннулирования привилегии на передачу роли используется раздел `ADMIN OPTION FOR`.

Если некоторая привилегия или роль была передана `PUBLIC`, то ей обладают все пользователи. Но нет возможности аннулировать такую привилегию или роль у отдельно указываемого пользователя. Привилегия или роль была передана всем, и аннулировать ее можно только сразу у всех, то есть у псевдопользователя `PUBLIC`.

59 Транзакции. Свойства ACID. Инициация и завершение транзакций в SQL. Проверка ограничений целостности и ее связь с механизмом транзакций. Точки сохранения.

Поддержка механизма транзакций – показатель уровня развитости СУБД.

Корректное поддержание транзакций является основой обеспечения целостности баз данных, поэтому транзакции вполне уместны и в однопользовательских персональных СУБД.

Транзакции составляют базис изолированности пользователей в многопользовательских системах.

Эти два аспекта (обеспечение целостности и изолированности) взаимосвязаны, что будет рассмотрено в ходе данной лекции.

59.1 Характеристики транзакции

Транзакция– последовательность операций над базой данных, которая воспринимается системой как единая операция и обладает свойствами АСИД.

Свойства АСИД (ACID):

- **Атомарность (Atomicity)** – транзакция выполняется как единая операция и либо результаты всех операций, ее составляющих, отражаются в базе данных, либо они там гарантировано отсутствуют («все или ничего»).
- **Согласованность (Consistency)** – транзакция переводит базу данных из одного согласованного состояния в другое согласованное состояние, она успешно завершается в том и только том случае, когда действия ее операций не нарушают целостность БД.
- **Изоляция (Isolation)** – две транзакции, выполняющиеся одновременно (параллельно или квазипараллельно), не должны никоим образом действовать друг на друга (результаты одной транзакции не должны быть

видны никакой другой, до тех пор, пока первая транзакция не завершится успешно).

- **Долговечность (Durability)** – после успешного завершения транзакции все внесенные ею изменения должны быть гарантированно сохранены в БД даже в случае аппаратных или программных сбоев.

59.2 Инициация транзакций

В SQL транзакции могут образовываться:

- явно с использованием оператора `START TRANSACTION`;
- неявно, когда выполняется оператор, для которого требуется контекст транзакции, а этого контекста не существует.

Большинство операторов SQL (за исключением ряда административных операторов) требуют наличие контекста транзакции.

59.2.1 Явная инициация транзакции

`START TRANSACTION mode_list`

Характеристики транзакции:

1. Режим доступа: `READ ONLY` или `READ WRITE`.
2. Уровень изоляции: `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` или `SERIALIZABLE`.
3. Размер области диагностики (количество сохраняемых диагностических сообщений): `DIAGNOSTIC SIZE` value (значение по умолчанию определяется в реализации).

Для характеристик транзакций, инициируемых неявно, используются либо значения по умолчанию, либо значения, определяемые оператором `SET TRANSACTION mode_list` (данный оператор недопустимо выполнять в контексте активной транзакции).

59.3 Завершение транзакций

Для завершения стартовавшей транзакции пользователь должен явно инициировать один из двух операторов:

- `COMMIT [WORK] [AND [NO] CHAIN]` – фиксация транзакции (завершение с фиксацией результатов в БД).
- `ROLLBACK [WORK] [AND [NO] CHAIN]` – откат транзакции (завершение с возвратом к предыдущему состоянию БД).

Если присутствует раздел `AND CHAIN`, то по завершении текущей транзакции образуется новая, наследующая все характеристики завершенной (при этом экономятся ресурсы, требуемые для создания транзакции).

Оператор `COMMIT` считается безусловно выполненным только тогда, когда это подтверждает СУБД после выполнения всех действий, необходимых для фиксации результата транзакции. Это делается с целью защиты от сбоев, произошедших в момент выполнения `COMMIT`.

59.4 Поддержка долговечности

- **Сбой после фиксации транзакции:** Транзакция долговечна (состояние БД восстанавливается по журналу или по журналу и резервной копии).
- **Сбой во время выполнения транзакции:** Возврат к состоянию до начала выполнения транзакции.
- **Сбой в процессе фиксации транзакции:** Транзакция считается незафиксированной и все ее изменения автоматически удаляются из базы данных при ее восстановлении.

59.5 Учебный пример

Операторы модификации таблицы `EMPLOYEE` вида:

- `INSERT INTO EMPLOYEE (..., DEPT_ID, ...) VALUES ROW (..., X, ...);`
- `UPDATE EMPLOYEE SET DEPT_ID = X WHERE ...;`
- `DELETE FROM EMPLOYEE WHERE ...;`

где X – значение `DEPT_ID`, отличное от `NULL`, а во множество удаляемых строк включается хотя бы одна строка, в которой значение столбца `DEPT_ID` отличается от `NULL`, нарушают выделенное ограничение целостности.

59.6 Транзакции и ограничения целостности

В контексте выполняемой транзакции каждое ограничение целостности может находиться в одном из двух режимов:

1. Режим немедленной проверки (`IMMEDIATE`) – ограничение проверяется сразу после выполнения любой операции, изменяющей состояние БД. Эта операция отвергается, если она нарушает хотя бы одно ограничение целостности, находящееся в данном режиме (транзакция при этом не откатывается).
2. Режим отложенной проверки (`DEFERRED`) – ограничение проверяется при выполнении операции `COMMIT`. При этом допускается нарушение таких ограничений внутри транзакции, но с тем условием, чтобы к моменту завершения транзакции все ограничения целостности были соблюдены. Транзакция откатывается (`COMMIT` трактуется как `ROLLBACK`), если ее операции нарушили хотя бы одно отложенно проверяемое ограничение целостности.

Для указания режима проверки к определению ограничения целостности добавляется следующая синтаксическая конструкция (в качестве заключительной конструкции определения ограничения):

```
INITIALLY { DEFERRED | IMMEDIATE } [ [NOT] DEFERRABLE ]
```

По умолчанию предполагается `INITIALLY IMMEDIATE NOT DEFERRABLE`.

Комбинация `INITIALLY DEFERRED NOT DEFERRABLE` недопустима.

59.7 Изменение режима проверки ограничений

Режим проверки ограничений, которые специфицированы с указанием ключевого слова `DEFERRABLE`, можно изменить в ходе выполнения транзакции с помощью оператора:

```
SET CONSTRAINTS { ALL | constraint_name_list } { DEFERRED | IMMEDIATE }
```

Если в списке имен будет явно указано хотя бы одно ограничение, определенное как `NOT DEFERRABLE`, оператор будет отвергнут (ALL – сокращенная форма задания списка имен всех ограничений целостности, определенных в БД с указанием ключевого слова `DEFERRABLE`).

Ограничения, находящиеся в режиме отложенной проверки, будут проверены сразу при переводе их в режим немедленной проверки (неявно – при выполнении `COMMIT` или явно – при выполнении `SET CONSTRAINTS ... IMMEDIATE`). В первом случае при обнаружении нарушений произойдет откат транзакции. При наличии большого числа ограничений в отложенном режиме и большого числа операций в транзакции выполнение `COMMIT` становится накладным.

59.8 Точки сохранения в транзакциях

Точка сохранения– пометка в последовательности операций транзакции, которую можно использовать для ее частичного отката с сохранением жизнеспособности и результатов операций, выполненных до точки сохранения.

- Установление точки сохранения: `SAVEPOINT savepoint_name`
- Удаление точки сохранения: `RELEASE SAVEPOINT savepoint_name`
- Откат до точки сохранения: `ROLLBACK TO SAVEPOINT savepoint_name`

В одной транзакции возможно установить несколько последовательных точек сохранения. При откате до некоторой точки сохранения неявно выполняется `RELEASE` для всех точек, определенных в транзакции после данной.

```
1 START
2 SAVEPOINT sp1
3 SAVEPOINT sp2
4 ROLLBACK TO SAVEPOINT sp1
5 COMMIT
```

Безопасные операции **Рискованные операции**
(способные нарушить отложенные ограничения целостности)

Безопасные	Рискованные

Точки сохранения не нарушают принцип атомарности, поскольку извне транзакции она все равно рассматривается как единая операция (точки сохранения не видны).

60 Сериализация транзакций: виды конфликтов транзакций и порождаемые ими феномены поведения транзакций. Уровни изоляции транзакций в SQL.

60.1 Сериализация транзакций

Чтобы добиться изолированности транзакций, в СУБД должны использоваться какие-либо методы регулирования совместного выполнения транзакций.

Пусть в системе одновременно выполняется некоторое множество транзакций $S = \{T_1, T_2, \ldots, T_n\}$. План (способ) выполнения множества транзакций S, в котором, вообще говоря, чередуются или реально параллельно выполняются операции разных транзакций, называется сериальным, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций $(T_{i1}, T_{i2}, \ldots, T_{in})$.

Сериализация транзакций– это механизм их выполнения по некоторому сериальному плану. Обеспечение такого механизма является основной функцией компонента СУБД (менеджера транзакций), ответственного за управление транзакциями. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей. Основная реализационная проблема состоит в выборе метода сериализации множества транзакций, который не слишком ограничивал бы чередование их операций или реальную параллельность.

