Базы данных. Экзаменационные вопросы.

1. Назначение и элементы автоматизированных информационных систем.

Автоматизированные информационные системы — человеко-машинные системы для поиска, сбора, накопления, хранения, передачи, обработки информации с использованием вычислительной техники, компьютерных информационных сетей, средств и каналов связи.

Основная цель АИС - хранение, обеспечение эффективного поиска и передачи информации по соответствующим запросам для наиболее полного удовлетворения информационных запросов большого числа пользователей.

Современные информационные системы состоят из нескольких видов обеспечивающих подсистем, к которым относятся: техническое, программное, информационное и организационное обеспечения.

Техническое обеспечение представляет собой комплекс технических средств, обеспечивающих функционирование информационной системы.

Программное обеспечение — это совокупность программ и документации на них, реализующих основные функции информационной системы.

Информационное обеспечение представляет собой совокупность информационной базы предметной области и средств и методов ее обработки.

Организационное обеспечение представляет собой комплекс методов и правил организации работы с информационной системой, а также описание должностных инструкций пользователей информационной системы.

2. Развитие архитектуры и технологий АИС.

Год	ЭВМ	Решаемые задачи	Тип АИТ
Конец 1950-х - начало 1960х гг.	I, II поколения	Использование ЭВМ для решения отдельных наиболее трудоемких задач по начислению заработной платы, материальному учету и др.; решение отдельных оптимизированных задач	Частичная электронная обработка данных
1960е гг. начало 1970х гг.	II, III поколения	Электронная обработка плановой и текущей информации, хранение в памяти ЭВМ нормативно-справочных данных, выдача	ЭСОД - электронная система обработки данных

		машинограмм на бумажных носителях	
1970-е гг.	III поколение	Комплексная обработка информации на всех этапах управленческого процесса деятельностью предприятия, организации, переход к разработке подсистем АСУ (материально-технического снабжения, товародвижения, контроль запасов и транспортных перевозок, учет реализации готовой продукции, планирование и управление)	Централизованная автоматизированная обработка информации в условиях ВЦ, ВЦКП (вычислительных центров коллективного использования)
1980-е гг.	IV поколение	Развитие АСУГП (АСУ технологическими процессами), САПР (систем автоматизированного проектирования), АСУП (АСУ предприятиями), ОАСУ (отраслевых АСУ), общегосударственных АСУ: плановых расчетов, статистики, материально-технического снабжения, науки и техники, финансовых расчетов и др. Тенденция к децентрализации обработки данных, решению задач в многопользовательском режиме, переход к безбумажной эксплуатации вычислительной техники	Специализация технологических решений на базе миниЭВМ, ПЭВМ и удаленного доступа к массивам данных с одновременной универсализацией способов обработки информации на базе мощных суперЭВМ.

Конец 1980-х гг. - по настоящее время	V поколение	Комплексное решение экономических задач; объектно-ориентированный подход в зависимости от системных характеристик предметной области; широкий спектр приложений; сетевая организация информационных структур; преобладание интерактивного взаимодействия пользователя в ходе эксплуатации вычислительной техники. Реализация интеллектуального человеко-машинного интерфейса, систем поддержки принятия решений, информационно-советующих систем	НИТ (новая информационная технология) - сочетание средств вычислительной техники, средств связи и ор

3. Базы данных. Системы баз данных. Элементы СБД.

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются **системой баз данных**, или, для краткости, просто базой данных.

В системе баз данных выделяют четыре основных компонента:

- данные;
- аппаратное обеспечение;
- программное обеспечение;
- пользователи.

Аппаратное обеспечение:

- накопители;
- сетевое оборудование;
- оперативная память
- процессор.

Программное обеспечение:

- СУБД;
- утилиты;
- средства разработки приложений (программы конечного пользователя);
- средства проектирования;
- генераторы счетов и др.

Пользователи:

- *Прикладные программисты* – пользователи, которые отвечают за написания прикладных программ (приложений), использующих БД.
- *Конечные пользователи* – пользователи, которые работают с базой данных через рабочую станцию (терминал). Конечный пользователь получает доступ к БД через приложения или используя интегрированный интерфейс СУБД. Конечный пользователь часто использует интерфейс, основанный на меню и различных формах, что облегчает работу.
- *Администраторы базы данных* организуют и отвечают за работу с БД.

4. Понятие и функции СУБД. Основные модули СУБД.

Система управления базами данных (СУБД) - это программное обеспечение, которое взаимодействует с конечными пользователями, приложениями и самой базой данных для сбора и анализа данных.

Основные функции СУБД:

- управление данными во внешней памяти (на дисках);
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД (язык определения данных, язык манипулирования данными).

Состав СУБД

Обычно современная СУБД содержит следующие компоненты:

- ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;
- процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;
- подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;
- сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

5. Архитектура систем баз данных.

Для решения этой задачи большинство современных СУБД основываются на архитектуре предложеной ANSI SPARC. В этой архитектуре определено 3 уровня абстракции (три уровня описания данных):

- Внешний
    - Внешние представления отображают потребности отдельных категорий пользователей и содержат те объекты предметной области, которые нужны данному пользователю.
- Концептуальный
    - На концептуальном уровне имеется обобщенное представление БД полученное как совокупность всех требований к данынм. Этот уровень содержит полную логическую структуру БД, то есть определяет что хранится в БД. На концептуальном уровне определяются:
        - Информационные объекты и связи между ними
        - Ограничение целостности
        - Семантическая информация
        - Требования к безопастности
        - Процесс обработки данных
- Внутрений
    - Схема физического представления базы данных в памяти ЭВМ. То есть как информация будет храниться в БД. В частности на этом уровне определяется:
        - Распределение элементов данных по файлам
        - Распределение дискового пространства
        - Детали хранения записей с данными
        - Сведения о размещении записей, структура индексов

На внешнем уровне имеется несколько внешних схем. Концептуальная схема и внутреняя схема.

Физический уровень СБД отвечает за воплощение внутреней схемы в вычислительной системе и определяет функционирование БД под управлением СУБД и операционной системы.

Структуризация систем БД по уровням дает следующие преймущества:

- Пользователь обращается к общему хранилищу данных но использует собственное представление данных.
- Прикладной программист работает с концептуальной схемой данных используя SQL а не с самими данными в файлах.
- Администратор БД может менять способ хранения данных не затрагивая концептуальную схему пользовательских представлений.
- Администратор данных может менять структуру БД то есть концептуальную схему при этом одельные пользовательские представления могут оставаться неизменными.
- Внутреняя схема БД не меняется при манипуляции с физическим местом хранилища.

Таким образом трехуровневая архитектура обеспечивает независимость прикладных пользовательских приложений от способа хранения данных на логическом и физическом уровне.

6. Этапы проектирования АИС.

Основные этапы проектирования баз данных

Концептуальное (инфологическое) проектирование

Концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных.

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER-диаграммам.

Чаще всего концептуальная модель базы данных включает в себя:

- описание информационных объектов или понятий предметной области и связей между ними.

- описание ограничений целостности, то есть требований к допустимым значениям данных и к связям между ними.

Логическое (даталогическое) проектирование

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам. Этот этап может быть в значительной степени автоматизирован.

На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Физическое проектирование

Физическое проектирование — создание схемы базы данных для конкретной СУБД. Специфика конкретной СУБД может включать в себя ограничения на именование объектов базы данных, ограничения на поддерживаемые типы данных и т. п. Кроме того, специфика конкретной СУБД при физическом проектировании включает выбор решений, связанных с физической средой хранения данных (выбор методов управления дисковой памятью, разделение БД по файлам и устройствам, методов доступа к данным), создание индексов и т. д.

7. Модель «сущность связь». Элементы модели.

Модель Сущность-Связь (ER-модель) (англ. entity-relationship model или entity-relationship diagram) — это модель данных, позволяющая описывать концептуальные схемы. Она предоставляет графическую нотацию, основанную на блоках и соединяющих их линиях, с помощью которых можно описывать объекты и отношения между ними какой-либо другой модели данных. В этом смысле ER-модель является средством описания моделей данных.

Сущность - это класс однотипных объектов, информация о которых должна быть учтена в модели.

Каждая сущность должна иметь наименование, выраженное существительным в единственном числе.

Каждая сущность в модели изображается в виде прямоугольника с наименованием.

Экземпляр сущности - это конкретный представитель данной сущности.

Экземпляры сущностей должны быть *различимы*, т.е. сущности должны иметь некоторые свойства, уникальные для каждого экземпляра этой сущности.

Атрибут сущности - это именованная характеристика, являющаяся некоторым свойством сущности.

Наименование атрибута должно быть выражено существительным в единственном числе (возможно, с характеризующими прилагательными).

Атрибуты изображаются в пределах прямоугольника, определяющего сущность:

Ключ сущности - это *неизбыточный* набор атрибутов, значения которых в совокупности являются *уникальными* для каждого экземпляра сущности. Неизбыточность заключается в том, что удаление любого атрибута из ключа нарушается его уникальность.

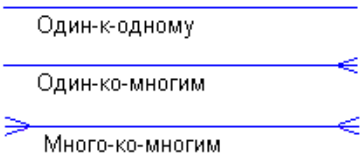
Сущность может иметь несколько различных ключей.

Ключевые атрибуты изображаются на диаграмме подчеркиванием (либо рядом с ключевым атрибутом рисуется знак ключа).

Связь - это некоторая ассоциация между *двумя* сущностями. Одна сущность может быть связана с другой сущностью или сама с собою.

Связи позволяют по одной сущности находить другие сущности, связанные с нею.

Каждая связь может иметь один из следующих ***типов связи***:



Также определяются виды связи: обязательная (mandatory) и необязательная (optional).

8. Модель «сущность связь». Порядок построения. Нотации диаграмм.

В процессе построения диаграммы можно выделить несколько очевидных этапов:

1. Идентификация представляющих интерес сущностей и связей.
2. Идентификация семантической информации в наборах связей (например, является ли некоторый набор связей отображением *1:n*).
3. Определение кардинальностей связей.

4. Определение атрибутов и наборов их значений (доменов).
5. Организация данных в виде отношений "сущность-связь".

