Вопросы к зачету

1. Файл, файловая система. Классификация файловых систем. Основные подходы к защите файловых систем.

С точки зрения прикладной программы **файл** — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса внешней памяти и обеспечение доступа к данным.

Термин **файловая система** (file system) используется для обозначения программной системы, управляющей файлами, и архива файлов, хранящегося во внешней памяти.

Изолированные файловые системы — каждый архив файлов (полное дерево каталогов) целиком располагался на одном дисковом пакете или логическом диске (как на винде), полное имя файла начинается с имени дискового устройства.

Централизованная (как в Multics) — обеспечивалась возможность представлять всю совокупность каталогов и файлов в виде единого дерева. Полное имя файла начиналось с имени корневого каталога, и пользователь не обязан был заботиться об установке на дисковое устройство каких-либо конкретных дисков (то есть один файл мог быть разнесен на диск, дискету и два винта).

Смешанная (*nix) — создается файловая система, а потом еще используется mount для подсоединения других устройств со своими файловыми каталогами.

Защита при многопользовательском режиме: **мандатный** и **дискреционный** подходы. **Мандатный**: каждый пользователь имеет отдельный мандат для работы с каждым файлом или не имеет его, много дополнительной информации.

Дискреционный — каждому зарегистрированному пользователю соответствует пара целочисленных идентификаторов: идентификатор группы, к которой относится пользователь, и его собственный идентификатор. Соответственно, при каждом файле хранится полный идентификатор пользователя (собственный идентификатор плюс идентификатор группы), который создал этот файл, и помечается, какие действия с файлом может производить он сам, какие действия с файлом доступны для остальных пользователей той же группы и что могут делать с файлом пользователи других групп. Для каждого файла контролируется возможность выполнения трех действий: чтение, запись и выполнение. Собственно, так в *nix.

2. СУБД. Основные функции СУБД. Типовая организация современной СУБД.

СУБД (система управления базами данных) — информационная система, которая обеспечивает сторонним программам доступ к структурированным данным, обеспечивая при этом некоторые функции:

- управление логической согласованностью и целостностью данных во внешней памяти;
- управление буферами оперативной памяти (работать напрямую с внешним дисковым устройством долго);
- управление транзакциями (с определенными свойствами, вопрос 3);
- журнализация и восстановление БД после сбоев;
- поддержание языков БД (универсальный язык запросов, который может быть использован из внешней программы и позволит не углубляться во внутреннюю структуру хранения данных, например SQL).

Типовая организация современной СУБД. Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть - **ядро** СУБД (часто его называют Data Base Engine), **компилятор языка БД** (обычно SQL), **подсистему поддержки времени выполнения**, **набор утилит**. В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию.

Основной функцией **компилятора языка БД** является компиляция операторов языка БД в некоторую выполняемую программу. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением подсистемы **поддержки времени выполнения**, представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

В отдельные **утилиты** БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д.

3. Транзакции. Свойства ACID. Сериализация транзакций.

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД.

ACID описывает требования к транзакционным системам. **Атомарность** — транзакция не может быть выполнена частично. **Согласованность** — после транзакции система должна остаться в согласованном состоянии (ограничения ссылок, типов, более сложные ограничения (например, сумма столбца равна заданному числу)). **Изолированность** — другие процессы не должны видеть данные в промежуточном состоянии. **Долговечность** — если пользователь получил подтверждение о выполнении, транзакция не будет отменена.

Под **сериализацией** параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. **Сериальный план** выполнения смеси транзакций — такой план, который приводит к сериализации транзакций.

4. Надежность СУБД. Классификация сбоев. Журнализация. Уровни журнализации. Типичные схемы использования журнала.

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под **надежностью** хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого **аппаратного** или **программного** сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые **мягкие сбои**, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и **жесткие сбои,** характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД.

Уровни журнализации. В разных СУБД изменения БД журнализуются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда — минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Основная идея журнализации. Во всех случаях придерживаются **стратегии "упреждающей" записи** в журнал (так называемого протокола Write Ahead Log - **WAL**). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Типичные схемы использования журнала. Индивидуальный откат транзакции (очевидно)**. При мягком сбое** во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать объекты, модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того, чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти. Этот процесс содержит много тонкостей, связанных с общей организацией управления буферами и журналом. Более подробно мы рассмотрим это в соответствующей лекции.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

5. Ранние дореляционные подходы к организации баз данных.

Наиболее общие характеристики ранних систем (основное выделено):

a. Эти системы активно использовались в течение многих лет, дольше, чем используется какая-либо из реляционных СУБД. На самом деле некоторые из ранних систем используются даже в наше время, накоплены громадные базы данных, и одной из актуальных проблем информационных систем является использование этих систем совместно с современными системами.
b. **Все ранние системы не основывались на каких-либо абстрактных моделях**. Как мы упоминали, понятие модели данных фактически вошло в обиход специалистов в области БД только вместе с реляционным подходом. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у различных конкретных систем.
c. **В ранних системах доступ к БД производился на уровне записей**. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к БД поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом.
d. Можно считать, что уровень средств ранних СУБД соотносится с уровнем файловых систем примерно так же, как уровень языка Кобол соотносится с уровнем языка Ассемблера. Заметим, что при таком взгляде уровень реляционных систем соответствует уровню языков Ада или APL.
e. **Не было оптимизации поиска/доступа**. Навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователя самого производить всю оптимизацию доступа к БД, без какой-либо поддержки системы.
f. После появления реляционных систем большинство ранних систем было оснащено "реляционными" интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

Системы, основанные на инвертированных списках (например, Datacom/DB, Adabas) . База данных, организованная с помощью инвертированных списков, похожа на реляционную БД, но с тем отличием, что хранимые таблицы и пути доступа к ним видны пользователям. При этом:

a. Строки таблиц упорядочены системой в некоторой физической последовательности.
b. Физическая упорядоченность строк всех таблиц может определяться и для всей БД (так делается, например, в Datacom/DB).
c. Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.
d. Общие правила определения целостности БД отсутствуют (возлагается на прикладную программу).
e. Явное оперирование с адресами в памяти
f. операции типа «найти первую запись», «найти следующую запись», те же, только с условиями «найти следующую, для которой значение равно чему-то» или «найти следующую, для которой значение больше чего-то», а также операции «взять», «изменить», «удалить»

Иерархические системы (например, Information Management System (IMS)). Строится что-то типа дерева (нециклический граф), есть запись-предок (отдел), для нее может вводится «связь» с записями-потомками, их может быть несколько (служащие отдела), или одна (руководитель), для дерева БД определен полный порядок обхода — сверху-вниз, слева-направо, взятия, изменения, удаления, гарантируется целостность в том плане, что у каждого потомка есть один родитель.

Сетевые системы (например, Integrated Database Management System (IDMS)) . Расширение иерархического подхода. Тоже «связи», но граф может быть циклическим. Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи. Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

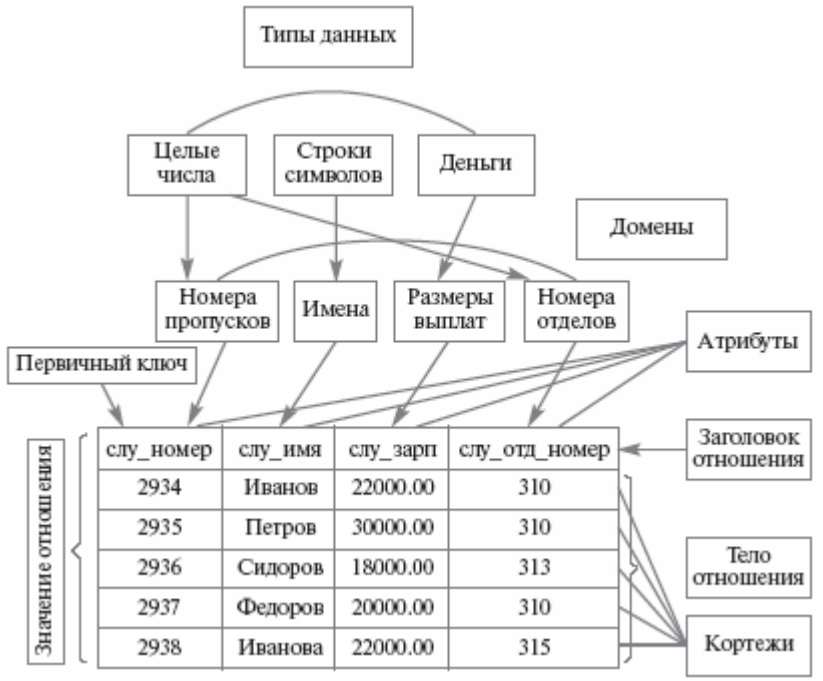
- Каждый экземпляр типа P является предком только в одном экземпляре L;
- Каждый экземпляр C является потомком не более, чем в одном экземпляре L.

-

6. Базовые понятия реляционной модели данных. Ключи. Неопределенные значения. Ссылочная целостность и способы ее поддержания. Атомарность атрибутов и 1НФ.

К числу достоинств реляционного подхода можно отнести:

- **наличие небольшого набора абстракций**, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
- **наличие простого и в то же время мощного математического аппарата**, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных;
- **возможность ненавигационного манипулирования данными** без необходимости знания конкретной физической организации баз данных во внешней памяти.



Тип данных — некая база, как в языках программирования (целые числа, строки), множество значений, над которыми определены операции, а также представление этих чисел при выводе (литералы).

Домены — подтипы, наследованы от типов, имеют некоторые ограничения. Семантически: данные сравнимы, если они в одном домене. Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа.

Схема отношения (заголовок отношения, отношение-схема) Hr — конечное именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}. Степень или "арность" схемы отношения — мощность этого множества. Степень отношения СОТРУДНИКИ равна четырем, то есть оно является 4-арным.

Кортеж, соответствующий данной схеме отношения tr(Hr) — множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, **кортеж — это набор именованных значений заданного типа.**

Отношение (тело отношения) — это множество кортежей, соответствующих одной схеме отношения.

Эволюция схемы базы данных — определение новых и изменение существующих схем отношения.

Реляционная база данных — набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Первичный ключ (формальное определение) — минимальное подмножество из множества атрибутов схемы, что в любое время значение первичного ключа однозначно идентифицирует кортеж, а любое другое собственное множество атрибутов этим свойством не обладает (в набор атрибутов первичного ключа не

должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства — однозначно определять кортеж).

Значения всех атрибутов **атомарны** (точнее скалярны), то есть пользователь не видит структуры, а оперирует только заданными для типов (доменов) операциями. Например, даже если мы присваиваем фотографии, пользователь делает это одной операцией, не задумываясь о том, как данные храняться. являются атомарными

1НФ — первая нормальная форма: таблица находится в первой нормальной форме (1НФ) тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто. **Нормализация** — процесс разбиения на разные подтаблицы, чтобы выполнить эти свойства. Например, таблица по отделам, где в каждом поле стоит список служащих — не нормализована, нормализована, если есть список служащих, и для каждого указан номер отдела. В реляционных базах данных допускаются только нормализованные отношения или отношения, представленные в первой нормальной форме.

Внешний ключ — ссылка на другой кортеж, реализуемая указанием его уникального первичного ключа.

Требование целостности сущности (entity integrity)*:* значение первичного ключа не может быть не определено. Что же такое не определено? Вводится NULL - *неопределенное значение* для любого типа данных. Для булевских вводится значение *unknown*.

Требование целостности по ссылкам (referential integrity): требование целостности по ссылкам, или **требование внешнего ключа** состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать).

Три подхода, для поддержания целостности по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

7. Реляционная алгебра Кодда. Перечислить все операции. Приоритет операций. Замкнутость реляционной алгебры.

Отношения, с которыми идет работа в реляционных БД, являются множествами. У Кодда появилась идея составить на них алгебру. Есть общие операции (теоретико-множественные), а есть специальные (реляционные операции). Везде вместо «отношение» ⇔ «тело отношения» можно читать «таблица», так проще понять.

В состав теоретико-множественных операций входят операции:

Объединение (UNION) двух отношений с одинаковыми заголовками производится отношение, включающее все кортежи, которые входят хотя бы в одно из отношений-операндов (таблица из строк первой и второй таблицы).

П*ересечение* (INTERSECT) двух отношений с одинаковыми заголовками производит отношение, включающее все кортежи, которые входят в оба отношения-операнда.

Р*азность* (MINUS) двух отношений с одинаковыми заголовками, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, которое является вторым операндом.

Д*екартово произведение* (TIMES) двух отношений, пересечение заголовков которых пусто, производится отношение, кортежи которого производятся путем объединения кортежей первого и второго операндов (сливаются столбцы).

Специальные реляционные операции включают:

- **О*граничение*** (WHERE) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.
- **П*роекции*** (PROJECT) отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого являются соответствующими подмножествами кортежей отношения-операнда.
- **С*оединение*** (JOIN) двух отношений по **некоторому условию** образуется результирующее отношение, кортежи которого производятся путем объединения кортежей первого и второго отношений и удовлетворяют этому условию. Заметим также, что в практических реализациях соединение обычно не выполняется именно как ограничение декартого произведения. Имеются более эффективные алгоритмы, гарантирующие получение такого же результата.
- **Р*еляционное деление*** (DIVIDE BY) имеет два операнда — бинарное и унарное отношения. Результирующее отношение состоит из унарных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда.

Дополнительно:

- **П*ереименование*** (RENAME) производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены.
- **П*рисваивание*** (:=) позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Приоритеты:

RENAME ≥ WHERE = PROJECT ≥ TIMES = JOIN = INTERSECT = DIVIDE BY ≥ UNION = MINUS

Алгебра не является замкнутой в математическом смысле (например, TIMES в случае одинаковых заголовков не является отношением). Но применение операции RENAME позволяет использовать алгебру Кодда почти как замкнутую алгебру.

Реляционное деление: объяснение для людей. Пусть заданы два отношения - A с заголовком $\{a_1, a_2, ..., a_n, b_1, b_2, ..., b_m\}$ и B с заголовком $\{b_1, b_2, ..., b_m\}$. Будем считать, что атрибут b_i отношения A и атрибут b_i отношения B не только обладают одним и тем же именем, но и определены на одном и том же домене. Назовем множество атрибутов $\{a_j\}$ составным атрибутом a, а множество атрибутов $\{b_j\}$ - составным атрибутом b. После этого будем говорить о реляционном делении бинарного отношения A(a,b) на унарное отношение B(b).

Результатом деления A на B является унарное отношение C(a), состоящее из кортежей v таких, что в отношении A имеются кортежи <v, w> такие, что множество значений {w} включает множество значений атрибута b в отношении B.
Предположим, что в базе данных сотрудников поддерживаются два отношения: СОТРУДНИКИ (ИМЯ, ОТД_НОМЕР) и ИМЕНА (ИМЯ), причем унарное отношение ИМЕНА содержит все фамилии, которыми обладают сотрудники организации. Тогда после выполнения операции реляционного деления отношения СОТРУДНИКИ на отношение ИМЕНА будет получено унарное отношение, содержащее номера отделов, сотрудники которых обладают всеми возможными в этой организации именами.

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

Подробнее — вопрос 7. Теоретико-множественные операции: объединение (UNION), пересечение (INTERSECT), разность (MINUS) и декартово произведение (TIMES). **Два отношения совместимы по объединению** в том и только в том случае, когда обладают одинаковыми заголовками (схемами отношений). Более точно, это означает, что в заголовках обоих отношений содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене. Тогда к ним можно применять объединение и получить правильное отношение. Иначе – нет. Но если типы одни и те же, а только названия полей отличаются, то можно применить RENAME.

Два отношения совместимы по взятию расширенного декартова произведения в том и только в том случае, если пересечение множеств имен атрибутов, взятых из их схем отношений, пусто. Любые два отношения всегда могут стать совместимыми по взятию декартова произведения, если применить операцию переименования к одному из этих отношений.

9. Реляционная алгебра Кодда. Специальные реляционные операции.

Подробнее — вопрос 7. **Ограничение**, пример: СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 WHERE (СЛУ_ЗАРП > 20000.00 AND (СЛУ_ОТД_НОМ = 310 OR СЛУ_ОТД_НОМ = 315)).

Проекция (возвращает не всю таблицу (отношение), а только часть ее): PROJECT СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 {СЛУ_ОТД_НОМ} – вернет отношение с одним полем «СЛУ_ОТД_НОМ».

Соединение: берем два отношения, строим их декартово произведение, к нему применяем условие. Пример: СЛУЖАЩИЕ JOIN ПРОЕКТЫ WHERE (СЛУ_ЗАРП > ПРО_ЗАРП). В результате будут поля из таблицы СЛУЖАЩИЕ и ПРОЕКТЫ, и строки, которые удовлетворяют условию.

Реляционное деление: есть два отношения, в них есть пересекающиеся поля в заголовке (например, отношение СЛУЖАЩИЕ и ПРОЕКТЫ и домен НОМЕРА_ПРОЕКТОВ). В результате СЛУЖАЩИЕ DIVIDE BY НОМЕРА_ПРОЕКТОВ получим тех служащих, которые участвуют в перечисленных проектах (1 или 2).

Отношение **НОМЕРА_ПРОЕКТОВ**

ПРО_НОМ
1
2

Результат операции **СЛУЖАЩИЕ** DIVIDE BY **НОМЕРА_ПРОЕКТОВ**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП
2934	Иванов	22400.00
2935	Петров	29600.00

10. Реляционная алгебра А. Базовые операции подробно с примерами.

Идеи Кодда были переработаны несколько лет тому назад Дейтом и Дарвеном и была предложены новая **алгебра А** (притом этот алгебра в математическом смысле).

Небольшое повторение. Пусть r — отношение, A — имя атрибута отношения r, T — имя соответствующего типа (т. е. типа или домена атрибута A), v — значение типа T. Тогда:

заголовком Hr отношения r называется множество атрибутов, т.е. упорядоченных пар вида <A, T>. По определению никакие два атрибута в этом множестве не могут содержать одно и то же имя атрибута A;

кортеж tr, соответствующий заголовку Hr, — это множество упорядоченных триплетов вида <A, T, v>, по одному такому триплету для каждого атрибута в Hr;

тело Br отношения r — это множество кортежей tr. Заметим, что (в общем случае) могут существовать такие кортежи tr, которые соответствуют Hr, но не входят в Br.