60.2 Виды конфликтов транзакций

Между транзакциями T_1 и T_2 могут существовать следующие виды конфликтов:

- **W/W** – транзакция T_2 пытается изменять объект, изменённый не закончившейся транзакцией T_1 (может привести к возникновению ситуации потерянных изменений).
- **W/R** – транзакция T_2 пытается читать объект, изменённый не закончившейся транзакцией T_1 (может привести к возникновению ситуации «грязного» чтения).
- **R/W** – транзакция T_2 пытается изменять объект, прочитанный не закончившейся транзакцией T_1 (может привести к возникновению ситуации неповторяющихся чтений).

Практические методы сериализации транзакций основываются на учёте этих конфликтов.

60.3 Таблица совместимости блокировок

Правила совместимости захватов одного объекта разными транзакциями приведены в таблице
В первом столбце перечислены возможные состояния объекта с точки зрения синхронизацион

Текущий режим	Запрашиваемый режим	Совместимость
X	X	нет
X	S	нет
X	IX	нет
X	IS	нет
X	SIX	нет
S	X	нет
S	S	да
S	IX	нет
S	IS	да
S	SIX	нет
IX	X	нет
IX	S	нет
IX	IX	да
IX	IS	нет
IX	SIX	нет
IS	X	нет
IS	S	да
IS	IX	нет
IS	IS	да
IS	SIX	нет
SIX	X	нет
SIX	S	нет
SIX	IX	нет
SIX	IS	нет
SIX	SIX	нет

Таблица 2: Таблица совместимости режимов гранулированных синхронизационных блокировок

60.4 Уровни изоляции транзакций в SQL

Существует возможность обеспечения разных уровней изолированности для разных транзакций, выполняющихся в одной системе баз данных.

Для корректного соблюдения ограничений целостности достаточно предотвратить только возникновение феномена «потерянных изменений».

В SQL (начиная с версии SQL-92) предусмотрено 4 уровня изолированности, что связано с наличием трёх оставшихся феноменов. На первом уровне могут проявляться все три феномена, каждый следующий устраняет возможность проявления одного из феноменов. И только четвертый уровень будет соответствовать предельной изолированности выполнения транзакций.

Существует ряд приложений, которым хватает первого уровня изолированности, например, прикладные или системные статистические утилиты, для которых некорректность индивидуальных данных несущественна. При этом удаётся существенно сократить накладные расходы СУБД и повысить общую эффективность.

60.4.1 Уровень изоляции `READ UNCOMMITTED`

На уровне изоляции `READ UNCOMMITTED` могут проявляться все три феномена. Одновременное задание `READ UNCOMMITTED` и режима доступа `READ WRITE` не

допускается (чтобы не допустить возможности зафиксировать «грязные» данные).

Уровень изоляции	Грязное чтение	Неповторяющееся чтение	Фантомы
READ UNCOMMITTED	**Да**	**Да**	**Да**
READ COMMITTED	+	+	-
REPEATABLE READ	+	+	+
SERIALIZABLE	-	-	-

Таблица 3: Феномены на разных уровнях изоляции транзакций

- `TRUE` – **Да**
- `FALSE` – **Нет**

Описание:

- `READ UNCOMMITTED`: Возможность видеть изменения, производимые ещё не зафиксированными параллельными транзакциями.
- `READ COMMITTED`: Допускаются изменения объектов, прочитанных другими одновременно выполняющимися транзакциями.
- `REPEATABLE READ`: Допускается добавление строк к таблицам, которые удовлетворяют условиям выборки, выполненной в других параллельных транзакциях.
- `SERIALIZABLE`: Предельная изолированность одно-временных выполняющихся транзакций.

61 Сериализация транзакций: сериальный план. Основные подходы к сериализации. Двухфазный протокол синхронизационных блокировок.

Чтобы добиться изолированности транзакций, в СУБД должны использоваться какие-либо методы регулирования совместного выполнения транзакций.

Пусть в системе одновременно выполняется некоторое множество транзакций $S = \{T_1, T_2, \ldots, T_n\}$. План (способ) выполнения множества транзакций S, в котором, вообще говоря, чередуются или реально параллельно выполняются операции разных транзакций, называется сериальным, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций $(T_{i1}, T_{i2}, \ldots, T_{in})$.

61.1 Основные подходы к сериализации транзакций

Существуют два базовых подхода к сериализации транзакций:

- основанный на синхронизационных захватах объектов базы данных;
- основанный на использовании временных меток.

Суть обоих подходов состоит в обнаружении конфликтов транзакций и их устранении.

Для каждого из подходов имеются две разновидности:

- пессимистическая;
- оптимистическая.

При применении пессимистических методов, ориентированных на ситуации, когда конфликты возникают часто, конфликты распознаются и разрешаются немедленно при их возникновении.

Оптимистические методы основываются на том, что результаты всех операций модификации базы данных сохраняются в рабочей памяти транзакций, реальная модификация базы данных, а также распознавание и разрешение конфликтов производится только на стадии фиксации транзакции.

Далее мы ограничимся рассмотрением более распространённых пессимистических разновидностей методов сериализации транзакций.

61.2 Двухфазный протокол синхронизационных блокировок (2PL)

Наиболее распространённым в централизованных СУБД, включающих системы, основанные на архитектуре «клиент-сервер», является подход, основанный на соблюдении двухфазного протокола синхронизационных захватов объектов баз данных (**Two-Phase Locking Protocol, 2PL**).

В общих чертах подход состоит в том, что перед выполнением любой операции в транзакции T над некоторым объектом базы данных o от имени транзакции T запрашивается синхронизационная блокировка объекта o в соответствующем режиме (в зависимости от вида операции):

- **Совместный режим – S (Shared)**: означающий совместную (по чтению) блокировку объекта и требуемый для выполнения операции чтения объекта.
- **Монопольный режим – X (eXclusive)**: означающий монопольную (по записи) блокировку объекта и требуемый для выполнения операций вставки, удаления и модификации объекта.

61.2.1 Совместимость блокировок

- Блокировки одних и тех же объектов по чтению (S) несколькими транзакциями совместимы, т.е. нескольким транзакциям допускается одновременно читать один и тот же объект (конфликта R/R не существует).
- Блокировка объекта одной транзакцией по чтению (S) не совместима с блокировкой другой транзакцией того же объекта по записи (X), т.е. никакой транзакции нельзя изменять объект, читаемый некоторой транзакцией, кроме самой этой транзакции, и никакой транзакции нельзя читать объект, изменяемый некоторой транзакцией, кроме самой этой транзакции (нарушение этого правила приведёт к возникновению конфликтов R/W или W/R).

- Блокировки одного и того же объекта по записи (X) разными транзакциями не совместимы, т.е. никакой транзакции нельзя изменять объект, изменяемый некоторой транзакцией, кроме самой этой транзакции (нарушение этого правила приведёт к возникновению конфликта W/W).

Для обеспечения сериализации транзакций синхронизационные блокировки объектов, произведённые по инициативе транзакции, можно снимать только при её завершении. Это требование порождает двухфазный протокол синхронизационных захватов – 2PL. В соответствии с этим протоколом выполнение транзакции разбивается на две фазы:

1. Первая фаза транзакции (выполнение операций над базой данных) – накопление блокировок.

2. Вторая фаза транзакции (фиксация или откат) – снятие блокировок.

Достаточно легко убедиться, что при соблюдении двухфазного протокола синхронизационных блокировок действительно обеспечивается сериализация транзакций на третьем уровне изолированности. При этом устраняется проявление трёх феноменов: «потерянных изменений», «грязного чтения» и «неповторяющегося чтения».

62 Сериализация транзакций: гранулированные и предикатные блокировки.

Подобные рассуждения привели к разработке механизма гранулированных синхронизационных блокировок.

При применении этого подхода синхронизационные блокировки могут запрашиваться по отношению к объектам разного уровня:

- файлам, таблицам и кортежам.

Требуемый уровень объекта определяется тем, какая операция выполняется. Например, для выполнения операции уничтожения таблицы объектом синхронизационной блокировки должна быть вся таблица, а для выполнения операции удаления кортежа – этот кортеж.

Объект любого уровня может быть заблокирован в режиме S или X.

Для согласования блокировок разного уровня вводятся специальный протокол гранулированных блокировок и новые типы блокировок:

- Перед установкой блокировки на некоторый объект базы данных в режиме S или X соответствующий объект верхнего уровня должен быть заблокирован в режиме `IS`, `IX` или `SIX`.

62.1 Режимы гранулированных синхронизационных блокировок

- **Intented for Shared lock (IS)**: означает намерение заблокировать некоторый объект o', входящий в o, в совместном режиме S (например, при чтении кортежей из таблицы `Tab` эта таблица должна быть заблокирована в режиме `IS`, а до этого в таком же режиме должна быть заблокирован файл, в котором располагается таблица `Tab`).

- **Intented for eXclusive lock (IX)**: означает намерение заблокировать некоторый объект o', входящий в o, в монопольном режиме X (например, для удаления кортежей из таблицы `Tab` эта таблица должна быть заблокирована в режиме `IX`, а до этого в таком же режиме должна быть заблокирован файл, в котором располагается таблица `Tab`).

- **Shared, Intented for eXclusive lock (SIX)**: означает совместную блокировку S всего составного объекта с намерением впоследствии блокировать какие-либо входящие в него объекты в монопольном режиме X (например, если выполняется длинная операция просмотра таблицы `Tab` с возможностью удаления некоторых просматриваемых кортежей, то экономичнее всего заблокировать таблицу `Tab` в режиме `SIX`, а до этого заблокировать в режиме `IS` файл, в котором располагается таблица `Tab`).

