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

Файл - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные.

Функции системы управления файлами:

- распределение внешней памяти
- отображение имен файлов в соответствующие адреса во внешней памяти
- обеспечение доступа к данным

Типы файловых систем:

- Изолированная файловая система — каждый архив файлов (полное дерево каталогов) целиком располагался на одном дисковом пакете или логическом диске, полное имя файла начинается с имени дискового устройства.
- Централизованная (Multics) — вся совокупность каталогов и файлов представляется как единое дерево, физически распределенное по всем внешним запоминающим устройствам.
- Смешанная (*nix) — на базовом уровне в этих ФС поддерживаются изолированные архивы файлов. Один из этих архивов объявляется корневой файловой системой. А потом еще используется mount для подсоединения других устройств со своими файловыми каталогами.

Защита ФС:

- Мандатный способ - по отношениюль к каждому зарегистрированному пользователю данной системы для каждого существующего файла указываются действия, которые разрешены/запрещены. Каждый пользователь имеет отдельный мандат для работы с каждым файлом.
- Дискреционный способ - каждому зарегистрированному пользователю соответствует пара целочисленных идентификаторов: собственный идентификатор и идентификатор группы, к которой относится пользователь. Каждый процесс, запущенный от имени пользователя, снабжается идентификатором этого пользователя. Каждый файл системы хранит полный идентификатор пользователя, который его создал, и информацию о том, какие действия с файлом может производить сам этот пользователь/остальные пользователи той же группы/пользователи других групп. Для каждого файла контролируется возможность чтения, записи, выполнения.

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

СУБД - система управления данными, выполняющая следующий набор функций:

- управление логически согласованными данными в долговременной памяти
- управление буферами оперативной памяти
- управление транзакциями
- журнализация и восстановление БД после сбоев
- поддержка языков БД

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.

- *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 изменения БД (commit), произведенные этой транзакцией во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД.

Свойства ACID:

- Атомарность - транзакция выполняется как атомарная операция (либо выполняется вся транзакция целиком, либо она целиком не выполняется)
- Согласованность - транзакция переводит БД из одного согласованного(целостного) состояния в другое согласованное(целостное) состояние. Внутри согласованность БД может нарушаться.
- Изоляция - транзакции разных пользователей не должны мешать друг другу.
- Долговечность - если транзакция выполнена, то результаты ее работы должны сохраниться а БД, даже если в следующий момент произойдет сбой системы.

Сериализация транзакций (сериализованный план выполнения смеси транзакций) - такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения.

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

Надежность СУБД - ее способность восстановить последнее согласованное состояние БД после любого сбоя.

Классификация сбоев:

- аппаратный сбой:
  - мягкий - внезапная остановка работы компьютера или аварийное выключение питания
  - жесткий - потеря информации на носителях внешней памяти
- программный сбой:
  - аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя). Можно рассматривать как особый вид мягкого аппаратного сбоя, т.к. требуется ликвидировать последствия только одной транзакции
  - аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной

Поддержание надежности хранения данных в БД достигается избыточностью хранимых данных за счет ведения журнала изменений БД.

Журнал - особая часть БД, недоступная пользователям СУБД. В журнал поступают записи обо всех изменениях основной части БД.

Уровни журнализации:

- логический - запись в журнале соответствует некоторой логической операции изменения БД (удаление строки из таблицы реляционной БД)
- физический - запись в журнале соответствует минимальной внутренней операции модификации страницы внешней памяти (изменение сегмента внешней памяти в таком-то интервале адресов)

Основная идея журнализации. Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). 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.
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 транзакций(RollBack)
- 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.
- Для восстановления БД после ***жесткого сбоя*** используют журнал и архивную копию БД. Архивная копия - это полная копия БД к моменту начала заполнения журнала. Восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя.

5. Ранние дореляционные подходы к организации баз данных.

Общие характеристики ранних систем:

- все ранние системы не основывались на каких-либо абстрактных моделях и не имели какой-либо математической модели
- в ранних системах доступ к БД производился на уровне записей
- вся оптимизация доступа к БД осуществлялась пользователем
- большинство ранних систем было впоследствии оснащено “реляционными” интерфейсами

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

Сетевые системы (например, Integrated Database Management System (IDMS)). Основаны на произвольных (возможно циклических) ориентированных графах и включают набор записей (это экземпляры типа записи) и набор связей между этими записями (это экземпляры типа 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.

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:

- Строки таблиц упорядочены системой в некоторой физической последовательности.
- Физическая упорядоченность строк всех таблиц может определяться и для всей БД (так делается, например, в Datacom/DB).
- Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.
- Общие правила определения целостности БД отсутствуют (возлагается на прикладную программу).
- Явное оперирование с адресами в памяти.

Достоинства и недостатки ранних дореляционных СУБД:

- Достоинства: развитые средства управления данными во внешней памяти на низком уровне, и как следствие, возможность доработки высокоэффективных прикладных систем.
- Недостатки: необходимость знания прикладной системы физической организации БД + отсутствие или сильная ограниченность правил определения целостности БД

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

Тип данных — множество значений, над которыми определены операции, а также представление этих чисел при выводе (литералы). Стандартные типы данных современных реляционных БД: символьные, числовые, специальные “числовые”(деньги), специальные “темпоральные” (дата, время), битовые строки.

Домены — подтипы, унаследованы от типов, имеют некоторые ограничения. Семантически: данные сравнимы, если они в одном домене. (домен - допустимое множество значений данного типа).

Схема отношения (заголовок отношения, отношение - схема) — конечное именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}.

Степень или "арность" схемы отношения — мощность этого множества.

Кортеж, соответствующий данной схеме отношения — множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме 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, такое что в любое время значение первичного ключа однозначно идентифицирует кортеж, а любое другое собственное множество его 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.

1НФ — первая нормальная форма: таблица находится в первой нормальной форме (1НФ), если 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. 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): у любого отношения должен существовать первичный ключ, и никакое значение первичного ключа не может быть не определено. NULL - неопределенное значение для любого типа данных. Для булевских вводится значение unknown.

Требование целостности по ссылкам (referential integrity): 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 полностью 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.
- 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.

Реляционная алгебра - набор **теоретико-множественных** и **реляционных** операций над отношениями реляционных БД.
Алгебра Кодда состоит из 8-ми основных операций и 2-х дополнительных.
Теоретико множественные:

- Объединение (UNION)
- Пересечение (INTERSECT)
- Разность (MINUS)
- Декартово произведение (TIMES)

Реляционные:

- Ограничение (WHERE)
- Проекция (PROJECT)
- Соединение (JOIN)
- Реляционное деление (DIVIDE BY)

Дополнительные:

- Переименование (RENAME)
- Присваивание (:=)

Приоритет операций:
RENAME >= WHERE = PROJECT >= TIMES = JOIN = INTERSECT = DIVIDE BY >= UNION = MINUS

Замкнутость реляционной алгебры:
Каждая операция алгебры должна быть замкнута относительно отношения, т. е. **результатом операции** всегда должно быть **отношение**, содержащее **заголовок** (иначе - схема, т. е. множество пар <имя_атрибута, имя_домена>) и **тело** (множество кортежей, т.е. пар <имя_атрибута, значение>, где значение принадлежит соответствующему домену).

Алгебра Кодда не является замкнутой из-за невозможности применить некоторые операции без предварительного переименования атрибутов.

Пример: Применение операции декартова произведения к отношениям, имеющим как минимум одну общую пару <имя-атрибута, имя-домена> (Недопустима как потеря, так и дублирование такой пары в результирующем заголовке, причем первое - несоответствие логике операции, второе - запрещено в реляционных БД. Проблема решается переименованием атрибута в одном из отношений).

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

Рассмотрим теоретико-множественные операции алгебры Кодда подробнее:

- **Объединение** (UNION): применяется к отношениям с **одинаковыми заголовками**, включает кортежи, входящие в тело хотя бы одного из отношений-операндов.
  (Объединяет строки двух таблиц в одну)

- **Пересечение** (INTERSECT): применяется к отношениям с **одинаковыми заголовками**, включает кортежи, входящие в тела обоих отношений-операндов.
  (Оставляет только те строки, что содержатся в обеих таблицах)

- **Разность** (MINUS): применяется к отношениям с **одинаковыми заголовками**, включает кортежи, входящие в тело первого отношения-операнда, но не входящие в тело второго.
  (Строки, которые есть в первой таблице, но нет во второй)

- **Декартово произведение**[1] (TIMES): применяется к отношениям с **непересекающимися заголовками**, состоит из кортежей, полученных путем объединения кортежей (как множеств пар <имя-атрибута, значение>), входящих в тела соответствующих отношений-операндов.

Под отношениями совместимыми по некоторой операции понимаются те отношения, на которых операция замкнута, т. е. возвращает корректное отношение.

Совместимость отношений по объединению:
Отношения совместимы по объединению тогда и только тогда, когда **обладают обладают одинаковыми заголовками**.
При этом, **если** у отличающихся пар <имя-атрибута, имя-домена> **соответствуют имена доменов**, то отношения “почти совместимы” и могут быть сведены к совместимым с помощью операции переименования.
(Примечание: выполнение всех этих условий также влечет корректность операций пересечения и разности)

Совместимость отношений по декартову произведению:
Отношения совместимы по декартову произведению тогда и только тогда, когда **пересечение имен атрибутов этих отношений пусто**.
Отношения **всегда** можно сделать совместимыми по декартову произведению с помощью операции переименования.

[1] Фактически, эта операция является *расширенным декартовым произведением*, так как результат - не пара элементов из соответствующих множеств, а их объединение

9. Реляционная алгебра Кодда. Специальные реляционные операции.

Рассмотрим реляционные операции алгебры Кодда подробнее:

- **Ограничение** (A WHERE comp): операндами являются отношение и условие ограничения. Меняется только тело отношения, в котором остаются только те кортежи, для которых выполнено условие.
  Изначально операция определяется для простых условий вида:
  - a comp-op b;
  - a comp-op const;

  где a и b - имена атрибутов, comp-op оператор сравнения.
  При этом с помощью теоретико-множественных операций ограничение можно обобщить и для условий более сложного вида, представляющих произвольные булевы выражения над простыми условиями:
  - A WHERE (comp1 AND comp2) ⇔ (A WHERE comp1) INTERSECT (A WHERE comp2);
  - A WHERE (comp1 OR comp2) ⇔ (A WHERE comp1) UNION (A WHERE comp2);
  - A WHERE NOT comp1 ⇔ A MINUS (A WHERE comp1);

  где comp1 и comp2 - условия простого вида.
- **Проекция** (PROJECT A {a1, a2, …, aN}): операндами являются отношение и подмножество его имен-атрибутов.
  Заголовок результирующего отношения определяется вторым операндом, а тело состоит из кортежей, каждый из которых - подмножество пар <имя_атрибута, значение> соответствующего исходного кортежа, состоящее только из пар, содержащих имена-атрибуты из второго операнда.
  (Фактически вырезает столбцы таблицы)

- **Соединение** (A JOIN B WHERE comp): операндами являются два отношения и условие. Результат эквивалентен (A TIMES B) WHERE comp, то есть ограничению декартова произведения.[2]
  Часто используемым частным случаем является *эквисоединение* (EQUIJOIN), определяемое как соединение с условием a = b. При этом, если эквисоединение производится по общему для отношений имени-атрибуту c: A.c = B.c,[3] то такую операцию называют *естественным соединением* (NATURAL JOIN).

- **Реляционное деление** (DIVIDE BY): операндами являются отношения A{a, b} и B{b} соответственно (здесь a и b не отдельно взятые имена-атрибуты, а **множества пар**, принадлежащих заголовкам отношений). То есть, заголовок отношения B полностью вложен в заголовок отношения A.
  Заголовок результирующего отношения состоит только из {a}.
  Тогда телом результирующего отношения будет множество кортежей **{v}** такое, что множество кортежей **{v} union {w}** принадлежит телу **A**, а **{w}** включает **все** тело **B**.

  Пример: СЛУЖАЩИЕ *DIVIDE BY* НОМЕРА_ПРОЕКТОВ вернет данные (соответствующие атрибутам отношения СЛУЖАЩИЕ, но за исключением номера проекта) тех служащих, что участвуют во всех проектах, перечисленных во втором операнде.

---

[2] На практике сама операция реализуется более эффективным способом, но представление через ограничение декартова произведения достаточно интуитивно.
[3] Через точку обозначена принадлежность соответствующему отношению.

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

Алгебра Кодда, несмотря на практичность, избыточна. Рассмотрим минимальную реляционную алгебру на примере алгебры A, предложенной Дейтом и Дарвеном.

Для этого введем более формальные обозначения:
Пусть r – отношение, A – имя атрибута отношения r, T – имя соответствующего типа или домена, v – значение типа T.

- Заголовком **Hr** отношения r называется **множество** пар атрибут-домен, т.е. упорядоченных пар вида **<A, T>**, по одной такой паре для каждого имени-атрибута в r.
- Кортеж **tr**, соответствующий заголовку Hr, – **множество** упорядоченных **триплетов** вида **<A, T, v>**, по одному такому триплету для каждого атрибута в Hr.
- Тело **Br** отношения r – **множество кортежей tr**. При этом не все допустимые Hr кортежи обязаны быть в Br.

Рассмотрим базовые операции алгебры A:

- **Реляционное дополнение**. Пусть **s** - результат операции **<NOT> r**.
  - Hs = Hr
  - Bs = {ts : exists tr (tr ∉ Br and ts = tr)}
    Заполняем тело нового отношения всеми кортежами допустимыми Hr, но не входящими в тело отношения r.

  Пример: <NOT> НОМЕРА_ПРОЕКТОВ.
  Пусть состав домена ДОПУСТИМЫЕ_НОМЕРА_ПРОЕКТОВ, на котором определен атрибут ПРО_НОМ отношения НОМЕРА_ПРОЕКТОВ, входит всего пять значений {1, 2, 3, 4, 5}.Тогда:

НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
1
2

<NOT> НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
3
4
5

- **Удаление атрибута**. Пусть **s** обозначает результат операции **r <REMOVE> A**.
  Для корректности операции необходимо, чтобы ∃ T: <A, T> ∈ Hr.
  - Hs = Hr minus {<A, T>}
  - Bs = {ts: exists tr exists v (tr ∈ Br and v ∈ T and <A, T, v> ∈ tr and ts = tr minus {<A,T,v>})}
    Из каждого кортежа тела отношения r выкидываем триплет, содержащий имя-атрибут A.

  Пример: СЛУЖАЩИЕ <REMOVE> ПРО_НОРМ.

СЛУЖАЩИЕ

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
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

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

- **Переименование**. Пусть **s** обозначает результат операции **r <RENAME> (A, B)**.
  Для корректности операции необходимо, чтобы ∃T1: <A, T1> ∈ Hr и <B, T2> ∉ Hr ∀T2.
  - Hs = (Hr minus {<A, T>}) union {<B, T>}
  - Bs = {ts: exists tr exists v (tr ∈ Br and v ∈ T and <A, T, v> ∈ tr and ts = (tr minus {<A, T, v>}) union {<B, T, v>})}
    Заменяем имя-атрибут на B в триплетах, содержавших A.