Заметим, что заголовок — это множество (упорядоченных пар вида <A, T>), тело — это множество (кортежей tr), и кортеж — это множество (упорядоченных триплетов вида <A, T, v>). Элемент заголовка — это атрибут (т. е. упорядоченная пара вида <A,T>); элемент тела — это кортеж; элемент кортежа — это упорядоченный триплет вида <A, T, v>. Любое подмножество заголовка — это заголовок, любое подмножество тела — это тело, и любое подмножество кортежа — это кортеж.

Операции указываем в угловых скобках, чтобы не путать с операциями алгебры логики. Во всех формальных спецификациях exists обозначает *квантор существования*; exists tr означает «существует такой tr, что».

1) **Реляционное дополнение.** Пусть *s* обозначает результат операции **<NOT>** *r*. Тогда:

- Hs = Hr (заголовок результата совпадает с заголовком операнда);
- Bs = {ts : exists tr (tr $\notin$ Br and ts = tr) } (в тело результата входят все кортежи, соответствующие заголовку и не входящие в тело операнда).

Чтобы привести пример использования операции <NOT>, предположим, что в состав домена ДОПУСТИМЫЕ_НОМЕРА_ПРОЕКТОВ, на котором определен атрибут ПРО_НОМ отношения НОМЕРА_ПРОЕКТОВ с рисунка слева, входит всего пять значений {1, 2, 3, 4, 5}. Тогда результат операции <NOT> НОМЕРА_ПРОЕКТОВ будет таким, как показано на рисунке справа.

НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
1
2

<NOT> НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
3
4
5

2) **Удаление атрибута.** Пусть *s* обозначает результат операции *r* **<REMOVE>** A. Для обеспечения возможности выполнения операции требуется, чтобы существовал некоторый тип (или домен) T такой, что <A, T> $\in$ Hr (т. е. в состав заголовка отношения *r* должен входить атрибут A). Тогда:

- Hs = Hr minus {<A, T>}, т. е. заголовок результата получается из заголовка операнда изъятием атрибута A;
- Bs = {ts : exists tr exists v (tr $\in$ Br and v $\in$ T and <A,T,v> $\in$ tr and ts = tr minus {<A,T,v>})}, т. е. в тело результата входят все кортежи операнда, из которых удалено значение атрибута A.

Фактически, берем любую таблицу (отношение), и удаляем один столбец.

3) **Переименование**. Пусть *s* обозначает результат операции *r* **<RENAME>** (A, B). Для обеспечения возможности выполнения операции требуется, чтобы существовал некоторый тип T, такой, что <A, T> $\in$Hr, и чтобы не существовал такой тип T, что <B, T> $\in$ Hr. (Другими словами, в схеме отношения *r* должен присутствовать атрибут A и не должен присутствовать атрибут B.) Тогда:

- Hs = (Hr minus {<A, T>}) union {<B, T>}, т. е. в схеме результата B заменяет A;
- Bs = {ts : exists tr exists v (tr $\in$ Br and v $\in$ T and <A, T, v> $\in$ tr and ts= (tr minus {<A, T, v>}) union {<B, T, v>})}, т. е. в кортежах тела результата имя значений атрибута A меняется на B.

Фактически, берем таблицу, меняем название столбца, получаем результат.

4) **Реляционная конъюнкция**. Пусть *s* обозначает результат операции r_1 **<AND>** r_2. Для обеспечения возможности выполнения операции требуется, чтобы если <A, T1>$\in$Hr$_1$ и <A, T2>$\in$Hr$_2$, то T1=T2. Другими словами, если в двух отношениях-операндах имеются одноименные атрибуты, то они должны быть определены на одном и том же типе (домене). Тогда:

- Hs = Hr_1 union Hr_2, т. е. заголовок результата получается путем объединения заголовков отношений-операндов, как в операциях TIMES и JOIN из предыдущей лекции;
- Bs = { ts : exists tr_1 exists tr_2 ((tr_1 ∈ Br_1 and tr_2 ∈ Br_2) and ts = tr_1 union tr_2)}; обратите внимание на то, что кортеж результата определяется как *объединение кортежей операндов*; поэтому:
  - если схемы отношений-операндов имеют непустое пересечение, то операция <AND> работает как естественное соединение;
  - если пересечение схем операндов пусто, то <AND> работает как расширенное декартово произведение;
  - если схемы отношений полностью совпадают, то результатом операции является пересечение двух отношений-операндов.

Заголовок *r*s является объединением заголовков r_1 и r_2. Тело *s* включает каждый кортеж, соответствующий заголовку *s* и являющийся надмножеством некоторого кортежа из тела r_1 и некоторого кортежа из тела r_2. Смотрите лучше пример =)

5) **Реляционная дизъюнкция**. Пусть *s* обозначает результат операции r_1 **<OR>** r_2. Для обеспечения возможности выполнения операции требуется, чтобы если <A, T1> ∈ Hr_1 и <A, T2> ∈ Hr_2, то должно быть T1 = T2 (одноименные атрибуты должны быть определены на одном и том же типе). Тогда:

- Hs = Hr_1 union Hr_2 (из схемы результата удаляются атрибуты-дубликаты);
- Bs = { ts : exists tr_1 exists tr_2 ((tr_1 ∈ Br_1 or tr_2 ∈ Br_2) and ts = tr_1 union tr_2)}; очевидно, что при этом:
  - если у операндов нет общих атрибутов, то в тело результирующего отношения входят все такие кортежи ts, которые являются объединением кортежей tr_1 и tr_2, соответствующих заголовкам отношений-операндов, и хотя бы один из этих кортежей принадлежит телу одного из операндов;
  - если у операндов имеются общие атрибуты, то в тело результирующего отношения входят все такие кортежи ts, которые являются объединением кортежей tr_1 и tr_2, соответствующих заголовкам отношений-операндов, если хотя бы один из этих кортежей принадлежит телу одного из операндов, и значения общих атрибутов tr_1 и tr_2 совпадают;
  - если же схемы отношений-операндов совпадают, то тело отношения-результата является объединением тел операндов.

Заголовок *s* есть объединение заголовков r_1 и r_2. Тело *s* состоит из всех кортежей, соответствующих заголовку *s* и являющихся надмножеством *либо* некоторого кортежа из тела r_1, *либо* некоторого кортежа из тела r_2. Смотрите лучше пример =)

11. Полнота алгебры А. Определение операций алгебры Кодда через алгебру А.

Более подробно – на стр. 83-92.

Покажем, что Алгебра А является **полной**, т. е. на основе введенных операций выражаются все операции алгебры Кодда, рассмотренной в предыдущей лекции.

К настоящему моменту в состав базовых операций Алгебры А входят операция **<REMOVE>** в качестве аналога операции **PROJECT**, а также операция переименования атрибутов **<RENAME>**. **UNION** является частным случаем операции **<OR>, TIMES, INTERSECT** и **NATURAL JOIN** – частные случаи операции **<AND>.** Нам осталось показать, что через операции Алгебры А выражаются операции взятия разности **MINUS**, ограничения **(WHERE)**, соединения общего вида **(JOIN)** и реляционного деления **(DIVIDE BY)**.

MINUS

r1 MINUS r2 = r1 <AND> <NOT> r2.

WHERE

Для простоты будем считать, что множества значений доменов в БД ограничено значениями, содержащимися в теле отношения. Рассмотрим ограничения с простыми условиями вида(имя отношения – **REL**):

- a **comp_op** const(=) Для выражения условий равенства в терминах алгебры а заводится вспомогательное отношение(**CONST_REL**), содержащее необходимые атрибуты и кортежи. Потом просто берется **REL <AND> CONST_REL**(эквивалентно **REL WHERE** a = const**)**
- a **comp_op** const(>) Опять строим вспомогательное отношение, содержащее необходимые нам данные(они считаются ручками, да) и снова делаем **REL <AND> CONST_REL**(эквивалентно **REL WHERE** a > const**)**
- a **comp_op** const(!=) собственно, все то же самое…
- a **comp_op** b(=) Для проверки этого ограничения(пример – СЛУ_НОМЕР = РУК_НОМЕР) строится следующая конструкция: **<REMOVE>**'ом удаляются все «лишние» атрибуты заголовка, затем переименовываются оставшийся атрибут(он должен быть из одного домена со сравниваемым атрибутом), чтобы совпадали имена у проверяемых значений и берется **<AND>** построенной и исходной таблицы
- a **comp_op** b(!=, >, <,…) Аналогично – используется техника работы со вспомогательными таблицами, удалением ненужных атрибутов и переименованием нужных. Ну и **<AND>,** куда ж без него…

JOIN

Взятие расширенного декартова произведения TIMES является частным случаем операции **<AND>**, ограничение построено, значит можно выразить JOIN**.**

Вот алгоритм в общем случае:

- выполнить над одним из отношений одну или несколько операций <RENAME>, чтобы избавиться от общих имен атрибутов;
- выполнить над полученными отношениями операцию <AND>, производящую расширенное декартово произведение;
- и для полученного отношения выполнить одну или несколько операций <AND> с отношениями-константами(так строится WHERE), чтобы должным образом ограничить его.

DIVIDE BY

Пусть имеются отношения r1{A, B} и r2{B}.

r1 **DIVIDE BY** r2 совпадает с результатом выражения (r1 **PROJEC**T A) **MINUS** (((r2 **TIMES** (r1 **PROJECT** A)) **MINUS** r1) **PROJECT** A) в терминах операций реляционной алгебры Кодда.

Тогда

r1 **DIVIDE BY** r2 =(r1 **<REMOVE>** B**) <AND> <NOT>** (((r2 **<AND>** (r1 **<REMOVE>** B)) **<AND> <NOT>** r1**) <REMOVE>** B) в терминах операций Алгебры А.

12. Реляционная алгебра А. Перечислить базовые операции. Избыточность алгебры А. Сокращение набора операций алгебры А.

Более подробно – вопрос 10 + стр. 92-95.

Базовые операции:

- Реляционное дополнение **<NOT>**
- Удаление атрибута r **<REMOVE>** A
- Операция переименования r **<RENAME>** (A, B)
- Операция реляционной конъюнкции r1 **<AND>** r2
- Операция реляционной дизъюнкции r1 **<OR>** r2

Избыточность алгебры А:

1) Для операций алгебры А **<AND>, <OR>, <NOT>** справедливы те же тождества, что и в классической булевой логике. Это проверяется по определению:

A **AND** B = **NOT** (**NOT** A **OR NOT** B)

A **OR** B = **NOT** (**NOT** A **AND NOT** B)

Мало того, для операций алгебры А существуют аналоги штриха Шеффера и стрелки Пирса:

<sh> (r1, r2) = **<NOT>** r1 **<OR>** **<NOT>** r2

<pi> (r1, r2) = **<NOT>** r1 **<AND>** **<NOT>** r2

Поэтому можно свести набор операций Алгебры А к трем операциям: **<sh>** (или **<pi>**), <**RENAME>** и <**REMOVE>.**

2) Избыточность операции переименования

Для иллюстрации воспользуемся отношением СЛУЖАЩИЕ из рис. ниже. Пусть нам нужен результат операции СЛУЖАЩИЕ <RENAME> (ПРО_НОМ, НОМЕР_ПРОЕКТА) (мы предполагаем, что множество значений домена атрибута ПРО_НОМ ограничено значениями, представленными в теле отношения СЛУЖАЩИЕ). Возьмем бинарное отношение ПРО_НОМ_НОМЕР_ПРОЕКТА , где каждый из кортежей содержит два одинаковых значения номера проекта и в тело отношения входят все значения домена атрибута ПРО_НОМ. Тогда, как показано на рис. , вычисление выражения (СЛУЖАЩИЕ <AND> ПРО_НОМ_НОМЕР_ПРОЕКТА) <REMOVE> (ПРО_НОМ) приводит к желаемому результату.

ПРО_НОМ_НОМЕР_ПРОЕКТА

ПРО_НОМ	НОМЕР_ПРОЕКТА
1	1
2	2

СЛУЖАЩИЕ <AND> **ПРО_НОМ_НОМЕР_ПРОЕКТА**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ	НОМЕР_ПРОЕКТА
2934	Иванов	22400.00	1	1
2935	Петров	29600.00	1	1
2936	Сидоров	18000.00	1	1
2937	Федоров	20000.00	1	1
2938	Иванова	22000.00	1	1
2934	Иванов	22400.00	2	2
2935	Петров	29600.00	2	2
2939	Сидоренко	18000.00	2	2
2940	Федоренко	20000.00	2	2
2941	Иваненко	22000.00	2	2

(**СЛУЖАЩИЕ** <AND> **ПРО_НОМ-НОМЕР_ПРОЕКТА**) <REMOVE> (ПРО_НОМ)

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	НОМЕР_ПРОЕКТА
2934	Иванов	22400.00	1
2935	Петров	29600.00	1
2936	Сидоров	18000.00	1
2937	Федоров	20000.00	1
2938	Иванова	22000.00	1
2934	Иванов	22400.00	2
2935	Петров	29600.00	2
2939	Сидоренко	18000.00	2
2940	Федоренко	20000.00	2
2941	Иваненко	22000.00	2

Тем самым, **можно сократить набор операций Алгебры А до двух операций: <sh> (или <pi>) и <REMOVE>**

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

Сравнение механизмов реляционной алгебры и реляционного исчисления

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка. В основе исчисления лежит понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы.

Приведем пример. Предположим, что мы работаем с базой данных, которая состоит из отношений

СЛУЖАЩИЕ {СЛУ_НОМ, СЛУ_ИМЯ, СЛУ_ЗАРП, ПРО_НОМ}
и
ПРОЕКТЫ {ПРО_НОМ, ПРОЕКТ_РУК, ПРО_ЗАРП}
В отношении ПРОЕКТЫ атрибут ПРОЕКТ_РУК содержит имена служащих, являющихся руководителями проектов, а атрибут ПРО_ЗАРП – среднее значение зарплаты, получаемой участниками проекта. Мы хотим узнать имена и номера служащих, которые являются руководителями проектов со средней заработной платой, превышающей 18000 руб.

Если бы для формулировки такого запроса использовалась реляционная алгебра, то мы получили бы, например, следующее алгебраическое выражение:

(СЛУЖАЩИЕ JOIN ПРОЕКТЫ
WHERE
(СЛУ_ИМЯ = ПРОЕКТ_РУК AND ПРО_ЗАРП > 18000.00))
PROJECT (СЛУ_ИМЯ, СЛУ_НОМ)

Это выражение можно было бы прочитать, например, следующим образом:
1) Выполнить эквисоединение отношений СЛУЖАЩИЕ и ПРОЕКТЫ по условию СЛУ_ИМЯ = ПРОЕКТ_РУК;
2) Ограничить полученное отношение по условию ПРО_ЗАРП > 18000.00;
спроецировать результат предыдущей операции на атрибут СЛУ_ИМЯ, СЛУ_НОМ.

Мы четко сформулировали последовательность шагов выполнения запроса, каждый из которых соответствует одной реляционной операции.

Если же сформулировать тот же запрос с использованием реляционного исчисления то мы получили бы два определения переменных:

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ и
RANGE ПРОЕКТ IS ПРОЕКТЫ

и выражение

СЛУЖАЩИЙ.СЛУ_ИМЯ, СЛУЖАЩИЙ.СЛУ_НОМ WHERE EXISTS (СЛУЖАЩИЙ.СЛУ_ИМЯ = ПРОЕКТ.ПРОЕКТ_РУК AND ПРОЕКТ.ПРО_ЗАРП > 18000.00).

Это выражение можно было бы прочитать, например, следующим образом: выдать значения СЛУ_ИМЯ и СЛУ_НОМ для каждого кортежа служащих такого, что существует кортеж проектов со

значением ПРОЕКТ_РУК, совпадающим со значением СЛУ_НОМ этого кортежа служащих, и значением ПРО_ЗАРП, большим 18000.00.

Во второй формулировке мы указали лишь характеристики результирующего отношения, но ничего не сказали о способе его формирования. В этом случае система сама должна решить, какие операции и в каком порядке нужно выполнить над отношениями СЛУЖАЩИЕ и ПРОЕКТЫ. Обычно говорят, что алгебраическая формулировка является процедурной, т. е. задающей последовательность действий для выполнения запроса, а логическая — описательной (или декларативной), поскольку она всего лишь описывает свойства желаемого результата.

Исчисление кортежей и доменов

В зависимости от того, что является областью определения переменной, различают исчисление кортежей и исчисление доменов. **В исчислении кортежей областями определения переменных являются тела отношений базы данных, т. е. допустимым значением каждой переменной является кортеж тела некоторого отношения. В исчислении доменов областями определения переменных являются домены, на которых определены атрибуты отношений базы данных, т. е. допустимым значением каждой переменной является значение некоторого домена.**

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

Более подробно –стр. 99-103

Исчисление кортежей. Кортежная переменная.

В исчислении кортежей областями определения переменных являются тела отношений базы данных, т. е. допустимым значением каждой переменной является кортеж тела некоторого отношения.

Для определения кортежной переменной используется оператор RANGE. Например, для того чтобы определить переменную СЛУЖАЩИЙ, **областью определения** которой является отношение СЛУЖАЩИЕ, нужно употребить конструкцию

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ

И этого определения следует, что в любой момент времени переменная СЛУЖАЩИЙ представляет некоторый кортеж отношения СЛУЖАЩИЕ. При использовании кортежных переменных в формулах можно ссылаться на значение атрибута переменной. Например, для того, чтобы сослаться на значение атрибута СЛУ_ИМЯ переменной СЛУЖАЩИЙ, нужно употребить конструкцию СЛУЖАЩИЙ.СЛУ_ИМЯ.

Правильно построенные формулы

Правильно построенная формула (Well-Formed Formula, **WFF**) служит для выражения условий, накладываемых на кортежные переменные.

Основой WFF являются простые условия, представляющие собой операции сравнения скалярных значений (значений атрибутов переменных или литерально заданных констант) - например, конструкции
СЛУЖАЩИЙ.СЛУ_НОМ = 2934
СЛУЖАЩИЙ.СЛУ_НОМ = ПРОЕКТ.ПРОЕКТ_РУК

По определению, простое сравнение является WFF, а WFF, заключенная в круглые скобки, представляет собой простое сравнение.

Также если form – WFF, а comp – простое сравнение, то NOT form, comp AND form, comp OR form и IF comp THEN form являются WFF.