62.1.1 Таблица совместимости режимов гранулированных блокировок

Текущий режим	Запрашиваемый режим	Совместимость
X	X	нет
X	S	нет
X	IX	нет
X	IS	нет
X	SIX	нет
S	X	нет
S	S	да
S	IX	нет
S	IS	да
S	SIX	нет
IX	X	нет
IX	S	нет
IX	IX	да
IX	IS	нет
IX	SIX	нет
IS	X	нет
IS	S	да
IS	IX	нет
IS	IS	да
IS	SIX	нет
SIX	X	нет
SIX	S	нет
SIX	IX	нет
SIX	IS	нет
SIX	SIX	нет

Таблица 4: Таблица совместимости режимов гранулированных синхронизационных блокировок

62.1.2 Обоснование совместимости блокировок

- **Блокировка X**:

  - Не совместима с S, IX, SIX.

- **Блокировка S**:
  - Совместима с S и IS.
  - Не совместима с X, IX, SIX.
- **Блокировка IX**:
  - Совместима только с IX.
  - Не совместима с S, X, SIX.
- **Блокировка IS**:
  - Совместима с S и IS.
  - Не совместима с X, IX, SIX.
- **Блокировка SIX**:
  - Не совместима ни с одним другим режимом.

62.2 Предикатные синхронизационные блокировки

Даже гранулированные синхронизационные захваты в общем случае не решают проблему фантомов. Для решения этой проблемы необходимо перейти от блокировок индивидуальных («физических») объектов базы данных к блокировке условий (предикатов), которым удовлетворяют эти объекты.

Проблема фантомов не возникает при использовании для блокировок уровня таблиц (режимы S и X), поскольку таблица как логический объект представляет собой неявное условие принадлежности для входящих в неё кортежей (т.е. это простой и частный случай предикатной блокировки).

Поскольку любая операция над реляционной базой данных задаётся некоторым условием, которому должны удовлетворять объекты этого набора, идеальным выбором было бы требовать синхронизационную блокировку в режиме S или X именно этого условия. Но если посмотреть на общий вид условий, допускаемых, например, в языке SQL, то становится абсолютно непонятно, как определить совместимость двух предикатных блокировок.

Ясно, что без этого использовать предикатные блокировки для сериализации транзакций невозможно, а в общей форме проблема неразрешима.

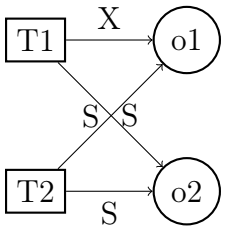
63 Синхронизационные тупики, способы их обнаружения и разрушения.

63.1 Синхронизационные тупики

Одним из наиболее чувствительных недостатков метода сериализации транзакций на основе синхронизационных блокировок (во всех рассмотренных вариантах) является возможность возникновения тупиков (deadlocks) между транзакциями.

Самый простой сценарий возникновения синхронизационного тупика между двумя транзакциями показан на рисунке. Транзакции T1 и T2 устанавливают монопольные блокировки объектов o1 и o2 соответственно. После этого

T1 требуется совместная блокировка объекта o2, а T2 – совместная блокировка объекта o1. Ни одно из этих требований блокировки не может быть удовлетворено, следовательно, ни одна из транзакций не может продолжаться. Поэтому монопольные блокировки объектов никогда не будут сняты, а требования совместных блокировок не будут удовлетворены.



Поскольку никакого естественного выхода из тупиков не существует, то такие ситуации необходимо обнаруживать и искусственно устранять.

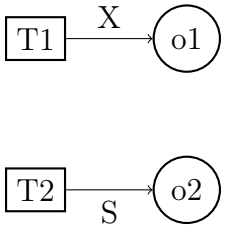
63.2 Распознавание синхронизационных тупиков

Основой обнаружения тупиковых ситуаций является построение или постоянное поддержание графа ожидания транзакций.

Граф ожидания транзакций – это ориентированный двудольный граф, в котором существует два типа вершин:

- вершины, соответствующие транзакциям (будем изображать их прямоугольниками);
- вершины, соответствующие объектам блокировок (будем изображать их окружностями).

Ребро из вершины-транзакции к вершине-объекту существует в том и только в том случае, если для этой транзакции имеется удовлетворённая блокировка данного объекта.



Ребро из вершины-объекта к вершине-транзакции существует тогда и только тогда, когда эта транзакция ожидает удовлетворения запроса блокировки данного объекта.

Легко показать, что в системе существует тупиковая ситуация в том и только в том случае, когда в графе ожидания транзакций имеется хотя бы один цикл.

Традиционной техникой нахождения циклов в ориентированном графе является редукция графа (для неё существует множество разновидностей).

Рассмотрим это на примере. Предположим, что все блокировки являются монопольными (т.е. для каждой вершины-объекта имеется не более одного входящего ребра).

1. Из графа ожидания удаляются все ребра, исходящие из вершин-транзакций, в которые не входят ребра из вершин-объектов (это основывается на том разумном предположении, что транзакции, не ожидающие удовлетворения запроса блокировок, могут успешно завершиться и освободить блокировки).

2. Удаляются ребра, входящие в вершины-транзакции, из которых не исходят ребра, ведущие к вершинам-объектам (транзакции, ожидающие удовлетворения блокировок, но не удерживающие заблокированные объекты, не могут быть причиной тупика).

3. Для тех вершин-объектов, для которых не осталось входящих ребер, но существуют исходящие, ориентация одного из исходящих ребер, выбираемого произвольным образом, изменяется на противоположную (это моделирует удовлетворение запроса блокировки).

4. После этого снова повторяются описанные действия и так до тех пор, пока не прекратится удаление ребер.

5. Если в графе остались дуги, то они обязательно образуют цикл.

63.3 Разрушение синхронизационных тупиков

Для обеспечения возможности продолжения работы хотя бы для части транзакций, попавших в тупик, необходимо его разрушить. Разрушение тупика заключается в выборе в цикле транзакций так называемой транзакции-«жертвы» и её принудительном откате (полном или до некоторой точки сохранения). После отката освобождаются блокировки, и может быть продолжено выполнение других транзакций.

Возможные критерии выбора «жертвы»:

1. самая «богатая» транзакция, т.е. удерживающая наибольшее количество заблокированных объектов (недостаток: такая транзакция обычно выполнялась дольше других и на её выполнение затрачено больше системных ресурсов);

2. самая «молодая» транзакция, т.е. существующая в системе наименьшее время (недостаток: она скорее всего заблокировала не так уж и много объектов и её принудительное завершение может не помочь разрушить тупик);

3. случайный выбор (в среднем этот подход может привести к хорошим результатам, но он не учитывает приоритет транзакций – можно случайно откатить наиболее важную транзакцию);

4. выбор с учётом многофакторной оценки «стоимости» транзакции, в которую с разными весами входят время выполнения, число накопленных блокировок, приоритет и другие критерии.

Разрушение синхронизационных тупиков

Естественно, что такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей, которого, к сожалению, в данной ситуации невозможно избежать.

Заметим, что в централизованных системах стоимость построения графа ожидания сравнительно невелика, но она становится слишком большой в распределённых СУБД, в которых транзакции могут выполняться в разных узлах сети.

Поэтому в распределённых системах обычно используются другие методы сериализации транзакций.

Варианты безтупиковых блокировочных протоколов:

- "no wait" (транзакция, запрашивающая блокировку и не получающая её, откатывается);
- "wait-die" («старшая» по времени старта транзакция всегда получает блокировку, но может подождать, если «младшая» её опередила, «младшая» откатывается если объект захвачен «старшей», но сохраняет прежнюю временную метку при рестарте).

64 Сериализация транзакций на основе временных меток.

Метод временных меток

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редкого возникновения конфликтов транзакций и не требующий построения графа ожидания транзакций, основан на использовании временных меток (Timestamp Ordering).

Основная идея метода временных меток, у которого существует множество разновидностей, состоит в следующем:

- если транзакция T_1 началась раньше транзакции T_2, то система обеспечивает такой сериальный план, как если бы транзакция T_1 была целиком выполнена до начала T_2.

Для этого каждой транзакции T предписывается временная метка $t(T)$, соответствующая времени начала выполнения транзакции T.

Метод временных меток

При выполнении операции над объектом o некоторая транзакция T помечает его своими идентификатором, временной меткой и типом операции (чтение или изменение).

Перед выполнением операции над объектом o транзакция T_2 выполняет следующие действия:

- Проверяет, помечен ли объект o какой-либо транзакцией T_1. Если не помечен, то помечает этот объект своей временной меткой и типом операции и выполняет операцию.
- Иначе транзакция T_2 проверяет, не завершилась ли транзакция T_1, пометившая этот объект. Если завершилась, то T_2 помечает объект o и выполняет свою операцию.
- Если транзакция T_1 не завершилась, то T_2 проверяет конфликтность операций. Если операции неконфликтны, то при объекте o запоминается идентификатор транзакции T_2, остаётся или проставляется временная метка с меньшим значением, и транзакция T_2 выполняет свою операцию.

