[1.Сферы применения баз данных и СУБД 1](#_Toc1208616032)

[2.Базы данных и файловые системы. Области применения файлов. Потребности информационных систем. 2](#_Toc1962942085)

[3.Основные функции СУБД. Типовая организация СУБД 3](#_Toc1148012352)

[4.Ранние подходы к организации БД. 4](#_Toc2094703142)

[5.Реляционный подход к организации БД. Базовые понятия баз данных реляционных. 5](#_Toc871896682)

[6.Фундаментальные свойства отношений. Реляционная модель данных. 6](#_Toc111119366)

[7.Базисные средства манипулирования реляционными данными. Реляционная алгебра. Реляционные исчисления. 8](#_Toc1195578735)

[8. Проектирование реляционных БД. Нормализация. 10](#_Toc1624389017)

[9.Основные понятия модели Сущность-связь. ER – диаграммы. 12](#_Toc967901720)

[10. Система System R. Основные цели System R. 15](#_Toc10628116)

[12. Транзакции и целостность баз данных. Сериализация транзакций. 17](#_Toc847353471)

[13. Изолированность пользователей. 19](#_Toc15509666)

[14. Журнализация изменений БД. Журнализация и буферизация. 21](#_Toc1381198286)

[15. Восстановление баз данных в различных ситуациях. 23](#_Toc694043971)

[16. Язык SQL, функции и основные возможности. Стандартизация SQL. 25](#_Toc708873386)

[17. Типы данных языка SQL. Средства определения схемы. 26](#_Toc452933777)

[18. Язык SQL, средства манипулирования данными. Структура запросов. 28](#_Toc967686126)

[Операторы DML 29](#_Toc992019646)

[Операторы DQL 29](#_Toc562331308)

[19. Использование SQL при прикладном программировании. 29](#_Toc1829702993)

[20. Проблемы оптимизации. Схема обработки запроса. 30](#_Toc1838069861)

[21. Архитектура "клиент-сервер". Серверы баз данных. 31](#_Toc1956077035)

[22. Распределенные БД. 33](#_Toc285306772)

[23. Системы управления базами данных следующего поколения. 35](#_Toc183974378)

[24. Объектно-ориентированные СУБД. 37](#_Toc1680367741)

# 1.Сферы применения баз данных и СУБД

Банк данных (БНД) – это система, состоящая из:

* одной или нескольких баз данных;
* системы управления базами данных (СУБД);
* словаря данных;
* администратора БД (АБД);
* самой вычислительной техники, на которой всё это реализовано;
* обслуживающий персонал.

База данных – совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в данной предметной области. Логическую структуру хранимых в БД данных называют моделью представления данных (модель данных). Физической моделью предметной области является БД.

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

СУБД – класс программ: Oracle, 1C, Access.

Приложение – программа для автоматизации, обработки информации в прикладной задаче.

Словарь данных, подсистема БНД для централизованного хранения инфы о структуре данных, взаимосвязь данных, формату их представлениях принадлежности данных пользователям, разграничениях доступа, кодах защиты и др.

АБД (администратор(ы) БД) – лицо или группа лиц отвечающих за выработку требований к БД, её проектирование, создание, эффективное использование, следит за функционированием ИС, обеспечивает защиту от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность.

Информационная система = БД + приложение работы с ней.

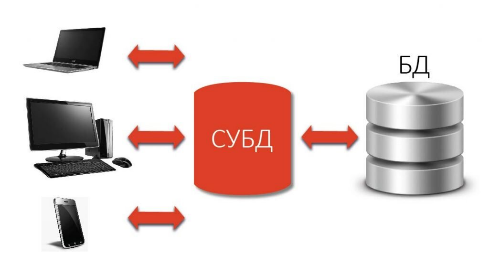


Рис. 1. Упрощенная схема использования СУБД

ЭВМ в 1945 появилась для выполнения численных расчетов. Это позволило сформировать методы численного решения математических задач. Развивались языки программирования, ориентированные на удобную запись вычислительных алгоритмов. Появилась тесная обратная связь с разработчиками новых архитектур ЭВМ.

