1. Концепция и технология баз данных. Понятие банка данных, базы данных, СУБД.

Концепция баз данных

До появления концепции БД и соответствующих этой концепции программных средств управление данными во внешней памяти производилось с помощью файловых систем, которые являются подсистемой ОС. Но их возможности для информационного моделирования ПО ограничены.

Основные черты концепции БД:

- данные отделяются от прикладной программы (ПП), появляется специальная программная надстройка для управления данными, называемая системой управления базами данных (СУБД); СУБД управляет данными и служит посредником между ними и ПП; ПП упрощаются, освобождаются от функций структуризации, хранения и поиска данных;
- появляются стандартизированные данные о фактографических данных – метаданные, управляемые СУБД; метаданные описывают информационные параметры и взаимосвязи фактографических данных о ПО;
  СУБД совместно с метаданными представляет собой стандартизированное инструментальное средство для моделирования ПО различной природы;
- происходит централизация (интеграция) данных, их многоаспектное использование для различных приложений, что сокращает избыточность данных, позволяет обеспечить более высокий уровень достоверности данных и оптимизировать различные процедуры ведения и использования БД.

Принято считать, что использование концепции баз данных позволяет:

1. повысить надежность, целостность и сохранность данных;
2. сохранить затраты интеллектуального труда;
3. обеспечить простоту и легкость использования данных;
4. обеспечить независимость прикладных программ от данных (изменений их описаний и способов хранения);
5. обеспечить достоверность данных;
6. обеспечить требуемую скорость доступа к данным;
7. стандартизовать данные в пределах одной предметной области;
8. автоматизировать реорганизацию данных;
9. обеспечить защиту от искажения и уничтожения данных;
10. сократить дублирование информации за счет структурирования данных;
11. обеспечить обработку незапланированных запросов к хранимой информации;
12. создать предпосылки для создания распределенной обработки данных.

Понятие базы данных

Под **базой данных** (БД) понимают совокупность хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений. Целью создания баз данных, как разновидности информационной технологии и формы хранения данных, является построение системы данных, не зависящих от принятых алгоритмов (программного обеспечения), применяемых технических средств и физического расположения данных в ЭВМ; обеспечивающих непротиворечивую и целостную информацию при нерегламентируемых запросах. БД предполагает многоцелевое ее использование (несколько пользователей, множество форм документов и запросов одного пользователя).

Понятие банка данных

Автоматизированный **банк данных** – это система информационных, математических, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоаспектного использования данных в некоторой предметной области. Банк данных включает в себя одну или несколько баз данных логически связанных между собой, систему управления ими (СУБД) и комплекс прикладных программ.

Банк данных должен обеспечить

- хранение и модификацию больших объемов многоаспектной информации;

- заданный уровень достоверности и непротиворечивость хранимой информации, ее восстановление после сбоев и отказов;
- поиск информации по произвольной совокупности признаков;
- одновременное обслуживание большого числа пользователей;
- оперативность в обработке запросов;
- простоту обращения;
- доступ к данным лишь тех пользователей, которые имеют необходимые полномочия.

Понятие СУБД

База данных предполагает наличие некоторого программного обеспечения, позволяющего пользователям работать с базой данной. Это программное обеспечение разрабатывается с помощью инструментальных программных средств, называемых **системой управления базами данных (СУБД)**. С помощью СУБД можно создавать базы данных, модифицировать данные в базе данных, вносить новые данные, разрабатывать пользовательские приложения. СУБД должна выполнять некоторые задачи по администрированию и поддержанию непротиворечивости данных. СУБД - это инструмент, с помощью которого создается та или иная конкретная база данных.

Отметим разницу между базой данных и системой управления базой данных. Если какая-то фирма пишет в объявлении, что она продает базу данных, то это означает, что она продает информацию. Если же в рекламе написано о СУБД, то следует ожидать, что Вам предложат программные средства, с помощью которых Вы соберете свою собственную базу данных. Хотя, в реальной жизни, понятия базы данных и системы управления базой данных часто смешивают.

2. Функции СУБД. Архитектура СУБД. Компоненты архитектуры и их характеристика.

Функции СУБД

- ### Непосредственное управление данными во внешней памяти

Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным в некоторых случаях (обычно для этого используются индексы).

- ### Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

- ### Управление транзакциями

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Таким образом, поддержание механизма транзакций является обязательным условием СУБД.
То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД.

- ### Журнализация

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя.
Поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

- ### Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков.

- **язык описания данных (ЯОД)** (SDL - Schema Definition Language), называемый также языком описания схем, - для построения структуры («шапки») таблиц БД;
- **язык манипулирования данными (ЯМД)** (DML - Data Manipulation Language) - для заполнения БД данными и операций обновления (запись, удаление, модификация);
- **язык запросов** - язык поиска наборов величин в файле в соответствии с заданной совокупностью критериев поиска и выдачи затребованных данных без изменения содержимого файлов и БД (язык преобразования критериев в систему команд).

В настоящее время функции всех трех языков выполняет язык SQL.

Архитектура СУБД. Компоненты архитектуры и их характеристика.

Архитектура СУБД обеспечивает потребности различных пользователей, выполнение их запросов, а также внутренние потребности, связанные с представлением данных в файлах и доступом к ним. Общепринятым в настоящее время является подход, обеспечивающий трехуровневое представление данных:

- на уровне внешних моделей соответствующих различным запросам различных пользователей;
- на логическом уровне, соответствующем интегральному взгляду на данные администратора ПО и администратора БД;
- на внутреннем уровне, соответствующем взгляду на данные системных программистов.

СУБД поддерживает различные описания данных на всех уровнях и их преобразования из одних видов представления в другие.

Компоненты архитектуры и их характеристика

- **Ядро**

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Ядро СУБД является основной резидентной частью СУБД.

- **Компилятор языка базы данных**

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу, как правило, на машинно-независимом исполняемом внутреннем коде

- **Подсистема поддержки времени исполнения**

которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД

- **Сервисные программы (внешние утилиты)**

Наконец, 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. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

3. Основные свойства баз данных.

- Высокое быстродействие (малое время отклика на запрос).
- Простота обновления данных.
- Независимость данных.
- Совместное использование данных многими пользователями.
- Безопасность данных - защита данных от преднамеренного или непреднамеренного нарушения секретности, искажения или разрушения.
- Стандартизация построения и эксплуатации БД (фактически СУБД).
- Адекватность отображения данных соответствующей предметной области.
- Дружелюбный интерфейс пользователя.

Время отклика - промежуток времени от момента запроса к БД до фактического получения данных.

Важнейшими являются первые два противоречивых требования: повышение быстродействия требует упрощения структуры БД, что, в свою очередь, затрудняет процедуру обновления данных, увеличивает их избыточность.
Независимость данных - возможность изменения логической и физической структуры БД без изменения представлений пользователей.
Независимость данных предполагает инвариантность к характеру хранения данных, программному обеспечению и техническим средствам. Она обеспечивает минимальные изменения структуры БД при изменениях стратегии доступа к данным и структуры самих исходных данных.
Безопасность данных включает их целостность и защиту.
Целостность данных - устойчивость хранимых данных к разрушению и уничтожению, связанных с неисправностями технических средств, системными ошибками и ошибочными действиями пользователей.
Она предполагает:
отсутствие неточно введенных данных или двух одинаковых записей об одном и том же факте;
защиту от ошибок при обновлении БД;
невозможность удаления (или каскадное удаление) связанных данных разных таблиц;
неискажение данных при работе в многопользовательском режиме и в распределенных базах данных;
сохранность данных при сбоях техники (восстановление данных).
Целостность обеспечивается триггерами целостности - специальными приложениями-программами, работающими при определенных условиях.
Защита данных от несанкционированного доступа предполагает ограничение доступа к конфиденциальным данным и может достигаться:
введением системы паролей;
получением разрешений от администратора базы данных (АБД);
запретом от АБД на доступ к данным;
формирование видов - таблиц, производных от исходных и предназначенных конкретным пользователям.
Стандартизация обеспечивает преемственность поколений СУБД, упрощает взаимодействие БД одного поколения СУБД с одинаковыми и различными моделями данных. Стандартизация (ANSI/SPARC) осуществлена в значительной степени в части интерфейса пользователя СУБД и языка SQL. Это позволило успешно решить задачу взаимодействия различных реляционных СУБД с помощью языка SQL.

4. Этапы проектирования баз данных и их характеристика.

- ### Этап формулирования и анализа требований

На этапе формулирования и анализа требований устанавливаются цели организации, определяются требования к БД. Они состоят из общих требований и специфических требований. Для формирования специфических требований обычно используется методика интервьюирования персонала различных уровней управления. Все требования документируются в форме, доступной конечному пользователю и проектировщику БД.

- ### Этап концептуального проектирования

Этап концептуального проектирования заключается в описании и синтезе информационных требований пользователей в первоначальный проект БД. Исходными данными могут быть совокупность документов пользователя при классическом подходе или алгоритмы приложений (алгоритмы бизнеса) при современном подходе. По окончании данного этапа получаем концептуальную модель, инвариантную к структуре базы данных. Часто она представляется в виде модели «сущность-связь».

- ### Этап логического проектирования

В процессе логического проектирования высокоуровневое представление данных преобразуется в структуру используемой СУБД. Основной целью этапа является устранение избыточности данных с использованием специальных правил нормализации. Цель нормализации - минимизировать повторения данных и возможные структурные изменения БД при процедурах обновления. Это достигается разделением (декомпозицией) одной таблицы в две или несколько с последующим использованием при запросах операции навигации. Заметим, что навигационный поиск снижает быстродействие БД, т.е. увеличивает время отклика на запрос. На выходе получаем СУБД-ориентированную структуру базы данных и спецификации прикладных программ. На этом этапе часто моделируют базы данных применительно к различным СУБД и проводят сравнительный анализ моделей.

- ### Этап физического проектирования

На этапе физического проектирования решаются вопросы, связанные с производительностью системы, определяются структуры хранения данных и методы доступа.

Различие уровней представления данных на каждом этапе проектирования реляционной базы данных:

КОНЦЕПТУАЛЬНЫЙ УРОВЕНЬ — Представление аналитика (используется инфологическая модель «сущность-связь»)

- сущности
- атрибуты
- связи

ЛОГИЧЕСКИЙ УРОВЕНЬ — Представление программиста

- записи
- элементы данных
- связи между записями

ФИЗИЧЕСКИЙ УРОВЕНЬ — Представление администратора

- группирование данных
- индексы
- методы доступа

Средства проектирования и оценочные критерии используются на всех стадиях разработки. В настоящее время неопределенность при выборе критериев является наиболее слабым местом в проектировании БД. Это связано с трудностью описания и идентификации большого числа альтернативных решений.
Основными причинами низкой эффективности проектируемых БД могут быть:

- недостаточно глубокий анализ требований (начальные этапы проектирования), включая их семантику и взаимосвязь данных;
- большая длительность процесса структурирования, делающая этот процесс утомительным и трудно выполняемым при ручной обработке.

В этих условиях важное значение приобретают вопросы автоматизации разработки.

5. Case-средства для проектирования БД. Общая характеристика. Примеры.

Можно вписать что-нито про ERWIN)

Подавляющее большинство подобных систем, представленных на рынке, обеспечивает автоматизированное преобразование диаграммных концептуальных схем баз данных, представленных в той или иной семантической модели данных, в реляционные схемы, специфицированные чаще всего на языке SQL. Не автоматические потому, что в типичной схеме SQL-ориентированной БД могут содержаться определения многих объектов (ограничений целостности общего вида, триггеров и хранимых процедур и т. д.), которые невозможно сгенерировать автоматически на основе концептуальной схемы. Поэтому на завершающем этапе проектирования реляционной схемы снова требуется ручная работа проектировщика.

CASE-технология базируется на методологии системного анализа. Под системным анализом понимают научную дисциплину, разрабатывающую общие принципы исследования сложных объектов и процессов с учетом их системного характера. Его основная цель - сосредоточить внимание на начальных этапах разработки. В рамках CASE-технологии системный анализ предназначен для отделения проектирования от программирования. В разработке в соответствии с CASE-технологией выделяются построение архитектуры и ее последующая реализация, поэтому системный анализ называют структурным системным анализом или просто структурным анализом. Важнейшими (базовыми) принципами являются деление (декомпозиция) и последующее иерархическое упорядочение.

Они дополняются следующими принципами.

- Принцип абстрагирования от несущественных деталей (с их «упрятыванием») с контролем на присутствие лишних элементов.
- Принцип формализации.
- Принцип концептуальной общности (структурный анализ - структурное программирование - структурное тестирование). Отсюда методология структурного анализа - метод исследования от общего обзора через детализацию к иерархической структуре со все большим числом уровней.
- Принцип непротиворечивости - обоснование и согласованность элементов.
- Принцип логической и физической независимости данных.
- Принцип непосредственного доступа (без программирования) конечного пользователя.

Эта технология положена в основу реализации программных CASE-средств.

Пакет CASE-средств обычно содержит 4 основных компонента.

- Средства централизованного хранения информации о всем проекте (своеобразная база данных проекта).
- Средства ввода данных для хранения.
- Средства анализа, проектирования и разработки.
- Средства вывода.

Для CASE-технологии (сокращенно - CASE) характерны четыре основных типа графических диаграмм:

- функциональное проектирование (DFD);
- моделирование данных (ERD);
- моделирование поведения (STD);
- структурные диаграммы (карты) - отношения между модулями и внутри- модульная структура.

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

Примеры

Проектирование БД существенно упрощается при применении ERWin (фирма Logic Works), Designer/2000 (Oracle), позволяющих проводить логическое моделирование данных, автоматическое преобразование данных в 3НФ.

6. Модели данных в БД. Основные понятия и определения.

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Упомянутые объекты позволяют моделировать структуру данных, а операторы — поведение данных. (***по Дейту***)

Используя это определение, можно эффективно разделить понятия модели данных и ее реализации.
Реализация (implementation) заданной модели данных — это физическое воплощение на реальной машине компонентов абстрактной машины, которые в совокупности составляют эту модель.
Короче говоря, модель — это то, о чем пользователи должны знать, а реализация - это то, чего пользователи не должны знать.

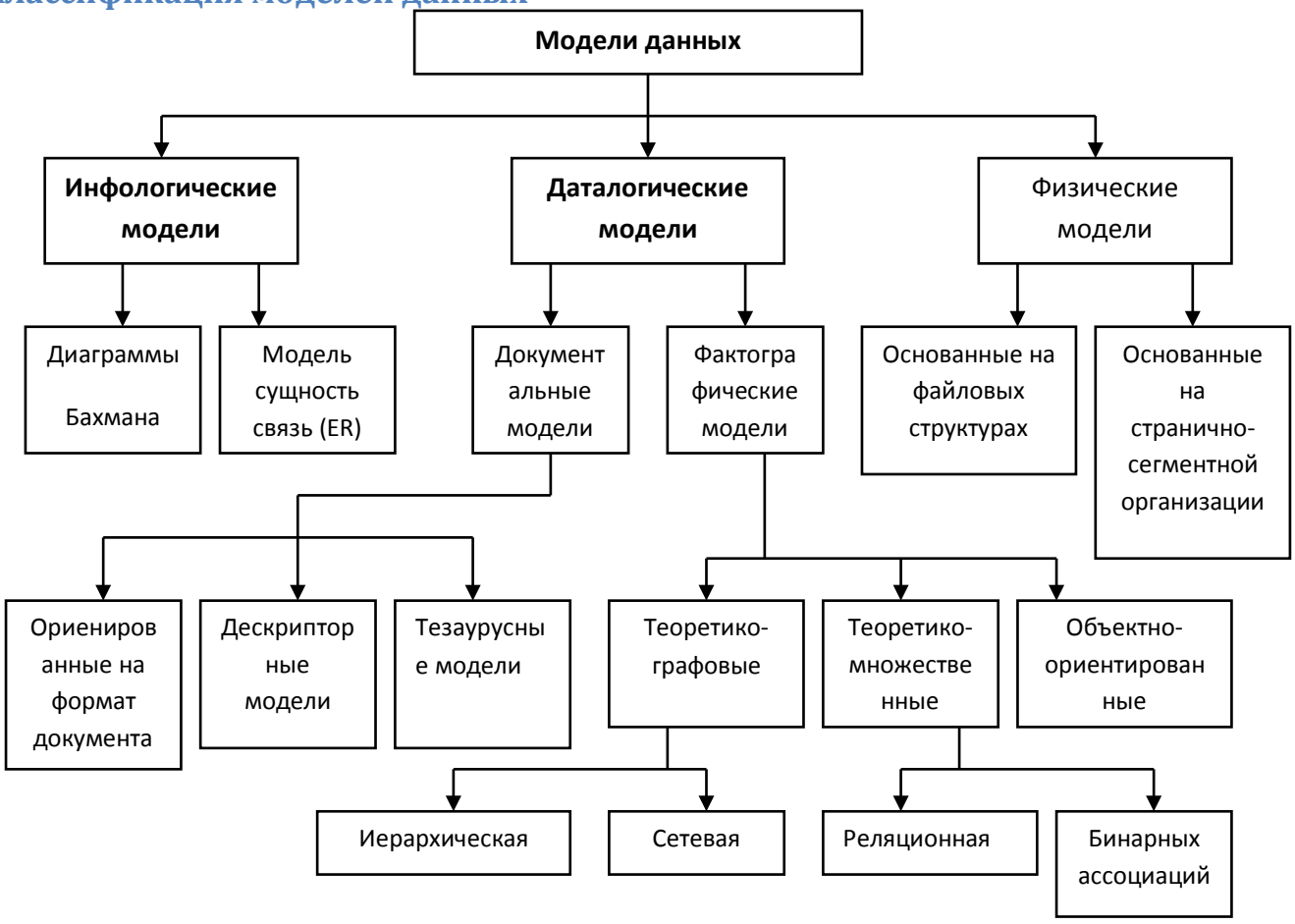
Одним из первых специалистов, который достаточно формально определил это понятие, был Э. Кодд. Он определил **модель данных как комбинацию трех компонентов**:

- Коллекции типов объектов данных, образующих базовые строительные блоки для любой базы данных, соответствующей модели
- Коллекции общих правил целостности, ограничивающих набор экземпляров тех типов объектов, которые законным образом могут появиться в любой такой базе данных
- Коллекции операций, применимых к таким экземплярам объектов для выборки и других целей.

(***по Кодду***)

Каждая БД и СУБД строится на основе некоторой явной или неявной модели данных. Все СУБД, построенные на одной и той же модели данных, относят к одному типу. Например, основой реляционных СУБД является реляционная модель данных, сетевых СУБД — сетевая модель данных, иерархических СУБД — иерархическая модель данных и т.д.

Классификация моделей данных



7.Характеристика компонент моделей данных (реляционной, иерархической, сетевой). Абстракции в моделях данных. Примеры.

Иерархическая модель данных

Иерархические, или **древовидные**, *структуры* данных разработаны и используются достаточно давно. Например, большинство *методов индексирования* базируются именно на древовидных структурах данных. Иерархическая *модель* данных близка по своей идее к иерархической *структуре* данных. Но *модель* описывает не конкретные методы работы и манипулирования ссылками, а **способ логического представления данных**, то, какими терминами оперирует проектировщик структуры базы данных, когда отражает реальные зависимости с помощью имеющихся в СУБД механизмов.

Иерархическая модель позволяет строить иерархию элементов. То есть у каждого элемента может быть несколько “наследников” и существует один “родитель”. Для каждого уровня связи вводится интерпретация, зависящая от предметной области и описывающая взаимоотношение между “родителями” и “наследниками”. Каждый **элемент** представляется с помощью **записи**. Структура данных, обычно используемая для представления этой записи об элементе, обычно содержит некоторые атрибуты, характеристики каждого элемента.

Попробуем представить себе базу данных для описания тематических сборников по некоторой теме. Прежде всего, выделим уровни иерархии.

Первый уровень - это **издательства.** Каждое издательство характеризуется своим *названием, юридическим адресом, номером счета в банке*.

Каждое издательство выпускает несколько **сборников**. То есть издательство является “родителем” для сборника и связано со сборником соотношением “издает” (“публикует”, “печатает” и т.д.). Для каждого сборника появляются такие атрибуты, как *размер, периодичность, цена, ответственный редактор, корректор* и т.д.

В каждом сборнике есть несколько **статей** (хотя бы, одна). То есть сборник и статья связаны соотношением “включает”. Далее, у каждой статьи есть *название, авторы*.

Авторы представляются отдельным элементом и образуют следующий уровень иерархии. Каждый автор характеризуется *фамилией, именем, отчеством, гонораром* и т.д. Статьи связаны с автором соотношением “написаны”.

Графическое представление этого примера приведено на *Рис.2.2.* Элементы нарисованы прямоугольниками, их названия даны обычным шрифтом. Связи нарисованы стрелками и их названия даны курсивом, атрибуты для каждого элемента на этой схеме не показаны - они являются частью элемента данных.

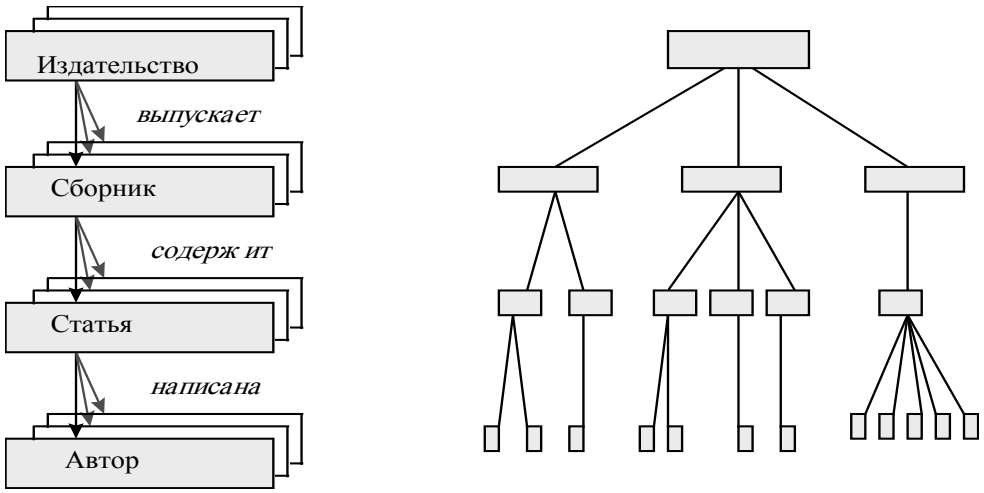


Рис.2.2. Графическое представление иерархической модели данных (справа пример какой-то конкретной базы данных)

Достоинства иерархических СУБД: возможность реализовать быстрый поиск нужных значений, когда условия запроса соответствуют иерархии в схеме базе данных.

Недостатки иерархических СУБД: например, если запрос не соответствует имеющейся иерархии, то и его программирование, и его исполнение, потребуют значительных усилий. Например, попытки реализовать запрос типа *“в скольких сборниках статей опубликовал свои статьи какой-либо автор”* может оказаться весьма трудной задачей (мы можем искать в направлении от статьи к автору, но не наоборот). Другим недостатком иерархической модели является сложность внесения в нее изменений. Если, по каким-то причинам изменились условия задачи, и модель предметной области перестала быть иерархической (например, в нашем примере, мы хотим иметь не только зависимость авторов от статьи, но и статей от автора), то приведение схемы базы данных в соответствие предметной области становится нетривиальной задачей.

Недостатки иерархической модели проистекают оттого, что данная модель слишком жесткая. Иерархическая модель очень хорошо подходит для устоявшихся предметных областей с четкими зависимостями “родитель-потомок”, то есть к моделям, где есть четкая субординация между понятиями.

Сетевая модель данных

Сетевая модель данных является развитием иерархической модели. В сетевой модели, так же как и в иерархической модели, есть понятие элемента данных и связи, которая может быть именована. Главное отличие сетевой модели от иерархической заключается в том, что к каждому элементу может идти связь не от одного элемента (“родителя”), а от нескольких.

Например, генеалогическое дерево, построенное только по мужской линии (или, только по материнской), является древовидной, иерархической структурой - у каждого человека (элемента), есть только один родитель. Если же включать в генеалогическое дерево всех родителей, то такое дерево с точки зрения структур данных будет уже не деревом, а сетью:

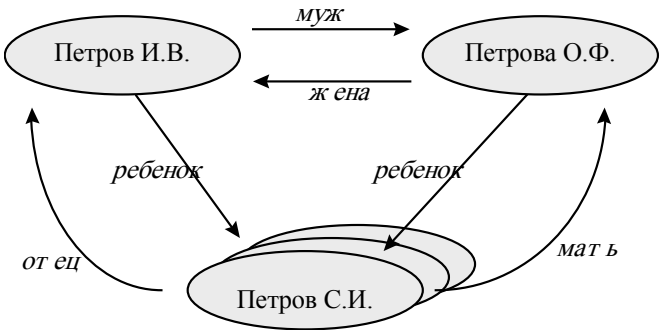


Рис.2.3. Представление фрагмента генеалогического дерева на основе сетевой модели данных

На данном рисунке представлены элементы только одно класса - описание людей, и на этом множестве для некоторых конкретных пар людей существуют связи, именуемые “муж”, “жена”, “отец”, “мать”, “ребенок”. Поэтому с точки зрения графического представления схемы этой базы данных (а не конкретных данных о семье Петровых), можно использовать следующий рисунок:

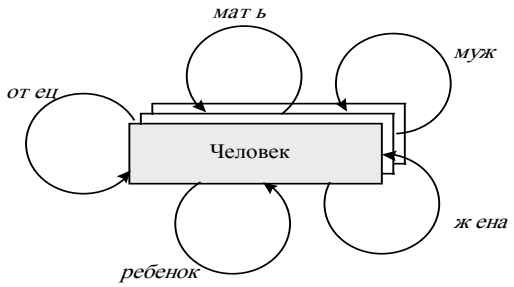


Рис.2.4. Представление схемы базы данных генеалогического дерева на основе сетевой модели данных

Сетевая модель данных основывается на понятии элемента данных и связей, задающих логику взаимоотношениями между данными. Связи от каждого элемента могут быть направлены на произвольное количество других элементов. На каждый элемент могут быть направлены связи от произвольного числа других элементов. Каждый элемент данных описывает некоторое понятие из предметной области и характеризуется некоторыми атрибутами. Для каждого элемента данных (элемент - это часть схемы) в реальной базе данных может существовать несколько экземпляров этого элемента. С каждым конкретным экземпляром по конкретной связи может быть связано разное число экземпляров другого элемента (например, у каждого человека разное число детей), но число видов связи одинаково для всех экземпляров одного элемента.