Способ реализации

Рассмотрим способ реализации системы, которая сможет по заданной WFF при существующем состоянии базы данных произвести результат: в некотором порядке просмотреть область определения переменной и к каждому очередному кортежу применить условие. Результатом будет то множество кортежей, для которых при вычислении условия производится значение true. Если в WFF используется две переменных, для каждого фиксированного значения первой рассматриваются возможные значения второй. И так далее для любого количества переменных.

Заметим, что описанный выше способ реализации, который приводит к получению области истинности рассмотренной формулы, в действительности является наиболее общим (и зачастую неоптимальным) способом выполнения операций соединения (он называется методом вложенных циклов – nested loops join).

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

Для определения кортежной переменной используется оператор RANGE. Например, для того чтобы определить переменную СЛУЖАЩИЙ, **областью определения** которой является отношение СЛУЖАЩИЕ, нужно употребить конструкцию

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ

Кванторы, свободные и связанные переменные

При построении WFF допускается использование **кванторов** существования (**EXISTS**) и всеобщности (**FORALL**). Если form – это WFF, в которой участвует переменная var, то конструкции **EXISTS** var (form) и **FORALL** var (form) представляют собой WFF. По определению, формула **EXISTS** var (form) принимает значение true в том и только в том случае, если в **области определения** переменной var найдется хотя бы одно значение (кортеж), для которого WFF form принимает значение true. Формула **FORALL** var (form) принимает значение true, если для всех значений переменной var из ее области определения WFF form принимает значение true.

Все переменные, входящие в WFF, при построении которой не использовались кванторы, являются **свободными**. Фактически, это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении WFF получено значение true, то эти значения кортежных переменных могут входить в результирующее отношение. Если же имя переменной использовано сразу после квантора при построении WFF вида EXISTS var (form) или FORALL var (form), то в этой WFF и во всех WFF, построенных с ее участием, var - это **связанная** переменная. На самом деле, правильнее говорить не о свободных и связанных переменных, а о свободных и связанных вхождениях переменных.

Целевые списки

Итак, WFF обеспечивают средства формулировки условия выборки из отношений БД. Чтобы можно было использовать исчисление для реальной работы с БД, требуется еще один компонент, который определяет набор и имена атрибутов результирующего отношения. Этот компонент называется **целевым списком** (target list).

Целевой список строится из целевых элементов, каждый из которых может иметь следующий вид:

- var.attr, где var – имя свободной переменной соответствующей WFF, а attr – имя атрибута отношения, на котором определена переменная var;
- var, что эквивалентно наличию подсписка var.attr1, var.attr2, ..., var.attrn, где {attr1, attr2, ..., attrn} включает имена всех атрибутов определяющего отношения;
- new_name = var.attr; new_name – новое имя соответствующего атрибута результирующего отношения.

Последний вариант требуется в тех случаях, когда в WFF используется несколько свободных переменных с одинаковой областью определения.

Выражением реляционного исчисления кортежей называется конструкция вида **target_list WHERE wff. Значением выражения является отношение, тело которого определяется WFF, а набор атрибутов и их имена - целевым списком.**

16. Исчисление доменов. Основные отличия от исчисления кортежей.

В исчислении доменов областью определения переменных являются не отношения, а домены. Основным формальным отличием исчисления доменов от исчисления кортежей является наличие **дополнительного множества предикатов**, позволяющих выражать так называемые условия членства. Если R – это n-арное отношение с атрибутами a1, a2, ..., an, то условие членства имеет вид R (ai1 : vi1, ai2 : vi2, ..., aim : vim) (m <= n), где vij – это либо литерально задаваемая константа, либо имя доменной переменной. **Условие членства принимает значение true в том и только в том случае, если в отношении R существует кортеж, содержащий указанные значения (v_ij) указанных атрибутов (a_ij).** Если vij – константа, то на атрибут aij накладывается жесткое условие, не зависящее от текущих значений доменных переменных; если же vij – имя доменной переменной, то условие членства может принимать разные значения при разных значениях этой переменной.

Примеры. WFF исчисления доменов
СЛУЖАЩИЕ (СЛУ_НОМ:2934, СЛУ_ИМЯ:'Иванов',
СЛУ_ЗАРП:22400.00, ПРО_НОМ:1)

примет значение true в том и только в том случае, когда в теле отношения СЛУЖАЩИЕ содержится кортеж <2934, 'Иванов', 22400.00, 1>. Соответствующие значения доменных переменных образуют область истинности этой WFF.

WFF
СЛУЖАЩИЕ (СЛУ_НОМ:2934, СЛУ_ИМЯ:'Иванов',
СЛУ_ЗАРП:22400.00, ПРО_НОМ:ПРО_НОМ)

будет принимать значение true для всех комбинаций явно заданных значений и допустимых значений переменной ПРО_НОМ, которые соответствуют кортежам, входящим в тело отношения СЛУЖАЩИЕ

Во всех остальных отношениях формулы и выражения исчисления доменов выглядят похожими на формулы и выражения исчисления кортежей. В частности, формулы могут включать кванторы, и различаются свободные и связанные вхождения доменных переменных.

17. 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.

Подробнее – стр. 124-126, стр.

Будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том, из каких отношений должна состоять БД и какие атрибуты должны быть у этих отношений. Будем приближать схемы отношений к хорошему состоянию путем нормализации – приведения к виду, обладающему определенными хорошими свойствами, в несколько шагов.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF) — **смотри вопрос 6**;
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм состоят в следующем:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей нормальной формы;
- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

Слабо нормализованные модели данных – формы 1НФ или 2НФ. Сильно нормализованные – 3НФ и далее.

OLTP и **OLAP**-системы

Можно выделить некоторые классы систем, для которых больше подходят сильно или слабо нормализованные модели данных.
Сильно нормализованные модели данных хорошо подходят для так называемых **OLTP**-приложений (On-Line Transaction Processing (**OLTP**)- оперативная обработка транзакций). Типичными примерами **OLTP**-приложений являются системы складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег, и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Сами транзакции выглядят относительно просто, например, "снять сумму денег со счета A, добавить эту сумму на счет B". Проблема заключается в том, что, во-первых, транзакций очень много, во-вторых, выполняются они одновременно (к системе может быть подключено несколько тысяч одновременно работающих пользователей), в-третьих, при возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции (не должно быть ситуации, когда деньги сняты со счета A, но не поступили на счет B). Практически все запросы к базе данных в OLTP-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для предоставления пользователям возможности выбора из различных справочников. Большая часть запросов, таким образом, известна заранее еще на этапе проектирования системы. Таким образом, критическим для **OLTP**-приложений является скорость и надежность выполнения коротких операций обновления данных. Чем выше уровень нормализации данных в **OLTP**-приложении, тем оно, как правило, быстрее и надежнее.

Другим типом приложений являются так называемые **OLAP**-приложения (On-Line Analitical Processing (**OLAP**) - оперативная аналитическая обработка данных). Это обобщенный термин, характеризующий принципы построения систем поддержки принятия решений (Decision Support System - DSS), хранилищ

данных (Data Warehouse), систем интеллектуального анализа данных (Data Mining). Такие системы предназначены для нахождения зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей), для проведения анализа "что если…". OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми их электронных таблиц или из других источников данных. Такие системы характеризуются следующими признаками:

- Добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из **OLTP**-приложения).
- Данные, добавленные в систему, обычно никогда не удаляются.
- Перед загрузкой данные проходят различные процедуры "очистки", связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, ошибочны.
- Запросы к системе являются нерегламентированными и, как правило, достаточно сложными. Очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса.
- Скорость выполнения запросов важна, но не критична.

Возвращаясь к проблеме нормализации данных, можно сказать, что в системах OLAP, использующих реляционную модель данных, данные целесообразно хранить в виде слабо нормализованных отношений, содержащих заранее вычисленные основные итоговые данные. Большая избыточность и связанные с ней проблемы тут не страшны, т.к. обновление происходит только в момент загрузки новой порции данных. При этом происходит как добавление новых данных, так и пересчет итогов.

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

Пусть задана **переменная отношения** R, и X и Y являются произвольными подмножествами заголовка R («составными» атрибутами).

В значении переменной отношения R атрибут Y **функционально зависит (**Functional Dependency – FD) от атрибута X в том и только в том случае, если каждому значению X соответствует в точности одно значение Y. В этом случае говорят также, что атрибут X функционально определяет атрибут Y (X является детерминантом (определителем) для Y, а Y является зависимым от X). Будем обозначать это как R.X –> R.Y. В общем случае X, Y – составные.

Пример:

СЛУ_НОМ	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ	ПРОЕКТ_РУК
2934	Иванов	22400.00	1	Иванов
2935	Петров	29600.00	1	Иванов
2936	Сидоров	18000.00	1	Иванов
2937	Федоров	20000.00	1	Иванов
2938	Иванова	22000.00	1	Иванов
2939	Сидоренко	18400.00	2	Иваненко
2940	Федоренко	20400.00	2	Иваненко
2941	Иваненко	22600.00	2	Иваненко

СЛУ_НОМ -> СЛУ_ИМЯ
СЛУ_НОМ -> СЛУ_ЗАРП
СЛУ_НОМ -> ПРО_НОМ
СЛУ_НОМ -> ПРОЕКТ_РУК
ПРО_НОМ –> ПРОЕКТ_РУК

Все вышеперечисленные FD - **инварианты**, или **ограничения целостности** этой переменной отношения. Это значит, что FD порождаются не случайными явлениями в данной переменной, а знаниями из предметной области и выполняются всегда, **даже при изменении отношения** (например, если A — ключ отношения, то для любого B из заголовка этого отношения выполнена функциональная зависимость FD: A->B). Но бывают FD, не являющиеся инвариантами.

Например,

СЛУ_ИМЯ –> СЛУ_НОМ

тоже FD, но является таковой только потому, что нет совпадающих имен, иначе бы не выполнялось. Это не ограничение целостности.

FD A –> B называется **тривиальной**, если A содержит B. Очевидно, что любая тривиальная FD всегда выполняется.

FD A –> C называется **транзитивной**, если существует такой атрибут B, что имеются функциональные зависимости A –> B и B –> C и отсутствует функциональная зависимость C –> A.

.

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

Замыканием множества FD S является множество FD S+, включающее все FD, логически выводимые из FD множества S. Пример: (CN) -> (CName, CZarplata) => CN->CName и CN->CZarplata

Подход к решению проблемы поиска замыкания S+ множества FD S впервые предложил Вильям Армстронг. Им был предложен **набор правил вывода** новых FD из существующих (эти правила обычно называют **аксиомами Армстронга**, хотя справедливость правил доказывается на основе определения FD).

- если B содержится в A, то A –> B (**рефлексивность**);
- если A–> B, то AC –> BC (**пополнение**);
- если A –> B и B –> C, то A –> C (**транзитивность**).

Истинность **первой аксиомы** Армстронга следует из того, что при B содержится в A FD A->B является тривиальной.

Справедливость **второй аксиомы** докажем от противного. Предположим, что FD AC->BC не соблюдается. Это означает, что в некотором допустимом теле отношения найдутся два кортежа t1 и t2, такие, что t1 {AC} = t2 {AC} (a), но t1 {BC} <> t2 {BC} (b) (здесь t {A} обозначает проекцию кортежа t на множество атрибутов A). По аксиоме рефлексивности из равенства (a) следует, что t1 {A} = t2 {A}. Поскольку имеется FD A->B, должно соблюдаться равенство t1 {B} = t2 {B}. Тогда из неравенства (b) следует, что t1 {C} <> t2 {C}, что противоречит наличию тривиальной FD AC->C. Следовательно, предположение об отсутствии FD ACBC не является верным, и справедливость второй аксиомы доказана.

Аналогично докажем истинность **третьей аксиомы** Армстронга. Предположим, что FD A->C не соблюдается. Это означает, что в некотором допустимом теле отношения найдутся два кортежа t1 и t2, такие, что t1 {A} = t2 {A}, но t1 {C} <> t2 {C}. Но из наличия FD AB следует, что t1 {B} = t2 {B}, а потому из наличия FD B->C следует, что t1 {C} = t2 {C}. Следовательно, предположение об отсутствии FD A->C не является верным, и справедливость третьей аксиомы доказана.

Можно доказать, что система правил вывода Армстронга **полна и совершенна** (sound and complete) в том смысле, что для данного 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. Тем не менее**, Дейт** по практическим соображениям предложил расширить базовый набор правил вывода еще **пятью правилами**:

- A –> A (самодетерминированность) – прямо следует из правила (1);
- если A –> BC, то A –> B и A –> C (декомпозиция) – из правила (1) следует, что BC –> B; по правилу (3) A –> B; аналогично, из BC –> C и правила (3) следует A –> C;
- если A –> B и A –> C, то A –> BC (объединение) – из правила (2) следует, что A –> AB и AB –> BC; из правила (3) следует, что A –> BC;
- если A –> B и C –> D, то AC –> BD (композиция) – из правила (2) следует, что AC –> BC и BC –> BD; из правила (3) следует, что AC –> BD;
- если A –> BC и B –> D, то A –> BCD (накопление) – из правила (2) следует, что BC –> BCD; из правила (3) следует, что A –> BCD.

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

Пусть заданы отношение R, множество Z атрибутов этого отношения (подмножество заголовка R, 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 –> Y входит в S+.

Алгоритм вычисления Z+

```
K := 0; Z[0] := Z;
DO
   K := K+1;
   Z[K] := Z[K-1];
   FOR EACH FD A→B IN S DO
      IF A⊆Z[K] THEN Z[K] := (Z[K] UNION B) END DO;
UNTIL Z[K] = Z[K-1];
Z+ := Z[K];
```

Пусть для примера имеется отношение с заголовком {A, B, C, D, E, F} и заданным множеством FD S = {A –> D, AB –> E, BF –> E, CD –> F, E –> C}. Пусть требуется найти {AE}+ над S.

На первом проходе тела цикла DO Z[1] равно AE. В теле цикла FOR EACH будут найдены FD A –> D и E –> C, и в конце цикла Z[1] станет равным ACDE. На втором проходе тела цикла DO при Z[2], равном ACDE, в теле цикла FOR EACH будет найдена FD CD –> F, и в конце цикла Z[2] станет равным ACDEF. Следующий проход тела цикла DO не изменит Z[3], и Z+ ({AE}+) будет равно ACDEF.

Алгоритм построения замыкания множества атрибутов Z над заданным множеством FD S помогает легко установить, входит ли заданная FD Z –> B в замыкание S+. Очевидно, что необходимым и достаточным условием для этого является вхождение составного атрибута B в замыкание Z.

Суперключом отношения R называется любое подмножество K заголовка R, включающее, по меньшей мере, хотя бы один возможный ключ R.

Одно из следствий этого определения состоит в том, что подмножество K заголовка отношения R является суперключом тогда и только тогда, когда для любого атрибута A (возможно, составного) заголовка отношения R выполняется FD K –> A. В терминах замыкания множества атрибутов K является суперключом тогда и только тогда, когда K+ совпадает с заголовком R.

21. Покрытие множества FD. Эквивалентные покрытия. Минимальное множество FD. Примеры. Алгоритм построения минимального эквивалентного множества. Минимальное покрытие множества функциональных зависимостей.

Множество FD S2 называется **покрытием** множества FD S1, если любая FD, выводимая из S1, выводится также из S2.

Легко заметить, что S2 является покрытием S1 тогда и только тогда, когда S1+ принадлежит S2+.

Два множества FD S1 и S2 называются **эквивалентными**, если каждое из них является покрытием другого, т. е. S1+ = S2+.

Множество FD S называется **минимальным** в том и только в том случае, когда удовлетворяет следующим свойствам:

- правая часть любой FD из S является множеством из одного атрибута (простым атрибутом);
- детерминант каждой FD из S обладает свойством минимальности; это означает, что удаление любого атрибута из детерминанта приводит к изменению замыкания S+, т. е. порождению множества FD, не эквивалентного S;
- удаление любой FD из S приводит к изменению S+, т. е. порождению множества FD, не эквивалентного S.

Пример:

Рассмотрим отношение СЛУЖАЩИЕ_ПРОЕКТЫ {СЛУ_НОМ, СЛУ_ИМЯ, СЛУ_ЗАРП, ПРО_НОМ, ПРОЕКТ_РУК}. Если считать, что единственным возможным ключом этого отношения является атрибут СЛУ_НОМ, то множество FD {СЛУ_НОМ->СЛУ_ИМЯ,
СЛУ_НОМ->СЛУ_ЗАРП,
СЛУ_НОМ->ПРО_НОМ,
СЛУ_НОМ->ПРОЕКТ_РУК} будет минимальным.

С другой стороны, множество FD
{СЛУ_НОМ->(СЛУ_ИМЯ, СЛУ_ЗАРП),
СЛУ_НОМ->СЛУ_ИМЯ,
СЛУ_НОМ->СЛУ_ЗАРП,
СЛУ_НОМ->ПРО_НОМ,
СЛУ_НОМ->ПРОЕКТ_РУК} не является минимальными.

Для любого множества FD S существует (и даже может быть построено) эквивалентное ему минимальное множество S.

Алгоритм построения минимального эквивалентного подмножества:

Приведем общую схему построения S- по заданному множеству FD S:

1) Декомпозиция: если A-> BC, то A->B и A->C , получили новое множество S1.
2) Удаление атрибутов из левой части без изменения замыкания: для каждой FD из S1, детерминант D {D1, D2, …, Dn} которой содержит более одного атрибута, будем пытаться удалять атрибуты Di, получая множество FD S2. Если после удаления атрибута Di S2 эквивалентно S1, то этот атрибут удаляется, и пробуется следующий атрибут. Получили S3.
3) Удаление FD не меняющих S. Для каждой FD f из множества S3 будем проверять эквивалентность множеств S3 и S3 MINUS {f}. Если эти множества эквивалентны, удалим f из множества S3, и в заключение получим множество S4, которое минимально и эквивалентно исходному множеству FD S.

Минимальным покрытием множества FD S называется любое минимальное множество FD S1, эквивалентное S.

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

Считаются правильными такие декомпозиции отношения, которые обратимы, т. е. имеется возможность собрать исходное отношение из декомпозированных отношений без потери информации. Такие декомпозиции называются **декомпозициями без потерь**.