Основные нотации:

- Нотация Чена (сущности изображаются в виде прямоугольников, их атрибуты в виде овалов, а отношения между сущностями – ромбами)
- Нотация «Crow’s foot» (атрибуты записываются прямо внутри прямоугольников, обозначающих сущности, а отношения записываются в виде текста над стрелкой)

9. Модель «сущность связь». Особые и нестандартные схемы. Расширения модели.

Особые и нестандартные схемы:

- Слабые сущности
- Атрибуты связей
- Многосторонние связи
- Рекурсивные связи
- Параллельные связи
- Циклы
- Связь уточнение\обобщение

EER - это высокоуровневая модель данных, которая включает в себя расширения исходной модели ER. Расширенные ERD - это высокоуровневые модели, которые отражают требования и сложности сложной базы данных.

В дополнение к концепциям модели ER EE-R включает в себя−

- Подклассы и суперклассы.
- Специализация и обобщение.
- Категория или тип объединения.
- Агрегирование.

Эти концепции используются для создания диаграмм EE-R.

Суперкласс - это сущность, которая может быть разделена на другие подтипы.

Подклассы - это группа сущностей с некоторыми уникальными атрибутами.Подкласс наследует свойства и атрибуты суперкласса.

Специализация и обобщение

Обобщение - это процесс обобщения сущности, которая содержит обобщенные атрибуты или свойства обобщенных сущностей.

Специализация - это процесс идентификации подмножеств сущности, которые имеют некоторые общие характеристики. Это нисходящий подход, при котором один объект разбивается на объект низкого уровня.

Категория или объединение

Связь одного суперкласса или подкласса с более чем одним суперклассом.

Агрегирование

Представляет взаимосвязь между целым объектом и его компонентом.

10. Объединение моделей локальных представлений при проектировании БД.

При объединении моделей локальных представлений проектировщик может формировать конструкции, являющиеся производными по отношению к использованным в локальных представлениях. Вводимые производные конструкции должны обеспечивать непротиворечивое представление данных.

При объединении представлений используются три основополагающие концепции: идентичность, агрегация и обобщение.

Два или более элементов модели идентичны, если имеют одинаковое семантическое (смысловое) значение.

Агрегация позволяет рассматривать связь между элементами модели как новый элемент.

При объединении представлений агрегация встречается в следующих трех формах:

- В одном представлении агрегатный объект определен как единое целое, а во втором - рассматриваются его составные части.
- Агрегатный объект как единое целое не определен ни в одном из представлений, но в обоих представлениях рассматриваются его различные составные части.
- Один и тот же агрегатный объект рассматривается в обоих представлениях, но составляющие различаются.

Обобщением называется абстракция данных, позволяющая трактовать класс различных подобных типов объектов как один поименованный обобщенный тип объекта. В обобщении подчеркивается общая природа объектов.

11. Понятие модели данных. Ранние модели данных.

Модель данных (или datamodel) - это абстрактная модель, которая организует элементы данных и стандартизирует их связь друг с другом и со свойствами объектов реального мира.

Наиболее распространенная модель принадлежит Дейту. Согласно дейту модель состоит из трех частей:

- Структурная часть
- Манипуляционная часть
- Целостная часть

В структурной части модели данных фиксируются основные логические структуры данных которые могут применяться на уровне пользователя. Например в реалиционной модели это просто отношение (таблица, сущность).

Манипуляционная часть модели данных содержит специализацию одного или нескольких языков предназначенных для написания запросов к БД. Языки могут быть абстрактно математическими или законченными производственными языками.

Основными видами операций свойственными большинству моделей данных являются:

- Идентификация элемента и его позиции в БД.
- Выборка элемента данных.
- Включение элемента данных.
- Исключение элемента данных.
- Модификация элемента данных.

К целостной части модели данных определяются механизмы ограничения целостности которые обязательно должны поддерживаться.

Ранние модели данных:

1. Модель данных инвертированных таблиц.
2. Иерархическая.
3. Сетевая.

- **Модель данных инвертированных таблиц**
    - Появилась в 60-х СУБД Datacom/DB.
    - Сейчас используется почти во всех реляционных СУБД, но не редактируются пользователями (используются для индексов).
    - База данных в модели инвертированных таблиц представляет собой совокупность таблиц, при этом пользователям видны и хранимые таблицы, и пути доступа к ним:
        - строки упорядочиваются в некоторой последовательности
        - упорядоченность может определяться для всей БД

        - для таблицы можно определить произвольное число ключей поиска, на основе которых строятся индексы.

    - Операции: класс «устанавливающие адрес» и класс «над адресуемыми записями». Правил ограничения целостности нет.

- **Иерархическая**

    - СУБД IMS (Information Management System) компании IBM – 1968 г.

    - Состоит из упорядоченного набора деревьев. Тип дерева состоит из одного «корневого» типа записи и упорядоченного набора из нуля или более типов поддеревьев (каждое из которых также является некоторым типом дерева).

    - Тип дерева в целом представляет собой иерархически организованный набор типов записи.

    - Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами.

    - Для иерархической базы данных определяется полный порядок обхода дерева: сверху-вниз, слева-направо.

    - В иерархической модели данных автоматически поддерживается целостность ссылок между предками и потомками.

    - Основное правило: никакой потомок не может существовать без своего родителя.

- **Сетевая**

    - СУБД IDMS – 1971 г. Концепция была определена Чарльзом Бахманом.

    - Сетевой подход к организации данных является расширением иерархического подхода.

    - В иерархических структурах запись-потомок должна иметь в точности одного предка; **в сетевой структуре** у потомка может иметься любое число предков.

    - Сетевая модель данных основывается на следующих понятиях (по возрастанию):

        - «элемент» (аналог атрибута)
        - «агрегат» (именованная совокупность элементов или других * агрегатов данных)

        - «запись» (агрегат, который не является частью другого агрегата, может иметь первичный ключ из одного или неск. элементов)
        - «набор» (групповое отношение, иерарх. отн. между записями двух типов, где первые – владельцы, а вторые – подчиненные)
        - «база данных»
        - ограничение данных – можно потребовать для конкретного типа связи отсутствие потомков, не участвующих ни в одном экземпляре этого типа связи (как в иерархической модели).
    - Любую сетевую модель можно представить как иерархическую с дублированием данных.

12.Понятие модели данных. Реляционная модель данных.

Реляционная модель данных - созданная Эдгаром Коддом логическая модель данных, описывающая:

- структуры данных в виде (изменяющихся во времени) наборов отношений;
- теоретико-множественные операции над данными: объединение, пересечение разность и декартово произведение;
- специальные реляционные операции: селекция, проекция, соединение и деление;
- специальные правила, обеспечивающие целостность данных.

Цели создания реляционной модели данных:

- обеспечение более высокой степени независимости от данных;
- создание прочного фундамента для решения семантических вопросов и проблем непротиворечивости и избыточности данных;
- расширение языков управления данными за счёт включения операций над множествами.

Структура данных в реляционной модели данных

Реляционная модель данных предусматривает структуру данных, обязательными объектами которой являются:

- отношение;
- атрибут;
- домен;
- кортеж;
- степень;
- кардинальность;

- первичный ключ.

  Отношение - это плоская (двумерная) таблица, состоящая из столбцов и строк.
  Атрибут - это поименованный столбец отношения.
  Домен - это набор допустимых значений для одного или нескольких атрибутов.
  Кортеж - это строка отношения.Степень определяется количеством атрибутов, которое оно содержит.
  Кардинальность - это количество кортежей, которое содержит отношение.
  Первичный ключ - это уникальный идентификатор для таблицы.

Соответствие между формальными терминами реляционной модели данных и неформальными:

- отношение (формальный термин) - таблица (неформальный термин);
- атрибут - столбец;
- кортеж - строка или запись;
- степень - количество столбцов;
- кардинальное число - количество строк;
- первичный ключ - уникальный идентификатор;
- домен - общая совокупность допустимых значений.

13. Правила преобразования модели «сущность-связь» в реляционную модель данных.

Требования:

1. Отсутствие многосторонних связей (связи, в которых участвуют три или более сущностей).
2. Отсутствие атрибутов связей, такие связи должны быть заменены промежуточной сущностью.
3. Отсутствие многозначных атрибутов (атрибуты, которые хранят несколько значений для одного экземпляра).

Алгоритм:

1. Для каждой сильной сущности создается отношение, включающее простые атрибуты (составные должны быть предварительно преобразованы в простые). Идентифицирующие атрибуты составляют первичный ключ.
2. Для каждой слабой сущности аналогично создается отношение. Первичный ключ для такого отношения формируется после преобразования связей данной слабой сущности.
3. Преобразование простых бинарных связей
   3.1. Один-ко-многим. Сущность, находящаяся на стороне связи «один», определяется как родительская, а сущность на стороне связи «многие» – как дочерняя. Атрибут(ы), составляющие первичный ключ родительской

сущности, копируются в дочернюю сущность и составляют в ней внешний ключ. Если одна из сущностей слабая: копия первичного ключа родительской сущности, образуя внешний ключ дочерней сущности, одновременно входит в состав её первичного ключа. Таким образом, первичный ключ в отношении оказывается составным.

3.2. Многие-ко-многим. Для реализации связи данного типа в реляционной модели необходимо создание дополнительного отношения “A_B”, в которое копируются атрибуты, составляющие первичные ключи связываемых сущностей. Данные атрибуты по отдельности образуют внешние ключи, а вместе – первичный ключ отношения “A_B”.

3.3. Один-к-одному. Обязательность сущностей:

а) обязательное участие обеих сущностей

Если принято решение об обязательности участия обеих сущностей в связи, то должно быть образовано одно отношение «A_B», в которое помещаются атрибуты обеих сущностей, а на роль первичного ключа назначается один из первичных ключей связываемых сущностей. Первичный ключ другой сущности может быть объявлен альтернативным ключом отношения.

б) обязательное участие одной сущности

В этом случае сущность, которая характеризуется необязательным участием в связи, обозначается как родительская, а сущность, которая должна обязательно участвовать в связи, - как дочерняя. Атрибут(ы), составляющие первичный ключ родительской сущности, копируются в дочернюю сущность и составляют в ней внешний ключ. Для контроля кратности связи внешний ключ должен быть объявлен также альтернативным ключом отношения.

