1. Сферы применения баз данных и СУБД

База данных (БД) – совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы, и отображающих состояние объектов и их взаимосвязей в данной предметной области.

БД применяется там, где требуется организованное хранение данных, доступ к ним и их обновление.

Система управления базами данных (СУБД) – это комплекс языковых программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями.

СУБД применяются везде, где важен удобный и систематизированный доступ к данным, обеспечение их безопасности и целостности.

Применение БД и СУБД:

Бизнес и корпоративное управление:

* Управление базами клиентов.
* Учёт товаров, складов, логистика (например, управление цепочками поставок).

IT и разработка программного обеспечения:

* Хранение и работа с данными для web- и мобильных приложений.
* Реализация сервисов авторизации, аналитики и логирования.
* Организация облачных решений и распределённых систем.

Государственное управление:

* Регистрация граждан (например, базы данных паспортов и налогов).
* Электронные госуслуги, базы данных голосований.
* Учёт бюджетов, инфраструктурных объектов.

Финансовый сектор:

* Ведение транзакций в банках и платёжных системах.
* Автоматизация расчётов, управление счетами.

Здравоохранение:

* Системы электронных медицинских карт.
* Базы данных для фармацевтики, клинических исследований.
* Управление расписаниями врачей, оборудованием и запасами лекарств.

Образование:

* Управление студенческими записями, оценками, расписаниями.
* Онлайн-обучение: базы данных курсов, тестов, материалов.
* Системы тестирования и сертификации.

Развлечения и СМИ:

* Стриминговые платформы (управление коллекциями фильмов, сериалов, музыки).
* Учёт просмотров, лайков, пользовательских предпочтений.
* Рекомендательные системы.

Транспорт и логистика:

* Управление расписаниями транспорта, маршрутами.
* Трекинг грузов, учёт перевозок.
* Автоматизация бронирования билетов.

Научные исследования и аналитика:

* Системы анализа больших данных (Big Data).
* Хранение и обработка экспериментальных данных.
* Управление научными проектами и ресурсами.

Социальные сети и коммуникации:

* Управление аккаунтами, профилями, перепиской.
* Организация взаимодействия между пользователями.
* Анализ поведения аудитории.

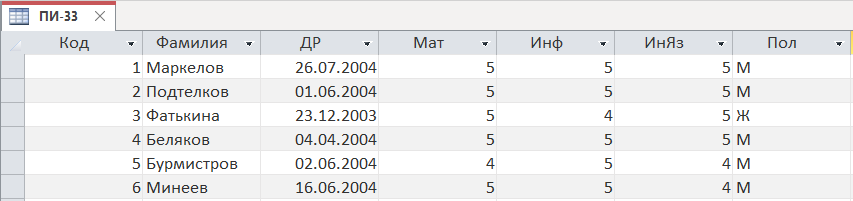
Производство:

* Учёт и управление оборудованием, материалами, процессами.
* Автоматизация сборочных линий.
* Мониторинг качества продукции.

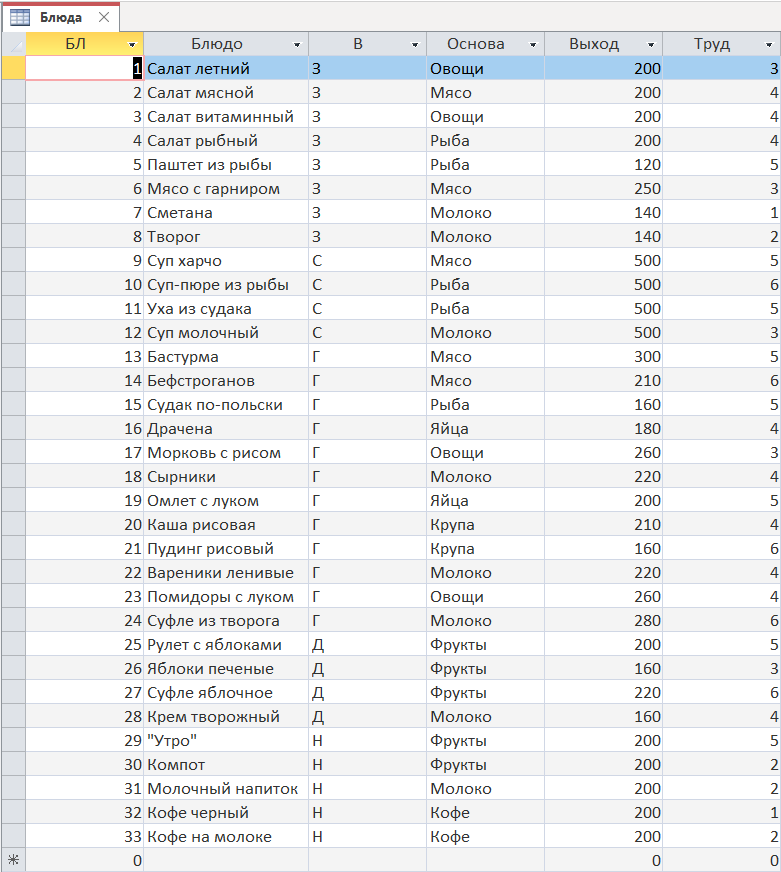
Личный опыт использования:

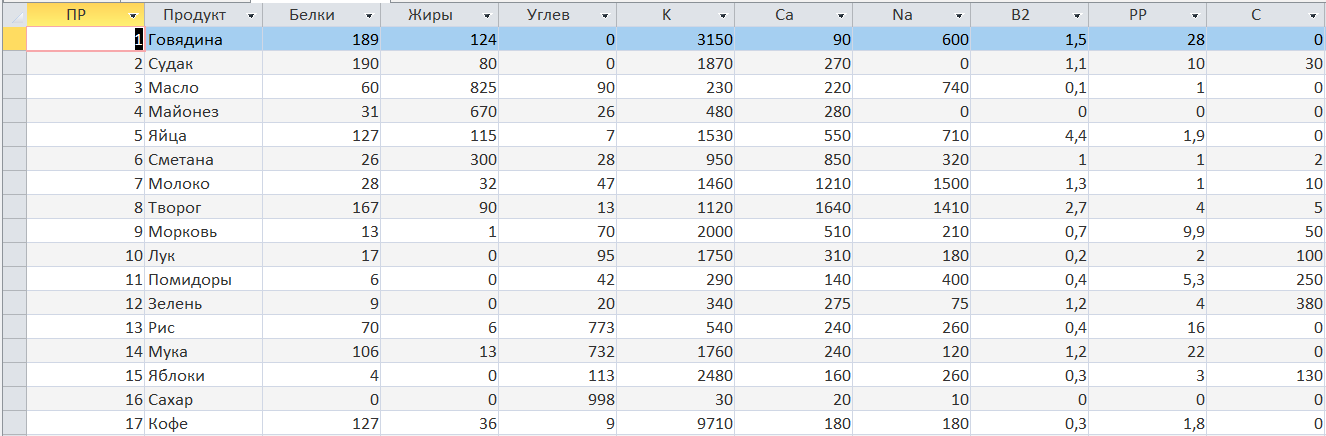
В ходе выполнения лабораторных работ по дисциплине «Базы данных» мы работали с примерами различных БД, аналоги которых могут применяться в различных организациях:

* База данных с информацией о студентах – фамилии, дата рождения, пол, успеваемость. Аналогичные БД могут использоваться в электронных журналах, ЕИС «Тандем» (личный кабинет ЧГУ) и т. д.

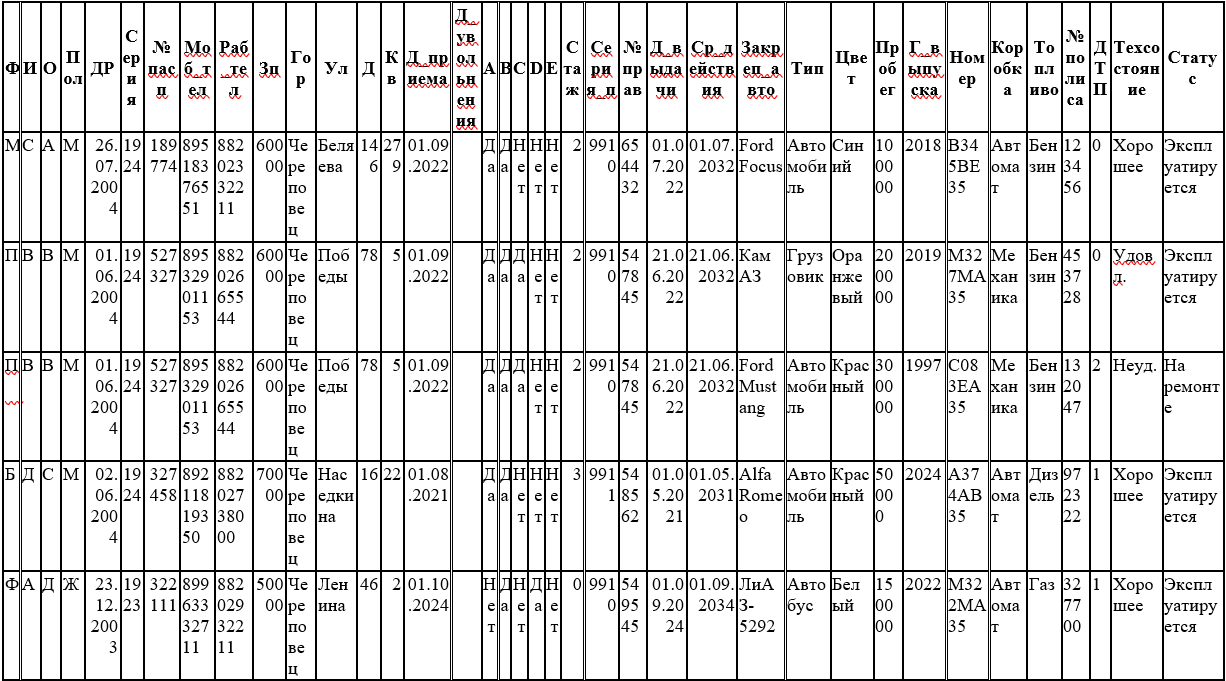


* База данных пансионата с информацией о блюдах, продуктах, поставщиках, рецептах и т. д. Может использоваться в любых заведениях общепита – кафе, ресторанах, столовых и т. д.





* База данных автопарка с информацией о водителях и автомобилях. Подобные БД могут использоваться в различных предприятиях, занимающихся пассажирскими или грузовыми перевозками (например, «Автоколонна №1456»)



Помимо ЛР, мы могли встречать БД и в других случаях:

* Телефонная книга – БД, содержащая информацию о контактах – имена, телефоны, возможно еще какие-то данные.
* Каталоги библиотек – содержат информацию о книгах: название, автор, год издания, издательство и т.д.
* Онлайн-бронирование билетов на концерты/спектакли/фильмы, на самолет/поезд/автобус или номеров в отеле.
* Онлайн-запись к врачу.

1. Базы данных и файловые системы. Области применения файлов. Потребности информационных систем.

База данных (БД) – совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы, и отображающих состояние объектов и их взаимосвязей в данной предметной области.

Банк данных – это система, состоящая из одной или нескольких баз данных, системы управления базами данных (СУБД), словаря данных, администратора БД (АБД), самой вычислительной техники, на которой все это реализовано.

Администратор(ы) БД – лицо (или группа лиц), отвечающее за выработку требований к БД, ее проектирование, создание, эффективное использование, сопровождение. Он следит за функционированием информационной системы, обеспечивает защиту от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность, достоверность информации.

Файл – именованная область внешней памяти, куда можно записать, и откуда считать данные.

Область применения файлов:

* Текстовые документы - последовательность записей со строками текста и байт со специальными символами.
* Объектные модули - последовательность записей или байт, определяемых системой программирования и неизвестных файловой системе.
* Содержащие образы выполняемых программ - структура известна редактору связи и загрузчику ОС.
* Файлы с графической, звуковой информацией, файлы документов.

Файлы БД могут иметь различные форматы. Например, в ходе выполнения ЛР мы использовали:

* .accdb – файл базы данных, используемый современными версиями Access
* .mdb – файл базы данных, используемый старыми версиями Access
* .dbf – файл базы данных, используемый старыми СУБД (например, Rebus).

Также популярным форматом файлов БД является .db. Есть и более редкие форматы, использующиеся в конкретных СУБД.

Файловая система (ФС) – это способ организации, хранения и управления данными на носителях информации (жёстких дисках, SSD, флешках и т. д.). Она определяет, как данные записываются, структурируются, хранятся и считываются на устройстве.



Конкретная файловая система определяет правила именования файлов, структуру файлов, способ доступа к данным файла. Она же распределяет внешнюю память, отображает имена файлов в соответствующие адреса ВЗУ, обеспечивает доступ к данным.

Существует 2 подхода к реализации файлов:

* ОС фирмы DEC: файл – это последовательность байт постоянного или переменного размера. Ее можно читать, записывать последовательно или переходить на запись с указанным номером. Записи структурированы на поля. Поля можно объявлять ключевыми
* ОС фирмы Unix: файл – это последовательность байт, из файла читают указанное число байт, начиная с начала либо с указанного байта

Преимущества ФС:

* Файловые системы позволяют эффективно и упорядоченно хранить файлы, поддерживают доступ к ним, обеспечивают безопасность.
* Есть разные файловые системы, каждая из которых имеет свои плюсы в зависимости от ситуации (например, стабильность, скорость работы, поддержка больших объемов данных и т.д.).
* Файловые системы являются основой взаимодействия между операционными системами и жесткими дисками, так что без них невозможно представить нормальное функционирование системы.

Недостатки ФС:

* У каждой файловой системы есть свои ограничения (например, размер файла, поддерживаемое количество файлов в каталоге).
* Некоторые файловые системы плохо работают на старых или нестандартных устройствах.
* Некоторые ОС могут не поддерживать все типы файловых систем.
* Файловые системы могут быть подвержены повреждениям (например, из-за сбоев питания или некорректного завершения работы), что может привести к потере данных или их повреждению.