- **Реляционная конъюнкция**. Пусть **s** обозначает результат операции **r1 <AND> r2**.
  Для корректности операции необходимо, чтобы одноименные атрибуты из отношений были определены на одинаковых доменах.
  - Hs = Hr1 union Hr2
  - Bs = {ts: exists tr1 exists tr2 ((tr1 ∈ Br1 **and** tr2 ∈ Br2) and ts = tr1 union tr2)}

  (Здесь следует отметить, что **union** понимается как корректное объединение множеств с точки зрения отношений в реляционной БД и равно пустому множеству, если операнды содержат разные элементы с одинаковыми атрибутами).
  (Чтобы получить tr1 union tr2, нужно взять их объединение, а затем из получившегося "множества триплетов" tr1 U tr2 = {<A, T, v>} удалить триплеты с одинаковыми именами атрибутов A (=> и одинаковыми доменами T), но разными значениями v)
  Отсюда следует, что:
  - если схемы отношений-операндов имеют **непустое пересечение**, то операция <AND> работает как *естественное соединение* (см. NATURAL JOIN);
  - 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.

  Примеры:

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2939	Сидоренко	18000.00	313
2940	Федоренко	20000.00	310
2941	Иваненко	22000.00	315

ПРОЕКТЫ

ПРО_НОМ	ПРОЕКТ_РУК
1	Иванов
2	Иваненко

СЛУЖАЩИЕ

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
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

(a) Результат операции **СЛУЖАЩИЕ** <AND> **ПРОЕКТЫ**

СЛУ_НОМЕР	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
2934	Иванов	22400.00	2	Иваненко
2935	Петров	29600.00	2	Иваненко
2939	Сидоренко	18000.00	2	Иваненко
2940	Федоренко	20000.00	2	Иваненко
2941	Иваненко	22000.00	2	Иваненко

(b) Результат операции **СЛУЖАЩИЕ_В_ПРОЕКТЕ_1** <AND> **ПРОЕКТЫ**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР	ПРО_НОМ	ПРОЕКТ_РУК
2934	Иванов	22000.00	310	1	Иванов
2935	Петров	30000.00	310	1	Иванов
2936	Сидоров	18000.00	313	1	Иванов
2937	Федоров	20000.00	310	1	Иванов
2938	Иванова	22000.00	315	1	Иванов
2934	Иванов	22000.00	310	2	Иваненко
2935	Петров	30000.00	310	2	Иваненко
2936	Сидоров	18000.00	313	2	Иваненко
2937	Федоров	20000.00	310	2	Иваненко
2938	Иванова	22000.00	315	2	Иваненко

(c) Результат операции **СЛУЖАЩИЕ_В_ПРОЕКТЕ_1** <AND> **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310

- **Реляционная дизъюнкция**. Пусть **s** обозначает результат операции **r1 <OR> r2**.
  Также как и для конъюнкции для корректности операции необходимо, чтобы одноименные атрибуты из отношений были определены на одинаковых доменах.
  - Hs = Hr1 union Hr2
  - Bs = {ts: exists tr1 exists tr2 ((tr1 ∈ Br1 **or** tr2 ∈ Br2) and ts = tr1 union tr2)}

  Заметим, что:
  - если у операндов нет общих атрибутов, то в тело результирующего отношения входят все такие кортежи ts, которые являются объединением кортежей tr1 и tr2, 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 tr1 и tr2, соответствующих заголовкам отношений-операндов, если хотя бы один из этих кортежей принадлежит телу одного из операндов, и значения общих атрибутов tr1 и tr2 совпадают;
  - если же схемы отношений-операндов совпадают, то тело отношения-результата является объединением тел операндов.

  Примеры:

ПРОЕКТЫ_1

ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК
ПРОЕКТ 1	Иванов
ПРОЕКТ 2	Иваненко

НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
1
2

Результат операции **ПРОЕКТЫ** <OR> **НОМЕРА_ПРОЕКТОВ**

ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК	ПРО_НОМ
ПРОЕКТ 1	Иванов	1
ПРОЕКТ 2	Иванов	1
ПРОЕКТ 3	Иванов	1
ПРОЕКТ 1	Иваненко	1
ПРОЕКТ 2	Иваненко	1
ПРОЕКТ 3	Иваненко	1
ПРОЕКТ 1	Иванов	2
ПРОЕКТ 2	Иванов	2
ПРОЕКТ 3	Иванов	2
ПРОЕКТ 1	Иваненко	2
ПРОЕКТ 2	Иваненко	2
ПРОЕКТ 3	Иваненко	2
ПРОЕКТ 1	Иванов	3
ПРОЕКТ 2	Иваненко	3

ПРОЕКТЫ_2

ПРО_НОМ	ПРОЕКТ_РУК
1	Иванов
2	Иваненко

ПРОЕКТЫ_2 <OR> **НОМЕРА_ПРОЕКТОВ**

ПРО_НОМ	ПРОЕКТ_РУК
1	Иванов
2	Иваненко
2	Иванов
1	Иваненко

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

Покажем, что Алгебра A является полной, т. е. на основе введенных операций выражаются все операции алгебры Кодда.

Тривиальные соответствия:

- PROJECT - <REMOVE>;
- Переименование - <RENAME>;
- UNION - частный случай операции <OR>;
- TIMES, INTERSECT и NATURAL JOIN – частные случаи операции <AND>;

Остается показать, что через операции Алгебры A выражаются операции взятия разности MINUS, ограничения (WHERE), соединения общего вида (JOIN) и реляционного деления (DIVIDE BY).

MINUS:
Если отношения r1 и r2 совместимы по объединению, то: r1 MINUS r2 = r1 <AND> <NOT> r2.

A WHERE comp:
Рассмотрим разные виды условий.

- **a comp-op const**. Для сравнения с константой создадим новое вспомогательное отношение **B** с одним атрибутом a (имя должно совпадать с именем атрибута в условии), содержащее все возможные **удовлетворяющие данному условию значения домена**, на котором определен атрибут a. Тогда: A <AND> B = A WHERE (a comp-op const).
- **a = b**. С помощью <REMOVE> уберем из отношения A все атрибуты, кроме атрибута b и переименуем его в атрибут a. Тогда <AND> с исходным отношением даст необходимый результат.
  A <AND> ((...(A <REMOVE> … <REMOVE> ...)…) <RENAME> (b, a)) = A WHERE (a = b).
- **a comp-op b**. Аналогично условию с константой создадим вспомогательное отношение **B** с атрибутами a и b, содержащее все возможные удовлетворяющие данному условию значения доменов, на которых определены атрибуты. Тогда: A <AND> B = A WHERE (a comp-op b). (Вообще говоря, второй пункт списка можно было свести к этому)

JOIN:
Вспомним, что: A JOIN B WHERE comp = (A TIMES B) WHERE comp.
TIMES - частный случай <AND>, WHERE была построена выше.

Таким образом для соединения общего вида необходимо:

- с помощью <RENAME> над одним из отношений избавить от общих имен-атрибутов;
- выполнить для полученных отношений <AND>, соответствующую расширенному декартову произведению;
- ограничить полученное отношение, выполнив одну или несколько <AND> с отношениями-константами (аналогично построению WHERE).

DIVIDE BY:
Пусть имеются отношения r1{a, b} и r2{b}.
Сначала выразим операцию реляционного деления через другие операции алгебры Кодда:
r1 DIVIDE BY r2 = (r1 PROJECT a) MINUS (((r2 TIMES (r1 PROJECT a)) MINUS r1) PROJECT a)

Рассмотрим по порядку:

1. r2 TIMES (r1 PROJECT a) - отношение со схемой {a, b}, в тело которого входят все возможные комбинации значений b из тела r2 и значений a из тела r1;

2. (r2 TIMES (r1 PROJECT a)) **MINUS r1** - то же отношение, но только те кортежи, которые не входят в r1;
3. ((r2 TIMES (r1 PROJECT a)) MINUS r1) **PROJECT a** - оставляем только нужный атрибут. На этом этапе мы получили те значения a, которые не должны попасть в результат;
4. **(r1 PROJECT a) MINUS** (((r2 TIMES (r1 PROJECT a)) MINUS r1) PROJECT a) - уберем из значений атрибута a (присутствующих в теле r1) значения, которые не должны попасть в результат.

Используя уже известные соответствия получим:
r1 DIVIDE BY r2 = (r1 <REMOVE> b) <AND> <NOT> (((r2 <AND> (r1 <REMOVE> b)) <AND> <NOT> r1) <REMOVE> b)

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

Базовые операции реляционной алгебры A:

- Реляционное дополнение <NOT>
- Удаление атрибута r <REMOVE> a
- Переименование r <RENAME> (a, b)
- Реляционная конъюнкция r1 <AND> r2
- Реляционная дизъюнкция r1 <OR> r2

Избыточность алгебры A:
Классический “базис” в булевой алгебре {NOT, AND, OR} избыточен и не является базисом в том смысле, что не удовлетворяет единственности разложения:
A AND B = NOT (NOT A OR NOT B)
A OR B = NOT (NOT A AND NOT B)
Аналогичные утверждения верны и для реляционных операций <NOT>, <AND> и <OR>.
Таким образом, в наборе базовых операций A можно оставить <AND> и <NOT> (или <OR> и <NOT>).

Штрих Шеффера и стрелка Пирса:
Штрих Шеффера: <sh> (r1, r2) = <NOT> r1 <OR> <NOT> r2
Стрелка Пирса: <pi> (r1, r2) = <NOT> r1 <AND> <NOT> r2

Как следствие можно свести набор операций Алгебры A к трем операциям: <sh> (или <pi>), <RENAME> и <REMOVE>.

Избыточность операции переименования:
Рассмотрим операцию A <RENAME> (a, b).
Создадим вспомогательное отношение B с атрибутами a и b, где каждый из кортежей содержит два одинаковых значения, соответствующих значениям атрибута a из отношения A.
Тогда: (A <AND> B) <REMOVE> a = A <RENAME> (a, b).

Таким образом можно сократить набор операций до <sh> (или <pi>) и <REMOVE>.

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

Предположим, что мы работаем с базой данных, которая состоит из отношений:

- `СЛУЖАЩИЕ {СЛУ_НОМ, СЛУ_ИМЯ, СЛУ_ЗАРП, ПРО_НОМ}`
- `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)

При этом мы хотим узнать имена и номера служащих, которые являются руководителями проектов со средней заработной платой, превышающей 18000 руб. Для этого сформулируем запросы с помощью реляционной алгебры и реляционного исчисления.

Запрос, сформулированный при помощи **реляционной алгебры**:

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

Такая формулировка является **процедурной**, т.е. задающей последовательность действий системы для выполнения запроса.

Запрос, сформулированный при помощи **реляционного исчисления**.
Определение переменных:

```
RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ
RANGE ПРОЕКТ IS ПРОЕКТЫ
```

(в зависимости от области определения переменных различают **исчисление кортежей** и **исчисление доменов**; здесь - первое)
Выражение:

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

В формулировке **реляционного исчисления** указаны **лишь характеристики результата**, но ничего не сказано о способе его формирования. В таком случае **система сама должна определить набор и порядок** необходимых **операций** над отношениями.

Оба механизма создания запросов эквивалентны и приводятся друг к другу.

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

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 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: 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.

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 переменные, для каждого фиксированного значения первой рассматриваются возможные значения второй. И так далее для любого количества переменных.

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

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***. 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.

Целевой список (target list) 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.

Отличия:

- В исчислении доменов областью определения переменных являются не отношения, а домены.
- 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 (a_i1 : v_i1, a_i2 : v_i2, ..., a_im : v_im) (m <= n), где v_ij – 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 v_ij – константа, то на атрибут a_ij накладывается жесткое условие, не зависящее от текущих значений доменных переменных; если же v_ij – имя доменной переменной, то условие членства может принимать разные значения при разных значениях этой переменной.

Примеры:

1. 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.
2. 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.

Проблема проектирования: из каких отношений должна состоять БД и какие атрибуты должны у них быть?

Нормализация - приведение отношения к некоторому виду, обладающему хорошими свойствами.

Список нормальных форм:

- Первая нормальная форма (1NF)
- Вторая нормальная форма (2NF)
- Третья нормальная форма (3NF)
- Нормальная форма Бойса-Кодда (BCNF)
- Четвертая нормальныя форма (4NF)
- Пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF)

Общие свойства нормальных форм:

- Каждая следующая НФ “лучше” предыдущей. (обладает большим количеством хороших свойств)
- При переходе к следующей НФ свойства предыдущих сохраняются.

Нормальные формы разделяют на слабо нормализованные - 1НФ и 2НФ - и сильно нормализованные - все остальные.

OTLP (On-line Transaction Processing) - оперативная обработка транзакций. Используются в банковских системах, системах заказов билетов и т.п. Работают с большим числом коротких транзакций простого вида, которые могут выполняться одновременно. Важна скорость и надежность, поэтому лучше подходят сильно нормализованные системы.

OLAP (In-line Analitical Processing) - оперативная аналитическая обработка данных. Используется для хранилищ данных, систем анализа данных. Оперируют с большими массивами данных, данные загружаются нечасто крупными блоками, обычно никогда не удаляются. Скорость работы не так критична. Удобно использовать слабо нормализованные формы.

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

Рассмотрим отношение R с атрибутами (возможно, составными) X и Y.
Будем говорить, что Y функционально зависит от X, если каждому значению X соответствует ровно одно значение Y. Обозначение: FD X->Y от Functional Dependency.

Пример В таблице со списком студентов есть атрибуты СТУД_НОМ, СТУД_ФИО, СТУД_ГР- номер студенческого, ФИО студента и номер группы. По номеру можно однозначно определить ФИО и группу, поэтому есть зависимости

FD СТУД_НОМ -> СТУД_ФИО,
FD СТУД_НОМ -> СТУД_ГР.

Эти FD - инварианты, или ограничения целостности. Это означает, что зависимости порождаются некоторыми знаниями из предметной области, а не случайными совпадениями[4].

FD A -> B называется тривиальной, если B ⊂ A. Любая тривиальная зависимость всегда выполняется.

FD A -> C называется транзитивной, если существует атрибут B такой, что существуют FD A -> B, FD B -> C и отсутствует FD C -> A.

[4] Мы знаем, что в реальной жизни номер студенческого уникален, поэтому зависимости всегда будут выполняться. Если, допустим, случайно оказалось, что в таблице нет одинаковых ФИО, то имеется, например, зависимость FD СТУД_ФИО -> СТУД_ГР, однако она не является инвариантом, поскольку при зачислении нового студента его ФИО может совпасть с чьим-то еще, и зависимость нарушится

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

Пусть задано множество функциональных зависимостей S.
Его замыканием назовем множество функциональных зависимостей S+, содержащее все зависимости, логически выводимые из S.

Аксиомы Армстронга[5] - набор правил логического вывода FD.

I. Если B ⊂ A, то A -> B - рефлексивность.

Верна, потому что зависимость является тривиальной.