в) необязательное участие обеих сущностей

Необходимо создание 3-го отношения по схеме, близкой к связи «многие-ко-многим». Но в данном случае атрибуты только одной из сущностей в данном 3-м отношении составляют первичный ключ, атрибуты другой объявляются альтернативным ключом.

4. Преобразование особых и нестандартных типов связей

4.1. Рекурсивные связи.

Рекурсивная связь «один-ко-многим» приводит к одному отношению, в котором появляется копия первичного ключа, которая становится внешним ключом отношения. Атрибут, который образует этот внешний ключ, должен быть переименован, чтобы исключить дублирование наименований атрибутов.

Рекурсивная связь «один-ко-многим» приводит к созданию двух отношений, одно из которых отражает рекурсивную связь.

Рекурсивная связь «многие-ко-многим» приводит к созданию одного или двух отношений в зависимости от степени (обязательности) участия в ней сущностей.

4.2. Параллельные связи. Преобразование состоит в том, что атрибуты, которые образуют внешний ключ, должны быть переименованы, чтобы исключить дублирование наименований атрибутов.

4.3. Связи уточнение/обобщение (is-a).

При рассмотрении связи такого типа обобщающая сущность («суперкласс»), определяется как родительская, а уточняющие сущности («подклассы»), – как дочерние. Схема преобразования такой структуры в реляционные отношения зависит от ограничения непересечения (т.е. может ли один объект относиться одновременно к нескольким подклассам) и степени (обязательности) участия суперкласса в связи (т.е. могут ли существовать объекты, не отнесенные ни к одному из подклассов).

Пересечение допускается – обязательное – одно отношение, с одним или несколькими определителями, позволяющими указать тип каждого кортежа.

Пересечение допускается – необязательное – Два отношения, одно – для суперкласса, второе – для всех подклассов, с одним
или несколькими определителями, позволяющими указать тип каждого кортежа.

Пересечение не допускается – обязательное – Несколько отношений: по одному отношению для каждого сочетания суперкласс/подкласс

Пересечение не допускается – необязательное – Несколько отношений: одно – для суперкласса, плюс по одному для каждого подкласса

14. Операции реляционной алгебры.

Реляционная алгебра - набор операций, которые принимают набор отношений в качестве операндов и возвращают отношение, как результат.

Реляционную алгебру можно описать, как процедурный язык, с помощью которого можно определить, как построить новое отношение из одного или нескольких существующих.

Первоначальный вариант разработан Коддом и имеет 8 операций.

Основные:

1. Объединение (R ∪ S). Результат – множество кортежей, принадлежащих R, S или обоим. Операция требует совместимости по объединению.
2. Разность (R - S). Результат – множество кортежей, принадлежащих R, но не принадлежащих S. Отношения должны быть совместимы по объединению.

3. Декартово произведение (R*S). Результат – все комбинации кортежей R и S. Кортежи результирующего отношения – конкретизация кортежей R и S.
4. Проекция ($\Pi_{i1,\dots,im}(R)$). Результат – отношение с выбранным подмножеством атрибутов. i1…im – атрибуты, по которым формируется результирующее отношение. Кортежи-дубликаты устраняются.
5. Селекция ($\sigma_F(R)$). Выборка из отношений кортежей, удовлетворяющих условию. F – предикат.

Проекция и селекция – унарные операции.

Дополнительные:

6. Пересечение ($R \cap S = R - (R - S)$). Результат содержит кортежи, которые присутствуют и в R, и в S. Отношения должны быть совместимы по объединению.
7. Частное $R \div S = \Pi_{1,\dots,n-k}(R) - \Pi_{1,\dots,n-k}(\Pi_{1,\dots,n-k}(R) * S - R)$
   R определено на множестве атрибутов A и имеет арность n, S определено на множестве B и имеет арность k. n>k; B подмножество A.
   C=A-B – множество атрибутов. R и S имеют k совпадающих атрибутов, в R присутствуют дополнительные (n-k) атрибутов, которых нет в S.
   Результат – набор кортежей R, определенных на множестве атрибутов C, для которых на совпадающих k атрибутах отношения R имеются комбинации всех кортежей отношения S.
8. Операции соединения, предполагающие выборку из декартова произведения:
   8.1. Θ-соединение. Соединение отношений по произвольному оператору сравнения кортежей этих отношений.
   $$R \frac{\bowtie}{i\Theta j} S = \sigma_{i\Theta(n+j)}(R * S)$$
   Θ – арифметический оператор сравнения; i, j – атрибуты R и S
   8.2. Естественное соединение. Соединение по общим атрибутам двух отношений с исключением одного из экземпляров каждого общего атрибута.
   $R \bowtie S = \Pi_{i1\dots im}(\sigma_{R.A1=S.A1 \wedge \dots \wedge R.Ak=S.Ak}(R*S))$,
   i1…im – список всех атрибутов отношений (R*S), за исключением S.A1…S.Ak
   Aj – имена столбцов в R и S
   8.3. Внешнее соединение. Естественное соединение, когда в результирующее отношение включаются кортежи одного отношения, не имеющие соответствия в другом. При этом в результирующем отношении вместо отсутствующих атрибутов указывается пустое значение.
   Левое внешнее соединение (в результирующее отношение попадают все кортежи отношения, стоящего слева от оператора соединения): $R \supset\!\triangleleft S$
   Правое внешнее соединение (в результирующее отношение попадают все кортежи отношения, стоящего справа от оператора соединения): $R \triangleright\!\subset S$.

Полное внешнее соединение (в результирующее отношение попадают все кортежи обоих отношений): R $\supset\subset$ S.

15. Правила построения формул реляционного исчисления с переменными-кортежами.

Реляционное исчисление кортежей (предложено Коддом). Областью определения переменных здесь являются реляционные отношения, т.е. допустимыми значениями переменной являются кортежи заданного отношения.

Задача реляционного исчисления кортежей состоит в нахождении таких кортежей, для которых предикат является истинным.

Формулы в реляционном исчислении имеют вид $\{t^{(i)} / \varphi(t)\}$, где $t^{(i)}$ – переменная-кортеж фиксированной длины i, или $\{t(R) / \varphi(t)\}$, где t(R) – переменная-кортеж со схемой отношения R; φ – формула, построенная из атомов и совокупности операторов.

Атомы формулы φ могут быть трех видов:

1. R(s). Здесь R – имя отношения, а s – переменная-кортеж той же арности, что и R. Атом означает, что кортеж s принадлежит отношению R.

2. s[i] θ u[j]. Здесь θ – арифметический оператор сравнения, а атом в целом означает, что i-тый компонент переменной-кортежа s находится в отношении θ с j-тым компонентом u.

3. a θ u[j]. Атом означает, что некоторое число a находится в отношении θ с j-тым компонентом переменной-кортежа u.

Существуют свободные и связанные вхождения переменных в формулу. Входящая в формулу переменная называется связанной, если ее вхождению предшествуют кванторы ∀, ∃. В противном случае переменная называется свободной.

Формулы, а также свободные и связанные вхождения переменных в эти формулы определяются следующим образом:

1. Каждый атом есть формула. Все переменные-кортежи, входящие в атом, являются свободными в этой формуле.

2. Если φ_1 и φ_2 – формулы, то $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\overline{\varphi 1}$, $\overline{\varphi 2}$ – тоже формулы.

3. Если φ – формула, то (∃s)(φ), (∀s)(φ) – также формулы. При этом первая запись означает, что существует кортеж s, удовлетворяющий условию φ, а вторая – что условие φ выполняется для любого кортежа s.

4. Формулы при необходимости могут заключаться в скобки. При этом принят следующий приоритет выполнения операций: арифметические операции, операции сравнения, кванторы, логические операторы (отрицание, конъюнкция, дизъюнкция).

5. Единственной свободно входящей в формулу φ переменной является переменная-кортеж t, множество значений которой собственно и определяется этой формулой. Все остальные входящие в формулу переменные-кортежи должны быть связаны кванторами.

6. Ничто другое не является формулой.

16. Формулы реляционного исчисления для основных операций.

1) Объединение ($R^{(n)} \cup S^{(n)}$). Представляет собой множество кортежей,

которые принадлежат R и S, либо им обоим.

$\{t^{(n)} / R(t) \vee S(t)\}$

2) Разность ($R^{(n)} – S^{(n)}$). Множество кортежей, принадлежащих R, но не

принадлежащих S.

$\{t^{(n)} / R(t) \wedge \overline{S}(t)\}$

3) Декартово произведение ($R^{(n)} * S^{(m)}$). Результат включает все

комбинации кортежей R и S.

$$\{t^{(n+m)}/(\exists u^{(n)} \in R: (t[1] = u[1] \wedge \dots \wedge t[n] = u[n])) \wedge (\exists v^{(m)} \in S: (t[n+1] = v[1] \wedge \dots \wedge t[n+m] = v[m]))\}$$

4) Проекция $\pi_{i1,\dots,ik}(R^{(n)})$. Результат отношения – выбранное подмножество атрибутов, в котором кортежи-дубликаты устраняются.

i1, ..., ik – атрибуты, по которым формируется результат.

$$\{t^{(k)}/(\exists u^{(n)} \in R: (t[1] = u[i_1] \wedge \dots \wedge t[k] = u[i_k]))\}$$

5) Селекция $\sigma_F(R^{(n)})$. F – формула, образованная константами, названиями элементов отношения, арифметическими операторами сравнения, логическими операторами. Селекция – множество кортежей, принадлежащих R, таких что при подстановке соответствующих элементов отношения в формулу F становится истинной.

$$\{t^{(n)}/R(t) \wedge F\}$$

17. Нормализация реляционных отношений.

Нормализация – метод создания набора отношений с заданными свойствами на основе требований к данным установленным в предметной области.

Нормализация основывается на формальном наборе правил, позволяющих определять отношения на основе их первичных или потенциальных ключей с учетом функциональной зависимости между атрибутами.

Варианты аппарата нормализации:

1. Как методология проектирования отношений, на основе некоторого представления данных, которые хотят получить пользователи.
2. Как набор тестов контроля соответствия спроектированных отношений формальным требованиям.

В основе лежит установление связей между атрибутами.