Использование БД в ЭВМ:

Для выполнения численных расчётов – позволило сформировать методы численного решения математических задач. Развивались ЯП, ориентированные на удобную запись численных алгоритмов. Появилась тесная обратная связь с разработчиками ЭВМ.

Использование БД в автоматизированных системах – возникло позже первого, потому что вычислительная техника обладала маленькой памятью. Необходима память, способная хранить данные после выключения питания. Решение: магнитные ленты и барабаны. Ленты были большие, ёмкие, последовательный доступ к данным, скорость к доступу мала. Барабаны имели произвольный доступ, но малый размер. С появлением магнитных дисков совершенствуется практика управления данными во внешней памяти. Тогда каждая программа сама определяла расположение данных во внешней памяти и выполняла обмен между ОЗУ и ВЗУ с помощью программно-аппаратных средств низкого уровня.

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

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

Файл – это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Конкретная файловая система определяет правила именования файлов, структуру файлов, способ доступа к данным файла. Она берёт на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

Первая развитая файловая система была разработана фирмой IBM для OS/360. Понятие файла в OS/360 было выбрано как основное абстрактное понятие, которому соответствовал любой внешний объект, включая внешние устройства.

“Сегодня” ВЗУ – это магнитный диск с подвижными головками; пакет магнитных пластин, между которыми пакет магнитных головок, шаг движений этих головок дискретный, каждому положению головок соответствует “цилиндр” магнитного диска. Каждая дорожка размечается на одинаковое количество блоков, так, чтобы в каждый записать по максимуму одно и то же число байт. Для обмена с магнитным диском надо знать номер “цилиндра”, номер поверхности, номер байт, которое нужно прочитать или записать от начала блока.

Рис. 2. Пример устройства ВЗУ Рис. 3. Примерные типы файлов

Существует 2 подхода к реализации файлов:

* ОС фирмы DEC: файл – это последовательность байт постоянного или переменного размера. Её 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. Современные файловые системы поддерживают многоуровневое именование файлов за счет поддержания во внешней памяти дополнительных файлов — каталоги, папки, директории. Таким образом полное имя файла состоит из списка имён каталогов.

Области применения файлов:

Для хранения текстовых документов (.txt) — это последовательность записей со строками текста и байт со специальными символами.

Объектные модули — последовательность записей или байт, определяется системой программирования и неизвестно файловой системе.

Содержащие образы выполняемых программ (.exe) — структура известна редактору связей и загрузчику ОС.

Файлы с графической и звуковой информацией, файлы документов.

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.

# 3.Основные функции СУБД. Типовая организация СУБД

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

Система управления базами данных (СУБД) – комплекс языковых и программных средств, предназначенных для создания, ведения и для совместного использования многими пользователями. СУБД должна:

1. Поддерживать логически согласованный набор файлов.

2. Обеспечивать язык манипулирования данными.

3. Восстанавливать информацию после сбоев.

4. Поддерживать параллельную работу нескольких пользователей.

Функции СУБД:

* Непосредственное управление данными ВЗУ - СУБД может обращаться к файлам через использование файловой системы или напрямую.
* Управление буферами ОЗУ.
* Управление транзакциями.
* Журнализация - одно из требований к СУБД - надёжное хранение данных. СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного и программного сбоя. Избыточная информация хранится в журнале изменений (ЖИ) БД. Журнал изменений БД - особая часть БД, недоступная пользователям СУБД, поддерживаемая с особой тщательностью. Журнал содержит записи обо всех изменениях в основной части БД. Ведется 2 вида журналов - локальный и общесистемный. есть 2 вида сбоев - мягкий и жёсткий. После мягкого сбоя используется локальный журнал для восстановления, после жёсткого - общесистемный.
* Поддержка языков баз данных.