- Если операции транзакций T_2 и T_1 конфликтуют, то если $t(T_1) > t(T_2)$ (т.е. транзакция T_1 является более «молодой», чем T_2), то производится откат T_1 и всех других транзакций, идентификаторы которых сохранены при объекте o, и T_2 выполняет свою операцию.

- Если же $t(T_1) < t(T_2)$ (T_1 «старше» T_2), то производится откат T_2; T_2 получает новую временную метку и начинается заново.

Метод временных меток

К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо.

Кроме того, в распределённых системах не очень просто вырабатывать глобальные временные метки с отношением полного порядка (это отдельная большая наука). Основная проблема: централизованный сервер времени будет узким местом, а распределённые «часы» требуют синхронизации, что увеличивает количество обменов между серверами.

Но в распределённых системах эти недостатки окупаются тем, что не нужно распознавать тупики, а построение графа ожидания в распределённых системах стоит очень дорого.

65 Версионный вариант алгоритма временных меток.

Версионные методы

Основная идея версионных алгоритмов сериализации транзакций состоит в том, что в базе данных допускается существование нескольких «версий» одного и того же объекта. Эти алгоритмы, главным образом, направлены на преодоление конфликтов транзакций категорий R/W и W/R, позволяя выполнять операции чтения над некоторой предыдущей версией объекта базы данных (сериализация остаётся в силе, т.к. предыдущая версия также соответствует некоторому целостному состоянию БД). В результате операции чтения выполняются без задержек и тупиков, свойственных механизмам синхронизационных захватов, а также без некоторых откатов, возможных при применении метода временных меток.

Алгоритмы управления транзакциями, основанные на поддержке версий, достаточно широко распространены в области SQL-ориентированных СУБД. В частности, подобные алгоритмы используются в СУБД Oracle и PostgreSQL.

На лекции будут рассмотрены:

- Версионный вариант алгоритма временных меток (Рид, 1983 г.)

- Версионный вариант двухфазного протокола синхронизационных захватов (Бернштейн, Хадзилакос, Гудман, 1987 г.)

- Версионно-блокировочный протокол для поддержки только читающих транзакций (Вейкум, Воссен, 2002 г.)

65.1 Версионный алгоритм временных меток

Одним из наиболее простых версионных алгоритмов является версионный вариант алгоритма временных меток (Multiversion Timestamp Ordering, MVTO).

Как и в простом методе временных меток, в алгоритме MVTO порядок выполнения операций транзакций задаётся порядком временных меток, которые получают транзакции во время старта.

Временные метки также используются для идентификации версий данных при чтении и модификации – каждая версия получает временную метку той транзакции, которая её записала.

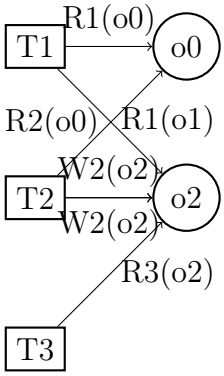
- Как и раньше, временную метку, полученную транзакцией T_i в начале её работы, будем обозначать как $t(T_i)$.
- Операция чтения объекта базы данных o, выполняемая в транзакции T_i, будет обозначаться как $R_i(o)$ – это запрос на чтение объекта o в данной транзакции.
- Для обозначения того, что транзакция T_i на самом деле читает версию объекта базы данных o, созданную транзакцией T_k, будем использовать запись $R_i(o_k)$.
- Операция изменения объекта базы данных o, выполняемая в транзакции T_i, будет обозначаться как $W_i(o)$.
- Для обозначения того, что транзакция T_i записывает версию объекта базы данных o, будем использовать запись $W_i(o_i)$.

65.2 Версионный алгоритм временных меток

Алгоритм MVTO работает следующим образом:

- Любая операция $R_i(o)$ преобразуется в операцию $R_i(o_k)$, где o_k – это версия объекта o, помеченная наибольшей временной меткой $t(T_k)$, такой что $t(T_k) \leq t(T_i)$, т.е. транзакции T_i для чтения даётся версия объекта o, созданная транзакцией T_k, которая не «моложе» T_i, но «моложе» любой другой транзакции T_n, создававшей свою версию объекта o (при этом транзакция T_k может быть ещё не завершённой).
- При обработке операции $W_i(o)$ выполняются следующие действия:
  1. Если к этому времени некоторой незафиксированной транзакцией T_n уже выполнена некоторая операция $R_n(o_k)$, такая что $t(T_k) \leq t(T_i) < t(T_n)$, то операция $W_i(o)$ не выполняется, а транзакция T_i откатывается.
  2. В противном случае $W_i(o)$ преобразуется в $W_i(o_i)$, т.е. образуется ещё одна версия объекта o.
- При откате любой транзакции уничтожаются все созданные ею версии объектов базы данных и откатываются все транзакции, прочитавшие хотя бы одну из этих версий (т.е. откаты транзакций могут быть «каскадными»).

- Выполнение операции фиксации транзакции T_i (`COMMIT`) откладывается до того момента, когда завершатся все транзакции, записавшие версии данных, прочитанные T_i. Без соблюдения этого требования не соблюдалось бы свойство долговечности (**durability**) транзакций, поскольку при откате некоторых транзакций потребовалось бы откатывать и ранее зафиксированные транзакции.



Преимущества алгоритма MVTO:

- отсутствие задержек и откатов при выполнении операций чтения.

Недостатки алгоритма MVTO:

- возможность возникновения каскадных откатов транзакций при выполнении операций записи;
- в базе данных может накапливаться произвольное число версий одного и того же объекта, и определение того, какие версии больше не требуются, является серьёзной технической проблемой (требуются «сборщики мусора»).

66 Версионный вариант двухфазного протокола синхронизационных блокировок.

Версионный вариант двухфазного протокола синхронизационных захватов

При описании двухверсионного варианта протокола двухфазного протокола синхронизационных захватов (Two-Version Two-Phase Locking Protocol, 2V2PL) будем называть:

- текущими версиями объектов базы данных версии, созданные зафиксированными транзакциями с наиболее поздним временем фиксации;
- незафиксированными версиями – версии, созданные ещё незавершившимися транзакциями.

При следовании протоколу 2V2PL в каждый момент времени существует не более одной незафиксированной версии каждого объекта базы данных.

Операции любой транзакции T_i над объектом базы данных o обрабатываются следующим образом:

- операция $R_i(o)$ немедленно выполняется над текущей версией объекта o;

- операция $W_i(o)$, приводящая к созданию новой версии объекта o, выполняется только после завершения (фиксации или отката) транзакции, создавшей незафиксированную версию объекта o;

- выполнение операции `COMMIT` откладывается до тех пор, пока не завершатся все транзакции T_k, прочитавшие текущие версии объектов базы данных, которые должны замениться незафиксированными версиями этих объектов, созданными транзакцией T_i.

66.1 Версионный вариант двухфазного протокола синхронизационных захватов

Для реализации такого поведения используются три типа блокировок:

- **RL (Read Lock)** – в этом режиме блокируется любой объект базы данных o перед выполнением операции чтения его текущей версии

  – удержание этой блокировки до конца транзакции гарантирует, что при повторном чтении объекта o будет прочитана та же версия этого объекта

- **WL (Write Lock)** – в этом режиме блокируется любой объект базы данных o перед выполнением операции, приводящей к созданию новой (незафиксированной) версии этого объекта

  – удержание этой блокировки до конца транзакции гарантирует, что в любой момент времени будет существовать не более одной незафиксированной версии любого объекта базы данных

- **CL (Commit Lock)** – блокировка устанавливается во время выполнения операции `COMMIT` транзакции и затрагивает любой объект базы данных, новую версию которого создала данная транзакция

  – удовлетворение этой блокировки для данной транзакции гарантирует, что завершились все транзакции, читавшие текущие версии объектов, новые версии которых были созданы при выполнении данной транзакции, и, следовательно, их можно заменить

Правила совместимости этих блокировок приведены в таблице.

	RL(o)	WL(o)	CL(o)
RL(o)	да	да	нет
WL(o)	да(o)	нет	нет
CL(o)	нет	нет	нет

Таблица 5: Правила совместимости блокировок

Операция `COMMIT` блокирует как чтение, так и запись для выполнения успешной замены версии объекта.

Операция чтения может блокироваться только на время фиксации транзакции, заменяющей текущую версию требуемого объекта базы данных.

Для выполнения операции записи требуется долговременная монопольная блокировка соответствующего объекта базы данных, которая, однако, в этом

случае совместима с блокировкой этого же объекта по чтению, поскольку в действительности блокируются разные версии этого объекта (соответственно, незафиксированная и текущая).

И, конечно, как и во всех схемах сериализации транзакций на основе блокировок, здесь возможны синхронизационные тупики.

67 Версионно-блокировочный протокол сериализации транзакций для поддержки только читающих транзакций.

При применении гибридного (версионно-блокировочного) протокола для поддержки эффективного выполнения транзакций, не изменяющих состояние базы данных (Multiversion Protocol for Read-Only Transactions, ROMV), при образовании каждой транзакции явно указывается её тип:

- только читающая (read-only) транзакция;
- изменяющая (update) транзакция.

В только читающих транзакциях допускается использование только операций чтения объектов базы данных, а в изменяющих транзакциях – операций и чтения, и записи.