II. Если A -> B, то AC -> BC - пополнение.

Пусть это не так. Тогда существуют кортежи t1, t2, проекции которых на A, C совпадают, а проекции на B, C различны. t1(AC) = t2(AC), поэтому из рефлексивности t1(A) = t2(A). Из исходной зависимости A -> B получаем t1(B) = t2(B). Значит, t1(C) ≠ t2(C), что противоречит тривиальной зависимости AC -> C.

III. Если A -> B и B -> C, то A -> C - транзитивность.

Если это не так, то существуют кортежи t1, t2: t1(A) = t2(A), t1(C) ≠ t2(C). Тогда из первой зависимости имеем t1(B) = t2(B). Из второй - t1(C) = t2(C) - противоречие.

Система Армстронга полна и совершенна (все, что выводится, можно вывести с ее помощью), однако Дейт для удобства предложил еще 5 правил.

1. A -> A - самодетерминированность. *Следует из I.*
2. Если A -> BC, то A -> B, A -> C - декомпозиция.

Из I имеем BC -> B, из транзитивности получаем A -> B. Вторая FD аналогично.

3. Если A -> B и A -> C, то A -> BC - объединение.

Из II следует A -> AB и AB -> BC. Из III выводим A -> BC.

4. Если A -> B и C -> D, то AC -> BD - композиция.

Из II следует AC -> BC и BC -> BD, по III получаем AC -> BD.

5. Если A -> BC и B -> D, то A -> BCD - накопление.

Из II следует BC -> BCD, по III получаем A -> BCD.

[5] Формально не являются аксиомами, потому что выводятся из определения FD.

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

Пусть заданы отношения R, множество атрибутов Z этого отношения и множество FD S, выполняемых на R.
Замыканием[6] Z над S называется наибольшее множество Z+ атрибутов отношения R таких, что FD Z -> Y входит в S+.

Алгоритм построения Z+.

1. Берем в качестве Z[0] множество Z.
2. На (k+1)-ом шаге добавляем к Z[k] такие атрибуты B, которые можно вывести из Z[k] используя S (существует A из Z[k] и зависимость A -> B из S).
3. Процесс останавливается, когда Z[k] = Z[k + 1].

Пример. Пусть 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.

1. Z[0] = {AE}
2. Z[1] = {ACDE}
3. Z[2] = {ACDEF}
4. Z[3] = {ACDEF} = {AE}+

Польза. Алгоритм позволяет установить, входит ли некоторая FD Z -> B в замыкание S+. Необходимым и достаточным условием этого является вхождение B в Z+.

Суперключ отношения R - подмножество заголовка, включающее в себя хотя бы один ключ R.

Множество является суперключом ⇔ из него выводимы все атрибуты отношения ⇔ его замыкание совпадает с заголовком отношения.

[6] Говоря проще, это все атрибуты, которые можно вычислить, зная Z и имея зависимости S.

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 из S2. Это эквивалентно $S1+ \subset S2+$.

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:

1. Правая часть любой FD из S является множеством из одного атрибута (простым атрибутом);
2. Левая часть каждой FD из S обладает свойством минимальности, то есть удаление из нее любого атрибута приводит к изменению замыкания S+.
3. Удаление любой FD из S приводит к изменению 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 эквивалентное ему минимальное множество S-.

Алгоритм построения S-:

1. Декомпозиция: разбиваем сложные зависимости A-> BC на простые A->B и A->C .
2. Пробуем перебором удалять атрибуты из левой части так, чтобы замыкание множества не изменялось.
3. Пробуем перебором удалять FD так, чтобы замыкание сохранялось.

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.

Будем считать корректными декомпозиции отношений, которые являются обратимыми, то есть можно собрать исходное отношение из декомпозированных без потери информации. Такие декомпозиции называются декомпозициями без потерь.

Теорема Хита. Пусть задано отношение R с заголовком {A, B, C} (атрибуты, вообще говоря, составные) и выполнено FD A -> B. Тогда

R = (R PROJECT {A, B}) NATURAL JOIN (R PROJECT {A, C}).

Обозначим R' = (R PROJECT {A, B}) NATURAL JOIN (R PROJECT {A, C}).
Покажем, что R ⊂ R'. Пусть кортеж {a, b, c} лежит в R. Тогда {a, b} лежит в проекции R PROJECT {A, B}, {a, c} лежит в проекции R PROJECT{A, C}, поэтому {a, b, c} лежит в R'.
Покажем, что R' ⊂ R. Пусть {a, b, c} лежит в R'. Тогда в R существуют столбцы {a, b, c} и {a, b*, c}. В силу зависимости A -> B имеем b = b*, а значит, столбец {a, b, c} принадлежит R.*

Атрибут B минимально зависит от атрибута A, если выполняется минимальная слева FD A -> B (ни один атрибут в A не является лишним).

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

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

Аномалии обновления - трудности, возникающие при добавлении, удалении или модификации кортежей[7].

Аномалии из-за неминимальных FD.
Рассмотрим отношение СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ, содержащее атрибуты СЛУ_НОМ (номер служащего), СЛУ_УРОВ (его уровень), СЛУ_ЗАРП (запрлата), ПРО_НОМ (номер проекта, в котором он работает), СЛУ_ЗАДАН (номер задания, которое выполняет в проекте).

При этом имеются зависимости СЛУ_НОМ -> СЛУ_УРОВ, СЛУ_УРОВ -> СЛУ_ЗАРП, {СЛУ_НОМ, ПРО_НОМ} -> СЛУ_ЗАДАН. Таким образом, ключом является {СЛУ_НОМ, ПРО_НОМ}, однако зависимость уровня и зарплаты от этого ключа неминимальна (достаточно знать СЛУ_НОМ).

Из-за этого возникают следующие аномалии.

1. Нельзя добавить служащего, не включив его в один из проектов (требование 1NF: первичный ключ не может содержать неопределенных значений)
2. Если служащий закончил работу над проектом, но не успел начать новый, то сохранить его кортеж также не удастся.
3. Для модификации разряда служащего требуется изменить все кортежи с соответствующим СЛУ_НОМ (которых может быть много).

Пример декомпозиции.
Можно декомпозировать отношение на два: {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП} и {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}. По теореме Хита данная декомпозиция является декомпозицией без потерь. Она решает описанные выше проблемы. При этом все зависимости в обоих отношениях минимальны.

Отношение находится во второй нормальной форме (2NF), если оно находится в 1NF, и каждый неключевой атрибут минимально зависит от первичного ключа.

[7] Обычно связаны с тем, что для выполнения одной операции требуется выполнить ее большое количество раз.

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

Зависимость A->C называется транзитивной, если существует атрибут B такой, что существуют FD A -> B, FD B -> C и отсутствует FD C -> A.

Аномалии из-за транзитивных FD.
Рассмотрим пример отношения, полученного в результате декомпозиции в прошлом вопросе: заголовок {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП} и зависимости СЛУ_НОМ -> СЛУ_УРОВ, СЛУ_УРОВ -> СЛУ_ЗАРП. Получается, что зарплата транзитивно зависит от номера служащего.

Возникают аномалии:

1. Добавление. Невозможно сохранить данные о новом уровне, пока не появится служащий с таким уровнем.
2. Удаление. При увольнении последнего служащего с данным уровнем информации об уровне утрачивается.
3. Модификация. При изменении размера зарплаты, соответствующей уровню, нужно менять значение СЛУ_ЗАРП для всех служащих этого уровня.

Пример декомпозиции.
Можно разбить на два отношения: {СЛУ_НОМ, СЛУ_УРОВ} и {СЛУ_УРОВ, СЛУ_ЗАРП}. Проблемы решаются, а транзитивные зависимости исчезают.

Отношение находится в третьей нормальной форме (3NF), если она находится в 2NF, и каждый неключевой атрибут нетранзитивно функционально зависит от первичного ключа.

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

Независимые проекции отношения - при декомпозиции на эти проекции возможно независимое обновление каждой из них, то есть естественное объединение этих проекций после обновления эквивалентно обновлению целого отношения[8].

Теорема Риссанена. Проекции R1 и R2 отношения R являются независимыми ⇔ выполнены

- Каждая FD в отношении R логически следует из FD в R1 и R2[9].
- Общие атрибуты R1 и R2 образуют возможный ключ хотя бы для одного из отношений[10].

Атомарное отношение - отношение, которое невозможно декомпозировть на атомарные проекции.

[8] В прошлом вопросе декомпозиция на {СЛУ_НОМ, СЛУ_УРОВ} и {СЛУ_УРОВ, СЛУ_ЗАРП} является независимой. Декомпозиция {СЛУ_НОМ, СЛУ_УРОВ}, {СЛУ_НОМ, СЛУ_ЗАРП} также в 3NF, однако не является независимой.

[9] В примере в прошлой сноске это не так: зависимость СЛУ_УРОВ -> СЛУ_ЗАРП не следует из зависимостей декомпозированных отношений. Такое ограничение называется ограничением базы данных.

[10] По сути требоване для корректной декомпозиции по теореме Хита.

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

Рассмотрим пример отношения с заголовком {СЛУ_НОМ, СЛУ_ИМЯ, ПРО_НОМ, СЛУ_ЗАДАН} и зависимостями {СЛУ_НОМ, ПРО_НОМ} -> СЛУ_ЗАДАН, {СЛУ_ИМЯ, ПРО_НОМ} -> СЛУ_ЗАДАН, СЛУ_НОМ -> СЛУ_ИМЯ, СЛУ_ИМЯ -> СЛУ_НОМ.
Имеем два пересекающихся ключа {СЛУ_НОМ, ПРО_НОМ} и {СЛУ_ИМЯ, ПРО_НОМ}.

Аномалии обновления.
При изменении имени служащего, требуется изменить имя во всех кортежах, соответствующих его номеру.

Решает проблему, например, декомпозиция {СЛУ_НОМ, СЛУ_ИМЯ} и {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}.

Отношение находится в нормальной форме Бойса-Кодда (BCNF), если любая нетривиальная и минимальная FD в ней имеет в качестве детерминанта (левой части) некоторый возможный ключ данного отношения.

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

Многозначная зависимость ставит в соответствие одному значению атрибута-аргумета множество значений второго атрибута. Более того, требуется, чтобы это множество не зависело от остальных атрибутов. Более точно, в отношении R с атрибутами A, B, C (вообще говоря, составными) имеется многозначная зависимость A ->-> B, если множество значений атрибута B, соответствующее паре значений атрибутов A и C, зависит от значения A и не зависит от C[11].

Многозначные зависимости обладают свойством двойственности - оказывается, что если A ->-> B, то A ->-> C.

Лемма Фейджина. Если в отношении R{A, B, C} выполняется A ->-> B ⇔ A ->-> C (обозначается A ->-> B | C).

Докажем, что из A ->-> B следует A ->-> C. (обратное аналогично)
Покажем, что множество значений {c} при фиксированном a не зависит от b из {b} (здесь {b}, {c} обозначают все значения, которые принимали B и C при заданном a). Предположим, что это не так. Тогда существуют c из {c}, b', b'' из {b} такие, что {a, b', c} принадлежит телу отношения, однако {a, b'', c} не принадлежит ему. Последнее противоречит A ->-> B (b'' принадлежит {b}, то есть некоторый кортеж {a, b'', c} принадлежит телу отношения, и, значит, значения b зависят от значений c).*

Теорема Фейджина. Пусть имеется отношение R с атрибутами (в общем случае, составными). R декомпозируется без потерь на R1 = {A, B} и R2 = {A, C} ⇔ для него выполняется A ->-> B | C.

Докажем необходимость декомпозируемости.
Пусть {a, b, c'} и {a, b', c} принадлежат R. Тогда {a, b} принадлежит R1, {a, c} принадлежит R2. Значит, {a, b, c} принадлежит R1 NATURAL JOIN R2, и следовательно, принадлежит R. Получили A ->-> B | C.
Достаточность.
R всегда вложено в R1 NATURAL JOIN R2, поэтому докажем обратное вложение.
Пусть {a, b, c} принадлежит R1 NATURAL JOIN R2. Тогда в отношении R есть строки {a, b, c'} и {a, b', c}, откуда с учетом многозначной зависимости {a, b, c} также лежит в R.

[11] Например, СЛУ_НОМ ->-> ПРО_НОМ ставит в соответствие номеру сотрудника номера проектов, в которых он участвует, и эти номера не зависят от его зарплаты.

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

В отношении R с атрибутами A, B, C (вообще говоря, составными) имеется многозначная зависимость A ->-> B, если множество значений атрибута B, соответствующее паре значений атрибутов A и C, зависит от значения A и не зависит от C.

Рассмотрим отношение с заголовком {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}. Пусть каждый служащий выполняет одни и те же задания независимо от проекта, то есть имеется многозначная зависимость СЛУ_НОМ ->-> ПРО_НОМ | СЛУ_ЗАДАН.

Возникающие из-за многозначных зависимостей аномалии.

1. При добавлении проекта к служащему нужно добавить столько кортежей, сколько заданий он выполняет.
2. Если служащий не участвует ни в одном проекте, невозможно сохранить данные о заданиях, которые он выполняет.
3. При изменении одного из заданий служащего необходимо изменить столько кортежей, во скольких проектах участвует этот служащий.

Пример декомпозиции.
Декомпозиция на {СЛУ_НОМ, ПРО_НОМ} и {СЛУ_НОМ, СЛУ_ЗАДАН} решает перечисленные проблемы, а многозначные зависимости превращаются в однозначные FD.

Отношение находится в четвертой нормальной форме (4NF), если оно находится в BCNF и все многозначные зависимости являются FD, детерминанты которых - возможные ключи отношения.

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

n-декомпозируемое отношение - можно декомпозировать без потерь на n частей.

Пример декомпозиции.
Рассмотрим отношение с заголовком {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}, единственным возможным ключом которого является сам заголовок. Опишем словами зависимость проекции/соединения: если служащий с номером СЛУЖ участвует про проекте ПР, в проекте ПР есть задание ЗАД, и служащий СЛУЖ выполняет задание ЗАД, то служащий СЛУЖ выполняет в проекте ПР задание ЗАД. Этого условия достаточно для декомпозиции отношения без потерь на {СЛУ_НОМ, ПРО_НОМ}, {СЛУ_НОМ, СЛУ_ЗАДАН}, {ПРО_НОМ, СЛУ_ЗАДАН}.

Формально, в отношении R с составными перекрывающимися атрибутами A, B, … , Z удовлетворяется зависимость проекции/соединения (PJD), если R можно получить путем естественного соединения проекций этого отношения на атрибуты A, B, … , Z.

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

Рассмотрим отношение с заголовком {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН} из прошлого вопроса, в котором имеется зависимость проекции/соединения с числовыми значениями.

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	2	B
2934	1	A

Аномалии из-за зависимости проекции/соединения.

1. Пусть мы хотим добавить кортеж {2941, 1, A}. Тогда для сохранения PJD необходимо добавить также кортеж {2934, 2, A}[12].