Если мы вернемся к нашему примеру про издательства тематических сборников (этот пример рассматривался в разделе про иерархические СУБД) и попытаемся расширить его, для того чтобы он более полно соответствовал реальным взаимоотношениям, то схема базы данных будет выглядеть следующим образом:

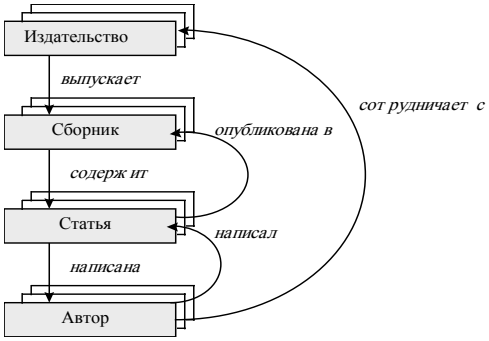


Рис.2.5. Представление расширенной схемы базы данных для описания издательств на основе сетевой модели

К достоинствам сетевой модели относится очень высокая скорость поиска и возможность адекватно представлять многие задачи в самых разных предметных областях. Высокая скорость поиска основывается на классическом способе физической реализации сетевой модели - на основе списков.

Главным недостатком сетевой модели, как, впрочем, и иерархической, является ее жесткость. Поиск данных, доступ к ним, возможен только по тем связям, которые реально существуют в данной конкретной модели. При поиске данных сетевая СУБД требует перемещаться только по существующим, заранее предусмотренным связям.

Реляционная структура данных

В конце 60-х годов появились работы, в которых обсуждались возможности применения различных табличных даталогических моделей данных, т.е. возможности использования привычных и естественных способов представления данных. Наиболее значительной из них была статья сотрудника фирмы IBM д-ра Э. Кодда (Codd E.F., A Relational Model of Data for Large Shared Data Banks. CACM 13: 6, June 1970), где, вероятно, впервые был применен термин "реляционная модель данных".

Будучи математиком по образованию, Э. Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как *отношение* – relation (англ.).

Наименьшая единица данных реляционной модели – это отдельное *атомарное* (неразложимое) для данной модели значение данных. Так, в одной предметной области фамилия, имя и отчество могут рассматриваться как единое значение, а в другой – как три различных значения.

Доменом называется множество атомарных значений одного и того же типа. Отношение на доменах D1, D2, ..., Dn (не обязательно, чтобы все они были различны) состоит из заголовка и тела. На рис. 2.6 приведен пример отношения для расписания движения самолетов.

Заголовок (интерпретация) состоит из такого фиксированного множества атрибутов A1, A2, ..., An, что существует взаимно однозначное соответствие между этими атрибутами Ai и определяющими их доменами Di (i=1,2,...,n).

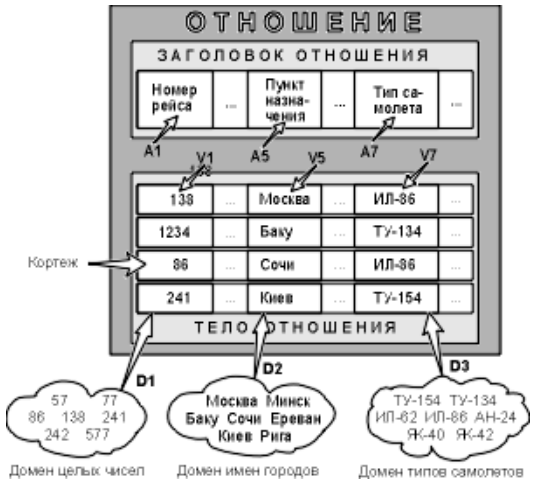


Рис. 2.6. Отношение с математической точки зрения (Ai - атрибуты, Vi - значения атрибутов)

Тело состоит из меняющегося во времени множества *кортежей*, где каждый кортеж состоит в свою очередь из множества пар атрибут-значение (Ai:Vi), (i=1,2,...,n), по одной такой паре для каждого атрибута Ai в заголовке. Для любой заданной пары атрибут-значение (Ai:Vi) Vi является значением из единственного домена Di, который связан с атрибутом Ai.

Степень отношения – это число его атрибутов. Отношение степени один называют унарным, степени два – бинарным, степени три – тернарным, ..., а степени n – n-арным.

Кардинальное число или *мощность отношения* – это число его кортежей. Кардинальное число отношения изменяется во времени в отличие от его степени.

Поскольку отношение – это множество, а множества по определению не содержат совпадающих элементов, то никакие два кортежа отношения не могут быть дубликатами друг друга в любой произвольно-заданный момент времени.

Пусть R – отношение с атрибутами A1, A2, ..., An. Говорят, что множество атрибутов K=(Ai, Aj, ..., Ak) отношения R является возможным ключом R тогда и только тогда, когда удовлетворяются два независимых от времени условия:

1. Уникальность: в произвольный заданный момент времени никакие два различных кортежа R не имеют одного и того же значения для Ai, Aj, ..., Ak.
2. Минимальность: ни один из атрибутов Ai, Aj, ..., Ak не может быть исключен из K без нарушения уникальности.

Каждое отношение обладает хотя бы одним возможным ключом, поскольку по меньшей мере комбинация всех его атрибутов удовлетворяет условию уникальности. Один из возможных ключей (выбранный произвольным образом) принимается за его первичный ключ. Остальные возможные ключи, если они есть, называются альтернативными ключами.

Вышеупомянутые и некоторые другие математические понятия явились теоретической базой для создания реляционных СУБД, разработки соответствующих языковых средств и программных систем, обеспечивающих их высокую производительность, и создания основ теории проектирования баз данных.

8 Реляционная модель данных (РМД). Основные определения. Интерпретация отношения в виде таблицы. Свойства табличного представления. Примеры.

8.1 Реляционная модель данных

Реляционная модель данных — логическая модель данных, прикладная теория построения баз данных, которая является приложением к задачам обработки данных таких разделов математики как теории множеств и логика первого порядка.
В системе исполняются, как минимум, три условия:

1. **Структурный аспект** (данные в базе воспринимаются пользователем только как таблицы)
2. **Аспект целостности** (таблицы отвечают определенным условиям целостности)
3. **Аспект обработки** (имеются операторы манипулирования таблицами, которые генерируют новые таблицы на основе уже имеющихся, минимум restrict, project, join)

Реляционная модель состоит из следующих компонентов (формально):

1. Неограниченный набор скалярных типов
2. Генератор типов отношений и соответствующая интерпретация для сгенерированных типов отношений
3. Возможность определения переменных отношения для указанных сгенерированных типов отношений
4. Операция реляционного присваивания для присваивания реляционных значений указанным переменным отношения
5. Неограниченный набор общих реляционных операторов (реляционная алгебра) для получений значений отношений из других значений отношений.

8.2 Интерпретация отношения как таблицы. Свойства.

В настоящее время в неформальном контексте термины **отношение** и **таблица** принято считать синонимами. Таблица ниже показывает примерные соответствия терминов:

таблица	отношение
запись (строка)	кортеж
поле (столбец)	атрибут

Для полного представления о реляционной модели данных важно правильно интерпретировать понятие отношения.

"Тело"отношения иногда называют его расширением, Это потому, что оно должно интерпретироваться как представление расширения какого-либо предиката, таким образом являясь частью множества, которое может быть сформировано путем замены свободных переменных этого предиката именами атрибутов.

Существует 1-к-1 связь между свободными переменными предиката и имен атрибутов. Каждый кортеж отношения содержит значения атрибутов, создающие предикат путем замены каждой из его свободных переменных. Результат есть утверждение, которое, по факту наличия кортежа в отношении, всегда будет истинным. Напротив, каждый кортеж, чье описание атрибутов совпадает с тем же у отношения, но содержимое ("тело") отсутствует в отношении, будет ложным. Это допущение редко выполняется в реальном мире: на практике в базах данных этот как правило означает, что истинность утверждения не определена (а не оно ложно). Например, отсутствие кортежа ("Джон "испанский") в таблице

знания языков не может быть однозначно принято как истинное утверждение, что Джон не говорит по-испански.

8.3 Альтернативы

Наиболее известными альтернативами реляционной модели являются иерархическая модель, и сетевая модель, рассмотренные в прошлых билетах.

9 Определение понятия отношения и его элементов. Ключ отношения, его свойства. Представление объектов и связей инфологической модели в РМД. Примеры.

9.1 Понятие

Каждое отношение имеет заголовок и тело; заголовок — это набор пар "имя-столбца: имя-типа а тело отношения состоит из набора строк, которые соответствуют заголовку.

Заголовок любого отношения можно рассматривать как предикат, а каждую строку в теле отношения как некоторое истинное высказывание, образованное в результате подстановки определенных значений фактических параметров соответствующего типа вместо формальных параметров этого предиката.

Другими словами, типы — это то (множество чего-то), что может стать предметом обсуждения, а отношения — это то (множество чего-то), что можно сказать об этом предмете. И типы, и отношения необходимы и достаточны для представления любых данных (на логическом уровне).

9.2 Ключ и его свойства

Потенциальный ключ — подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости).

Уникальность означает, что не существует двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают (равны).

Минимальность (несократимость) означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности. Иными словами, если из потенциального ключа убрать любой атрибут, он утратит свойство уникальности.

Поскольку все кортежи в отношении по определению уникальны, в нём всегда существует хотя бы один потенциальный ключ (например, включающий все атрибуты отношения). В отношении может быть одновременно несколько потенциальных ключей. Один из них может быть выбран в качестве первичного ключа отношения, тогда другие потенциальные ключи называют альтернативными ключами.

9.3 Инфологическая модель

Модель **"сущность-связь"** (англ. “Entity-Relationship model”), или ER-модель, является наиболее известным представителем класса семантических (концептуальных, инфологических) моделей предметной области. ER-модель обычно представляется в графической форме. Основные преимущества ER-моделей:

1. наглядность;
2. модели позволяют проектировать базы данных с большим количеством объектов и атрибутов;
3. ER-модели реализованы во многих системах автоматизированного проектирования баз данных

Основные элементы ER-моделей:

1. объекты (сущности);
2. атрибуты объектов;
3. связи между объектами.

Сущность - любой объект предметной области, имеющий атрибуты. Связь между сущностями характеризуется:

1. типом связи (1:1, 1:M, M:M);
2. классом принадлежности. Класс может быть обязательным и необязательным. Если каждый экземпляр сущности участвует в связи, то класс принадлежности – обязательный, иначе – необязательный.

10 Средства манипулирования данными (ЯМД), основанные на реляционной алгебре. Теоретико-множественные операции. Примеры.

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

10.1 Теоретико-множественные операции

В состав теоретико-множественных операций входят операции:

- объединения отношений;
- пересечения отношений;
- взятия разности отношений;
- взятия декартова произведения отношений.

10.2 Базовые операции ЯМД

Конкретный язык манипулирования реляционными БД называется реляционно-полным, если любой запрос, формулируемый с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления, может быть сформулирован с помощью одного оператора этого языка.

Известно и доказуемо, что механизмы реляционной алгебры и реляционного исчисления эквивалентны.

К базовым средствам манипулирования данными относятся "поисковые"варианты операторов UPDATE и DELETE. Эти варианты называются поисковыми, потому что при задании соответствующей операции задается логическое условие, налагаемое на строки адресуемой оператором таблицы, которые должны быть подвергнуты модификации или удалению. Кроме того, в такую категорию языковых средств входит оператор INSERT, позволяющий добавлять строки в существующие таблицы. Логично начать изложение именно с оператора INSERT, поскольку, для того чтобы можно было что-либо модифицировать в таблицах или удалять из таблиц, нужно, чтобы в таблицах содержались какие-то строки.

11 ЯМД, основанный на реляционной алгебре. Специальные операции реляционной алгебры. Полная система операций реляционной алгебры. Примеры.

11.1 Базовые механизмы

В манипуляционной составляющей реляционной модели данных определяются два базовых механизма манипулирования реляционными данными – основанная на теории множеств реляционная алгебра и базирующееся на математической логике (точнее, на исчислении предикатов первого порядка) реляционное исчисление. В свою очередь, обычно выделяются два вида реляционного исчисления – исчисление кортежей и исчисление доменов.

Все эти механизмы обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатом их "вычисления"также являются отношения. В результате любое выражение или формула могут интерпретироваться как отношения, что позволяет использовать их в других выражениях или формулах.

Конкретный язык манипулирования реляционными БД называется реляционно-полным, если любой запрос, формулируемый с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления, может быть сформулирован с помощью одного оператора этого языка.

Крайне редко алгебра или исчисление принимается в качестве полной основы какого-либо языка БД. Обычно (например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций.

11.2 Расширенная алгебра Кодда

Набор основных алгебраических операций состоит из восьми операций, которые делятся на два класса – теоретико-множественные операции и специальные реляционные операции. В состав теоретико-множественных операций входят операции:

- объединения отношений *UNION*;
- пересечения отношений *INTERSECT*;
- взятия разности отношений *MINUS*;
- взятия декартова произведения отношений *TIMES*.

Специальные реляционные операции включают:

- ограничение отношения *WHERE, RESTRICT*;
- проекцию отношения *PROJECT*;
- соединение отношений *JOIN*;
- деление отношений *DIVIDE BY*.

Кроме того, в состав алгебры включается операция **присваивания**, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и операция **переименования** *RENAME* атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

11.3 Приоритет операций

RENAME	4
WHERE	3
PROJECT	3
TIMES	2
JOIN	2
INTERSECT	2
DIVIDE BY	2
UNION	1
MINUS	1

.

12 Нормализация отношений, назначение и общая характеристика шагов нормализации. Понятие канонической схемы. Примеры.

12.1 Понятие нормализации

Говорят, что переменная отношения находится в определенной нормальной форме, если она удовлетворяет заданному набору условий. Процедуру нормализации можно охарактеризовать как последовательное приведение заданного набора переменных отношения к некоторой все более желательной форме. Следует отметить, что эта процедура обратима, т.е. всегда можно использовать ее результат (например, множество переменных отношения, находящихся в ЗНФ) для обратного преобразования (в исходную переменную отноше ния, находящуюся в 2НФ). Возможность выполнения обратного преобразования является весьма важной характеристикой, поскольку это означает, что в процессе нормализации сохраняется информация. Каждой нормальной форме соответствует определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером может служить ограничение первой нормальной формы – значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную избыточность. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в БД информации.

12.1.1 Свойства нормальных форм

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

В основе процесса проектирования лежит метод нормализации, т. е. декомпозиции отношения, находящегося в предыдущей нормальной форме, на два или более отношений, которые удовлетворяют требованиям следующей нормальной формы.

12.2 Шаги нормализации

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

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

12.3 Понятие канонической схемы

Каноническая схема - модель данных, с помощью которой представляются существующие в реальном мире структуры данных, вне зависимости от конкретных приложений или аппаратных средств, используемых для представления данных и их эксплуатации, являя собой простейшую форму представления из возможных. .

13 1-ая нормальная форма (1НФ) отношения. Определение. Метод приведения отношения к 1НФ.

13.1 Определение 1НФ

Таблица находится в первой нормальной форме, если каждый её атрибут атомарен, то есть может содержать только одно значение. Таким образом, не существует 1NF таблицы, в полях которых могут храниться списки значений.

Замечание. В реляционной модели отношение всегда находится в 1 (или более высокой) нормальной форме в том смысле, что иные отношения не рассматриваются в реляционной модели. То есть само определение понятия отношение заведомо подразумевает наличие 1NF.

Таблица удовлетворяет 1НФ, строго говоря, только при выполнении следующих условий:

1. Отсутствует установленный порядок следования строк сверху-вниз.
2. Отсутствует установленный порядок следования столбцов слева-направо.
3. Отсутствуют дублирующиеся строки.
4. Каждое пересечение строки и столба содержит строго одно значения из возможного домена (и ничего более).
5. Все поля стандартны (то есть, у них нет скрытых компонент, таких как номер строки, номер объекта и т.п.)

13.2 Атомарность атрибутов

Атрибут атомарен, если его значение теряет смысл при любом разбиении на части или переупорядочивании. И наоборот, если какой-либо способ разбиения на части не лишает атрибут смысла, то атрибут неатомарен. Одно и то же значение может быть атомарным или неатомарным в зависимости от смысла этого значения.

13.3 Метод приведения к 1НФ

Для приведения таблицы к 1NF обычно требуется разбить таблицу на несколько отдельных таблиц. .

14 Понятие функциональной зависимости (ФЗ) в отношениях. Свойства и аксиомы ФЗ. Примеры.

14.1 Определения

Функциональная зависимость — это связь типа "многие к одному" между двумя множествами атрибутов заданной переменной отношения (она представляет собой наиболее широко распространенный и важный вид ограничения целостности).

Определение (значение переменной отношения в определенный момент)**.** *Пусть R является отношением, а X и Y произвольными подмножествами множества атрибутов отношения r. Тогда Y функционально зависимо от x, что в символическом виде записывается как $X \to Y$ (X функционально определяет Y) тогда и только тогда, когда каждое значение множества x отношения r связано точно с одним значением множества Y отношения r. Иначе говоря, если два кортежа отношения R совпадают по значению X, они совпадают и по значению Y.*

Формально:

$$r\,(R)\,,\ A \subseteq R,\ B \subseteq R\,(A \to B) \Leftrightarrow ((\forall t_1, t_2 \in r : t_1\,(A) = t_2\,(A)) \Rightarrow (t_1\,(B) = t_2\,(B)))$$

Определение (множество всех значений, которые принимает переменная отношения)**.** *Пусть R является переменной отношения, а X и Y — произвольными подмножествами множества атрибутов переменной отношения R. Тогда Y функционально зависимо от X, (что в символическом виде записывается как $X \to Y$) тогда и только тогда, когда для любого допустимого значения переменной отношения R каждое значение множества X отношения R связано точно с одним значением множества Y отношения R. Иначе говоря, для любого допустимого значения переменной отношения R, если два кортежа переменной отношения R совпадают по значению X, они также совпадают и по значению Y.*

Отметим, что если X является потенциальным ключом переменной отношения R, то все атрибуты Y переменной отношения R должны обязательно быть функционально зависимыми от X.

Каждая переменная отношения обязательно удовлетворяет некоторым тривиальным функциональным зависимостям; причем функциональная зависимость тривиальна тогда и только тогда, когда ее правая (зависимая) часть является подмножеством ее левой части (детерминанта).

Левую и правую части символической записи функциональной зависимости иногда называют, соответственно, **детерминантом** и **зависимой частью**. Как говорится в определении, детерминант и зависимая часть являются множествами атрибутов. Когда множество содержит только один атрибут, оно называется одноэлементным множеством.

14.2 Замыкания множества зависимостей

Одни функциональные зависимости могут подразумевать другие функциональные зависимости. Например,

$$(A \to B) \wedge (B \to C) \Rightarrow (A \to C)$$

Множество S^+ всех функциональных зависимостей, которые подразумеваются данным множеством функциональных зависимостей S называется **замыканием** множества S.

14.3 Замыкания множества атрибутов

Пусть Z - некоторое множество атрибутов отношения r, а S - множество функциональных зависимостей этого отношения.

Замыканием Z^+ множества атрибутов Z в пределах S называется такое множество всех атрибутов A_i отношения r, что функциональная зависимость $Z \to A_i$ является членом замыкания S^+.

$$r\,(R)\,,\ S,\ Z \subseteq R,\ A_i \subseteq R,\ i = \overline{1,n}$$

$$Z^+ = \left\{ A_i : (Z \to A_i) \in S^+ \right\}$$

14.4 Аксиомы Армстронга

Пусть A, B, C, D - произвольные подмножества множества атрибутов заданной переменной отношения R.

1. Рефлексивность. Если $B \subset A$, то $A \to B$
2. Дополнение. Если $A \to B$, то $AC \to BC$
3. Транзитивность. Если $A \to B, B \to C$, то $A \to C$
4. Самоопределение. $A \to A$
5. Декомпозиция. Если $A \to BC$, то $A \to B, A \to C$
6. Объединение. Если $A \to B, A \to C$, то $A \to BC$
7. Композицияи. Если $A \to B, C \to D$, то $AC \to BD$
8. **Общая теорема объединения**. Если $A \to B, C \to D$, то $A \cup (C - B) \to BD$

Теорема (Хита)**.** *Пусть R{A, υ, C} является переменной отношения, где A, υ и C множества атрибутов этой переменной отношения. Если R удовлетворяет функциональной зависимости* $A \to C$ *то R равна соединению ее проекций по атрибутам {A,B} И {A,C}.*

.

15 2-ая нормальная форма (2НФ) отношения. Определение полной функциональной зависимости и 2НФ. Характеристика отношения во 2НФ. Алгоритм приведения ко 2НФ. Теорема Хита. Примеры.

15.1 Определение 2НФ

Таблица находится во второй нормальной форме, если она находится в первой нормальной форме, и при этом любой её атрибут, не входящий в состав возможного ключа, функционально полно зависит от каждого возможного ключа. Функционально полная зависимость означает, что атрибут функционально зависит от всего составного ключа, но при этом не находится в функциональной зависимости от какой-либо из входящих в него атрибутов (частей).

Другими словами: в 2NF нет неключевых атрибутов, зависящих от части составного ключа (плюс выполняются условия 1NF).

15.2 Теорема Хита

Теорема (Хита). *Пусть R{A, v, C} является переменной отношения, где A, v и C множества атрибутов этой переменной отношения. Если R удовлетворяет функциональной зависимости* $A \rightarrow C$ *то R равна соединению ее проекций по атрибутам {A,B} И {A,C}.*

15.3 Алгоритм приведения к 2НФ и 3НФ

Ниже приведен алгоритм, с помощью которого может быть выполнена декомпозиция без потерь произвольной переменной отношения R (с сохранением функциональных зависимостей) на множество D проекций, находящихся в 3НФ (2НФ получаем в процессе приведения). Предположим, что дано множество функциональных зависимостей S, удовлетворяемых в переменной отношения R. В таком случае декомпозиция может быть выполнена, как показано далее.

1. Инициализировать D значением пустого множества.
2. Пусть I является неприводимым покрытием для S.
3. Пусть x — множество атрибутов, присутствующих в левой части некоторой функ циональной зависимости $X \rightarrow Y$ из I.
4. Пусть полным множеством функциональных зависимостей из I с левой частью X является $X \rightarrow Y_1, X \rightarrow Y_2, ..., X \rightarrow Y_n$.
5. Пусть объединением $Y_1, Y_2, ..., Y_n$ является z.
6. Заменить множество D объединением множества D и проекции R по X и z.
7. Повторить шаги 4—6 для каждого отдельного X. Таким образом получена форма 2НФ.
8. Пусть $A_1, A_2, ..., A_n$ являются теми атрибутами R (если только они вообще имеются), которые все еще не охвачены этим алгоритмом (т.е. не включены ни в одну переменную отношения из D); заменить множество D объединением множе ства D и проекции R по $A_1, A_2, ..., A_n$.
9. Если ни одна переменная отношения из D не включает некоторый потенциальный ключ переменной отношения R, заменить D объединением D и проекции R по рас сматриваемому потенциальному ключу переменной отношения R.

.

16 3-я нормальная форма (3НФ) отношения. Определение транзитивной зависимости и 3НФ. Алгоритм приведения к 3НФ. Нормальная форма Бойса-Кодда (НФБК). Определение и алгоритм приведения к НФБК. Характеристика отношения в 3НФ и в НФБК. Примеры.

16.1 Определение 3НФ

Согласно определению Кодда, таблица находится в 3NF тогда и только тогда, когда выполняются следующие условия:

- Отношение R (таблица) находится во второй нормальной форме;
- Каждый непервичный атрибут R находится в нетранзитивной (то есть прямой) зависимости от каждого ключа R.

Непервичный (неключевой) атрибут R — это атрибут, который не принадлежит ни одному из возможных (альтернативных) ключей R.
Транзитивная зависимость — это функциональная зависимость, при которой $X \to Z$ (X определяет Z) не напрямую, а посредством отношения $X \to Y$ и $Y \to Z$ (отношение $Y \to X$ не является обязательным условием).

Определение 3NF, эквивалентное определению Кодда, но по-другому сформулированное, дал Карло Заниоло. Согласно ему, таблица находится в 3НФ тогда и только тогда, когда для каждой из ее функциональных зависимостей $X \to A$ выполняется хотя бы одно из следующих условий:

- X содержит A (то есть $X \to A$ — тривиальная функциональная зависимость)
- X — суперключ
- A — первичный атрибут (то есть A входит в состав альтернативного ключа).

Определение Заниоло четко определяет разницу между 3NF и более строгой нормальной формой Бойса-Кодда (НФБК): НФБК исключает третье условие (A — первичный атрибут).

16.2 Алгоритм приведения к 3НФ

Ниже приведен алгоритм, с помощью которого может быть выполнена декомпозиция без потерь произвольной переменной отношения R (с сохранением функциональных зависимостей) на множество D проекций, находящихся в 3НФ (2НФ получаем в процессе приведения). Предположим, что дано множество функциональных зависимостей S, удовлетворяемых в переменной отношения R. В таком случае декомпозиция может быть выполнена, как показано далее.