Изменяющие транзакции выполняются в соответствии с обычным протоколом 2PL:

- перед выполнением операции чтения или записи объекта базы данных o этот объект должен быть заблокирован в режиме S или X соответственно;
- блокировки объектов удерживаются до конца изменяющей транзакции.

Каждая операция записи объекта o создаёт его новую версию, которая при завершении транзакции помечается временной меткой, соответствующей моменту фиксации этой транзакции.

67.1 Гибридный протокол для поддержки только читающих транзакций

Каждая только читающая транзакция при своём образовании получает соответствующую временную метку.

При выполнении операции чтения объекта базы данных o читающая транзакция получает доступ к версии объекта o, образованной изменяющей транзакцией, которая хронологически последней зафиксировалась к моменту образования данной читающей транзакции.

При использовании протокола ROMV в базе данных может возникать произвольное число версий объектов. Поэтому требуется создание специального сборщика мусора, который должен удалять ненужные версии данных.

Простейший сборщик мусора удаляет все неиспользуемые версии, значения временных меток которых меньше значения временной метки старейшей активной только читающей транзакции.

Преимущества протокола ROMV:

- Принципиальное отсутствие синхронизационных задержек при выполнении операций чтения только читающих транзакций.
- В сравнении с версионным алгоритмом временных меток MVTO выигрывает в отсутствии откатов только читающих транзакций.

Недостатки протокола ROMV:

- При работе изменяющих транзакций возможно возникновение синхронизационных тупиков и откатов, и здесь требуется использовать обычные методы распознавания и разрушения тупиков.
- Требуется эффективный сборщик мусора для удаления ненужных версий данных.

68 Ситуации, требующие восстановления базы данных и общие принципы восстановления информации. Понятие журнала и основные подходы к его ведению. Индивидуальные откаты транзакций.

68.1 Ситуации, при которых требуется производить восстановление состояния БД

1. **Индивидуальный откат транзакции** (в случае явного завершения оператором `ROLLBACK` или инициируемого системой в различных ситуациях: нарушение отложенных ограничений целостности, выбор транзакции в качестве «жертвы» при разрушении тупика, возникновение исключительной ситуации в прикладной программе):
    - В данной ситуации нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.
2. **Восстановление после внезапной потери содержимого оперативной памяти** (мягкий сбой) в результате аварийного выключения электропитания, при возникновения неустранимого сбоя процессора, срабатывания контроля основной памяти и т.п.:
    - Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти СУБД.
3. **Восстановление после поломки основного внешнего носителя базы данных** (жесткий сбой) – редкая ситуация вследствие достаточно высокой надежности современных устройств внешней памяти, тем не менее, СУБД должна быть в состоянии восстановить базу данных даже и в этом случае:
    - Здесь основой восстановления является архивная копия и журнал изменений базы данных.

68.2 Подходы к организации журнализации

Во всех трёх случаях основой восстановления является хранение избыточных данных. Эти избыточные данные хранятся в журнале, содержащем последовательность записей об изменении базы данных.

1. **Поддержка отдельных локальных журналов изменений БД для каждой транзакции в основной памяти СУБД и общего журнала изменений БД во внешней памяти:**
   - Используются для индивидуальных откатов транзакций и восстановления состояния БД после мягких и жестких сбоев.
   - Позволяет быстро выполнять индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общем журналах.
2. **Поддержка только общего журнала изменений БД:**
   - Используется и при выполнении индивидуальных откатов.
   - Чаще применяется, так как избегает дублирования информации.

69 Управление буферным пулом основной памяти. Буферизация (кеширование) блоков базы данных и журнала.

69.1 Управление буферным пулом БД

В правильно организованных СУБД поддерживается собственная стратегия замещения страниц буферного пула. Задача, которую решает СУБД, очень похожа на задачу, которую решает операционная система при управлении виртуальной памятью.

- В случае операционной системы, если некоторый процесс требует обеспечения доступа к странице виртуальной памяти, отсутствующей в основной памяти, и нет свободных страниц основной памяти, то:
  1. В соответствии с некоторым критерием выбирается некоторая занятая страница основной памяти;
  2. Она освобождается, т.е. изымается из виртуальной памяти какого-то процесса и, может быть, копируется на диск;
  3. Освободившаяся страница подключается к виртуальной памяти запросившего процесса с предварительным считыванием с диска (области своппинга) нужных данных.

69.2 Управление буферным пулом БД

В случае СУБД, если при выполнении некоторой операции в некоторой транзакции требуется доступ к некоторому блоку базы данных, и копия этого блока отсутствует в буферном пуле, СУБД должна:

1. Выделить какую-либо страницу буферного пула;
2. Считать в неё с диска требуемый блок базы данных;
3. Предоставить доступ к этой странице запросившей операции;
4. Если в буферном пуле нет свободных страниц, тогда СУБД в соответствии с некоторым критерием:
   (a) Находит некоторую занятую страницу;
   (b) Освобождает её;
   (c) Возможно, выталкивает во внешнюю память.

Основная разница между этими случаями состоит в критерии выборки занятой страницы для «откачки».

69.3 Управление буферным пулом БД

Почти всегда операционная система стремится заменить страницу, к которой предположительно дольше всего не будет обращений, но, поскольку предвидение будущего невозможно, оно аппроксимируется прошлым.

В частности, в одном из популярных алгоритмов замещения страниц **LRU** (Least Recently Used) принимается предположение, что дольше всего в будущем не потребуется та страница, к которой дольше всего не обращались в прошлом.

69.4 Управление буферным пулом БД

В стратегии замещения страниц буферного пула СУБД тоже чаще всего используется некоторая разновидность алгоритма LRU. Но СУБД располагает большей информацией о страницах буферного пула, чем операционная система о страницах основной памяти.

- Например, если в некоторой транзакции выполняется сканирование некоторой таблицы без использования индекса, и при переходе к следующему кортежу был затребован доступ к некоторому блоку базы данных с соответствующим перемещением копии этого блока в некоторую страницу буферного пула, то подсистема управления буферным пулом «знает», что эта страница еще точно потребуется до тех пор, пока не будет прочитан последний кортеж сканируемой таблицы, располагающийся в данной странице.

69.5 Управление буферным пулом БД

Более того, СУБД «знает», какой блок базы данных потребуется после завершения просмотра кортежей данного блока, и может заранее переместить его копию в некоторую страницу буферного пула («упреждающая выборка», **prefetch**).

Кроме того, некоторые блоки базы данных заведомо требуются чаще других блоков:

- При любом просмотре таблицы на основе некоторого индекса гарантированно потребуется доступ к корневому блоку соответствующего B-дерева.

- При вставке кортежа в любую таблицу или удалении из неё кортежа будет необходимо должным образом изменить все определённые для неё индексы, и для этого тоже гарантированно потребуется доступ к корневым блокам всех соответствующих B-деревьев.

69.6 Управление буферным пулом БД

Поэтому в стратегии замещения страниц буферного пула базы данных обычно используется алгоритм LRU с приоритетами страниц:

- **Высокоприоритетные страницы** стареют (т.е. становятся кандидатами на замещение) медленнее, чем низкоприоритетные страницы.
- В частности, страницы, содержащие копии корневых блоков индексов, являются настолько высокоприоритетными, что обычно никогда не замещаются.
- Кроме того, поддерживается предварительное считывание в буферную память копий блоков, доступ к которым вскоре понадобится.

70 Физическая синхронизация. Протокол 2PL для физической синхронизации и борьба с физическими синхронизационными тупиками.

70.1 Физическая синхронизация

Поскольку в СУБД может одновременно («параллельно») выполняться несколько транзакций, вполне реальна ситуация, когда в двух одновременно выполняемых операциях требуется доступ к одному и тому же блоку базы данных, т.е. к одной и той же буферной странице, содержащей копию этого блока.

Понятно, что в одновременном доступе для чтения содержимого блока ничего плохого нет, но параллельное изменение блока может привести к непредсказуемым результатам.

Следует заметить, что координацию параллельного доступа к страницам буферного пула не обеспечивает логическая синхронизация (на уровне кортежей, таблиц, файлов), используемая для сериализации транзакций.

70.2 Физическая синхронизация

Например, предположим, что в двух параллельно выполняемых транзакциях одновременно выполняются операции модификации кортежей:

- с `tid` = (n, 1),
- с `tid` = (n, 2).

Если в СУБД используются блокировки на уровне кортежей, то система допустит параллельное выполнение этих двух операций, и они будут одновременно изменять страницу, содержащую копию блока базы данных с номером n. При выполнении обеих операций может потребоваться перемещение кортежей

внутри этого блока, и понятно, что в результате ничего хорошего, скорее всего, не получится.

Аналогично, логическая синхронизация может легко допустить параллельное выполнение нескольких операций, требующих обновления одного и того же индекса. Некоординированное параллельное обновление B-дерева с большой вероятностью приводит к разрушению его структуры.

70.3 Физическая синхронизация

Поэтому при выполнении операций низкого уровня (ядра СУБД) необходимо поддерживать дополнительную «физическую» синхронизацию, в которой единицами блокировки служат страницы буферного пула (или блоки) базы данных.

В пределах операции перед чтением из страницы буферного пула блока базы данных требуется запросить у подсистемы управления буферным пулом блокировку соответствующей страницы блока в режиме S, а перед записью в страницу (в блок) – её блокировку в режиме X.