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

2. Если мы захотим теперь удалить кортеж {2934, 1, A}, то для сохранения PJD требуется удалить также кортеж {2941, 1, A}.

Пример декомпозиции.
3-декомпозиция на пары {СЛУ_НОМ, ПРО_НОМ}, {ПРО_НОМ, ПРО_ЗАДАН}, {ПРО_ЗАДАН, СЛУ_НОМ} решает описанные проблемы.

В отношении R зависимость проекции/соединения *(A, B, ..., Z) называется подразумеваемой возможными ключами, если 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), если каждая нетривиальная PJD в R подразумевается возможными ключами R[13].

Нормализация после 5NF не имеет смысла, аномалии обновлений нельзя будет устранить декомпозицией.

---

[12] При добавлении кортежа мы узнали, что в проекте 1 есть задание A. Поскольку служащий 2934 участвует в проекте 1 и умеет делать задание A, он также должен делать задание A в проекте 1.
[13] Это гарантирует, что любой кортеж будет уникален в каждом составном атрибуте A, B, …, Z, и потому не потребуется производить лишних операций из-за PJD.

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

Существует 2 основных подхода к хранению отношений:

1. Покортежное хранение
   Единица хранения информации - кортеж.
   + Обеспечивает быстрый доступ к целому кортежу
   - Хранится много дублирующейся информации
   - Лишние обмены с внешней памятью, если нужна только часть кортежа
2. Хранение отношения по столбцам
   Единица хранения информации - столбец с исключёнными дубликатами.
   + Минимизированы затраты внешней памяти (за счёт исключения дубликатов)
   + Возможность использования значений столбца для оптимизации операции соединения
   - Сборка целого кортежа(или его части) требует существенных затрат

Построчное (покортежное) хранение отношений.
Внешняя память представляется как набор сегментов, каждый из которых состоит из страниц. Каждому кортежу сопоставляется уникальный идентификатор **tid** (tuple identifier), который не изменяется всё время существования кортежа.
tid - пара вида <номер страницы, номер описателя кортежа в странице>.

Перемещение кортежа не изменяет его исходный tid. При переразмещении изменяется только описатель в исходной странице, который теперь будет содержать новый tid, указывающий на реальное положение кортежа в новой странице. Дальнейшее переразмещение кортежа не увеличит уровень переадресации(не возникнет ситуации, когда tid указывает на описатель, в котором tid, который указывает на ещё один описатель, в котором очередной tid указывает на очередной описатель и так 10 раз). Поиск кортежа описывается цепочкой вида:
исходный tid=<N,i> → страница номер N, в ней запись номер i, в которой содержится описатель содержащий tid=<M,j> → страница номер M, в ней запись номер j, в которой содержится нужный кортеж.

Проблема “длинных” данных:(мб в этом билете и не нужно)

- Хранение длинных данных в отдельных файлах(вне БД). Соответствующие данные в кортеже заменяются на имя файла.
- Хранение данных в отдельном наборе страниц внешней памяти, связанном физическими ссылками
- Хранение длинных данных в виде B-деревьев последовательностей байтов в отдельном наборе страниц внешней памяти.

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

Индекс - специальная конструкция в СУБД, обеспечивающая механизмы эффективного прямого доступа к произвольному кортежу отношения по значению некоторого ключа. Она же обеспечивает механизм последовательного просмотра кортежей в порядке возрастания или убывания значения ключа.
Суть: давайте хранить упорядоченный набор значений ключа. И каждому этому значению будет соответствовать некоторый список идентификаторов кортежа.
Вопрос: как искать ключ?

Техника хранения на основе B-деревьев.
B-дерево - сильно ветвистое сбалансированное дерево во внешней памяти.
Сбалансированное значит, что длина пути от корня до любого листа примерно одинакова.
Сильно ветвистое значит, что у каждого узла дерева много узлов-потомков.

Физическая реализация B-дерева - древовидная мульти-списочная структура внутренних и листовых страниц внешней памяти.
Листовая страница имеет следующую структуру:
(<key_i, TIDs_i>...),
где

- key_1 < key_2 < … < key_n
- TID_s_i = {tid_{i1}, tid_{ij},...,tid_{im}} - множество всех идентификаторов соответствующих ключу key_i
- Все листовые страницы связаны одно- или двунаправленным списком

Внутренняя страница имеет следующую структуру:
(<key_i,N_i>...)
где

- $key\_1 \leq key\_2 \leq ... \leq key\_n$
- N_i - ссылка на внутреннюю или листовую страницу, где содержатся ключи со значениями из интервала [key_i,...,key_{i+1}]

Поиск в таком дереве осуществляется по значению ключа, и в силу ветвистости и сбалансированности требует малого количества обменов с внешней памятью.
В сбалансированном дереве с размером внутренней страницы в n ключей для хранения m записей потребуется глубина $log_n m$.

Автоматическое поддержание свойства сбалансированности B-дерева:
При занесении новой записи выполняется:

- Поиск листовой страницы
- Считывание страницы в буфер(который больше страницы) и её модификация(добавление нового значения)
- Если при этом размер используемой части буфера не превышает размер страницы - всё хорошо и мы можем сразу вытолкнуть буфер во внешнюю память
- Если происходит переполнение буфера - выполняется расщепление страницы. Запрашивается новая страница внешней памяти и содержимое буфера делится пополам между исходной страницей и вновь созданной
- В связи с созданием новой листовой страницы необходимо модифицировать внутреннюю страницу, которая должна ссылаться на новую. В связи с этой модификацией может снова возникнуть переполнение буфера(в соответствующей внутренней странице уже достигнуто

максимально допустимое число ссылок на листья). В таком случае создаётся новая внутренняя страница и действия с обновлениями записей повторяются на уровне выше
- В самом худшем случае эта цепочка переполнений дойдёт до корневой страницы и придётся расщепить её на две чтобы создать новую корневую страницу, которая будет ссылаться как раз на эти 2 страницы. В данном случае происходит увеличение глубины дерева на 1

При удалении записи выполняются следующие действия:

- Поиск записи по ключу
- Считывание листовой страницы в буфер обмена и реальное удаление записи
- Если после удаления размер области, занятой в буфере остатками страницы оказывается таков что: его сумма с размером занятой области в соседних(с точки зрения дерева, предыдущей и последующей) листовых страницах больше чем размер страницы, тогда операция завершается
- Иначе страница сливается с предыдущей или последующей(т.е. одной из соседних). Ненужная страница при этом помечается как свободная, а сквозные ссылки между листовыми страницами модифицируются.
- Теперь необходимо схожим образом модифицировать внутреннюю страницу-предка удалённой(т.к. из страницы-предка удаляется запись). Это происходит аналогично предыдущему алгоритму для листовой вершины и может снова возникнуть необходимость слияния на уровне выше.
- В предельном случае сливаются два последних потомка корневой страницы и она освобождается. В этом случае происходит уменьшение глубины дерева на 1.

Для повышения эффективности также применяют более сложные приёмы:

- Упреждающее расщепление(чуть раньше чем при полном переполнении страницы)
- Переливания(для равновесного заполнения всех страниц)
- Слияния 3 в 2(порождение 2 листовых страниц на основе 3 соседних)

Хеширование

Общая идея - придумать некую функцию, которая будет сопоставлять значению ключа некое меньшее число (свёртку ключа). Свёртка значения ключа потом используется для доступа к записи. $i = f(X)$ где X - ключ, вообще говоря, произвольный, f - хеш-функция, i - обычно int индекс в некотором фиксированном интервале (свёртка ключа X).

Классический случай – свертка ключа используется как индекс в массиве, содержащем пары ключей и tid (их много из-за коллизий), называемых “цепочки переполнения”.

Коллизия - ситуация, в которой хеш-функция сопоставляет двум разным ключам одинаковую свёртку.

Основным требованием к хэш-функции является равномерное распределение значений свертки.

Главное преимущество хеширования - доступ к данным почти всегда за одно обращение. Если таблица заполнена слишком сильно или переполнена это преимущество теряется.

Отсюда и главный недостаток метода - ограниченность таблицы. Чтобы сделать её больше заранее определённого лимита - необходимо менять хеш-функцию.

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

Виды проектирования баз данных:

Концептуальное проектирование - процесс создания БД, не зависящий от любых физических аспектов ее представления (просто хотим построить БД с котиками и выделили их характеристики).
Логическое проектирование - процесс создания модели информации с учетом выбранной модели организации данных, но независимо от типа целевой СУБД и других физических аспектов реализации (пример - приведение к нормальной форме, придумали, как загнать данные про котиков, например, в таблицу)
Физическое проектирование - процесс описания реализации БД на ЗУ (запоминающих устройств) с указанием структур хранения и методов доступа, используемых для эффективной обработки (написала БД с прописанными доступами к памяти, где лежит информация про котиков).

Недостатки проектирования в терминах отношений:

- неудобство для проектировщиков (на первых этапах нужны специалисты из предметной области; во многих предметных областях трудно осуществить моделирование информации на основе плоских таблиц);
- отсутствие наглядности (недостаточно средств для представления смысла данных (все мы не очень любим анализировать таблицы, правда?), нет механизма для разделения объектов предметной области и связи между ними (связи также организуются в виде какого-либо атрибута));
- невозможность автоматизации процесса проектирования (не может сама построить функциональные связи, все зависимости выделяются вручную)

Информационная модель - способ представления понятий или объектов предметной области, описывающий существенные для данного представления совокупности объектов, их параметры, поведение и отношения между ними.

Достоинства информационного моделирования:

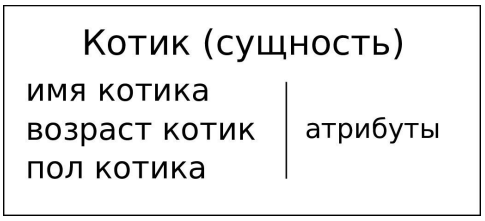
- построение наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на ранних стадиях проектирования (например представили все в виде диаграмм и стрелочек таким образом, что все выглядит понятным);
- информационная модель является важной документацией для работы с БД;
- существуют CASE-системы (CASE - Computer-Aided Software Engineering), преобразующие концептуальные схемы (диаграммы) в реляционные (на SQL).

Средства автоматизации: CASE-система

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

ER (Entity-Relationship) - информационная модель, позволяющая строить концептуальные схемы БД

Основные понятия
Сущность (тип сущности) - объект, информация о котором должна сохраняться и быть доступной (например, котик).
Атрибут сущности - любая деталь, которая служит для уточнения, идентификации, числовой характеристики или выражение состояния сущности (цвет, возраст, состояние котика).
Связь - графически изображаемая ассоциация (ненаправленная линия), устанавливаемая между двумя сущностями, может быть установлена либо между двумя сущностями, либо сама с собой.
Роль - конец связи



Характеристики роли:
- имя роли (имеет, принадлежит)
- степень роли - сколько сущностей может быть привязано по этой связи (допустимое количество мячиков - много или один)
- обязательность роли (обязательно выполнено или не обязательно)

Представление на диаграммах - вставленные рисунки.

Пример:
У котика может быть много мячиков (пример связи один ко многим (для связи 1 и 1 надо убрать вилку - котик несчастный и у него один мячик)), как известно, каждый **котик** (один - одноточечный вход) обязательно должен **иметь** (имя роли) мячик (пример обязательной роли), но **мячики** (пример трехточечного входа - вилка - мячиков может быть много) не обязательно **принадлежит** (имя роли) котику (пример необязательной роли).

Уникальные идентификаторы типов сущностей - характеристики сущности (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).

Получение реляционной схемы из ER-диаграммы. Переход от информационной модели к локальной.

Алгоритм:

1) каждый тип сущности превращается в отношение (таблицу), имя сущности - имя таблицы, каждый экземпляр сущности - кортеж (строка);
2) каждый атрибут превращается в столбец таблицы (атрибут отношения);
3) компоненты индивидуального идентификатора сущности - первичный ключ отношения (позволяет однозначно определить нужный кортеж), если в индивидуальный идентификатор сущности входит связь, то в качестве дополнительного атрибута добавляется копия идентификатора сущности, находящейся на противоположном конце связи (если у котика есть хозяин (связь) и она помогает однозначно определить конкретного котика, то в отношение (таблицу) добавляется атрибут (столбец), содержащий информацию о хозяине, достаточную для его однозначного определения (например номер паспорта));
4) связи многие к одному становятся внешними ключами (если у одного хозяина есть много котиков, то в таблице котиков будет столбец “хозяин”, содержащий данные о хозяине каждого котика), необязательные связи соответствуют столбцам, допускающим неопределенные значения, в случае связи один к одному для добавления атрибута (столбца) можно выбрать любое отношение (таблицу);
5) при наличии связи многие ко многим создается отдельное отношение (таблица) с двумя атрибутами, содержащими ключи для каждого из связанных отношений (если у многих котиков много разных хозяев, то в таблице будут содержаться все пары котиков, соответствующие схеме питомец - хозяин).

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

Тип сущности может быть расщеплен на несколько подтипов, каждый из которых включает общие атрибуты и/или связи.
Тип сущности, на основе которого определяются подтипы, называется супертипом.
Пример: тип - котик, подтипы - кот, кошка, котенок.
ER - модель не ограничивает подтипизацию (мб сколько угодно уровней).

Наследование: на основе некоторого супертипа выделяются подтипы с некоторыми добавленными свойствами.

Особенности механизма наследования (A - супертип, B - подтип):

1) включение - для любого b из B_i следует, что b лежит в A;
2) отсутствие собственных экземпляров у супертипа - для любого a из A следует, что a лежит в B_i;
3) разъединенность подтипов - если b лежит в B_i, то b не лежит в B_j.

Пример:
Среди котиков (супертип) могут выделяться подтипы коты (доп. свойство - кастрированный или нет), кошки (доп. свойства - беременная или нет), котята (доп. свойство - слепой или нет).

На диаграмме подтипы отображаются как атрибуты-сущности (квадратик внутри квадратика под общими атрибутами для супертипа).

Отображение в реляционную схему
Рассмотрим супертип A и k его подтипов B_1, ... , B_k. У типа A есть m атрибутов a_1, …, a_m. У каждого i-го подтипа есть свои n(i) атрибутов $b_{i,1}$, ... , $b_{i,n(i)}$. Существует два способа представления наследования в виде реляционных схем. X - некоторое значение атрибута (определенное).

1 способ - общая таблица всех подтипов
A

T	a_1	...	a_m	$b_{1,1}$	...	$b_{1,n(i)}$	...	$b_{k,1}$	...	$b_{k,n(k)}$
'B_1'	X	...	X	X	...	X		NULL	...	NULL
...	...	...	...	...	...	...	...	...	...	...
'B_k'	X	...	X	NULL	...	NULL	...	X	...	X

Чтобы извлечь данные подтипа: PROJECT A{a_1, …, a_m, $b_{i,1}$, …, $b_{i,n(i)}$} WHERE t = 'B_i'.