Рациональные варианты группировки атрибутов в отношения должны отвечать следующим требованиям:

1. Число выбранных атрибутов для первичных ключей должно быть минимально, в идеале простой ПК (1 атрибут)
2. Выбранный состав отношений должен быть по возможности минимальным
3. Процедуры обработки и обновления данных не должны вызывать осложнений.

Для ненормализованных отношений характерна избыточность данных: одна и та же информация повторяется в нескольких кортежах.

При работе с отношениями, содержащими избыточные данные могут возникать проблемы, которые называются аномалиями обновления и подразделяются на аномалии вставки, удаления и модификации. (Вставки – нельзя добавить информацию об одном объекте, не указав информацию о связанных. Удаления – при удалении кортежа удаляется информация о связанных объектах. Модификации – при изменении значения одного из атрибутов должны быть изменены все кортежи, содержащие это значение).

В основе нормализации лежит понятие функциональной зависимости между атрибутами.

Функциональная зависимость является семантическим свойством атрибутов отношения, то есть определяется, исходя из смысла хранимой информации.

X, Y – атрибуты R

X→Y (функционально зависит), если в любой момент времени каждому значению X соответствует не более одного значения Y

X – детерминант

Тривиальная ФЗ – справедлива при любых условиях, то есть обусловлена структурой самой зависимости

XZ→X (имеет место, когда справа подмножество детерминанта)

X, Y – атрибуты R

Y функционально полно зависит от X, если атрибут Y функционально зависит от X, но не зависит не от одного собственного подмножества X

Нормализация осуществляется в виде последовательных тестов на соответствие некоторого отношения требованиям заданной НФ.

Виды НФ: 1НФ. 2НФ, 3НФ, НФБК(Бойса-Кодда), 4НФ, 5НФ

Каждая последующая НФ предъявляет более жесткие требования.

1НФ: каждый кортеж содержит строго 1 значение для каждого из атрибутов

2НФ: находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от ПК

3НФ: во 2 НФ, каждый не ключевой атрибут нетранзитивно зависит от ПК.

Общая характеристика состоит в том, что в них рассмотрены зависимости атрибутов от ПК

НФБК: в структуре его функциональной зависимости каждый детерминант является потенциальным ключом.

Более строгая версия 3НФ. Различие: ФЗ X→Y допускается в 3НФ, если Y не является ПК, а X не обязательно является ПК

18. Законы алгебраических преобразований реляционных выражений.

Для оптимизации реляционных выражений используются следующие законы алгебраически преобразований.

1) Законы коммутативности для соединения и произведения. Здесь и далее E_i – реляционное выражение, а F_j – условие, налагаемое на реляционное выражение.
$$E_1 \triangleright\triangleleft E_2 \equiv E_2 \triangleright\triangleleft E_1, E_1 \times E_2 \equiv E_1 \times E_2, E_1 \underset{F_1}{\triangleright\triangleleft} E_2 \equiv E_1 \underset{F_1}{\triangleright\triangleleft} E_2.$$
2) Закон ассоциативности для соединений и произведения.
$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3),$$
$$(E_1 \triangleright\triangleleft E_2) \triangleright\triangleleft E_3 \equiv E_1 \triangleright\triangleleft (E_2 \triangleright\triangleleft E_3),$$
$$(E_1 \underset{F_1}{\triangleright\triangleleft} E_2) \underset{F_2}{\triangleright\triangleleft} E_3 \equiv E_1 \underset{F_1}{\triangleright\triangleleft} (E_2 \underset{F_2}{\triangleright\triangleleft} E_3),$$
3) Каскад проекций
$$\pi_{A1,\ldots,An}(\pi_{B1,\ldots,Bm}(E)) \equiv \pi_{A1,\ldots,An}(E), \text{ если } A \subseteq B.$$
4) Каскад селекций
$$\sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2}(E).$$
5) Перестановка селекций и проекций
$$\sigma_F(\pi_{A1,\ldots,An}(E)) \equiv \pi_{A1,\ldots,An}(\sigma_F(E)).$$

В более общем случае если условие F вовлекает атрибуты $B_1, \ldots, B_m$, которых нет среди $A_1, \ldots, A_n$, то
$$\pi_{A1,\ldots,An}(\sigma_F(E)). \equiv \pi_{A1,\ldots,An}(\sigma_F(\pi_{A1,\ldots,An,B1,\ldots,Bm}(E)).$$
6) Перестановка селекции с декартовым произведением

а) Если в формуле F_1 используются атрибуты E_1.
$$\sigma_{F1}(E_1 \times E_2) \equiv \sigma_{F1}(E_1) \times (E_2).$$

б) Если в формуле F_1 используются атрибуты E_1, а в формуле F_2 – атрибуты E_2.

$\sigma_F(E_1 \times E_2) \equiv \sigma_{F1}(E_1) \times \sigma_{F2}(E_2), F = F_1 \wedge F_2.$

в) Если в формуле F_1 используются атрибуты E_1, а в формуле F_2 – атрибуты E_1 и E_2.

$\sigma_F(E_1 \times E_2) \equiv \sigma_{F2}(\sigma_{F1}(E_1) \times E_2), F = F_1 \wedge F_2.$

7) Перестановка селекции с объединением

$\sigma_F(E_1 \cup E_2) \equiv \sigma_{F1}(E_1) \cup \sigma_{F2}(E_2).$

Предполагается, что если имена атрибутов E_1 отличны от имен E_2, то формула F модифицируется в F_1 и F_2, в которых используются соответствующие имена.

8) Перестановка селекции с разностью

$\sigma_F(E_1 - E_2) \equiv \sigma_{F1}(E_1) - \sigma_{F2}(E_2).$

F модифицируется в F_1 и F_2 аналогично (7).

9) Перестановка проекции с декартовым произведением

$\pi_{A1,\ldots,An}(E_1 \times E_2) \equiv \pi_{B1,\ldots,Bm}(E_1) \times \pi_{C1,\ldots,Ck}(E_2), B \subseteq A, C \subseteq A, n = m + k.$

10) Перестановка проекции с объединением.

$\pi_{A1,\ldots,An}(E_1 \cup E_2) \equiv \pi_{B1,\ldots,Bn}(E_1) \cup \pi_{C1,\ldots,Cn}(E_2).$

Если имена атрибутов E_1 и E_2 отличаются, то надо заменить их на $A_1,\ldots,A_n$

19. Алгоритм Ульмана оптимизации реляционных выражений (с примером).

Используя законы алгебраических преобразований, Дж. Ульман предложил следующий алгоритм оптимизации: исходное реляционное выражение представляется в виде бинарного дерева, узлы которого – двуместные операторы (×, ∪, –).

Шаг 1. Представить каждую селекцию $\sigma_{F1 \wedge \ldots \wedge Fn}(E)$ в виде каскада $\sigma_{F1}(\ldots(\sigma_{Fn}(E))\ldots)$ по правилу 4.

Шаг 2. Переместить каждую селекцию в дереве насколько это возможно вниз, используя законы 4-8.

Шаг 3. Переместить каждую проекцию в дереве вниз, насколько это возможно, используя законы 3, 5, 9, 10. При этом закон 3 приводит к исчезновению некоторых проекций, а закон 5 расщепляет проекцию на 2, одна из которых может перемещаться по дереву вниз. Проекция исключается, если она проецирует выражение на все атрибуты.

Шаг 4. Комбинировать каждый каскад проекций или селекций в одиночную проекцию или селекцию с последующей проекцией, используя законы 3-5.

Шаг 5. Разбиваем внутренние узлы полученного в результате дерева на группы следующим образом: каждый узел, представляющий двуместный оператор (×, ∪, –), принадлежит группе, которая образована из предков с операторами селекции или проекции и потомков данного узла с унарными операторами селекции или проекции, завершающимися в листьях. Исключением является случай, когда используется двуместный оператор × без последующей селекции, так как в этом случае селекция комбинируется с произведением, образуя Θ-соединение.

20. Язык запросов SQL. Компоненты языка. SELECT-запросы для операций реляционной алгебры.

Язык запросов:

Назначение:

- Создание БД и схемы данных, таблиц с полным описание их структуры
- Выполнение операций изменения
- Выборка

Особенности:

- Непроцедурный язык, в котором указывают какие данные нужно получить, а не как их нужно получить
- Используются термины «таблица», «строка», «столбец»
- Допускаются повторяющиеся строки таблицы
- Определяется порядок столбцов
- Допускает сортировка строк
- Допускаются пустые значения

DDL – язык определения данных, предназначен для определения структуры БД и управления доступом к данным

DML – язык манипулирования данными

Базовые конструкции

SELECT [DISTINCT] <список столбцов>

[FROM] <имена таблиц>

[JOIN] <взаимосвязи между таблицами>

[WHERE] <критерии отбора строк>

[GROUP BY] <столбцы для группировки>

[HAVING] <критерии отбора для групп строк>

[ORDER BY] <столбцы для сортировки результатов>

Псевдоним таблицы

*SELECT * FROM books A WHERE A.price>20*

*SELECT * FROM books AS A WHERE A.price>20*

Псевдоним поля таблицы

SELECT money_sum=price FROM books WHERE price<50

SELECT price AS money_sum FROM books WHERE price<50

WHERE: сравнение

*SELECT * FROM R WHERE B = ‘b’*

*SELECT * FROM R WHERE B <> ‘b’*

*SELECT * FROM R WHERE B = ‘b’ OR C<>25*

WHERE: диапазон

*SELECT * FROM R WHERE C >= 10 AND C<=25*

*SELECT * FROM R WHERE C BETWEEN 10 AND 25*

WHERE: принадлежность множеству

*SELECT * FROM R WHERE B=‘b’ OR B=‘c’ OR B=‘d’*

*SELECT * FROM R WHERE B IN (‘b’, ’c’, ’d’)*

WHERE: соответствие шаблону

*SELECT * FROM R WHERE B LIKE ‘b%’*

*SELECT * FROM R WHERE B LIKE ‘b___’*

*SELECT * FROM R WHERE B LIKE ‘%black%’*

WHERE: обработка NULL-значений

IS NULL, IS NOT NULL (стандарт ISO)

*SELECT * FROM titles WHERE price IS NULL*

Реализация операций реляционной алгебры:

- Селекция $\sigma_{B=”b”(}R)$

  *SELECT * FROM R WHERE B = ‘b’*

- Проекция $\pi_{B,C(}R)$

  SELECT DISTINCT B,C FROM R

- Декартово произведение R * S

  *SELECT * FROM R,S*

- Объединение R ∪ S R(A,B,C) S(D,E,F)

  SELECT A,B,C FROM R

  UNION

  SELECT D,E,F FROM S

- Естественное соединение

  R(A,B,C) S(B,C,D)

SELECT DISTINCT R.,S.D FROM R,S*

WHERE R.B = S.B AND R.C = S.C

- Θ – соединение

  SELECT R.,S.* FROM R,S*

  WHERE R.B < S.D

- Разность R – S

  *SELECT * FROM R WHERE NOT*

  *EXISTS (SELECT * FROM S WHERE R.A*

  = S.D AND R.B = S.E AND R.C = S.F)

- Пересечение R ∩ S = R – (R – S)

  *SELECT * FROM R WHERE EXISTS*

  *(SELECT * FROM S WHERE R.A = S.D*

  AND R.B = S.E AND R.C = S.F)

21. Язык запросов SQL. Внешнее и внутреннее соединение в запросах выборки. Вложенные запросы и подзапросы.