Совместимость блокировок обычная (т.е. как и для транзакций).

Но блокировки страниц буферного пула нужны не только для координации параллельного доступа к страницам при параллельном выполнении транзакций. При выполнении операций низкого уровня могут возникать ошибки, обнаруживаемые в середине операции, уже после того, как одна или несколько страниц буферного пула блоков базы данных были изменены.

70.4 Физическая синхронизация

Например, может выполняться операция вставки кортежа в некоторую таблицу, нарушающая уникальность некоторого индекса, определённого над этой таблицей. Нарушение уникальности этого индекса будет обнаружено при попытке вставить в него новый ключ, но до этого новый кортеж уже мог быть размещён в блоке данных, и некоторые индексы уже могли быть успешно обновлены.

При обнаружении ошибки операции нужно ликвидировать все её следы в базе данных и выдать соответствующий код ошибки. Проще всего сделать это, произведя обратные изменения всех страниц (блоков базы данных), которые были изменены при прямом выполнении операции. Но для этого требуется, чтобы все страницы (блоки), заблокированные при выполнении операции, оставались заблокированными до конца этой операции.

Тем самым, для подсистемы управления буферным пулом операции низкого уровня являются почти тем же, чем являются транзакции для подсистемы управления транзакциями.

70.5 Физическая синхронизация

Достаточным условием (хотя и слишком жестким) корректного выполнения операций является соблюдение двухфазного протокола синхронизационных блокировок над страницами буферного пула в пределах операций.

Это условие не является необходимым. Каждую операцию уровня ядра СУБД можно разбить на последовательность «микроопераций» и потребовать соблюдения двухфазного протокола синхронизационных блокировок в пределах микроопераций.

Например, операцию `INSERT` уровня ядра (вставки кортежа в таблицу) можно разбить на следующие микрооперации:

1. Нахождение блока данных для вставки;
2. Вставка кортежа в найденный блок;
3. Обновление индекса 1;
4. ...
5. Обновление индекса n, где n – число индексов, определённых для данной таблицы.

Общий принцип состоит в том, что в пределах одной микрооперации блокируются все блоки базы данных, которые обязаны быть изменены согласованным образом.

70.6 Физическая синхронизация

Потребность в блокировке страниц (блоков базы данных) делает практически невозможной разработку подсистемы управления буферным пулом с гарантированным отсутствием синхронизационных тупиков физического уровня. Следовательно, данная подсистема должна уметь их распознавать и разрушать.

Для борьбы с физическими синхронизационными тупиками можно обойтись без построения и анализа графа ожидания. Число разных микроопераций в ядре ограничено и не слишком велико. Для каждой микрооперации известно максимальное число физических блокировок (и операций записи/чтения блоков базы данных), которые могут потребоваться при её выполнении. Если известно максимальное число транзакций, которые могут одновременно выполняться в СУБД, то можно оценить максимальное время ожидания доступа к буферной странице, если отсутствует синхронизационный тупик.

Тогда, если при выполнении микрооперации транзакция не дожидается разрешения доступа к запрошенному блоку базы данных за установленное время, она откатывает недовыполненную микрооперацию с восстановлением состояния изменённых буферных страниц, снимает все установленные физические блокировки и выполняет микрооперацию заново.

71 Протокол Write Ahead Log и его связь с буферизацией.

71.1 Протокол упреждающей записи в журнал

Эта проблема не возникает при индивидуальных откатах транзакций, поскольку в этих случаях содержимое основной памяти не утрачено, и при восстановлении можно пользоваться содержимым как буфера журнала, так и буферных страниц базы данных.

Но если произошёл мягкий сбой, и содержимое буферов утрачено, то для проведения восстановления базы данных необходимо иметь некоторое согласованное состояние журнала и базы данных во внешней памяти.

Основным принципом согласованной политики выталкивания буфера журнала и буферных страниц базы данных является то, что запись об изменении объекта базы данных должна оказаться во внешней памяти журнала раньше, чем сам изменённый объект окажется во внешней памяти базы данных.

Соответствующий протокол журнализации (и управления буферизацией) называется **WAL** (Write Ahead Log, «пиши сначала в журнал») и состоит в том, что если требуется вытолкнуть во внешнюю память буферную страницу, содержащую изменённый объект базы данных, то перед этим нужно гарантировать выталкивание во внешнюю память журнала буферной страницы журнала, содержащей запись об изменении этого объекта.

71.1.1 Протокол упреждающей записи в журнал

При следовании протоколу WAL:

- Если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции.

Обратное неверно, т.е.:

- Если во внешней памяти журнала содержится запись о некоторой операции изменения объекта базы данных, то сам изменённый объект может отсутствовать во внешней памяти базы данных.

Дополнительное условие на выталкивание буферов накладывается тем требованием, что каждая успешно завершённая транзакция должна быть реально зафиксирована во внешней памяти.

- Какой бы сбой ни произошёл, система должна быть в состоянии восстановить состояние базы данных, содержащее результаты всех транзакций, зафиксированных до момента сбоя.

71.1.2 Протокол упреждающей записи в журнал

Самым простым решением было бы выталкивание буфера журнала, за которым следует массовое выталкивание буферов страниц базы данных, изменявшихся данной транзакцией. Довольно часто так и делают, но это вызывает существенные накладные расходы при выполнении операции фиксации транзакции.

Оказывается, что минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является:

- Выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы данных этой транзакцией;

- При этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце транзакции.

Рассмотрим теперь, как можно выполнять операции восстановления базы данных в различных ситуациях, если в системе поддерживается общий для всех транзакций журнал с общей буферизацией записей, поддерживаемый в соответствии с протоколом WAL.

72 Физически согласованное состояние базы данных. Схема восстановления базы данных после мягкого сбоя до точки физической согласованности.

Одной из основных проблем при восстановлении после мягкого сбоя является то, что одна логическая операция изменения базы данных может изменять несколько физических блоков базы данных, например, блок данных и несколько блоков индексов. Разные блоки буферизуются и выталкиваются независимо, что может привести к несогласованности внешней памяти после мягкого сбоя, когда часть блоков соответствует объекту до изменения, а часть — после.

Состояние внешней памяти базы данных называется физически согласованным, если наборы страниц всех объектов согласованы, то есть соответствуют состоянию любого объекта или до его изменения, или после. Будем считать, что точки физической согласованности базы данных отмечаются в журнале. Тогда восстановление после мягкого сбоя имеет следующий вид, в зависимости от отношения времени транзакции и точки согласованности.

1. Если транзакция завершилась до точки согласованности, то ничего делать не надо, так как все её результаты гарантированно находятся во внешней памяти.

2. Если транзакция началась после точки согласованности и не успела завершиться, то тоже ничего делать не надо, так как её результатов во внешней памяти нет никаких.

3. Если такая транзакция завершилась успела, то нужно повторить для неё все операции в прямом направлении (раз транзакция завершилась, то в журнале есть все нужные записи).

4. Если транзакция началась до точки согласованности и завершилась до сбоя, то нужно повторно выполнить операции, произведённые после точки согласованности, потому что во внешней памяти есть только записи операций, произведённых до точки согласованности (а так как транзакция завершилась, то у нас есть журнал со всеми нужными данными).

5. Если транзакция началась до точки согласованности, но не успела завершиться до сбоя, то нужно отменить (выполнить в обратном порядке) операции, произведённые до сбоя (они у нас есть из-за точки согласованности, и транзакция должна быть отменена, так как не успела завершиться).

73 Способы восстановления физически согласованного состояния (теневой механизм и журнализация постраничных изменений).

Одной из основных проблем при восстановлении после мягкого сбоя является то, что одна логическая операция изменения базы данных может изменять несколько физических блоков базы данных, например, блок данных и несколько блоков индексов. Разные блоки буферизуются и выталкиваются независимо,

что может привести к несогласованности внешней памяти после мягкого сбоя, когда часть блоков соответствует объекту до изменения, а часть — после.

Состояние внешней памяти базы данных называется физически согласованным, если наборы страниц всех объектов согласованы, то есть соответствуют состоянию любого объекта или до его изменения, или после.

Для обеспечения наличия точек физической согласованности используются два основных подхода: использование теневого механизма и журнализация постраничных изменений базы данных.

73.1 Теневой механизм

Теневой механизм был изначально предложен для поддержания целостности файлов при мягких сбоях. Для файла, представимого как набор блоков, при модификации любого блока во внешней памяти выделяется новый блок, где эта модификация происходит. Текущая таблица отображений логических блоков в физические изменяется, а теневая остаётся прежней. Если происходит сбой, во внешней памяти остаётся теневая копия, содержащая состояние файла до его открытия. Для восстановления файла достаточно прочитать заново его теневую таблицу отображения.

В контексте базы данных теневой механизм используется так: периодически выполняются операции установки точки физической согласованности, при которых завершаются все логические операции и выталкиваются все страницы буферов, содержимое которых было изменено. Теневая таблица отображения заменяется текущей (текущая становится теневой).

Здесь есть проблема, связанная с необходимостью атомарности операции этой замены, чтобы в любой момент времени теневая таблица отображения была корректной. Если в процессе операции возникнёт мягкий сбой, то и текущая таблица будет утрачена, и теневая повреждена. Чтобы этого не произошло, во внешней памяти поддерживаются две области хранения таблицы отображений, и область начинает использоваться только после полного изменения. Флаг, показывающий, какую область надо хранить, хранится в одном блоке, а запись на внешний диск одного блока считается атомарной, так что здесь проблем нет.