Примеры ФС:

* FAT32 – широко используется в флешках, SD-картах и т. п. Поддерживается большинством ОС, включая Windows, MacOS, Linux. Максимальный размер файла – 4 Гб
* NTFS – основная файловая система Windows. Поддерживает большие файлы и разделы.
* exFAT – используется для смартфонов, камер, флешек и SD-карт. В отличие от FAT32, поддерживает большие файлы и тома. Совместимо с Windows и MacOS.

Информационная система (ИС) – это совокупность взаимосвязанных элементов (людей, оборудования, программного обеспечения, данных и методов), предназначенных для сбора, хранения, обработки, анализа и передачи информации для достижения определённых целей.

Функции информационных систем:

* Сбор данных: автоматизированный или ручной ввод информации.
* Хранение: организация информации в базах данных или других структурах.
* Обработка: анализ данных, вычисления, построение отчётов.
* Передача: обмен информацией между пользователями или другими системами.
* Вывод данных: предоставление результатов обработки в удобной форме (таблицы, графики, отчёты).

Информационные системы призваны хранить, выбирать, модифицировать информацию. Структура информации различна, но есть общие черты. Сначала вопрос структуризации решался написанием надстроек над файловыми системами. Они повторяли друг друга. На пути унификации появляются СУБД как средство, выделившее и обобщившее ту часть информационной системы, что отвечает за управление сложноструктурированными данными.

Основные потребности информационных систем:

1. Хранение данных
2. Обеспечение доступа к данным
3. Обработка данных
4. Безопасность
5. Масштабируемость
6. Интеграция
7. Надёжность
8. Удобство использования
9. Анализ и поддержка принятия решений
10. Автоматизация процессов
11. Актуальность данных
12. Основные функции СУБД. Типовая организация современной СУБД.

Система управления базами данных (СУБД) – это комплекс языковых программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями.

Функции СУБД:

1. Непосредственое управление файлами БД – обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей. В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти.
2. Управление буферами ОЗУ - СУБД обычно работают с БД, размеры которого существенно больше доступного объема оперативной памяти. Если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Способом реального увеличения этой скорости является буферизация данных в оперативной памяти.
3. Управление транзакциями

Транзакция - последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется и СУБД фиксирует изменения БД, произведенные этой транзакцией, либо ни одно из этих изменений не отражается на состоянии БД. На основе транзакций поддерживается логическая целостность БД.

Транзакция воспринимается единицей активности пользователя. Для многопользовательских систем, пока первый пользователь обращается к БД, другому пользователю доступа нет.

1. Нормализация – декомпозиция отношения, находящегося в предыдущей НФ в два или более отношений, удовлетворяющих требованиям следующей НФ.
2. Журнализация. Одно из требований к СУБД - надежность сохранения данных. Она должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного и программного сбоя. Для восстановления БД нужна дополнительная информация. Она хранится в журнале изменений БД.

Журнал – это особая часть БД, недоступная пользователям СУБД, поддерживаемая с особой тщательностью. Он содержит записи обо всех изменения в основной части БД. Известны 2 вида журналов: локальный и общесистемный. После мягкого сбоя используется локальный журнал, после жесткого – общесистемный.

1. Поддержка языков БД

SQL (Structured Query Language) - язык структурированных запросов

SQL = DDL + DML + DQL

DDL (Data Definition Language) - язык описания данных

DML (Data Manipulation Language) - язык манипуляции данными

DQL (Data Query Language) - язык запросов данных

Типовая организация современной СУБД:

* Внутренняя часть СУБД (ядро СУБД, DBE - DataBase Engine) – отвечает за управление данными во внешней памяти, управление ОЗУ, управление транзакциями, журнализацию. Обладает собственным интерфейсом, который недоступен пользователям, но используется в программах
* Компилятор языка. Проблема современных СУБД в том, что язык SQL – не процедурный. Поэтому, компилятор должен решать, как выполнить оператор, прежде чем произвести программу. Результат компиляции представлен в машинных кодах или во внутреннем машиннонезависимом коде
* Подсистема поддержки времени выполнения. Направлена на выполнение машиннонезависимого кода. Является интерпретатором
* Утилиты – процедуры, которые накладно выполнять с использованием языка БД: загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности и т. д.

1. Ранние подходы к организации БД.

Под ранними подходами понимается, что они ранние относительно современных подходов к организации БД, которые стали развиваться в конце XX – начале XXI века. Сегодня распространены реляционные СУБД.

Ранние подходы исторически предшествовали реляционным СУБД. На основе методов, используемых в них, организованы и реляционные СУБД. Их знания полезны для определения путей развития СУБД.

Общие характеристики:

* Они использовались в течение длительного времени до появления реляционных СУБД. Были накоплены огромные БД, их надо было использовать совместно с реляционными
* Они не строились на абстрактных моделях
* В них был интерактивный доступ на уровне записей. Это требовало писать прикладные программы со своим интерфейсом

История развития БД

В 1940–1950-х годах компьютеры использовались для обработки данных, которые хранились в виде отдельных файлов. Каждое приложение работало со своими файлами напрямую. Это вызывало ряд проблем: отсутствие стандартизации, дублирование данных, сложность поиска и модификации информации.

В 1950-1960-х годах с развитием ЭВМ появились файловые системы как первый шаг к управлению данными. Они обеспечивали базовые функции: хранение, доступ и обновление данных. Однако, файлы и программы были жестко связаны, возникали сложности при изменении структуры данных, а также была ограничена масштабируемость.

В 1960-х годах началась работа над системами, которые обеспечивали бы более гибкое управление данными.

1. Системы, основанные на инвертированных списках:

Дата появления: 1950-1960-е гг.

Представители: модуль GIN в PostgreSQL, расширение Oracle Text для Oracle, движок InnoDB для MySQL

Организация доступа к данным применяется и в современных СУБД на основе индексов. Структура данных похожа на реляционные СУБД, но таблицы и пути доступа к ним видны пользователям. При этом строки таблиц упорядочены в некоторой физической последовательности. Эта упорядоченность может определяться и для всей БД. Может быть определено произвольное число ключей поиска. Индексы автоматически поддерживаются системой и видны пользователям.

Манипулирование данными – было реализовано 2 класса операторов:

* Операторы, устанавливающие адрес записи (прямые поисковые операторы и операторы, находящие записи относительно другой).
* Операторы работы над записями.

Правила ограничения целостности тогда не было, СУБД это не делала, за это отвечал программист

2. Иерархические СУБД

Дата появления: 1960-е гг.

Представители: IBM IMS, Windows Registry, System 2000

Структура данных: такие БД состоят из упорядоченного набора деревьев. Дерево состоит из одной корневой записи и упорядоченного набора поддеревьев. Каждое дерево - иерархически организованный набор записей. Запись-потомок должен иметь одного предка. Все потомки с общим предком называются близнецами.

В СУБД использовались понятия "сегмент" вместо "запись", а запись БД – это все дерево сегментов.

Операции манипулирования данными:

* Найти указанное дерево
* Перейти от одного дерева к другому
* Перейти от одной записи к другой внутри дерева
* Вставить запись
* Удалить запись
* И т. д.

Работала целостность ссылок между предками и потомками: никакой потомок не может существовать без родителя

3. Сетевые СУБД:

Дата появления: 1960-е гг.

Представители: IDS, CA-IDMS, TurboIMAGE, DMS-1100

Сетевой подход расширяет иерархический: потомок может иметь любое число предков. Сетевая БД состоит из записей и связей между ними. Для связи предка с потомком выполняются условия:

* Каждый предок является таковым только в одной связи
* Каждый потомок является таковым только в одной связи

Операции манипулирования данными:

* Найти конкретную запись
* Перейти от предка к одному потомку по некоторой связи
* Перейти к следующему потомку в связи
* Включить в связь
* Исключить из связи
* И т. д.

Реляционные СУБД появились в 1970-х годах, их концепция сформулирована Эдгаром Коддом.

Плюсы ранних подходов:

* У них развитые средства управления данными в ВЗУ на низком уровне
* Возможность построения вручную эффективных прикладных программ
* Возможность экономии памяти за счет разделения объектов

Минусы ранних подходов:

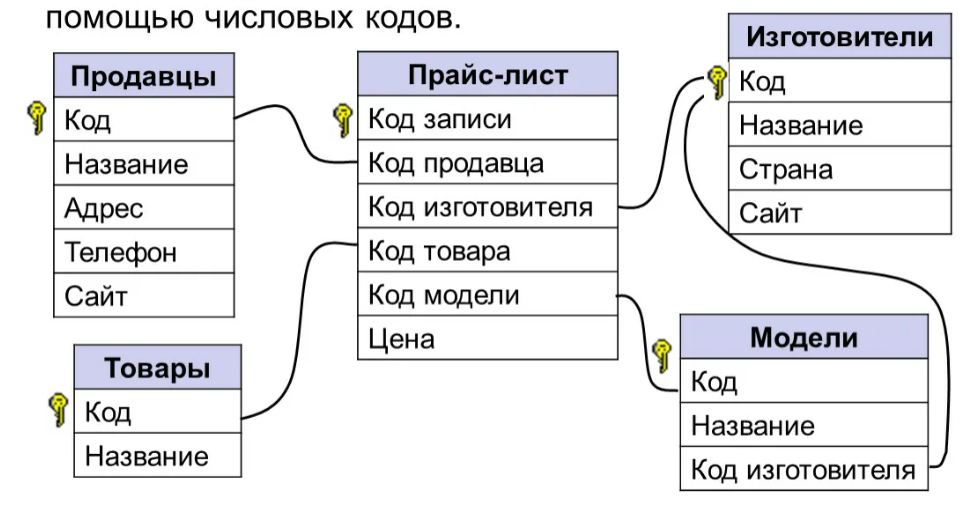
* Слишком сложно пользоваться
* Необходимо знание физической организации
* От физической организации зависят прикладные системы
* Логика программы перегружена деталями доступа к БД

1. Реляционный подход к организации БД. Базовые понятия реляционных баз данных.

Relation - отношение

Реляционный подход к организации базы данных – это метод структурирования и хранения данных в виде таблиц (отношений), где данные представлены в виде строк и столбцов. В этом подходе данные хранятся в нормализованных таблицах, которые могут быть связаны между собой с помощью ключей (первичных и внешних). Этот подход обеспечивает удобство, целостность данных и возможность эффективно манипулировать данными через специальные запросы.

Этот тип БД наиболее распространен сегодня.



Плюсы:

* Наличие небольшого набора абстракций, на основе которых просто моделировать наиболее распространенные предметные области
* Наличие простого, но мощного математического аппарата из теории множеств и математической логики
* Есть возможность манипулирования данными без необходимости знания конкретной физической организации во внешней памяти

Минусы:

* Критикуют за ограниченность реляционной СУБД в нетрадиционных областях: в системах автоматизации проектирования, машиностроении и т. д., где используются сложные структуры данных

Тип данных - наряду с традиционными (текстовый, числовой, дата и т. д.) есть и специальные.

Домен - сравнимо с подтипами в языках программирования. Это допустимое потенциальное множество значений данного типа.

Атрибут – одна из характеристик объекта (столбец, колонка, поле).

Схема отношения – именованное множество пар (имя атрибута + имя типа данных). Степень (арность) схемы отношения - мощность этого множества.

Схема БД – набор именованных схем отношений.

Кортеж – это множество пар (имя атрибута + значение), которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения.

Отношение – это множество кортежей, соответствующих одной схеме отношения. Различают отношение-схему и отношение-экземпляр. Набор кортежей в отношении - тело отношения.

Реляционная БД – это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

|  |  |  |
| --- | --- | --- |
| Электронные таблицы | Реляционные модели | Реляционные БД |
| Таблица | Отношение | Таблица БД |
| Столбец (столбик, колонка) | Атрибут | Поле |
| Строка | Кортеж | Запись |
| «Шапка» | Схема отношения | Структура таблицы |

1. Фундаментальные свойства отношений. Реляционная модель данных.

Фундаментальные свойства отношений описывают ключевые характеристики, которые должны быть выполнены для того, чтобы данные, представленные в таблицах реляционной базы данных, соответствовали требованиям этой модели. Эти свойства позволяют гарантировать целостность и правильность работы с данными в реляционных базах данных:

1. Значения атрибутов должны быть атомарны. Т. к. домен – это потенциальное множество значений простого типа данных, то среди значений домена не могут содержаться множества значений.
2. Отсутствие кортежей-дубликатов (в таблице не должно быть повторных записей). Свойство следует из определения отношения как множества кортежей. А в классической теории множеств - "каждое множество состоит из различных элементов". Это свойство требует наличия у каждого отношения первичного ключа - атрибута или набора атрибутов, значения которых однозначно определяют кортеж отношения. Могут быть альтернативные ключи.
3. Отсутствие упорядоченных кортежей. Это следует из определения: "отношения – это множество кортежей".
4. Отсутствие упорядоченных атрибутов

Реляционная модель данных – это метод организации и хранения данных в базе данных с использованием таблиц (или отношений), которые соединены между собой с помощью ключей и позволяют эффективно манипулировать данными.

Реляционная модель состоит из 3 частей:

1. Структурная часть - в этой части говорится, что единственной структурой данных, используемой в реляционной БД, является нормализованное n-арное отношение
2. Манипуляционная часть - в этой части утверждаются 2 фундаментальных механизма реляционной БД:
   * Реляционная алгебра - в основе классическая теория множеств.
   * Реляционное исчисление - в основе классический логически аппарат исчисления предикатов первого порядка.

Основная функция манипуляционной части - обеспечение меры реляционности любого конкретного языка реляционной БД.

Язык называется реляционным, если он обладает не меньшей выразительностью и мощностью, чем реляционная алгебра и реляционное исчисление.