СЛУЖАЩИЕ_ПРОЕКТЫ

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ	ПРОЕКТ_РУК
2934	Иванов	22000.00	1	Иванов
2941	Иваненко	22000.00	2	Иваненко

Декомпозиция (1). Отношения **СЛУЖ** и **СЛУ_ПРО**

СЛУ_НОМ	СЛУ_ИМЯ	СЛУ_ЗАРП
2934	Иванов	22000.00
2941	Иваненко	22000.00

СЛУ_НОМ	ПРО_НОМ	ПРОЕКТ_РУК
2934	1	Иванов
2941	2	Иваненко

Декомпозиция (2). Отношения **СЛУЖ** и **ЗАРП_ПРО**

СЛУ_НОМ	СЛУ_ИМЯ	СЛУ_ЗАРП
2934	Иванов	22000.00
2941	Иваненко	22000.00

СЛУ_ЗАРП	ПРО_НОМ	ПРОЕКТ_РУК
22000.00	1	Иванов
22000.00	2	Иваненко

Рис. 7.3 Две возможные декомпозиции отношения СЛУЖАЩИЕ_ПРОЕКТЫ

Анализ рис. 7.3 показывает, что в случае декомпозиции (1) мы не потеряли информацию о служащих. Вторая декомпозиция не дает возможности получить данные о проекте служащего, поскольку Иванов и Иваненко получают одинаковую зарплату, следовательно, эта декомпозиция приводит к потере информации. Что же привело к тому, что одна декомпозиция является декомпозицией без потерь, а вторая – нет?

Схема этого отношения, естественно (поскольку соединение – естественное), совпадает со схемой отношения СЛУЖАЩИЕ_ПРОЕКТЫ, но в теле появились лишние кортежи, наличие которых и приводит к утрате исходной информации. Интуитивно понятно, что это происходит потому, что в отношении ЗАРП_ПРО отсутствуют функциональные зависимости СЛУ_ЗАРП->ПРО_НОМ и СЛУ_ЗАРП->ПРОЕКТ_РУК, но точнее причину потери информации в данном случае мы объясним несколько позже. Корректность же декомпозиции 1 следует из теоремы Хита:

Теорема Хита. Пусть задано отношение r {A, B, C} (A, B и C, в общем случае, являются составными атрибутами) и выполняется FD AB. Тогда r = (r PROJECT {A, B}) NATURAL JOIN (r PROJECT {A, C}).

СЛУЖАЩИЕ_ОТДЕЛЫ_ПРОЕКТЫ

СЛУ_НОМ	СЛУ_ОТД	ПРО_НОМ
2934	630	1
2941	631	1
2934	630	2
2941	631	2

Декомпозиция.
Отношения СЛУЖ_ОТДЕЛЫ и СЛУЖ_ПРОЕКТЫ

СЛУ_НОМ	СЛУ_ОТД
2934	630
2941	631

СЛУ_НОМ	ПРО_НОМ
2934	1
2941	1
2934	2
2941	2

Рис. 7.4. Результат естественного соединения отношений СЛУЖ и ЗАРП_ПРО

Доказательство.

Прежде всего, докажем, что в теле результата естественного соединения (обозначим этот результат через r1) содержатся все кортежи тела отношения r. Действительно, пусть кортеж {a, b, c} принадлежит r. Тогда по определению операции взятия проекции {a, b} принадлежит (r PROJECT {A, B}) и {a, c} принадлежит (r PROJECT {A, C}). Следовательно, {a, b, c} принадлежит r1. Теперь докажем, что в теле результата естественного соединения нет лишних кортежей, т. е. что если кортеж {a, b, c} принадлежит r1, то {a, b, c} принадлежит r. Если {a, b, c}- принадлежит r1, то существуют {a, b}, принадлежащее (r PROJECT {A, B}), и {a, c} , принадлежащее (r PROJECT {A, C}). Последнее условие может выполняться в том и только в том случае, когда существует кортеж {a, b*, c}= r. Но поскольку выполняется FD AB, то b = b* и, следовательно, {a, b, c} = {a, b*, c}. **Конец доказательства.**

Атрибут B минимально зависит от атрибута A, если выполняется минимальная слева FD AB.

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

Атрибут B минимально зависит от атрибута A, если выполняется минимальная слева FD AB.

Аномалии обновления, возникающие из-за наличия неминимальных FD

Во множество FD отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ входит много FD, в которых детерминантом является не возможный ключ отношения (соответствующие стрелки в диаграмме начинаются не с {СЛУ_НОМ, ПРО_НОМ}, т. е. некоторые функциональные зависимости атрибутов от возможного ключа не являются минимальными). Это приводит к так называемым аномалиям обновления. Под **аномалиями обновления** понимаются трудности, с которыми приходится сталкиваться при выполнении операций добавления кортежей в отношение (INSERT), удаления кортежей (DELETE) и модификации кортежей (UPDATE). Обсудим сначала аномалии обновления, вызываемые наличием FD СЛУ_НОМ->СЛУ_УРОВ (эти аномалии связаны с избыточностью хранения значений атрибутов СЛУ_УРОВ и СЛУ_ЗАРП в каждом кортеже, описывающем задание служащего в некотором проекте).

1) **Добавление кортежей**. Мы не можем дополнить отношение СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ данными о служащем, который в данное время еще не участвует ни в одном проекте (ПРО_НОМ является частью первичного ключа и не может содержать неопределенных значений). Между тем часто бывает, что сначала служащего принимают на работу, устанавливают его разряд и размер зарплаты, а лишь потом назначают для него проект.
2) **Удаление кортежей**. Мы не можем сохранить в отношении СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ данные о служащем, завершившем участие в своем последнем проекте (по той причине, что значение атрибута ПРО_НОМ для этого служащего становится неопределенным). Между тем характерна ситуация, когда между проектами возникают перерывы, не приводящие к увольнению служащих.
3) **Модификация кортежей**. Чтобы изменить разряд служащего, мы будем вынуждены модифицировать все кортежи с соответствующим значением атрибута СЛУ_НОМ. В противном случае будет нарушена естественная FD СЛУ_НОМ->СЛУ_УРОВ (у одного служащего имеется только один разряд).

Возможная декомпозиция

Для преодоления этих трудностей можно произвести декомпозицию переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ на две переменных отношений – СЛУЖ {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП} и СЛУЖ_ПРО_ЗАДАН {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}. На основании теоремы Хита эта декомпозиция является декомпозицией без потерь, поскольку в исходном отношении имелась FD {СЛУ_НОМ, ПРО_НОМ}->СЛУ_ЗАДАН.

На рис. 8.3 показаны диаграммы множеств FD этих отношений, а на рис. 8.4 – их значения.

Рис. 8.3. Диаграммы FD в переменных отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН

Теперь мы можем легко справиться с операциями обновления.

Добавление кортежей. Чтобы сохранить данные о принятом на работу служащем, который еще не участвует ни в каком проекте, достаточно добавить соответствующий кортеж в отношение СЛУЖ.

Удаление кортежей. Если кто-то из служащих прекращает работу над проектом, достаточно удалить соответствующий кортеж из отношения СЛУЖ_ПРО_ЗАДАН. При увольнении служащего нужно удалить кортежи с соответствующим значением атрибута СЛУ_НОМ из отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН.

Модификация кортежей. Если у служащего меняется разряд (и, следовательно, размер зарплаты), достаточно модифицировать один кортеж в отношении СЛУЖ.

Значение переменной отношения СЛУЖ

СЛУ_НОМ	СЛУ_УРОВ	СЛУ_ЗАРП
2934	2	22400.00
2935	3	29600.00
2936	1	20000.00
2937	1	20000.00

Значение переменной отношения СЛУЖ_ПРО_ЗАДАН

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	A
2935	1	B
2936	1	C
2937	1	D
2934	2	D
2935	2	C
2936	2	B
937	2	A

Рис. 8.4. Значения переменных отношений

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

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

Аномалии обновлений, возникающие из-за наличия транзитивных функциональных зависимостей:

Функциональные зависимости переменной отношения СЛУЖ по-прежнему порождают некоторые аномалии обновления. Они вызываются наличием транзитивной FD СЛУ_НОМ->СЛУ_ЗАРП (через FD СЛУ_НОМ->СЛУ_УРОВ и СЛУ_УРОВ->СЛУ_ЗАРП). Эти аномалии связаны с избыточностью хранения значения атрибута СЛУ_ЗАРП в каждом кортеже, характеризующем служащих с одним и тем же разрядом.

1) **Добавление кортежей**. Невозможно сохранить данные о новом разряде (и соответствующем ему размере зарплаты), пока не появится служащий с новым разрядом. (Первичный ключ не может содержать неопределенные значения.)
2) **Удаление кортежей**. При увольнении последнего служащего с данным разрядом мы утратим информацию о наличии такого разряда и соответствующем размере зарплаты.
3) **Модификация кортежей**. При изменении размера зарплаты, соответствующей некоторому разряду, мы будем вынуждены изменить значение атрибута СЛУ_ЗАРП в кортежах всех служащих, которым назначен этот разряд (иначе не будет выполняться FD СЛУ_УРОВ->СЛУ_ЗАРП).

Возможная декомпозиция

Для преодоления этих трудностей произведем декомпозицию переменной отношения СЛУЖ на две переменных отношений – СЛУЖ1 {СЛУ_НОМ, СЛУ_УРОВ} и УРОВ {СЛУ_УРОВ, СЛУ_ЗАРП}. По теореме Хита, это снова декомпозиция без потерь по причине наличия, например, FD СЛУ_НОМ->СЛУ_УРОВ.

На рис. 8.5 показаны диаграммы FD этих переменных отношений, а на рис. 8.6 – их возможные значения.

Рис. 8.5. Диаграммы FD в отношениях СЛУЖ1 и УРОВ

Как видно из рис. 8.6, это преобразование обратимо, т. е. любое допустимое значение исходной переменной отношения СЛУЖ является естественным соединением значений отношений СЛУЖ1 и УРОВ. Также можно заметить, что мы избавились от трудностей при выполнении операций обновления.

Добавление кортежей. Чтобы сохранить данные о новом разряде, достаточно добавить соответствующий кортеж к отношению УРОВ.

Удаление кортежей. При увольнении последнего служащего, обладающего данным разрядом, удаляется соответствующий кортеж из отношения СЛУЖ1, и данные о разряде сохраняются в отношении УРОВ.

Модификация кортежей. При изменении размера зарплаты, соответствующей некоторому разряду, изменяется значение атрибута СЛУ_ЗАРП ровно в одном кортеже отношения УРОВ.

Значение переменной отношения **СЛУЖ1**

СЛУ_НОМ	СЛУ_УРОВ
2934	2
2935	3
2936	1
2937	1

Значение переменной отношения **УРОВ**

СЛУ_УРОВ	СЛУ_ЗАРП
2	22400.00
3	29600.00
1	20000.00

Рис. 8.6. Тела отношений СЛУЖ1 и УРОВ

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

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

Необходимые и достаточные условия независимости проекций отношения обеспечивает **теорема Риссанена**:

Проекции r1 и r2 отношения r являются независимыми тогда и только тогда, когда:

- каждая FD в отношении r логически следует из FD в r1 и r2;
- общие атрибуты r1 и r2 образуют возможный ключ хотя бы для одного из этих отношений.

Атомарным отношением называется отношение, которое невозможно декомпозировать на независимые проекции.

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

Рассмотрим случай, когда у отношения имеется несколько возможных ключей, и некоторые из этих возможных ключей «перекрываются», т. е. содержат общие атрибуты.

Аномалии обновлений, связанные с наличием перекрывающихся возможных ключей

Например, пусть имеется переменная отношения СЛУЖ_ПРО_ЗАДАН1 {СЛУ_НОМ, СЛУ_ИМЯ, ПРО_НОМ, СЛУ_ЗАДАН} с множеством FD, показанным на рис. 8.7.

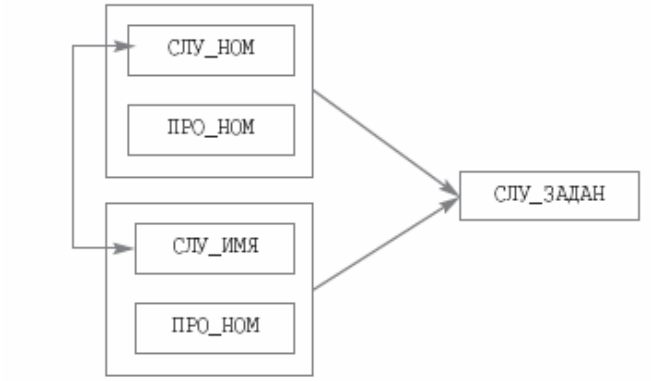


Рис. 8.7. Диаграмма FD отношения СЛУЖ_ПРО_ЗАДАН1

В отношении СЛУЖ_ПРО_ЗАДАН1 служащие уникально идентифицируются как по номерам удостоверений, так и по именам. Следовательно, существуют FD СЛУ_НОМ->СЛУ_ИМЯ и СЛУ_ИМЯ->СЛУ_НОМ. Но один служащий может участвовать в нескольких проектах, поэтому возможными ключами являются {СЛУ_НОМ, ПРО_НОМ} и {СЛУ_ИМЯ, ПРО_НОМ}. На рис. 8.8 показано возможное значение переменной отношения СЛУЖ_ПРО_ЗАДАН1.

СЛУ_НОМ	СЛУ_ИМЯ	ПРО_НОМ	СЛУ_ЗАДАН
2934	Иванов	1	A
2941	Иваненко	2	B
2934	Иванов	2	B
2941	Иваненко	1	A

Рис. 8.8. Возможное значение переменной отношения СЛУЖ_ПРО_ЗАДАН1

Очевидно, что, хотя в отношении СЛУЖ_ПРО_ЗАДАН1 все FD неключевых атрибутов от возможных ключей являются минимальными и транзитивные FD отсутствуют, этому отношению свойственны аномалии обновления. Например, в случае изменения имени служащего требуется обновить атрибут СЛУ_ИМЯ во всех кортежах отношения СЛУЖ_ПРО_ЗАДАН1, соответствующих данному служащему. Иначе будет нарушена FD СЛУ_НОМ->СЛУ_ИМЯ, и база данных окажется в несогласованном состоянии.

Нормальная форма Бойса-Кодда

Причиной отмеченных аномалий является то, что в требованиях 2NF и 3NF не требовалась минимальная функциональная зависимость от первичного ключа атрибутов, являющихся компонентами других возможных ключей. Проблему решает нормальная форма, которую исторически принято называть нормальной формой Бойса-Кодда и которая является уточнением 3NF в случае наличия нескольких перекрывающихся возможных ключей. Переменная отношения находится в нормальной форме Бойса-Кодда (BCNF) в том и только в том случае, когда любая выполняемая для этой переменной отношения нетривиальная и минимальная FD имеет в качестве детерминанта некоторый возможный ключ данного отношения.

Переменная отношения СЛУЖ_ПРО_ЗАДАН1 может быть приведена к BCNF путем одной из двух декомпозиций:

1) СЛУЖ_НОМ_ИМЯ {СЛУ_НОМ, СЛУ_ИМЯ} и
СЛУЖ_НОМ_ПРО_ЗАДАН {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}

2) СЛУЖ_НОМ_ИМЯ {СЛУ_НОМ, СЛУ_ИМЯ} и
СЛУЖ_ИМЯ_ПРО_ЗАДАН {СЛУ_ИМЯ, ПРО_НОМ, СЛУ_ЗАДАН} (FD и значения результирующих переменных отношений выглядят аналогично).

Очевидно, что каждая из декомпозиций устраняет трудности, связанные с обновлением отношения СЛУЖ_ПРО_ЗАДАН1.

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

В переменной отношения R с атрибутами A, B, C (в общем случае, составными) имеется **многозначная зависимость** B от A (AB) в том и только в том случае, когда множество значений атрибута B, соответствующее паре значений атрибутов A и C, зависит от значения A и не зависит от значения C.

Многозначные зависимости обладают интересным свойством **«двойственности»,** которое демонстрирует следующая лемма.

Лемма Фейджина

Если в отношении R {A, B, C} выполняется MVD AB, то выполняется MVD AC. (Записывается как A->>B|C).

Доказательство леммы. Пусть <a,b,c'> принадлежит r и <a,b',c> принадлежит r. Из того, что A->>B следует, что <a,b',c>=<a,b,c> принадлежит r, следовательно, c от b не зависит, но зависит от a. По определению A->>C.

Доказано.

Таким образом, MVD AB и AC всегда составляют пару. Поэтому обычно их представляют вместе в форме A->> B | C.

FD является частным случаем MVD, когда множество значений зависимого атрибута обязательно состоит из одного элемента. Таким образом, если выполняется FD AB, то выполняется и MVD AB

Теорема Фейджина

r PROJECT (AB) NJ r PROJECT (AC) = r тогда и только тогда, когда A->>B|C. (то есть отношение декомпозируется без потерь)

Докажем достаточность условия теоремы.

Пусть <a,b,c'> и <a,b',c> принадлежат r. Следовательно, <a,b> принадлежит r1 <a,c> принадлежит r2. Следовательно, <a,b,c> принадлежит r1 NJ r2. Значит, <a,b,c> принадлежит r. По определению A->>B|C.

Доказательство необходимости условия теоремы.

1) Предположим, что <a,b,c> принадлежит r , следовательно, <a,b> принадлежит r1 <a,c> принадлежит r2. Следовательно, <a,b,c> принадлежит r1 NJ r2.
2) Из того, что <a,b,c> принадлежит r1 NJ r2, следует, что <a,b> принадлежит r1 <a,c> принадлежит r2. Значит, существуют такие <a,b,c'> и <a,b',c>, принадлежащие r, что <a,b,c> также принадлежит r (т.к. существует многозначная зависимость A->>B|C)

Конец доказательства.

Теорема Фейджина обеспечивает основу для декомпозиции отношений, удаляющей «аномальные» многозначные зависимости, с приведением отношений в четвертую нормальную форму.

(прим. Доказательства леммы и теоремы взяты из лекций Маши)

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

В переменной отношения R с атрибутами A, B, C (в общем случае, составными) имеется **многозначная зависимость** B от A (AB) в том и только в том случае, когда множество значений атрибута B, соответствующее паре значений атрибутов A и C, зависит от значения A и не зависит от значения C.