Недостаток теневого метода: производится очень большой перерасход внешней памяти.

73.2 Журнализация постраничных изменений

Журнализация постраничных изменений производится вместе с логической журнализацией операций изменения базы данных. После мягкого сбоя первый этап состоит в постраничном откате недовыполненных логических операций.

Чтобы распознать, нуждается ли страница внешней памяти базы данных в восстановлении, при выталкивании страницы из буферного пула в неё помещается номер последней записи о постраничном изменении этой страницы, который же запоминается в самой записи. Тогда, чтобы понять, надо ли применять данную запись о постраничном изменении, достаточно сравнить номер, содержащийся в блоке, с номером, содержащимся в журнальной записи. Если в блоке номер меньше, то это значит, что страница вытолкнута во внешнюю память не была, и применять запись не требуется.

В подходе с журнализацией постраничных изменений имеются два поднаправления:

1. **Поддержка общего журнала логических и страничных операций**: имеет более сложную структуру и больший размер.
2. **Поддержка отдельного журнала постраничных изменений** (физического):
    - Логический и физический журналы имеют разную природу:
        - Логический журнал должен поддерживать выполнение операций как в прямом, так и в обратном порядке;
        - Физический журнал поддерживает только обратный порядок.
    - Логический журнал начинает заполняться заново только после резервного копирования базы данных или архивирования журнала, до этого времени он растёт.
    - Предельный размер журнала определяется администратором базы данных и должен согласовываться с интервалом времени, через которое производится резервное копирование.
    - Физический журнал существует сравнительно недолго и занимает меньше места.

При установлении точки физической согласованности:

- Прекращают инициироваться новые логические операции;
- После завершения всех выполняемых происходит выталкивание во внешнюю память модифицированных страниц буферного пула;
- Затем во внешнюю память логического журнала выталкивается запись о точке физической согласованности;
- Если всё прошло успешно, то физический журнал пишется заново и можно работать дальше.
- Если выталкивание записи о точке физической согласованности прошло неуспешно, это воспринимается как мягкий сбой.

74 Восстановление базы данных после жесткого сбоя. Архивация базы данных и журнала.

74.1 Журнал изменений

Журнал изменений — это специальный набор данных, хранящий последовательность записей обо всех изменениях базы данных. Потенциальное переполнение логического журнала регулируется так: устанавливаются жёлтая и красная зоны. При достижении жёлтой зоны запрещается образование новых транзакций. Если работавшие транзакции не успеют завершиться до достижения красной зоны, то они будут откачены. После завершения транзакций (или отката) производится архивация базы данных или журнала.

74.2 Архивация базы данных и журнала

Самым простым способом архивирования является создание копии базы данных по указанию администратора или при переполнении журнала. Но можно выполнять архивацию базы данных реже, чем переполняется журнал, архивируя журнал вместо базы данных. Тогда при восстановлении достаточно иметь исходную архивную копию, последовательность архивных журналов и последний логический журнал. В лоб выполнять все имеющиеся в журналах операции долго, но их можно сильно оптимизировать, выделив для каждого объекта последовательности журнальных записей в хронологическом порядке и заменив их одной записью-результатом. Также можно совместить несколько последовательных журналов, полных или сжатых. То есть остаётся исходная архивная копия, один сжатый архивный журнал и последний логический журнал. При этом работа по сжатию журналов может выполняться по ходу их создания, а не в сам момент восстановления.

При жестком сбое теряются данные из внешней памяти. Очевидно, в таком случае журнала изменений недостаточно, нужна архивная копия базы данных. При восстановлении сначала копируется база данных из архивной копии, затем для всех завершившихся транзакций их операции выполняются (более точно — выполняются все операции из журнала и откат для тех транзакций, которые не успели завершиться). После этого получается хронологически последнее до момента жесткого сбоя логически согласованное состояние базы данных.

При некоторой дисциплине выполнения логических операций базы данных при восстановлении можно выполнять операции из журнала последовательно, не обращая внимания на принадлежность их разным транзакциям. В частности, если сериализация транзакций основана на блокировках объектов, то запись в буфере логического журнала об изменении объекта появляется только после удовлетворения блокировки, и только после записи выполняется сама операция изменения.

Вообще говоря, можно восстановиться после жесткого сбоя не только до последнего логически согласованного состояния, но и до момента сбоя, то есть получить возможность продолжать незаконченные транзакции. Но так обычно не делают, потому что восстановление после жесткого сбоя — долгая операция.

Если журнал операций утрачен, то всё, что можно сделать — вернуться к архивной копии.

75 Объектно-ориентированная модель данных. Её структурная, манипуляционная и целостная части.

75.1 Объектно-ориентированная модель

В объектно-ориентированной модели данных база данных — это набор объектов (контейнеров данных) произвольного типа. В этом заключается её принципиальное отличие от SQL-ориентированной и реляционной моделей данных, где база данных представляет собой набор именованных контейнеров данных одного родового типа: таблиц или отношений соответственно.

В объектно-ориентированной модели данных вводятся две разновидности типов: литеральные и объектные типы. Экземпляры литеральных типов не име-

ют собственного идентификатора и не могут самостоятельно храниться в базе данных. Экземпляры объектных типов идентифицируются и хранятся.

76 Объектные расширения языка SQL. Возможные подходы к объектно-реляционному отображению без использования объектных расширений SQL.

76.1 Объектные расширения SQL

К объектным расширениям языка SQL (SQL:1999, SQL:2003) относятся:

- Структурные типы данных, определяемые пользователем;
- Типизированные таблицы;
- Типизированные представления;
- Ссылочные типы.

Кроме того, в стандартах SQL:1999, SQL:2003 появились типы данных, не относящиеся к объектным расширениям, но позволяющие хранить в таблицах неатомарные значения, что также относится к решению проблемы «Impedance Mismatch».

76.2 Типы данных, позволяющие хранить неатомарные значения

Типы данных, не относящиеся к ОО расширениям, но позволяющие хранить неатомарные значения:

- **Массивы (1999)**: `datatype ARRAY [max_cardinality] (тип_элемента, индексация от 1)`;
- **Мультимножества (2003)**: `datatype MULTISET`;
- **Анонимные строчные типы (1999)**: `ROW(field1, ..., fieldN)`, где `field ::= name datatype options`;
- **Многомерные массивы (2019)**: `datatype MDARRAY [dimension_list]`, где `dimension ::= optional_axis_name(min_limit : max_limit)` [?].

Тип данных, позволяющий хранить неатомарные значения:

- **datatype** — тип элемента, любой допустимый в SQL тип данных, кроме самого конструируемого типа коллекции.

76.3 Типы данных, определяемые пользователем

- **Индивидуальные типы**: `CREATE TYPE UDT_name AS base_type_name FINAL [method_specification_list];`
- **Структурные типы**: `CREATE TYPE UDT_name [UNDER UDT_name] AS (attribute_definit [INSTANTIABLE | NOT INSTANTIABLE] { FINAL | NOT FINAL } [reference_type_specif [method_specification_list]`

Индивидуальный тип — именованный тип данных, основанный на единственном предопределённом типе. Индивидуальный тип не наследует от своего опорного типа набор операций над значениями (требуется явное приведение индивидуального типа к его базовому типу либо в самом типе нужно явно определить операцию).

Структурный тип данных — именованный тип данных, включающий один или более атрибутов любого из допустимых в SQL типов данных (включая другие структурные типы, коллекции, анонимные строчные типы). Дополнительные механизмы: наследование от ранее определённого структурного типа (только одиночное), определяемые пользователями методы (поведение структурного типа), ссылки.

76.4 Типизированные таблицы

```
CREATE TABLE typed_table_name OF UDT_name [UNDER typed_table_name] [(typed_table_el
typed_table_element ::= constraint_definition | self-ref_column_definition | column
```

Типизированные таблицы:

- При определении типизированной таблицы указывается ранее определённый структурный тип, и если в нём содержится n атрибутов, то в таблице образуется $n + 1$ столбец, дополнительный (первый) столбец называется самоссылающимся и содержит типизированные уникальные идентификаторы строк, которые могут генерироваться системой при вставке строк в типизированную таблицу, явно указываться пользователями или состоять из комбинации значений других столбцов.
- Можно определить подтаблицу типизированной таблицы, если структурный тип подтаблицы является непосредственным подтипом структурного типа супертаблицы.

Типизированные представления:

```
CREATE VIEW typed_view_name OF UDT_name UNDER base_table_name AS query_expression
```

В случае типизированных представлений выражение запроса должно основываться на единственной типизированной таблице (базовой или представляемой), при этом базисная таблица и определяемое представление должны быть ассоциированы с одним и тем же структурным типом.

76.5 Ссылочные типы и значения

Спецификация ссылки в UDT:

- `REF IS SYSTEM GENERATED`
- `REF USING predefined_type`

Определение «самоссылающегося» столбца:

- `REF IS column_name SYSTEM GENERATED`
- `REF IS column_name USER GENERATED`
- `REF USING (attribute_list)`
- `REF IS DERIVED`

Типом «самоссылающегося» столбца является ссылочный тип, ассоциированный со структурным типом типизированной таблицы. Способ генерации значений ссылочного типа указывается при определении соответствующего структурного типа (супертипа) и подтверждается при определении типизированной таблицы (супертаблицы).