1. Целостная часть - в этой части реляционной модели фиксируются 2 базовых требования целостности, которые поддерживаются в любой реляционной СУБД:
   * Целостность сущностей: любой кортеж любого отношения отличим от любого другого кортежа этого отношения. Решается этот вопрос введением первичного ключа.
   * Целостность по ссылкам: при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений.

|  |  |  |
| --- | --- | --- |
| Электронные таблицы | Реляционные модели | Реляционные БД |
| Таблица | Отношение | Таблица БД |
| Столбец (столбик, колонка) | Атрибут | Поле |
| Строка | Кортеж | Запись |
| «Шапка» | Схема отношения | Структура таблицы |

1. Базисные средства манипулирования реляционными данными. Реляционная алгебра. Реляционное исчисление.

Базисные средства манипулирования реляционными данными – это основные инструменты и операции, которые используются для работы с реляционными базами данных (РБД), обеспечивающими хранение и обработку данных в виде таблиц. Эти операции позволяют извлекать, изменять, добавлять и удалять данные из реляционных таблиц.

Основные базисные средства:

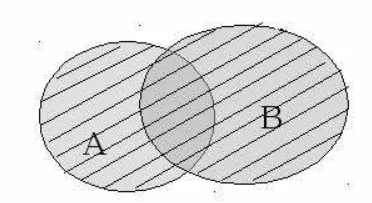
* SQL;
* Реляционные операции;
* Нормализация данных;
* Операции с транзакциями.

Реляционная алгебра – это формальная система, которая используется для описания операций над реляционными данными в рамках реляционной модели данных. Она состоит из набора операций, которые принимают одну или несколько отношений (таблиц) в качестве входных данных и возвращают новое отношение в качестве результата. Реляционная алгебра служит теоретической основой для работы с реляционными базами данных, а операции реляционной алгебры лежат в основе SQL-запросов.

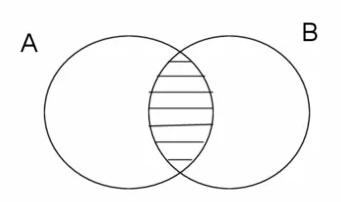
Основные алгебраические операции:

1. Теоретико-множественные:

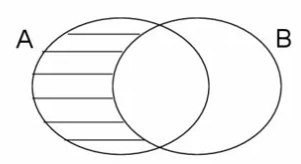
* Объединение отношений (∪) – возвращает кортежи, которые есть хотя бы в одном из отношений



* Пересечение отношений (∩) – возвращает кортежи, которые есть в обоих отношениях



* Разность отношений (-) – возвращает кортежи, которые есть в 1 отношении, но нет во 2-ом



* Произведение отношений (\*) – создает все возможные комбинации кортежей из 2 отношений

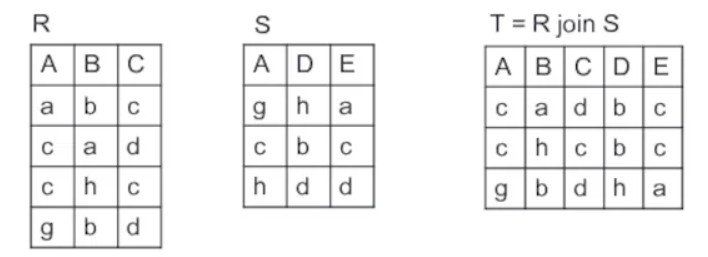


1. Специальные реляционные операции:

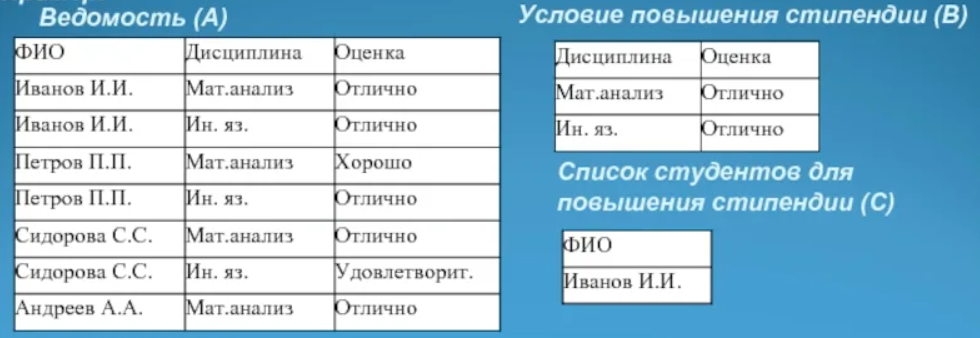
* Ограничение отношения - это процесс отбора только тех кортежей из отношения, которые соответствуют заданным условиям. Результатом операции является новое отношение, содержащее только отобранные строки.
* Проекция отношения – отбирает только нужные атрибуты из отношения



* Соединение отношений – соединяет 2 отношения по общим атрибутам



* Деление отношений – используется, чтобы найти кортежи из одного отношения, связанные с кортежами другого отношения



1. Также в состав алгебры включены:

* Операция присваивания – позволяет сохранить результат реляционной операции в новой (или существующей) переменной отношения
* Операция переименования атрибутов – позволяет изменить имя атрибутов

Реляционное исчисление – это формальная система, которая используется для описания запросов к реляционным данным с использованием логики и математических выражений. В отличие от реляционной алгебры, где операции выполняются над отношениями (таблицами), реляционное исчисление фокусируется на описании условий для выборки данных, а не на процедурах манипулирования самими данными.

Реляционное исчисление и реляционная алгебра описывают одни и те же операции, но с разной точки зрения. Реляционная алгебра работает с операциями на отношениях (таблицах), а реляционное исчисление – с логическими выражениями, которые описывают, какие кортежи или атрибуты должны быть выбраны на основе условий.

В теории, любое выражение реляционной алгебры может быть переведено в реляционное исчисление, и наоборот. Однако реляционное исчисление часто используется для описания запросов в более декларативной форме, где не нужно указывать, какие именно операции должны быть выполнены, а только условия для выборки данных.

1. Проектирование реляционных БД. Нормализация.

Проектирование реляционных баз данных – процесс создания структуры базы данных, которая соответствует требованиям приложения или организации, а также эффективно управляет данными. Процесс проектирования можно разделить на несколько этапов:

1. Анализ требований
2. Концептуальное проектирование (ER-моделирование)
3. Логическое проектирование (Перевод ER-модели в реляционную)
4. Нормализация
5. Физическое проектирование

При проектировании БД решаются 2 проблемы:

1. Как отобразить объекты модели данных - логическое проектирование
2. Как обеспечить эффективность выполнения запросов к БД, т. е. как имея ввиду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур потребовать - физическое проектирование

Таким образом, суть проектирования в том, что выбираются отношения, из каких будет состоять БД, и определяется, какие атрибуты должны быть у этих отношений

Проектирование реляционных БД производится методом последовательных приближений к удовлетворительному набору схем отношений. Исходная точка является представлением предметной области в виде одного или нескольких отношений.

Процесс проектирования – это процесс нормализации схем отношений, причем каждая следующая нормальная форма (НФ/NF) обладает свойствами лучшими, чем предыдущая.

Каждой НФ соответствует некоторый определенный набор ограничений.

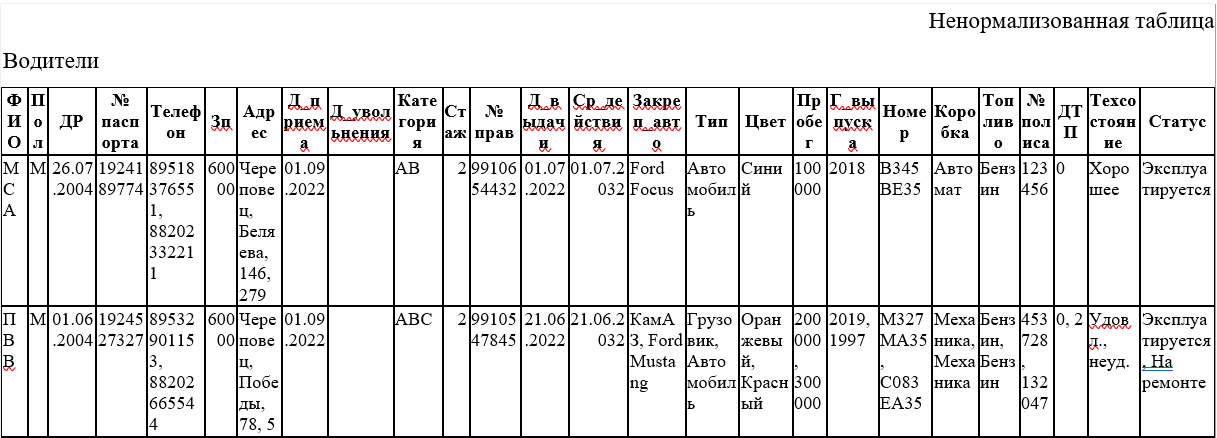
Существуют такие нормальные формы:

* Первая НФ (1НФ/1NF)
* Вторая НФ (2НФ/2NF)
* Третья НФ (3НФ/3NF)
* НФ Бойса-Кодда (BCNF)
* Четвертая НФ (4НФ/4NF)
* Пятая НФ (5НФ/5NF)

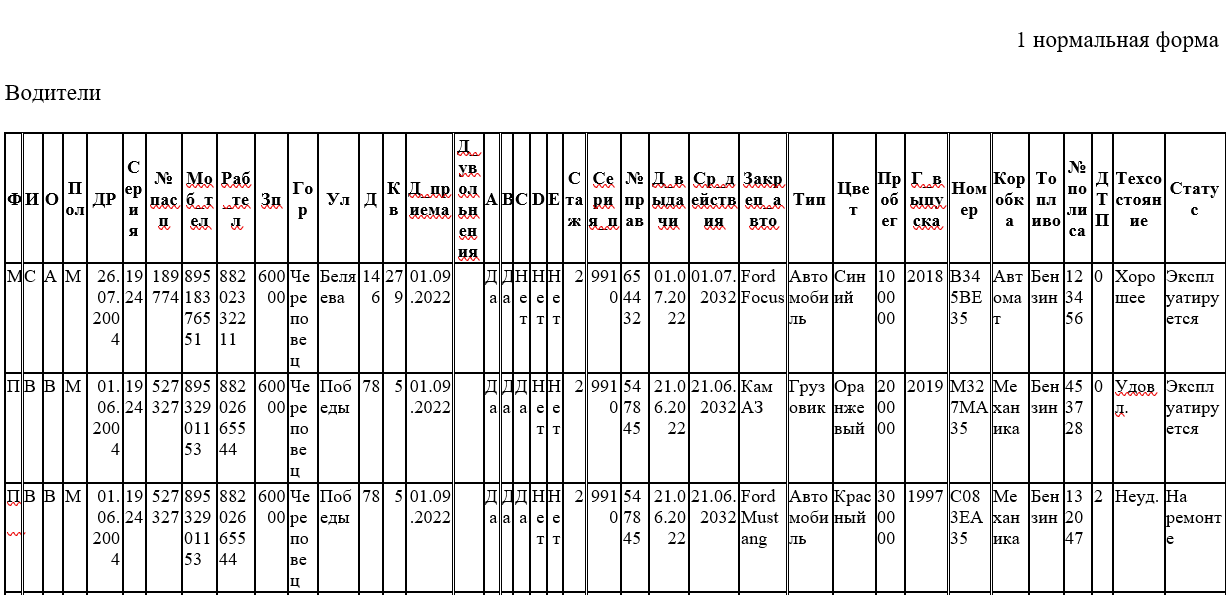
Основные свойства НФ:

* Каждая следующая НФ в некотором смысле лучше предыдущей
* При переходе к следующей НФ свойства предыдущих НФ сохраняются

Нормализация – это декомпозиция отношения, находящегося в предыдущей НФ в два или более отношений, удовлетворяющих требованиям следующей НФ.

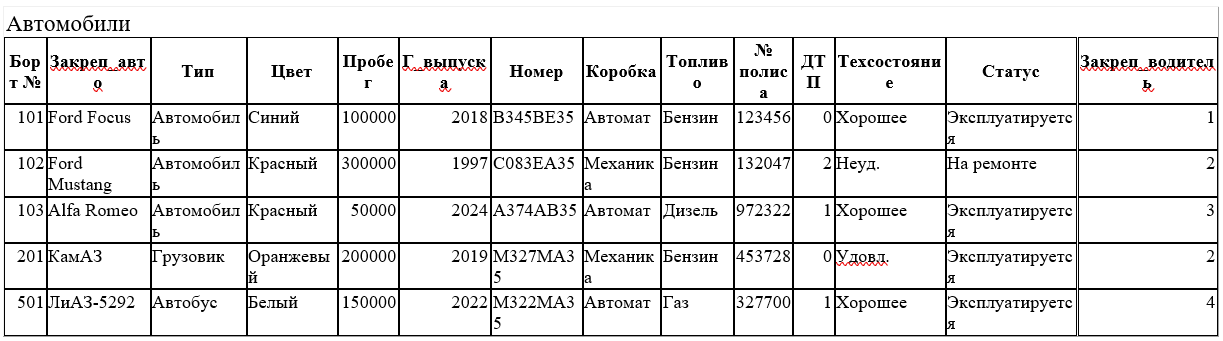


* 1НФ – значения атрибутов отношения должны быть атомарны (неделимы).

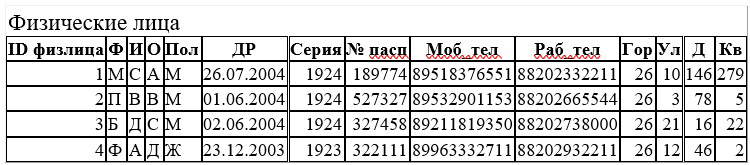


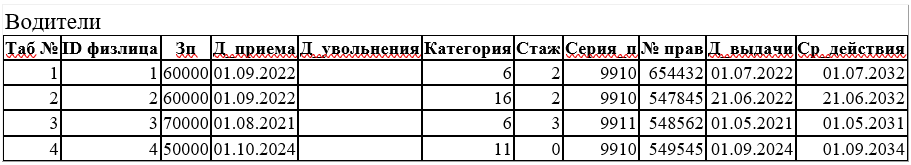
* 2НФ – отношение находится во 2НФ в том и только в том случае, когда отношение находится в 1НФ и каждый неключевой атрибут полностью зависит от первичного ключа.