Чтобы перейти к вопросам дальнейшей нормализации, рассмотрим еще одну возможную (четвертую) интерпретацию переменной отношения СЛУЖ_ПРО_ЗАДАН. Предположим, что каждый служащий может участвовать в нескольких проектах, но в каждом проекте, в котором он участвует, им должны выполняться одни и те же задания. Возможное значение четвертого варианта переменной отношения СЛУЖ_ПРО_ЗАДАН показано на рис. 9.1.

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	A
2934	1	B
2934	2	A
2934	2	B
....	....	....
2941	1	A
2941	1	D

Рис. 9.1. Возможное значение переменной отношения СЛУЖ_ПРО_ЗАДАН (четвертый вариант)

Аномалии обновлений при наличии многозначных зависимостей и возможная декомпозиция

В новом варианте переменной отношения единственным возможным ключом является заголовок отношения {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}. Кортеж {сн, пн, сз} входит в тело отношения в том и только в том случае, когда служащий с номером сн выполняет в проекте пн задание сз. Поскольку для каждого служащего указываются все проекты, в которых он участвует, и все задания, которые он должен выполнять в этих проектах, для каждого допустимого значения переменной отношения СЛУЖ_ПРО_ЗАДАН должно выполняться следующее ограничение (B_СПЗ обозначает тело отношения):

$$\text{IF } (\{\text{сн}, \text{пн}_1, \text{сз}_1\} \in \text{B}_{\text{СПЗ}} \text{ AND } \{\text{сн}, \text{пн}_2, \text{сз}_2\} \in \text{B}_{\text{СПЗ}})$$

$$\text{THEN } (\{\text{сн}, \text{пн}_1, \text{сз}_2\} \in \text{B}_{\text{СПЗ}} \text{ AND } \{\text{сн}, \text{пн}_2, \text{сз}_1\} \in \text{B}_{\text{СПЗ}})$$

Наличие такого ограничения (как мы скоро увидим, это ограничение порождается наличием многозначной зависимости) приводит к тому, что при работе с отношением СЛУЖ_ПРО_ЗАДАН проявляются аномалии обновления.

1) **Добавление кортежа.** Если уже участвующий в проектах служащий присоединяется к новому проекту, то к телу значения переменной отношения СЛУЖ_ПРО_ЗАДАН требуется добавить столько кортежей, сколько заданий выполняет этот служащий.
2) **Удаление кортежей.** Если служащий прекращает участие в проектах, то отсутствует возможность сохранить данные о заданиях, которые он может выполнять.
3) **Модификация кортежей.** При изменении одного из заданий служащего необходимо изменить значение атрибута СЛУ_ЗАДАН в стольких кортежах, в скольких проектах участвует служащий.

Трудности, связанные с обновлением переменной отношения СЛУЖ_ПРО_ЗАДАН, решаются путем его декомпозиции на две переменных отношений: СЛУЖ_ПРО_НОМ {СЛУ_НОМ, ПРО_НОМ} и СЛУЖ_ЗАДАНИЕ {СЛУ_НОМ, СЛУ_ЗАДАН}. Значения этих переменных отношений, соответствующие значению переменной отношения СЛУЖ_ПРО_ЗАДАН с рис. 9.1, показаны на рис. 9.2.

Легко видеть, что декомпозиция, представленная на рис. 9.2, является декомпозицией без потерь и что эта декомпозиция решает перечисленные выше проблемы с обновлением переменной отношения

Значение переменной отношения **СЛУЖ_ПРО_НОМ**

СЛУ_НОМ	ПРО_НОМ
2934	1
2934	2
....	....
2941	1

Значение переменной отношения **СЛУЖ_ЗАДАНИЕ**

СЛУ_НОМ	СЛУ_ЗАДАН
2934	A
2934	B
....	....
2941	A
2941	D

СЛУЖ_ПРО_ЗАДАН.

Рис. 9.2. Значения переменных отношений СЛУЖ_ПРО_НОМ и СЛУЖ_ЗАДАНИЕ

Добавление кортежа. Если некоторый уже участвующий в проектах служащий присоединяется к новому проекту, то к телу значения переменной отношения СЛУЖ_ПРО_НОМ требуется добавить один кортеж, соответствующий новому проекту.

Удаление кортежей. Если служащий прекращает участие в проектах, то данные о заданиях, которые он может выполнять, остаются в отношении СЛУЖ_ЗАДАНИЕ.

Модификация кортежей. При изменении одного из заданий служащего необходимо изменить значение атрибута СЛУ_ЗАДАН в одном кортеже отношения СЛУЖ_ЗАДАНИЕ.

Переменная отношения r находится в **четвертой нормальной форме (4NF)** в том и только в том случае, когда она находится в BCNF, и все MVD r являются FD с детерминантами – возможными ключами отношения r (нет многозначных зависимостей).

29. N-декомпозируемые отношения. Пример декомпозиции. Зависимость проекции/соединения.

В переменной отношения R с атрибутами (возможно, составными) A и B MVD AB называется **тривиальной**, если либо AB, либо A UNION B совпадает с заголовком отношения R.

Нетривиальная многозначная зависимость: существует многозначная зависимость A->>B|C, но нет функциональных зависимостей A->B и A->C.

Тривиальная MVD всегда удовлетворяется. При AB она вырождается в тривиальную FD. В случае A UNION B = HR требования многозначной зависимости соблюдаются очевидным образом.

Отношение называется **n-декомпозируемым**, есть его можно декомпозировать на n частей.

Для **примера** n-декомпозируемого отношения при n > 2 рассмотрим пятый вариант переменной отношения СЛУЖ_ПРО_ЗАДАН, в которой имеется единственно возможный ключ {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН} и отсутствуют нетривиальные MVD. Пример значения переменной отношения приведен на рис. 9.3.

Возможное значение переменной отношения **СЛУЖ_ПРО_ЗАДАН** (пятый вариант)

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	A
2934	1	B
2934	2	A
2941	1	A

Значение переменной отношения **СЛУЖ_ПРО_НОМ** = (**СЛУЖ_ПРО_ЗАДАН** PROJECT {**СЛУ_НОМ, ПРО_НОМ**})

СЛУ_НОМ	ПРО_НОМ
2934	1
2934	2
2941	1

Значение переменной отношения **СЛУЖ_ЗАДАНИЕ** = (**СЛУЖ_ПРО_ЗАДАН** PROJECT {**ПРО_НОМ, СЛУ_ЗАДАН**})

ПРО_НОМ	СЛУ_ЗАДАН
1	A
1	B
2	A

Значение переменной отношения **СЛУЖ_ПРО_НОМ** = (**СЛУЖ_ПРО_ЗАДАН** PROJECT {**СЛУ_НОМ, СЛУ_ЗАДАН**})

СЛУ_НОМ	СЛУ_ЗАДАН
2934	A
2934	B
2941	A

Результат естественного соединения значений **СЛУЖ_ПРО_НОМ** NATURAL JOIN **ПРО_НОМ_ЗАДАН**

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН	
2934	1	A	
2934	1	B	
2934	2	A	
2941	1	A	
2941	1	B	← Лишний кортеж

Рис. 9.3. Возможное значение переменной отношения СЛУЖ_ПРО_ЗАДАН (пятый вариант), результаты проекций и результат частичного естественного соединения

Как показано на рис. 9.3, результат естественного соединения проекций СЛУЖ_ПРО_НОМ и ПРО_НОМ_ЗАДАН почти совпадает с телом исходного отношения СЛУЖ_ПРО_ЗАДАН, но в нем присутствует один лишний кортеж, который исчезнет после выполнения заключительного естественного

соединения с проекцией СЛУЖ_ЗАДАНИЕ. Читателям предлагается убедиться, что исходное отношение будет восстановлено при любом порядке естественного соединения трех проекций.

Зависимость проекции/соединения

Если служащий с номером сн участвует в проекте пн, и в проекте пн выполняется задание сз, и служащий с номером сн выполняет задание сз, то служащий с номером сн выполняет задание сз в проекте пн.

В общем виде такое ограничение называется зависимостью **проекции/соединения**.

Формальное определение:

Пусть задана переменная отношения R, и A, B, …, Z являются произвольными подмножествами заголовка R (составными, перекрывающимися атрибутами). В переменной отношения R **удовлетворяется зависимость проекции/соединения** (Project-Join Dependency – PJD) *(A, B, …, Z) тогда и только тогда, когда любое допустимое значение r переменной отношения R можно получить путем естественного соединения проекций этого значения на атрибуты A, B, …, Z.

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

В переменной отношения СЛУЖ_ПРО_ЗАДАН выполняется PJD *({СЛУ_НОМ, ПРО_НОМ}, {ПРО_НОМ, СЛУ_ЗАДАН}, {СЛУ_НОМ, СЛУ_ЗАДАН}). Наличие такой PJD обеспечивает возможность декомпозиции отношения на три проекции, но возникает вопрос, зачем это нужно? Чем плохо исходное отношение СЛУЖ_ПРО_ЗАДАН? Ответ обычный: этому отношению свойственны аномалии обновления. Для примера предположим, что значением СЛУЖ_ПРО_ЗАДАН является отношение, показанное на рис. 9.4.

1) **Добавление кортежей.** Если к ТСП31 (рис. 9.4) добавляется кортеж <2941, 1, A>, то должен быть добавлен и кортеж <2934, 1, A>. Действительно, в теле отношения появятся кортежи <2934, 1, B>, <2941, 1, A> и <2934, 2, A>. Ограничение целостности требует включения и кортежа <2934, 1, A>. Интересно, что добавление кортежа <2934, 1, A> не нарушает ограничение целостности и, тем самым, не требует добавления кортежа <2941, 1, A>.

Возможное значение переменной отношения **СЛУЖ_ПРО_ЗАДАН** (ТСП31)

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	B
2934	2	A

Результат добавления к ТСП31 кортежа <2941, 1, A> (ТСП32)

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	B
2934	2	A
2941	1	A
2934	1	A

Рис. 9.4. Иллюстрации аномалий обновления в отношении СЛУЖ_ПРО_ЗАДАН при наличии зависимости соединения

2) **Удаление кортежа.** Если из ТСП32 удаляется кортеж <2934, 1, A>, то должен быть удален и кортеж <2941, 1, A>, поскольку в соответствии с ограничением целостности наличие второго кортежа означает наличие первого. Интересно, что удаление кортежа <2941, 1, A> не нарушает ограничения целостности и не требует дополнительных удалений.

Устранение аномалий обновления в 3-декомпозиции

Результат декомпозиции переменной отношения СЛУЖ_ПРО_ЗАДАН с телом значения ТСП31

СЛУЖ_ПРО_НОМ

СЛУ_НОМ	ПРО_НОМ
2934	1
2934	2

СЛУЖ_ЗАДАНИЕ

СЛУ_НОМ	СЛУ_ЗАДАН
2934	B
2934	A

ПРО_НОМ_ЗАДАН

ПРО_НОМ	СЛУ_ЗАДАН
1	B
2	A

Добавление данных о служащем с номером 2941, выполняющем задание A в проекте 1

СЛУЖ_ПРО_НОМ

СЛУ_НОМ	ПРО_НОМ
2934	1
2934	2
2941	1

СЛУЖ_ЗАДАНИЕ

СЛУ_НОМ	СЛУ_ЗАДАН
2934	B
2934	A
2941	A

ПРО_НОМ_ЗАДАН

ПРО_НОМ	СЛУ_ЗАДАН
1	B
2	A
1	A

Результат естественного соединения отношений после добавления данных

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	B
2934	2	A
2941	1	A
2934	1	A

Рис. 9.5. Иллюстрация декомпозиции отношения с зависимостью соединения

После выполнения декомпозиции трудности с обновлением автоматически снимаются. Действительно, декомпозируем отношение СЛУЖ_ПРО_ЗАДАН на три отношения: СЛУЖ_ПРО_НОМ {СЛУ_НОМ, ПРО_НОМ}, СЛУЖ_ЗАДАНИЕ {СЛУ_НОМ, СЛУ_ЗАДАН} и ПРО_НОМ_ЗАДАН {ПРО_НОМ, СЛУ_ЗАДАН}. Результат декомпозиции значения переменной отношения СЛУЖ_ПРО_ЗАДАН с телом ВСП31 показан в верхней части рис. 9.5.

Теперь если мы хотим добавить данные о служащем с номером 2941, выполняющем задание A в проекте 1, то, естественно, вставим кортеж <2941, 1> в отношение СОТР-ПРО_НОМ, кортеж <2941, A> в отношение СОТР-ЗАДАНИЕ и кортеж <1, A> в отношение ПРО_НОМ-ЗАДАН. Результат этих операций показан в средней части рис. 9.5.

Но если выполнить естественное соединение декомпозированных отношений с телами, полученными после добавления данных о служащем с номером 2941, выполняющем задание A в проекте 1, то будет получено значение-отношение с заголовком отношения СЛУЖ_ПРО_ЗАДАН и телом ВСП32 (нижняя часть рис. 9.5). Тем самым, проведенная декомпозиция позволила избежать сложностей при выполнении добавления кортежей с получением корректных результатов.

Аналогично можно проиллюстрировать простоту и корректность операций удаления кортежей.

В переменной отношения R PJD *(A, B, …, Z) называется **подразумеваемой возможными ключами** в том и только в том случае, когда каждый составной атрибут A, B, …, Z является суперключом R, т. е. включает хотя бы один возможный ключ R.

В переменной отношения R зависимость проекции/соединения *(A, B, …, Z) называется **тривиальной,** если хотя бы один из составных атрибутов A, B, …, Z совпадает с заголовком R.

Переменная отношения R находится в **пятой нормальной форме**, или в нормальной форме проекции/соединения (5NF, или PJ/NF – Project-Join Normal Form) в том и только в том случае, когда каждая нетривиальная PJD в R подразумевается возможными ключами R.

5NF является «окончательной» нормальной формой, которой можно достичь в процессе нормализации на основе проекций. «Окончательность» понимается в том смысле, что у отношения, находящегося в 5NF, отсутствуют аномалии обновлений, которые можно было бы устранить путем его декомпозиции. Другими словами, такие отношения далее нормализовать бессмысленно.

31. Подходы к физическому хранению отношений. Построчное хранение отношений. Понятие tid-a.

Возникают следующие разновидности объектов во внешней памяти базы данных:

1. Отношения – основная часть базы данных
2. Служебные отношения (хранят информацию о схеме, параметрах ключа и т.д.)
3. индексы - управляющие структуры, ускоряющие доступ
4. журналы (надежность хранения данных)
5. служебная информация, поддерживаемая для внутренних потребностей нижнего уровня системы (например, информация о свободной памяти).

Хранение Отношений:

1. Покортежное или построчное (быстрый доступ, занимает больше места).

Внешняя память – множество сегментов, разбитых на страницы.

tid – идентификатор кортежа (обычно, <№страницы, № кортежа>).

В случае перемещения кортежа меняется описание, но tid сохраняется.

2. Постолбцовое (сохранение места, но трудоемкая сборка). Редко применимо и дальше не рассматривается.

32. Понятие индексов в базе данных. Техника хранения на основе B-деревьев. Методы хеширования.

Основное назначение индексов состоит в обеспечении эффективного прямого доступа к кортежу 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, т.е. не содержать дубликатов ключа.

Полезным свойством индекса является обеспечение последовательного просмотра кортежей таблицы в заданном диапазоне значений ключа в порядке возрастания или убывания значений ключа.

Механизм мульти-индекса… Никому не нужен! Если нужен, то вот… используется для оптимизации эквисоединения таблиц. Суть в том что мульти-индексы организуются для таблиц с одинаковыми атрибутами (один или несколько из атрибутов становится ключом мульти-индекса). Значению ключа сопоставляется набор кортежей этих таблиц, значения выделенных атрибутов которых совпадают со значением ключа.

Общей идеей любой организации индекса (для прямого доступа по ключу и просмотра в порядке возрастания/убывания значений ключа) является хранение упорядоченного списка значений ключа (со списком идентификаторов кортежей для каждого значения ключа). Одна организация индекса отличается от другой, главным образом, в способе поиска ключа с заданным значением.

B-деревья – способ организации индексов, дерево во внешней памяти (накладные расходы велики, менять информацию легче)

Механизм B-деревьев (B+) ориентирован на хранение во внешней памяти 2 типов страниц:

a. Внутренних – это последовательность значений ключей и ссылка на № следующей страницы, либо на листовую.

N_1 ключ$_1$ N_2 ключ$_2$ N_3 ключ$_3$ . . . N_m ключ$_m$ N_{m+1}

При этом выдерживаются следующие свойства:

i. ключ$_1$ < ключ$_2$ <...< ключ$_m$

ii. в странице дерева N_m находятся ключи k со значениями ключ$_m$ <= k <= ключ$_{m+1}$.

b. Листовых

ключ$_1$ список$_1$ ключ$_2$ список$_2$ . . . ключ$_k$ список$_k$

i. ключ$_1$ < ключ$_2$ < ... < ключ$_k$;

ii. список$_r$ – упорядоченный список идентификаторов кортежей (tid), включающих значение ключr;

iii. листовые страницы связаны одно- или двунаправленным списком.

B-дерево – сбалансировано (ветви до листьев одинаковой глубины).

Глубина ~$\log_m N$, где m – число ключей во внутренней странице, N – число кортежей.

При создании дерева, в случае переполнения порождается новая страница, в которую переливается половина предыдущей. => частые расщепления и слияния.

Решение:

1. Упреждения расщепления (раннее расщепление по достижении некого уровня)
2. Переливание (равновесие соседних вершин)
3. Слияние 3-в-2 (а не 2-в-1)

Хеширование – способ организации индексов.

1. Классический случай – свертка ключа используется как адрес в таблице, содержащей ключи и записи. Основным требованием к хэш-функции является равномерное распределение значение свертки
2. Расширяемое хэширование (Extendible Hashing) – принцип использования деревьев цифрового поиска в основной памяти. В основной памяти поддерживается справочник, организованный на основе бинарного дерева цифрового поиска, ключами которого являются значения хэш-функции, а в листовых вершинах хранятся номера блоков записей во внешней памяти.
3. Линейное хэширование (Linear Hashing) – является то, что для адресации блока внешней памяти всегда используются младшие биты значения хэш-функции. Если возникает потребность в расщеплении, то записи перераспределяются по блокам так, чтобы адресация осталась правильной.

33. Виды проектирования баз данных. Недостатки проектирования в терминах отношений. Понятие информационной модели. Достоинства информационного моделирования. Средства автоматизации проектирования баз данных.