Декартово произведение R * S

*SELECT * FROM R,S*

Естественное соединение

R(A,B,C) S(B,C,D)

SELECT DISTINCT R.,S.D FROM R,S*

WHERE R.B = S.B AND R.C = S.C

Θ – соединение

SELECT R.,S.* FROM R,S*

WHERE R.B < S.D

- Эквисоединение (внутреннее соединение) – INNER JOIN
- Внешнее соединение
  - Полное внешнее соединение - FULL [OUTER] JOIN
  - Внешнее соединение слева - LEFT [OUTER] JOIN
  - Внешнее соединение справа - RIGHT [OUTER] JOIN

Внутреннее соединение – включает строки соединяемых таблиц, удовлетворяющие условию соединения

1. WHERE table1.t1_id=table2.t1_id

2. FROM table1 INNER JOIN table2 ON table1.t1_id=table2.t1_id (или просто JOIN ... ON)

- Полное внешнее соединение – включает все строки из двух таблиц, в том числе не имеющие совпадений в «присоединяемой» таблице
  FROM table1 FULL OUTER JOIN table2 ON
  table1.t1_id=table2.t1_id (или FULL JOIN...ON)
- Внешнее соединение слева – включает все строки из таблицы слева и только те строки из таблицы справа, которые удовлетворяют условию соединения
  FROM table1 LEFT OUTER JOIN table2 ON
  table1.t1_id=table2.t1_id (или LEFT JOIN...ON)
- Внешнее соединение справа - включает все строки из таблицы справа и только те строки из таблицы слева, которые удовлетворяют условию соединения
  FROM table1 RIGHT OUTER JOIN table2 ON
  table1.t1_id=table2.t1_id (или RIGHT JOIN...ON)

Вложенные запросы представляют собой выражение SELECT, которые подставляются в выражение FROM другого оператора SELECT.

*SELECT * FROM table1 T1 INNER JOIN*

*(SELECT * FROM table2 T2 WHERE*

T2.attr='value') AS T3 ON T1.t1_id=T3.t1_id

Виды подзапросов:

- Скалярный подзапрос – выбирает одно значение
- Строковый подзапрос – выбирает одну строку данных
- Подзапрос по столбцу – выбирает один столбец данных
- Табличный подзапрос

Скалярный:

SELECT R.a, R.b FROM R

WHERE R.c = (SELECT S.s_id FROM S

WHERE S.s_name='test')

Необходимо, чтобы скалярный подзапрос возвращал строго одно значение!

SELECT R.a, R.b FROM R

WHERE R.c = (SELECT S.s_id FROM S

WHERE S.s_name='test'

LIMIT 1)

Подзапрос по столбцу (операторы ANY/SOME):

ANY/SOME – true, если хотя бы для одной строки условие истинно

SELECT R.a, R.b FROM R

WHERE R.c > ANY (SELECT S.size FROM S

WHERE S.s_name LIKE 'test%')

SELECT R.a, R.b FROM R

WHERE R.c > SOME (SELECT S.size FROM S

WHERE S.s_name LIKE 'test%')

Подзапросы с EXISTS:

Пример: получить список книг, которые не продаются ни в одном из магазинов

SELECT T1.title_id, T1.title

FROM titles T1

WHERE NOT EXISTS

*(SELECT **

FROM titles T2 INNER JOIN sales S ON

T2.title_id=S.title_id

WHERE T2.title_id=T1.title_id)

Табличные подзапросы: оператор IN

Подзапрос по столбцу может встраиваться в основной запрос с помощью оператора in.

1. *Некоррелированный:*

SELECT A FROM R WHERE B + C IN

(SELECT B + C FROM S WHERE D = "d")

2. *Коррелированный (связанный):*

SELECT A FROM R WHERE "d" IN

(SELECT D FROM S WHERE R.B + R.C = B+C)

Подзапросы с IN

Список книг, имя автора которых начинается с «А»

SELECT T.title_id, T.title

FROM titles T INNER JOIN titleauthor TA ON

T.title_id=TA.title_id

WHERE TA.au_id IN

(SELECT A.au_id FROM authors A WHERE

A.au_fname LIKE 'a%')

Связанные подзапросы с IN

Связанный подзапрос выполняется для каждого ряда, удовлетворяющего другим условиям основного запроса.

Пример: выбрать книги ценой менее 20, продажи

которых принесли более 200.

SELECT T.title_id, T.title

FROM titles T

WHERE T.price<20 AND T.title_id IN

(SELECT S.title_id

FROM sales S

*WHERE S.title_id=T.title_id AND S.qty*T.price>200)*

22. Язык запросов SQL. Агрегирующие функции. Группирующие SELECT-запросы. Формирование сводных отчетов.

Агрегирующие функции:

- COUNT(exp) – число значений выражения, не равных NULL
- COUNT(*) – число строк (рядов)
- MAX(exp), MIN(exp) – максимальное и минимальное значение выражения
- SUM(exp) – сумма всех значений выражения
- AVG(exp) – среднее значение выражения

Для всех функций, кроме COUNT(*), обрабатываются только непустые значения (NOT NULL)

Примеры:

- Общее количество книг

SELECT COUNT(*) AS t_count FROM titles

- Книги с максимальной ценой

*SELECT * FROM titles WHERE*

price=(SELECT MAX(price) FROM titles)

DISTINCT может быть указан перед именем столбца для исключения повторяющихся строк из обработки функциями COUNT(), SUM(), AVG().

Количество различных книг, которые были проданы в заданный период

SELECT COUNT(DISTINCT title_id) AS book_count

FROM sales

WHERE sale_date BETWEEN '01.01.2014' AND '31.03.2014'

Группирующие запросы –SELECT-запросы, в которых присутствует конструкция GROUP BY.

В них результаты SELECT-запроса сгруппированы по значениям заданных колонок таблицы, и из каждой группы в результат попадает только одна строка.

В результате получается такой набор строк, в котором любые две строки отличаются друг от друга.

SELECT <список выборки> ...

GROUP BY <колонки группировки>

Каждый элемент в списке выборки должен иметь единственное значение для любой группы.

Колонки группировки должны обязательно включать все поля списка выборки, за исключением агрегатных значений.

В колонки группировки могут входить и дополнительные колонки.

При проведении группировки данных все пустые значения в одной колонке рассматриваются как равные.

Если две строки таблицы в одном и том же группируемом столбце содержат значения NULL и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Ограничения на выполнение группирования

1. Конструкция WHERE выполняется до проведения группировки.

2. Дополнительные ограничения на группы могут быть заданы в конструкции HAVING.

На практике HAVING используется при необходимости отбора данных по значению агрегатных функций, рассчитанных для групп.

Сводные отчеты

GROUP BY ROLLUP(список выражений)

Позволяет указать, что кроме итоговых значений для каждого варианта в списке выражений необходимо рассчитать «обобщенные итоги» по каждому элементу списка выражений. Раскрывает итоги только в порядке следования выражений в списке (по иерархии).

Пример:

Количество проданных экземпляров каждой книги, книг автора, общее количество по всем авторам

SELECT au_lname, t.title_name, SUM(S.qty)

FROM titles T INNER JOIN titleauthor TA ON

T.title_id=TA.title_id INNER JOIN authors A ON

A.au_id=TA.au_id INNER JOIN sales S ON

t.title_id=S.title_id

GROUP BY ROLLUP(au_lname, t.title_name)

GROUP BY CUBE(список выражений)

Позволяет указать, что кроме итоговых значений для каждого варианта в списке выражений необходимо рассчитать «обобщенные итоги» по каждому элементу списка выражений. Раскрывает все возможные комбинации итогов.

Пример:

Количество проданных экземпляров каждой книги в разрезе авторов, названий, магазинов + итоги по авторам, названиям, магазинам

SELECT au_lname, title, stor_name, SUM(qty)

FROM titles T INNER JOIN sales S ON

S.title_id=T.title_id

INNER JOIN titleauthor TA ON TA.title_id=T.title_id

INNER JOIN authors A ON A.au_id=TA.au_id

INNER JOIN stores ST ON S.stor_id=ST.stor_id

GROUP BY CUBE(au_lname, title, stor_name)

ROLLUP(a, b, c)	CUBE(a, b, c)
(a, b, c) (a, b) (a)	(a, b, c) (a, b) (a, c)

()	(a) (b, c) (b) (c) ()

GROUP BY GROUPING SETS(список выражений)

Позволяет рассчитать итоги для каждого уникального значения каждого столбца из списка выражений.

Если в списке оператора GROUPING SETS несколько столбцов заключено во внутренние скобки, они считаются единым выражением.

Пример:

Количество проданных экземпляров по каждой книге, каждому автору, каждому магазину

SELECT au_lname, title, stor_name, SUM(qty)

FROM titles T INNER JOIN sales S ON

S.title_id=T.title_id

INNER JOIN titleauthor TA ON TA.title_id=T.title_id

INNER JOIN authors A ON A.au_id=TA.au_id

INNER JOIN stores ST ON S.stor_id=ST.stor_id

GROUP BY GROUPING SETS(au_lname, title, stor_name)

23. Язык запросов SQL. Запросы модификации данных.