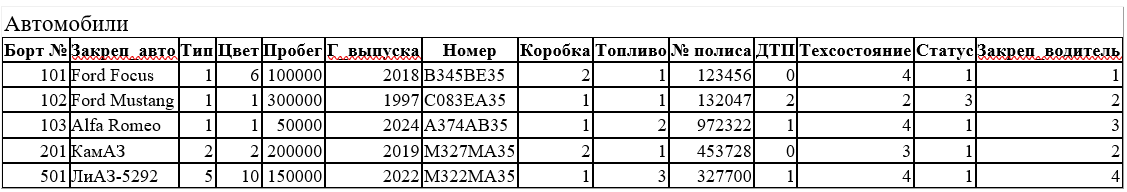


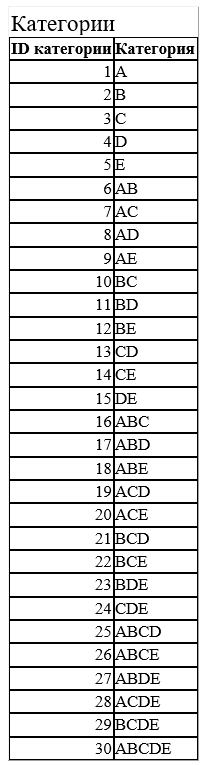
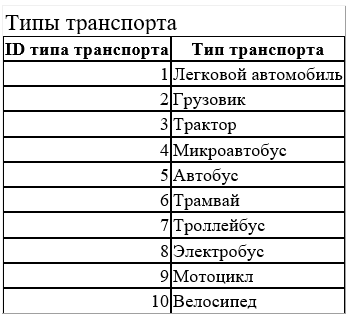
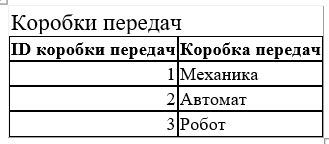
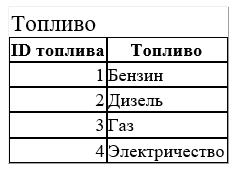
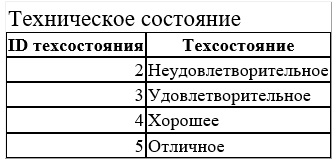
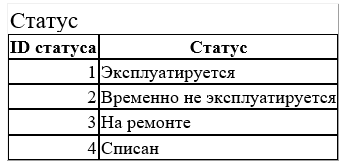


* 3НФ – отношение находится в 3НФ в том и только в том случае, когда отношение находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.







Нормализацию часто завершают на 3НФ.

При использовании ненормализованных таблиц возможны проблемы: избыточность данных, аномалии обновления, удаления, ввода.

1. Основные понятия модели «Сущность-Связь». ER-диаграммы.

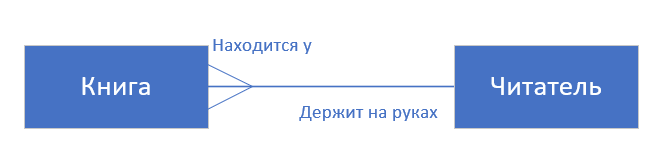
Реляционных БД достаточно для моделирования большинства предметных областей. Но при реальном проектировании обнаруживается их ограниченность, у модели нет достаточных средств для отражения смысла данных. Трудно моделировать предметную область на основе плоской таблицы.

Проектирование – это выделение сущностей и связей между ними. Модель не предполагает аппарата для разделения сущностей и связей. В развитии реляционной модели появилась семантическая модель данных Entity Relationship (ER). Эта модель рисуется в виде ER-диаграмм. Моделирование базируется на графических диаграммах. Сущность: связь-атрибут.

Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. Диаграмма - прямоугольник, внутри которого имя сущности (при этом имя сущности – имя типа, а не некоторого конкретного экземпляра).



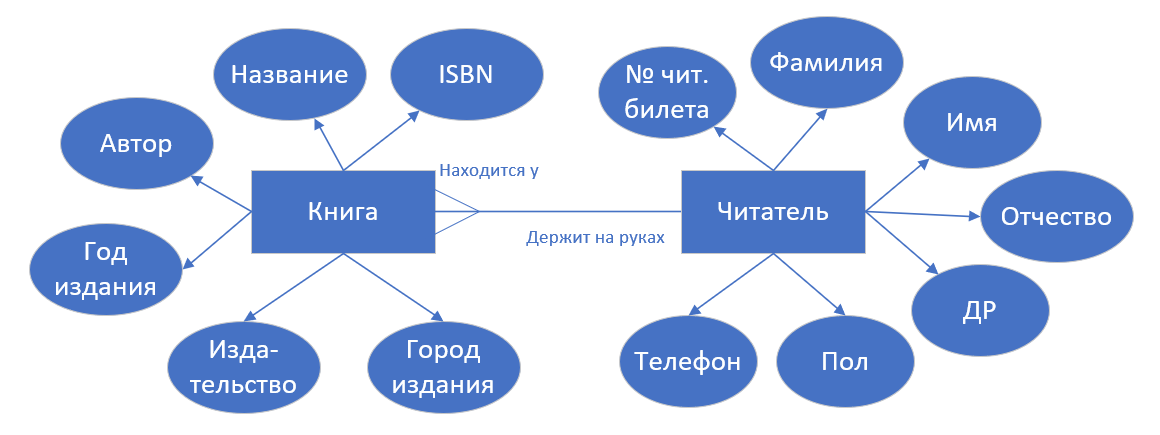
Связь - графически изображаемая ассоциация, устанавливаемая между 2 разными сущностями (она бинарна). На концах указывается имя конца связи (сколько экземпляров данной сущности связывается), обязанность связи (т. е. любой экземпляр сущности должен участвовать в связи).



Каждый билет предназначен только для 1 пассажира. Каждый пассажир может иметь 1 или более билетов.

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

Уникальный идентификатор (id) - такой атрибут или их комбинация, однозначно отличающая любой экземпляр сущности от других экземпляров этой сущности.



1. Система System R. Основные цели System R.

System R – это одна из первых экспериментальных реляционных систем управления базами данных (СУБД), разработанная в 1970-х годах в IBM для демонстрации концепции реляционной модели данных, предложенной Эдгаром Ф. Коддом. Этот проект сыграл ключевую роль в развитии реляционных СУБД и непосредственно повлиял на создание коммерческих реляционных систем, таких как Oracle и IBM DB2.

Одной из значительных особенностей System R было внедрение языка SQL. SQL стал основным языком для взаимодействия с реляционными СУБД, и с тех пор его спецификация была стандартизирована и используется в большинстве современных реляционных СУБД.

Основные цели System R:

1. Демонстрация реляционной модели данных: Целью создания System R было продемонстрировать практическую применимость реляционной модели данных, предложенной Эдгаром Коддом, в реальных условиях. Система показывала, как можно эффективно организовывать и управлять данными с использованием таблиц, отношений и ключей.
2. Разработка и тестирование языка SQL: Одной из ключевых целей System R было создание языка запросов для реляционных баз данных, который в будущем станет основным стандартом — SQL. В System R был разработан первый вариант SQL, который позволял работать с данными, создавать таблицы, извлекать, обновлять и удалять данные.
3. Оптимизация выполнения запросов: в System R были реализованы механизмы для оптимизации запросов, что включало алгоритмы для улучшения производительности обработки SQL-запросов. Это было важно для повышения скорости работы с большими объемами данных.
4. Управление транзакциями и целостностью данных: ещё одной целью было создание механизмов для обеспечения целостности данных и управления транзакциями. Это включало такие концепции, как контроль за целостностью данных и поддержка ACID-свойств (атомарность, согласованность, изолированность и долговечность), что является основой для надежных и безопасных реляционных СУБД.
5. Разработка реляционного менеджера данных: целью было создать эффективный менеджер данных, который бы обеспечивал хранение данных в реляционном формате, а также выполнение операций над этими данными (например, выборки, вставки, обновления и удаления).
6. Обоснование реляционной модели: Одной из целей System R было научное подтверждение эффективности реляционной модели данных в реальных приложениях и исследованиях, а также её возможностей по сравнению с другими моделями баз данных, например, с сетевой или иерархической моделью данных. В System R была реализована реляционная модель данных, которая включает использование таблиц для представления данных и использования SQL для взаимодействия с базой данных.

Основные компоненты System R:

* Менеджер данных: отвечал за управление реляционными данными, хранение данных и выполнение запросов.
* Язык запросов: это был один из первых вариантов SQL, который использовался для выполнения операций с данными, таких как выборка, вставка, обновление и удаление данных.
* Оптимизатор запросов: включал механизмы для оптимизации выполнения SQL-запросов, что также стало важной частью современных СУБД.

Разработка System R привела к значительным изменениям в области баз данных. Хотя сама система никогда не стала коммерческим продуктом, её концепции легли в основу всех современных реляционных СУБД, включая IBM DB2, Oracle Database, MySQL и другие.

1. Структуры внешней памяти, методы организации индексов. Хранение отношений.

Внешняя память – это устройства, которые используются для долговременного хранения данных, например, жесткие диски, SSD и другие типы хранения данных.

Основные структуры внешней памяти:

1. Блоки данных — это минимальные единицы ввода-вывода, которые СУБД передает между памятью и диском.
2. Страница (Page): внешняя память организована в виде страниц, которые представляют собой блоки данных фиксированного размера (обычно от 4 до 16 Кб). Все данные базы данных разбиваются на страницы, и каждая страница загружается или выгружается в оперативную память целиком.
3. Файлы данных (Data Files): Внешняя память организована в виде файлов данных, которые хранят информацию о таблицах, индексах и других объектах базы данных. Эти файлы делятся на блоки или страницы данных.

Для ускорения поиска и других операций в больших объемах данных применяются структуры индексов. Индексы аналогичны телефонным книгам, где поиск по ключу (например, имени) дает быстрый доступ к записям.

Методы организации индексов:

1. B-дерево – это сбалансированное дерево поиска, где каждый узел может содержать несколько ключей. Оно позволяет выполнять операции поиска, вставки и удаления за время O(log n)
2. Хеш-индексы – в хеш-индексе данные хешируются с помощью функции хеширования, которая сопоставляет ключи с уникальными значениями, позволяющими быстро находить нужные записи.
3. Битовые карты – это индекс, который использует битовую карту для представления наличия или отсутствия определенных значений. Каждый бит в карте соответствует определенному значению в таблице.
4. Деревья отрезков – часто используются для индексации данных, которые могут быть представлены в виде диапазонов (например, для поиска числовых интервалов).
5. Графовые индексы – применяются в социальных сетях, системах рекомендаций, анализе маршрутов и других задачах, где данные могут быть представлены в виде графов.

Хранение отношений:

* Покортежное хранение – это традиционный способ хранения, при котором каждый кортеж (строка) отношения хранится как единица данных в памяти или на диске. Основным преимуществом является то, что весь кортеж можно получить за один доступ (один диск-обмен), что ускоряет операции, требующие работы с полными записями (например, выборка всей строки). Однако здесь возникает дублирование данных для общих значений в разных кортежах, что увеличивает использование памяти, особенно если атрибуты таблицы имеют высокую степень повторяемости.
* Постолбцовое хранение – в этом подходе каждый столбец хранится отдельно, при этом дублирующиеся значения в столбцах не повторяются (например, значений типа «женщина», «мужчина» для столбца «пол» могут быть сохранены как одно уникальное значение). Это позволяет существенно уменьшить объем памяти, поскольку данные хранятся в более компактной форме, без избыточности. Также такой подход может ускорить операции, использующие только часть данных (например, только один столбец для выборки), поскольку можно загружать только нужные данные. Однако для сборки полного кортежа требуется выполнить дополнительные операции для восстановления данных из разных столбцов, что может замедлить выполнение запросов, особенно если требуются операции с несколькими столбцами одновременно.

1. Транзакции и целостность баз данных. Сериализация транзакций.

Транзакция (ТА) – это такая неделимая с точки зрения воздействия на БД последовательность операторов работы с данными, что либо результаты всех операторов, входящих в ТА, отображаются в БД, либо их воздействие отсутствует. Лозунг ТА - все или ничего.

При завершении операций оператором «commit» результаты фиксируются во внешней памяти. Если оператором «rollback» - все отменяется.

В Access завершение транзакции оператором COMMIT происходит в следующих случаях:

* после выполнения операций через SQL-запросы;
* при переходе к другой записи;
* при закрытии таблицы, запроса или формы;
* при удалении записи через интерфейс программы.

В Access завершение транзакции оператором ROLLBACK происходит в следующих случаях:

* при отмене изменений через комбинации Ctrl+Z или кнопки «Отменить» в меню;
* при ошибке в процессе выполнения транзакции.

ТА обеспечивают целостность БД и изолированность пользователей.

Целостность базы данных (БД) – это принцип, обеспечивающий корректность, непротиворечивость и точность данных в базе.

Иногда целостность нельзя не нарушить, поэтому состояние нецелостности допускается внутри транзакций.

Различают 2 вида ограничений целостности (ОЦ):

* Немедленно проверяемое ОЦ – это ограничение, проверку которого бессмысленно или невозможно откладывать. Реализуется оператором высокого уровня. При этом не вся ТА отменяется, а лишь оператор.
* Откладываемое ОЦ – это ограничение на БД, а не на операции. Проверяется в конце ТА.

Чтобы добиться изолированности ТА, в СУБД есть методы регулирования совместного их выполнения.

Сериализация – это механизм выполнения ТА по некоторому сериальному плану. Результат совместного выполнения ТА эквивалентен результату некоторого последовательного выполнения этих же ТА. Методы сериализации основываются на учете 3 конфликтов: WW, RW, WR.