Проектирование БД.

Процесс создания модели, использование информации, не зависящей от любых физических аспектов ее представления.

Концептуальное

↓

Процесс создания модели, используемой по предполагаемой информации с учетов выбранной модели организации данных, но независящей от типа целевой СУБД и др. физических аспектов ее реализации.

Логическое

↓

Процесс описания реализации БД с учетом выбранной целевой СУБД и указанием структур хранения и методов доступа, используемых для организации эффективной обработки данных.

Физическое

Ограниченность проектирования:

1. Реляционная модель данных не предлагает какого-либо механизма для разделения сущностей объектов предметной области и связей между ними,
2. Модель не обеспечивает достаточных средств для представления смысла данных,
3. Во многих прикладных областях трудно моделировать предметную область на основе плоских таблиц,
4. Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет какие-либо формализованные средства для представления этих зависимостей,
5. На ранних стадиях требуется участие специалистов данной ПО.

Информационная модель – способ представления понятий или объектов ПО, описание сущностей для данного предприятия совокупности объектов, их параметры, поведение и отношение между ними.

Достоинства информационного моделирования:

1. Построение мощной и наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на стадии проектирования схемы реляционной БД
2. Информационная модель в виде документации (хотя бы в виде вручную нарисованных диаграмм и комментариев к ним), которая может оказаться очень полезной не только при проектировании схемы реляционной БД, но и при эксплуатации, сопровождении и развитии уже заполненной БД.
3. На рынке представлены CASE-системы, обеспечивает автоматизированное преобразование диаграммных концептуальных схем баз данных, представленных в той или иной семантической модели данных, в реляционные схемы, специфицированные чаще всего на языке SQL

CASE – computer aided software engineering.

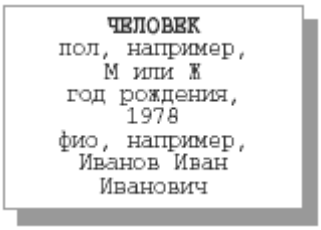
Система поддержки технической автоматизации проектрирования, реализации и сопроводления сложных тпрограммных систем на всех этапах их жизненного цикла

34. ER-модель. Основные понятия. Представление на диаграммах сущностей, атрибутов и связей. Примеры. Уникальные идентификаторы типов сущностей.

Entity – Relation модель (сущность - связь) предназначается для описания ПО в целях проектирования БД(моделирование базовых понятий на простых графиках и диаграммах).

Основные понятия:

Сущность - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступной (изображается прямоугольником)



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

(Пример справа)

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями.

ER-модели связи бинарные. В любой связи выделяются два конца, на каждом из которых указываются *имя* конца связи, *степень* конца связи (сколько экземпляров данного типа сущности должно присутствовать в каждом экземпляре связи), *обязательность* связи.

Связь представляется в виде ненаправленной линии, соединяющей две сущности или ведущей от сущности к ней же самой. При этом в месте «стыковки» связи с сущностью используются *трехточечный* вход (в связи могут/должны использоваться много экземпляров сущности) либо *одноточечный* вход(может/должен участвовать только один экземпляр сущности).

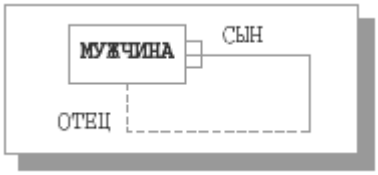
Обязательный конец связи изображается сплошной линией, а необязательный – прерывистой линией.

Примеры.

«Многие к одному»

- каждый БИЛЕТ предназначен для одного и только одного ПАССАЖИРА;
- каждый ПАССАЖИР может иметь один или более БИЛЕТОВ.

«Рекурсивная модель»



- каждый МУЖЧИНА является сыном одного и только одного МУЖЧИНЫ;
- каждый МУЖЧИНА может являться отцом одного или более МУЖЧИН.

Как отмечалось выше, при определении типа сущности необходимо гарантировать, что каждый экземпляр сущности является отличимым от любого другого экземпляра той же сущности. Необходимо сообщить системе автоматизации проектирования БД, каким образом будут различаться эти экземпляры, т. е. сообщить, как конструируются уникальные идентификаторы экземпляров каждого типа сущности.

Уникальным идентификатором сущности может быть атрибут, комбинация атрибутов, связь, комбинация связей или комбинация связей и атрибутов, уникально отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

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

1. Каждый **простой тип сущности** превращается в таблицу.
   Имя сущности – имя таблицы. Экземпляр – строка.
   Простым типом сущности называется тип сущности, не являющийся подтипом и не имеющий подтипов (см. опр. в Б36).
2. Каждый **атрибут** => столбец таблицы
3. Компоненты **уникального идентификатора сущности** превращаются в первичный ключ таблицы. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи
4. **Связи «многие к одному»** (и «один к одному») становятся внешними ключами, т.е. образуется копия уникального идентификатора сущности на конце связи «один», и соответствующие столбцы составляют внешний ключ таблицы, соответствующей типу сущности на конце связи «многие». Необязательные связи соответствуют столбцам внешнего ключа, допускающим наличие неопределенных значений; обязательные связи – не допускают.
   Если между двумя типами сущности A и B имеется связь «один к одному», то соответствующий внешний ключ по желанию проектировщика может быть объявлен как в таблице A, так и в таблице B. Чтобы подчеркнуть единственность в определении таблицы, соответствующий столбец должен быть специфицирован как возможный ключ таблицы.
5. **связь «многие ко многим»** между типами сущности A и B создается дополнительная таблица AB с двумя столбцами, один из которых содержит уникальные идентификаторы экземпляров сущности A, а другой – уникальные идентификаторы экземпляров сущности B.

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

Тип сущности может быть расщеплён на несколько **подтипов**, каждый из которых включает общие атрибуты и/или связи. Тип сущности, на основе которого определяются подтипы, называется **супертипом**.

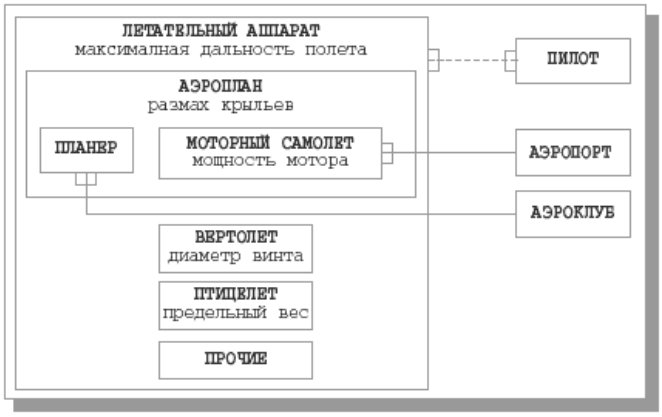
ER-модели не ограничивает подтипизацию, обычно оказывается достаточно 2-3 уровней.

Простой тип – не является подтипом и не имеет подтипов.

Особенности механизма наследования:

1. Включение: для любого b Є Bi => b Є A
2. Отсутствие собственных экземпляров у супертипа: для любого a Є A => a Є Bi.
3. Разъединенность подтипов: для любого b Є Bi => b !Є Bj.

Пример:



Супертипы и подтипы.

a. Общая таблица для всех подтипов

A

T	a_1	a_2	…	…	…	a_M	b_{11}	…	…	…	b_{1k1}	…	…	…	b_{nkn}
'B_1'	X	X	X	X	X	X	X	X	X	X	X	NULL	NULL	NULL	NULL

T-признак подтипа, X-некоторое значние.

Для супертипа верно что (см след билет):

1. для любого b Є Bi => b Є A
2. для любого a Є A => a Є Bi
3. для любого b Є Bi => b !Є Bj

Извлечь объекты супертипа: PROJECT A{a_1…a_m, b_{i1}…b_{ik}}WHERE t ='Bi'

b. Отдельная таблица для каждого подтипа

Собрать супертип: PROJECT B_1{a_1…a_m}UNION…UNION B_n{a_1…a_m}

Достоинства а.

- соответствие логике супертипов и подтипов; поскольку любой экземпляр любого подтипа является экземпляром супертипа, логично хранить вместе все строки, соответствующие экземплярам супертипа;
- обеспечение простого доступа к экземплярам супертипа и не слишком сложный доступ к экземплярам подтипов;
- возможность обойтись небольшим числом таблиц.

Недостатки а.

- усложнение программного кода приложений
- общая таблица потенциально может стать узким местом при многопользовательском режиме
- расход внешней памяти (много NULL)

Достоинства б.

- действуют более понятные правила работы с подтипами (каждому подтипу соответствует одноименная таблица);
- упрощается логика приложений; каждая программа работает только с нужной таблицей.

Недостатки б.

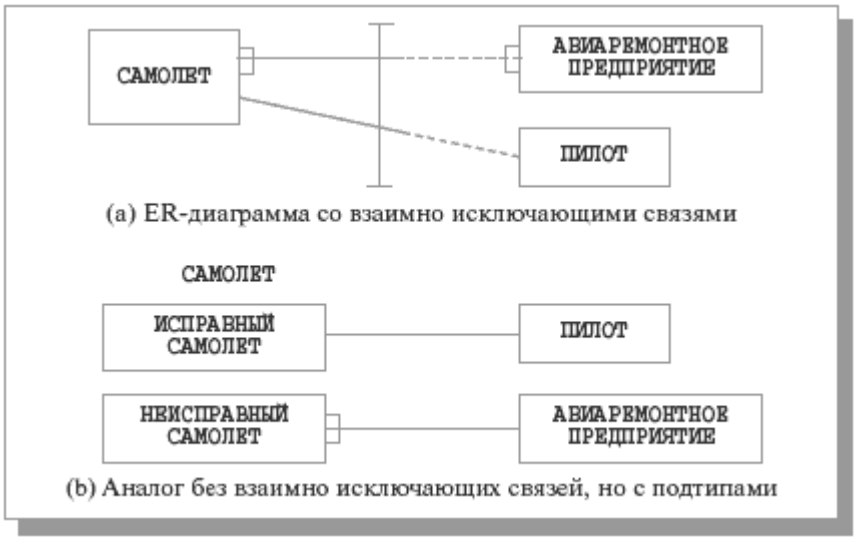
- в общем случае требуется слишком много отдельных таблиц;
- усложнение доступа к экземплярам супертипа;

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

Для заданной сущности можно определить такой набор связей с другой сущностью, что для каждого экземпляра сущности может/должен существовать только одной связи из данного набора.

Диаграмма со взаимно исключающими связями может быть преобразована к диаграмме с наследованием 2 способами:

1. Специализация (внедрение подтипов). Пример, см рис Б.
2. Введение общего супертипа (обобщение).
   Так чтобы у заданной сущности была только одна связь с этим супертипом (который содержал бы в себе остальные экземпляры сущностей).



6. Представление взаимоисключающих связей.

Преобразовать взаимоисключающие связи по пункту 6.

Если связь 1-мн (причем конец много подсоединен к сущности), то следующие выходы:

a. **общее хранение внешних ключей;**

Если внешние ключи всех потенциально связанных таблиц имеют общий формат, то можно применить способ (а), т. е. создать два столбца: идентификатор связи и идентификатор сущности (возможно, составной).

b. **раздельное хранение внешних ключей**.

В таблице как минимум n столбцов (любой может быть составным – внешний ключ связи)

Достоинства а.

- минимальное число столбцов

Недостатки а.

- усложнение выполнения операции соединения

Достоинства б.

- Упрощение операции соединение (A *NATURAL JOIN* B)

Недостатки б.

- Требуется большое число столбцов (в любом может быть NULL) => расход памяти.

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

UML позволяет моделировать разные виды систем: чисто программные, чисто аппаратные, программно-аппаратные, смешанные, явно включающие деятельность людей и т. д.

Стандарт: 12 диаграмм классов, позволяющих описать статические (структурные) и динамические (поведенческие) свойства систем.

Основные понятия:

Диаграмма классов может включать **комментарии** (соединяются пунктиром с той сущностью, которую поясняют) и **ограничения** (заключают в { }). Ограничения могут неформально задаваться на естественном языке или же могут формулироваться на языке объектных ограничений OCL (Object Constraints Language).

Классом называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой (изображается прямоугольником).

Атрибутом класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Атрибут является абстракцией состояния объекта.

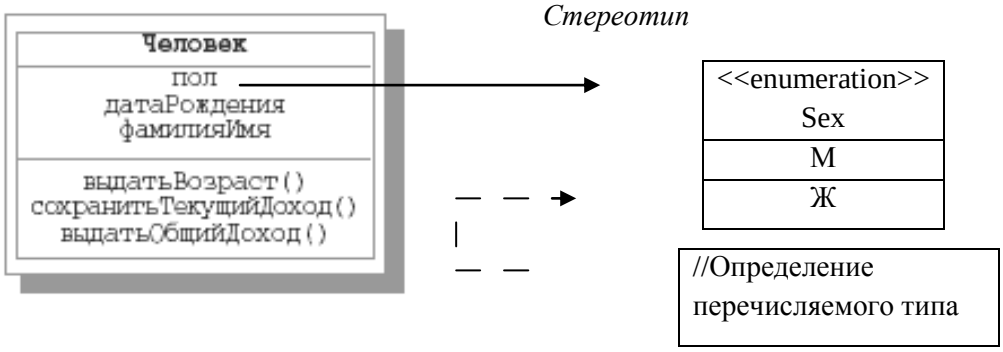
Операцией класса называется именованная услуга, которую можно запросить у любого объекта этого класса. Операция – это абстракция поведения объекта.

Стереотип – механизм расширения семантики UML, позволяющий создать новые элементы UML на основе существующих, с учетом особенности решения задачи.

В стандарте предусмотрен набор стереотипов.

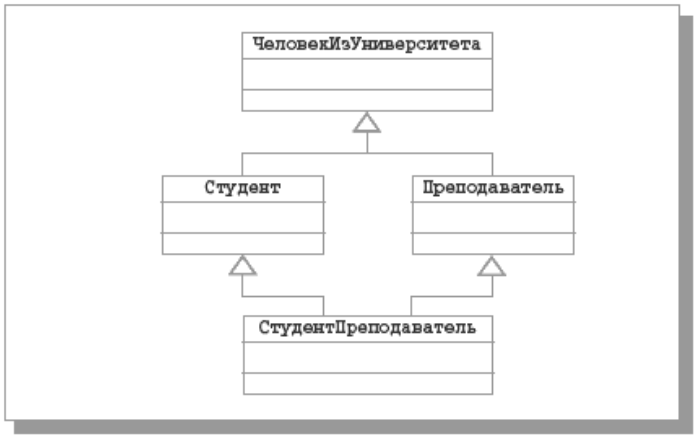
Класс

Стереотип



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

Связью-обобщением называется связь между общей сущностью (суперкласс/родителем) и более специализированным типом этой сущности (подкласс/потомок).
Полиморфизмом по включению – объекты потомка могут использоваться везде, где могут использоваться объекты предка. Потомок наследует все атрибуты и операции предка.
У класса есть скрытое свойство *isAbstract* – может ли класс иметь объекты.
Множественное наследование – подкласс определяется на основе нескольких суперклассов.



Ассоциацией называется структурная связь между объектами одного класса и объектами другого или того же самого класса (изображается линией).
В ассоциации могут связываться два класса, и тогда она называется бинарной. Допускается создание ассоциаций, связывающих n классов – n-арные ассоциации.

С ассоциацией связаны 5 важных понятий: *имя, роль, кратность, агрегация и навигация*.

У связи может быть **имя**, треугольник указывает направление чтения имени связи; у каждого класса может быть **роль** в ассоциации;
кратность (задается числом, интервалом, списком значений и интервалов, * - бесконечность) показывает, сколько объектов класса может/должно участвовать в экземпляре ассоциации.
Агрегация в UML используется для отображения связи «часть-целое» (класс-целое имеет более высокий концептуальный уровень).
Иногда связь «части» и «целого» настолько сильна, что «часть» может быть частью только 1 «целого», и уничтожение «целого» приводит к уничтожению всех его «частей». Такие агрегации– **композитные.**

Навигация может осуществляться в обоих направлениях. Для некоторых ассоциаций желательно ограничить навигацию (стрелка).

С точки зрения библиотеки разумно произвести навигацию от объекта-библиотеки к связанным с ним объектам-книгам. Однако вряд ли потребуется по данному экземпляру книги узнать, в какой библиотеке она находится

40. Язык OCL. Инварианты OCL. Основные типы данных и выражения.

В UML 2 способа определения ограничений: на естественном языке и на языке OCL. Более точный и лаконичный способ формулировки ограничений обеспечивает язык OCL (Object Constraints Language).
Из языка UML в OCL заимствованы, в первую очередь, следующие концепции:

- класс, атрибут, операция;
- объект (экземпляр класса);
- ассоциация;
- тип данных (включая набор типов Boolean, Integer, Real и String);
- значение (экземпляр типа данных).

В OCL предопределены структурные типы, которые являются разновидностями типов коллекций (collection):

- математическое множество **(set),** неупорядоченная коллекция, нет одинаковых элементов;
- мультимножество **(bag),** неупорядоченная коллекция, которая может содержать элементы-дубликаты;
- последовательность **(sequence),** упорядоченная коллекция, которая может содержать элементы-дубликаты.
- упорядоченное множнество **(ordered set),** упорядоченная коллекция, нет одинаковых элементов;

OCL может применяться для определения ограничений, описывающих пред- и постусловия операций классов, и ограничений, представляющих собой инварианты классов.

Инвариант класса – это логическое выражение, вычисление которого должно давать true при создании любого объекта данного класса и сохранять истинное значение в течение всего времени существования этого объекта.

Синтаксис определения инварианта:

```
context <class_name> inv:
<OCL-выражение>
```

Пример (существует человек, обладающий контрольным пакетом акций)

```
Context Компания  inv:
Self.акционер =>Exists
(a:человек|a.количествоакций >= self.количествоакций div 2 +1)
```

Операции над объектами

<объект>.<имя_атрибута> (значение атрибута)

<объект>.<имя_роли> (переход по соединению, под именем имеется ввиду имя роли объекта на другом конце связи)

<объект>.<имя_операции>(фактический параметр, т.е. можно вызвать операцию)

Операции над коллекциями

<коллекция>⟶<имя операции> (<список фактических параметров>)

Select(iterat/логическое выражение)	Результат – новая коллекция (из старых элементов, удовлетворяющих логическому выражению)
collect(iterat/выражение)	Результат – новая коллекция (элементы вычисляются по выражению)
exists(iterat/логическое выражение)	True, если выполнено хоты бы для 1 элемента
forAll(iterat/логическое выражение)	True, если выполнено для всех элементов
Size()	Число элементов
union, intersect, symmetricDifference	
At(index)	
Count(element)	
Includes/excludes(element)	

Тип	**Операции**
Boolean	And or xor implies if_then_else
Integer	+-*/ div mod min max сравнение
Real	+-*/ abs float round min max сравнение
String	Concat size substring toUpper toLower

41. Получение реляционной схемы из диаграммы классов. Основные проблемы и рекомендации.