SQL - Structured Query Language = DDL + DML + DQL.

DDL (SDL) - Data Definition Language - язык определения данных.

DML - Data Manipulation Language - язык управления данными.

DQL - Data Query Language - язык запроса данных.

Типовая организация СУБД:

* Внутренняя часть (ядро) СУБД - DBE (Data Base Engine) - отвечает за управление данными во внешней памяти, управление ОЗУ, управление ТР, журнализацию, обладает собственным интерфейсом, недоступна пользователю, но используется в программах.
* Компилятор языка (транслятор) - переводчик с языка SQL на машинный. Проблема современных СУБД в том, что язык СУБД непроцедурный, поэтому компилятор должен решать, как выполнить оператор прежде чем произвести программу. Результат компиляции представлен или в машинных кодах, или во внутреннем машинно-независимом коде.
* Подсистема поддержки времени выполнения - работа направлена на выполнение машинно-независимого кода, является интерпретатором.
* Утилиты - процедуры, которые накладно выполнять использованием языка БД: загрузка, выгрузка БД (по окончании работы принято БД закрывать), сбор статистики, глобальная проверка целостности. Рис. 4. Типовая организация СУБД



# 4.Ранние подходы к организации БД.

Они исторически предшествовали реляционным СУБД. На основе методов, используемых в них, организованы и реляционные.

1) Они использовались долго, были накоплены ГРОМАДНЫЕ базы данных, их надо использовать совместно с реляционными.

2) Они не строились на абстрактных моделях. В них был организован интерактивный доступ на уровне записей. Это требовало писать прикладные программы со своим интерфейсом.

Вывод по ранним подходам к СУБД

Достоинства

1) Развитые средства управления данными в ОЗУ на низком уровне.

2) Возможность построения вручную эффективных прикладных программ.

3) Возможность экономии памяти за счёт разделения объектов .

Недостатки

1) Слишком сложно пользоваться.

2) Необходимо знание физической организации (куда что записали в память).

3) От физической организации зависят прикладные системы.

4) Логика программы перегружена деталями доступа к БД.

**Системы, основанные на инвертированных списках.**

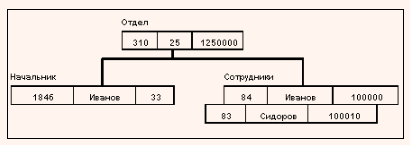
Похожа на реляционную БД, но с тем отличием, что хранимые таблицы и пути доступа к ним видны пользователям. При этом: 1) строки таблиц упорядочены системой в некоторой физической последовательности; 2) физическая упорядоченность строк всех таблиц может определяться и для всей БД; 3) для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы.

Datacom/DB и Adabas

**Иерархические системы.**

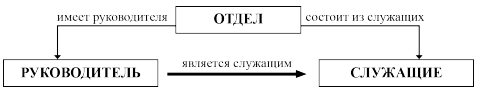
Иерархическая БД состоит из упорядоченного набора деревьев. Тип дерева состоит из одного "корневого" типа записи и упорядоченного набора из нуля или более типов поддеревьев. Тип дерева в целом представляет собой иерархически организованный набор типов записи. Дерево - иерархически организованный набор записей. Запись - потомок, должен иметь одного предка.





**Сетевые системы.**

Является расширением иерархического. Потомок может иметь любое число предков. Состоит из набора записей и набора связей между этими записями. Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка.



# 5.Реляционный подход к организации БД. Базовые понятия баз данных реляционных.

От слова relation - отношение.

Достоинства:

* Подход наиболее распространен сегодня.
* Наличие небольшого набора абстракций, на основе которых просто моделировать распространенные предметные области.
* Наличие простого, но мощного математического аппарата из теории множеств и мат. логики.
* Возможность манипулирования данными без знания конкретной физической реализации во внешней памяти.