Методы сериализации:

* Метод синхронизационных захватов (СЗ) объектов БД
* Метод временных меток

Перед выполнением операций над объектом БД, от имени ТА запрашивается СЗ объекта в каком-то из режимов:

* Совместный режим (S) – разделяемый захват объекта, требуется для чтения
* Монопольный режим (X) – для операций занесения данных, удаления, модификации

Захваты объектов по R совместимы. Захват по R несовместим с захватом по W. Захваты по W несовместимы.

Что считать объектом для захвата:

* Файл - физический объект, где хранятся несколько отношений
* Отношение - логический объект с множеством кортежей
* Кортеж - элементарный физический объект БД
* Страница данных - физический объект, который хранит кортежи одного или нескольких отношений и индексную, служебную информацию

Операция над кортежем - это операция над страницей, где он находится, и над отношением, и над файлом. Чем крупнее объект СЗ, тем меньше этих СЗ будет поддерживаться, и на них будет меньше расходов. Но возрастает вероятность конфликтов между ТА. Сегодня реализованы покортежные захваты.

Выполнение СЗ приводит к тупикам между ТА:

1. Т1 и Т2 устанавливают монопольные захваты объектов A1 и A2 соответственно.
2. Т1 требует совместный захват A2, а Т2 - совместный захват A1
3. Ни одна из ТА не продолжится. Монопольные захваты не будут сняты, а совместные не будут удовлетворены.

Обнаруживают тупики через граф ожидания ТА. По этому графу обнаруживается цикл, где и выбирается ТА-жертва и ее откатывают.

1. Изолированность пользователей.

ТА обеспечивают изолированность пользователей, т. е. создается иллюзия того, что каждый работает с БД в одиночку. ТА выступает единицей изолированности пользователя.

Изолированность пользователей – это принцип, который гарантирует, что операции, выполняемые одним пользователем (или транзакцией), не будут влиять на операции других пользователей, которые работают с той же базой данных одновременно. Это особенно важно в многозадачных системах, где одновременно может происходить несколько операций с базой данных.

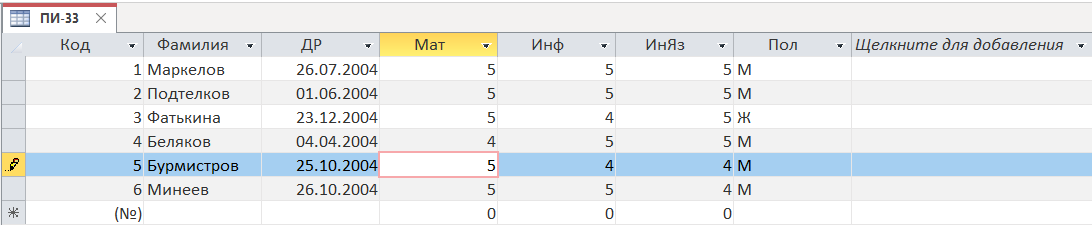
Существуют 4 уровня изолированности:

1. Отсутствие потерянных изменений

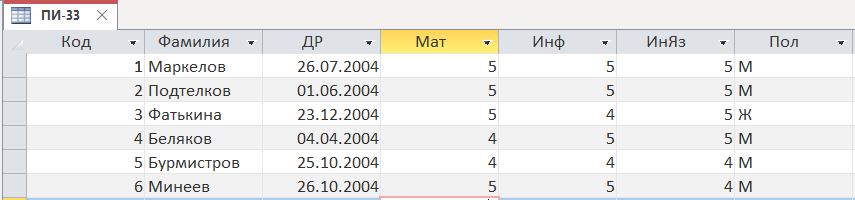
Т1 изменяет объект A базы данных. До завершения Т1, Т2 также пытается изменить объект. Т2 завершается оператором rollback. Тогда при повторном чтении объекта A, Т1 не видит изменений объекта, произведенных ранее. Это противоречит требованию изолированности пользователей.

Решение: до завершения Т1 никакая ТА не может изменять объект.

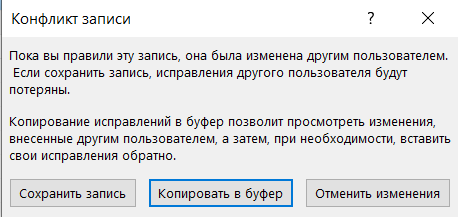
Отсутствие потерянных изменений – это минимальное требование СУБД к параллельно выполняемым ТА.



Т1: изменение данных, не завершена



Т2: изменение данных, отменена



Ошибка при попытке завершить Т1

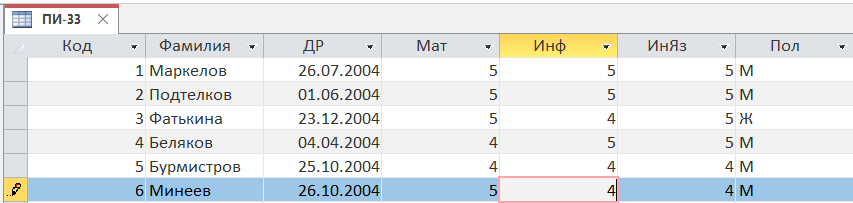
Если бы не было защиты от отсутствия потерянных изменений, данные пользователя №1 были бы утеряны из-за правки от пользователя №2.

1. Отсутствие чтения "грязных" данных (проблема промежуточных данных)

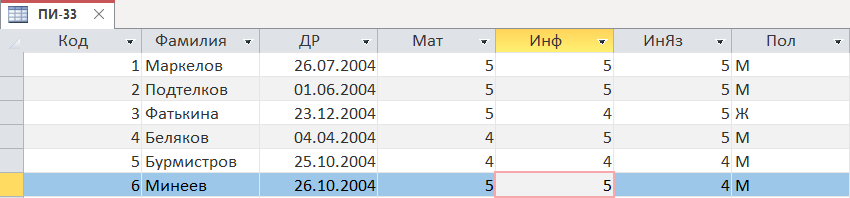
Т1 изменяет объект A. Параллельно с этим Т2 читает. Т. к. операция изменения не завершена, Т2 видит несогласованные "грязные" данные.

Несоответствие требованию изолированности: каждый пользователь начинает свою ТА при согласованном состоянии БД.

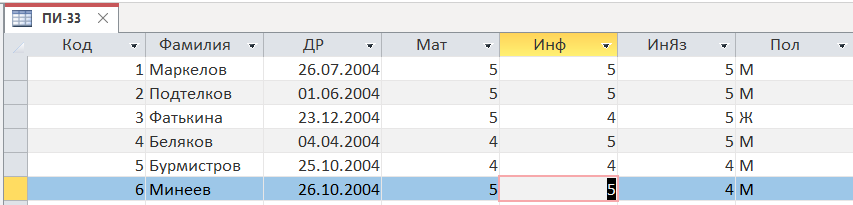
Решение: до завершения ТА, изменившей объект, никакая ТА не должна читать этот объект.



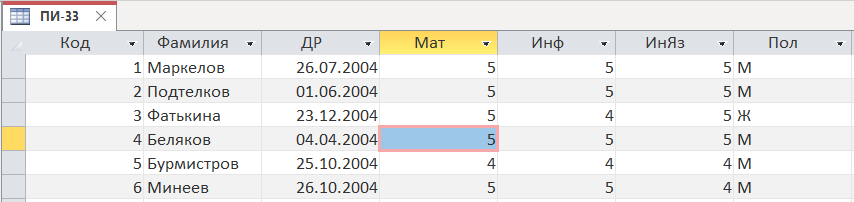
Т1: изменение данных, не завершена



Т1: чтение данных, изменений нет



Т1: изменение данных, отменена



Т2: чтение данных, изменений нет

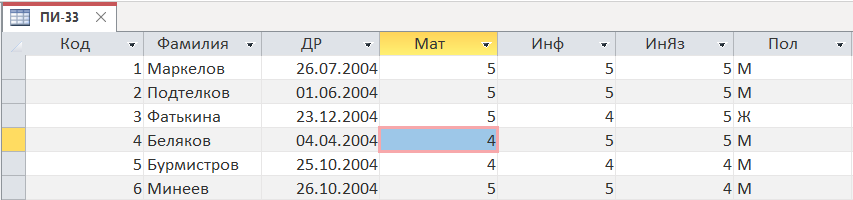
Если бы не было защиты от отсутствия чтения «грязных данных», пользователь №2 увидел бы ошибочные данные, которые в итоге не были внесены пользователем №1 в БД.

1. Отсутствие неповторяющихся чтений

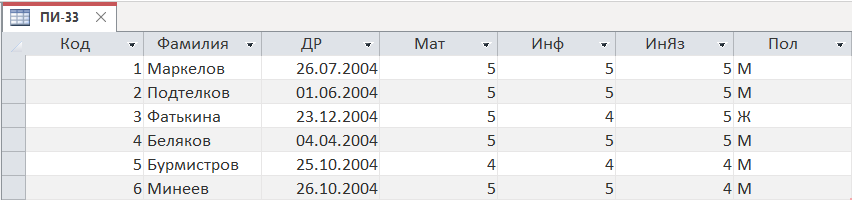
Т1 читает объект A. До завершения Т1, Т2 изменяет объект и успешно завершается commit'ом. Т1 повторно читает объект и видит его измененное состояние.

Неповторяющееся чтение.

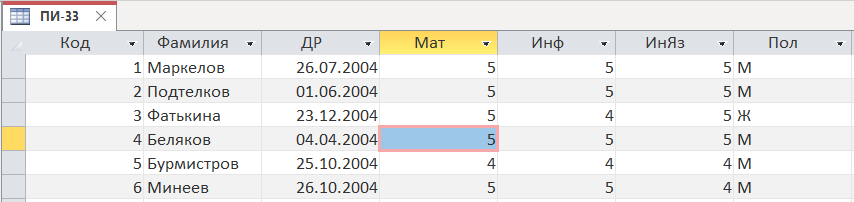
Решение: до завершения Т1 никакая другая ТА не должна изменять объект.



Т1: чтение данных



Т1: изменение данных, завершена



Т2: чтение данных, изменения появились

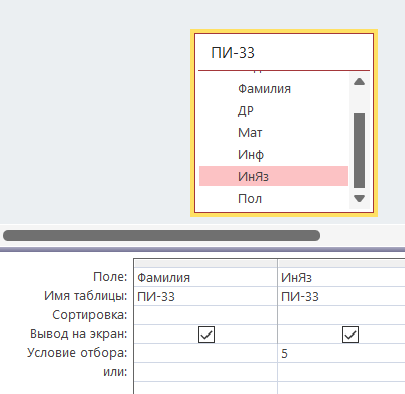
Если бы не было защиты от неповторяющихся чтений, пользователь №1 видел бы неактуальную информацию без внесенных пользователем №2 изменений в БД.

1. Проблема кортежей-фантомов

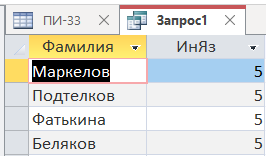
Т1 выполняет оператор выборки кортежа отношения R с условием выборки S. До завершения Т1, Т2 вставляет в R новый кортеж, удовлетворяющий S и успешно завершается. Т1 повторно выполняет этот оператор и в результате видит новый кортеж.

Противоречие идее изолированности.

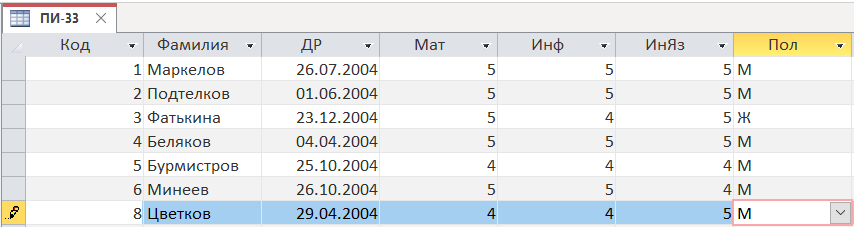
Решение: требуется более высокий уровень синхронизации ТА



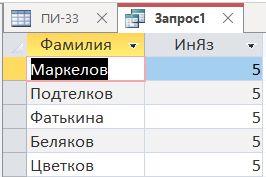
Т1: создание запроса



Т1: результат запроса



Т2: добавление нового кортежа



Т1: результат повторного запроса

Такой результат говорит о том, что в Access нет защиты от кортежей-«фантомов».

1. Журнализация изменений БД. Журнализация и буферизация.

Одним из основных требований к развитым СУБД является надежность хранения баз данных. Для восстановления согласованного состояния БД после аппаратного или программного сбоя поддерживается журнал изменений БД (ЖИ).

Журнализация – это процесс записи всех изменений, происходящих в базе данных, с целью обеспечения долговечности и восстановления данных. Это важная часть механизма управления транзакциями, позволяющая восстанавливать базу данных после сбоев, потерь данных или других проблем.

Журнализация изменений помогает отслеживать, какие операции были выполнены, какие данные были изменены, и позволяет восстанавливать состояние базы данных на момент до сбоя.

Журнал транзакций – это файл или структура данных, в которой сохраняются все операции, выполняемые в базе данных, например, вставка, обновление или удаление записей.

Запись в журнал транзакций обычно включает в себя: идентификатор транзакции, тип операции, затронутые строки и старые и новые значения данных.

Общие принципы восстановления:

* результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
* результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Ситуации, при которых требуется восстановление:

* Индивидуальный откат ТА - завершение оператором rollback, откат ТА по инициативе системы, исключительные ситуации в программе, выбор ТА в качестве жертвы
* Потеря содержимого ОЗУ при "мягком" сбое - при аварийном выключении питания, при сбое процессора
* Поломка ВЗУ

В основе восстановления хранение избыточных данных в ЖИ БД и архивной копии.

Возможны два основных варианта ведения журнальной информации.

* для каждой транзакции поддерживается отдельный локальный журнал изменений БД. (индивидуальных откатов транзакций)
* кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после мягких и жестких сбоев.