Рекомендации:

- не увлекайтесь определением операций в классах
- старайтесь избегать множественное наследие и осторожно использовать одиночное
- реализ. в СУБД с точно заданными кратностями возможно, но предпочтит. доп. триггеров и ограничений (уменьшает эффективность)
- агрег. ассоциации не естественны, компоретные (ПОПРАВЬТЕ, НЕ РАЗОБРАЛ) влияют только на способ поддержки ссылочной целостности
- однонаправленные связи естественны только для объектно-ориентированных СУБД. Для реляционных БД поддержка вызывает дополнительные накладные расходы, значит, снижается эффективность.
- не злоупотребляйте ограничениями (снижается эффективность)

42. Язык баз данных SQL. Основные отличия SQL-ориентированной модели от реляционной модели. Стандарт SQL:2003 – основные тома. Структура языка SQL (три различных схемы).

В стандартах SQL определяется собственная модель данных, она похожа на реляционную, но значительно отличается. Итак, 2 важных отличия:

- В SQL данные – это *набор таблиц*, каждая таблица содержит *мультимножество* строк, соответствующих заголовку таблицы. В реляционной модели "фундаментальная абстрактная «родовая» структура данных отношение, представляет собой множество кортежей." (*прим.Валеры. Я сам не понял смысла, извините*)
- В SQL для таблицы поддерживается порядок столбцов, соответствующий порядку их определения. В реляционной модели атрибуты отношения не упорядочены.

Другими словами, таблица – это вовсе не отношение, хотя во многом они похожи.

Отмечается, что из этого следует, что:

- в SQL отсутствует обязательное предписание об ограничении целостности сущности.
- В базе данных могут существовать таблицы, для которых не определен первичный ключ.

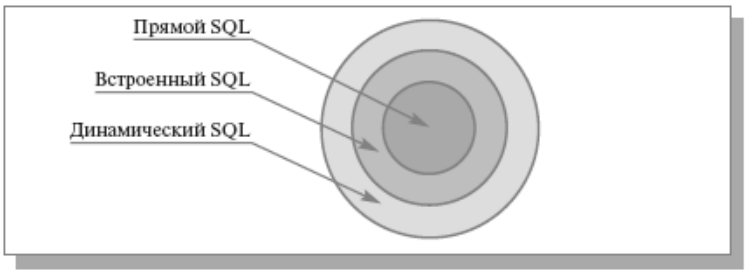
С другой стороны, если для таблицы определен первичный ключ, то для нее ограничение целостности сущности поддерживается так же как и в реляционной модели данных.
Ссылочная целостность в модели данных SQL поддерживается в обязательном порядке, но в трех разных вариантах, лишь один из которых полностью соответствует реляционной модели. Это связано с интенсивным использованием в SQL неопределенных значений.

Стандарт SQL:2003 – основные тома:

- 9075-1, SQL/Framework;
- 9075-2, SQL/Foundation;
- 9075-3, SQL/CLI;
- 9075-4, SQL/PSM;
- 9075-9, SQL/MED;
- 9075-10, SQL/OLB;
- 9075-11, SQL/Schemata;
- 9075-13, SQL/JRT;
- 9075-14, SQL/XML.

Структура языка SQL

1. Классификация, ориентированная на *производителей* СУБД (изнутри). Здесь каждая компания может выбрать уровни, которые они реализуют для своей БД. **Язык разбивается на** базовый (entry)**,** промежуточный (intermediate) **и** полный (full) **уровни.**
2. Классификация, ориентированная на программиста приложений БД(снаружи):
Прямой – например,интерактивная консоль(вы пишете SELECT * FROM… , и вам сразу выдают результат)
встраиваемый – например**,** реализация библиотек для языков программирования, что бы можно было в этих программах использовать SQL,
динамический – здесь сам SQL запрос может динамически формироваться по ходу программы(в предыдущем сама "строка SQL запросы" была как бы "постоянна").



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

Все данные, хранящиеся в таблицах, типизированы. Каждому столбцу определяемой таблицы приписывается свой собственный тип. СУБД должна следить за соответствием типов (допустимыми значениями).

Типы данных

Логические типы (TRUE, FALSE, UNKNOWN). В стандарте UNKNOWN и NULL не различаются.
Точные числовые типы (SMALLINT, INTEGER). С дробной частью – NUMERIC (p,s), DECIMAL(p,s). P – число десятичных цифр, s – шкала (число десятичных цифр в дробной части). Для p,s возможны значения по умолчанию. Для NUMERIC p – строгое, для), DECIMAL – максимально возможное. Пример: 1234,5 – в NUMERIC недопустимо, в), DECIMAL допустимо.
Приближенные числовые типы (вещественные типы REAL, DOUBLE PRECISION, FLOAT(p)). Параметр p – число значимых цифр в мантиссе.
Символьные строки. CHAR(x) – строки фиксированной длины x (по умолчанию x = 1), VARCHAR – произвольной длины, но не более x (по умолчанию x = 1). CLOB (character large object) – этот тип данных предназначен для определения столбцов, хранящих большие и разные по размеру группы символов. При определении столбца задается спецификация CLOB (z), где z задает максимальный размер соответствующей группы символов. Максимально возможное значение параметра z определяется в реализации, но, очевидно, что оно должно быть существенно больше максимально возможного значения параметра x, присутствующего в типах CHAR и CHAR VARYING (z >> x; если длина меньше x, то справа добавляются пробелы).
Битовые строки (BIT, BIT VARYING(x), BLOB(x)).
Дата и время. DATE : “YYYY-MM-DD”, TIME “HH:MM-SS:FF” (число символов в FF определяется параметром p), TIMESTAMP “YYYY-MM-DD HH:MM-SS:FF”(по умолчанию p = 6, для секунд допустим диапазон 0..61), TIME WITH TIME ZONE – +HH:MM, -HH:MM по Гринвичу. Стандартные функции: CURRENT_DATE(_TIME, _STAMP).
Временные интервалы (INTERVAL ‘start(p)[TO end (q)]’). P—точность (по умолчанию p = 2), разница не привязана к началу/концу интервала.
(прим. остальных типов в лекциях Маши нет)
Преобразование типов. В SQL поддерживаются явные и неявные преобразования значений одного типа к значениям другого. Неявные (интуитивные) преобразования типов не всегда удобны, недостаточно гибки и иногда могут вызывать ошибки. Поэтому, как показывает предыдущий подраздел, число допустимых неявных преобразований типов в SQL весьма ограничено. Однако в SQL существует специальный оператор CAST, с помощью которого можно *явно* преобразовывать типы или домены в более широких пределах допускаемых преобразований. Конструкция имеет следующий синтаксис:

```
CAST ({scalar-expression | NULL } AS
    {data_type | domain_name})
```

44. Средства работы с доменами в SQL.

Домен является долговременно хранимым, именованным объектом схемы базы данных. Домены можно создавать (определять), изменять (изменять определения) и ликвидировать (отменять определение). Имена доменов можно использовать при определении столбцов таблиц**. Можно считать, что в SQL *определение домена* представляет собой вынесенное за пределы определения индивидуальной таблицы «родовое» определение столбца, которое можно использовать для определения различных реальных столбцов реальных базовых таблиц**.

Определение домена.

Для определения домена в SQL используется оператор CREATE DOMAIN. Общий синтаксис этого оператора следующий:[88)]

domain_definition ::= CREATE DOMAIN domain_name [AS] data_type
[default_definition]
[domain_constraint_definition_list]

Здесь domain_name задает имя создаваемого домена[89)], data_type есть спецификация определяющего типа данных. В необязательных разделах default_definition и domain_constraint_definition_list специфицируются значение домена по умолчанию[90)] и набор ограничений целостности, которые будут применяться к любому столбцу, определенному на этом домене.

Раздел default_definition имеет вид

DEFAULT { literal | niladic_function | NULL }[91)]

Здесь literal представляет любое допустимое литеральное значение определяющего типа домена, NULL обозначает неопределенное значение, а niladic_function может задаваться в одной из следующих форм:

USER
CURRENT_USER
SESSION_USER
SYSTEM_USER
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP[92)]

Если в операторе CREATE DOMAIN значение по умолчанию не специфицируется, считается, что такого значения нет. Однако позже к определению домена можно добавить раздел значения по умолчанию с помощью оператора ALTER DOMAIN. Кроме того, этот оператор позволяет удалить раздел значения по умолчанию из существующего определения домена.

Элемент списка domain_constraint_definition_list имеет вид

[CONSTRAINT constraint_name]
CHECK (conditional_expression)

Необязательный раздел CONSTRAINT constraint_name позволяет определить имя нового ограничения целостности. Если явное указание имени отсутствует, ограничению назначается имя, автоматически генерируемое системой. Что касается вида условного выражения, служащего собственно ограничением целостности, то в стандарте запрещается лишь прямое или косвенное использование в нем домена, в определение которого входит данное условное выражение.[93)] Однако наиболее естественным (и наиболее распространенным) видом ограничения домена является следующий:

CHECK (VALUE IN (list_of_valid_values))

Такое ограничение запрещает появление в любом столбце, определенном на данном домене, любого значения определяющего типа, не входящего в список допустимых значений.

Изменение домена.

Для изменения характеристик ранее определенного домена используется оператор SQL ALTER DOMAIN. Синтаксис этого оператора выглядит следующим образом:

domain_alternation ::=
ALTER DOMAIN domain_name domain_alternation_action
domain_alternation_action ::=
domain_default_alternation_action
| domain_constraint_alternation_action

Как видно из синтаксических правил, при изменении определения домена можно выполнить действие по изменению раздела значения по умолчанию либо изменить ограничение домена. Для первого варианта действует следующий синтаксис:

domain_default_alternation_action ::=
SET default_definition
| DROP DEFAULT

В случае установки нового значения по умолчанию (SET) это значение автоматически применяется ко всем столбцам, определенным на данном домене. Более точно, это значение становится новым значением по умолчанию. Операция не оказывает влияния на состояние существующих строк таблиц базы данных. В случае отмены раздела значения по умолчанию в определении домена (DROP) существовавшее значение домена по умолчанию становится значением по умолчанию каждого столбца, который определен на данном домене и для которого не специфицировано собственное значение по умолчанию.

Действие по изменению ограничения домена определяется следующим синтаксисом:

domain_constraint_alternation_action ::=

ADD domain_constraint_definition

| DROP CONSTRAINT constraint_name

Действие по добавлению нового определения ограничения домена (ADD) приводит к тому, что новое условие добавляется через AND к существующему ограничению домена. Если к моменту выполнения соответствующего оператора ALTER DOMAIN существуют столбцы некоторых таблиц, текущие значения которых противоречат новому ограничению, то СУБД должна отвергнуть этот оператор ALTER DOMAIN. Действие по отмене ограничения домена (DROP) приводит к исчезновению соответствующей части общего ограничения соответствующего домена, что, естественно, не влияет на существующие значения столбцов имеющихся таблиц.

Отмена домена.

Чтобы отменить ранее созданное определение домена, нужно воспользоваться оператором DROP DOMAIN в следующем синтаксисе:

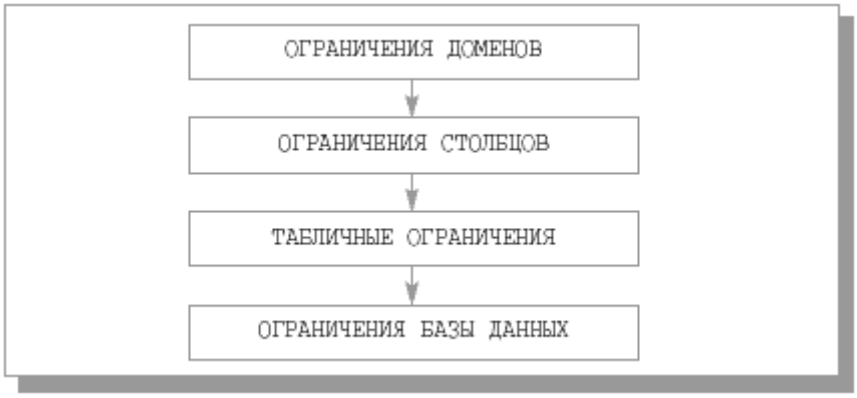
DROP DOMAIN domain_name {RESTRICT | CASCADES}

45. Средства определения, изменения и отмены определения базовых таблиц в SQL.

Определение базовой таблицы.

Определение столбца: возможно использование определенного ранее домена.

Определение ограничений целостности: для столбца, для таблицы, для всей базы. Ограничения первичного ключа, ограничения внешнего ключа.



Пример:

```
(1)   CREATE TABLE EMP (
(2)   EMP_NO EMP_NO PRIMARY KEY,
(3)   EMP_NAME VARCHAR(20) DEFAULT 'Incognito' NOT NULL,
(4)   EMP_BDATE DATE DEFAULT NULL CHECK (
      VALUE >= DATE '1917-10-24'),
(5)   EMP_SAL SALARY,
(6)   DEPT_NO DEPT_NO DEFAULT NULL REFERENCES
      DEPT ON DELETE SET NULL,
(7)   PRO_NO PRO_NO DEFAULT NULL,
(8)   FOREIGN KEY PRO_NO REFERENCES PRO (PRO_NO)
      ON DELETE SET NULL,
(9)   CONSTRAINT PRO_EMP_NO CHECK
    ((SELECT COUNT (*) FROM EMP E
      WHERE E.PRO_NO = PRO_NO) <= 50));
```

46. Базовые средства манипулирования данными в языке SQL.

К базовым средствам манипулирования относятся UPDATE, DELETE и INSERT.

INSERT для вставки строк в существующие таблицы:

Общий синтаксис оператора *INSERT* выглядит следующим образом:

```
INSERT INTO table_name
    { [ (column_commalist) ] query_expression
    | DEFAULT VALUES
```

где query_expression – может быть любым выражением-запросом.
Так например возможна вставкв всех строк какой-то таблицы:

```
INSERT INTO EMP (EMP_NO, EMP_NAME, EMP_BDATE) TABLE EMP_TEMP;
```

Вставка одной строки (используется конструктор строки):

```
INSERT INTO EMP
  ROW (2445, 'Brown', '1985-04-08', 16500.00, 630, 772);
```

Оператор UPDATE для модификации существующих строк в существующих таблицах

Общий синтаксис оператора UPDATE выглядит следующим образом:

```
UPDATE table_name SET update_assignment_commalist
   WHERE conditional_expression
update_assignment ::= column_name =
   { value_expression | DEFAULT | NULL }
```

Примеры:

Простой:

```
  UPDATE EMP SET DEPT_NO = 632, EMP_SAL = EMP_SAL + 1000.00
  WHERE PRO_NO = 772;
```

Сложный:

```
    UPDATE EMP SET EMP_SAL = (SELECT AVG (EMP1_SAL)
    FROM EMP EMP1
    WHERE EMP.DEPT_NO = EMP1.DEPT_NO)
       + 1000.00, PRO_NO = NULL
    WHERE (SELECT EMP1.EMP_SAL
        FROM EMP EMP1, DEPT
        WHERE EMP.DEPT_NO = DEPT.DEPT_NO
          AND DEPT_MNG = EMP1.EMP_NO AND) > 30000.00;
```

Оператор DELETE для удаления строк в существующих таблицах

Общий синтаксис оператора DELETE выглядит следующим образом:

```
DELETE FROM table_name
    WHERE conditional_expression
```

Пример:
Простой:

```
      DELETE FROM EMP WHERE PRO_NO = 772;
```

Сложный:

```
      DELETE FROM EMP WHERE EMP_SAL >
      (SELECT EMP1.EMP_SAL
       FROM EMP EMP1, DEPT
       WHERE EMP.DEPT_NO = DEPT.DEPT_NO
         AND DEPT.DEPT.MNG = EMP1.EMP_NO);
```

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

Триггером называется хранимая в базе данных процедура, автоматически вызываемая СУБД при возникновении соответствующих условий.

В языке обеспечиваются возможности определения триггеров, которые вызываются («срабатывают») при вставке одной или нескольких строк в указанную таблицу, при модификации одной или нескольких строк в указанной таблице или при удалении одной или нескольких строк из указанной таблицы.

Таблица, с которой связывается определение триггера, называется *предметной таблицей* (subject table), а оператор SQL, выполнение которого приводит к срабатыванию триггера, мы будем называть *инициирующим* (triggering SQL statement).

Основными областями использования механизма триггеров являются:

- *Журнализация и аудит*. С помощью триггеров можно отслеживать изменения таблиц, для которых требуется поддержка повышенного уровня безопасности. Данные об изменении таблиц могут сохраняться в других таблицах и включать, например, идентификатор пользователя, от имени которого выполнялась операция обновления; временную метку операции обновления; сами обновляемые данные и т. д.
- *Согласование и очистка данных*. С любым простым оператором SQL, обновляющим некоторую таблицу, можно связать триггеры, производящие соответствующие обновления других таблиц. Например, с операцией вставки новой строки в таблицу EMP (прием на работу нового служащего) можно было связать триггер, модифицирующий значения столбцов DEPT_EMP_NO и DEPT_TOTAL_SAL[162)] строки таблицы DEPT со значением столбца DEPT_NO, которое соответствует номеру отдела нового служащего.
- *Операции, не связанные с изменением базы данных*. В триггерах могут выполняться не только операции обновления базы данных. Стандарт SQL позволяет определять хранимые процедуры (которые могут вызываться из триггеров), посылающие электронную почту, печатающие документы и т. д.

Типы триггеров и их выполнение:

Триггеры BEFORE и AFTER

Если в определении триггера указано ключевое слово BEFORE, то триггер будет срабатывать непосредственно до выполнения операции обновления базовой таблицы соответствующим инициирующим оператором SQL. При задании ключевого слова AFTER триггер будет вызываться немедленно после выполнения инициирующего оператора.