Недостатки:

* Ограниченность в некоторых нетрадиционных областях - системах автоматизации проектирования, машиностроении, там, где используются сложные структуры данных.

Реляционная база данных — это составленная по реляционной модели база данных, в которой данные, занесенные в таблицы, имеют изначально заданные отношения.

Транзакция — это комплекс последовательных операций с применением операторов SQL, имеющих определенную цель. Все транзакции должны отвечать четырем требованиям ACID:

Атомарность — транзакция является неделимым блоком и выполняется или полностью, или никак.

Согласованность — завершенная транзакция сохраняет согласованность базы данных.

Изолированность — параллельные транзакции не могут влиять друг на друга.

Устойчивость — никакой сбой в системе не может влиять на результат завершенной транзакции.

Базовые понятия:

Тип данных: наряду с традиционными (текстовый, символьный, числовой, дата время), есть и специальные типы данных.

Домен: Схожее с подтипами в языках программирования. Домен — это допустимое потенциальное множество значений данного типа.

Атрибут: характеристика того или иного объекта (Фамилия, И, О, оценка по одному предмету, по-другому - это всё атрибуты)

Схема отношения: именованное множество пар типа (имя атрибута, имя типа данных). Степень (арность) типа отношения: мощность (кол-во полей).

Схема базы данных — набор именованных схем отношений.

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

Отношение — множество кортежей, соответствующих одной схеме отношения. Различают отношения-схему и отношения-экземпляр.

Тело кортежа — набор кортежей.

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



# 6.Фундаментальные свойства отношений. Реляционная модель данных.

Фундаментальные свойства отношений:

1) Значения атрибутов должны быть атомарны. Так как 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 повторяющихся записей. Свойство следует из определения отношения как множества кортежей. А в классической теории множеств - "каждое множество состоит из различных элементов". Это свойство требует у каждого отношения наличия первичного ключа - (набора) атрибутов, значение которых однозначно определяют кортеж отношения. Могут быть альтернативные ключи. Внешний ключ - ну есть и есть.

3) Отсутствие упорядоченности кортежей. Это следует из определения "отношение – это множество кортежей".

4) Отсутствие упорядоченности атрибутов. Для ссылки на значение атрибута всегда используется имя атрибута.

Более подробно о фундаментальных свойствах отношений можно прочитать в ["Введении в СУБД"](http://www.osp.ru/dbms/1995/03/source/dbms_in.html#part_4_2) С.Д.Кузнецова.

Реляционная модель данных

Определения:

Декартово произведение: Для заданных конечных множеств (не обязательно различных) декартовым произведением называется множество произведений вида: , где

Пример: если даны два множества A и B , их декартово произведение будет иметь вид С=A\*B

Отношение: Отношением R, определенным на множествах называется подмножество декартова произведения . При этом:

множества называются доменами отношения

элементы декартова произведения называются кортежами

число n определяет степень отношения ( n=1 - унарное, n=2 - бинарное, ..., n-арное)

количество кортежей называется мощностью отношения

Пример: на множестве С из предыдущего примера могут быть определены отношения R1 (a1\*b1, a3\*b2) или R2 (a1\*b1, a2\*b1, a1\*b2)



Общая характеристика реляционной модели данных

Реляционная модель состоит из трёх частей:

1) структурная часть

* говорится, что единственной структурой данных, используемой в РелБД, является нормализованное n-арное (7-арное например) отношение.

2) манипуляционная часть

* утверждается два фундаментальных механизма манипулирования РелБД:

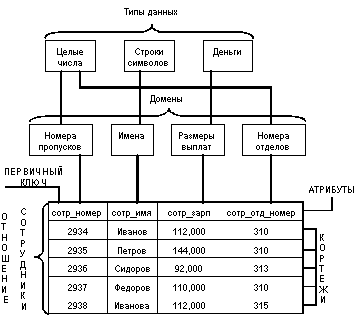
1. реляционная алгерба - в основе классическая теория множеств
2. реляционное счисление - в основе классический логический аппарат исчисления предикатов 1-го порядка

* основная функция - обеспечение меры реляционности любого конкретного языка реляционных баз данных: язык называется реляционным, если он обладает не меньшей выразительностью и мощностью, чем реляционная алгебра или реляционное исчисление.