Буферизация – это процесс хранения данных в памяти (обычно в оперативной памяти) для временной обработки перед их записью на постоянный носитель (например, диск). Буферизация используется для оптимизации операций ввода/вывода (I/O) и повышения производительности при работе с базой данных.

Основная цель буферизации – уменьшить количество операций записи и чтения с диска, что позволяет значительно ускорить обработку запросов. Вместо того чтобы каждый раз обращаться к диску для чтения или записи данных, система может сначала выполнять операции в памяти, а затем, когда накопится достаточно данных или когда потребуется запись, она делает одну операцию на диске.

1. Восстановление баз данных в различных ситуациях.

Восстановление базы данных – это процесс приведения базы данных к ее последнему согласованному состоянию после сбоя системы, ошибки пользователя или других нештатных ситуаций. Восстановление важно для обеспечения долговечности данных и гарантии их целостности, особенно в многопользовательских и высоконагруженных системах.

Восстановление баз данных в различных ситуациях:

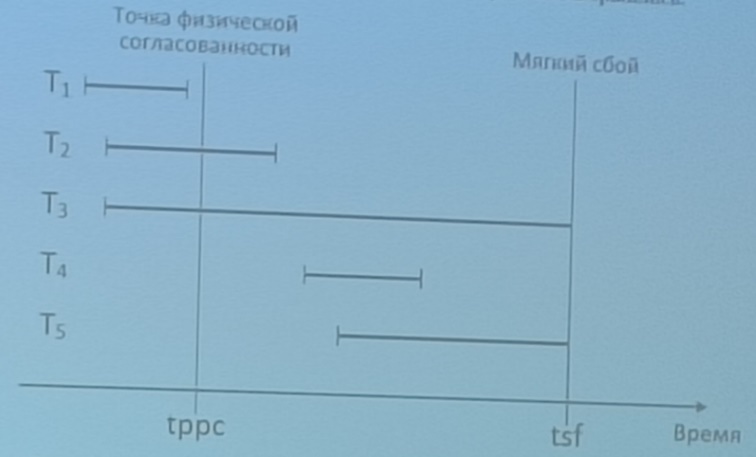
1. Индивидуальный откат транзакции

* Выбирается очередная запись из списка данной транзакции.
* Выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE
* Любая из этих обратных операций также журнализуются.
* При успешном завершении отката в журнал заносится запись о конце транзакции.

1. Восстановление после «мягкого» сбоя

«Мягкий» сбой – это потеря содержимого оперативной памяти.

После мягкого сбоя набор страниц внешней памяти базы данных может оказаться несогласованным, т.е. часть страниц внешней памяти соответствует объекту до изменения, часть - после изменения. Состояние внешней памяти базы данных называется физически согласованным, если наборы страниц всех объектов соответствуют состоянию объекта либо после его изменения, либо до изменения.



В ЖИ отмечаются точки физической согласованности (ТФС) – моменты времени, когда в ВЗУ содержатся согласованные результаты операций, завершившихся до момента "мягкого" сбоя и отсутствуют результаты операций, которые не завершились.

Что сделать для восстановления БД:

* T1 - ничего не надо делать, т. к. ТА закончилась до ТФС, все результаты отражены в ВЗУ
* T2 - необходимо повторно выполнить оставшуюся часть ТА (redo)
* T3 - выполняют в обратном направлении 1 часть операции (undo). В ВЗУ нет результатов T3, которые выполнены после ТФС, но есть результаты до ТФС
* T4 - она успела начаться после ТФС и закончится до "мягкого сбоя". Поэтому нужно выполнить полную повторную прямую интерпретацию команд этой ТА (redo)
* T5 - ничего не поможет, т. к. результатов нет в ВЗУ

1. Восстановление после «жесткого сбоя»

«Жесткий» сбой – поломка основного внешнего носителя БД.

Для восстановления последнего согласованного состояния базы данных после жесткого сбоя журнала изменений базы данных явно недостаточно.

Основой восстановления в этом случае являются журнал и архивная копия базы данных. Восстановление начинается с обратного копирования базы данных из архивной копии.

Этапы восстановления:

* по журналу в прямом направлении выполняются все операции;
* для транзакций, которые не закончились к моменту сбоя, выполняется откат.

1. Язык SQL, функции и основные возможности. Стандартизация SQL.

SQL (Structured Query Language – язык структурированных запросов) – это язык для управления и обработки данных в реляционных базах данных. Он используется для выполнения операций с базой данных, таких как создание, обновление, удаление и извлечение данных.

SQL = DDL + DML + DQL, где

* DDL (Data Definition Language) – язык описания данных
* DML (Data Manipulation Language) – язык манипуляции данными
* DQL (Data Query Language) – язык запросов данных

DDL включает команды, которые используются для создания, изменения и удаления структур базы данных, таких как таблицы, индексы, схемы и другие объекты. Это операции, которые определяют структуру базы данных (CREATE, ALTER, DROP).

DML включает команды для работы с данными внутри таблиц базы данных. Эти команды позволяют добавлять, обновлять, удалять и извлекать данные. Основные операции DML касаются именно данных, а не структуры базы данных (INSERT, UPDATE, DELETE).

DQL включает команду SELECT, которая используется для извлечения данных из базы данных. В отличие от DML, который также включает операции вставки, обновления и удаления, DQL полностью сосредоточен на извлечении данных.

SQL появился в середине 70-х гг. в рамках System-R. Уже тогда он был полным языком БД. Язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов.

Стандартизация SQL – это процесс разработки и принятия международных стандартов для языка SQL. Основная цель стандартизации SQL – обеспечить совместимость между различными системами управления базами данных (СУБД), а также создать универсальный и единообразный подход к запросам, манипуляциям с данными, определению структуры базы данных и другим операциям, связанным с базами данных.

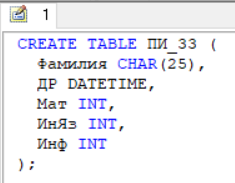
Стандарты SQL разрабатываются международными организациями, такими как ISO (International Organization for Standardization) и ANSI (American National Standards Institute). Стандарты описывают как синтаксис SQL, так и семантику команд и операций.

Стандарты SQL:

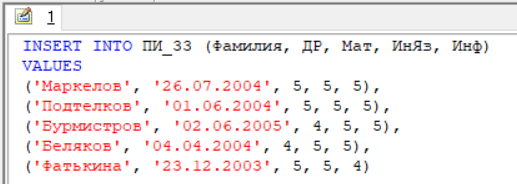
* SQL-1 (SQL-89) – в этом стандарте полностью отсутствуют такие важные разделы, как манипулирование схемой БД и динамический SQL. Многие важные аспекты языка в соответствии со стандартом определяются в реализации
* SQL-2 (SQL-92) – манипулирование схемой БД, управление транзакциями и сессиями, подключение к БД, динамический SQL
* SQL-3 (SQL-99) – механизм триггеров и возможность использования абстрактных типов данных

Пример использования в ЛР:

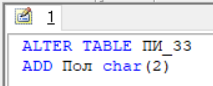
Создание таблицы:



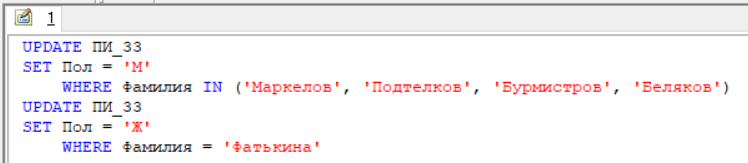
Внесение данных:



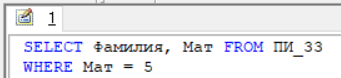
Добавление полей:



Обновление записей:



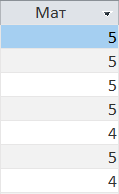
Запросы:



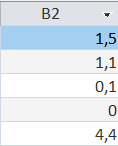
1. Типы данных языка SQL. Средства определения схемы.

Типы данных:

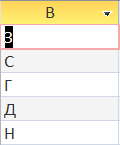
* Целые числа – int/integer, smallint, tinyint



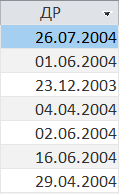
* Число с плавающей точкой – float, real



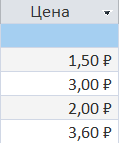
* Символьные данные – varchar



* Дата и время – datetime, smalldatetime



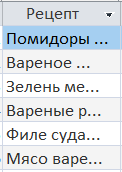
* Денежные суммы – money, smallmoney



* Двоичные числа – binary



* Длинный текст – text, image



* Булевы значения – bit

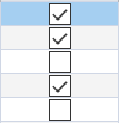
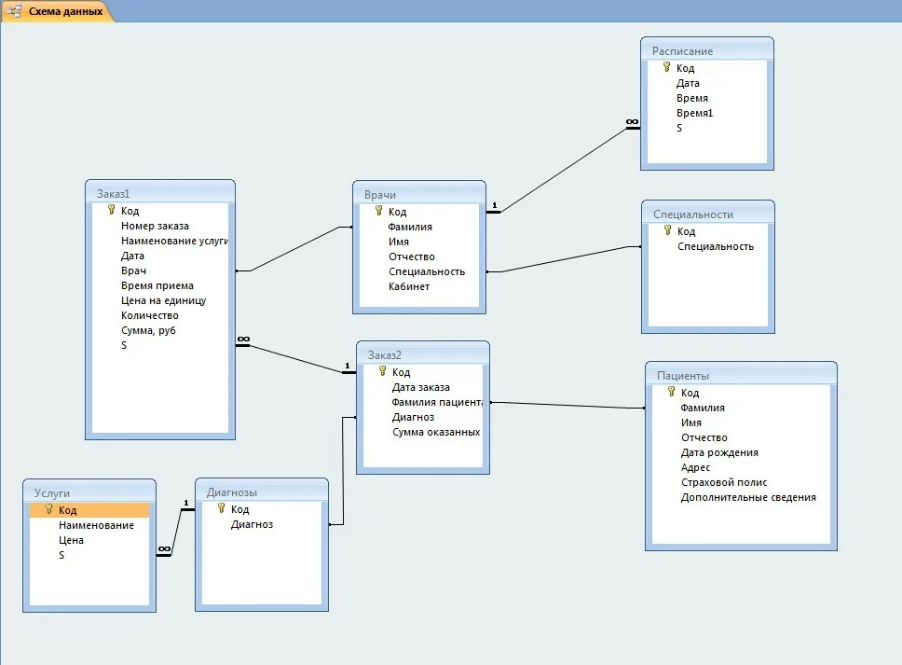


Схема базы данных включает в себя описание всех объектов базы данных, таких как таблицы, представления, индексы, ограничения и связи между ними.



Средства определения схемы могут быть реализованы через SQL-команды, программные средства и системы управления базами данных (СУБД), которые предоставляют функционал для проектирования и создания схемы. К ним относятся:

* ER-диаграммы
* Встроенные средства СУБД
* UML-диаграммы

DDL включает команды, которые используются для создания, изменения и удаления структур базы данных, таких как таблицы, индексы, схемы и другие объекты. Это операции, которые определяют структуру базы данных.

SQL-команды DDL:

* CREATE — используется для создания различных объектов схемы базы данных (например, таблиц, индексов, представлений).

CREATE TABLE имя\_табл (

Имя\_поля1 тд,

Имя\_поля2 тд,

…

);

* ALTER — используется для изменения существующих объектов схемы (например, добавления или удаления колонок в таблице).

ALTER TABLE имя\_табл

ADD имя\_поля тд // добавление поля

MODIFY имя\_поля новый\_тд // изменение типа данных поля

DROP COLUMN имя\_поля // удаление поля

* DROP — используется для удаления объектов схемы (например, таблиц, индексов или представлений).

DROP TABLE имя\_табл

1. Язык SQL, средства манипулирования данными. Структура запросов.

Средства манипулирования данными позволяют выполнять операции с данными, такие как вставка, обновление, удаление и извлечение данных. Эти операции помогают изменять и извлекать данные из базы данных, а также обеспечивать их актуальность и целостность.

DML (Data Manipulation Language) – язык манипуляции данными: DML включает команды, которые работают с содержимым базы данных, а именно с данными в таблицах. Основные операции, включаемые в DML, — это вставка, обновление и удаление данных.

Основные команды DML:

* INSERT — добавление новых записей в таблицу:

INSERT INTO имя\_табл

[(список\_полей)]

VALUES (список\_значений)

* UPDATE — обновление данных в существующих записях.

UPDATE имя\_табл

SET имя\_поля1 = выражение,

SET имя\_поля2 = выражение,

...

WHERE условие

* DELETE — удаление данных из таблицы.

DELETE FROM имя\_табл

[WHERE условие]

DQL – язык запросов к данным.

* SELECT - производит выборку и вычисление над данными из списка таблиц.

Результат выполнения - проекция таблицы, которая может иметь (all) или не иметь (distinct) повторяющиеся строки.

В отборе участвуют записи из таблиц, указанных после from. список содержит имена полей, участвующих в запросе, а также выражения над полями.

В списке полей из разных таблиц перед именем поле указывается имя таблицы, отделяемое точкой (имя\_табл.имя\_поля)

SELECT [ALL | DISTINCT]

список

FROM список\_табл

[where условие]

[group by имя\_поля1, имя\_поля2, ...]

[having условие\_поиска]

[order by спецификация]

SELECT \* FROM табл – вывод всей таблицы

1. Использование SQL при прикладном программировании.

В SQL-89 определены способы взаимодействия БД и прикладных программ, написанных на традиционных ЯПВУ (Fortran, Cobol, PL/1, Pascal, C).