1. Инициализировать D значением пустого множества.
2. Пусть I является неприводимым покрытием для S.
3. Пусть x — множество атрибутов, присутствующих в левой части некоторой функ циональной зависимости $X \to Y$ из I.
4. Пусть полным множеством функциональных зависимостей из I с левой частью X является $X \to Y_1, X \to Y_2, ..., X \to Y_n$.
5. Пусть объединением $Y_1, Y_2, ..., Y_n$ является z.
6. Заменить множество D объединением множества D и проекции R по X и z.
7. Повторить шаги 4—6 для каждого отдельного X. Таким образом получена форма 2НФ.
8. Пусть $A_1, A_2, ..., A_n$ являются теми атрибутами R (если только они вообще имеются), которые все еще не охвачены этим алгоритмом (т.е. не включены ни в одну переменную отношения из D); заменить множество D объединением множе ства D и проекции R по $A_1, A_2, ..., A_n$.

9. Если ни одна переменная отношения из D не включает некоторый потенциальный ключ переменной отношения R, заменить D объединением D и проекции R по рас сматриваемому потенциальному ключу переменной отношения R.

16.3 Нормальная форма Бойса-Кодда

Отношение находится в BCNF тогда и только тогда, когда каждая его нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ. Менее формально, переменная отношения находится в нормальной форме Бойса-Кодда тогда и только тогда, когда детерминанты всех ее функциональных зависимостей являются потенциальными ключами.

Определение. *Пусть R является переменной отношения, а X и Y — произвольными подмножествами множества атрибутов переменной отношения R. Y функционально зависимо от X тогда и только тогда, когда для любого допустимого значения переменной отношения R, если два кортежа переменной отношения R совпадают по значению X, они также совпадают и по значению Y. Подмножество X называют детерминантом, а Y — зависимой частью.*

Функциональная зависимость тривиальна тогда и только тогда, когда ее правая (зависимая) часть является подмножеством ее левой части (детерминанта). Ситуация, когда отношение будет находиться в 3NF, но не в BCNF, возникает, например, при условии, что отношение имеет два (или более) потенциальных ключа, которые являются составными и имеют общий атрибут. На практике такая ситуация встречается достаточно редко, для всех прочих отношений 3NF и BCNF эквивалентны.

16.4 Алгоритм приведения к НФБК

Алгоритм, состоящий из четырех шагов, с помощью которого произвольная переменная отношения R может быть подвергнута декомпозиции без потерь на множество D проекций НФБК (но при этом не обязательно сохраняются все зависимости).

1. Инициализировать множество D так, чтобы в нем содержалась только переменная отношения R.
2. Для каждой переменной отношения τ из множества D, не находящейся в НФБК, выполнить шаги 3 и 4.
3. Пусть $X \to Y$ является функциональной зависимостью для τ, которая нарушает требования НФБК.
4. Заменить переменную отношения τ из множества D двумя ее проекциями: по атрибутам X и Y и по всем атрибутам, кроме тех, что находятся в Y.

16.5 Пример 3НФ и НФБК

Предположим, создаётся таблица бронирования для теннисных кортов на день: *(Номер корта, Время начала, Время окончания, Тариф, Член клуба)*. Тариф зависит от выбранного корта и членства в клубе. Таким образом, возможны следующие составные первичные ключи: *(Номер корта, Время начала), (Номер корта, Время окончания), (Тариф, Время начала), (Тариф, Время окончания)*.

Таблица соответствует второй и третьей нормальной форме, так как атрибуты, не входящие в состав первичного ключа, зависят от составного первичного ключа целиком (2NF) и нет транзитивных зависимостей (3NF).

Тем не менее, существует функциональная зависимость тарифа от номера корта. То есть, по ошибке можно нарушить логическую целостность и, например, приписать тариф Premium для первого корта, хотя тариф Premium может относиться только ко второму корту.
Можно улучшить структуру, разбив таблицу на две: *(Номер корта, Время начала, Время окончания, Член клуба)* и *(Тариф, Номер корта, Член клуба)*. Данное отношение будет соответствовать НФБК.

17 Многозначные зависимости (МЗ). Определение. Свойства и аксиомы МЗ. Четвертая нормальная форма (4НФ) отношения. Характеристика отношения в 4НФ.

17.1 Определение многозначных зависимостей

Определение. *Пусть R - переменная отношения, а A, B и C являются произвольными подмножествами множества атрибутов переменной отношения R. Тогда подмножество в многозначно зависит от подмножества A, что символически выражается следующей: $\to\to$ (читается как "A многозначно определяет B"), тогда и только тогда, когда в каждом допустимом значении R множество значений B, соответствующее заданной паре значений A, C, зависит только от значения A и не зависит от значения C.*

Многозначная зависимость $A \to\to B$ называется **тривиальной**, если выполняется хотя бы одно из условий:

1. Множество A является надмножеством B;
2. $B \subseteq A$
3. Объединение A и B образует весь заголовок отношения.
4. $A \cup B = R$

17.2 Свойства и аксиомы МЗ

17.2.1 Связные пары

Феджин показал, что многозначные зависимости образуют связные пары (в обозначениях определения):

$$(A \to\to B) \Leftrightarrow (A \to\to C).$$

Поэтому их часто представляют вместе в символической записи:

$$A \to\to B|C$$

17.2.2 Функциональные зависимости

Всякая функциональная зависимость является многозначной. Другими словами, функциональная зависимость - это многозначная зависимость, в которой множество зависимых значений, соответствующее заданному значению детерминанта, всегда имеет единичную мощность.

$$(A \to B) \Rightarrow (A \to\to B)$$

17.2.3 Аксиомы

Пусть у нас есть отношение $r(R)$ и множества атрибутов $A, B, C, D \subseteq R$. Для сокращения записи вместо $X \cup Y$ будем писать просто XY.

Группа 1: базовые правила.

1. Дополнение
2. $(A \cup B \cup C = R) \wedge (B \cap C \subseteq A) \Rightarrow ((A \to\to B) \Leftrightarrow (A \to\to C))$
3. Транзитивность
4. $(A \to\to B) \wedge (B \to\to C) \Rightarrow (A \to\to C \backslash B)$
5. Рефлексивность
6. $(B \subseteq A) \Rightarrow (A \to\to B)$
7. Приращение

8. $(A \to\to B) \wedge (C \subseteq D) \Rightarrow (AD \to\to BC)$

Группа 2: выводятся несколько дополнительных правил, упрощающих задачу вывода многозначных зависимостей.

1. Псевдотранзитивность
2. $(A \to\to B) \wedge (BC \to\to D) \Rightarrow (AC \to\to D \backslash BC)$
3. Объединение
4. $(A \to\to B) \wedge (A \to\to C) \Rightarrow (A \to\to BC)$
5. Декомпозиция
6. $(A \to\to BC) \Rightarrow (A \to\to B \cap C) \wedge (A \to\to B \backslash C) \wedge (A \to\to C \backslash B)$

Группа 3: устанавливается связь между функциональными и многозначными зависимостями.

1. Репликация (копирование)
2. $(A \to B) \Rightarrow (A \to\to B)$
3. Слияние
4. $(A \to\to B) \wedge (C \to D) \wedge (D \subseteq B) \wedge (B \cap C = nothing) \Rightarrow (A \to D)$

Группа 4: для функциональных зависимостей, выводятся из вышеприведенных правил. $(A \to\to B) \wedge (AB \to C) \Rightarrow (A \to C \backslash B)$

Правила вывода Армстронга вместе с изложенными здесь правилами групп 1 и 3 образуют полный (используя их, можно вывести все остальные многозначные зависимости, подразумеваемые данным их множеством) и надежный («лишних» многозначных зависимостей вывести нельзя; выведенная многозначная зависимость справедлива везде, где справедливо то множество многозначных зависимостей, из которого она была выведена) набор правил вывода многозначных зависимостей.

Теорема (Феджина)**.** *Пусть дано отношение $r\,(A, B, C)$. Отношение r будет равно соединению его проекций $r\,[A, B]$ и $r\,[A, C]$ тогда и только тогда, когда для отношения r выполняется нетривиальная многозначная зависимость $A \to\to B|C$.*

$$(r\,(A, B, C) = r\,[A, B] \;\; JOIN \; r\,[A, C]) \Leftrightarrow (A \to\to B|C)$$

17.3 Четвертая нормальная форма отношения

Определение. *Отношение находится в 4NF, если оно находится в НФБК и не содержит нетривиальных многозначных зависимостей. То есть все многозначные зависимости являются, по сути, функциональными зависимостями от ключей отношения.*

17.4 Пример

Предположим, что рестораны производят разные виды пиццы, а службы доставки ресторанов работают только в определенных районах города. Составной ключ таблицы такого отношения включает три поля: {Ресторан, Вид пиццы, Район доставки}.

Такая таблица не соответствует 4NF, так как существует многозначная зависимость:
{Ресторан} →→ {Вид пиццы}
{Ресторан} →→ {Район доставки}

То есть, например, при добавлении нового вида пиццы придется внести по одной новой записи для каждого района доставки. Возможна логическая аномалия, при которой определенному виду пиццы будут соответствовать лишь некоторые районы доставки из обслуживаемых рестораном районов.

Для предотвращения аномалии нужно разбить многозначную зависимость — разместить независимые факты в разных таблицах. В данном примере — {Ресторан, Вид пиццы} и {Ресторан, Район доставки}.

18 18. Общая характеристика языка SQL. Стандарты SQL, способы его реализации. Структура языка SQL.

18.1 Введение

SQL - универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. SQL основывается на реляционной алгебре.

SQL является, прежде всего, информационно-логическим языком, предназначенным для описания хранимых данных, для извлечения хранимых данных и для модификации данных. SQL не является языком программирования.
Язык SQL представляет собой совокупность операторов.

18.2 Стандарты SQL

Первый официальный стандарт языка SQL был принят ANSI в 1986 году и ISO (Международной организацией по стандартизации) в 1987 году (так называемый SQL-86) и несколько уточнён в 1989 году. Дальнейшее развитие языка поставщиками СУБД потребовало принятия в 1992 году нового расширенного стандарта (ANSI SQL-92 или просто SQL2). Следующим стандартом стал SQL:1999 (SQL3). В настоящее время действует стандарт, принятый в 2003 году (SQL:2003) с небольшими модификациями, внесёнными позже.

18.3 Структура SQL (операторы)

Операторы определения данных (Data Definition Language, DDL)

1. **CREATE** создает объект БД (саму базу, таблицу, представление, пользователя и т.д.)
2. **ALTER** изменяет объект
3. **DROP** удаляет объект

Операторы манипуляции данными (Data Manipulation Language, DML)

1. **SELECT** считывает данные, удовлетворяющие заданным условиям
2. **INSERT** добавляет новые данные
3. **UPDATE** изменяет данные
4. **DELETE** удаляет данные

Операторы определения доступа к данным (Data Control Language, DCL)

1. **GRANT** предоставляет пользователю (группе) разрешения на определенные операции с объектом
2. **REVOKE** отзывает ранее выданные разрешения
3. **DENY** задает запрет, имеющий приоритет над разрешением

Операторы управления транзакциями (Transaction Control Language, TCL)

1. **COMMIT** применяет транзакцию.
2. **ROLLBACK** откатывает все изменения, сделанные в контексте текущей транзакции.
3. **SAVEPOINT** делит транзакцию на более мелкие участки.

18.4 Преимущества и недостатки

Преимущества:

1. Независимость от конкретной СУБД
2. Наличие стандартов
3. Декларативность

Недостатки:

1. Несоответствие реляционной модели данных
2. Сложность
3. Отступления от стандартов
4. Сложность работы с иерархическими структурами

18.5 Уровни соответствия стандарту

На текущий момент все усилия по проверке СУБД на соответствие стандарту ложатся на её производителя.

До 1996 года вопросами соответствия коммерческих реализаций SQL стандарту занимался в основном Национальный институт стандартов и технологий (NIST), который и устанавливал уровень соответствия стандарту.

Впервые понятие «уровня соответствия» было предложено в стандарте SQL-92. А именно, ANSI и NIST определяли четыре уровня соответствия реализации этому стандарту:

1. Entry (базовый)
2. Transitional (переходный)
3. Intermediate (промежуточный)
4. Full (полный)

Легко можно понять, что каждый последующий уровень соответствия заведомо подразумевал соответствие предыдущему уровню. Далее, согласно данной лесенке стандартов любая СУБД, которая соответствовала уровню Entry, могла заявлять себя как “SQL-92 compliant”, хотя на самом деле переносимость и соответствие стандарту ограничивалось набором возможностей, входящих в этот уровень.

Положение изменилось с введением стандарта SQL:1999. Отныне стандарт приобрёл модульную структуру — основная часть стандарта была вынесена в раздел SQL/Foundation, все остальные были выведены в отдельные модули. Соответственно, остался только один уровень совместимости — Core, что означало поддержку этой основной части. Поддержка остальных возможностей оставлена на усмотрение производителей СУБД. Аналогичное положение имело место и с последующими версиями стандарта.

19. Операторы ЯМД в T-SQL: состав и назначение. Примеры.

Язык манипулирования (управления) данными (ЯМД) или Data Manipulation Language (DML) — это семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных.

Состав и назначение

INSERT — осуществляет вставку строк в таблицу

Синтаксис

```
INSERT
[INTO]
[(column_list)]
[ <OUTPUT Clause> ]
{VALUES ( { DEFAULT | NULL | expression } [ ,...n ] ) [ ,...n ]}
```

Пример

```
INSERT INTO Table
VALUES (35, ‘Ivanov’, ‘Ivan’, ‘M’);
```

DELETE — осуществляет удаление строк из таблицы

Синтаксис

```
DELETE
[ TOP ( expression ) [ PERCENT ] ]
[ FROM ]
[ <OUTPUT Clause> ]
[ FROM <table_source> [ ,...n ] ]
[ WHERE { <search_condition> }]
```

Пример

```
DELETE FROM Table
WHERE Name != ‘Ivan’;
```

UPDATE — осуществляет изменение данных в таблице

Синтаксис

```
UPDATE { table_alias }
SET
{ column_name= { expression | DEFAULT | NULL }}
[ <OUTPUT Clause> ]
[ FROM { <table_source> } [ ,...n ] ]
[ WHERE { <search_condition> ]
```

Пример

```
UPDATE Table
SET Name = ‘Ivan’
WHERE Surname LIKE ‘Ivanov’
```

SELECT — осуществляет выборку данных из таблиц по запросу.

Синтаксис

```
<SELECT statement> ::=
<query_expression>
[ ORDER BY { order_by_expression | column_position [ ASC | DESC ] } [ ,...n ] ]
[ COMPUTE { { AVG | COUNT | MAX | MIN | SUM } (expression )} [ ,...n ]
[ BY expression [ ,...n ] ]
]

<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [  { UNION [ ALL ] | EXCEPT | INTERSECT }
        <query_specification> | ( <query_expression> ) [...n ] ]
```

```
<query_specification> ::=
SELECT [ ALL | DISTINCT ]
< select_list >
[ INTO new_table ]
[ FROM { <table_source> } [ ,...n ] ]
[ WHERE <search_condition> ]
[ <GROUP BY> ]
[ HAVING < search_condition > ]
```

Пример

```
SELECT fio, uch_zavedenie, pol, family_status, spec, kat_obucheniya, mp, gp, mo, gok
FROM uzb
JOIN vuz ON vuz_k = vuz.cod
JOIN family ON sem_polog_k = family.cod
JOIN kat_obuch ON kat_obuch_k = kat_obuch.cod
WHERE kat_obuch_k IN
(
SELECT cod
FROM kat_obuch
WHERE kat_obucheniya= 'УЧАЩИЙСЯ'
)
AND data_rogden like '%66'
AND spec in ('000601','000701','001001','001501','001801')
AND gp in ('88', '90')
ORDER BY gp, spec, uch_zavedenie
```

20. Способы определения правил целостности БД в T-SQL. Задание правил целостности на уровне домена и таблицы.

Самым простым способом поддержки целостности данных является создание правил или контрольных ограничений, причем двух типов:
- на уровне поля
- на уровне таблицы

Ограничители это элементарные проверки или условия, которые выполняются для операций вставки и модификации значения столбца. Если данная проверка не проходит или условие не выполняется, то вставка или модификация отменяется, а в программу клиента передается ошибка. SQL-серверы, как правило, поддерживают следующие ограничители.

- **NOT NULL** - проверка на непустое значение. NULL - специальное понятие в СУБД, которое означает "пусто".
- **UNIQUE** - проверка на уникальность.
- **PRIMARY KEY** - первичный ключ. Значение в столбце считается первичным ключом, если оно непустое и уникально в пределах столбца данной таблицы.

SQL-технология позволяет на уровне столбца задавать домены значений, т.е. строго определенные наборы или диапазоны значений, для помещаемых в столбец данных. В частности можно реализовывать ограничения ссылочной целостности и проверки фиксированного условия. Ограничение ссылочной целостности не позволяет значениям из столбца одной таблицы принимать значения кроме как из присутствующих в столбце другой таблицы. Это делается при помощи ограничителей FOREIGN KEY (внешний ключ) и REFERENCES (указатель ссылки). Таблица, содержащая FOREIGN KEY, считается родительской таблицей. Таблица, содержащая REFERENCES, считается дочерней таблицей. Внешний ключ и указатель ссылки могут находиться в одной таблице, т.е. родительская таблица одновременно является дочерней.

- **FOREIGN KEY** - внешний ключ. Назначает столбец или комбинацию столбцов в текущей (родительской) таблице в качестве внешнего ключа для ссылки из других таблиц.
- **REFERENCES** - указатель ссылки (или родительский ключ). Указывает на столбец (комбинацию столбцов) в родительской таблице, ограничивающую значения в текущей (дочерней) таблице.

CHECK - проверка фиксированного условия. В данном ограничителе явно указывается условие, которое должно выполняться для вставляемого или модифицируемого значения в столбце. Например: check (user in 'ALEX','JUSTAS') - в столбце user могут содержаться только значения 'ALEX' и 'JUSTAS', попытка вставки значения 'SHTIRLITZ' будет интерпретирована как ошибочная

К одному столбцу можно применять несколько проверочных ограничений. Кроме того, можно применять одно проверочное ограничение к нескольким столбцам. Для этого ограничение нужно создать на уровне таблицы. Например, с помощью проверочного ограничения на несколько столбцов можно подтвердить то, что любая строка со значением **USA** в столбце **country/region** может принимать двухсимвольное значение в столбце **state**. Это позволяет выполнить проверку сразу нескольких условий из одного выражения.

Пример

```
create table poss2
--ограничения на уровне поля, то есть для каждого конкретного столбца в лоб задаем ограничения
 (nomer integer check (nomer between 1 and 700000),
fio char (40) not null check (fio not like '%.%' or fio not like '%-%'),
pol char(1) check (pol like 'М' or pol like 'Ж'),
kat_obuch_k char(2) check (kat_obuch_k between '01' and '17'),
vuz_k integer check (vuz_k between 128955 and 8199999),
gp char(2) check (gp between '00' and '99'),
gok char(2) check (gok between '00' and '99'),
        --ограничения на уровне таблицы, мы тут уже используем не один столбец, а несколько
        constraint kat_obuch_const check
                ((kat_obuch_k in ('01', '02') and
                (convert (integer, gok) - convert (integer, gp))=1) or
                (kat_obuch_k='08' and (convert(integer, gok) -
                convert (integer, gp))=4) or
                (kat_obuch_k in('10', '11') and
                (convert (integer, gok) - convert (integer, gp)) in (1, 2))))
```

21. T-SQL. Хранимые процедуры и их назначение. Типы хранимых процедур. Операторы создания, запуска, изменения и удаления хранимых процедур. Пример хранимой процедуры.

Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде. Выполнение в базе данных хранимых процедур вместо отдельных операторов SQL дает пользователю следующие **преимущества**:

- **Хранимые процедуры регистрируются на сервере.**
- **Хранимые процедуры могут иметь атрибуты безопасности** (например, разрешения) и цепочки владения, кроме того, к ним можно прикреплять сертификаты.
  Пользователи могут обладать разрешениями на выполнение хранимых процедур вместо прямых разрешений для работы с объектами, на которые ссылаются эти процедуры.
- **Хранимые процедуры позволяют сделать защиту приложений более надежной**.
  Параметризованные хранимые процедуры могут защитить приложения от атак, осуществляемых путем инжекции кода SQL.
- **Хранимые процедуры поддерживают модульное программирование**.
  Процедуру можно создать один раз и потом по мере необходимости вызывать ее в программе любое число раз. Это делает обслуживание приложения более удобным и позволяет унифицировать доступ приложений к базе данных.
- **Хранимые процедуры представляют собой именованный код, дающий возможность отсроченного связывания.**
  Это обеспечивает уровень косвенности, упрощающий развитие программного кода.
- **Хранимые процедуры позволяют уменьшить сетевой трафик**.
  Операцию, занимающую сотни строк программного кода Transact-SQL, можно выполнить в одной инструкции, которая обрабатывает процедуру, а не отправляет этот код по сети.

Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных. Они вызываются клиентской программой, другой хранимой процедурой или триггером. Разработчик может управлять правами доступа к хранимой процедуре, разрешая или запрещая ее выполнение. Изменять код хранимой процедуры разрешается только ее владельцу или члену фиксированной роли базы данных. При необходимости можно передать права владения ею от одного пользователя к другому.

Типы хранимых процедур

В SQL Server имеется несколько типов хранимых процедур.

- **Системные хранимые** процедуры предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Системные хранимые процедуры имеют префикс sp_, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.
- **Пользовательские хранимые процедуры** реализуют те или иные действия. Хранимые процедуры – полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.
- **Временные хранимые процедуры** существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные. Локальные временные хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя, перезапуске или остановке сервера. Глобальные временные хранимые процедуры доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

Создание, изменение и удаление хранимых процедур

Создание хранимой процедуры предполагает решение следующих задач:

- определение типа создаваемой хранимой процедуры: временная или пользовательская. Кроме этого, можно создать свою собственную системную хранимую процедуру, назначив ей имя с префиксом sp_ и поместив ее в системную базу данных.

- планирование прав доступа. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь;
- определение параметров хранимой процедуры. Подобно процедурам, входящим в состав большинства языков программирования, хранимые процедуры могут обладать входными и выходными параметрами;
- разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:
[{CREATE | ALTER } PROCEDURE] имя_процедуры
[{@имя_параметра тип_данных }
AS { [BEGIN] sql_statement [;] [...n] [END] }

Удаление хранимой процедуры осуществляется командой:
DROP PROCEDURE {имя_процедуры} [,...n]

Для **выполнения хранимой процедуры** используется команда:
[[EXEC UTE] имя_процедуры
[[@имя_параметра=]{значение | @имя_переменной}

Для **обращения к процедуре** можно использовать команды:
EXEC my_proc1 или my_proc1

Пример Процедура без параметров. Создать процедуру для уменьшения цены товара первого сорта на 10%.
CREATE PROC my_proc2
AS
UPDATE Товар SET Цена=Цена*0.9
WHERE Сорт='первый'

22. T-SQL. Курсоры: назначение, описание, применение. Пример

Курсор — ссылка на контекстную область памяти. В некоторых реализациях информационно-логического языка SQL— получаемый при выполнении запроса результирующий набор и связанный с ним указатель текущей записи.

Методика использования курсора языка Transact-SQL в хранимой процедуре или триггере такова:

1. Воспользуйтесь инструкцией DECLARE CURSOR, чтобы связать курсор языка Transact-SQL с инструкцией SELECT. Инструкция DECLARE CURSOR определяет также характеристики курсора, например имя курсора и тип курсора (read-only или forward-only).
2. Воспользуйтесь инструкцией OPEN для выполнения инструкции SELECT и заполнения курсора.
3. Воспользуйтесь инструкцией FETCH INTO для выборки отдельных строк и перемещения данных для каждого столбца в указанную переменную. После этого другие инструкции языка Transact-SQL могут ссылаться на эти переменные для доступа к выбранным значениям данных. Курсоры языка Transact-SQL не поддерживают выборку группы строк.
4. После завершения работы с курсором примените инструкцию CLOSE. Закрытие курсора освобождает некоторые ресурсы, например результирующий набор курсора и его блокировки на текущей строке, однако структура курсора будет доступна для обработки, если снова выполнить инструкцию OPEN. Поскольку курсор все еще существует на этом этапе, повторно использовать его имя не удастся.
5. Инструкция DEALLOCATE полностью освобождает все ресурсы, выделенные курсору, в том числе имя курсора. После освобождения курсора его необходимо перестроить заново с помощью инструкции DECLARE.

Пример:

```
--Объявляем переменные для хранения фамилии и имени
DECLARE @LastName varchar(50), @FirstName varchar(50);
--Объявляем курсор на фамилию имя
DECLARE contact_cursor CURSOR FOR
SELECT LastName, FirstName FROM Person.Person
WHERE LastName LIKE 'B%'
ORDER BY LastName, FirstName;
--Открываем курсор
OPEN contact_cursor;
-- Делаем fetch, по сути шаг, выбор следующего FETCH NEXT. Запоминаем данные в переменные
-- Замечание: Переменные в таком же порядке, как столбцы в SELECT
FETCH NEXT FROM contact_cursor
INTO @LastName, @FirstName;
-- Проверяем @@FETCH_STATUS есть ли еще строки.
WHILE @@FETCH_STATUS = 0
BEGIN
   -- Печатаем содержимое переменных.
   PRINT 'Contact Name: ' + @FirstName + ' ' +  @LastName
   -- Сдвигаем курсор дальше, запоминаем в переменные
   FETCH NEXT FROM contact_cursor
   INTO @LastName, @FirstName;
END
--Закрываем курсор
CLOSE contact_cursor;
--Удаляем курсор
DEALLOCATE contact
```

23. T-SQL. Триггеры и их назначение. Типы триггеров. Операторы создания, изменения, включения/отключения, удаления триггеров. Ограничения использования триггеров. Примеры.