3) целостная часть

* есть два требования целостности, которые должны поддерживаться в любой реляционной СУБД:

1. целостность сущностей - любой кортеж любого отношения отличим от любого другого кортежа этого отношения. Решается этот вопрос введением первичного ключа
2. целостность по ссылкам - при соблюдении нормализованности отношений сложной сущности реального мира представляются в реляционной базе данных в виде нескольких кортежей нескольких отношений



Реляционная модель данных включает следующие компоненты:

* Структурный аспект (составляющая) — данные в базе данных представляют собой набор отношений.
* Аспект (составляющая) целостности — отношения отвечают определённым условиям целостности. Реляционная модель данных поддерживает декларативные ограничения целостности уровня домена (типа данных), уровня отношения и уровня базы данных.
* Аспект (составляющая) обработки (манипулирования) — реляционная модель данных поддерживает операторы манипулирования отношениями (реляционная алгебра, реляционное исчисление).

# 7.Базисные средства манипулирования реляционными данными. Реляционная алгебра. Реляционные исчисления.

Реляционная алгебра основные алгебраические операции:

* теоретико-множественные операции -
  + **объединение** отношений

Отношения-операнды в этом случае должны быть определены по одной схеме. Результирующее отношение содержит все строки операндов за исключением повторяющихся.

* + **пересечение** отношений

На входе операции два отношения, определенные по одной схеме. На выходе - отношение, содержащие кортежи, которые присутствуют в обоих исходных отношениях.

* + **разность** отношений

Операция во многом похожая на ПЕРЕСЕЧЕНИЕ, за исключением того, что в результирующем отношении содержатся кортежи, присутствующие в первом и отсутствующие во втором исходных отношениях.

* + **декартово произведение** отношений

Входные отношения могут быть определены по разным схемам. Схема результирующего отношения включает все атрибуты исходных. Кроме того:

* степень результирующего отношения равна сумме степеней исходных отношений
* мощность результирующего отношения равна произведению мощностей исходных отношений.
* Специальные реляционные операции (только в БД)
  + **ограничение** отношений (выборка, горизонтальное подмножество)

На входе используется одно отношение, результат - новое отношение, построенное по той же схеме, содержащее подмножество кортежей исходного отношения, удовлетворяющих условию выборки.

* + **проекция** отношения (вертикальное подмножество)

Операция проекции представляет из себя выборку из каждого кортежа отношения значений атрибутов, входящих в список A, и удаление из полученного отношения повторяющихся строк.

* + **соединение** отношений

Данная операция имеет сходство с ДЕКАРТОВЫМ ПРОИЗВЕДЕНИЕМ. Однако, здесь добавлено условие, согласно которому вместо полного произведения всех строк в результирующее отношение включаются только строки, удовлетворяющие опредленному соотношению между атрибутами соединения (А1,A2) соответствующих отношений.

* + **деление** отношений

Пусть отношение R , называемое делимым, содержит атрибуты (A1,A2,...,An). Отношение S - делитель содержит подмножество атрибутов A: (A1,A2,...,Ak) (k<n). Результирующее отношение C определено на атрибутах отношения R, которых нет в S, т.е. Ak+1,Ak+2,...,An. Кортежи включаются в результирующее отношение C только в том случае, если его декартово произведение с отношением S содержится в делимом R.

Термины

|  |  |  |
| --- | --- | --- |
| **Термины эл. таблицы** | **Термины реляционных данных** | **Термины реляционных СУБД** |
| Таблица | Отношения | Таблица БД |
| Столбец / Колонка | Атрибут | Поле |
| Строка | Кортеж | Запись |
| "Шапка" | Схема-отношение | Тип данных |

Реляционное исчисление.