Оператор INSERT предназначен для добавления строк в таблицу.

Основные формы оператора INSERT:

Добавление заданных значений (1 строки)

INSERT [INTO] <таблица>

[список колонок]

VALUES <список значений>

Добавление набора данных (N строк)

Примеры добавления N строк:

Добавление выборки данных INSERT-SELECT

INSERT INTO klient1

*SELECT * FROM klient WHERE*

date_reg>’01.01.2003’

Добавление результатов хранимой процедуры

INSERT-EXEC

INSERT INTO klient1

EXEC sp_get_klients

Выборка в новую таблицу SELECT INTO FROM

SELECT INTO klient1 FROM klient WHERE

date_reg>’01.01.2003’

Оператор UPDATE предназначен для изменения содержимого существующих строк таблицы.

Основная форма UPDATE:

UPDATE <таблица>

SET <имя_колонки1>=<значение>,...

WHERE <условие поиска>

Примеры сложных UPDATE:

UPDATE employee SET salary=

CASE klass

*WHEN 4 THEN salary*2.0*

*WHEN 3 THEN salary*1.5*

*WHEN 2 THEN salary*1.25*

ELSE salary

END

UPDATE titles SET titles.sum_sale=

(SELECT SUM(sales.qty) FROM sales WHERE

titles.title_id=sales.title_id)

Оператор DELETE предназначен для

удаления строк из таблицы.

DELETE [FROM] <таблица>

WHERE <условие>

Удаление всех строк из таблицы

DELETE FROM klients

Удаление всех книг с ценой менее 5

DELETE FROM titles WHERE price<5

Удаление всех книг с ценой ниже средней

DELETE FROM titles WHERE price<(Select AVG(price) from titles)

Оператор TRUNCATE TABLE предназначен для полной

очистки таблицы.

По действию TRUNCATE TABLE аналогичен DELETE FROM без условия, но выполняется намного быстрее.

Например:

TRUNCATE TABLE titleauthor

24. Транзакции в базах данных.

Транзакция – действие или ряд логически связанных действий, выполняемых одним пользователем, который осуществляет чтение или модификацию данных в БД.

Транзакция – логическая единица работы с БД.

Любая транзакция всегда должна переводить БД из одного согласованного состояния в другое. Хотя допускается, что согласованность состояния БД может нарушаться в ходе выполнения транзакция, здесь имеется в виду логическая согласованность данных.

Свойства транзакций:

A – Atomicity (неразрывность) – Любая транзакция представляет собой неделимую логическую операцию, которая может быть выполнена вся целиком, либо не выполнена вообще (обеспечивается СУБД)

C – consistency (согласованность) – каждая транзакция должна переводить БД из одного согласованного состояния в другое, частично это контролирует СУБД за счет механизма ограничения целостности, частично разработчик за счет правильного выстраивания логики транзакции

I – isolation (изолированность) одновременно выполняемые транзакции работают независимо друг от друга, промежуточные результаты одной транзакции не видны

другим до ее завершения (разработчик может управлять уровнем изоляции) Изолированность обеспечивается подсистемой параллельного доступа

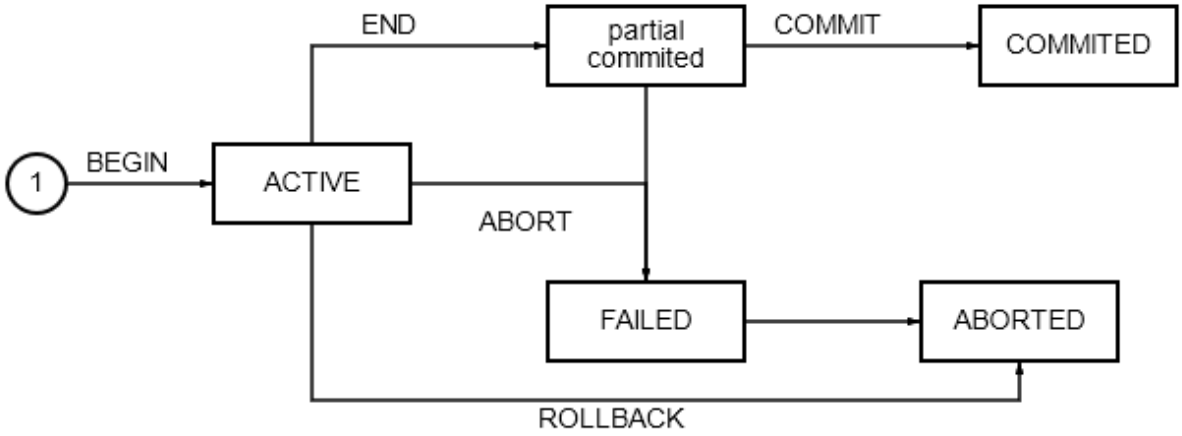
D – durability (устойчивость) – результаты успешно завершенной транзакции должны храниться в БД и не должны быть утеряны в результате последующих сбоев (реализуется подсистемой восстановления СУБД)

Жизненный цикл транзакции:

Begin transaction – оператор считается полученным при поступлении первого оператора SQL

COMMIT – успешное завершение, фиксация результатов

ROLLBACK – отмена, ошибка



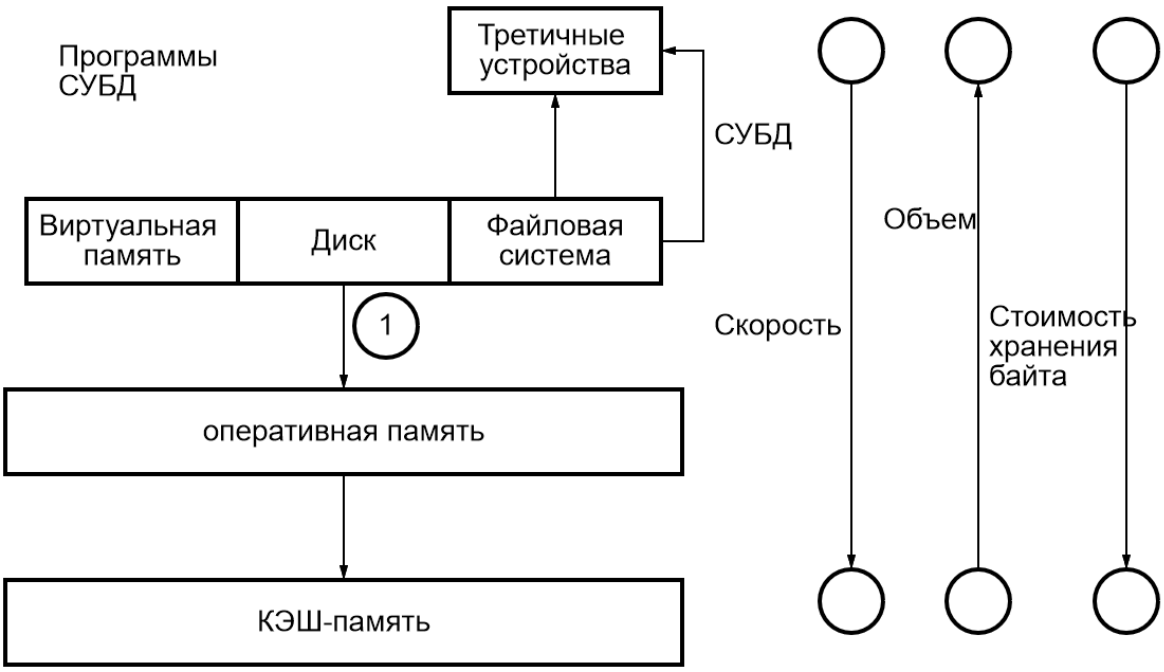
Уровни изоляции:

1. Uncommited read – чтение при неподтвержденной транзакции, позволяет читать любые данные не зависимо от подтверждения.
2. Commited read – чтение при подтверждении транзакции, не допускает чтения грязных данных.
3. Repeatable read – читающая транзакция «не видит» изменения данных, которые были ею ранее прочитаны. При этом никакая другая транзакция не может изменять данные, читаемые текущей транзакцией, пока та не окончена.
4. Serializable – транзакции полностью изолируются друг от друга. Результат выполнения нескольких параллельных транзакций должен быть таким, как если бы они выполнялись последовательно
5. Snapshot – любая другая транзакция будет читать подтвержденные значения в том виде, в каком они существовали непосредственно перед началом выполнения транзакции этого уровня изоляции.

Если транзакция работает в режиме serializable, то все затронутые ею таблицы будут заблокированы. Виды блокировок:

1. Разделяемая блокировка. Разрешает чтение элемента, но не его обновление.
2. Исключительная блокировка. Разрешает и чтение, и обновление.
3. Двухфазная блокировка. Транзакция выполняется по протоколу двухфазной блокировки, если в ней все операции блокировки предшествуют первой операции разблокировки.

25. Принципы хранения данных в БД. Иерархия устройств памяти. Повышение эффективности системы баз данных за счет аппаратных средств.



1. Кэш-память. Работает совместно с процессором. Скорость доступа к данным наивысшая. Данные копируются процессором из ОЗУ в кэш. Результат обработки размещается в кэше, а затем отображается в ОЗУ.
2. ОЗУ. Обеспечивает размещение программ и данных. Устройство оперативной памяти обеспечивает произвольный доступ к ячейкам.
3. Дисковая память (вторичные устройства хранения). В дисковой памяти располагаются файловая система и файл подкачки.
4. Третичные устройства хранения. Устройства для резервного хранения информации. Оперативный доступ к данным не обеспечивается.

Ключевая проблема производительности систем БД сосредоточена в области обмена данными между дисковой и оперативной памятью (1).

В СБД существуют разные механизмы повышения производительности в частности с использованием высокопроизводительных аппаратных средств, использованием эффективных алгоритмов доступа к данным.

Обеспечение производительности на аппаратном уровне.

1. Использование высокопроизводительных дисковых накопителей

2. Использование специализированных дисковых контроллеров. Дает возможность создавать дисковые массивы. За счет применения RAID удается увеличить скорость операций чтение/запись и повысить отказоустойчивость.