Способы взаимодействия БД и программы:

* Операторы языка SQL собраны в модуль. Используются как процедуры. Прикладная программа вызывает процедуру модуля, передает ей фактические параметры и получает в ответ еще какие-то параметры. Модуль компилируется отдельно от прикладной программы и при этом порождается набор хранимых процедур.
  + Достоинства: прикладная программа имеет меньший размер, взаимодействие с СУБД локализовано
  + Недостатки: для понимания смысла прикладной программы требуется чтение 2 текстов, синтаксис модулей бывает разным
* Встроенный SQL. Непосредственно в программе на ЯПВУ имеются SQL-операторы. Обмен данными через переменные. SQL-операторы компилируются каждый раз при их использовании, точнее при каждом первом использовании оператора при данном запуске программы.
  + Достоинства: прикладная программа более самосодержащаяся (2 в 1), стандарт встраивания единообразен
  + Недостатки: синтаксис SQL-операторов контрастирует с синтаксисом базового языка

SQL используется для следующих задач в приложениях:

* Хранение данных: Данные, такие как информация о пользователях, продуктах, заказах и транзакциях, сохраняются в реляционных базах данных.
* Манипуляция данными: Программы используют SQL для выполнения операций добавления, обновления, удаления или выборки данных.
* Аналитика и отчёты: SQL помогает в реализации аналитических функций для получения статистики, агрегации данных и создания отчетов.
* Обеспечение целостности данных: С помощью SQL-программ реализуются механизмы валидации, ограничения и транзакции.
* Безопасность: SQL-запросы применяются в сочетании с механизмами авторизации для защиты данных.

Основные области применения SQL:

* Web-разработка: SQL используется для управления базами данных, которые поддерживают работу веб-приложений
* Мобильные приложения: многие мобильные приложения используют SQL для локального или удалённого хранения данных
* Аналитика данных: SQL — основной инструмент для анализа данных в корпоративной среде
* Финансовые приложения: SQL широко применяется в разработке финансовых систем
* Игровая индустрия: SQL используется для управления игровыми данными
* Электронная коммерция: SQL активно применяется для управления данными в интернет-магазинах
* Администрирование баз данных: SQL применяется для управления структурой базы данных и контроля доступа

1. Проблемы оптимизации. Схема обработки запроса.

Проблемы оптимизации в реляционной СУБД связаны с тем, что выполнение SQL-запросов может занимать значительное время и ресурсы из-за объёмов данных, сложности запросов, неэффективной структуры базы и других факторов. Основная цель оптимизации — минимизировать затраты на выполнение запросов и максимально эффективно использовать доступные ресурсы.

Оптимизация запросов в реляционной СУБД – это такой способ разработки запросов, когда по его начальному представлению вырабатывается процедурный план его выполнения, наиболее оптимальный для существующих в БД управляющих структур.

Основные проблемы оптимизации:

1. Отсутствие индексов или их неправильное использование – индексы значительно ускоряют поиск данных. Однако отсутствие индексов или их использование на ненужных полях приводит к увеличению времени выборки данных и сложностям с обновлением индексов при изменении данных.

Решение: создать индекс

1. Низкая эффективность запросов – неоптимальные запросы (например, запросы с избыточными таблицами или ненужными вычислениями) приводят к большим накладным расходам.

Решение: минимизировать выборку – запросить только нужные поля

1. Избыточные данные – если данные хранятся в избыточной форме (например, дублируются значения), это увеличивает объём хранимой информации и время выполнения операций.

Решение: Нормализация базы данных – разделение таблиц на более мелкие и избавление от избыточности.

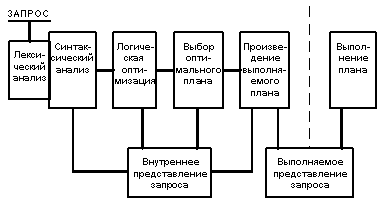
1. Плохая структура базы данных – неудачная структура базы данных может привести к лишним соединениям или сложным запросам.

Решение: Оптимизировать схему базы данных. Использовать денормализацию для уменьшения числа соединений, если это критично для производительности.

1. Большие объёмы данных - при работе с большими объёмами данных выполнение запросов может занимать значительное время.

Решение: разделение таблиц, архивирование старых данных

Схема обработки запроса:



1. Пользователь формулирует SQL-запрос, чтобы извлечь, изменить или удалить данные из базы данных. Этот запрос может быть написан вручную в командной строке или с использованием какого-либо интерфейса (например, в графическом интерфейсе СУБД, или в приложении).
2. Запрос отправляется в СУБД и подвергается лексическому и синтаксическому анализу, Пользователь не видит этих этапов, но они обеспечивают корректность запроса, который он написал. Если запрос содержит ошибки (например, синтаксические), СУБД вернет сообщение об ошибке, и пользователь должен будет исправить запрос. Если ошибок нет, система формирует внутреннее представление запроса, которое отражает структуру его обращения к данным. Эта информация выбирается из каталогов БД.
3. После того как запрос прошел анализ, СУБД начинает его оптимизировать, чтобы выполнить запрос быстрее и с меньшими затратами. Преобразования бывают эквивалентные, семантические (т. е. новое представление не является эквивалентным начальному, но гарантируется, что результат выполнения совпадает с запросом в начальной форме). Внутреннее представление остается непроцедурным, но уже стремится к этому. Пользователь в этот момент не участвует напрямую, но оптимизация запроса может повлиять на результаты выполнения. Например, запрос, который изначально работает медленно, может быть оптимизирован для улучшения производительности.
4. СУБД выбирает лучший план выполнения запроса из нескольких возможных вариантов: для каждого плана оценивается стоимость выполнения запроса на основе статистической информации о состоянии БД. Из них выбирается самый "дешевый". Пользователь не видит этого этапа, но он может повлиять на выбор плана косвенно, например, через структуру данных (путем правильного индексирования таблиц, выбора структуры базы данных и т.д.).
5. Из оптимального плана формируется выполняемое представление запроса. Оно может быть и программой в машинных кодах, и машинно-независимым - более удобным в интерпретации. На этом этапе пользователь не участвует напрямую, но он уже может видеть, как СУБД начинает выполнять запрос.
6. Реальное выполнение запроса – это либо выполнение программы, либо вызов интерпретатора и передача ему плана. Когда запрос выполняется, результаты выполнения возвращаются пользователю. Это может быть набор данных, сообщение о успешном выполнении операции или информация об ошибке, если что-то пошло не так. Пользователь получает результаты в том виде, в котором они были запрашиваемы, и может использовать их для дальнейших действий (например, вывод на экран, сохранение в файл, анализ и т.д.).

Пользователь активно участвует на начальном этапе (формирование запроса), а также может влиять на результаты через оптимизацию запросов и настройку базы данных.

1. Архитектура "клиент-сервер". Серверы баз данных.

Архитектура клиент-сервер – это модель взаимодействия, где функции и задачи системы распределены между двумя основными компонентами: клиентом и сервером. Эта архитектура широко используется в информационных системах, приложениях и базах данных.

Сервер – это компьютер или приложение, управляющее некоторым ресурсом.

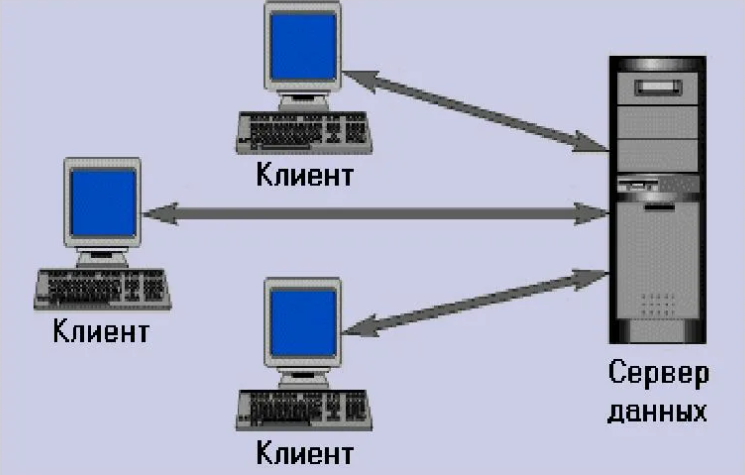
Клиент – это компьютер или приложение, которое использует этот ресурс.

Ресурсом могут быть:

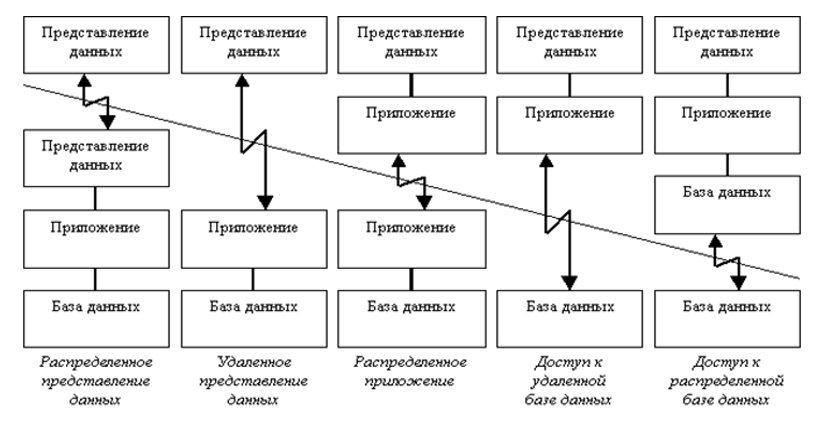
* Файловая система
* Служба печати
* Почтовые службы
* Web-сервер
* Сервер БД

В архитектуре "клиент-сервер" на основе БД участники решают задачи управления данными, обработки и представления.

Двухзвенная модель «клиент-сервер»



Распределение функций:



1. В модели распределенного представления имеется мощный компьютер-сервер, а клиент вырожден - лишь отражает информацию на мониторе. Компьютер мощный, чтобы обслуживать несколько терминалов. По этой модели работают неоднородные гетерогенные системы. Серверная часть обеспечивает унифицированный интерфейс, а клиенты - функции конечного оборудования или преобразование одного формата данных в другой.
2. Модель распределенной базы данных предполагает наличие мощного компьютера-клиента. Данные хранятся на компьютере-клиенте и на компьютере-сервере. Взаимосвязь обеих баз данных может быть двух разновидностей: в локальной и удаленной базах хранятся отдельные части единой базы данных; локальная и удаленная базы данных являются синхронизируемыми друг с другом копиями.
3. Модель распределенной функции применяется в системах, где общая часть прикладной функции реализована на сервере, а специфические функции на клиенте
4. Модель удаленного доступа - функции представления информации и функции логики прикладной программы выполняются на клиенте, запрос к серверу происходит через сеть операторами SQL или через библиотеку API. Данные хранятся и на клиенте, и на сервере.

Трехзвенная модель «клиент-сервер»



Каждая из 3 функций реализуется на отдельном компьютере:

* Управление данными - сервер БД
* Обработка данных - сервер приложений - он решает прикладную задачу как службу предоставления услуг клиенту; их может быть несколько - каждый со своим сервисом
* Представление данных - клиент - трактуется более широко, служит для приема данных, в т. ч. от датчиков

В более сложных системах взаимодействия сервер некоторого клиента сам становится клиентом по отношению к другому серверу.

Сервер базы данных – это программное или аппаратное обеспечение, которое предоставляет доступ к базе данных, обрабатывает запросы пользователей и управляет данными. Сервер БД является важным элементом в архитектуре клиент-сервер и отвечает за хранение, управление и обеспечение целостности данных.

Основные функции сервера базы данных:

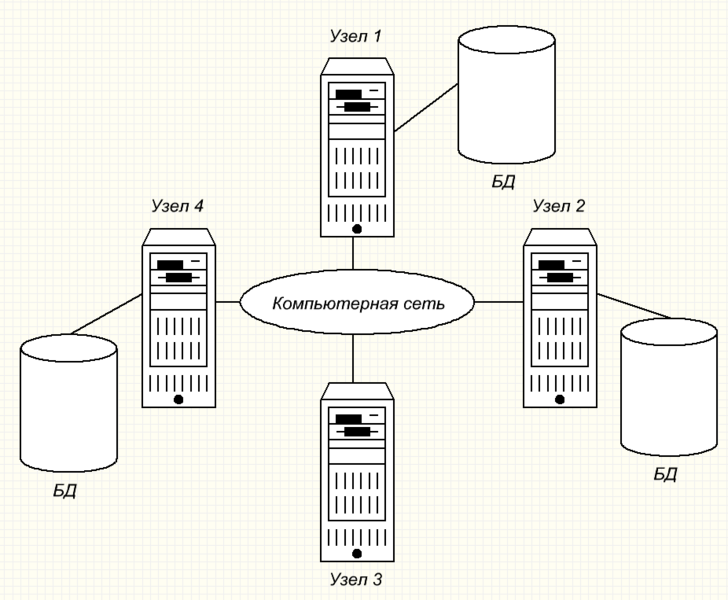
1. Хранение данных: сервер БД сохраняет данные в структурированном виде, как правило, в виде таблиц (в реляционных СУБД).
2. Обработка запросов: выполнение операций над данными (вставка, обновление, удаление, выборка) на основе запросов, отправленных клиентами через язык SQL.
3. Управление транзакциями: обеспечение атомарности, согласованности, изолированности и долговечности (ACID) операций.
4. Обеспечение целостности данных: проверка ограничений (например, уникальность, внешние ключи) и соблюдение правил бизнес-логики.
5. Обеспечение безопасности: управление доступом к данным (аутентификация, авторизация) и шифрование при необходимости.
6. Оптимизация запросов: построение оптимальных планов выполнения запросов для повышения производительности.
7. Резервное копирование и восстановление: создание резервных копий данных и восстановление после сбоев.

Структура сервера базы данных:

* Подсистема взаимодействия с клиентским приложением: С ожидает запроса К на установление соединения, затем порождает новый процесс, кот обеспечивает связь с данным клиентом.
* Подсистема синтаксического разбора запросов: Данный модуль отвечает за компиляцию поступающих от клиентов через интерфейсные процессы запросов во внутренний код, который будет исполняться сервером.
* Подсистема планирования выполнения запросов: Данный модуль должен составить такой план выполения запроса, чтобы он был обработан наиболее быстро.
* Подсистема выполнения транзакций: Здесь выполняется оптимизированный код запроса, обновляются индексы, выполняются в случае необходимости триггеры и хранимые процедуры. Ведется журнал транзакций.
* Подсистема управления памятью: Этот компонент отвечает за считывание данных с диска в оперативную память, синхронизацию обновленний с данными диске и т.д.