Два базовых механизма манипулирования реляционными данными - основанная на теории множеств реляционная алгебра и базирующееся на математической логике (точнее, на исчислении предикатов первого порядка) реляционное исчисление. В свою очередь, обычно рассматриваются два вида реляционного исчисления - исчисление доменов и исчисление предикатов.

Все эти механизмы обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатом вычисления также являются отношения. Механизмы реляционной алгебры и реляционного исчисления эквивалентны - для проверки степени реляционности некоторого языка БД можно пользоваться любым из этих механизмов.

Выражения реляционной алгебры строятся на основе алгебраических операций, она также имеет процедурную интерпретацию. Для формулы реляционного исчисления однозначная интерпретация, вообще говоря, отсутствует. Формула только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения. Поэтому языки реляционного исчисления являются более непроцедурными или декларативными.

Реляционное исчисление. В реляционном исчислении следует правильно указать параметры конечного результата. Выбор пути вычисления (последовательность элементарных операций) возлагается на СУБД. Можно сказать, что реляционная алгебра дает четкие предписания, а реляционное исчисление описывает нужный результат.

Виды исчисления:

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

Тип (кортежа): имена и типы атрибутов; набор значений.

Синтаксис: переменная :: отношение [where <переменная>=’значение’] (S :: Students)/

Условия:

1) простые условия (S.id < 5);

составные (∧, ∨, ¬) (S where id=5 ∨ id=6);

3) с кванторами (∀, ∃) (S where ∃S (S,id=5 ∧ S.id=6)).

1. Исчисление доменов. В исчислении доменов областями определения переменных являются домены, на которых определены атрибуты отношений базы данных, т.е. допустимым значением каждой переменной является значение некоторого домена.

Тип (значения): имена и типы атрибутов; набор значений.

Синтаксис: переменная :: тип (Sid :: int).

Условие принадлежности: Отношение {Атр1 = Знач1, ...}

# 8. Проектирование реляционных БД. Нормализация.

При проектировании баз данных решается две проблемы:

- как отобразить объекты предметной области в абстрактные модели наших данных? (логическое проектирование БД).

- как обеспечить эффективность выполнения запросов к БД, т.е. как, имея ввиду особенности конкретной СУБД, расположить данные в ВЗУ, создание каких дополнительных структур потребовать? (физическое проектирование баз данных).

Реляционная БД содержит как структурную, так и семантическую информацию. Структура базы данных определяется числом и видом включенных в нее отношений, и связями типа "один ко многим", существующими между кортежами этих отношений. Семантическая часть описывает множество функциональных зависимостей, существующих между атрибутами этих отношений. Если даны два атрибута X и Y некоторого отношения, то говорят, что Y функционально зависит от X, если в любой момент времени каждому значению X соответствует ровно одно значение Y.

Функциональная зависимость обозначается X -> Y. X и Y могут представлять собой не только единичные атрибуты, но и группы, составленные из неск. атрибутов одного отнош. Можно сказать, что функциональные зависимости представляют собой связи типа "один ко многим", существующие внутри отнош. Некоторые функ. завис. могут быть нежелательны.

Избыточная функциональная зависимость - зависимость, заключающая в себе такую информацию, которая может быть получена на основе других зависимостей, имеющихся в базе данных. Корректной считается такая схема БД, в которой отсутствуют избыточные функциональные зависимости. В противном случае приходится прибегать к процедуре декомпозиции (разложения) имеющегося множества отношений. При этом порождаемое множество содержит большее число отношений, которые являются проекциями отношений исходного множества. Обратимый пошаговый процесс замены данной совокупности отношений другой схемой с устранением избыточных функциональных зависимостей называется нормализацией.

Нормализация – это декомпозиция отношения находящегося в предыдущей нормальной форме, в 2 или более отношения, удовлетворяющих следующей нормальной форме.