Триггеры INSERT, UPDATE и DELETE

Выбор одного из этих ключевых слов при определении триггера указывает на природу события, которое должно приводить к срабатыванию триггера. При задании ключевого слова INSERT к срабатыванию триггера может привести только выполнение операции вставки строк в предметную таблицу. Если указываются ключевые слова UPDATE или DELETE, то число возможных событий, приводящих к срабатыванию триггера, возрастает. Кроме явных операций модификации строк предметной таблицы или удаления из нее строк к срабатыванию триггера могут привести ссылочные действия

Триггеры ROW и STATEMENT

Если в определении триггера присутствует конструкция FOR EACH ROW, то триггер будет вызываться для каждой строки предметной таблицы, обновляемой инициирующим SQL-оператором. Если же задано FOR EACH STATEMENT (или явная спецификация FOR EACH отсутствует), то триггер сработает один раз на всем протяжении процесса выполнения инициирующего SQL-оператора.

Раздел WHEN

Включение в определение триггера раздела WHEN с соответствующим условным выражением позволяет более точно специфицировать условие применимости триггера. Вычисление условного выражения производится над строками предметной таблицы, и триггер срабатывает только в том случае, когда значением условного выражения является true. Понятно, что виды и интерпретация логических выражений, допускаемых в разделе WHEN, различаются у триггеров с FOR EACH ROW и у триггеров с FOR EACH STATEMENT. В первом случае условное выражение вычисляется для одной строки, которая должна быть обновлена инициирующим SQL-оператором. Во втором – условное выражение вычисляется для всей предметной таблицы целиком и, по всей видимости, должно базироваться на «кванторных» предикатах. Следует также понимать, что вычисление условия раздела WHEN данного триггера производится только в том случае, если произошло событие срабатывания триггера.

Выполнение триггеров.

По ходу выполнения триггера возможна ситуация, при которой вызывается внутренний (вторичный) триггер, а также третичный и.т.д – глубина вложенности вызовов неограниченна. (Рекурсия, короче).

Выполнению триггера соответствует контекст выполнения. При завершении выполнения внутреннего триггера контекст меняется на контекст внешнего триггера.

48. Общая структура оператора выборки в языке SQL и схема выполнения.

Для выборки данных в прямом SQL используется оператор SELECT, возвращающий набор из одной или нескольких строк одинаковой структуры и задаваемый в следующем синтаксисе:

```
SELECT [ ALL | DISTINCT ] select_item_commalist
  FROM table_reference_commalist
   [ WHERE conditional_expression ]
    [ GROUP BY column_name_commalist ]
     [ HAVING conditional_expression ]
      [ ORDER BY order_item_commalist ]
```

На первом шаге выполняется раздел FROM.

На втором шаге выполняется раздел WHERE. Условное выражение (conditional_expression) этого раздела применяется к каждой строке таблицы T, и результатом является таблица T1, содержащая те и только те строки таблицы T, для которых результатом вычисления условного выражения является true. (Заголовки таблиц T и T1 совпадают.)

Если в операторе выборки присутствует раздел GROUP BY, то он выполняется на третьем шаге.

Каждый элемент списка имен столбцов (column_name_commalist), указываемого в этом разделе, должен быть одним из имен столбцов таблицы T1. В результате выполнения раздела GROUP BY образуется *сгруппированная таблица* T2, в которой строки таблицы T1 расставлены в минимальное число групп, таких, что во всех строках одной группы значения столбцов, указанных в списке имен столбцов раздела GROUP BY (*столбцов группировки*), одинаковы.

Если в операторе выборки присутствует раздел HAVING, то он выполняется на следующем шаге. Условное выражение этого раздела применяется к каждой группе строк таблицы T2, и результатом является сгруппированная таблица T3, содержащая те и только те группы строк таблицы T2, для которых результатом вычисления условного выражения является true. Условное выражение раздела HAVING строится по синтаксическим правилам, общим для всех условных выражений, но обладает той спецификой, что применяется к группам строк, а не к отдельным строкам. Поэтому предикаты, из которых строится это условное выражение, должны быть предикатами на группу в целом. В них могут использоваться имена столбцов группировки (*инварианты группы*) и так называемые агрегатные функции (COUNT, SUM, MIN, MAX, AVG) от других столбцов.

Выполнение раздела ORDER BY производится в последнюю очередь, следующим образом. Выбирается первый элемент списка сортировки, и строки таблицы T4 расставляются в порядке возрастания (если в элементе присутствует спецификация ASC; при отсутствии спецификации ASC/DESC предполагается наличие ASC) или в порядке убывания (при наличии спецификации DESC) в соответствии со значениями выражения, содержащегося в данном элементе, которые вычисляются для каждой строки таблицы T4. Далее выбирается второй элемент списка сортировки, и в соответствии со значениями заданного в нем выражения и порядка сортировки расставляются строки, которые после первого шага сортировки образовали группы с одинаковым значением выражения первого элемента списка сортировки. Операция продолжается до исчерпания списка элементов сортировки. Результирующий отсортированный список строк является окончательным результатом запроса.

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

Представляемая таблица – представление таблице, которое можно использовать в операторе выборки наряду с базовыми таблицами
(table_reference ::= table_primary).
оператор создания представления в общем случае определяется следующими синтаксическими правилами (простой вид):
create_view ::= CREATE VIEW table_name
[column_name_comma_list]
AS query_expression - запрос
Имя таблицы, задаваемое в определении представления, существует в том же пространстве имен, что и имена базовых таблиц, и, следовательно, должно отличаться от всех имен таблиц (базовых и представляемых), созданных тем же пользователем.
Если имя представления встречается в *разделе FROM* какого-либо оператора выборки, то вычисляется *выражение запроса*, указанное в разделе AS, и оператор выборки работает с результирующей таблицей этого *выражения запроса*.

Порождаемая таблица derived_table задается *выражением запроса*, заключенным в круглые скобки:
derived_table ::= (query_expression)

Агрегатные и кванторные функции

агрегатные:
Всё это функции над мультимножеством строк (сгруппированных, если GROUP BY, или всей таблицей). На осн. м/множества строк производятся множества значений

COUNT – количество значений
MAX
MIN
AVG – среднее значение
SUM

COUNT (*) – количество строк (всегда считаются различными)

(аргументы следующих **квантроных функций** – СУТЬ логические выражения)

EVERY: true , если лог. выр. True для всех строк
SOME (ANY): true , если лог. выр. True для хотя бы одной строки

DISTINCT – удаление дубликатов строк из результата.

Если на входе пустое множество, то
COUNT = 0 EVERY = true MAX, MIN, AVG, SUM = NULL

50. Предикаты языка SQL.

Логич. выражение – булево выражение, которое строится на основе предикатов с исп.операторов AND, OR, NOT И ()

S – скал.произведение

R – строк.табл

|R| - значение в строке

T – таблица

Предикат позволяет специфицировать усл-е, результатом которого может быть true, false или unknown.

Для любых аргументов предикатов правила:

1) совместимости типов
2) равенство степеней строк. операндов |Rx| = |Ry|
3) для люб. предик. есть обратный NOT pred = NOT (pred)

Предикаты:

1) Предикат сравнения: Rx comp-op Ry
2) Предикат between: условие вх-я в диап. Значений (Rx between Ry and Rz)
3) Предикат is null: является неопр.значением аргумента
4) Предикат in: условие вхождения в указ.множество
5) Предикат like: string LIKE pattern ESCAPE symbol
   в pattern - любой одиноч.символ, % - произв. п/строка
6) Предикат similar
7) Предикат exists: получили ли пустую таблицу (TRUE ⇔ | $T_{рез}$ |>0 , иначе FALSE)
8) Предикат unique: = TRUE, когда в $T_{рез}$ отсут. 2 одинак. стр
9) Предикат overlaps: пересечение по времени 2х событий
10) Предикат сравнения с квантором:
    Rx comp_op ALL T (квантор всеобщности)
    Rx comp_op SOME T (квантор существования)
11) Предикат match: соответствие подстроки рез-ту 2/л запросу
12) Предикат is distinct: Rx IS DISTINCT FROM Ry = TRUE ⇔ Rx и Ry - дубликаты

51. Управление транзакциями в SQL. Средства инициации и завершения транзакций. Понятие точки сохранения. Уровни изоляции SQL-транзакций.

Транзакция ACID – послед. операций, обладающих св-вами атомарности, согласованности, изоляц, долговечн.

Транзакция , образованная явным образом: START TRANSACTION либо неявно, когда выполняется оператор, для которого требуется контекст транзакции, а его нет.

Для завершения транзакции: COMMIT (транзакция сохр-ся в БД)
ROLLBACK (система возвращается в сост-е до старта)

SET TRANSACTION mode_commalist
mode ::=isolation_level
| access_mode
| diagnostics_size
isolation_level ::= READ UNCOMMITED - уровень изоляции
| READ COMITTED
| REPEATABLE READ
| SERIALIZABLE
access_mode ::= READ ONLY - режим доступа
| READ WRITE
diagnostics_size ::= DIAGNOSTIC SIZE value_specification - № диап.элем-в, кот. могут размещаться в области диагностики

Точки сохранения.

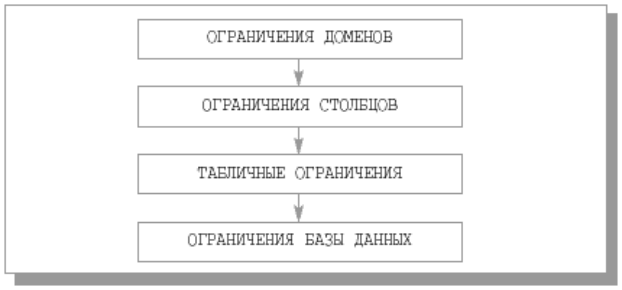
Точка сохранения представляет собой своего рода пометку в последовательности операций транзакции, которую в дальнейшем можно использовать для частичного отката транзакции с сохранением жизнеспособности транзакции и результатов операций, выполненных в транзакции до точки сохранения.
Установления точки сохранения: SAVEPOINT savepoint_name

Уровни изоляции SQL-транзакции

В стандарте SQL:1999 уровни изоляции определяются на основе нескольких феноменов, которые могут возникать при выполнении транзакций:

- **Феномен «грязного» чтения (dirty read)** - Этому феномену подвержены транзакции, в которых допускается возможность видеть изменения объектов базы данных, производимые другими одновременно выполняемыми и еще не зафиксированными транзакциями. В SQL феномен «грязного» чтения может наблюдаться у транзакций, выполняемых на уровне изоляции READ UNCOMMITTED
- **Феномен неповторяемого чтения (unrepeatable read)** - Этому феномену подвержены транзакции, читающие некоторые объекты базы данных и допускающие изменения уже прочитанных объектов другими транзакциями. В SQL феномен неповторяемого чтения может наблюдаться у транзакций, выполняемых на уровне изоляции READ COMMITTED (этот уровень изоляции, как показывает его название, гарантирует отсутствие феномена «грязного» чтения).
- **Феномен фантомов** - Этому феномену подвержены транзакции, производящие выборку строк и таблиц базы данных и допускающие добавление к данным таблицам другими транзакциями строк, которые удовлетворяют условию выборки. В SQL феномен фантомов может наблюдаться у транзакций, выполняемых на уровне изоляции REPEATABLE READ (этот уровень изоляции, как показывает его название, гарантирует отсутствие феномена неповторяемого чтения).

52. Иерархия видов ограничений целостности в SQL.



В стандарте SQL дополнительные ограничения базы данных называются ASSERTION, т.е. **общие ограничения целостности (ограничения БД)**.
Для определения общего ограничения целостности служит оператор *CREATE ASSERTION*, задаваемый в следующем синтаксисе:

```
CREATE ASSERTION constraint_name
   CHECK (conditional_expression)
```

Пример:

```
CREATE ASSERTION MIN_EMP_BDATE CHECK
   ((SELECT MIN(EMP_BDATE)) FROM EMP) >= '1917-10-24')
```

В контексте каждой выполняемой транзакции каждое ограничение целостности должно находиться в одном из двух режимов:
1) немедленной проверки (immediate): проверяются при выполнении в транзакции любой операции, изменяющей состояние базы данных, и, возможно ROLLBACK,
2) отложенной проверки (deferred): проверяются при завершении транзакции (выполнении операции COMMIT).
спецификация INITIALLY для ограничения целостности (любого вида):

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

Подробнее:
INITIALLY IMMEDIATE - в режиме *немедленной проверки*,
INITIALLY DEFERRED – находиться в режиме отложенной проверки, а также DEFERRABLE - для данного ограничения м б установлен режим отложенной проверки, NOT DEFERRABLE – что не может,
INITIALLY DEFERRED NOT DEFERRABLE является недопустимой возможности смены режима подразумевается наличие спецификации DEFERRABLE.

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

1) Метод авторизации в SQL относится к мандатным методам.

2) В SQL – 9 типов привилегий + привилегия на передачу привилегий.

3) Поддерживается принцип сокрытия информации об объектах от субъектов, у которых нет доступа.

4) Субъекты в SQL – пользователи и роли. Роль соответствует динамически определенной группе, каждая из которых отражает привилегии на исполнение данной роли, а также всеми привилегиями данной роли для доступа к объектам БД

54. Передача и аннулирование привилегий и ролей в SQL.

1) Создатель любого объекта БД автоматически становится его владельцем. Владелец облает всеми привилегиями для выполнения действий над данным объектом. Он может передать все/часть своих привилегий другим пользователям или ролям, в том числе привилегию на передачу привилегий (рекурсия).
2) С любой пользой и любой ролью в SQL связан уникальный идентификатор инициализаций. Средства создания идентификатора пользователя в стандарте SQL не определено. Но должен существовать пользователь с идентификатором PUBLIC – псевдопольз., соотв. любому пользователю. Для создания/удаления роли поддерживаются операторы CREATEROLE и DROPROLE.
3) Для передачи привилегии роли поддерживается оператор GRANT, для удаления – REMOVE.

55. Объектно-ориентированная модель данных. Ее структурная, манипуляционная и целостная части. Реализации

Объектная БД – набор контейнеров произвольного типа (объекты)

Типы данных:

Литеральные:

1) атомарные
    - числовые
    - строковые
    - темпоральные
    - логические
2) конструированные
    - структуры
    - коллекции (мн-ва, мульти мн-ва, массивы, списки, словари)

Объектные:

1) атомарные
2) объект: коллекция

Литеральные – только в качестве типов атрибутов внутри объекта значение д. типов не имеют идентификаторов и не могут самостоятельно храниться в БД.

Объектные – у любого типасуществует конструктор (созд. и инициал.) и отдельный атрибут OID (объект. идентификатор). OID уникален по всей БД (ключ – только в пределах таблицы). OID генерирует БД и не зависит от состояния объекта (ключ зависит от состояния объекта)

3 вида эквивалентностей объектов:

- идентичность (идентификаторы совпадают)
- эквивалентность по значению (значение всех их атрибутов одинаково, в случае сложных объектов - рекурсивно)
- поверхностная эквивалентность, когда атрибуты идентичны (простые – по значению, объектные – по идентификатору)

Объектные коллекции могут храниться в БД самостоятельно и имеют идентификатор.

Манипулирование данными

Операторы могут вызываться из любого языка программирования и наоборот

В OQL результатом запроса является:

- индивидуальный объект
- коллекция объектов
- коллекция литеральных значений
- индивидуальные литеральные значения

ODL позволяет описать схему БД в виде набора интерфейсов объектных типов.

Подходы к созданию объектов:

1) устойчивость – характеристика всех экземпляров всех объектных типов
2) вызов конструктора предпол. создание транзитного экземпляра
3) смешанный: тип объекта определяется параметрами конструктора

Удаление объектов:

1) объект физически удаляется в любом случае
2) объект имеет счетчик ссылок. При ненулевом значении счетчика выставляется флаг уничтожения, а объект физически удаляется после того, как счетчик=0.

Ограничения целостности:

объекты совпадают тогда и только тогда, когда они идентичны, значит, нет ограничения целостности как в реляц. БД. Не существует потомков без предка.

Реализации: Versant, Objectivity DB, db4objects (Java/.NET)

с середины 2000х ODBMS.ORG – 2008 – новый стандарт

56. Объектно-реляционные расширения языка SQL. Возможные подходы к объектно-реляционному отображению без использования объектно-реляционных расширений SQL.

Определение пользователем типов данных и типизированных таблиц.

UDT:

- Индивидуальный тип. Основан на единственном предопределенном типе (типа typedef), но без наследственных операций, их нужно указывать.
- Структурный тип. Именованный тип данных, включающий 1 или более атрибутов любых из допустимых в SQL типов данных, в том числе в данных структурного типа можно использовать механизм наследственности от ранее определенного структурного типа.

При определении типизированных таблиц указывается ранее определенный структурный тип, и если в нем N атрибутов, то в таблице N+1 столбец “лишний” – самоссылающийся и содержит типизированный уникальный идентификатор строк.

Способ генерации значений при определении структурного типа

2) SYSTEM GENERATED – аналог ООБД
3) USER DEFINED (TYPE) – опр. тип уникального идентификатора, пользователь указывает значение, когда вставляет строки
4) USER DEFINED (список столбцов) – аналог ключа

57. Истинная реляционная модель данных. Ее структурная, манипуляционная и целостная части. Реализации.

Типы данных

- скалярный (инкапсулир. тип, внутренняя структура которого скрыта от пользователя)
- кортежный (определяется указанием заголовка в вие множества пар <имя атр., тип атр.>, значение – триплет <имя, тип, значение>)
- отношение (определяется заголовком <имя, тип>, значение-заголовок, совпадение заголовка в определении + тело как множество кортежей)

Любой кортеж в отношении содержит ровно 1 значение для любого атрибута, значит, отношение в 1NF.
БД – набор долговременно хранимых отношений.
Манипуляция данными:

- алгебра A
- язык D (принципы: для запросов – алгебраический подход, запросы, адрес к сложным данным, формулируется более точно, чем на SQL.)

Ограничения целостности:

1) обязательное определение хотя бы 1 возможного ключа
2) поддержка ограничения целостности как произвольных условных выражений, которые логически эквивалентны замкнутой правильно построенной формуле.
3) Рекомендуется использовать ср. под. ссылоч. целостности DATAPHOR.