Триггер (англ. trigger) — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено наступлением определенного события (действием)
Основное назначение триггеров состоит в автоматическом использовании их в качестве реакции на некоторые события, происходящие с таблицами, с которыми связанны триггеры. Это свойство триггеров позволяет использовать их для реализации сложных форм ограничений целостности данных. Кроме того, рассматриваемое свойство превращает сервер из пассивного наблюдателя за происходящими изменениями данных в систему, оперативно реагирующую на такие изменения. Правила, в соответствие с которыми осуществляются активные действия сервера, определяются триггерами

Типы триггеров

SQL Server поддерживает три основных **типа триггеров**: **триггеры DML, триггеры DDL и триггеры входа**.

- **Триггеры DDL** выполняются, когда на сервере или в базе данных возникает событие языка определения данных (DDL). Эти события в основном соответствуют инструкциям языка Transact-SQL, начинающимся ключевыми словами CREATE, ALTER или DROP. Системные хранимые процедуры, выполняющие операции, подобные операциям DDL, также могут запускать триггеры DDL.
  Пример
  В следующем примере показано, как триггер DDL может быть использован для предотвращения изменения или удаления любой таблицы базы данных.

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
  PRINT 'You must disable Trigger "safety" to drop or alter tables!'
  ROLLBACK ;
```

- **Триггеры входа** вызывают срабатывание хранимых процедур в ответ на событие LOGON. Это событие вызывается при установке пользовательского сеанса с экземпляром SQL Server. Можно использовать триггеры входа для проверки и управления сеансами сервера, например для отслеживания входов в систему, ограничения входов в SQL Server или ограничения числа сеансов для конкретного имени входа
- **DML-триггеры** выполняются при возникновении событий языка обработки данных (DDL) в базе данных. DML-события возникают при выполнении инструкций INSERT, UPDATE или DELETE, изменяющих данные в указанной таблице или представлении. DML-триггеры могут обращаться к другим таблицам и содержать сложные инструкции Transact-SQL. Триггер и инструкция, при выполнении которой он срабатывает, считаются одной транзакцией, которую можно откатить назад внутри триггера. При обнаружении серьезной ошибки (например, нехватки пространства на диске) вся транзакция автоматически откатывается назад.
  Пример

```
CREATE TRIGGER reminder1
ON Sales.Customer
AFTER INSERT, UPDATE
AS RAISERROR ('Notify Customer Relations', 16, 10);
```

  Чтобы управлять срабатыванием DML-триггера, можно указать один из двух параметров.
  - **Триггеры AFTER** срабатывают после обработки действия, вызывающего срабатывание (INSERT, UPDATE или DELETE), триггеров INSTEAD OF и ограничений. Триггеры AFTER можно устанавливать путем указания ключевых слов AFTER или FOR. Так как эффект от ключевого слова FOR тот же, что и от AFTER, DML-триггеры с ключевым словом FOR также относятся к триггерам AFTER.
  - **Триггеры INSTEAD OF** срабатывают вместо действия, вызывающего срабатывание, и перед обработкой ограничений. Если в таблице имеются триггеры AFTER, они сработают после обработки ограничений. В случае нарушения ограничений выполняется откат действий триггеров INSTEAD OF, а триггер AFTER не срабатывает.

В каждой таблице или представлении может быть один триггер INSTEAD OF для каждого из действий, вызывающих срабатывание (UPDATE, DELETE или INSERT). Таблица может содержать несколько триггеров AFTER для каждого из вызывающих срабатывание действий.

Операторы создания, изменения

DML Trigger

```
[CREATE|ALTER] TRIGGER trigger_name
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement }
```

DDL Trigger

```
[CREATE|ALTER] TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement }
```

Logon Trigger

```
[CREATE|ALTER] TRIGGER trigger_name
ON ALL SERVER
{ FOR| AFTER } LOGON
AS { sql_statement }
```

Операторы удаления

DML Trigger

```
DROP TRIGGER [schema_name.] trigger_name [ ,...n ] [ ; ]
```

DDL Trigger

```
DROP TRIGGER trigger_name [ ,...n ]
ON { DATABASE | ALL SERVER }
```

Logon Trigger

```
DROP TRIGGER trigger_name [ ,...n ]
ON ALL SERVER
```

Операторы включения, выключения

```
[ENABLE|DISABLE] TRIGGER { trigger_name [ ,...n ] | ALL }
ON { object_name | DATABASE | ALL SERVER }
```

24. T-SQL. Ссылочная целостность. Правила ссылочной целостности и поддержка их с помощью триггеров. Примеры.

Ссылочная целостность. База данных не должна содержать каких-либо несогласованных значений внешнего ключа. В этом определении термин "несогласованное значение внешнего ключа" обозначает значение внешнего ключа в некоторой ссылающейся переменной отношения, для которого не существует согласованного значения соответствующего потенциального ключа в соответствующей переменной отношения, указанной в ссылке. Иными словами, это ограничение можно сформулировать просто как следующее требование: "Если значение B ссылается на A, то A должно существовать".
Ссылочная целостность сохраняет определенные связи между таблицами при добавлении или удалении строк. В SQL Server ссылочная целостность основана на связи первичных и внешних ключей (либо внешних и уникальных ключей) и обеспечивается с помощью ограничений FOREIGN KEY и CHECK. Ссылочная целостность гарантирует согласованность значений ключей во всех таблицах. Этот вид целостности требует отсутствия ссылок на несуществующие значения, а также обеспечивает согласованное изменение ссылок во всей базе данных при изменении значения ключа.
При обеспечении ссылочной целостности SQL Server не допускает следующих действий пользователей.

- Добавления или изменения строк в связанной таблице, если в первичной таблице нет соответствующей строки.
- Изменения значений в первичной таблице, которое приводит к появлению потерянных строк в связанной таблице.
- Удаления строк из первичной таблицы, если имеются соответствующие ей строки в связанных таблицах.

Ограничение FOREIGN KEY предотвращает возникновение ситуации несогласованности. Ограничение обеспечивает целостность ссылок следующим образом: оно запрещает изменение данных в таблице первичного ключа, если такие изменения сделают недопустимой ссылку в таблице внешнего ключа. Если при попытке удалить строку в таблице первичного ключа или изменить значение этого ключа окажется, что удаленному или измененному значению первичного ключа соответствует значение в ограничении FOREIGN KEY в другой таблице, то действие выполнено не будет. Для успешного изменения или удаления строки с ограничением FOREIGN KEY необходимо сначала удалить данные внешнего ключа в соответствующей таблице либо изменить данные в таблице внешнего ключа, которые связывают внешний ключ с данными другого первичного ключа.

Поддержка ссылочной целостности с помощью триггеров

Триггеры DML часто используются для соблюдения бизнес-правил и целостности данных. В SQL Server декларативное ограничение ссылочной целостности обеспечивается инструкциями ALTER TABLE и CREATE TABLE. Однако декларативное ограничение ссылочной целостности не обеспечивает ссылочную целостность между базами данных. Ограничение ссылочной целостности подразумевает выполнение правил связи между первичными и внешними ключами таблиц. Для обеспечения ограничений ссылочной целостности используйте в инструкциях ALTER TABLE и CREATE TABLE ограничения PRIMARY KEY и FOREIGN KEY. Если ограничения распространяются на таблицу триггера, они проверяются после срабатывания триггера INSTEAD OF и до выполнения триггера AFTER. В случае нарушения ограничения выполняется откат действий триггера INSTEAD OF, и триггер AFTER не срабатывает.
Пример использования триггеров для поддержки ссылочной целостности – каскадное удаление или обновление данных

Пример каскадного удаления

```
CREATE TRIGGER delTrigger ON vuz_gorod
FOR DELETE AS
BEGIN
        DELETE vuz
        FROM deleted,vuz
        WHERE  vuz.cod=deleted.cod_vuza
END
```

Пример каскадного обновления

```
CREATE TRIGGER updTrigger ON vuz_gorod
for update as
begin
```

```
        update vuz
        set vuz.cod = inserted.cod_vuza, vuz.uch_zavedenie = inserted.nazvanie_vuza
        from deleted, vuz, inserted
        where ((deleted.cod_vuza = vuz.cod)
                or (deleted.nazvanie_vuza = vuz.uch_zavedenie))
end
```

25. T-SQL. Персональные, списковые и количественные запросы. Агрегатные функции. Особенности использования фразы group by. Реализация количественного запроса по одному или нескольким столбцам с использованием T-SQL. Примеры.

Агрегатные функции

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме COUNT(*), фактический (т.е. требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций - null.
Вычисление функции COUNT(*) производится путем подсчета числа строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением null во всех строках.
Стандартом предусмотрены следующие агрегатные функции:

Функция	Описание
COUNT(*)	Возвращает количество строк источника записей.
COUNT(<имя поля>)	Возвращает количество значений в указанном столбце.
SUM(<имя поля>)	Возвращает сумму значений в указанном столбце.
AVG(<имя поля>)	Возвращает среднее значение в указанном столбце.
MIN(<имя поля>)	Возвращает минимальное значение в указанном столбце.
MAX(<имя поля>)	Возвращает максимальное значение в указанном столбце.

Все эти функции возвращают единственное значение. При этом функции COUNT, MIN и MAX применимы к любым типам данных, в то время как SUM и AVG используются только для числовых полей. Разница между функцией COUNT(*) и COUNT(<имя поля>) состоит в том, что вторая при подсчете не учитывает NULL-значения.

Предложение GROUP BY

Предложение GROUP BY позволяет вам определять подмножество значений в особом поле в терминах другого пол, и применять функцию агрегата к подмножеству. Это дает вам возможность объединять пол и агрегатные функции в едином предложении SELECT. Например, предположим что вы хотите найти наибольшую сумму приобретений полученную каждым продавцом. Вы можете сделать раздельный запрос для каждого из них, выбрав MAX (amt) из таблицы Порядков для каждого значения пол snum. GROUP BY, однако, позволит Вам поместить их все в одну команду:

```
=============== SQL Execution Log ==============
|                                  |
| SELECT snum, MAX (amt)                  |
| FROM  Orders                          |
| GROUP BY snum;                          |
| ========================================== |
|  snum                              |
|  ------   --------                  |
|   1001     767.19                     |
|   1002    1713.23                     |
|   1003      75.75                    |
|   1014    1309.95                     |
 ============================================
```

GROUP BY применяет агрегатные функции независимо от серий групп которые определяются с помощью значения поля в целом. В этом случае, каждая группа состоит из всех строк с тем же самым значением пол snum, и MAX функция применяется отдельно для каждой такой группы. Это значение пол, к которому применяется GROUP BY, имеет, по определению, только одно значение на группу вывода, также как это делает агрегатная функция. Результатом является совместимость которая позволяет агрегатам и полям

объединяться таким образом. Вы можете также использовать GROUP BY с многочисленными полями. Совершенству вышеупомянутый пример далее, предположим что вы хотите увидеть наибольшую сумму приобретений получаемую каждым продавцом каждый день. Чтобы сделать это, вы должны сгруппировать таблицу Порядков по датам продавцов, и применить функцию MAX к каждой такой группе, подобно этому:

```
 ============== SQL Execution Log ==============
|                                  |
| SELECT snum, odate, MAX (amt)              |
| FROM  Orders                          |
| GROUP BY snum, odate;                    |
| ========================================== |
|  snum       odate                    |
| ------    ----------    --------          |
|  1001     10/03/1990     767.19          |
|  1001     10/05/1990    4723.00          |
|  1001     10/06/1990    9891.88          |
|  1002     10/03/1990    5160.45          |
|  1002     10/04/1990     75.75          |
|  1002     10/06/1990    1309.95          |
|  1003     10/04/1990    1713.23          |
|  1014     10/03/1990    1900.10          |
|  1007     10/03/1990    1098.16          |
 ==============================================
```

На месте имен полей могут быть использованы их порядковые номера в списке полей результирующей таблицы.

Вариант двумерного статистического запроса к БД

Выдать статистику по всем учебным заведениям и категориям обучения учащихся России, принятых на учебу в 1990 году.

В этом запросе требуется, во-первых: выдать общее количество учащихся, принятых на учебу в 1990 году; во-вторых: распределить это количество по разным учебным заведениям, т.е. для каждого учебного заведения определить количественные характеристики; в-третьих: каждую количественную характеристику каждого учебного заведения распределить по разным категориям обучения, т.е. определить количественные характеристики каждой категории обучения в каждом учебном заведении.

Другими словами, если использовать только SELECT - предложения, то необходимо выполнить три запроса (и получить три разных отчета):

1) запрос получения общего количества учащихся, принятых в 1990 году;
2) запрос с условием выборки по году приема (значение '90'), с предложением group by для группирования значений по полю "учебное заведение" и с агрегатной функцией count(*) в списке выборки для оценки количества значений учебных заведений в каждой группе;
3) запрос с условием выборки по году приема (значение '90'), с предложением group by для группирования значений по двум полям: "учебное заведение" и "категория обучения" и с агрегатной функцией count(*) в списке выборки для оценки количества значений категорий обучения в каждой группе.

Замечание. То, что функция count(*) подсчитывает количество значений в группах для последнего указанного в списке выборки поля, является ограничением языка SQL, поэтому для подсчета статистики по двум полям (атрибутам) требуется выполнять два отдельных запроса.

Перечисленные выше запросы выглядят следующим образом:

1)
```
select count(*) from poss where gp='90'
```
2)
```
select vuz.uch_zavedenie,
count(*)
from poss, vuz
where gp='90' and vuz_k=vuz.cod
group by vuz.uch_zavedenie
```
3)
```
select vuz.uch_zavedenie, kat_obuch.kat_obucheniya,
count(*)
from poss, vuz, kat_obuch
where gp='90' and vuz_k=vuz.cod
```

```
        and kat_obuch_k=kat_obuch.cod
        group by vuz.uch_zavedenie, kat_obuch.kat_obucheniya
```

Замечание. Значения полей "учебное заведение" и "категория обучения" при выдаче отчета должны быть раскодированы, поэтому в запросах используется операция соединения таблиц.

То, что результат двумерного статистического запроса расположен в трех разных отчетах, не совсем удовлетворяет пользователя. Для исправления этого нужно использовать программирование в хранимой процедуре.

Для объединения всех количественных характеристик значений атрибутов и общего итога по запросу в один отчет необходимо в теле хранимой процедуры использовать курсор (см. конспект курса). В качестве предложения select для курсора нужно взять запрос номер 3), в котором определяются количества значений по последнему в списке выборки атрибуту (в данном запросе "категория обучения"). Суммируя количественные характеристики значений категорий обучения можно получить количественные характеристики соответствующих значений учебных заведений ("учебное заведение" - первый атрибут в списке выборки), а также общий итог по запросу (т.е. по приему 1990 года).

Для создания процедуры в верхней области окна необходимо ввести текст процедуры (команда create procedure):

```
create procedure zapr8 as
declare @yz char (25) /* переменная для названия учебного заведения */
declare @ko char (16) /*переменная для названия категории обучения */
declare @it1 int /* переменная для итога по полю категория обучения */
declare @it2 int /* переменная для итога по полю учебное заведение */
declare @itall int /* переменная для итога по всему запросу */
declare @yz1 char (25) /*переменная для названия учебного заведения*/
/* объявление курсора для двумерного статистического запроса */
declare y cursor for
select vuz.uch_zavedenie, kat_obucheniya, count (*)
from poss join vuz on vuz_k=vuz.cod
        join kat_obuch on kat_obuch_k=kat_obuch.cod
where gp='90'
group by vuz.uch_zavedenie, kat_obuch.kat_obucheniya
select @it2=0 /*начальное значение итога по полю уч. заведение=0*/
select @itall=0 /* начальное значение итога по запросу = 0 */
open y /* открытие курсора */
fetch y into @yz, @ko, @it1
/* считывание первого итога по полю категория обучения */
if (@@fetch_status=-2) begin
print 'Ошибка при выполнении первого FETCH'
close y /* закрытие курсора и останов процедуры в случае ошибки */
return
end
if (@@fetch_status=-1) begin
print 'Данные не найдены'
close y /*закрытие курсора и останов процедуры в случае отсутствия данных по запросу */
return
end
select @yz1=@yz /*запоминание названия учебного заведения в @yz1*/
print @ko+' -'+str(@it1) /*печать названия категории обучения и итога*/
select @it2=@it2+@it1 /* подсчет итога по полю учебное заведение */
select @itall=@itall+@it1 /* подсчет общего итога по запросу */
/* цикл обработки запроса */
while (@@fetch_status=0)
begin
        fetch y into @yz, @ko, @it1 /* считывание очередного итога по полю категория обучения */
if (@yz1 !=@yz) begin /* если название учебного заведения поменялось, то печать */
        print 'учебное заведение ' + @yz1 + ' - ' +str(@it2) /* старого названия учебного заведения и итога */
select @yz1=@yz/*присвоение нового названия учебного заведения */
select @it2=0 /* новый итог = 0 */
```

```
end
if (@@fetch_status=-1) break /* при окончании данных немедленный выход из цикла */
print @ko + ' - ' +str(@it1) /* печать очередных названия категории обучения и итога */
select @it2=@it2+@it1 /* подсчет итога по полю учебное заведение */
select @itall=@itall+@it1 /* подсчет общего итога по запросу */
end
close y /* закрытие курсора по окончании цикла */
if (@@fetch_status=-2) begin
print 'Ошибка при выполнении FETCH'
return /* останов процедуры в случае ошибки */
end
print 'учебное заведение ' + @yz + ' - ' +str(@it2) /* печать последнего итога по учебному заведению */
print '---------------------------------'
print 'всего по запросу-'+str(@itall) /*печать общего итога по запросу */
deallocate y /* освобождение курсора */
return                                  [F5]
```

26. Транзакция, ее определение и назначение. Свойства транзакций.

Концепция транзакций – неотъемлемая часть любой клиент-серверной базы данных.

Под **транзакцией** понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), приводящая к одному из двух возможных результатов: либо последовательность выполняется, если все операторы правильные, либо вся транзакция откатывается, если хотя бы один оператор не может быть успешно выполнен. Обработка транзакций гарантирует целостность информации в базе данных. Таким образом, транзакция переводит базу данных из одного целостного состояния в другое.

Поддержание механизма транзакций – показатель уровня развитости СУБД. Корректное поддержание транзакций одновременно является основой обеспечения целостности БД. Транзакции также составляют основу изолированности в многопользовательских системах, где с одной БД параллельно могут работать несколько пользователей или прикладных программ. Одна из основных задач СУБД – обеспечение изолированности, т.е. создание такого режима функционирования, при котором каждому пользователю казалось бы, что БД доступна только ему. Такую задачу СУБД принято называть параллелизмом транзакций.

Большинство выполняемых действий производится в теле транзакций. По умолчанию каждая команда выполняется как самостоятельная транзакция. При необходимости пользователь может явно указать ее начало и конец, чтобы иметь возможность включить в нее несколько команд.

При выполнении транзакции система управления базами данных должна придерживаться определенных правил обработки набора команд, входящих в транзакцию. В частности, разработано четыре правила, известные как требования ACID, они гарантируют правильность и надежность работы системы.

ACID-свойства транзакций

Характеристики транзакций описываются в терминах ACID (Atomicity, Consistency, Isolation, Durability – *неделимость, согласованность, изолированность, устойчивость*).

- Транзакция **неделима** в том смысле, что представляет собой единое целое. Все ее компоненты либо имеют место, либо нет. Не бывает частичной транзакции. Если может быть выполнена лишь часть транзакции, она отклоняется.
- Транзакция является **согласованной**, потому что не нарушает бизнес-логику и отношения между элементами данных. Это свойство очень важно при разработке клиент-серверных систем, поскольку в хранилище данных поступает большое количество транзакций от разных систем и объектов. Если хотя бы одна из них нарушит целостность данных, то все остальные могут выдать неверные результаты.
- Транзакция всегда **изолирована**, поскольку ее результаты самодостаточны. Они не зависят от предыдущих или последующих транзакций – это свойство называется сериализуемостью и означает, что транзакции в последовательности независимы.
- Транзакция **устойчива**. После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное (до начала транзакции) состояние, т.е. происходит фиксация транзакции, означающая, что ее действие постоянно даже при сбое системы. При этом подразумевается некая форма хранения информации в постоянной памяти как часть транзакции.

Указанные выше правила выполняет сервер. Программист лишь выбирает нужный уровень изоляции, заботится о соблюдении логической целостности данных и бизнес-правил. На него возлагаются обязанности по созданию эффективных и логически верных алгоритмов обработки

данных. Он решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций. Следует по возможности использовать небольшие транзакции, т.е. включающие как можно меньше команд и изменяющие минимум данных. Соблюдение этого требования позволит наиболее эффективным образом обеспечить одновременную работу с данными множества пользователей.

SQL Server работает в следующих режимах транзакций:

- Автоматическое принятие транзакций

  Каждая отдельная инструкция является транзакцией.

- Явные транзакции

  Каждая транзакция явно начинается с инструкции BEGIN TRANSACTION и явно заканчивается инструкцией COMMIT или ROLLBACK.

- Неявные транзакции

  Новая транзакция неявно начинается, когда предыдущая транзакция завершена, но каждая транзакция явно завершается инструкцией COMMIT или ROLLBACK.

- Транзакции контекста пакета

  Будучи применимой только к множественным активным результирующим наборам (MARS), явная или неявная транзакция Transact-SQL, которая запускается в сеансе MARS, становится транзакцией контекста пакета. SQL Server автоматически выполняет откат транзакции контекста пакета, если эта транзакция не принята при завершении пакета или не выполнен ее откат.

27. Блокировки при реализации транзакций. Свойство устойчивости транзакций. T-SQL.

Блокировкой называется временное ограничение на выполнение некоторых операций обработки данных. Блокировка может быть наложена как на отдельную строку таблицы, так и на всю базу данных. **Управлением блокировками** на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. Транзакции и блокировки тесно связаны друг с другом. Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение требований ACID. Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

Блокировка представляет собой метод управления параллельными процессами, при котором объект БД не может быть модифицирован без ведома транзакции, т.е. происходит блокирование доступа к объекту со стороны других транзакций, чем исключается непредсказуемое изменение объекта. Различают два вида блокировки:

- блокировка записи – транзакция блокирует строки в таблицах таким образом, что запрос другой транзакции к этим строкам будет отменен;
- блокировка чтения – транзакция блокирует строки так, что запрос со стороны другой транзакции на блокировку записи этих строк будет отвергнут, а на блокировку чтения – принят.

В СУБД используют протокол доступа к данным, позволяющий избежать проблемы параллелизма. Его суть заключается в следующем:

- транзакция, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить блокировку чтения на эту строку;
- транзакция, предназначенная для модификации строки данных, накладывает на нее блокировку записи;
- если запрашиваемая блокировка на строку отвергается из-за уже имеющейся блокировки, то транзакция переводится в режим ожидания до тех пор, пока блокировка не будет снята;
- блокировка записи сохраняется вплоть до конца выполнения транзакции.

Если в системе управления базами данных не реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа:

- проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т.к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;
- проблема **"грязного" чтения** возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;
- проблема **неповторяемого чтения** является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;
- проблема чтения **фантомов** появляется после того, как одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в специально разработанном стандарте определены четыре уровня блокирования. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

- уровень 0 – запрещение "загрязнения" данных. Этот уровень требует, чтобы изменять данные могла только одна транзакция; если другойтранзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции;
- уровень 1 – запрещение "грязного" чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой;
- уровень 2 – запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;
- уровень 3 – запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

Блокировки, используемые уровнями изоляции, подразделяются на:

- *разделяемые блокировки (S-locks)*, которые могут одновременно устанавливаться несколькими пользователями;
- *исключительные блокировки* (X-locks), которые устанавливаются только одним пользователем, получающим эксклюзивный доступ к данным.

Существуют следующие логические и физические уровни *блокировок*:

- *блокировка на уровне таблицы* (table-level locking);
- *блокировка на уровне строк* (row-level locking);
- *блокировка на уровне элемента таблицы* (item-level locking);
- *блокировка на уровне БД* (dbspace-level locking);
- *блокировка на уровне табличного пространства* (tablespace-level locking);

блокировка на уровне страницы или блока (page-level locking).

SET TRANSACTION ISOLATION LEVEL (SQL Server 2008 R2)

Управляет поведением блокировки и версиями строк инструкций Transact-SQL, выданных при подключении к SQL Server.

Синтаксис