2 способ - отдельная таблица для каждого подтипа

Для каждого подтипа B_i составляется таблица следующего вида:
B_i

a_1	...	a_m	$b_{i,1}$	...	$b_{i,n(i)}$
X	...	X	X	...	X

Чтобы извлечь данные супертипа: PROJECT B_1{a_1, …, a_m} UNION … UNION PROJECT B_n{a_1, …, a_m}.

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

Взаимно исключающие связи
Для заданной сущности может быть задан такой набор связей с другой сущностью, что для каждого экземпляра сущности может (или должна) существовать только одна связь и данного набора.

Пример:
Пусть котики бывают домашние и бездомные, тогда у домашних котиков будет связь с хозяином, а у бездомных котиков будет связь с приютом, в котором они содержатся. Легко заметить, что котики не могут жить и дома и в приюте, но пока мы говорим о сущности “котик”, а не о конкретном экземпляре сущности, связь будет построена и с сущностью приют, и с сущностью хозяин.

Отображение диаграммы в реляционную схему

1 подход - обобщение: рассматривать набор взаимоисключающих сущностей как подтипы некоторого супертипа, тогда у сущности будет связь с одним супертипом (объединить подтипы “хозяин” и “приют” в супертип “владелец”);

2 подход - специализация: воспринимать сущность, для которой определены взаимно исключающие связи, как супертип и выделить в нем подтипы, тогда у каждого подтипа будет соотвующая связь (разделить супертип “котики” на подтипы “домашние котики” и “бездомные котики”).

Таким образом, в диаграмме не будет взаимно исключающих связей, но будет наследование, которое может быть реализовано путем создания общей таблицы для подтипов или таблиц для каждого из подтипов (см. вопрос 37).

Если имеется взаимно исключающая множественная связь типа один ко многим, причем многие относятся к сущности (котикам), то есть два способа преобразования ER-диаграмм в реляционные схемы.

1 способ - общее хранение внешних ключей: если взаимно исключающие связи задаются в одном формате (например оба - строки), то в отношение (таблицу) может быть добавлено два атрибута (столбца) - идентификатор связи (приют или хозяин) и идентификатор сущности (название приюта или имя хозяина).

2 способ - раздельное хранение внешних ключей: если взаимно исключающие связи задаются в разных формата (название приюта и номер паспорта хозяина), то в отношение (таблицу) добавляется число атрибутов (столбцов), соответствующее числу взаимно исключающих связей, каждый из которых содержит либо идентификатор сущности, либо не определен (если котик живет у хозяина, то в столбце “хозяин” будет указан номер паспорта хозяина, а в столбце “приют” будет NULL).

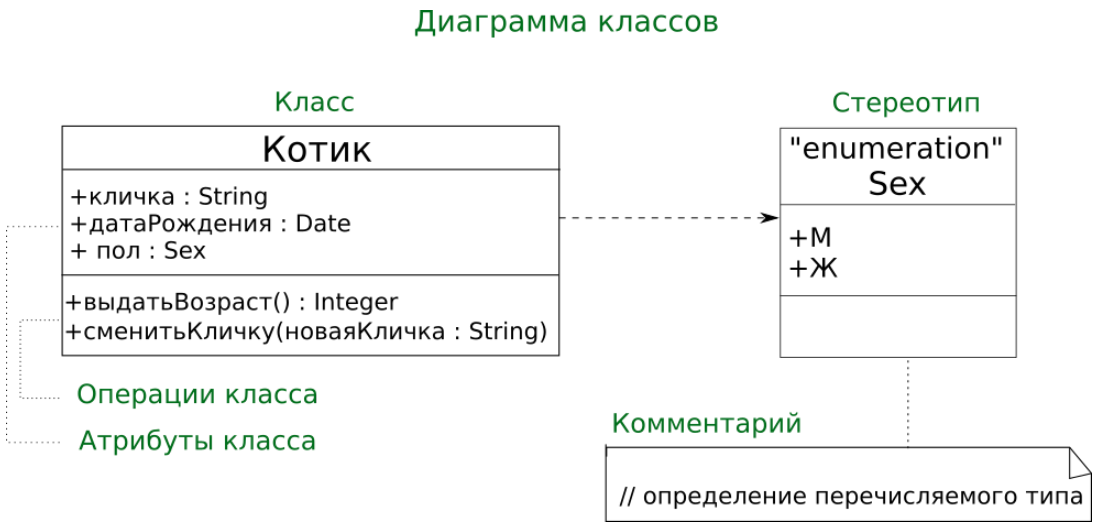
38. Диаграммы классов языка UML. Основные понятия. Отображение классов, стереотипов, комментариев и ограничений на диаграммах. Примеры.

UML (Unified Modeling Language) создан для моделирования разных систем (аппаратных, программных, смешанных, с участием человека и др.), описание их статических (структурных) и динамических (поведенческих) свойств.

Основные понятия

Диаграмма классов - типы объектов моделируемой системы и статические отношения различного рода, которые существуют между ними, может включать комментарии и ограничения (ограничения могут быть записаны и на естественном языке, и на языке OCL).
Класс - именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой (котик)
Атрибут класса - именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства (абстракция состояния объекта), задается именем и типом (цвет котика - String, возраст котика - Integer).
Операция класса - именованная услуга, которую можно запросить у любого объекта этого класса (абстракция поведения объекта), задается именем и сигнатурой (узнать, сколько лет котику), есть свойство операции query - запрос (операции с таким свойством не могут менять состояние экземпляров данного класса).
Стереотип - механизм расширения семантики UML, позволяющий создавать новые элементы UML на основе существующих с учетом особенностей решаемой задачи.

Диаграмма + пример:



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

UML (Unified Modeling Language) создан для моделирования разных систем (аппаратных, программных, смешанных, с участием человека и др.), описание их статических (структурных) и динамических (поведенческих) свойств.

Категории связей

Три типа связей:

- зависимость;
- обобщение;
- ассоциация.

Зависимость - связь по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый.
Пример: использование класса “Sex” для определения “котика”.

Обобщение - связь между общим классом (суперклассом) и более специализированной его разновидностью (подклассом) (правила наследования из ER-модели: первый выполняется по умолчанию, остальные - нет).
Пример: выделение из “котика” “кота”, “котенка” и “кошки” (см. вопрос 36 - наследование в ER-моделях)

Ассоциации - структурная связь между объектами одного класса и объектами другого или того же самого класса. Допускается создание n-арных ассоциаций.
Ассоциации может быть присвоено имя, характеризующее связь.
Пример: связь “котик - хозяин”, имя ассоциации “владение”.

Агрегатные ассоциации отображают отношение “часть - целое”.
Пример: связь “котик” - “хвост”.

Роль класса - имя, характеризующее класс в связи.
Пример: связь “котик - хозяин”, здесь роль класса “котик” - “питомец”

Кратность роли - характеристика, указывающая, сколько объектов класса с данной ролью может участвовать в каждом экземпляре ассоциации (один или много котиков).

На диаграмме

- зависимость - пунктирная линия со стрелкой;
- обобщение - сплошная линия с треугольной контурной стрелкой;
- ассоциация - сплошная линия (в общем виде, ненаправленная);
- агрегатная ассоциация - сплошная линия с контуром ромба на конце “целого” (закрашенная, если при уничтожении целого уничтожается и часть).

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

OCL (Object Constraint Language) - часть спецификации UML, нужен для определения ограничений, описывающих пред- и постусловия операций классов, а также ограничений, являющихся инвариантами классов.

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

Синтаксис определения инварианта (для некоторого класса):
context <class_name> inv:
<OCL logical expression>

Основные типы (предопределенные типы данных OCL):

- скалярные:
  - Integer;
  - Real;
  - Boolean;
  - String;
- коллекции:
  - set (неупорядоченная коллекция, не содержащая одинаковых элементов);
  - orderedSet (упорядоченный set);
  - bag (неупорядоченная коллекция, которая может содержать одинаковые элементы);
  - sequence (упорядоченный bag).

Выражения OCL состоят из:

- предопределенных типов данных OCL;
- типов данных, определяемых моделью UML (классы);
- атрибутов классов, их ролей и ассоциаций;
- операций классов, не изменяющих состояния моделируемой системы.

Примечание: далее можно написать, что выражения строятся из операций над представленными элементами и ничего больше не писать (иначе получается слишком много и бесчеловечно, на мой взгляд).

Операции над скалярными типами:

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

Операции над объектными типами (классами UML):

- получение значения атрибута <объект>.<имя атрибута>;
- переход по экземпляру к ассоциации <объект>.<имя роли, противоположной по отношению к объекту>;
- вызов операции класса <объект>.<имя операции>(<список фактических параметров>);
- self() - текущий объект класса, в котором определен инвариант.

Операции над коллекциями OCL:
<коллекция> -> <имя операции>

- конструкторы коллекций
  - select() - новая коллекция из тех, для которых условие выполнено (true);

    - reject() - новая коллекция из тех, для которых условие не выполнено (false);
    - collect() - результат применения выражения к каждому элементу коллекции (в скобках не логическое выражение);
- кванторы
    - exists() - логическое выражение выполняется хотя бы для одного;
    - forAll() - логическое выражение выполняется для всех
- теоретико-множественные операции
    - union - объединение;
    - intersect - пересечение;
    - ‘-’ - вычитание;
- прочие операции
    - count() - подсчитывает число вхождений элементов в коллекцию;
    - includes() - проверяет наличие данного элемента в коллекции;
    - excludes() - проверяет отсутствие данного элемента в коллекции;
    - size() - размер коллекции;
    - at() - применима только к упорядоченным коллекциям и вернем элемент на позиции индекса.

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

Алгоритм - аналогично ER-модели, см. вопрос 35.

Основные проблемы и рекомендации:

1) операции в классах могут быть реализованы не во всех РСУБД;
2) связи-зависимости при преобразованиях в РСУБД не используются;
3) старайтесь избегать множественного наследования и осторожно используйте одиночное наследование классов;
4) эффективно в РСУБД реализуются только ассоциации “один ко многим”, ассоциации с заданной кратностью роли могут быть реализованы, но не эффективно;
5) для РСУБД агрегатные ассоциации неестественны, композиции влияют, как правило, только на способ поддержки ссылочной целостности (каскадное удаление);
6) определение большого числа ограничений может привести к потере эффективности.

Обобщение: большинство усложнений UML в сравнении с ER-моделями (операции классов, связи-зависимости, ассоциативные связи) приводят к накладным расходам или снижению эффективности при получении реляционной схемы.

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

Язык SQL предназначен не только для удобной формулировки запросов, но и для манипулирования схемой БД, определения структур физического уровня и прочего.

Отличия SQL-модели от реляционной:

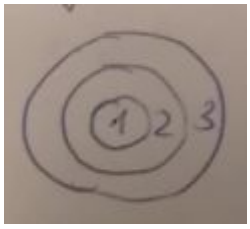
1. SQL - ориент. БД - это набор таблиц, содержащих некоторое мультимножество строк (могут быть повторения), соответствующих заголовку таблиц.
2. Поддерживается порядок столбцов, соответствующий порядку их определения, т.е. таблица это не отношение.

Части стандарта SQL:2003

1) Framework
2) Foundation
3) PSM (хранение процедур)
4) OLB (связь с ООП)
5) CLI
6) Schemata
7) JRT
8) XML
9) MED (управление внешними данными)

Структура языка SQL

Различные схемы основаны на принципе деления языка на уровни

1. базовый, промежуточный и полный (для производителей СУБД)
2. 

(Тарковский гений)
    - 2.1. прямой SQL (прямое взаимодействие пользователей с СУБД)
    - 2.2. встроенный SQL (взаимодействие прямого SQL с программами пользователей)
    - 2.3. динамический SQL (взаимодействие приложений с базами)

Язык БД SQL. Основные отличия SQL-орient. модели от реляц. модели. Стандарт SQL:2003 - основные тома. Структура языка SQL (3 рамоч. схемы)

Язык SQL предназначен не только для удобной формулировки запросов, но и для манипулирования схемой БД, опр структур физического уровня и пр.

Отличия SQL-модели от реляц.

1) SQL-орient. БД - это набор таблиц, содержащих нек-ое мультимножество строк (мб повторения), соотв заголовку табл.
2) Поддерживается порядок столбцов, соотв порядку их определения, т.е таблица это не отношение.

Части стандарта SQL:2003

1) Framework
2) Foundation
3) PSM (хранимые процедуры)
4) OLB (связь с ООП)
5) CLI
6) Schemata
7) JRT
8) XML
9) MED (управление внеш. данными)

Структура языка SQL.

Различные схемы основаны на принципе деления языка на уровни.

1) базовый, промежуточный и полный (для производителей СУБД)
2) 

1: прямой SQL (прямое в-ие польз с СУБД)
2: встроенный SQL (в-ие прямого SQL с программами польз-лей)
3: динамический SQL (обращение приложений)

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

Все данные в таблицах типизированы. Каждому столбцу приписывается свой тип.

Типы данных:

1) Логические. TRUE, FALSE, UNKNOWN = NULL
2) Точные числовые типы
    a) истинно целые типы (INTEGER, SMALLINT)
    b) типы с дробной частью (NUMERIC, DECIMAL)
3) Приближенные числовые типы. <мантисса, порядок>. REAL, FLOAT, DOUBLE PRECISION
4) Символьные строки. CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT
5) Битовые строки. BIT, BIT VARYING, BIT LARGE OBJECT
6) Дата и время, временная метка (возможно с временной зоной)
    DATE: ‘yyyy-mm-dd’
    TIME: ‘hh:mm-ss: f...f’
    TIMESTAMP: ‘yyyy-mm-dd hh:mm-ss: f...f’
    TIME WITH TIMEZONE = TIME с дополнительным компонентом зоны
7) Временные интервалы. INTERVAL start(s) [TO end(t)]. Разница между значением даты/времени (YEAR, MONTH и т.д.)

<u>Преобразование типов</u>

В SQL есть явное и неявное преобразование одного типа к другому.

- CAST: явное преобразование типов. CAST({expression|NULL} AS {type})
- Неявное преобразование: тип A приводим к типу B тогда и только тогда, когда если ожидается тип B может быть использован тип A.
  Например, тип CHARACTER(x) приводим CHARACTER(y) для всех y≥x

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

Все данные в таблицах типизированы. Каждому столбцу приписывается свой тип.

Типы данных.

1) логические
   TRUE, FALSE, UNKNOWN = NULL.
2) точные числовые типы
   - → истинно целые типы (INTEGER, SMALLINT)
   - → типы с дробной частью (NUMERIC, DECIMAL)
3) приближенные числовые типы
   <мантисса, порядок>. REAL, FLOAT, DOUBLE PRECISION
4) символьные строки
   CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT
5) битовые строки
   BIT, BIT VARYING, BIT LARGE OBJECT
6) дата и время, временная метка (вкл с временной зоной)
   DATE: 'yyyy-mm-dd'
   TIME: 'hh:mm-ss:f..f'
   TIMESTAMP: 'yyyy-mm-dd hh:mm-ss:f..f'
   TIME WITH TIMEZONE = TIME с доп компонентом зоны
7) временные интервалы
   INTERVAL start(s) [TO end(s)] Разница между зна даты/времени
   YEAR, MONTH и т.д.