Цель нормализации: экономия памяти, устранение избыточности, исключение противоречия, упрощение управления.

Проблемы ненормализованных таблиц: избыточность данных (повтор инф.), аномалии обновления (в разных местах, разные данные), аномалии удаления, аномалии ввода.

Эля устранения этих проблем используется нормализация.

Основное понятие нормализации: Нормальная Форма.

Основные свойства НФ: 1) каждая следующая НФ в некотором смысле лучше предыдущей, 2) при переходе к следующей НФ свойства предыдущих сохраняются.

Процесс проектирования - процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая. Каждой нормальной форме соответствует некоторый определенный набор ограничений.

Существуют такие нормальные формы (НФ):

1) Первая нормальная форма (1НФ) - значения атрибутов отношения должны быть атомарны. Отношение находится в первой НФ.

2) Вторая нормальная форма (2НФ) - отношение находится во 2НФ в том и только в том случае, когда отношение находится в первой НФ и каждый неключевой атрибут полностью зависит от первичного ключа.

3) Третья нормальная форма (3НФ) - отношение находится в 3НФ в том и только том случае, когда отношение находится во второй НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

4) Нормальная форма Бойса-Кодда (БКНФ) - отношение находится в BCNF, если оно находится во 3НФ и в ней отсутствуют зависимости атрибутов первичного ключа от неключевых атрибутов.

5) Четвертая нормальная форма (4НФ) - отношение находится в 4NF если оно находится в BCNF и в нем отстутсвуют многозначные зависимости, не являющиеся функциональными зависимостями.

6) Пятая нормальная форма (5НФ) - отношение находится в 5НФ тогда и только тогда, когда любая зависимость по соединению в нем определяется только его возможными ключами.

Для перехода к 2НФ:

1. Определить на какие части можно разбить первичный ключа так, чтобы некоторые из не ключевых полей зависели от одной их этих частей причем эти части могут содержать несколько атрибутов.
2. Создать новую таблицу для каждой части такого ключа и группы зависящих от нее полей и переместить их в эту таблицу.
3. Удалить из исходной таблицы поля, перемещенные в другую таблицу, кроме тех которые станут внешним ключом.

Переход к 3НФ:

1. Определить все поля, от которых зависят другие.
2. Создать новую таблицу для каждого такого поля или группы зависящих от него полей и переместить их в эту таблицу. Поле, от которого зависит все остальные поля станет первичным ключом новой таблицы.
3. Удалить перемещенные поля их исходной, оставив лишь те их них, которые станут внешними ключами.

Нормализацию часто завершают на третьей нормальной форме.

Денормализация - процесс приведения БД к предыдущей НФ с целью увеличения производительности запросов.

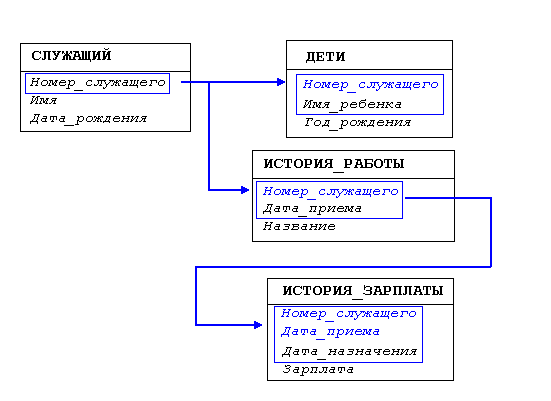
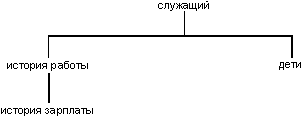


Рис. Исходное отношение Рис. Нормализованное множество отношений

# 9.Основные понятия модели Сущность-связь. ER – диаграммы.

Реляционные БД достаточны для моделирования большинства предметных областей, но при реальном проектировании обнаруживается их ограниченность; в модели нет достаточных средств для отражения смысла данных.