```
SET TRANSACTION ISOLATION LEVEL
   { READ UNCOMMITTED
   | READ COMMITTED
   | REPEATABLE READ
   | SNAPSHOT
   | SERIALIZABLE
   }
[ ; ]
```

READ UNCOMMITTED:

Указывает, что инструкции могут считывать строки, которые были изменены другими транзакциями, но еще не были зафиксированы.

READ COMMITTED:

Указывает, что инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы. Это предотвращает чтение«грязных» данных. Данные могут быть изменены другими транзакциями между отдельными инструкциями в текущей транзакции, результатом чего будет неповторяемое чтение или недействительные данные. Этот параметр в SQL Server установлен по умолчанию.

REPEATABLE READ:

Указывает на то, что инструкции не могут считывать данные, которые были изменены, но еще не зафиксированы другими транзакциями, а также на то, что другие транзакции не могут изменять данные, читаемые текущей транзакцией, до ее завершения.

SNAPSHOT:

Указывает на то, что данные, считанные любой инструкцией транзакции, будут согласованы на уровне транзакции с версией данных, существовавших в ее начале. Транзакция распознает только те изменения, которые были зафиксированы до ее начала. Инструкции, выполняемые текущей транзакцией, не видят изменений данных, произведенных другими транзакциями после запуска текущей транзакции. Таким образом достигается эффект получения инструкциями в транзакции моментального снимка зафиксированных данных на момент запуска транзакции.

SERIALIZABLE:

Указывает следующее.

- Инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы.
- Другие транзакции не могут изменять данные, считываемые текущей транзакцией, до ее завершения.
- Другие транзакции не могут вставлять новые строки со значениями ключа, которые входят в диапазон ключей, считываемых инструкциями текущей транзакции, до ее завершения.

28. База данных и ее объекты. Структура языка SQL: операторы определения объектов БД.

Данные SQL Server 2000 организованы с помощью нескольких различных объектов, которые доступны пользователям при подключении к базе данных.

Объект	Описание
Таблица	Двухмерный объект, состоящий из строк и столбцов, который используется для хранения данных в реляционной базе данных. В каждой таблице хранится информация об одном из типов объектов, моделируемых базой данных. Например, в базе данных образовательного учреждения одна таблица может содержать сведения о преподавателях, вторая — о студентах, а третья — о расписании
Тип данных	Атрибут, задающий тип информации, которая может храниться в столбце, параметре или переменной. SQL Server поддерживает несколько системных типов данных; в дополнение к ним разрешается создавать пользовательские типы данных
Представление	Объект БД, на который в SQL-операторах можно ссылаться так же, как на таблицу. Представления определяются с помощью SQL-операторов и являются аналогами объектов, содержащих результирующие наборы, получаемые при выполнении этих операторов
Хранимая	Откомпилированный набор операторов Transact-SQL. процедура хранимый под определенным именем и обрабатываемый как единое целое. SQL Server предоставляет хранимые процедуры для управления SQL Server и вывода сведений о БД и пользователях. Они называются системными хранимыми процедурами
Функция	Фрагмент кода, действующий как единая логическая сущность. Функцию можно вызвать по имени, при этом разрешается задать ряд необязательных входных параметров. Она возвращает сведения о состоянии и необязательные выходные значения. Функции поддерживаются многими языками программирования, в том числе C, Visual Basic и Transact-SQL. В TransactSQL имеется ряд встроенных функций, которые изменить невозможно, а также поддерживаются функции, которые создают и корректируют пользователи
Индекс	Объект реляционной БД, обеспечивающий быстрый доступ к строкам таблицы на основе значений ключа, а так же уникальность строк в таблице. SQL Server поддерживает как кластерные, так и некластерные индексы. Первичный ключ таблицы индексируется автоматически. При полнотекстовом поиске сведения о ключевых словах и их расположении в данном столбце хранятся в полнотекстовом индексе
Ограничение	Свойство, назначаемое столбцу таблицы, которое позволяет предотвратить занесение недопустимых данных в столбец. Например, ограничения UNIQUE или PRIMARY_KEY предотвращают занесение значений, дублирующих существующие. Ограничение CHECK предотвращает занесение значения, не соответствующего критерию поиска, а NOT NULL — пустого значения
Правило	Объект БД, связанный со столбцами или с пользовательскими типами данных, который задает значения данных, приемлемые в данном столбце. Лучше использовать ограничения CHECK, которые предоставляют ту же самую функциональность и соответствуют стандарту SQL-92
Умолчание	Значение, автоматически присваиваемое системой данным, параметру, режиму сопоставления или имени, если оно не задано пользователем. Также определяет действие, автоматически выполняемое при конкретных событиях в отсутствие действий, заданных пользователем
Триггер	Хранимая процедура, исполняемая при модификации данных в заданной таблице. Триггеры часто создают для поддержки ссылочной целостности или согласованности логически связанных данных в различных таблицах

Выражение — это комбинация символов и операторов, которая получает на вход скалярную величину, а на выходе дает другую величину или исполняет какое-то действие. В Transact-SQL выражения делятся на 3 типа: DDL, DCL и DML.

- DDL (Data Definition Language) — используются для создания объектов в базе данных. Основные представители данного класса: CREATE — создание объектов, ALTER — изменение объектов, DROP — удаление объектов.

Язык DDL (data definition language, язык определения данных) применяется для определения объектов баз данных (таких как базы данных, таблиц и представления) и для управления этими объектами. Операторы языка DDL обычно включают в себя команды CREATE, ALTER и DROP для каждого из объектов, с которым производится работа. Например, для первоначального создания таблицы, для изменения ее свойств (скажем, для создания или для удаления колонок) и для уничтожения таблицы соответственно применяются операторы CREATE TABLE, ALTER TABLE и DROP TABLE.

Оператор CREATE TABLE

Применим DDL для создания в базе данных MyDB таблицы с именем Customer_Data (Сведения_о_заказчиках). Для создания таблицы применяется оператор CREATE TABLE. Наша таблица-пример будет задана как имеющая четыре колонки, при помощи следующих операторов:

```
Use MyDB
CREATE TABLE Customer_Data
(customer_id        smallint,
first_name          char(20),
last_name           char(20),
phone               char(10))
GO
```

Эти операторы создают структуру таблицы Customer_Data. Таблица Customer_Data останется пустой до тех пор, пока в нее не будут введены данные или она не будет заполнена при помощи массового копирования.

Оператор ALTER TABLE

Оператор ALTER TABLE применяется для изменения определения или атрибутов таблицы. Применим для добавления в существующую таблицу Customer_Data колонки middle_initial:

```
ALTER TABLE Customer_Data
ADD middle_initial char(1)
GO
```

Теперь определение таблицы содержит не четыре колонки, как было первоначально, а пять колонок.

Оператор DROP TABLE

Оператор DROP TABLE применяется для уничтожения определений таблиц и всех данных, индексов, триггеров, ограничений и специальных разрешений, относящихся к удаляемым таблицам. Для уничтожения нашей таблицы Customer_Data применяется команда:

```
DROP TABLE Customer_Data
GO
```

29. T-SQL. Поиск данных с помощью оператора Select. Структура команды Select. Функции between, in, like и null. Агрегатные функции. Опции group by, having, order by. Примеры.

Оператор SELECT позволяет производить *выборки* данных из таблиц и преобразовывать к нужному виду полученные результаты.
Синтаксис:

```
SELECT [ALL | DISTINCT ] {*|[имя_столбца [AS новое_имя]]} [,...n]
  FROM  имя_таблицы [[AS] псевдоним] [,...n]
  [WHERE  <условие_поиска>]
  [GROUP BY  имя_столбца [,...n]]
  [HAVING <критерии выбора групп>]
  [ORDER BY имя_столбца [,...n]]
```

Символом * можно выбрать все поля, а вместо имени поля применить выражение из нескольких имен. Если обрабатывается ряд таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. Имя_таблицы.Имя_поля.
SELECT – закрытая операция: *результат запроса* к таблице представляет собой другую таблицу.

Параметр WHERE определяет критерий отбора записей из входного набора. Но в таблице могут присутствовать повторяющиеся записи (дубликаты). Предикат ALL задает включение в выходной набор всех дубликатов, отобранных по критерию WHERE. Нет необходимости указывать ALL явно, поскольку это значение действует по умолчанию. Предикат DISTINCT следует применять в тех случаях, когда требуется отбросить блоки данных, содержащие *дублирующие записи* в выбранных полях. Значения для каждого из приведенных в инструкции SELECT полей должны быть уникальными, чтобы содержащая их запись смогла войти в выходной набор.
С помощью WHERE-параметра пользователь определяет, какие блоки данных из приведенных в списке FROM таблиц появятся в *результате запроса*. За ключевым словом WHERE следует перечень *условий поиска*, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов *условий поиска* (или предикатов):

1. *Сравнение*

В языке SQL можно использовать следующие операторы *сравнения*: = – равенство; < – меньше; > – больше; <= – меньше или равно; >= – больше или равно; <> – не равно. Более сложные предикаты могут быть построены с помощью логических операторов AND, OR или NOT, а также скобок, используемых для определения порядка вычисления выражения.

2. *Диапазон*

Оператор BETWEEN используется для поиска значения внутри некоторого интервала, определяемого своими минимальным и максимальным значениями. При этом указанные значения включаются в *условие поиска*.
Пример. Вывести список товаров, цена которых лежит в диапазоне от 100 до 150.

```
SELECT Название, Цена
   FROM Товар
   WHERE Цена Between 100 And 150
```

При использовании отрицания NOT BETWEEN требуется, чтобы проверяемое значение лежало вне границ заданного *диапазона*.

3. *Принадлежность множеству*

Оператор IN используется для *сравнения* некоторого значения со списком заданных значений, при этом проверяется, соответствует ли результат вычисления выражения одному из значений в предоставленном списке. При помощи оператора IN может быть достигнут тот же результат, что и в случае применения оператора OR, однако оператор IN выполняется быстрее.
Пример. Вывести список клиентов из Москвы или из Самары.

```
SELECT Фамилия, ГородКлиента
  FROM Клиент
  WHERE ГородКлиента in ("Москва", "Самара")
```

NOT IN используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

4. ***Соответствие шаблону***

С помощью оператора LIKE можно выполнять *сравнение* выражения с заданным шаблоном, в котором допускается использование символов-заменителей:

- Символ % – вместо этого символа может быть подставлено любое количество произвольных символов.
- Символ _ заменяет один символ строки.
- [] – вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях.
- [^] – вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Пример. Найти клиентов, у которых в номере телефона вторая цифра – 2 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
   FROM Клиент
   WHERE Клиент.Телефон Like "_[2,4]%"
```

5. ***Значение NULL***

Оператор IS NULL используется для *сравнения* текущего значения со *значением* NULL – специальным значением, указывающим на отсутствие любого значения. NULL – это не то же самое, что знак пробела (пробел – допустимый символ) или ноль (0 – допустимое число). NULL отличается и от строки нулевой длины (пустой строки).

Пример. Найти сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

```
SELECT Фамилия, Телефон
   FROM Клиент
   WHERE Телефон Is Null
```

IS NOT NULL используется для проверки присутствия значения в поле.

Агрегатные функции.

С помощью *итоговых (агрегатных) функций* в рамках SQL-запроса можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

- Count (Выражение) - определяет количество записей в выходном наборе SQL-запроса;
- Min/Max (Выражение) - определяют наименьшее и наибольшее из множества значений в некотором поле запроса;
- Avg (Выражение) - эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.
- Sum (Выражение) - вычисляет сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово DISTINCT. Оно не имеет смысла для функций MIN и MAX, однако его использование может повлиять на результаты выполнения функций SUM и AVG, поэтому необходимо заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово DISTINCT может быть указано в любом запросе не более одного раза.

Агрегатные функции могут использоваться только в списке предложения SELECT и в составе предложения HAVING. Во всех других случаях это недопустимо.

Пример. Определить первое по алфавиту название товара.

```
SELECT Min(Товар.Название) AS Min_Название
FROM Товар
```

Пример. Определить количество сделок.

```
SELECT Count(*) AS Количество_сделок
FROM Сделка
```

Предложение GROUP BY

Часто в запросах требуется формировать промежуточные итоги, что обычно отображается появлением в запросе фразы "для каждого...". Для этой цели в операторе SELECT используется предложение GROUP BY. Запрос, в котором присутствует GROUP BY, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная суммарная строка.

Пример. Вычислить средний объем покупок, совершенных каждым покупателем.

```
SELECT Клиент.Фамилия, Avg(Сделка.Количество)
   AS Среднее_количество
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фамилия
```

Предложение HAVING

При помощи HAVING отражаются все предварительно сгруппированные посредством GROUP BY блоки данных, удовлетворяющие заданным в HAVING условиям. Это дополнительная возможность "профильтровать" выходной набор.

Условия в HAVING отличаются от условий в WHERE:

- HAVING исключает из результирующего набора данных группы с результатами агрегированных значений;
- WHERE исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию;
- в условии поиска WHERE нельзя задавать агрегатные функции.

Пример. Определить фирмы, у которых общее количество сделок превысило три.

```
SELECT Клиент.Фирма, Count(Сделка.Количество)
   AS Количество_сделок
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фирма
HAVING Count(Сделка.Количество)>3
```

Предложение ORDER BY

В общем случае строки в результирующей таблице SQL-запроса никак не упорядочены. Однако их можно требуемым образом отсортировать, для чего в оператор SELECT помещается фраза ORDER BY, которая сортирует данные выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевым словом ORDER BY через запятую. Способ сортировки задается ключевым словом, указываемым в рамках параметра ORDER BY следом за названием поля, по которому выполняется сортировка. По умолчанию реализуется сортировка по возрастанию. Явно она задается ключевым словом ASC. Для

выполнения сортировки в обратной последовательности необходимо после имени поля, по которому она выполняется, указать ключевое слово DESC. Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результата или нет. Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.
Во фразе ORDER BY может быть указано и больше одного элемента.
Пример. Вывести список фирм и клиентов. Названия фирм упорядочить в алфавитном порядке, имена клиентов в каждой фирме отсортировать в обратном порядке.

```
SELECT Клиент.Фирма, Клиент.Фамилия
    FROM Клиент
    ORDER BY Клиент.Фирма, Клиент.Фамилия DESC
```

30. T-SQL. Операторы создания и удаления таблиц БД, индексов.

Таблица – основной объект для хранения информации в реляционной базе данных. Она состоит из содержащих данные строк и столбцов, занимает в базе данных физическое пространство и может быть постоянной или временной.
Поле, также называемое в реляционной базе данных столбцом, является частью таблицы, за которой закреплен определенный тип данных. Каждая таблица базы данных должна содержать хотя бы один столбец. Строка данных – это запись в таблице базы данных, она включает поля, содержащие данные из одной записи таблицы.
Базовый синтаксис оператора создания таблицы:

```
CREATE TABLE имя_таблицы
  (имя_столбца тип_данных
    [NULL | NOT NULL ] [,...n])
```

Удаление таблицы из БД:

```
DROP TABLE имя_таблицы [RESTRICT | CASCADE]
```

Оператор DROP TABLE дополнительно позволяет указывать, следует ли операцию удаления выполнять каскадно. Если в операторе указано ключевое слово RESTRICT, то при наличии в базе данных хотя бы одного объекта, существование которого зависит от удаляемой таблицы, выполнение оператора DROP TABLE будет отменено. Если указано ключевое слово CASCADE, автоматически удаляются и все прочие объекты базы данных, чье существование зависит от удаляемой таблицы, а также другие объекты, зависящие от удаляемых объектов.

Пример.

```
create table member
( member_no int not null,
lastname char(50) not null,
firstname char(50) not null,
photo image null )
```

Этим оператором создается таблица member, состоящая из четырех колонок:
member_no - имеет тип int, значения null не допускаются
lastname - имеет тип char(50) - 50 символов, значения null не допускаются
firstname - аналогично lastname
photo - имеет тип image (изображение), допускается значение null

Индексы представляют собой структуру, позволяющую выполнять ускоренный доступ к строкам таблицы на основе значений одного или более ее столбцов. Наличие индекса может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания. **Индекс** – это набор ссылок, упорядоченных по определенному столбцу таблицы, который в данном случае будет называться индексированным столбцом. Физически индекс – всего лишь упорядоченный набор значений из индексированного столбца с указателями на места физического размещения исходных строк в структуре базы данных
Создание индекса

```
CREATE [UNIQUE][CLUSTERED|NONCLUSTERED]INDEX index_name
 ON [[database.]owner.]table_name(column_name[,column_name]...)
[WITH]
 [FILLFACTOR = x]
 [[,] IGNORE_DUP_KEY]
 [[,] {SORTED_DATA | SORTED_DATA_REORG}]
 [[,] {IGNORE_DUP_ROW | ALLOW_DUP_ROW}]]
[ON segment_name]
```

Создает индекс для перечисленных колонок на указанном сегменте.

CLUSTERED - создавать кластеризованный индекс, т.е. такой индекс, при котором в листьях B-дерева, образующего индекс, находятся не ссылки на данные, а собственно страницы данных.

FILLFACTOR - позволяет управлять заполнением страниц B-дерева индекса, задается в процентах, 100% - полное заполнение.

Пример

```
CREATE UNIQUE CLUSTERED INDEX au_id_ind
    ON authors (au_id)
```

Удаление индекса:

```
DROP INDEX имя_индекса
```

В среде SQL Server реализовано несколько типов индексов:

- кластерные индексы;
- некластерные индексы;
- уникальные индексы.

Некластерные индексы – наиболее типичные представители семейства индексов. В отличие от кластерных, они не перестраивают физическую структуру таблицы, а лишь организуют ссылки на соответствующие строки.

Принципиальным отличием **кластерного индекса** от индексов других типов является то, что при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса. Логическая структура таблицы в этом случае представляет собой скорее словарь, чем индекс. Данные в словаре физически упорядочены, например по алфавиту.

Кластерные индексы могут дать существенное увеличение производительности поиска данных даже по сравнению с обычными индексами. Если в таблице определен некластерный индекс, то сервер должен сначала обратиться к индексу, а затем найти нужную строку в таблице. При использовании кластерных индексов следующая порция данных располагается сразу после найденных ранее данных. Благодаря этому отпадают лишние операции, связанные с обращением к индексу и новым поиском нужной строки в таблице. В таблице может быть определен только один кластерный индекс.

Уникальный индекс: Уникальность значений в индексируемом столбце гарантируют уникальные индексы. При их наличии сервер не разрешит вставить новое или изменить существующее значение таким образом, чтобы в результате этой операции в столбце появились два одинаковых значения.

Уникальный индекс является своеобразной надстройкой и может быть реализован как для кластерного, так и для некластерного индекса. В одной таблице может существовать один уникальный кластерный и множество уникальных некластерных индексов.

Средства языка SQL предлагают несколько способов определения индекса:

- автоматическое создание индекса при создании первичного ключа;
- автоматическое создание индекса при определении ограничения целостности UNIQUE;
- создание индекса с помощью команды CREATE INDEX.

31. T-SQL. Операторы загрузки таблиц, удаления и обновления данных таблицы. Типы данных.

Создание таблиц с помощью оператора **CREATE TABLE** (упрощенный синтаксис):
CREATE TABLE имя_таблицы (
имя_столбца тип_данных [NULL | NOT NULL] [CONSTRAINTS],
имя_столбца тип_данных[NULL|NOT NULL] [CONSTRAINTS] , ...);
Пример.
CREATE TABLE member
(member_no int NOT NULL,
lastname char(50) NOT NULL,
firstname char(50) NOT NULL)

Оператор **ALTER TABLE** используется для обновления схемы существующей таблицы.
ALTER TABLE имя_таблицы (
ADD | DROP имя_столбца тип_данных [NULL | NOT NULL] [CONSTRAINTS] ,
ADD I DROP имя_столбца тип_данных [NULL|NOT NULL] [CONSTRAINTS] ,…);

Удаление таблиц (удаление именно таблиц, а не их содержимого) с помощью оператора **DROP TABLE**:
DROP TABLE имя_таблицы;

Оператор **INSERT** добавляет в таблицу одну строку.
INSERT INTO имя_таблицы [(имена_столбцов, ...)] VALUES[значения, ...);
Пример.
INSERT INTO Customers(cust_id, cust_contact, cust_email, cust_name, cust_address, cust_city, cust_state, cust_ZIP)
VALUES('4000000006' , NULL, NULL, 'Toy Land', '123 Any Street', 'New York', 'NY' , '11111');

Копирование данных из одной таблицы в другую с помощью оператора **SELECT INTO**.
Это — другая форма добавления данных, при использовании которой оператор INSERT вообще не применяется. Чтобы скопировать содержимое какой-то таблицы в новую (которая создается "на лету").
SELECT * INTO имя_таблицы1 FROM имя_таблицы2;

Оператор **UPDATE** обновляет одну или несколько строк таблицы.
UPDATE имя_таблицы SET имя_столбца = значение, [WHERE ...];

Оператор UPDATE очень прост в использовании, он состоит из трех основных частей:
- имени таблицы, подлежащей обновлению;
- имен столбцов и их новых значений;
- условий фильтрации, определяющих, какие именно строки должны быть обновлены.

Пример. У клиента 10 появился адрес электронной почты, поэтому его запись нужно обновить.
UPDATE Customers
SET cust_email = 'kim@thetoystore.com'
WHERE cust_id = '10';
Для обновления нескольких столбцов необходим иной синтаксис:
UPDATE Customers SET cust_contact = 'Sam Roberts', cust_email = 'sam@toyland.com' WHERE cust_id = '10';

Оператор **DELETE** удаляет одну или несколько строк таблицы.
DELETE FROM имя_таблицы [WHERE ...];
Его можно использовать двумя способами:
- для удаления из таблицы определенных строк;
- для удаления из таблицы всех ее строк.

Применять оператор delete следует с особой осторожностью, потому что можно ошибочно удалить все строки таблицы.

Следующий оператор удаляет одну строку из таблицы Customers:

```
DELETE FROM Customers
WHERE cust_id = '10';
```

Оператор **DELETE FROM** требует, чтобы вы указали имя таблицы, из которой должны быть удалены данные. Предложение WHERE фильтрует строки, определяя, какие из них должны быть удалены. В нашем примере должна быть удалена строка, относящаяся к клиенту 10. Если бы предложение WHERE было пропущено, этот оператор удалил бы все столбцы таблицы.

Ключевое слово FROM

В некоторых реализациях SQL за delete может опционально следовать ключевое слово from. Однако хорошим тоном считается всегда указывать это ключевое слово, даже если в нем нет необходимости. Поступая таким образом, вы обеспечите переносимость своего SQL-кода между СУБД.

Оператор DELETE не принимает имена столбцов или метасимволы. Он удаляет строки целиком, а не отдельные столбцы. Для удаления определенного столбца следует использовать оператор UPDATE.

Если необходимо удалить значения из всех строк таблицы, не используйте оператор delete. Вместо него нужно применить оператор **truncate table**, который выполняет то же самое, но делает это намного быстрее (потому что изменения данных не регистрируются).

Microsoft SQL Server поддерживает следующие **типы данных**:

Тип данных	**Обозначение**	**Размер, байт**
Бинарные данные	binary varbinary[(n)]	1-8000
Символы	char[(n)] varchar[(n)]	1-8000 (до 8000 символов)
Символы Unicode	nchar[(n)] nvarchar[(n)]	1-8000 (до 4000 символов)
Дата и время	datetime smalldatetime	8 4
Точные числа	decimal[(p[,s])] numeric[(p[,s])]	5-17
Приблизительные числа	float[(n)] real	4-8 4
Глобальный идентификатор	uniqueidentifier	16
Целые числа	int smallint, tinyint	4 2, 1
Денежки	money, smallmoney	8, 4
Специальные	bit, cursor, sysname, timestamp	1, 0-8
Текст и изображение	text, image	0-2 Гб
Текст Unicode	ntext	0-2 Гб

32. T-SQL . Задание ограничений целостности в команде create table. Примеры.

1) Создание контрольных ограничений на уровне поля.

Рассмотрим структуру таблицы poss:

а) поле номер (nomer) - контрольным ограничением для него является промежуток чисел от 1 до 700000;

б) поле фамилия, имя, отчество (fio) - контрольным ограничением для него является запрет использования символов "." (точка) и "-" (минус);

в) поле дата рождения (data_rogden) разбивается на три подполя:

1. день рождения (d_rogd) - контрольным ограничением является промежуток символов от "01" до "31";
2. месяц рождения (m_rogd) - контрольным ограничением является промежуток символов от "01" до "12";
3. год рождения (g_rogd) - контрольным ограничением является промежуток символов от "00" до "99";

г) поле пол (pol) - контрольным ограничением для него является использование либо символа "М", либо символа "Ж"

На основе перечисленных для каждого поля структуры таблицы poss контрольных ограничений можно создать таблицу с именем poss1.

```
create table poss1
(nomer integer check (nomer between 1 and 700000),
fio char (40) not null check (fio not like '%.%' or fio not like '%-%'),
d_rogd char(2) check (d_rogd between '01' and '31'),
m_rogd char(2) check (m_rogd between '01' and '12'),
g_rogd char(2) check (g_rogd between '00' and '99'),
pol char(1) check (pol like 'M' or pol like 'Ж'))        [F5]
```

Проверка действий контрольных ограничений осуществляется оператором insert (или update), т.е. при вводе записей в таблицу.

2) Создание контрольных ограничений на уровне таблицы.

Рассмотрим некоторые значения поля "категория обучения":

а) значения "п/ф(вуз)" и "п/ф(техн.)" (коды "01" и "02") обозначают обучаемого на подготовительных факультетах (в вузе и техникуме), срок обучения которого не превышает одного года;

б) значения "студент" и "студент-заочник" (коды "03" и "04") обозначают обучаемого в высшем учебном заведении, срок обучения которого не превышает 5 или 6 лет;

Срок обучения можно подсчитать, если найти разницу между годом окончания учебного заведения и годом приема. Для этого нужно использовать следующее выражение:

convert (integer, gok) - convert (integer, gp)

Для создания рассмотренных контрольных ограничений воспользуемся возможностью создания контрольных ограничений на уровне таблицы в команде create table таблицы с именем poss2:

```
create table poss2
(nomer integer check (nomer between 1 and 700000),
fio char (40) not null check (fio not like '%.%' or fio not like '%-%'),
d_rogd char(2) check (d_rogd between '01' and '31'),
m_rogd char(2) check (m_rogd between '01' and '12'),
g_rogd char(2) check (g_rogd between '00' and '99'),
pol char(1) check (pol like 'M' or pol like 'Ж'),
constraint kat_obuch_const check
        ((kat_obuch_k in ('01', '02') and
        (convert (integer, gok) - convert (integer, gp))=1) or
        (kat_obuch_k in ('03', '04') and
        (convert (integer, gok) - convert (integer, gp)) in (5, 6)))   [F5]
```

Проверка действий контрольных ограничений осуществляется оператором insert (или update), т.е. при вводе записей в таблицу.