Определение ссылочного типа на структурный тип (при типизации столбцов традиционных таблиц, полей строчных типов и атрибутов структурных типов):

```
REF(UDT_name) [SCOPE typed_table_name REFERENCES ARE [NOT] CHECKED [ON DELETE actio
```

Если раздел `SCOPE` отсутствовал в определении атрибута структурного типа, то его можно добавить в конструкции `column_options` для соответствующего столбца типизированной таблицы.

С типизированной таблицей можно обращаться, как с традиционной таблицей, считая, что у неё имеются неявно определённые столбцы (атрибуты структурного типа), а можно относиться к строкам типизированной таблицы, как к объектам соответствующего структурного типа, OID которых содержатся в «самоссылающемся» столбце.

76.6 Основные подходы к объектно-реляционному отображению

1. **Использование OO API (JDBC, ADO.NET)**:
   - Обеспечивают доступ к реляционным данным и их извлечение в форме, более привлекательной для разработчиков объектно-ориентированных приложений (в виде объектов).
2. **Реляционная модель как основа, объекты подстраиваются под неё**:
   - Используются известные методики преобразования, например, преобразование ER-модели в реляционную.
   - **BLOB-стратегия**: упаковка части атрибутов в BLOB, за кодирование и декодирование которого отвечает middleware.
   - Источник: Семенов, Морозов, Порох [**?**].

3. **Сохранение метаданных в БД**:

   - В БД сохраняются не только состояния объектов, но и метаданные, описывающие их структуру (определения классов).
   - Middleware обеспечивает сборку объектных данных и их запись по нескольким таблицам с использованием метаданных.
   - Источник: Klein, Stonis, Jankauskas [**?**].

77 Истинная реляционная модель данных. Её структурная, манипуляционная и целостная части.

77.1 Истинная реляционная модель данных

Ключевая идея третьего манифеста — чтобы достичь требуемой объектной функциональности, не надо абсолютно ничего делать с реляционной моделью; на основе идей Э. Кодда можно реализовать СУБД, обеспечивающие возможности по части представления и хранения данных произвольно сложной структуры, не меньшие тех, которые обеспечивают объектные и SQL-ориентированные СУБД.

Основное препятствие — тезис Кодда о нормализации отношений (1НФ): в реляционной базе данных должны содержаться только отношения, атрибуты которых определены на «доменах, элементы которых являются атомарными (не составными) значениями».

К. Дейт: «Я согласен с Коддом, что желательно оставаться в рамках логики первого порядка, если это возможно. В то же время я отвергаю идею "атомарных значений по крайней мере, в смысле абсолютной атомарности. В Третьем манифесте мы допускаем наличие доменов, содержащих значения произвольной сложности. Они могут быть даже отношениями. Тем не менее, мы остаёмся в рамках логики первого порядка.»

77.2 Типы данных истинной реляционной модели

Три категории типов данных:

1. **Скалярные типы**: инкапсулированные типы, реальная внутренняя структура которых скрыта от пользователей. Предлагаются механизмы определения новых скалярных типов и операций над ними. Типом атрибута определяемого скалярного типа может являться любой определённый к этому моменту скалярный тип, кортежный тип или тип отношения.

   Некоторые базовые скалярные типы данных должны быть предопределены в системе. В число этих типов должен входить тип **truth value** (булевский тип) с двумя значениями `true` и `false`.

2. **Кортежные типы**: определяются с помощью генератора типа `TUPLE` с указанием множества пар `<имя_атрибута, тип_атрибута>` (заголовка кортежа). Типом атрибута кортежного типа может являться любой определённый к этому моменту скалярный тип, любой кортежный тип и тип отношения. Значением кортежного типа является кортеж, представляющий собой множество триплетов `<имя_атрибута, тип_атрибута, значение_атрибута>`, которое соответствует заголовку кортежа этого типа.

3. **Типы отношений**: определяются с помощью генератора типа `RELATION` с указанием некоторого заголовка кортежа. Значением типа отношения является заголовок отношения, совпадающий с заголовком кортежа этого типа отношения, и тело отношения, представляющее собой множество кортежей, соответствующих этому заголовку.

   Кортежные типы и типы отношений не являются инкапсулированными: имеется возможность прямого доступа к атрибутам. Для всех разновидностей типов данных поддерживается модель множественного наследования, позволяющая определять новые типы данных на основе уже определённых типов.

77.3 База данных в истинной реляционной модели

База данных — набор долговременно хранимых именованных переменных отношений, каждая из которых определена на некотором типе отношения. При таких определениях значениями атрибутов отношения могут быть не только значения скалярных типов, но и кортежи, и другие отношения. Третий манифест: «Каждый кортеж в отношении R содержит в точности одно значение v для каждого атрибута A в заголовке отношения H. Иными словами, R находится в первой нормальной форме, 1NF».

77.4 Объектные свойства

- **Инкапсуляция скалярных типов**:
  - Определение операций над типом отделено от определения самого типа.
  - Для скалярного типа определяются несколько представлений:
    - Реальное, доступно только разработчику типа;
    - Возможные, доступны всем.
  - Для инициализации представлений вводится специальная операция **selector**.
- **Наследование**:
  - Объединение множеств экземпляров всех подтипов даёт множество экземпляров супертипа (у супертипа нет собственных экземпляров).
  - Используется модель “inheritance by constraint” (к подтипу относятся экземпляры супертипа, удовлетворяющие заданному ограничению — новый атрибут можно добавить только в том случае, если он выражается через атрибуты супертипа).
  - Для одиночного наследования действует правило разъединённости (множества подтипов не пересекаются).
  - Если множества подтипов пересекаются, то от них нужно унаследовать новый подтип (множественное наследование), коллизии имен атрибутов разрешаются путём переименования в новом подтипе.
- **Полиморфизм**:

– Позднее связывание (полиморфизм по включению) должно основываться на анализе всех аргументов, а не только первого («связывание по self») как в языках ООП.

77.5 Неопределённые значения в истинной реляционной модели

Подход Дейта:

- Для типа T определим тип T', множество значений которого включает множество всех допустимых значений типа T и ещё одно «специальное значение» ξ.
- Определим следующие операции сравнения: $\xi = \xi \equiv \texttt{true}$, $\xi < \xi \equiv \texttt{false}$, $\xi > \xi \equiv \texttt{false}$, $\xi \neq \xi \equiv \texttt{false}$, t `comp_op` $\xi \equiv \texttt{false}$, ξ `comp_op` $t \equiv \texttt{false}$.
- Только там, где требуются неопределённые значения, вместо значений типа T будем хранить значения типа T'.

Обнаруженный парадокс: «специальное значение» ξ оказывается меньше минимального и больше максимального значения типа T.

Подход Дарвена:

- Пусть в отношении R n атрибутов не могут содержать неопределённых значений, а $n+1$ атрибут — может.
- Тогда в БД следует хранить два отношения: $R1$, состоящее из n атрибутов и хранящее все кортежи, где значение атрибута $n+1$ не определено, и $R2$, состоящее из $n+1$ атрибута и хранящее кортежи, где значения атрибута $n+1$ являются определёнными.
- Для получения отношения R в приложении отношение $R1$ расширяется атрибутом $n+1$, при этом приложение само решает, какие значения прописывать в данный атрибут.
- Далее отношения $R1$ и $R2$ объединяются.
- **Недостаток**: требуется слишком много отношений, если большинство атрибутов могут иметь неопределённые значения (решение практически бессмысленно).

77.6 Манипулирование данными в истинной реляционной модели

Эталонные средства манипулирования данными: реляционная алгебра Кодда, реляционная алгебра A.

Языковые средства — язык запросов D:

- Для выражения запросов используется алгебраический подход;
- Запросы, адресуемые к сложным данным, формулируются более точно, чем на SQL;
- Это же касается сложных операций обновления;

- Язык обладает вычислительной полнотой;
- Язык претендует на то, чтобы стать открытым стандартом и заменить SQL.

77.7 Поддержка целостности данных в истинной реляционной модели

В число обязательных требований истинной реляционной модели входит требование определения хотя бы одного возможного ключа для каждой переменной отношения.

Кроме того, говорится, что «любое условное выражение, которое является (или логически эквивалентно) замкнутой правильно построенной формулой (WFF) реляционного исчисления, должно быть допустимо в качестве спецификации ограничения целостности».

Средства поддержки декларативной ссылочной целостности фигурируют только в разделе рекомендуемых возможностей: «В D следует включить некоторую декларативную сокращённую форму для выражения ссылочных ограничений (называемых также ограничениями внешнего ключа)».

77.8 Реализации истинной реляционной модели

- **Единственное коммерческое решение**:
  - **Dataphor** (также известен как D4) — разработка 1999-2001, с 2001 по 2008 — коммерческий продукт компании Alphora, после её приобретения Database Consulting Group, продукт выпускается под открытой лицензией, текущий релиз в 2018 году, исходный код на C#.
- **Открытые проекты** (университеты и индивидуальные разработчики):
  - **Alf** (Ruby)
  - **Dee** (Python)
  - **DuroDBMS** (множественные языки)
  - **Rel** (Java)
  - **TclRAL** (TCL)