Преобразование типов

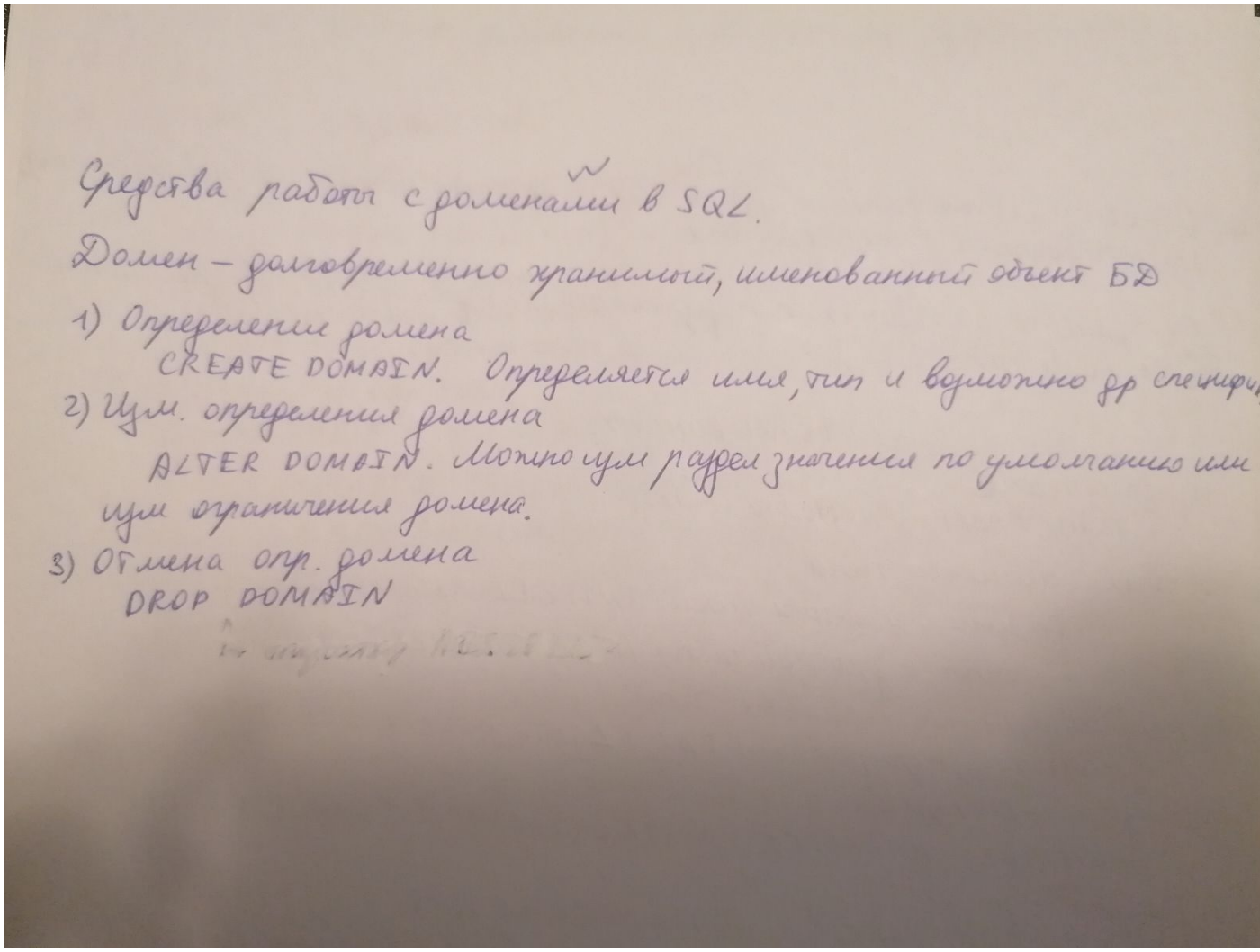
В SQL есть явное и неявное преобр одного типа к другому.

- CAST: явное преобр. типов
  CAST ({expression/NULL} AS {type}).
- Неявное преобр: тип A приводим к типу B ⇔ где ожидается тип B мб исп. тип A.
  Например: тип CHARACTER(x) приводим к CHARACTER(y) ∀ y ≥ x

44. Средства работы с доменами в SQL.

Домен - долговременно хранимый именованный объект базы данных.

1) Определение домена. CREATE DOMAIN. Определяется имя, тип и, возможно, другие спецификации
2) Изменение определения домена. ALTER DOMAIN. Можно изменить раздел значения по умолчанию или изменить ограничение домена
3) Отмена определения. DROP DOMAIN



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

1) Определение базовой таблицы

CREATE TABLE - создание базовой таблицы

- определение столбца (имя, типа (указывается явно или с помощью домена))
- значение столбца по умолчанию: DEFAULT
- ограничение целостности столбца
  NOT NULL, ограничение уникальности (PRIMARY KEY, UNIQUE)

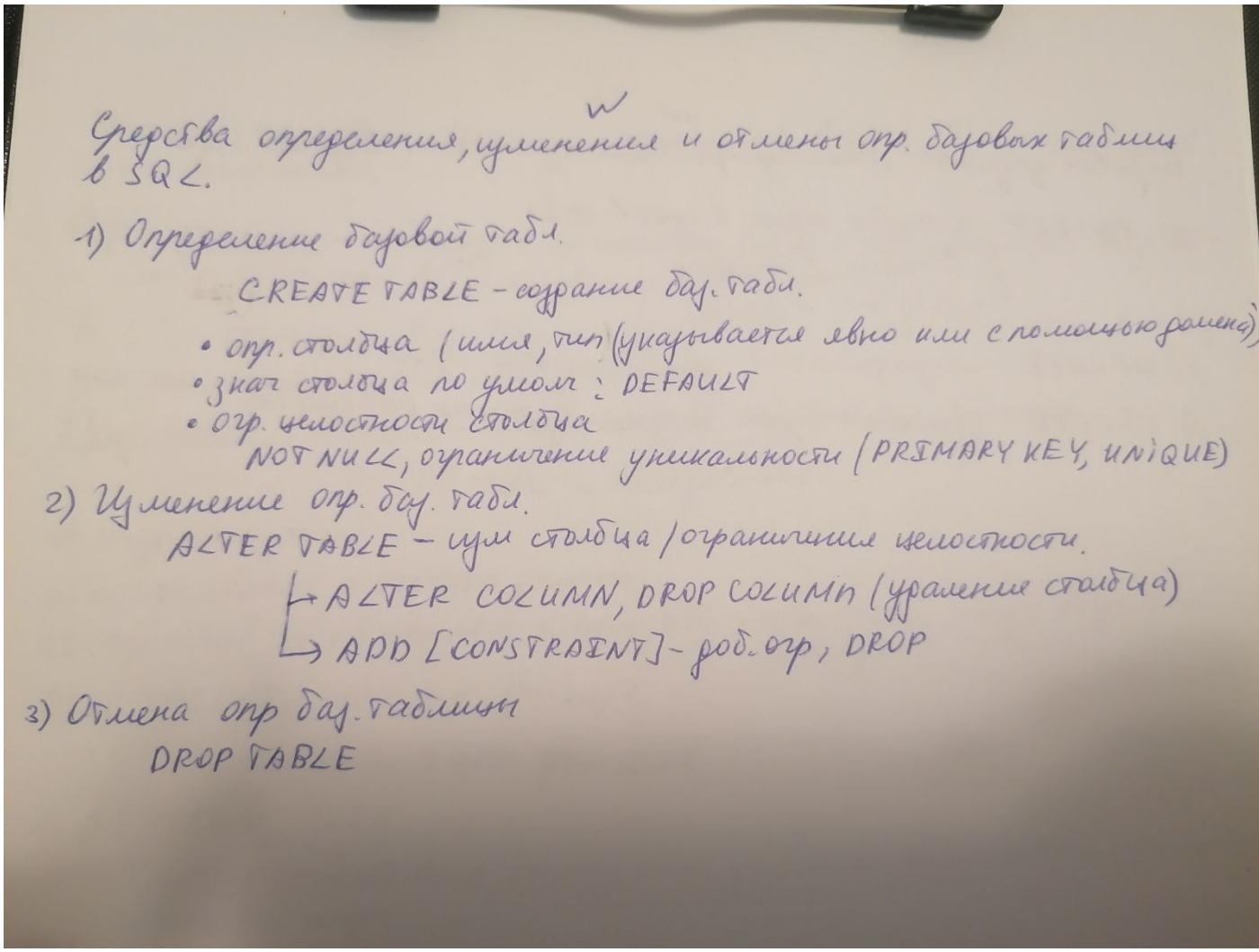
2) Изменение определения базовой таблицы

ALTER TABLE - изменение столбца/ограничения целостности

- ALTER COLUMN, DROP COLUMN (удаление столбца)
- ADD [CONSTRAINT] - добавление ограничения, DROP

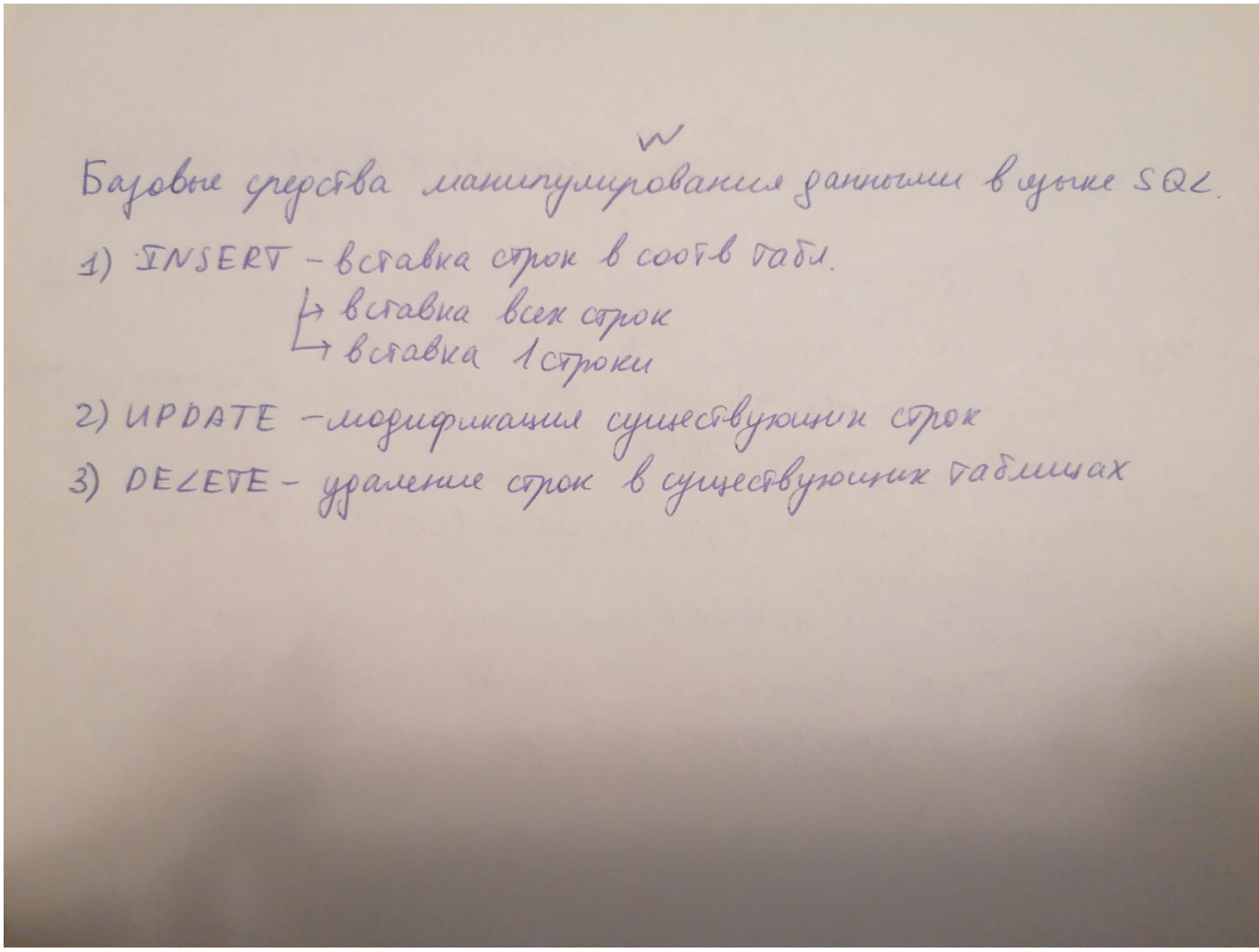
3) Отмена определения базовой таблицы

DROP TABLE



46. Базовые средства манипулирования данными в языке SQL.

4) INSERT - вставка строк в соответствующей таблице:
    a) вставка всех строк
    b) вставка одной строки
5) UPDATE - модификация существующих строк
6) DELETE - удаление строк в существующих таблицах



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

Триггер - процедура, хранимая в базе данных и автоматически вызываемая СУБД при возникновении соответствующих условий.

В SQL триггеры вызываются при вставке одной/нескольких строк в таблицу, при их модификации или удалении.

Предметная таблица - таблица, с которой связано определение триггера.

Области использования

1) журнализация
2) согласование и очистка данных
3) операции, не связанные с изменением базы данных (процедуры могут вызываться из триггеров)

DROP TRIGGER - отмена определения триггера

Типы триггеров

1) BEFORE и AFTER. Триггер срабатывает немедленно до/после выполнения инициирующего оператора
2) INSERT, UPDATE, DELETE. Вызываются при, соответственно, вставке строки, модификации, удаления
3) ROW, STATEMENT. Триггер сработает либо для каждой строки, либо 1 раз
4) WHEN. Более точно специфицируется условие применимости триггера.

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

Опр: **триггер** – процедура, хранимая в БД и автоматически вызываемая СУБД при возникновении соотв условий.

В SQL триггеры вызываются при вставке 1/нескольких строк в табл, при их модификации или удалении.

Предметная таблица – табл, с которой связано опр триггера.

Области использования

1) сигнализация
2) согласование и очистка данных.
3) операции, не связанные с исп БД (процедуры могут вызываться из триггеров)

DROP TRIGGER – отм опр триггера.

Типы триггеров

1) BEFORE и AFTER

   Триггер срабатывает немедленно до/после выполнения инициирующего оператора.

2) INSERT, UPDATE, DELETE

   Вызываются при соотв вставке строки, модификации/удалении.

3) ROW, STATEMENT

   Триггер срабатает либо для каждой стр, либо 1 раз

4) WHEN

   Более точно специфицируется условие применимости триггера.

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

Оператор выборки SELECT возвращает набор одной/нескольких строк одинаковой структуры.