Syntax:

CONSTRAINT constraint_name
{
PRIMARY KEY
| UNIQUE
| NOT NULL
| REFERENCES foreign_table
[(foreign_field1, foreign_field2)]
}

CONSTRAINT constraint_name
{
PRIMARY KEY (primary1 [, primary2 [, ...]])
| UNIQUE (unique1 [, unique2 [, ...]])
| NOT NULL (notnull1 [, notnull2 [, ...]])
| FOREIGN KEY (ref1 [, ref2 [, ...]])
| REFERENCES foreign_table [(foreign_field1 [, foreign_field2 [, ...]])]
}

constraint_name: Is the name of the constraint being created.
PRIMARY KEY: Is a parameter that identifies the column or set of columns whose values uniquely identify each row in a table. Each table can only have one primary key constraint.
UNIQUE: Is a constraint that enforces the uniqueness of the values in a set of columns.
NOT NULL: Is a parameter that indicates whether a column can or cannot contain null values.
REFERENCES: Is a keyword that indicates a relationship between two tables is being established..
foreign_table: Is the name of the table that the relationship is to be made with.
foreign_field1: Is a parameter that lists the name of the field(s) from the foreign_table on which to create the foreign key.
primary1: Is a parameter that specifies a list of fields that are to be used as the primary key.
unique1: Is a parameter that specifies a list of fields that are to be unique.
notnull1: Is a parameter that specifies a list of fields that cannot have null values.
ref1: Is a parameter that specifies a list of fields on which a foreign key is to be created.
The CONSTRAINT clause is used to maintain data integrity by providing limits on the values that can be inserted into a column or table.

While a CONSTRAINT clause is somewhat similar to an INDEX, a CONSTRAINT can establish a relationship with another table. To place a constraint on a single field in a CREATE TABLE or ALTER TABLE statement, follow the definition of that field with a CONSTRAINT clause. This consists of a name for the constraint and one of the following reserved words: PRIMARY KEY, UNIQUE, NOT NULL or REFERENCES.

Example.

```
CREATE TABLE Names (NameID INTEGER CONSTRAINT NameIDKey PRIMARY KEY,
FirstName VARCHAR (20), LastName VARCHAR (20), DateOfBirth DATETIME);
```

33. Понятие об администрировании баз данных. Средства администрирования БД в SQLServer 2005.

Основные функции группы администратора БД

1. **Анализ предметной области**: описание предметной области, выявление ограничений целостности, определение статуса (доступности, секретности) информации, определение потребностей пользователей, определение соответствия "данные—пользователь", определение объемно-временных характеристик обработки данных.
2. **Проектирование структуры БД**: определение состава и структуры файлов БД и связей между ними, выбор методов упорядочения данных и методов доступа к информации, описание БД на языке описания данных (ЯОД).
3. **Задание ограничений целостности при описании структуры БД и процедур обработки БД**:
   - задание декларативных ограничений целостности, присущих предметной области;
   - определение динамических ограничений целостности, присущих предметной области в процессе изменения информации, хранящейся в БД;
   - определение ограничений целостности, вызванных структурой БД;
   - разработка процедур обеспечения целостности БД при вводе и корректировке данных;
   - определение ограничений целостности при параллельной работе пользователей в многопользовательском режиме.
4. **Первоначальная загрузка и ведение БД**:
   - разработка технологии первоначальной загрузки БД, которая будет отличаться от процедуры модификации и дополнения данными при штатном использовании базы данных;
   - разработка технологии проверки соответствия введенных данных реальному состоянию предметной области. База данных моделирует реальные объекты некоторой предметной области и взаимосвязи между ними, и на момент начала штатной эксплуатации эта модель должна полностью соответствовать состоянию объектов предметной области на данный момент времени;
   - в соответствии с разработанной технологией первоначальной загрузки может понадобиться проектирование системы первоначального ввода данных.
5. **Защита данных:**
   - определение системы паролей, принципов регистрации пользователей, создание групп пользователей, обладающих одинаковыми правами доступа к данным;
   - разработка принципов защиты конкретных данных и объектов проектирования; разработка специализированных методов кодирования информации при ее циркуляции в локальной и глобальной информационных сетях;
   - разработка средств фиксации доступа к данным и попыток нарушения системы защиты;
   - тестирование системы защиты;
   - исследование случаев нарушения системы защиты и развитие динамических методов защиты информации в БД.
6. **Обеспечение восстановления БД**:
   - разработка организационных средств архивирования и принципов восстановления БД;
   - разработка дополнительных программных средств и технологических процессов восстановления БД после сбоев.
7. **Анализ обращений пользователей БД**: сбор статистики по характеру запросов, по времени их выполнения, по требуемым выходным документам
8. **Анализ эффективности функционирования БД:**
   - анализ показателей функционирования БД;
   - планирование реструктуризации (изменение структуры) БД и реорганизации БнД.
9. **Работа с конечными пользователями**:
   - сбор информации об изменении предметной области;
   - сбор информации об оценке работы БД;
   - обучение пользователей, консультирование пользователей;

- разработка необходимой методической и учебной документации по работе конечных пользователей.

10. **Подготовка и поддержание системных средств:**
    - анализ существующих на рынке программных средств и анализ возможности и необходимости их использования в рамках БД;
    - разработка требуемых организационных и программно-технических мероприятий по развитию БД;
    - проверка работоспособности закупаемых программных средств перед подключением их к БД;
    - курирование подключения новых программных средств к БД.
11. **Организационно-методическая работа по проектированию БД**:
    - выбор или создание методики проектирования БД;
    - определение целей и направления развития системы в целом;
    - планирование этапов развития БД;
    - разработка общих словарей-справочников проекта БД и концептуальной модели;
    - стыковка внешних моделей разрабатываемых приложений;
    - курирование подключения нового приложения к действующей БД;
    - обеспечение возможности комплексной отладки множества приложений, взаимодействующих с одной БД.

Средства администрирования БД в SQL Server 2005

1. Редактор кода в среде SQL Server Management Studio обеспечивают следующие возможности.
    - Шаблоны, которые могут быть использованы для быстрой подготовки сценариев для SQL Server, служб SQL Server 2005 Analysis Services (SSAS) и SQL Server 2005 Compact Edition. Шаблоны — это файлы, содержащие базовый набор инструкций, необходимых для создания объектов в базе данных.
    - Выделение цветом синтаксических конструкций, облегчающее читаемость сложных инструкций.
    - Создание запросов в графическом конструкторе запросов методом перетаскивания.
    - Представление окон запросов в виде вкладок окна документа или в виде отдельных документов.
    - Представление результатов выполнения запроса в виде табличной сетки или текстового окна с возможностью перенаправления в файл.
    - Отображение табличной сетки результатов в виде отдельных окон с вкладками.
    - Графическое отображение результатов инструкции Showplan, отражающих логические шаги построения плана выполнения инструкции Transact-SQL. Management Studio при подключении к экземплярам SQL Server 2005 получает план от SQL Server Database Engine в формате XML, а при подключении к экземплярам SQL Server 2000 — в текстовом виде.
    - Среда изменения текста с развитыми возможностями, поддерживающая поиск и замену, комментирование блоков, пользовательские шрифты и цвета и нумерацию строк. Некоторые типы редакторов поддерживают дополнительные возможности, такие как структурирование и автозавершение.
    - Режим SQLCMD для выполнения сценариев, содержащих команды операционной системы.

Редакторы запросов содержат следующие окна.

- Редактор запросов. Это окно используется для ввода и выполнения сценариев.
- Результаты. Это окно используется для просмотра результатов выполнения запроса. Результаты в нем могут отображаться в виде текста или табличной сетки.
- Сообщения. В этом окне отображаются сведения о том, как выполнен запрос. Например, в окне «Сообщения» могут выводиться сообщения об ошибках или число возвращенных строк.
- Статистика клиента. В этом окне отображаются сведения о выполнении запроса, сгруппированные по категориям. При выборе пункта **Включить статистику клиента** из меню **Запрос** в ходе выполнения запроса появляется окно **Статистика клиента**. Статистика успешно выполненных запросов приводится вместе со

средними значениями. Чтобы сбросить средние значения, выберите пункт **Сбросить статистику клиента** в меню **Запрос**.

2. Диспетчер конфигурации SQL Server — это средство, предназначенное для управления службами, связанными с SQL Server; для настройки сетевых протоколов, которые используются SQL Server; а также для управления конфигурацией подключений с клиентских компьютеров SQL Server. Диспетчер конфигурации SQL Server представляет собой оснастку консоли управления, доступ к которой можно получить из меню «Пуск» и которую можно добавить в любой экран консоли управления. Консоль управления (mmc.exe) для открытия диспетчера конфигурации SQL Server использует файл SQLServerManager.msc в папке Windows System32. Диспетчер конфигурации SQL Server сочетает в себе функциональные возможности следующих средств SQL Server 2000: программа Server Network Utility, программа Client Network Utility и диспетчер служб.
   Диспетчер конфигурации SQL Server и среда SQL Server Management Studio используют инструментарий WMI для просмотра и изменения некоторых параметров сервера. Инструментарий WMI обеспечивает единообразный интерфейс с API-вызовами, которые управляют операциями с реестром, запрашивающими средства SQL Server, а также улучшенный контроль и управление выбранными SQL-службами оснастки «Диспетчер конфигурации SQL Server».
3. Помощник по настройке ядра СУБД

34. T-SQL. Командные и объектные полномочия. Команды grant и revoke. Примеры.

На уровне MSSQL существует два типа прав доступа (привилегий): объектные и командные. Объектные права доступа определяют, кто может получать доступ и работать с данными в таблицах и представлениях и кто может запускать хранимые процедуры. Командные права доступа определяют, кто может удалять и создавать объекты в базе данных.

Объектные права доступа позволяют контролировать права доступа для таблиц, столбцов таблиц, представлений и хранимых процедур. Существуют следующие типы объектных прав доступа:

Тип объекта	**Возможные команды**
Таблица	SELECT, UPDATE, DELETE, INSERT, REFERENCE
Столбец	SELECT, UPDATE
Представление	SELECT, UPDATE, DELETE, INSERT
Хранимая процедура	EXECUTE

Назначение прав:

```
GRANT
   { ALL [ PRIVILEGES ] | разрешение [ ,...n ] }
   {
      [ ( столбец [ ,...n ] ) ] ON { таблица | представление }
      | ON { таблица | представление } [ ( столбец [ ,...n ] ) ]
      | ON { хранимая_процедура | расширенная_процедура }
      | ON { определенныя_пользователем_функция }
   }
TO учетная_запись [ ,...n ]
[ WITH GRANT OPTION ]
[ AS { группа | роль } ]
```

Параметр WITH GRANT OPTION является необязательным и определяет режим, при котором передаются не только права на указанные действия, но и право передавать эти права другим пользователям. Передавать права в этом случае пользователь может только в рамках разрешенных ему действий.

Отмена прав:

```
REVOKE [ GRANT OPTION FOR ]
   { ALL [ PRIVILEGES ] | разрешение [ ,...n ] }
   {
      [ ( столбец [ ,...n ] ) ] ON { таблица | представление }
      | ON { таблица | представление } [ ( столбец [ ,...n ] ) ]
      | ON { хранимая_процедура | расширенная_процедура }
      | ON { определенныя_пользователем_функция }
   }
{ TO | FROM }
   учетная_запись [ ,...n ]
[ CASCADE ]
[ AS { группа | роль } ]
```

Пример. Общий формат оператора назначения привилегий для объекта типа таблица будет иметь следующий синтаксис:

```
GRANT {[SELECT][,INSERT][,DELETE][,UPDATE (<список столбцов>)]}
  ON <имя_таблицы>
```

TO {<имя_пользователя> | PUBLIC } [WITH GRANT OPTION]

Тогда резонно будет выполнить следующие назначения:

```
GRANT INSERT
  ON Tab1
  TO user2
GRANT SELECT
  ON Tab1
  TO user3
```

Эти назначения означают, что пользователь user2 имеет право только вводить новые строки в отношение Tab1, а пользователь user3 имеет право просматривать все строки в таблице Tab1.

Командные права доступа определяют, кто может выполнять административные действия. Командные права могут быть назначены только системным администраторам, пользователям, которым назначена роль sysadmin, или владельцам баз данных. Ниже приведены командные права доступа, которые можно предоставить или аннулировать:

- CREATE DATABASE - право создания базы данных.
- CREATE DEFAULT - право создания стандартного значения для столбца таблицы.
- CREATE PROCEDURE - право создания хранимой процедуры.
- CREATE RULE - право создания правила для столбца таблицы.
- CREATE TABLE - право создания таблицы.
- CREATE VIEW - право создания представления.
- BACKUP DATABASE - право создания резервной копии базы данных.
- BACKUP TRANSACTION - право создания резервной копии журнала транзакций.

Назначение прав:

```
GRANT { ALL | оператор [ ,...n ] }
TO учетная_запись [ ,...n ]
```

Отмена прав:

```
REVOKE { ALL | оператор [ ,...n ] }
TO учетная_запись [ ,...n ]
```

35. T-SQL. Добавление, удаление и обновление данных в представлении. Примеры.

Представление – это виртуальная таблица, определяемая запросом, содержащим оператор SELECT. Эта виртуальная таблица состоит из данных одной или нескольких реальных таблиц, а для пользователей представление выглядит, как реальная таблица. И действительно, с представлением можно работать, как с обычной таблицей. Пользователи могут обращаться к этим виртуальным таблицам в операторах T-SQL таким же образом, как и к таблицам. К представлению можно применять операции SELECT, INSERT, UPDATE и DELETE.
На самом деле представление хранится просто как заранее определенный оператор SQL. При доступе к представлению оптимизатор запросов SQL Server объединяет текущий выполняемый оператор SQL с запросом, который был использован для определения данного представления.
Типы представлений:

- **Подмножество колонок таблицы.** Представление может состоять из одной или нескольких колонок таблицы. Видимо, это наиболее распространенный тип представления, который можно применять для упрощения или безопасности данных.
- **Подмножество строк таблицы.** Представление может содержать любое нужное количество строк. Этот тип представления также полезен для обеспечения безопасности.
- **Связывание двух и более таблиц.** Вы можете создать представление с помощью операции связывания (join). Сложные операции связывания можно упростить, если использовать для этого представление.
- **Агрегированная информация.** Вы можете создать представление, содержащее агрегированные данные. Этот тип представления также используется для упрощения сложных операций.

Представления можно также использовать для объединения секционированных данных. Данные большой таблицы можно секционировать на несколько меньших таблиц, чтобы облегчить управление этими данными, а затем с целью упрощения доступа можно использовать представления для слияния этих таблиц в одну более крупную виртуальную таблицу.
Одним из преимуществ использования представлений является то, что они всегда содержат самые свежие данные. Оператор SELECT, определяющий представление, выполняется только при доступе к этому представлению, поэтому все изменения, внесенные в базовые таблицы представления, отражаются в этом представлении.
Ограничения представлений:

- **Ограничения по колонкам.** Представление может использовать до 1024 колонок таблицы. Если вам требуется ссылка на большее число колонок, то придется использовать какой-либо другой метод.
- **Ограничение базы данных.** Представление можно создать по таблице только в той базе данных, к которой осуществляет доступ создатель представления.
- **Ограничение безопасности.** Создатель представления должен иметь доступ ко всем колонкам, входящим в это представление.
- **Правила целостности данных.** Любые обновления, модификации и т.п., вносимые в представление, не могут нарушать правил целостности данных. Например, если базовая таблица не допускает null-значений, то они также не допускаются этим представлением.
- **Ограничение на количество уровней вложенности представлений.** Представления могут формироваться на основе других представлений – иными словами, вы можете создать представление, имеющее доступ к другим представлениям. Допускается до 32 уровней вложенности представлений.
- **Ограничение оператора SELECT.** Используемый для представления оператор SELECT не может содержать оператора ORDER BY, COMPUTE или COMPUTE BY или ключевого слова INTO.

Создание представлений

Оператор CREATE VIEW имеет следующий синтаксис:

CREATE VIEW имя_представления [(колонка, колонка ...)]
[WITH ENCRYPTION]

AS
ваш оператор SELECT
[WITH CHECK OPTION]

Ключевое слово WITH ENCRYPTION указывает, что определение представления (оператор SELECT, определяющий представление) должно шифроваться. SQL Server использует для шифрования операторов SQL тот же метод, что и для паролей. Этот метод обеспечения безопасности может оказаться полезным, если вы не хотите, чтобы определенные классы пользователей знали, к каким таблицам осуществляется доступ.

Ключевое слово WITH CHECK OPTION указывает, что операции модифицирования данных, применяемые к представлению, должны отвечать критериям, содержащимся в операторе SELECT. Например, можно запретить операцию модифицирования данных, применяемую к представлению для создания строки таблицы, которая не видна внутри этого представления.

Пример. Выбор подмножества колонок из таблицы Employee.

```
CREATE VIEW emp_vw
AS
        SELECT name,
                phone,
                office
        FROM    Employee
```

Пример. Выбор подмножества строк, где Dept = 1.

```
CREATE VIEW emp_vw2
AS
        SELECT  *
        FROM     Employee
         WHERE  Dept = 1
```

Пример. Связывание таблиц Employee2 и MAnager в одну виртуальную таблицу значением mAnager.id:

```
CREATE VIEW org_chart

AS
        SELECT          Employee.ename, MAnager.mname
        FROM            Employee, MAnager
        WHERE           Employee.manager_id = mAnager.id
        GROUP  BY       MAnager.mname, Employee.ename
```

Использование T-SQL для изменения и удаления представлений

Для изменения представлений с помощью T-SQL используйте оператор **ALTER VIEW**.

```
ALTER VIEW имя_представления [(колонка, колонка, ...)]
[WITH ENCRYPTION]
AS
ваш оператор SELECT
[WITH CHECK OPTION]
```

Для удаления представления используйте оператор **DROP VIEW**.

```
DROP VIEW имя_представления
```

36. Тенденции развития СУБД. Понятие ООСУБД, принципы и проблемы реализации.

Объектно-ориентированная СУБД — реализующая объектно-ориентированный подход. Эта система управления обрабатывает данные как абстрактные объекты, наделённые свойствами, в виде неструктурированных данных, и использующие методы взаимодействия с другими объектами окружающего мира.
Пример Объектно-ориентированной СУБД:

- IBM Lotus Notes/Domino
- Jasmine
- ObjectStore
- Caché
- СООБЗ Cerebrum

К основным описательным моментам, связанным с ООБД, в литературе относят:
- объекты (в ООБД любая сущность – объект и обрабатывается как объект); отметим, что здесь используется понятие «объект» объектно-ориентированного программирования;
- классы (понятие «тип данных» из реляционной модели заменяется понятиями «класс» и «подкласс»);
- наследование (классы образуют иерархию наследования, заимствуя свойства друг друга);
- атрибуты (характеристики объекта моделируются его атрибутами);
- сообщения и методы (каждый класс имеет определенную совокупность методов, классы взаимодействуют друг с другом посредством механизма сообщений);
- инкапсуляция (внутренняя структура объектов скрыта);
- идентификаторы объектов – дескрипторы.

Объектно-ориентированная парадигма.

Общепринятого определения "объектно-ориентированной модели данных" не существует. Сейчас можно говорить лишь о неком "объектном" подходе к логическому представлению данных и о различных объектно-ориентированных способах его реализации.
Мы знаем, что любая модель данных должна включать три аспекта: структурный, целостный и манипуляционный. Посмотрим, как они реализуются на основе объектно-ориентированная парадигмы программирования:

Структура:

Структура объектной модели описываются с помощью трех ключевых понятий:

- **инкапсуляция** - каждый объект обладает некоторым внутренним состянием (хранит внутри себя запись данных), а также набором методов - процедур, с помощью которых (и *только* таким образом) можно получить доступ к данным, определяющим внутреннее состояние объекта, или изменить их. Таким образом, объекты можно рассматривать как самостоятельные сущности, отделенные от внешнего мира.
- **наследование** - подразумевает возможность создавать из классов объектов новые классы объекты, которые наследуют структуру и методы своих предков, добавляя к ним черты, отражающие их собственную индивидуальность. Наследование может быть простым (один предок) и множественным (несколько предков).
- **полиморфизм** - различные объекты могут по разному реагировать на одинаковые внешние события в зависимости от того, как реализованы их методы.

Целостность данных:

Для поддержания целостности объектно-ориентированный подход предлагает использовать следующие средства:

- автоматическое поддержание отношений наследования
- возможность объявить некоторые поля данных и методы объекта как "скрытые", не видимые для других объектов; такие поля и методы используются только методами самого объекта
- создание процедур контроля целостности внутри объекта

Средства манипулирования данными:

К сожалению, в объектно-ориентированном программировании отсутствуют общие средства манипулирования данными, такие как реляционная алгебра или реляционное счисление. Работа с

данными ведется с помощью одного из объектно-ориентированных языков программирования общего назначения, обычно это SmallTalk, C++ или Java.

Подведем теперь некоторые итоги:

В объектно-ориентированных базах данных, в отличие от реляционных, хранятся не записи, а объекты. ОО-подход представляет более совершенные средства для отображения реального мира, чем реляционная модель:

- естественное представление данных. В реляционной модели все отношения принадлежат одному уровню, именно это осложняет преобразование иерархических связей модели "сущность-связь" в реляционную модель. ОО-модель можно рассматривать послойно, на разных уровнях абстракции.
- имеется возможность определения новых типов данных и операций с ними.

В то же время, ОО-модели присущ и ряд недостатков:

- осутствуют мощные непроцедурные средства извлечения объектов из базы. Все запросы приходится писать на процедурных языках, проблема их оптимизации возлагается на программиста.
- вместо чисто декларативных ограничений целостности (типа явного объявления первичных и внешних ключей реляционных таблиц с помощью ключевых слов PRIMARY KEY и REFERENCES) или полудекларативных триггеров для обеспечения внутренней целостности приходится писать процедурный код.

Очевидно, что оба эти недостатка связаны с отсутствием развитых средств манипулирования данными. Эта задача решается двумя способами - расширение ОО-языков в сторону управления данными (стандарт ODMG), либо добавление объектных свойств в реляционные СУБД (SQL-3, а также так называемые объектно-реляционных СУБД).

37. Тенденции развития СУБД. ОРСУБД. Принципы и проблемы реализации . Пример.

Объектно-реляционная СУБД (ОРСУБД) — реляционная СУБД (РСУБД), поддерживающая некоторые технологии, реализующие объектно-ориентированный подход.
Разница между объектно-реляционными и объектными СУБД: первые являют собой надстройку над реляционной схемой, вторые же изначально объектно-ориентированы. Главная особенность и отличие объектно-реляционных, как и объектных, СУБД от реляционных заключается в том, что О(Р)СУБД интегрированы с Объектно-Ориентированным (ОО) языком программирования, внутренним или внешним как C++, Java. Характерные свойства ОРСУБД - 1) комплексные данные, 2) наследование типа, и 3) объектное поведение.
Комплексные данные могут быть реализованы через постоянно-хранимые объекты (persistent objects). Создание комплексных данных в большинстве существующих ОРСУБД основано на предварительном определении схемы через определяемый пользователем тип (UDT - user-defined type). Используются также встроенные конструкторы составных типов, например массив (ARRAY).
Иерархия структурных комплексных данных предлагает дополнительное свойство, **наследование типа**. То есть структурный тип может иметь подтипы, которые используют все его атрибуты и содержат дополнительные атрибуты, специфицированные в подтипе.
Объектное поведение закладывается через описание программных объектов. Такие объекты должны быть сохраняемыми и переносимыми для обработки в базе данных, поэтому они называются обычно как постоянные (или долговременные) объекты. Внутри базы данных все отношения с постоянным программным объектом есть отношения с его объектным идентификатором (OID).
Объектно-реляционными СУБД являются, к примеру, широко известные Oracle Database, Microsoft SQL Server 2005, PostgreSQL, а также Sav Zigzag, IBM Cloudscape, FirstSQL/J.

Общая идея заключается в применении гибких подходов, позволяющих не ограничивать собственные возможности догматическими положениями, а, напротив, применяя различные приемы, адаптироваться к любой возникшей ситуации. Именно эту идею и проповедуют разработчики объектно-реляционных баз данных. Так, соответствующие базы данных соединяют в себе лучшие качества реляционных и объектно-ориентированных баз данных. Рассмотрим кратко основные элементы, осуществляющие объектные расширения в реляционных базах данных, уже используемые на сегодняшний день:

- Хранение больших объемов данных. Наряду с теми данными, которые хранились в БД традиционно, современные объектно-реляционные базы данных позволяют хранить в столбцах таблицы картинки, видеоролики и другие большие документы.

- Списки в столбцах. Более того, объектно-реляционные базы данных позволяют хранить в столбцах целые списковые структуры.

- Пользовательские расширения. В объектно-реляционных базах данных пользователи имеют возможность вмешиваться в изначально предоставляемый СУБД инструментарий, создавая, в частности, новые пользовательские типы данных.

- Хранимые процедуры. В определенном смысле хранимые процедуры также являются объектным расширением, осуществляя необходимые пользователю воздействия на данные (стандартный для ООП процедурный подход).

38. Понятие OLAP и OLTP системы. Принципы и реализации многомерных СУБД.