Продвинутые функции RAID контроллеров: Имеется мощный специализированный процессор и энергонезависимая кэш-память ~1Гб. Он умеет кэшировать запросы на изменение данных на диске выполнять их в отложенном режиме, также умеет осуществлять предиктивное считывание.

26. Организация данных в БД. Представление и хранение элементов данных. Последовательная индексация.

Каждая БД состоит из множества файлов, в которых размещаются сами данные и информация для эффективной работы с данными (словари, каталоги, индексы).

Элементарной структурной единицей в БД является поле. Поле обеспечивает хранение значения стандартного типа данных, например, Integer, Date. Поля могут быть фиксированной или переменной длины.

Запись – это структура, состоящая из нескольких полей и заголовка. Заголовок содержит дополнительную информацию о записи: дата/время последнего обновления, схема записи и др.

Записи переменной длины содержат дополнительно указатели для отыскания полей, в том числе внешние адреса полей за пределами этой записи.

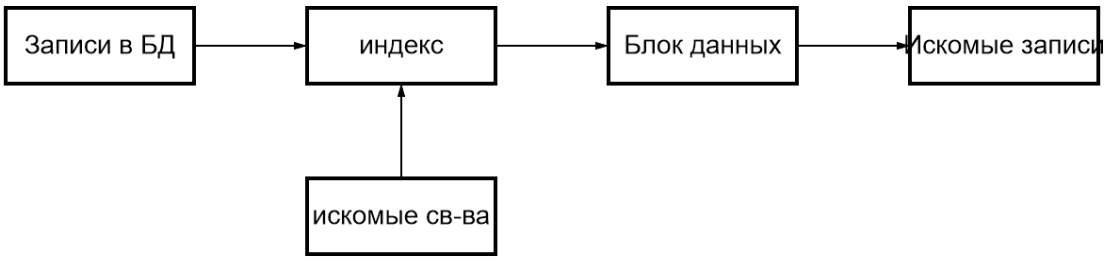
Записи сохраняются внутри более крупных единиц – блоков.

Блок содержит заголовок и множество записей. Является единицей обмена между оперативной и дисковой памятью.

Индексация в БД.

Индекс – некоторая структура данных, которая в качестве «параметра» принимает заданное «свойство» записи и быстро находит записи, обладающие этим свойством.

Под свойством записей понимаются значения одного или нескольких полей.



Удачный индекс позволяет отыскать требуемые записи, обращаясь только к небольшой части записей.

Поля, на основе значений которых индекс конструируется, называют ключом поиска.

В качестве индексов могут быть использованы различные структуры:

- Индексы для последовательных файлов
- Вторичные индексы для неупорядоченных файлов
- Б-деревья
- Хэш-таблицы

Индексы для последовательных файлов.

«Последовательный файл» - структура, в которой записи упорядочены по ключу поиска.

Индекс – это структура, содержащая значение ключа поиска и указатель на запись данных, содержащую это значение ключа поиска.

Плотный индекс содержит запись для каждого значения ключа поиска.

Разряженный индекс содержит записи не для каждого значения ключа поиска, например, только по одному значению на каждый блок данных.

Ключами индекса в этом случае являются ключевые значения первых записей блоков данных.

Многоуровневые индексы.

Если построить индекс для рассмотренных ранее индексов, то получим индекс второго уровня, при этом индекс 1-го уровня может быть как плотным, так и разряженным, а 2-го только разряженным.

Можно построить индексы 3-го и более уровня для ускорения поиска.

Проблема многоуровневого индекса состоит в поддержании баланса между количеством записей данных и количеством уровней.

«Последовательный файл» с повторяющимися значениями ключа поиска

В этом случае индекс содержит запись для каждого значения ключа, то есть является плотным.

Индексы используются при выполнении операций в БД:

- Поиск по значению ключа
- Группировка
- Сортировка
- Distinct
- Соединение по совпадению атрибутов

Накладные расходы на поддержку:

- Индексы хранятся в БД, занимают дополнительное место

- Требуют перестроения при модификации данных

27. Способы доступа к данным: вторичные индексы, B-деревья, хэш-таблицы.

Вторичные индексы.

В ситуации, когда данные неупорядочены по ключу поиска, индекс называется вторичным.

Плотный вторичный индекс содержит запись для каждой записи с данными.

Могут быть применены индексы 2-го и более высокого уровня.

Для сокращения объема хранимых данных в индексе вводится вспомогательная структура, называемая сегментом указателей, она занимает место между индексом и файлом данных.

Сегменты указателей подменяют собой 1-ый плотный уровень индексов.

Б-деревья

Б-дерево является механизмом для поддержания баланса в индексах, его листьями являются указатели на записи данных, а промежуточными узлами записи в индексах.

Ключевые особенности:

- Сбалансированность. Длины всех путей от корневой вершины до любой из вершин листьев равны.
- Автоматическая поддержка количества уровней индексации, отвечающего размеру индексируемого файла данных
- Эффективное управление размером свободных областей внутри блоков, исключение необходимости использования блоков переполнения. Уровень заполнения более 50%.

Каждому Б-дереву поставлен в соответствие параметр n. n – количество значений ключа поиска, размещаемых в одном блоке Б-дерева, при этом блок содержит также n+1 указателей.

В индексных структурах на основе Б-деревьев листы заполняются следующим образом: первые n полей указывают на записи с указанным индексом, n+1 поле указывает на следующий лист.

Последний уровень Б-дерева (листья) – плотный индекс.

Промежуточные блок содержат копии ключей.

Хэш-таблицы

Хэшированием или хэш-индексированием называется технология быстрого прямого доступа к хранимой записи на основе заданного значения некоторого поля, при этом совсем не обязательно, чтобы поле было ключевым.

Недостаток индексных схем состоит в том, что для обнаружения записей необходимо обращаться к индексам. Хэширование избавляет от необходимости поддерживать и просматривать индексы. Хэширование отличается от индексирования тем, что в файле может быть любое количество индексов, но только одно хэш-поле.

Во избежание неэффективного использования дискового пространства следует найти такую хэш-функцию, чтобы можно было сузить диапазон до оптимальной величины с учетом возможности резервирования дополнительного пространства.

Основные особенности технологии хэширования:

- Каждая хранимая запись БД размещается по адресу, который вычисляется с помощью специальной хэш-функции на основе значения некоторого поля данной записи, т.е. хэш-поля, а вычисленный адрес называется хэш-адресом;
- Для сохранения записи в СУБД сначала вычисляется хэш-адрес новой записи, после чего диспетчер файлов помещает эту запись по вычисленному адресу;
- Для извлечения нужной записи по заданному значению хэш-поля в СУБД сначала вычисляется хэш-адрес, а затем диспетчеру файлов посылается запрос на извлечение записи по вычисленному адресу.

Недостатки:

- Физическая последовательность записей внутри хранимого файла почти всегда отличается от последовательности ключевого поля, а также любой другой логически заданной последовательности, а между последовательно размещенными записями могут быть промежутки неопределенной протяженности. Практически всегда физическая последовательность записей в хранимом хэшированием файле иная по сравнению с заданной в нем логической последовательностью.
- Возможность возникновения ситуаций, когда две или более различных записей имеют одинаковые адреса, поэтому иногда возникает необходимость функцию исправлять.

С увеличением размера хранимого файла количество совпадений адресов увеличивается, что приводит к значительному увеличению времени на поиск информации в наборах конфликтующих записей. Это можно устранить, если реорганизовать файл, т.е. загрузить данный файл, используя новую хэш-функцию, что решается при помощи расширяемого хэширования. При его использовании необходимо, чтобы все значения хэш-поля были уникальны, а это может быть реализовано только в том случае, если хэш-поле является ключевым.

28. Основные понятия и развитие моделей вычислений. Модели централизованного управления, автономных персональных вычислений, файл-сервер.

Под моделью вычислений подразумевают совокупность аппаратно-программных средств, схему их взаимодействия между собой и пользователями.

Исторически одной из первых моделей вычислений является модель с использованием централизованной хост-ЭВМ. В такой схеме вычислений пользователь получает доступ к вычислительным ресурсам ЭВМ через сеть неинтеллектуальных терминалов (т.е. терминалов, не обладающих никакими вычислительными возможностями). Центральный компьютер полностью отвечает за взаимодействие с пользователем и управление данными в многопользовательской среде.

Преимуществом такой модели вычислений является их централизация. Централизованные системы позволяют совместно использовать вычислительные ресурсы (диски, принтеры, оперативную память) с высокой эффективностью, а также обеспечивать высокую надежность и актуальность хранимых данных.

Самым большим недостатком такой схемы вычислений является линейная зависимость вычислительной мощности центральной ЭВМ от числа пользователей и, как следствие, высокая стоимость аппаратуры и программного обеспечения. Несмотря на устойчивую тенденцию снижения стоимости оборудования, такие системы по-прежнему остаются одними из дорогостоящих (отношение "цена/производительность" остается достаточно высокой).

Модель автономных персональных вычислений

Преимуществом такой модели вычислений является их автономность в использовании вычислительных ресурсов, т.е. централизованное использование компьютера, но на рабочем месте и независимо от других таких же компьютеров.

Однако у независимых персональных вычислений есть и свои проблемы. Эти проблемы порождают распределенность данных (невозможность совместной работы с данными различных пользователей) по персональным компьютерам в случае, когда эти данные должны использоваться совместно в рамках одной организации.

Проблемы совместного использования данных, расположенных на персональных компьютерах, привели к разработке концепции локальной вычислительной сети, которая восстанавливает преимущества коллективных вычислений и сохраняет простоту использования персональных компьютеров. Наличие вычислительной сети компьютеров характерно для всех моделей распределенных вычислений.

Модель вычислений "файл-сервер" (или архитектура "файл-сервер")

Основывается на понятии сервера. С одной стороны, сервер есть узел вычислительной сети (компьютер с сети), предназначенный для предоставления совместно используемых ресурсов и услуг, а с другой – программный компонент, предоставляющий общий функциональный сервис другим программным компонентам вычислительной сети.