SELECT [ALL | DISTINCT] …
FROM …
[WHERE …]
[GROUP BY …]
[HAVING …]
[ORDER BY…]
То, что курсивом - необязательно

7) Выполняется раздел FROM
8) Выполняется раздел WHERE. Условное выражение применяется ко всем строкам таблицы $T. \rightarrow T_1$
9) Выполняется GROUP BY. Каждый элемент списка имён столбцов должен входить в имена столбцов $T_1 \rightarrow T_2$
10) Выполняется HAVING. Условное выражение применяется к каждой группе строк таблицы T_2 и результатом является таблица, содержащая те группы строк, в которых выполняется выражение $\rightarrow T_3$
11) Выполняется ORDER BY. Выбирается 1 элемент списка сортировки и строки таблицы расставляются в порядке возрастания/убывания (ASC/DESC) выражения, вычисленного для каждой строки. и тд $\rightarrow T_4$

Общая структура оператора выборки в языке SQL и схема выполнения.

Оператор выборки SELECT возвр набор одной/неск-ких строк одинаковой структуры.

```
SELECT [ALL/DISTINCT]...
  FROM ...
  [WHERE...]
  [GROUP BY...]
  [HAVING...]
  [ORDER BY..]
```

} необяз

1) вып раздел FROM
2) вып раздел WHERE
   Условное выражение применяется ко всем стр табл $T_0 \rightarrow T_1$
3) вып GROUP BY
   Каждый эл-т списка имен столбцов должен входить в имена столбцов $T_1 \rightarrow T_2$
4) вып HAVING
   Условное выражение применяется к каждой группе строк табл T_2 и рез-том явл табл., содержащая те группы строк, в которых вып. выражение $\rightarrow T_3$
5) вып ORDER BY
   Вып-ется 1эл-т списка сортировки и строки табл. расставляются в порядке возраст/убыв (ASC/DESC) выражения, вычисленного для каждой строки. и т.д. $\rightarrow T_4$.

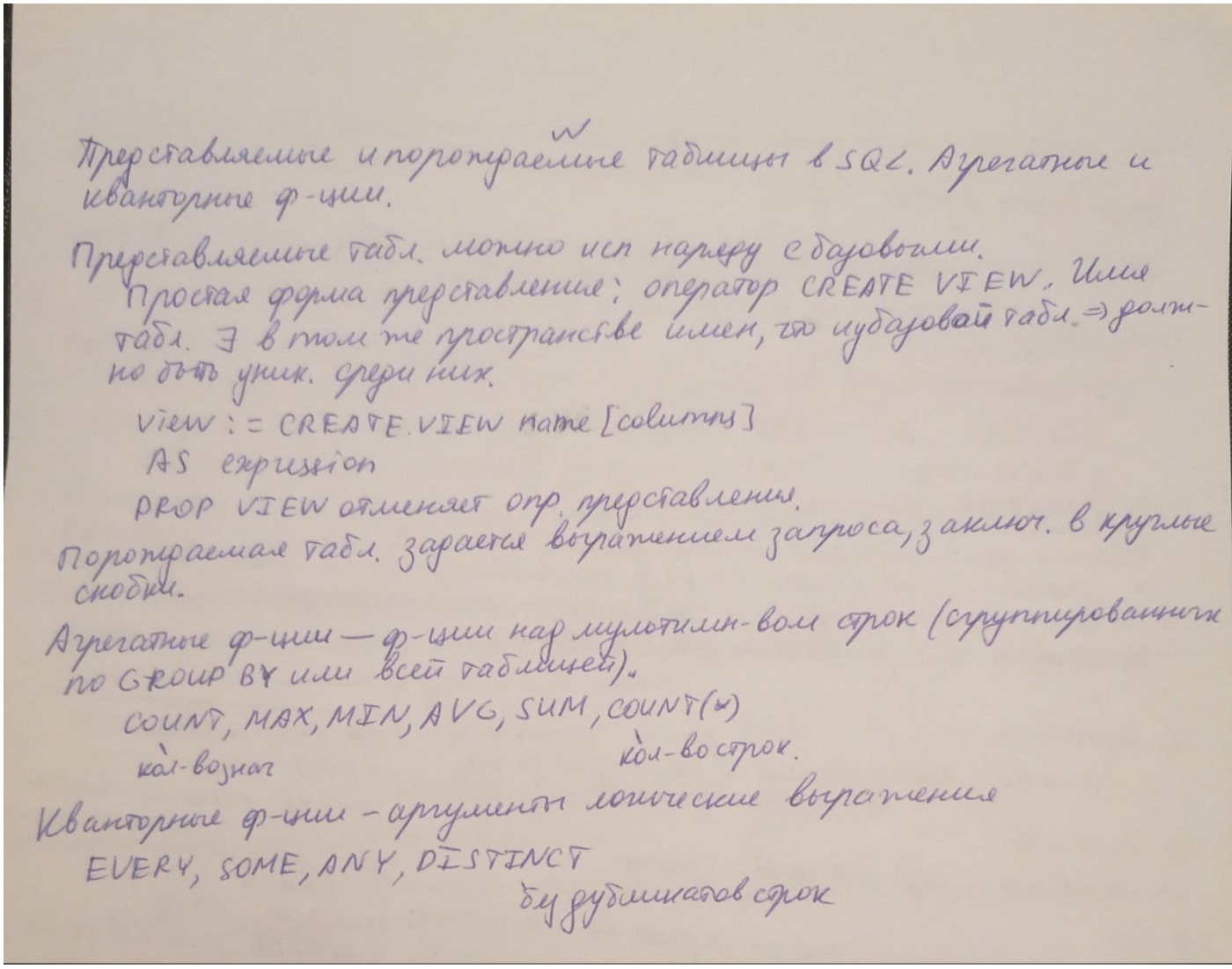
49. Представляемые и порождаемые таблицы в SQL. Агрегатные и кванторные функции.

Представляемые таблицы можно использовать наряду с базовыми.
Простая форма представления: оператор CREATE VIEW. Имя таблицы существует в том же пространстве имён, что и у базовой таблицы, поэтому должно быть уникальным среди них.
view := CREATE VIEW name[columns]
AS expression
PROP VIEW отменяет определённые представления

Порождаемая таблица задаётся выражением запроса, заключённым в круглые скобки.

Агрегатные функции - функции над мультимножеством строк (сгруппированных по GROUP BY или всей таблицей).
COUNT (количество значений), MAX, MIN, AVG, SUM, COUNT(*) (количество строк)

Кванторные функции - аргументы логические выражение.
EVERY, SOME, ANY, DISTINCT (без дубликатов строк)



50. Предикаты языка SQL.

Предикаты помогают специфицировать условие логического выражения.

1) предикат сравнения - сравнение двух строчных значений

```
SELECT DISTINCT comp.price
FROM comp
WHERE comp.name = ‘Lenovo’
```

- числа сравниваются обычным способом
- строки: если длина x не равна длине y, то более короткая строка расширяется символами набивки. Затем лексикографическое сравнение.
- битовые строки сравниваются по битам
- …

2) between - условие вхождения в диапазон значений

```
SELECT NAME
FROM comp
WHERE price BETWEEN 35000 AND 50000
```

3) is null - являются ли неопределёнными все значения строки
4) in - условие вхождения строчного значения во множество значений

```
SELECT NAME
FROM comp
WHERE price IN (30000, 50000, 70000)
```

5) like - сопоставление исходной строки с шаблоном

```
SELECT title
FROM base
WHERE title LIKE ‘%introduction’
```

6) similar - отличается от like расширением возможностей задания шаблона (построение регулярных выражений)
7) exists - если мощность таблицы результата больше 0, то true
8) unique - отсутствие дубликатов в запросе
9) overlaps - проверка пересечения во времени двух событий
10) сравнение с квантором

x comp_op ALL T (всеобщность)
x comp_op SOME T (существование)

11) match - соответствие строчного значения результату табличного подзапроса
12) is distinct - являются ли две строки дубликатами

Предикаты языка SQL.

Предикаты помогают специфицировать условия логич. выражения.

1) предикат сравнения.

Сравнение 2-ух строчных значений

```
SELECT DISTINCT comp.price
FROM comp
WHERE comp.name = 'Lenovo'
```

- Числа сравниваются обычным способом
- строки: если длина X ≠ длине Y, то более короткая строка расширяется символами набивки. Затем лексикографич. сравнение.
- битовые стр сравниваются по битам

...

2) between

Условие вхождения в диапазон знач.

```
SELECT NAME
FROM comp
WHERE price BETWEEN
      35000 AND 50000
```

3) is null

Явл ли неопр все знач. строки

4) in

Условие вхожд. строчного знач. во мн-во знач.

```
SELECT NAME
FROM comp
WHERE price IN (30.000,
      50.000, 70.000).
```

5) like

Сопоставление исх строки с шаблоном

```
SELECT title
FROM base
WHERE title LIKE
  '% Introduction'
```

6) similar

отлич от like расширением возможностей задания шаблона (постр регулярных выраж)

7) exists

мощность табл рез-та > 0 => true

8) unique

отсутствие дубликатов в запросе

9) overlaps

проверка пересеч. во времени 2-ух событий.

10) сравнение с квантором

x comp_op ALL T (всеобщность)

x comp_op SOME T (существование)

11) match

соотв строчного знач. рез-ту табличного подзапроса

12) is distinct - явл ли 2 стр дубликатами

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

Транзакция ACID - последовательность операций над базой данных, обладающая свойствами атомарности, согласованности, изоляции и долговечности.

Транзакции могут образовываться явным образом с помощью START TRANSACTION либо неявно, когда нет контекста транзакции, нужного для выполнения оператора.
Например, SELECT, UPDATE требуют контекст.

Для завершения транзакции используется COMMIT (фиксация результатов транзакции) или ROLLBACK (удаление всех выполненных операций).

Точка сохранения - пометка в последовательности транзакций, которая может быть использована для частичного отката транзакции с сохранением результата транзакций, выполненных до этой точки.
Установление точки: SAVEPOINT name

Уровни изоляции транзакций

1) феномен “грязного” чтения

Можно видеть изменения объектов, производимые другими ещё не зафиксированными транзакциями
В SQL наблюдается у транзакций, выполняющихся на уровне READ UNCOMMITTED

2) Феномен неповторяемого чтения

Транзакции читают некоторые объекты и допускают изменения уже прочитанных другими транзакциями объектов.
В SQL уровень READ COMMITTED

3) Феномен фантомов

Транзакции, выбирающие строки и таблицы базы данных и допускающие добавление к этим таблицам другими транзакциями строк, удовлетворяющих условию выборки.
В SQL уровень изоляции REPEATABLE READ

W

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

Опр: Транзакция ACID - пос-ть операций над БД, обладающая св-вами атомарности, согласованности, изоляции и долговечности.

Транзакции могут образовываться явным образом с помощью START TRANSACTION, либо неявно, когда нет контекста транзакции, нужного для ~~старта~~ выполнения оператора.

Например, SELECT, UPDATE требуют контекст.

Для завершения транзакции исп COMMIT (фиксация рез-тов тран) или ROLLBACK (удаление всех вып. операций)

Опр: Точка сохранения - пометка в пос-ти транзакций, которая мб использована для частичного отката транзакции с сохранением рез-та транзакций, выполненных до этой точки.

Установление точки: SAVEPOINT name.

Уровни изоляции транзакций.

1) феномен „грязного" чтения

Можно видеть изменения объектов, произведенные другими еще не зафикс. транзакциями.

В SQL наблюдается у транзакций, выполняющихся на уровне READ UNCOMMITTED.

2) феномен неповторяемого чтения

Транзакции читают нек-ые объекты и допускают изменения уже прочитанных др. транзакциями объектов.

В SQL уровень READ COMMITTED.

3) феномен фантомов

Транзакции, выполняющие строки и табл БД и допускающие добавление к этим табл. ~~строк~~ другими транзакциями строк, удовл условию выборки.

В SQL уровень изоляции REPEATABLE READ.

52. Иерархия ограничений в SQL. Средства определения и отмены общих ограничений (ограничений БД). Проверка ограничений и ее связь с механизмом транзакций.

Иерархия ограничений целостности

ограничения доменов
↓
ограничения столбцов
↓
табличные ограничения
↓
ограничения базы данных

Также есть дополнительные ограничения базы данных (общие ограничения целостности), которые называются ASSERTION.
Для определения общего ограничения служит оператор CREATE ASSERTION.
Для отмены общего ограничения используется DROP ASSERTION.

Проверка ограничений

Может быть немедленная или отложенная.
Определяется следующая конструкция:

- INITIALLY IMMEDIATE: в начале выполнения транзакции ограничение находится в режиме немедленной проверки
- INITIALLY DEFERRED: в начале любой транзакции ограничения находятся в режиме отложенной проверки.

Режим проверки может быть изменён с помощью SET CONSTRAINTS
DEFERRABLE - для этого ограничения может быть установлен режим отложенный проверки
NOT DEFERRABLE - не может

Иерархия ограничений в SQL. Средства огр и отмены общих огр. Проверка ограничений и ее связь с механизмом транзакций.

Иерархия ограничения целостности.

ограничения доменов
↓
огр. столбцов
↓
табличные огр.
↓
огр. баз данных.

общие огр целостности.

Также есть доп. огр. баз данных, которые наз. ASSERTION.
Для огр. общего ограничения служит оператор CREATE ASSERTION
Для отмены общего огр исп. DROP ASSERTION

Проверка ограничений

Может быть немерленная или отложенная.
Определяется спец. конструкция:

- INITIALLY IMMEDIATE: в нач. вып. транзакции ограничение находится в режиме немерл. проверки.
- INITIALLY DEFERRED: в нач в транзакции огр. находится в режиме отложенной проверки.

Режим проверки может быть изменен с помощью SET CONSTRAINTS
DEFERRABLE - для этого огр мб установлен режим отлож. проверки
NOT DEFERRABLE - не может.

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

В SQL контролируется доступ к следующим объектам БД:

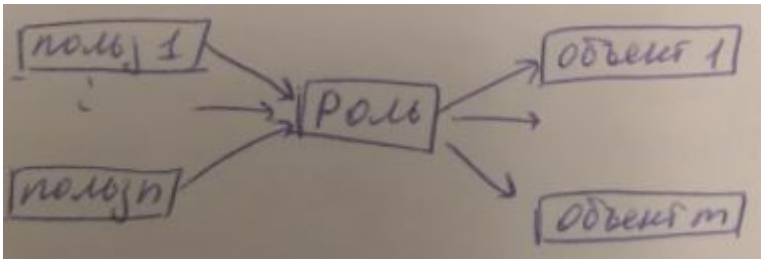
- таблицы и их столбцы
- представления
- домены
- триггеры
- преобразования
- и прочее

Вид защиты	**Привилегия**	**Объекты защиты**
просмотр	SELECT	таблицы, столбцы, подпрограммы
вставка	INSERT	таблицы, столбцы
модификация	UPDATE	таблицы, столбцы
удаление	DELETE	таблицы
использование	USAGE	домены, типы, определённые пользователем
выполнение	EXECUTIVE	подпрограммы
инициирование	TRIGGER	таблицы
ссылка	REFERENCES	таблицы, столбцы

Пользователь характеризуется своим идентификатором авторизации (authID).
authID в SQL:

- идентификатор пользователя
- имя роли

Роль идентифицирует группу пользователей с одними и теми же привилегиями для доступа к тем или иным объектам



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

В SQL контролируется доступ к след. объектам БД:

- таблицы и их столбцы
- представления
- домены
- триггеры
- преобразования

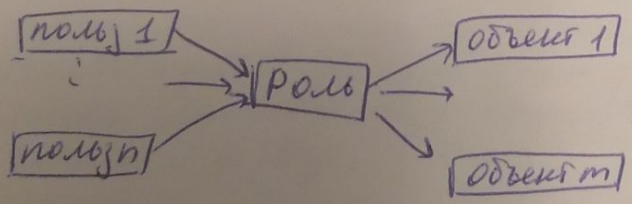
и пр.