В последнее время все более популярными становятся **системы оперативной аналитической обработки (OLAP - OnLine Analytical Processing)**. Системы OLAP позволяют значительно улучшить качество анализа баз данных. Особенно большое значение это имеет в бизнесе, где необходимо оперативно принимать решение о наиболее перспективных направлениях развития производства. Фундаментальное отличие системы OLAP от обычной базы данных заключается в следующем:

· система OLAP характеризуется статичностью данных, можно сказать, что база данных используется только для чтения, что позволяет более компактно располагать данные на страницах базы данных, увеличив плотность записи;

· для проведения эффективного анализа в системе OLAP обычно создается множество индексов, ускоряющих проведение анализа и выборки данных;

· система OLAP должна выполнять базовые операции численного и статистического анализа данных; кроме того, необходимо реализовать многомерное представление информации, что позволяет более наглядно представлять структуру данных;

· для более эффективного анализа данных должна поддерживаться возможность создания материализованных представлений, что позволяет резко увеличить производительность выполнения типичных запросов;

· система OLAP объединяет данные из множества источников (например, из разных баз данных, нередко расположенных на разных серверах с различной архитектурой SQL Server и Oracle) и предоставляет их пользователям в логически завершенной форме;

· система OLAP должна обеспечивать сохранность конфиденциальных данных, обеспечивая совместный доступ к информации множеству пользователей.

Корпорация Microsoft предлагает мощный инструмент для поддержки систем принятия решений - Microsoft Decision Support Services, являющийся полноценной реализацией системы OLAP. Microsoft DSS поставляется как отдельный компонент в составе SQ:L Server и реализован в виде отдельной службы операционной системы, оптимизирующей исполнение запросов, не изменяющих данные.

OLAP играет ключевую роль при построении хранилищ данных. Использование MS DSS при создании баз данных позволяет реализовать базовые функции для широкого спектра приложений. Возможность построения больших распределенных баз данных, оперативный анализ их содержимого, интеграция множества источников данных с помощью технологии OLE DB делают привлекательным построение корпоративных баз данных и хранилищ на основе серверов SQL Server.

Другой вариант использования SQL Server -это построение **систем управления обработкой транзакций (OLTP - OnLine Transaction Processing).** В противоположность системам OLAP системы OLTP характеризуются большим количеством изменений в базе данных. Множество пользователей одновременно обращаются к записям в базе данных, выполняя их чтение, добавление, удаление или изменение. Причем несколько пользователей могут одновременно пытаться изменить одну и ту же запись. База данных должна быть построена как система OLTP, если требуется реализация одного из следующих аспектов работы:

· одновременный доступ; система OLTP должна гарантировать, что только один пользователь в конкретный момент времени сможет изменять данные;

· целостность изменений; все выполняемые в базе данных изменения выполняются в виде транзакций; система OLTP гарантирует, что все включенные в транзакцию шаги будут выполнены как единое целое.

Системы OLTP характеризуются длительным блокированием данных, поэтому необходимо реализовать возможность резервного копирования. Узким местом систем OLTP являются операции диского ввода-вывода. Большое количество изменений данных требует высокой скорости обмена между оперативной памятью и жестким диском. Для повышения производительности можно использовать дисковые массивы или группы файлов, размещенные на разных физических дисках.

Многомерные базы данных (Multi-value Database)

Многомерные базы данных — технология, которая длительное время воспринималась как новинка, — сегодня является решением, которое предлагает не только высокую производительность и простоту использования, но и обеспечивает возможности, необходимые для разработки, расширения и быстрого развертывания бизнес-приложений при сокращении ИТ-затрат. Системы на основе многомерных баз данных идеально подходят для потребностей как для рынков среднего и малого бизнеса (SMB), так и крупных предприятий.

Многомерные базы данных отличаются от реляционных прежде всего трехмерностью — поддержкой неограниченного числа значений в поле, и находят свое применение там, где необходима эффективная и простая работа с большими массивами символьной информации. В многомерных СУБД данные организованы в виде упорядоченных многомерных массивов, удовлетворяющих требованиям защиты от несанкционированного доступа в организации. Они обеспечивают более быструю реакцию на запросы данных за счет того, что обращения поступают к относительно небольшим блокам данных, необходимых для конкретной группы пользователей. Для достижения сравнимой производительности реляционные системы требуют тщательной проработки схемы базы данных, определения способов индексации и специальной настройки. Ограничения SQL остаются реальностью, что не позволяет реализовать в реляционных СУБД многие встроенные функции, легко обеспечиваемые в системах основанных на многомерном представлении данных.

Основные преимущества многомерных СУБД

- Общая простота системы, что позволяет осуществлять быстрое встраивание технологий многомерных СУБД в приложения. Системы на основе многомерных баз данных требуют меньше специальных навыков по разработке и администрированию;
- Относительно низкая общая стоимость владения, а также быстрый возврат инвестиций;
- В случае использования многомерных СУБД поиск и выборка данных осуществляется значительно быстрее, чем при многомерном концептуальном взгляде на реляционную базу данных, так как многомерная база данных обеспечивает оптимизированный доступ к запрашиваемым ячейкам;
- Многомерные СУБД легко справляются с задачами включения в информационную модель разнообразных встроенных функций, тогда как объективно существующие ограничения языка SQL делают выполнение этих задач на основе реляционных СУБД достаточно сложным, а иногда и невозможным.

Наиболее коммерчески успешными из известных программных продуктов, основанных на многомерных технологиях, являются СУБД UniVerse компании Rocket Software и СУБД jBASE одноименной компании jBASE International.

39 Распределенные СУБД. Основные принципы реализации.

Распределенная база данных — это набор файлов (отношений), хранящихся в разных узлах информационной сети и логически связанных таким образом, чтобы составлять единую совокупность данных (связь может быть функциональной или через копии одного и того же файла). Распределенная БД предполагает хранение и выполнение функций управления данными в нескольких узлах и передачу данных между этими узлами в процессе выполнения запросов. Разбиение данных в распределенной базе данных может достигаться путем хранения различных таблиц на разных компьютерах или даже хранения разных частей и фрагментов одной таблицы на разных компьютерах. Для пользователя (или прикладной программы) не должно иметь значения, каким образом распределены данные между компьютерами. Работать с распределенной базой данных, если она действительно распределенная, следует так же, как и с централизованной, т. е. размещение базы данных должно быть прозрачно.
Несмотря на то, что распределенная база данных состоит из нескольких локальных баз данных, у пользователя должна сохраняться иллюзия работы с централизованной базой данных, что вызывает потребность в использовании некоторого общего представления о данных — **глобальной концептуальной схемы**. Определение данных в такой концептуальной схеме должно быть аналогичным определению в централизованной базе данных.
Отличия начинаются, когда требуется хранить данные в нескольких узлах. Чтобы произвести разбиение данных, нужно секционировать таблицы глобальной схемы на фрагменты. Существует два типа секционирования**: горизонтальное и вертикальное**. При секционировании таблицы по строкам выполняется горизонтальное секционирование, при разбиении по столбцам — вертикальное.
Таким образом, архитектура распределенной СУБД должна содержать информацию о секционировании исходных таблиц базы данных, что предполагает создание **дополнительного уровня — фрагментного.**
Для реализации и объяснения распределенной природы базы данных выделяются два уровня: фрагментный и уровень распределенного представления. Последний показывает географическое распределение данных по рабочим станциям, расположение экземпляра каждого фрагмента.

Дейт установил **12 свойств** или качеств идеальной распределенной базы данных:

1. Локальная автономия (local autonomy) – управление данными на каждом из узлов распределенной системы выполняется локально.
2. Независимость узлов (no reliance on central site) – в идеальной системе все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных.
3. Непрерывность операции (continuous operation) – возможность непрерывного доступа к данным в рамках DDB вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах.
4. Прозрачность расположения (location independence) – пользователь, обращающийся к DDB, ничего не должен знать о реальном, физическом размещении данных в узлах информационной системы.
5. Прозрачная фрагментация (fragmentation independence) – возможность распределенного (то есть на различных узлах) размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: горизонтальная и вертикальная. Первая означает хранение строк таблицы на различных узлах (фактически, хранение строк одной логической таблицы в нескольких идентичных физических таблицах на различных узлах). Вторая означает распределение столбцов логической таблицы по нескольким узлам.
6. Прозрачное тиражирование (асинхронный в общем случае процесс переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы) (replication independence) – тиражирование возможно и достигается внутрисистемными средствами.

7. Обработка распределенных запасов (distributed query processing) – возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке SQL.
8. Обработка распределенных транзакций (distributed transaction processing) – возможность выполнения операций обновления распределенной базы данных (INSERT, UPDATE, DELETE), не разрушающего целостность и согласованность данных. Эта цель достигается применением двухфазового или двухфазного протокола фиксации транзакций (two-phase commit protocol), ставшего фактическим стандартом обработки распределенных транзакций.
9. Независимость от оборудования (hardware independence) – в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей — от мэйнфреймов до персоналок.
10. Независимость от операционных систем (operationg system independence) – вытекает из предыдущего и означает многообразие операционных систем, управляющих узлами распределенной системы.
11. Прозрачность сети (network independence) – спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными базами данных; в распределенной системе возможны любые сетевые протоколы.
12. Независимость от баз данных (database independence) – в распределенной системе могут мирно сосуществовать СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов.

Исходя из определения Дэйта, СУБД в общем случае можно рассматривать как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечивается от различных поставщиков. Связи между узлами — это потоки тиражируемых данных. Топология DDB варьируется в широком диапазоне: возможны варианты иерархии, структур типа звезда и т. д. В целом топология DDB определяется географией информационной системы и направленностью потоков тиражирования данных.

40. Общая характеристика и возможности системы.

40.1. Назначение Access

Иногда Access (файл MDB) используется просто как ядро, которое управляет данными, находящимися с таблицами. Пользователи работают с этими данными через внешние приложения, созданные разработчиками, например, на Visual Basic, Delphi или C++. В других ситуациях Access, наоборот, используется только для предоставления пользовательского интерфейса для работы с данными, которые физически расположены на серверах баз данных, например, SQL Server, Oracle, IBM D2 и т.п.

Access –это комплекс программных средств, предназначенных для создания структуры новой базы данных, наполнения её содержимым, редактирования содержимого, отбора данных в соответствии с заданными критериями, их упорядочивания, оформления, печати.
Access работает под управлением *Windows* и поэтому может использовать все возможности DDE и OLE.
(DDE
Dynamic Data Exchange [DDE][динамический обмен данными] - механизм взаимодействия приложений в операционных системах Microsoft Windows и OS/2.
OLE
Object Linking and Embedding [OLE][связывание и внедрение объектов] - технология связывания и внедрения объектов в другие документы и объекты, разработанные корпорацией Microsoft.)
Для осуществления динамического обмена данными с другими приложениями можно использовать макросы или процедуры на **VBA**, **Visual Basic для приложений**.
В **Access** база данных обозначает файл, содержащий набор информации. Каждая база данных в **Access** состоит из основных объектов: *таблиц, запросов, форм, отчётов, страниц, макросов* и *модулей,* которые хранятся в одном файле с расширением mdb.
Access может работать одновременно только с одной базой данных.
ТАБЛИЦА – это объект, определяемый и используемый для хранения данных. Каждая таблица включает информацию определённого типа. *Таблица* содержит поля (столбцы), в которых хранятся данные и записи (строки). В записи собрана вся информация о конкретном объекте. Для каждой *таблицы* можно определить первичный ключ и один или несколько индексов с целью увеличения скорости доступа к данным.
Access позволяет изменить структуру *таблицы*, просматривать, редактировать, удалять и добавлять записи, осуществлять поиск, замену, сортировку данных, изменять вид *таблицы*, создавать связь между таблицами и удалять их.
ЗАПРОС – это объект, который позволяет пользователю получить данные из одной или несколько таблиц.
ФОРМЫ – это объект, предназначенный для просмотра, ввода и редактирования записей базы данных.
ОТЧЁТ – это объект, предназначенный для создания документа, который впоследствии может бать распечатан либо включён в документ другого приложения.
СТРАНИЦЫ – это объект, представляющий собой специальный тип *Web-страниц,* предназначенный для просмотра и работы через Интернет или интрасеть с данными.
МАКРОС – это объект, представляющий собой последовательность макрокоманд для автоматизации наиболее часто выполняемых действий при работе с базой.
МОДУЛЬ – это объект, автоматизирующий комплексные операции и предоставляющий программисту более полный контроль, чем *макрос*.

41. Способы представления информации. Примеры.

40.2. Режима работы с таблицами

В новой версии Microsoft Access существуют четыре режима работы с таблицами: режим Таблицы (Datasheet View), режим Конструктора (Design View), режим Сводной таблицы (PivotTable View) и режим Сводной диаграммы (PivotChart View). Существует также дополнительный режим — режим Предварительного просмотра, который позволяет увидеть расположение данных на листе перед осуществлением печати таблицы.

40.3. Режим Таблицы

В режиме Таблицы осуществляется работа с данными, находящимися в таблице: просмотр, редактирование, добавление, сортировка и т. п.

В верхней части таблицы располагаются имена *полей,* ниже следуют *записи,* в которые вносятся данные. Одна запись всегда является *текущей,* и рядом с ней расположен *указатель текущей записи.* В нижней части окна расположены *кнопки навигации,* позволяющие перемещать указатель текущей записи по таблице. Там же находятся поле номера текущей записи, кнопка создания новой записи и указатель общего количества записей в таблице.

40.4. Режим Конструктора

В режиме Конструктора создается или модифицируется структура таблицы, т. е. задаются имена полей таблицы и их типы, поля описываются, задаются их свойства.
Ключевое поле таблицы помечается специальным значком — ключик в поле выделения в левой части окна. При выделении поля в нижней части окна будут показаны параметры именно для этого поля.

40.5. Режим Сводной таблицы

В режимах Сводной таблицы и Сводной диаграммы удобно выполнять анализ данных, динамически изменяя способы их представления.

Новый режим — Сводной таблицы — позволяет представлять табличные данные в более удобном и обозримом виде. Сводная таблица позволяет группировать, суммировать или каким-то другим образом обрабатывать данные из обычной таблицы Access. Этот режим является с одной стороны аналогом сводных таблиц в Excel, а с другой стороны — развитием уже давно используемых в Access перекрестных запросов.

40.6. Режим Сводной диаграммы

Режим Сводной диаграммы тесно связан со сводной таблицей. Это просто графическое представление сводной таблицы. В предыдущих версиях Access диаграммы могли использоваться только в отчетах. Теперь они стали динамическими, и есть возможность создавать эти диаграммы и управлять ими в интерактивном режиме прямо на экране монитора.

42. Структура объектов системы и их классификация. Примеры.

Таблицы: Таблицы играют ключевую роль в базах данных, поскольку именно в них хранится информация. База данных может содержать тысячи таблиц, размеры которых ограничиваются только доступным пространством на жестком диске компьютера. Для таблиц обычно используются режим таблицы, предназначенный для ввода данных, и режим конструктора, позволяющий просмотреть и модифицировать структуру таблицы.
Из всех типов объектов только таблицы предназначены для хранения информации. Остальные используются для просмотра, редактирования, обработки и анализа данных - иначе говоря, для обеспечения эффективного доступа к информации.

Запросы: Запросы предназначены для поиска в базе данных информации, отвечающей определенным критериям. Найденные записи, называемые результатами запроса, можно просматривать, редактировать и анализировать различными способами. Кроме того, результаты запроса могут использоваться в качестве основы для создания других объектов Access. В сущности, запрос представляет собой вопрос, сформулированный в терминах базы данных. Существует различные типы запросов. Наиболее распространенными являются запросы на выборку, параметрические и перекрестные запросы. Для создания простых запросов используется мастер, в менее тривиальных случаях можно создать запрос вручную в режиме конструктора.

Формы: Информация хранится в таблицах в том виде, в котором была введена. Формы позволяют упростить и сделать более эффективными ввод и обработку содержимого таблиц. В сущности, форма представляет собой окно, куда можно поместить элементы управления, предназначенные для ввода и отображения данных. Access включает панель, которая содержит многие стандартные элементы управления Windows, в том числе поля, надписи, флажки и кнопки выбора. Формы выглядят и функционируют примерно так же, как диалоговые окна в приложениях Microsoft Windows.Формы используются для ввода и редактирования записей в таблицах базы данных. Подобно таблицам и запросам, их можно отображать в трех режимах: в режим формы, предназначенном для ввода данных, в режиме таблицы, где данные представлены в табличном формате, и в режиме конструктора, позволяющем изменить внешний вид, содержание и функционирование формы..

Отчеты: Отчеты используются для отображения информации, содержащейся в таблицах, в отформатированном виде, который легко читается как на экране компьютера, так и на бумаге. Помимо данных, извлеченных из нескольких таблиц и запросов, отчеты могут включать элементы оформления, свойственные печатным документам, как, например, названия, заголовки и колонтитулы. Отчет можно отобразить в трех режимах: в режиме конструктора, позволяющем изменить внешний вид и макет отчета, в режиме просмотра образца, где можно просмотреть все элементы готового отчета, но в сокращенном виде, и в режиме предварительного просмотра, где отчет отображается в том виде, в каком будет напечатан.

Страницы : Чтобы предоставить доступ к информации, хранящейся в базе данных, пользователям Интернета или интранета, можно создать страницы, называемые страницами доступа к данным. Работа с данными на странице доступа в Web осуществляется примерно так же, как в Access - пользователи могут просматривать таблицы, выполнять запросы и заполнять поля форм. Хотя публикация информации из базы данных в Web на первый взгляд кажется сложной, Access включает мастер, которые берет на себя большую часть кропотливой работы по созданию страницы доступа. При желании созданную мастером страницу можно доработать в режиме конструктора.

Макросы: Макросы представляют собой небольшие программы, с помощью которых обеспечивается реакция Access на такие события, как открытие формы, щелчок кнопки или обновление записи. Это особенно удобно, если предполагается передать базу данных неквалифицированным пользователям. Например, можно написать макросы, содержащие последовательность команд, выполняющих рутинные задачи, или связать такие действия, как открытие формы или печать отчета, с кнопками кнопочной формы.

Модули: Модули представляют собой программы на Visual Basic for Applications (VBA), языке программирования высокого уровня, разработанного Microsoft для создания приложений Windows. Помимо стандартного набора команд VBA, каждая программа Microsoft Office имеет собственные команды. В отличие от макросов, позволяющих автоматизировать не более пяти, шести десятков операций, VBA включает сотни команд и может неограниченно расширяться за счет дополнений, вносимых другими компаниями и частными лицами. Программы VBA используются для решения задач, слишком сложных для макросов, как, например, извлечение определенной информации из рабочих листов Excel.

43. Средства создания и коррекции структуры базы данных. Примеры.

40.7. Создание базы данных

Для создания базы данных в Microsoft Access можно использовать два способа. Простейший способ создания базы данных - использование мастера баз данных для создания всех необходимых таблиц, форм и отчетов. Имеется также возможность создать пустую базу данных, а затем добавить в нее таблицы, формы, отчеты и другие объекты - это наиболее гибкий способ, но он требует отдельного определения каждого элемента базы данных. В обоих случаях созданную базу данных можно в любое время изменить и расширить.

40.8. Сохранение информации о структуре

Логичнее будет не писать для каждой структуры базы данных свои функции по созданию таблиц в соответствии с данным алгоритмом, а использовать общие функции. Тогда общие функции должны откуда-то брать информацию о структуре базы данных: таблицах, столбцах и их свойствах, индексах, связях. Можно хранить эту информацию в таблицах интерфейсной части приложения. Заполнять таблицы информацией о структуре базы данных программно по следующему алгоритму:

-для каждой таблицы базы данных: записать в таблицу имена столбцов;
- для каждого столбца при этом записать в таблицу значения всех его (столбца) свойств;
- записать в таблицу информацию об индексах;
- для каждого индекса записать в таблицу информацию о столбцах;
- записать в таблицу информацию о связях;
- для каждой связи записать информацию о столбцах.

Таким образом, мы сохранили информацию о структуре базы данных и можем использовать эту информацию многократно.

Итак, для сохранения информации о структуре базы данных в таблицах, рассмотренных выше, необходимо запустить процедуру faGetStructure при этом, передав ей в качестве параметра полное имя файла базы данных, структуру которой необходимо сохранить.
Данная процедура использует вспомогательную процедуру faGetTableList в случае, если таблица Structure окажется пустой. Вспомогательная процедура заполнит данную таблицу названиями всех таблиц из сохраняемой базы данных. Можно использовать данную особенность для того, чтобы сохранить информацию не о всей структуре базы данных, а только о части, представленной определёнными таблицами, для этого нужно вручную заполнить таблицу Structure названиями таблиц сохраняемой структуры.
Процедура faGetStructure для каждого наименования таблицы из Structure сохранит в таблицах информацию о столбцах и их свойствах таблиц, индексов, ключей.
Таким образом, мы сохранили информацию о структуре базы данных и можем использовать эту информацию многократно. Если структура базы данных поменяется, то достаточно удалить всю информацию из таблицы Structure и заново вызвать процедуру faGetStructure.

40.9. Коррекция сохранённой структуры

Сохранённая информация о структуре требует коррекции, иначе могут возникнуть ошибки при создании новой базы данных. Коррекция заключается в удалении из таблицы PropertiesT свойств столбцов таблицы всех полей, относящихся к свойствам с именами “Seed” (значение идентификатора для первой строки таблицы) и “Increment” (приращение для столбца идентификатора), т.к. эти свойства невозможно установить программно. Для проведения этой коррекции можно использовать такой запрос:

```
DELETE PropertiesT.Name
FROM PropertiesT
WHERE (((PropertiesT.Name)="Seed" Or (PropertiesT.Name)="Increment"));
```

Также требуется подкорректировать значения свойства “Nullable” у столбцов с логическим типом данных: оно должно быть всегда равно нулю. Для этой цели можно использовать такой запрос:

UPDATE ColumnsT INNER JOIN PropertiesT ON ColumnsT.ID = PropertiesT.ColumnT SET PropertiesT.[Value] = "0"
WHERE (((ColumnsT.Type)=11) AND ((PropertiesT.Name)="Nullable"));
Также эмпирическим путём выяснилось, что для свойства “Fixed Length” («фиксированная длина») ключевых полей с числовым типом данных не может быть значение “False” («ложь»), хотя по неизвестным причинам при сохранении такое значение этого свойства записывается. Поэтому для данного случая также требуется коррекция. В виде запроса эта коррекция выглядит следующим образом:
UPDATE ColumnsT INNER JOIN PropertiesT ON ColumnsT.ID = PropertiesT.ColumnT SET PropertiesT.[Value] = "-1"
WHERE (((PropertiesT.Value)="0") AND ((ColumnsT.Type)=3) AND ((ColumnsT.Key)=True) AND ((PropertiesT.Name)="Fixed Length"));
Само собой разумеется, что значение свойства “JET OLEDB:Allow Zerro Length” («допускается нулевая длина») для ключевых полей с числовым типом данных не может быть “True” («истина»), но и такое бывает, поэтому и для этого варианта предусматриваем коррекцию в виде запроса:
UPDATE ColumnsT INNER JOIN PropertiesT ON ColumnsT.ID = PropertiesT.ColumnT SET PropertiesT.[Value] = "0"
WHERE (((PropertiesT.Value)="-1") AND ((ColumnsT.Type)=3) AND ((ColumnsT.Key)=True) AND ((PropertiesT.Name)="Jet OLEDB:Allow Zero Length"));

40.10. **Создание новой БД по сохранённой структуре**

Теперь можно создавать неограниченное число раз чистых копий баз данных, аналогичных по структуре исходной, информация о которой была сохранена и скорректирована.
Для создания новой пустой базы данных сохранённой структуры нужно вызвать процедуру faCreateStructure и передать ей в качестве параметра полное имя файла создаваемой базы данных. Сначала эта процедура создаёт новую базу данных как каталог ADOX, затем, используя информацию из таблиц, создаёт каждую таблицу, столбцы и заполняет их свойства. Для заполнения свойств столбцов используется вспомогательная процедура faSetPropertiesC. Данная процедура заполняет свойства, согласуясь с типом самого свойства.
Далее процедура создаёт и заполняет свойства индексов и ключей таблиц базы данных.
В результате получаем копию сохранённой базы данных, но без данных.

44. Организация обработки данных. Примеры.

Для обработки таблиц Microsoft Access использует мощный язык SQL (Structured Query Language) - структурированный язык запросов. Access значительно упрощает задачу обработки данных. При любой обработке данных из нескольких таблиц Access использует связи, заданные пользователем при конструировании таблиц. В Access имеется также средство графического задания запроса - "запрос по образцу" (QBE- query by example).

Одним из стандартных объектов Microsoft Access является **запрос**. Запросы используются для просмотра, анализа и изменения данных из одной или нескольких таблиц. Например, можно использовать запрос для отображения данных из одной или нескольких таблиц и отсортировать их в определенном порядке, выполнить вычисления над группой записей, осуществить выборку из таблицы по определенным условиям. В общем случае данные, отраженные в запросах, являются результатом применения разнообразных операций реляционной алгебры над данными. Сам запрос не содержит данных, но позволяет выбирать данные из таблиц и выполнять над ними разнообразные операции. Запросы могут служить источником данных для форм и отчетов Microsoft Access. В Microsoft Access существует также понятие фильтра, который в свою очередь является набором условий, позволяющих отбирать подмножество записей и сортировать их. Основное сходство между запросами на выборку и фильтрами заключается в том, что в них проводится извлечение подмножества записей из базовой таблицы или запроса. Фильтр обычно используется при работе в режиме формы или в режиме таблицы для просмотра или изменения подмножества записей.

Запрос можно использовать:

- для просмотра подмножества записей таблицы без предварительного открытия этой таблицы или формы;
- для объединения в виде одной таблицы на экране данных из нескольких таблиц;
- для просмотра отдельных полей таблицы;
- для выполнения вычислений над значениями полей.

Основные отличия запросов и фильтров заключаются в следующем:

- фильтры не позволяют добавить еще несколько таблиц, записи из которых включаются в возвращаемый набор записей;
- фильтры не позволяют указать поля, которые должны отображаться в результирующем наборе записей;
- фильтры не могут быть выведены как отдельный объект в окне базы данных;
- запросы могут использоваться только с закрытой таблицей, запросом или формой;
- фильтры не позволяют вычислять суммы, средние значения, подсчитывать количество записей и находить другие итоговые значения

Способы создания запросов в Access

- Наиболее просто создается запрос при помощи Мастера запросов.
- Конструктор запросов позволяет создавать новые и изменять существующие запросы.
- В режиме Конструктора можно переключаться в режим Окно SQL, позволяющее создавать SQL запросы.