Файловый сервер является обычно центральным узлом сети, на котором хранятся файлы коллективного пользования и который является также концентратором совместно используемых периферийных устройств (например, принтера или дискового накопителя большой емкости). Файловый сервер не принимает участия в обработке приложения. Он выполняет сетевой транспорт совместно используемых данных (часто пересылая файл целиком конечному пользователю).

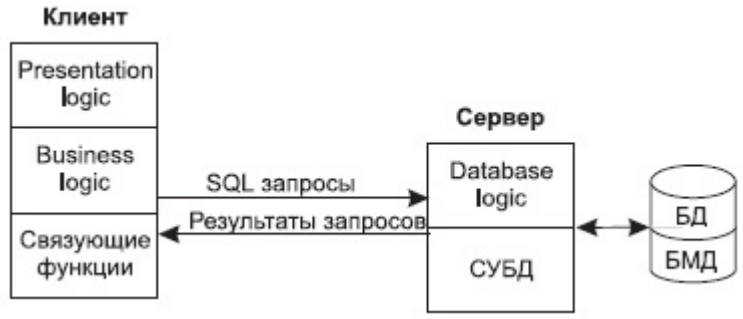
Преимуществом такой модели является, несомненно, корпоративное использование территориально распределенных вычислительных ресурсов, имеющее одним из своих следствий создание глобальных вычислительных систем и новых технологий обмена информацией.

Однако у такой модели есть два крупных недостатка при разработке многопользовательских приложений. Интенсивный обмен данными (рост трафика сети) приводит к быстрому достижению ее пропускной способности и тем самым к снижению (из-за увеличения времени реакции приложения за счет времени ожидания) производительности многопользовательской системы.

Другая проблема – это обеспечение согласованности данных, т.е. одновременного разделения доступа к одним и тем же данным группой пользователей. Обычно файл блокируется для других пользователей, когда его начинает обрабатывать приложение. В случае, когда часть файла реплицируется на конечный узел для обработки, снижается актуализация данных, что может быть неприемлемо для систем оперативной обработки информации.

29. Технология клиент-сервер. Модель доступа к удаленным данным.

В модели удаленного доступа (Remote Data Access, RDA) база данных хранится на сервере. На сервере же находится ядро СУБД. На клиенте располагается презентационная логика и бизнес-логика приложения. Клиент обращается к серверу с запросами на языке SQL.



Преимущества данной модели:

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгрузил сервер БД, сводя к минимуму общее число процессов в операционной системе;

- сервер БД освобождается от несвойственных ему функций; процессор или процессоры сервера целиком загружаются операциями обработки данных, запросов и транзакций. (Это становится возможным, если отказаться от терминалов, не располагающих ресурсами, и заменить их компьютерами, выполняющими роль клиентских станций, которые обладают собственными локальными вычислительными ресурсами);
- резко уменьшается загрузка сети, так как по ней от клиентов к серверу передаются не запросы на ввод-вывод в файловой терминологии, а запросы на SQL, и их объем существенно меньше. В ответ на запросы клиент получает только данные, релевантные запросу, а не блоки файлов, как в ФС-модели.

Основное достоинство RDA-модели — унификация интерфейса "клиент-сервер", стандартом при общении приложения-клиента и сервера становится язык SQL.

Недостатки:

- все-таки запросы на языке SQL при интенсивной работе клиентских приложений могут существенно загрузить сеть;
- так как в этой модели на клиенте располагается и презентационная логика, и бизнес-логика приложения, то при повторении аналогичных функций в разных приложениях код соответствующей бизнес-логики должен быть повторен для каждого клиентского приложения. Это вызывает излишнее дублирование кода приложений;
- сервер в этой модели играет пассивную роль, поэтому функции управления информационными ресурсами должны выполняться на клиенте. Действительно, например, если нам необходимо выполнять контроль страховых запасов товаров на складе, то каждое приложение, которое связано с изменением состояния склада, после выполнения операций модификации данных, имитирующих продажу или удаление товара со склада, должно выполнять проверку на объем остатка, и в случае, если он меньше страхового запаса, формировать соответствующую заявку на поставку требуемого товара. Это усложняет клиентское приложение, с одной стороны, а с другой — может вызвать необоснованный заказ дополнительных товаров несколькими приложениями.

30. Технология клиент-сервер. Модель сервера базы данных.

Данную модель поддерживают большинство современных СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Основу данной модели составляет механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называется механизмом поддержки доменной структуры.

В этой модели бизнес-логика разделена между клиентом и сервером. На сервере бизнес-логика реализована в виде хранимых процедур — специальных программных модулей, которые хранятся в БД и управляются непосредственно СУБД. Клиентское приложение обращается к серверу с командой запуска хранимой процедуры, а сервер выполняет эту процедуру и регистрирует все изменения в БД, которые в ней предусмотрены. Сервер возвращает клиенту данные, релевантные его запросу, которые требуются клиенту либо для вывода на экран, либо для выполнения части бизнес-логики, которая расположена на клиенте. Трафик обмена информацией между клиентом и сервером резко уменьшается.

Централизованный контроль в модели сервера баз данных выполняется с использованием механизма триггеров. Триггеры также являются частью БД.

Ядро СУБД проводит мониторинг всех событий, которые вызывают созданные и описанные триггеры в БД, и при возникновении соответствующего события сервер запускает соответствующий триггер. Каждый триггер представляет собой также некоторую программу, которая выполняется над базой данных. Триггеры могут вызывать хранимые процедуры.

Механизм использования триггеров предполагает, что при срабатывании одного триггера могут возникнуть события, которые вызовут срабатывание других триггеров.

В данной модели сервер является активным, потому что не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД.

И хранимые процедуры, и триггеры хранятся в словаре БД, они могут быть использованы несколькими клиентами, что существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях.

Для написания хранимых процедур и триггеров используется расширение стандартного языка SQL, так называемый встроенный SQL.

Недостатком данной модели является очень большая загрузка сервера. Действительно, сервер обслуживает множество клиентов и выполняет следующие функции:

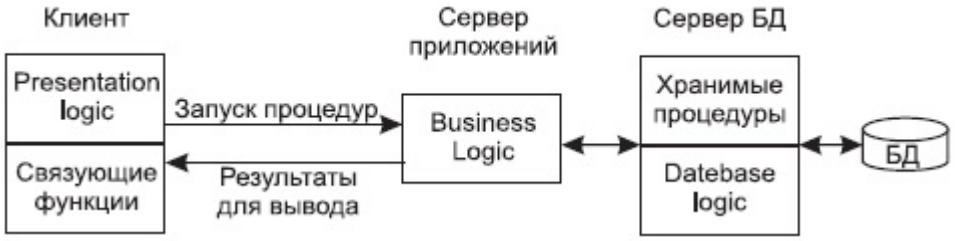
- осуществляет мониторинг событий, связанных с описанными триггерами;

- обеспечивает автоматическое срабатывание триггеров при возникновении связанных с ними событий;
- обеспечивает исполнение внутренней программы каждого триггера;
- запускает хранимые процедуры по запросам пользователей;
- запускает хранимые процедуры из триггеров;
- возвращает требуемые данные клиенту;
- обеспечивает все функции СУБД: доступ к данным, контроль и поддержку целостности данных в БД, контроль доступа, обеспечение корректной параллельной работы всех пользователей с единой БД.

31. Технология клиент-сервер. Модель сервера приложений. Модель тонкого клиента.

Модель сервера приложений

Эта модель является расширением двухуровневой модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Этот промежуточный уровень содержит один или несколько серверов приложений.



В этой модели компоненты приложения делятся между тремя исполнителями:

- Клиент обеспечивает логику представления, включая графический пользовательский интерфейс, локальные редакторы; клиент может запускать локальный код приложения клиента, который может содержать обращения к локальной БД, расположенной на компьютере-клиенте. Клиент исполняет коммуникационные функции front-end части приложения, которые обеспечивают доступ клиенту в локальную или глобальную сеть. Дополнительно реализация взаимодействия между клиентом и сервером может включать в себя управление распределенными транзакциями, что соответствует тем случаям, когда клиент также является клиентом менеджера распределенных транзакций.
- Серверы приложений составляют новый промежуточный уровень архитектуры. Они спроектированы как исполнения общих незагружаемых функций для клиентов. Серверы приложений поддерживают функции клиентов как частей взаимодействующих рабочих групп, поддерживают сетевую доменную операционную среду, хранят и исполняют наиболее общие правила бизнес-логики, поддерживают каталоги с данными, обеспечивают обмен сообщениями и поддержку запросов, особенно в распределенных транзакциях.

- Серверы баз данных в этой модели занимаются исключительно функциями СУБД: обеспечивают функции создания и ведения БД, поддерживают целостность реляционной БД, обеспечивают функции хранилищ данных (warehouse services). Кроме того, на них возлагаются функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и поддержки устаревших (унаследованных) приложений (legacy application).

Отметим, что эта модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных. В этой модели большая часть бизнес-логики клиента изолирована от возможностей встроенного SQL, реализованного в конкретной СУБД, и может быть выполнена на стандартных языках программирования, таких как C, C++. Это повышает переносимость системы, ее масштабируемость.

Тонкий клиент — вид клиента, который может переносить выполнение задач по обработке информации на сервер, не применяя свои мощности по вычислению для их внедрения. Все вычислительные ресурсы подобного клиента максимально ограничены, важно, чтобы их хватало для старта нужного сетевого ПО, применяя, к примеру, веб-интерфейс.

Одним из наиболее распространенных примеров такого типа клиента считается ПК с заранее установленным веб-браузером, который применяется для функционирования с веб-програмами.

Характерная черта тонких клиентов — применение терминального режима функционирования. В такой ситуации, терминальный сервер применяется для процесса отправки и получения информации пользователя, что и является базовым отличием от процесса независимой обработки информации в толстых клиентах.

Плюсы тонкого клиента:

- Минимальное аппаратное обслуживание;
- Низкий риск возникновения неисправности;
- Минимальные технические требования к аппаратному оборудованию.

Негативные стороны:

- При сбое на сервере «пострадают» все подключенные пользователи;
- Нет возможности работать без активного подключения к сети;
- При взаимодействии с большим массивом данных может снижаться объем производительности основного сервера.