1. Распределенные БД.

Распределённые базы данных – это базы данных, где данные хранятся на нескольких узлах (компьютерах, серверах) в сети. Эти узлы могут находиться в разных географических местах, но для пользователя база данных выглядит как единая система.



Особенности распределённых баз данных:

1. Физическая распределённость: данные хранятся на нескольких серверах или устройствах, которые могут быть географически удалены друг от друга.
2. Логическая целостность: пользователь взаимодействует с базой как с единой системой, даже если данные хранятся на разных узлах.
3. Взаимосвязанность узлов: узлы базы данных обмениваются данными и обеспечивают согласованность.

Преимущества распределённых баз данных:

1. Надёжность: даже если один узел выходит из строя, другие продолжают работу.
2. Производительность: задачи обработки данных можно распределять между узлами, снижая нагрузку на каждый из них.
3. Масштабируемость: легко добавлять новые узлы для увеличения объёма данных или обработки запросов.
4. Локальная доступность: данные, необходимые для работы в определённом регионе, могут храниться ближе к пользователям этого региона, что уменьшает задержки.
5. Гибкость: возможность использовать различные платформы и технологии на разных узлах.

Недостатки распределённых баз данных:

1. Сложность управления: настройка и администрирование распределённой системы сложнее, чем у централизованных БД.
2. Согласованность данных: труднее обеспечить, чтобы данные на всех узлах были актуальными (особенно при сбоях).
3. Задержки: доступ к удалённым узлам может быть медленнее из-за сетевых ограничений.
4. Безопасность: распределённость данных усложняет защиту системы от несанкционированного доступа.

Архитектуры распределённых баз данных:

* Гомогенная архитектура: все узлы используют одну и ту же СУБД и имеют одинаковую структуру.
* Гетерогенная архитектура: узлы могут использовать разные СУБД и различные структуры данных.

Типы распределённых баз данных:

* Фрагментированное хранение: база данных разбивается на части (фрагменты), которые хранятся на разных узлах.
* Реплицированное хранение: полные копии базы данных хранятся на нескольких узлах для повышения доступности.
* Смешанное хранение: сочетание фрагментирования и репликации.

1. Системы управления базами данных следующего поколения.

Направления развития СУБД:

* Поиск совершенных моделей представления данных и типов данных
* Разработка новых архитектур СУБД
* Расширение областей применения БД
* Улучшение сервиса пользователей, администраторов, разработчиков
* Обработка сверхбольших объемов информации

Развиваются СУБД, поддерживающие несколько моделей данных. Такие СУБД позволяют обрабатывать символьную и графическую информацию, работать со знаниями, аудио-, видеоинформацией.

СУБД следующего поколения – это современные системы управления базами данных, которые разрабатываются с учётом новых требований и вызовов, связанных с большими объёмами данных, распределённостью, высокой скоростью обработки и гибкостью. Такие СУБД отличаются от традиционных реляционных баз данными особенностями архитектуры, подходами к обработке данных и применяемыми технологиями.

Основные характеристики СУБД следующего поколения:

* Поддержка больших данных: обработка больших объёмов структурированных, полуструктурированных и неструктурированных данных. Эффективная работа с потоковыми данными.
* Масштабируемость: горизонтальная масштабируемость за счёт добавления серверов. Обеспечение высокой производительности в распределённых системах.
* Работа в облаке: поддержка гибридных решений (локальное + облачное хранение данных). Облачные сервисы баз данных, такие как Google BigQuery, Amazon Aurora, Azure Cosmos DB.
* Поддержка новых типов данных: документы, графы, ключ-значение, временные ряды, геоданные и другие форматы.
* Новые модели согласованности: переход от строгой ACID-согласованности к более гибким моделям (например, BASE), чтобы повысить производительность и надёжность.
* Высокая производительность: использование технологий распределённой обработки и параллелизма. Индексация в реальном времени, оптимизация запросов и кэширование.
* Гибкость схем данных: возможность изменения структуры данных "на лету" без необходимости остановки системы. Поддержка схем, которые могут меняться динамически.

Классификация СУБД следующего поколения:

1. NoSQL базы данных:

Созданы для работы с неструктурированными данными и обеспечения гибкости.

Типы:

* Документоориентированные: MongoDB, Couchbase.
* Графовые: Neo4j, ArangoDB.
* Ключ-значение: Redis, DynamoDB.
* Колонноориентированные: Cassandra, HBase.

Применение: Большие данные, социальные сети, IoT.

2. NewSQL базы данных:

Объединяют преимущества реляционных СУБД (ACID, SQL) и NoSQL (масштабируемость, производительность).

Примеры: CockroachDB, Google Spanner, TiDB.

Применение: финансовые системы, сложные аналитические задачи.

3. Гибридные СУБД:

Поддерживают как реляционные, так и нереляционные модели данных.

Пример: Azure Cosmos DB, OrientDB.

Применение: разные задачи в одной системе (например, хранение документов и реляционных данных).

4. СУБД для потоковых данных:

Заточены на обработку данных в реальном времени.

Примеры: Apache Kafka, Apache Flink.

Применение: системы мониторинга, финансовые транзакции.

5. Базы данных для машинного обучения:

СУБД, оптимизированные для хранения и обработки данных, используемых в ML.

Примеры: TensorFlow Extended (TFX), MLDB.

1. Объектно-ориентированные СУБД.

Объектно-ориентированные системы управления базами данных (ООСУБД) – это системы управления базами данных, которые используют объектно-ориентированный подход для представления, хранения и управления данными. В таких системах данные представлены в виде объектов, а связь между ними осуществляется через механизмы объектно-ориентированного программирования (ООП), такие как наследование, полиморфизм и инкапсуляция.

Это направление возникло в середине 70-х гг. Однако наиболее активно это направление развивается в последние годы.

Понятия, работающие в ООСУБД:

* Объект и его идентификатор
* Атрибуты и методы
* Классы
* Иерархия, наследование классов

Сущность реального мира моделируется в виде объекта.

Объект имеет идентификатор – состояние, поведение.

Состояние – это набор значений и его атрибутов.

Поведение – набор методов, оперирующих над состоянием.

Множество объектов с одним набором атрибутов и методов образуют класс.

Новое качество у ООСУБД в поведенческом аспекте объектов. Раньше существовал разрыв между структурной и поведенческой частями - структурная часть поддерживалась аппаратом БД, а поведенческая создавалась изолированно.

Поддержка объектно-ориентированных концепций:

* Инкапсуляция: данные и методы объединены в одном объекте.
* Наследование: возможность создавать классы на основе существующих.
* Полиморфизм: одинаковый интерфейс для различных типов объектов.
* Иерархия классов: организация объектов через суперклассы и подклассы.

Примеры ООСУБД:

1. O2 — это объектно-ориентированная СУБД, которая была разработана для работы с объектами в базах данных. Она поддерживает стандарт объектно-ориентированного программирования и предлагает мощные средства для работы с данными, используя концепции ООП:

* Хранение объектов: Поддерживает хранение данных в виде объектов, которые могут включать как данные, так и методы.
* Наследование: О2 поддерживает механизмы наследования, позволяя объектам наследовать атрибуты и поведение от других объектов.
* Инкапсуляция и полиморфизм: Эти принципы ООП также поддерживаются для управления доступом и реализации разнообразных действий с объектами.
* Моделирование сложных данных: O2 предназначена для работы с большими объемами сложных данных и является мощным инструментом для разработки и использования сложных систем.
* Использовалась в системах, где необходима интеграция с объектно-ориентированными языками программирования, такими как C++, Java и другие.
* Применялась в научных и инженерных приложениях, где важно хранение объектов, таких как графы, деревья и другие сложные структуры.

1. Orion — это объектно-ориентированная СУБД, предназначенная для управления объектами в базах данных с использованием объектно-ориентированного подхода. Эта система была разработана для работы в распределенных средах и была одной из первых СУБД, которая поддерживала объектную модель данных:

* Моделирование данных: Orion предоставляет средства для моделирования объектов и их связей, поддерживает сложные типы данных, такие как коллекции объектов.
* Транзакции и безопасность: Поддерживает ACID-транзакции, что обеспечивает надежность и целостность данных.
* Дистрибуция данных: Поддержка распределенных баз данных, что позволяет управлять объектами в разных узлах сети и обеспечивать масштабируемость системы.
* Интерфейсы и язык запросов: Orion использует запросы на основе объектно-ориентированного языка, что позволяет напрямую работать с объектами в базе данных, без необходимости их преобразования в реляционную модель.
* Orion использовалась в крупных распределенных приложениях, где требовалась высокая производительность и возможность масштабирования. Она нашла применение в таких областях, как телекоммуникации, финансовые сервисы и другие высоконагруженные системы.
* Эта система позволяла легко интегрировать сложные структуры данных в приложения и обеспечивать эффективное управление ими.

1. ObjectDB – это объектно-ориентированная СУБД, предназначенная для Java-программ:

* Работает с объектами Java напрямую.
* Позволяет хранить сложные объекты, как они есть, без необходимости преобразования их в реляционные таблицы.
* Поддержка транзакций и индексации для быстрого поиска.
* Простой в использовании и интеграции с Java-приложениями.
* Используется в Java-приложениях для хранения и обработки данных, когда важна интеграция с объектной моделью приложения.

1. db4o — это объектно-ориентированная база данных с открытым исходным кодом. Она поддерживает хранение и управление объектами, которые используются в языке программирования:

* Объекты сохраняются и извлекаются в том виде, в каком они есть в коде.
* Поддержка языков программирования, таких как Java и .NET.
* Легкость в использовании для разработки приложений.
* Позволяет хранить сложные объекты, такие как коллекции и графы.
* Широко используется в приложениях на Java и .NET для упрощения взаимодействия с данными.

1. Versant Object Database — это высокопроизводительная объектная СУБД, которая поддерживает объектно-ориентированные принципы. Она интегрируется с языками программирования, такими как Java, C#, и C++:

* Поддержка сложных объектов и транзакций.
* Подходит для распределенных и масштабируемых приложений.
* Обеспечивает быструю работу с объектами, улучшая производительность по сравнению с реляционными СУБД в некоторых случаях.
* Используется в высоконагруженных приложениях, где важна работа с объектами и требования к производительности.

1. Описание: GemStone/S — объектно-ориентированная СУБД, которая используется для работы с объектами, написанными на языке Smalltalk:

* Поддерживает многозадачность, транзакции и распределенные вычисления.
* Обеспечивает долговечность данных и их целостность.
* Предназначена для корпоративных приложений, которые требуют масштабируемости и высокой доступности.
* Применяется в корпоративных приложениях на Smalltalk, где требуется эффективная работа с объектами и управление большим объемом данных.

Языки ООСУБД:

1. Smalltalk-80 - это объектно-ориентированный язык программирования, который был одним из первых, разработанных с нуля с ориентацией на объекты. Он был использован в нескольких объектно-ориентированных СУБД, таких как GemStone и ObjectStore, благодаря своей гибкости и чистоте объектной модели:

* В Smalltalk все, включая числа, строки и даже классы, рассматриваются как объекты.
* Он поддерживает динамическую типизацию и полноценно работает с наследованием, инкапсуляцией и полиморфизмом.
* Smalltalk активно использовался в системах для построения объектно-ориентированных баз данных, таких как GemStone/S и ObjectStore. Эти системы позволяют объектам Smalltalk быть непосредственно сохраненными и извлеченными из базы данных без необходимости преобразования в реляционную форму.

1. Common Lisp - это язык программирования, который поддерживает функциональное, процедурное и объектно-ориентированное программирование.

* В CLOS объекты могут быть созданы, наследованы и полиморфно взаимодействовать.
* Объектно-ориентированное программирование в Lisp тесно связано с его метапрограммированием, что позволяет создавать новые структуры типов данных на лету.
* С помощью CLOS и других механизмов Lisp можно эффективно работать с объектами в базах данных, интегрируя их с объектно-ориентированными СУБД.
* Язык поддерживает расширенные метапрограммирующие возможности, такие как динамическая генерация и модификация классов и методов, что делает его удобным для создания и интеграции сложных систем управления объектами.

1. OQL — это запросный язык, разработанный специально для работы с объектно-ориентированными базами данных. Он аналогичен SQL, но ориентирован на работу с объектами:

* Позволяет делать запросы к объектам, а не к строкам таблиц.
* Поддерживает операции выборки, фильтрации, сортировки и агрегации объектов.
* Операции включают работу с вложенными объектами, наследованием и полиморфизмом.

1. ODM - это язык для работы с объектными данными в объектно-ориентированных СУБД. Включает в себя стандартные операции по работе с объектами, а также их сохранение и извлечение из базы данных.

* Управляет как структурированными, так и неструктурированными объектами.
* Может использоваться в сочетании с языками программирования, такими как C++ или Java, для интеграции с объектными хранилищами.

Преимущества ООСУБД:

* Интеграция с ООП-языками: легко интегрируются с языками программирования, использующими объектно-ориентированный подход.
* Гибкость: поддерживают сложные структуры данных, которых нет в реляционных СУБД.
* Простота моделирования: упрощается работа с данными, так как объектная модель ближе к реальным приложениям.
* Поддержка мультимедиа и сложных данных: эффективно работают с изображениями, видео, аудио, геоданными и другими сложными типами данных.
* Наследование: позволяет повторно использовать структуру и функциональность объектов.

Недостатки ООСУБД:

* Меньшая зрелость и распространённость: по сравнению с реляционными СУБД, объектные менее популярны и развиты.
* Сложность обучения: требует от разработчиков знаний как баз данных, так и объектно-ориентированного программирования.
* Производительность: объектная структура может быть менее эффективной для простых запросов по сравнению с реляционными базами.
* Меньшая стандартизация: в отличие от SQL, единых стандартов для ООСУБД практически нет.
* Совместимость: интеграция с другими СУБД может быть затруднена.