45. Способы ускорения поиска данных: индексация и сортировка. Примеры.

40.11. Индексация

Индекс – средство, ускоряющее поиск и сортировку в таблицы за счет использования ключевых значений, которое позволяет обеспечить уникальность строк таблицы. Автоматически создаются индексы для полей, на которые назначены ограничение первичного или вторичного ключа. Для остальных столбцов индексы можно создать явно в режиме конструктора для таблицы: свойства поля. Для индексированное поле предусмотрено три варианта: Нет (по умолчанию), Да (допускаются совпадения) и Нет (Совпадения не допускаются).
При индексировании создается отдельный индексный файл, при этом сама база данных остается неизменной.

Индексированные поля на обеих сторонах запроса соединяют (join) или создают связь (relationship) между полями. Jet создает индексы на полях, которые были связаны в окне «Схема данных» (relationships window), если их там еще нет. Это может ускорить выполнение запроса, позволяя оптимизатору запроса использовать более сложную внутреннюю стратегию связывания. Следует как можно больше использовать первичные ключи (primary keys) в противоположность уникальным индексам. Кроме того, если индексируются поля, которые необходимо часто сортировать, быстродействие запроса увеличивается.

40.12. Сортировка

При сортировке изменяется база данных, порядок следования записей.

Операция сортировки данных используется для удобства нахождения нужной информации. По умолчанию, когда таблица открывается в режиме Таблицы, она упорядочивается по значению ключевого поля. Если ключевое поле для таблицы не определено, записи выводятся в порядке их ввода в таблицу. Также можно отсортировать записи по значению любого другого поля.

Таким образом можно упорядочить записи по любому столбцу таблицы. Ограничения существуют только на тип данных, которые можно упорядочивать — нельзя сортировать значения полей типа MEMO, гиперссылки или объекты OLE.

Чтобы правильно применять сортировку, нужно знать несколько простых правил.

- При сортировке в возрастающем порядке записи, содержащие пустые поля (с пустыми значениями), указываются в списке первыми.
- Числа, находящиеся в текстовых полях, сортируются как строки символов, а не как числовые значения. Если нужно отсортировать их в числовом порядке, все текстовые строки должны содержать одинаковое количество символов. Если строка содержит меньшее количество символов, то сначала нужно вставить незначащие нули.
- При сохранении таблицы сохраняется и порядок ее сортировки.
- Иногда нужно выполнить сортировку по значению нескольких полей. Для этого необходимо переместить сначала сортируемые столбцы таким образом, чтобы они оказались, во-первых, рядом, а во-вторых, с учетом приоритетов, — приоритеты устанавливаются слева направо, т. к. первыми будут сортироваться значения в крайнем левом столбце.

46. Способы организации связи между файлами. Примеры.

40.13. Установка связей между таблицами

Создание связей между таблицами - последний этап физического проектирования БД. Связь между таблицами устанавливает отношение между совпадающими значениями в ключевых полях. В большинстве случаев связывают первичный ключ главной таблицы с внешним ключом подчинённой таблицы (часто имеющим то же имя). Существуют четыре типа отношений:

1. Один - к одному. Запись таблицы А может иметь не более одной связанной записи в таблице В и наоборот. Ключевое поле в таких таблицах должно быть уникальным.
2. Многие - к одному. Одной записи в таблице А может соответствовать одна запись в таблице В, а одной записи в таблице В - несколько записей в таблице А. В таблице В ключевое поле должно быть уникальным.
3. Один - ко многим . Каждой записи в таблице А могут соответствовать несколько записей в таблице В, а запись в таблице В не может иметь более одной соответствующей записи в таблице А. В таблице А ключевое поле должно быть уникальным. Отношения вида 2 и 3 отличаются тем, какая таблица является главной.
4. Многие - ко многим. Такая таблица не нормализована. Одной записи таблицы А может соответствовать несколько записей таблицы В и наоборот. Уникальных ключей нет. Все ключи внешние. В этом случае таблицу надо нормализовать, используя инструмент *Анализатор таблиц.*

Связанные поля не обязательно должны иметь одинаковые имена, но обязательно должны иметь одинаковые типы данных.

40.14. Импорт БД и установка связи с таблицами вне БД

Следует импортировать таблицу, если:

- файл небольшой и не подвергается частым изменениям;
- не нужно использовать данные совместно с другими пользователями;
- заменяется старое приложение , но не оставляется прежний формат;
- необходимо выиграть в быстродействии с копией в формате Access.

Следует выбрать установку связи с файлом, если:

- файл достаточно большой (свыше 1 Гб);
- данные часто меняются;
- базу данных необходимо распределить и поместить данные на сетевой сервер.

40.15. Инструмент для установки связей

Если таблицы, с которыми необходимо установить связи, содержатся в базе данных, созданной в одной из новых версий dBASE или Paradox (dBASE 7 или Paradox 8), нужно установить средство Borland Database Engine (BDE) версии *4.x* или выше, чтобы связать таблицы с базой данных Access. Можно установить связь с таблицами базы данных, созданной в более ранних версиях - dBASE III, IV и 5.0 Paradox *3jc, 4jc* и 5.0 без BDE - но таблицы будут открыты только для чтения. Ср дство BDE устанавливается вместе с установкой Paradox 8.0 или dBASE 7.

47. Средства создания приложений Примеры.

40.16. Назначение Access

Иногда Access (файл MDB) используется просто как ядро, которое управляет данными, находящимися с таблицами. Пользователи работают с этими данными через внешние приложения, созданные разработчиками, например, на Visual Basic, Delphi или C++. В других ситуациях Access, наоборот, используется только для предоставления пользовательского интерфейса для работы с данными, которые физически расположены на серверах баз данных, например, SQL Server, Oracle, IBM D2 и т.п.

40.17. Задачи приложений Access

- *создание обычных форм* — программных интерфейсов для занесения/изменения/просмотра данных в базе данных и Web-форм (они называются страницами доступа к данным);
- *создание отчетов к базам данных*, в том числе параметризованных;
- *создание программной логики приложения* обычным способом — на VBA (модули) и для начинающих пользователей — макросы (макросы всегда можно преобразовать в модули);
- *вспомогательные действия* — печать, экспорт и преобразование данных (хотя для преобразования данных обычно удобнее использовать объектную модель DTS), загрузка данных, репликация и т.п.

40.18. Этапы проектирования и создания приложений

Первый этап — *сбор информации о потребностях предприятия, его подразделений и пользователей*: какие унаследованные системы на нем работают, с чем нужно обеспечивать совместимость, как организованы информационные потоки, какова изменчивость системы и т.п. Часто при этом используются специальные средства, такие, как Visio и ERWin.

Второй этап — *выбор архитектуры приложения, выбор подходящей системы управления базами данных и проектирование СУБД*. На этом этапе определяется, будет ли информация храниться с использованием **ядра Jet** или **клиент-серверной системы, проектируется система таблиц для хранения информации и отношений между ними, проектируются некоторые другие объекты базы данных**. Кроме того, определяется архитектура приложения — сколько в ней будет уровней, будут ли использоваться терминальные или Web-технологии, будет ли применяться репликация и т.п.

Третий этап — *реализация СУБД и бизнес-логики приложения*. На этом этапе проектируются, создаются, настраиваются, заполняются исходными данными объекты базы данных — таблицы, представления, хранимые процедуры, формы, отчеты, макросы, программные модули и т.п. При создании приложений в Access большая часть этих операций выполняется при помощи **графического интерфейса** разработчика. **Код VBA** используется для проверки вводимых пользователем значений, для работы с элементами управления на форме, переключения между формами, отчетами, другими элементами управления, обращения к внешним объектным моделям и т.п. На этом этапе опять-таки могут помочь Visio и ERWin.

Четвертый этап — *оптимизация производительности базы данных*, этап, который часто упускается разработчиками. Задача эта комплексная, но включает в себя в том числе и оптимизацию **кода VBA**.

Пятый этап — *тестирование и отладка приложения* .

Шестой этап — *развертывание приложения*.

48. Средства задания ссылочной целостности.

40.19. Определение

Ссылочная целостность – это свойство, обеспечиваемое реляционными системами управления базами данных (РСУБД), которое предотвращает ввод несогласованных данных пользователями или приложениями. В большинстве РСУБД имеются различные правила ссылочной целостности, которые можно применить при создании связи между двумя таблицами.
Ссылочная целостность – это предохранительное устройство системы управления базами данных, гарантирующее, что каждый внешний ключ соответствует первичному ключу. Например, номера заказчиков являются первичными ключами в файле заказчиков и внешними ключами в файле заказов. При удалении записи заказчика должны быть удалены соответствующие записи заказов; в противном случае они останутся без исходной связи. Если СУБД не проверяет этого, соответствующий механизм должен программироваться в приложении.
Ссылочная целостность – это ограничение базы данных, гарантирующее, что ссылки между данными являются действительно правомерными и неповрежденными. Ссылочная целостность является фундаментальным принципом теории баз данных и проистекает из той идеи, что база данных должна не только сохранять данные, но и активно содействовать обеспечению их качества.

40.20. Преимущества ссылочной целостности

- **Улучшенное качество данных.** Очевидным преимуществом является поддержка качества данных, хранимых в базе данных. Ошибки могут по-прежнему существовать, но, по крайней мере, ссылки будут подлинными и неповрежденными.
- **Убыстрение разработки.** Ссылочная целостность объявляется. Это гораздо продуктивнее (на один или два порядка), чем написание специального программного кода.
- **Меньшее число ошибок.** Объявления ссылочной целостности являются гораздо более лаконичными, чем эквивалентный программный код. По существу, такие объявления приводят к повторному использованию проверенного и оттестированного кода общего назначения в сервере баз данных, а не к новой реализации одной и той же логики от случая к случаю.
- **Согласованность между приложениями.** Ссылочная целостность обеспечивает качество данных для нескольких прикладных программ, которые могут обращаться к базе данных.

40.21. Средства задания ссылочной целостности

В MS-Access команды SQL для определения ссылочной целостности поддерживаются только частично, но средство графического связывания является более сильным. С помощью графического интерфейса можно определить ссылочную целостность, а также ссылочные действия *cascade* и *no action*.

- **Cascade**. Удаление записи может привести к удалению записей с соответствующим значением внешнего ключа. Например, можно захотеть, чтобы при удалении записи о компании удалялись все адреса компании.
- **No action**. В качестве альтернативы можно запретить удаление записи, если имеются зависимые записи с соответствующим значением внешнего ключа. Например, если вы продали некоторой компании товары, то вы можете захотеть предотвратить удаление записи об этой компании.

49. CASE-средство ERwin. Назначение, состав и характеристика инструментальных средств Erwin. Основные этапы проектирования концептуальной модели предметной области и КМ базы данных с использованием CASE-средства ERwin. Примеры.

40.22. CASE-средство ERwin. Назначение, состав и характеристика инструментальных средств Erwin.

ERwin - CASE-средство для моделирования баз данных.
CASE - Computer-Aided Software Engineering - автоматизированная разработка программного обеспечения, CASE-технология программные и инструментальные средства для автоматизации процесса создания и внедрения программного обеспечения
ERwin - средство разработки структуры базы данных (БД). ERwin сочетает графический интерфейс Windows, инструменты для построения ER-диаграмм, редакторы для создания логического и физического описания модели данных и прозрачную поддержку ведущих реляционных СУБД и настольных баз данных. С помощью ERwin можно создавать или проводить обратное проектирование (реинжиниринг) баз данных.

40.23. Основные этапы проектирования концептуальной модели предметной области и КМ базы данных с использованием CASE-средства ERwin.

ERwin имеет два уровня представления модели - логический и физический.

Логический уровень - это абстрактный взгляд на данные, на нем данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире, например "Постоянный клиент", "Отдел" или "Фамилия сотрудника". Объекты модели, представляемые на логическом уровне, называются сущностями и атрибутами. Модель данных может быть построена на основе другой логической модели, например на основе модели процессов . Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.
Физическая модель данных, напротив, зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация о всех объектах БД. Поскольку стандартов на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько разных физических моделей. Если в логической модели не имеет значения, какой конкретно тип данных имеет атрибут, то в физической модели важно описать всю информацию о конкретных физических объектах - таблицах, колонках, индексах, процедурах и т. д. Разделение модели данных на логические и физические позволяет решить несколько важных задач.

Подуровни логического уровня модели данных

Различают 3 подуровня логического уровня модели данных, отличающиеся по глубине представления информации о данных:

- диаграмма сущность-связь (Entity Relationship Diagram (ERD);
- модель данных, основанная на ключах (Key Based model (KB);
- полная атрибутивная модель (Fully Attributed model (FA).

Диаграмма сущность-связь включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные сущности и связи между ними,которые удовлетворяют основным требованиям, предъявляемым к ИС. Диаграмма сущность-связь может включать связи "многие ко многим" и не включать описание ключей. Как правило, ERD используется для презентаций и обсуждения структуры данных с экспертами предметной области.

Модель данных, основанная на ключах, - более подробное представление данных. Она включает описание всех сущностей и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области.

Полная атрибутивная модель - наиболее детальное представление структуры данных: представляет данные в третьей нормальной форме и включает все сущности, атрибуты и связи.

Подуровни физического уровня модели данных

Различают два подуровня физического уровня модели данных:

- трансформационная модель (Transformation Model);
- модель СУБД (DBMS Model).

Трансформационная модель содержит информацию для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области. ERwin поддерживает ведение отдельных проектов, позволяя проектировщику выделять подмножество модели в виде предметных областей (Subject Area). Трансформационная модель позволяет проектировщикам и администраторам баз данных лучше представлять, какие объекты базы данных хранятся в словаре данных, и проверить, насколько физический уровень модели данных удовлетворяет требованиям к ИС.

Модель СУБД автоматически генерируется из трансформационной модели и является точным отображением системного каталога СУБД. ERwin непосредственно поддерживает эту модель путем генерации системного каталога.

50. CASE-средство ERwin. Компоненты диаграммы Erwin и основные виды представления диаграммы. Инструменты для создания логической модели БД.

40.24. Компоненты диаграммы ERwin

Основные компоненты диаграммы ERwin-это **сущности, атрибуты и связи**. Каждая **сущность** является множеством подобных индивидуальных объектов, называемых экземплярами. Каждый **экземпляр** индивидуален и должен отличаться ото всех остальных экземпляров. **Атрибут** выражает определенное свойство объекта. На физическом уровне сущности соответствует таблица, экземпляру сущности - строка в таблице, а атрибуту - колонка таблицы.

Сущность можно определить как объект, событие или концепцию, информация о которых должна сохраняться. Сущности должны иметь наименование с четким смысловым значением, именоваться существительным в единственном числе, не носить "технических" наименований и быть достаточно важными для того, чтобы их моделировать. Именование сущности в единственном числе облегчает в дальнейшем чтение модели. Фактически имя сущности дается по имени ее экземпляра. Примером может быть сущность ***Заказчик*** (но не ***Заказчики***) с атрибутами ***Номер заказчика, Фамилия заказчика*** и ***Адрес заказчика.*** На уровне физической модели ей может соответствовать таблица ***Customer*** с колонками ***Customer_number, Customer_name*** и ***Customer_address.***

Связь является логическим соотношением между сущностями. Каждая связь должна именоваться глаголом или глагольной фразой (Relationship Verb Phrases) (рис. 2.2.14). Имя связи выражает некоторое ограничение или бизнес-правило и облегчает чтение диаграммы, например:

- Каждый КЛИЕНТ <*размещает*> ЗАКАЗы;
- Каждый ЗАКАЗ <*выполняется*> СОТРУДНИКОМ.

Мощность связи (Cardinality) служит для обозначения отношения чис- экземпляров родительской сущности к числу экземпляров дочерней. Различают 4 типа мощности (рис. 2.2.18):

- общий случай, когда одному экземпляру родительской сущности соответствуют 0, 1 или много экземпляров дочерней сущности не помечается каким-либо символом;
- символом P помечается случай, когда одному экземпляру родительской сущности соответствуют 1 или много экземпляров дочерней сущности (исключено нулевое значение);
- символом Z помечается случай, когда одному экземпляру родительской сущности соответствуют 0 или 1 экземпляр дочерней сущности (исключены множественные значения);
- цифрой помечается случай точного соответствия, когда одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности.

40.25. Основные виды представлений диаграммы

Режим "сущности" служит для удобства обзора большой диаграммы или размещения прямоугольников сущностей на диаграмме.
Режим "определение сущности" служит для презентации диаграммы .
Режим "атрибуты" является основным при проектировании на логическом и физическом уровнях.
Режим "первичные ключи" - показываются только атрибуты/колонки, составляющие первичный ключ.
Режим "пиктограммы". Для презентационных целей каждой таблице может быть поставлена в соответствие пиктограмма (bitmap).
Режим "показ глагольной фразы". На дугах связей показываются глагольные фразы, связывающие сущности (для логического уровня) или имена внешних ключей (для физического уровня).

40.26. Инструменты для создания модели в ERwin

редакторы, связанные с сущностью в целом (определение сущности, дополнительная информация, триггеры, индексы, характеристики таблицы, хранимые процедуры, связанные с таблицей);
редакторы атрибутов (определение атрибутов, колонки таблицы в физическом представлении модели, репозитарий средства 4GL, например, расширенные атрибуты в PowerBuilder

51. Сущности и связи в ERwin. Альтернативные ключи, инвертированные индексы, унификация атрибутов, связи категоризации.

40.27. Сущности и связи в ERwin

См. Билет 52

40.28. Альтернативные ключи

Первичный ключ (primary key) - это атрибут или группа атрибутов, однозначно идентифицирующая экземпляр сущности.
В одной сущности могут оказаться несколько атрибутов или наборов атрибутов, претендующих на роль первичного ключа. Такие претенденты называются **потенциальными ключами (candidate key).** Для того чтобы стать первичным, потенциальный ключ должен удовлетворять ряду требований.
Уникальность. Два экземпляра не должны иметь одинаковых значений возможного ключа.
Компактность. Возможный ключ не должен содержать ни одного атрибута, удаление которого не приводило бы к утрате уникальности.
Альтернативный ключ **(Alternate Key)** - это потенциальный ключ, не ставший первичным.

40.29. Ивертированные индексы

При работе ИС часто бывает необходимо обеспечить доступ к нескольким экземплярам сущности, объединенным каким-либо одним признаком. Для повышения производительности в этом случае используются **неуникальные индексы.** ERwin позволяет на уровне логической модели назначить атрибуты, которые будут участвовать в неуникальных индексах. Атрибуты участвующие в неуникальных индексах, называются **инверсионными входами (Inversion Entries).** Inversion Entry - это атрибут или группа атрибутов, которые не определяют экземпляр сущности уникальным образом, но часто используются для обращения к экземплярам сущности. ERwin генерирует неуникальный индекс для каждого Inversion Entry.

40.30. Унификация атрибутов

Комбинирование или объединение идентичных атрибутов называется **унификацией.**

Унификация производится, поскольку правила нормализации запрещают существование в одной сущности двух атрибутов с одинаковыми именами. В некоторых случаях (рис. 2.2.41) этот результат соответствует действительности. Сотрудники работают в отделах, каждый сотрудник ведет несколько проектов. Сущность *Отдел* связана идентифицирующей связью с сущностями *Сотрудник* и *Проект,* ее первичный ключ *Номер отдела* мигрирует в состав первичного ключа дочерних сущностей в качестве внешнего ключа. Но сущность *Сотрудник,* в свою очередь, тоже имеет идентифицирующую связь с сущностью *Проект* и атрибуты ее первичного ключа (в том числе *Номер отдела*-второй раз!) мигрируют в состав первичного ключа сущности *Проект.*

Рис. 2.2.41. Унификация атрибута

40.31. Связи категоризации

Некоторые сущности определяют целую категорию объектов одного типа. В ERwin в таком случае создается сущность для определения категории и для каждого элемента категории, а затем вводится для них связь категоризации. Родительская сущность категории называется супертипом, а дочерние - подтипом.
Например, сущность "сотрудник" может содержать данные как о штатных работниках, так и о временно нанятых. Первые и вторые имеют различные, частично пересекающиеся наборы атрибутов (минимальное пересечение подтипов составляет первичный ключ). Общая часть этих атрибутов, включая первичный ключ, помещается в сущность-супертип "сотрудник". Различная часть (например, данные почасовой оплаты для временных работников и данные о зарплате и отпуске для штатных работников) помещается в сущности-подтипы.

52. Прямое и обратное проектирование. Синхронизация с базой данных. Интерфейсы к СУБД. Поддержка задания правил целостности и начальных значений.

41.1. Прямое и обратное проектирование.

Процесс генерации схемы базы данных из модели данных называется прямым проектированием (Forward Engineering). При генерации схемы ERwin включает триггеры ссылочной целостности, хранимые процедуры, индексы, ограничения и другие возможности, доступные при определении таблиц в выбранной СУБД. Процесс генерации модели из схемы базы данных называется **обратным проектированием** (Reverse Engineering). ERwin позволяет создать модель данных путем обратного проектирования имеющейся базы данных. После того как модель создана, можно переключиться на другой сервер (модель будет конвертирована) и произвести прямое проектирование структуры базы данных для другой СУБД. Кроме режима прямого и обратного проектирования ERwin поддерживает синхронизацию между моделью и системным каталогом СУБД на протяжении всего жизненного цикла создания ИС.

41.2. Синхронизация с базой данных.

После обратного проектирования базы данных, при добавлении или изменении объектов в ERwin или непосредственно на сервере базы данных, возникают расхождения между моделью данных и физической схемой. Когда это происходит, ERwin позволяет синхронизировать определения, так чтобы информация, хранимая в ERwin, соответствовала информации, хранящейся на сервере.
Во время процесса синхронизации ERwin сравнивает имена логической модели и имена физической схемы, хранящиеся на сервере, и показывает , какие из объектов схемы не синхронизированы.

41.3. Интерфейсы к СУБД

ERwin поддерживает прямой интерфейс с основными СУБД: DB2, Informix, Ingres, NetWare SQL, ORACLE, Progress, Rdb, SQL/400, SQLBase, SQL Server, Sybase System 10, Watcom SQL.
ERwin поддерживает также настольные (desktop) СУБД: Microsoft Access, FoxPro, Clipper, dBASE III, dBASE IV и Paradox.
Проектирование на физическом уровне выполняется в терминах той базы данных, которую предполагается использовать в системе. Важно, что ERwin "известны" соответствия между возможностями СУБД различных производителей, вследствие чего возможна конвертация физической схемы, спроектированной для одной СУБД, в другую.
Для создания физической структуры БД может быть запрошена генерация DDL-скрипта (data definition language). При этом используется диалект SQL для выбранного типа и версии сервера.

41.4. Поддержка задания правил целостности и начальных значений

В ERwin поддерживаются два типа правил (проверок допустимости значений) и начальных (по умолчанию) значений. Правило и умолчание может быть указано для проверки со стороны клиента (например, в PowerBuilder) и со стороны сервера.

53. Генерация отчетов.

Для создания документации на основе модели процессов или модели данных можно воспользоваться как встроенными средствами ER- win, так и внешними генераторами отчетов. Каждый из этих способов имеет свои достоинства и недостатки. Рассматриваются все встроенные средства ERwin, а также два внешних генератора отчетов - RPTwin и Crystal Reports. Общий перечень рассматриваемых средств создания отчетов приведен в табл.

Таблица 3.1.1. Средства создания отчетов по моделям отчетов и моделям данных

Средство создания отчетов	Достоинства
Report Template Builder	Легкость использования, возможность публикации на Web-сереерах не только текстовых отчетов, но и диаграмм моделей процессов и моделей данных. Возможность экспорта отчетов в текстовые файлы, файлы RTF
Data Browser	Легкость использования, возможность создания собственных шаблонов, хранимых представлений отчетов и экспорта отчетов в текстовые файлы, HTML, файлы MS Office и RPTwin
RPTwin	Получение качественных отчетов, фильтрация, группировка и сортировка. Имеется встроенный язык формул для обработки данных
Crystal Reports	Получение отчетов презентационного качества, сложное форматирование данных, сложная обработка данных, создание диаграмм, экспорт отчетов во все распространенные форматы

Создание отчетов по модели данных с помощью Report Browser

Для генерации отчетов в ERwin имеется эффективный и простой в использовании инструмент - Report Browser. Он позволяет выполнять предопределенные отчеты (объединенные по типам), сохранять результаты их выполнения, создавать собственные отчеты, печатать и экспортировать их в распространенные форматы. Каждый отчет может быть настроен индивидуально, данные в нем могут быть отсортированы и отфильтрованы.

Создание отчетов в RPTwin

RPTwin является специализированным генераторам отчетов, который позволяет создавать качественные отчеты по моделям процессов и данных. RPTwin входил в поставку ранних версий ERwin. Функциональность RPTwin позволяет создавать не просто отчеты высокого качества, что само по себе очень важно. Включение в RPTwin более 40 функций позволяет производить сложную обработку данных, получая при этом результат, который невозможно получить средствами ERwin.

Использование Crystal Reports для создания отчетов

Crystal Reports является признанным лидером среди недорогих генераторов отчетов, работающих на платформе Windows. Простота использования и широкие функциональные возможности делают этот инструмент очень удобным для создания наглядных высококачественных отчетов, иллюстрирующих все детали функциональной модели, созданной в BPwin. Crystal Reports позволяет создавать отчеты, используя в качестве источников данных текстовые файлы, настольные базы данных (dBase, Paradox, Access и др.), реляционные СУБД (Oracle, MS SQLServer, Sybase, Informix и др.) и специальные источники данных, например файловую систему или OLE DB.