Вид защиты	Привилегия	Объекты защиты
просмотр	SELECT	табл, столбцы, подпрограммы
вставка	INSERT	табл, столбцы
модификация	UPDATE	—"—"—"—
удаление	DELETE	табл.
использование	USAGE	домены, типы, опр пользователем
выполнение	EXECUTE	подпрограммы
инициирование	TRIGGER	таблицы
ссылка	REFERENCES	табл, столбцы

Пользователь характеризуется своим идентификатором авторизации (authID)

authID в SQL → идентиф. пользов.
→ имя роли

Роль идентифиц. группу пользов с одними и теми же привилегиями для доступа к тем или иным объектам



54. Передача и аннулирование привилегий и ролей в SQL.

Роль идентифицирует группу пользователей с одними и теми же привилегиями доступа к определённым объектам.

CREATE ROLE - создание новой роли. Имя роли должно отличаться от любых идентификаторов авторизации (authID).

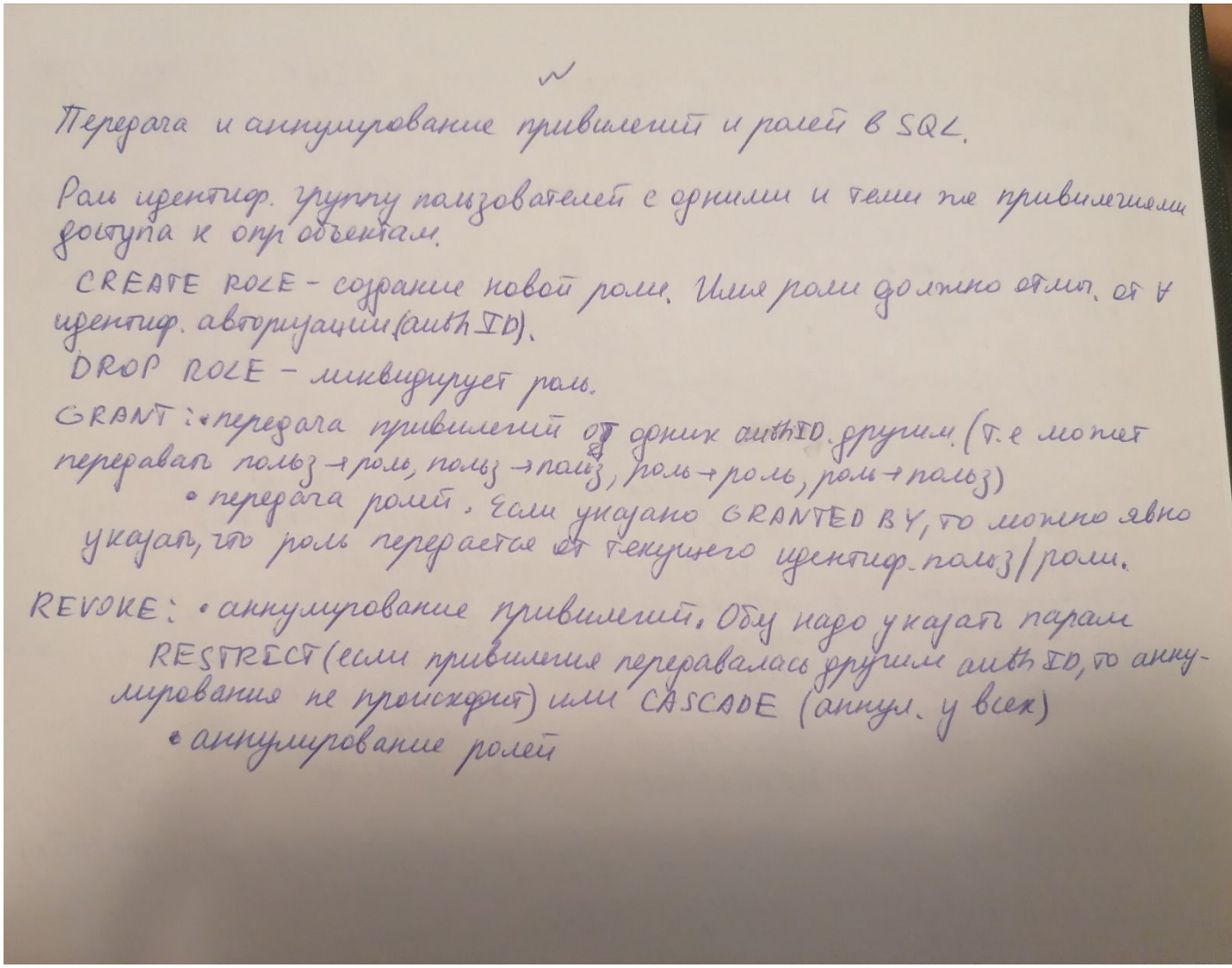
DROP ROLE - ликвидирует роль.

GRANT:

- передача привилегий от одних authID другим (т.е. может передавать пользователь→роль, пользователь→пользователь, роль→роль, роль→пользователь.
- передача ролей. Если указано GRANTED BY, то можно явно указать, что роль передаётся от текущего идентификатора пользователя/роли.

REVOKE:

- аннулирование привилегий. Обязательно надо указать параметр RESTRICT (если привилегия передавалась другим authID, то аннулирования не происходит) или CASCADE (аннулирование у всех)
- аннулирование ролей



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

В объектно-ориентированной модели данных (ODMG) база данных - это набор объектов произвольного типа.

Всего 2 типа:

- материальные - обычные типы данных
- объектные
    - атомарные - для определения нужно указать внутреннюю структуру и набор операций
    - коллекции - можно определить типы множеств, мультимножеств, список и словарей. Можно создать объект.

Манипулирование данными в ODMG.

Для этого есть язык OQL - язык запросов.

- есть высокоуровневые средства для работы с множествами объектов, структурами, списками и массивами.
- функциональный язык, не ограничивающий композицию операций
- не является вычислительно полным языком
- нет явных операций обновления, могут вызываться операции, определённые в целях обновления
- декларативный доступ к объектам

Ограничения целостности.

В ODMG 2 объекта считаются совпадающими, если являются одним и тем же объектом, следовательно нет аналога ограничений целостности в реляционной модели. В атомарном типе можно задать ключ, но это не будет ограничениями целостности. Он способствует более эффективной организации запросов.

Ссылочная целостность поддерживается, если между двумя атомарными объектными типа определена связь “1 ко многим”.

.

N

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

В объектно-ориент. модели данных (ODMG) БД - набор объектов произв. типа.

Всего 2 типа:

- литеральные - обычные типы данных
- объектные
  - → атомарные (для определения нужно указать внутр структуру и набор операций)
  - → коллекции (можно опр типы мн-в, мультимн-в, списков и словарей)

  можно создать ~~ед~~ объект.

Манипулирование данными в ODMG

Для этого есть язык OQL - язык запросов.

- есть высокоуровневые средства для работы с мн-вами объектов, структурами, списками и массивами.
- функциональный язык, не ограничивающий композицию операций.
- не явл вычислительно полным языком.
- нет явных операций обновления, могут вызываться операции, опр в целях обновления.
- декларативный доступ к объектам.

Ограничения целостности.

В ODMG 2 объекта считаются совпадающими, если явл. одним и тем же объектом. =) нет аналога огр. целостности в реляц. модели. В атомарном типе можно задать ключ, но это не будет огр целостности. Он способствует более эфф. организации запросов.

Ссылочная целостность порр, если между 2-мя атомарными объектными типами опр свяр „1 ко мн".

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

Объектная модель включает структурные типы данных, определённых пользователем (UDT) и типизированные таблицы. То есть можно определять новые типы данных, отличные от встроенных, а типизированные таблицы - это таблицы, строки которых являются экземплярами типа UDT.

1) UDT
    - индивидуальные типы - основаны на единственном встроенном типе. Этот тип нельзя непосредственно использовать в операциях базового типа, но можно явно привести его к базовому.
    - структурные типы - именованный тип данных, включающий 1 или более атрибутов любых допустимых в SQL типов данных.
2) Типизированные таблицы

Указывается ранее определённый структурный тип. Если в нём n атрибут, то n+1-ый столбец в таблице содержит уникальный типизированный идентификатор строк.

Способ генерации значений при определении структурного типа:

1. SYSTEM GENERATED - аналог ООБД
2. USER DEFINED (TYPE) - определённый тип уникального идентификатора. Пользователь указывает значение, когда вставляет строки.
3. USER DEFINED (список столбцов) - аналог ключа.

Объектно-реляционные расширения языка SQL. Возможные подходы к объектно-реляц. отображению без исп. объектно-реляц. расширений SQL.

Объектная модель вкл. структурные типы данных, опр пользователем (UDT) и типизированные таблицы.

То есть можно опр новые типы данных, отличные от встроенных и типиз. таблицы – табл, строки которых явл экземплярами типа UDT.

1) UDT

- индивидуальные типы – основанные один-ом встроенном типе. Этот тип нельзя непосредственно исп. в операциях баз. типа, но можно явно привести его к базовому.
- структурные типы – именованной тип данных, вкл 1 или более атрибутов ∀ допустим в SQL типа данных.

2) Типизированные табл.

Указывается ранее опр структурный тип. Если в нем n атрибутов, то n+1 столбец в табл содержит уник. типизированный идентификатор строк.

Способ генерации значений при опр структурного типа

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

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

В истинной реляционной модели вводятся три типа данных:

1. скалярные типы - инкапсулированный тип;
2. кортежные типы: генератор TUPLE с парой <имя, тип>;
3. типы отношений: генератор RELATION с заголовком кортежа.

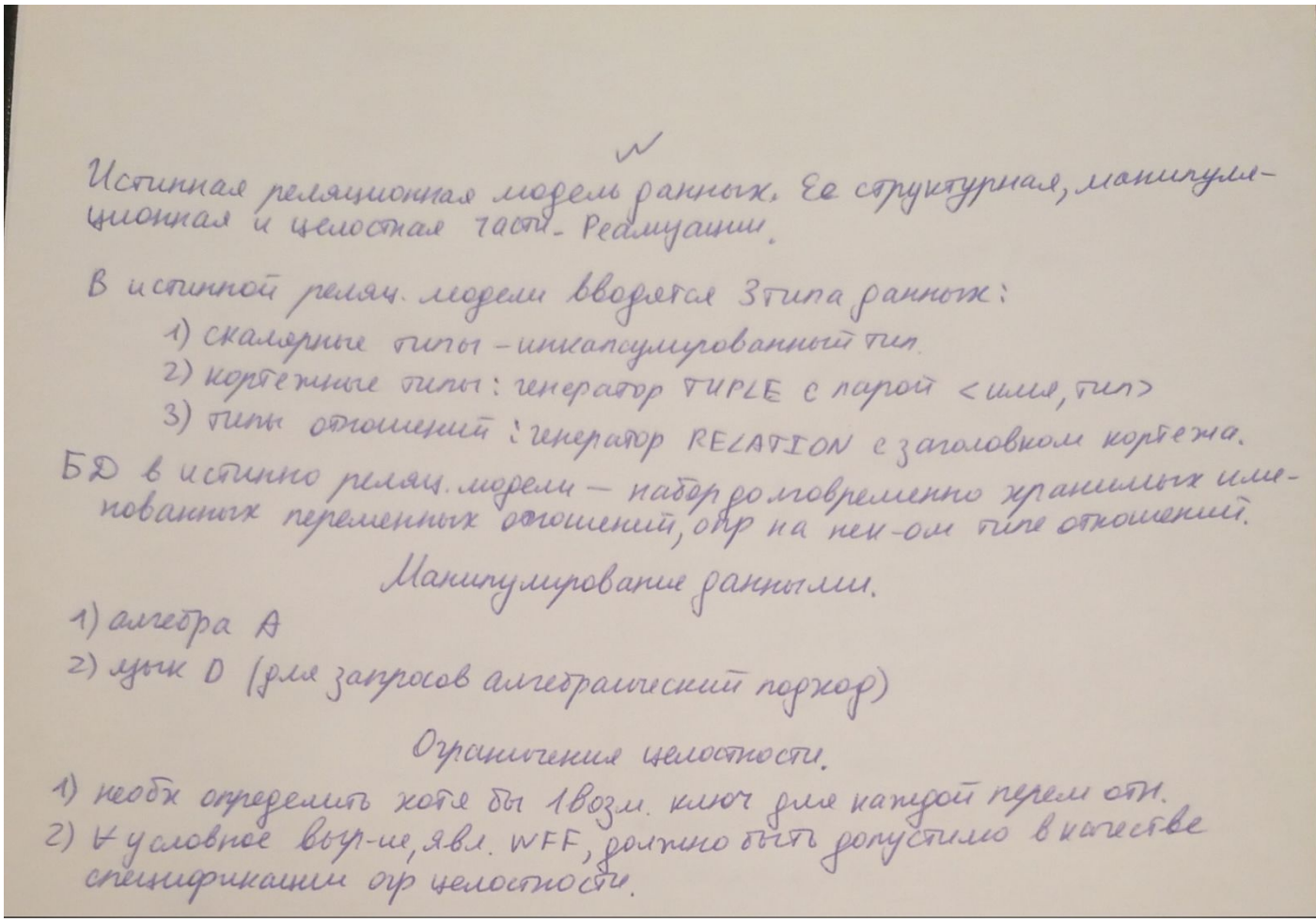
База данных в истинной реляционной модели - это набор долговременно хранимых именованных переменных отношений, определённых на некотором типе отношений.

Манипулирование данными:

1) алгебра A
2) язык D (для запросов алгебраический подход)

Ограничения целостности:

1) необходимо определить хотя бы один возможных ключ для каждой переменной отношения
2) любое условное выражение, являющееся WFF, должно быть допустимо в качестве спецификации ограничений целостности.



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

В истинной реляц. модели вводятся 3 типа данных:
1) скалярные типы – инкапсулированный тип
2) кортежные типы: генератор TUPLE с парой <имя, тип>
3) типы отношений: генератор RELATION с заголовком кортежа.

БД в истинно реляц. модели – набор долговременно хранимых именованных переменных отношений, опр на нек-ом типе отношений.

Манипулирование данными.

1) алгебра A
2) язык D (для запросов алгебраический подход)

Ограничения целостности.

1) необх определить хотя бы 1 возм. ключ для каждой перем отн.
2) ∀ условное выр-ие, явл. WFF, должно быть допустимо в качестве спецификации огр целостности.