Проектирование начинается с выделения сущностей и связей между ними, а модель не предлагает аппарат для разделения сущностей и связей. Возникает необходимость использовать другую модель.

Модель была предложена Питером Ченом (Peter Pin-Shan Chen) в 1976 г. ER-диаграмма отображает отношения набора сущностей, хранящиеся в базе данных. Другими словами, можно сказать, что ER-диаграммы объясняют логическую структуру баз данных.

Основными понятиями ER-модели являются сущность, связь и атрибут.

Сущность – реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности — это имя типа, а не некоторого конкретного экземпляра этого типа.

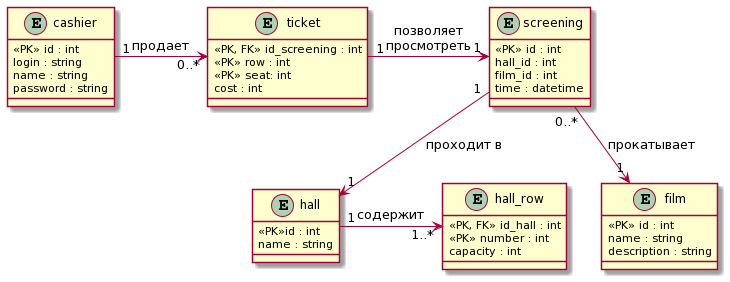


Связь — это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной. В любой связи выделяются два конца, на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (любой ли экземпляр данной сущности должен участвовать в данной связи).

Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой. При этом в месте "стыковки" связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много (many) экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный - прерывистой линией.



Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.





|  |  |
| --- | --- |
| **Обозначение** | **Значение** |
|  | **Набор независимых сущностей** |
|  | **Набор зависимых сущностей** |
|  | **Атрибут** |
|  | **Ключевой атрибут** |
|  | **Набор связей** |

Нормальные формы:

В 1НФ ER-схемы устраняются повторяющиеся атрибуты или группы атрибутов.

Во 2НФ устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

В 3НФ устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

Получение реляционной схемы из ER-схемы

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем;

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

-все подтипы в одной таблице (а)

-для каждого подтипа - отдельная таблица (б)

Шаг 7. Имеется два способа работы при наличии исключающих связей:

-общий домен (создаются два столбца: идентификатор связи и идентификатор сущности).

-явные внешние ключи (для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения).

# 10. Система System R. Основные цели System R.

System R разрабатывалась в исследовательской лаборатории фирмы IBM в 1975-1979 г.г. System R практически доказала жизнеспособность реляционного подхода к управлению базами данных.

Основными целями разработчиков System R являлись следующие:

-обеспечить ненавигационный интерфейс высокого уровня пользователя с системой;

-Обеспечить многообразие допустимых способов использования СУБД, включая программируемые транзакции, диалоговые транзакции и генерацию отчетов;

-поддерживать динамически изменяемую среду баз данных, в которой отношения, индексы, представления, транзакции и другие объекты могут легко добавляться и уничтожаться без приостановки нормального функционирования системы;

-обеспечить возможность параллельной работы с одной базой данных многих пользователей с допущением параллельной модификации объектов базы данных при наличии необходимых средств защиты целостности базы данных;

-обеспечить средства восстановления согласованного состояния баз данных после разного рода сбоев аппаратуры или программного обеспечения;

-обеспечить гибкий механизм, позволяющий определять различные представления хранимых данных и ограничивать этими представлениями доступ пользователей к базе данных по выборке и модификации на основе механизма авторизации;

-обеспечить производительность системы при выполнении упомянутых функций, сопоставимую с производительностью существующих СУБД низкого уровня.

Основой System R является реляционный язык SQL. Средствами SQL определяются все доступные пользователю объекты баз данных: таблицы, индексы, представления.

В System R под целостным состоянием базы данных понимается состояние, удовлетворяющее набору сохраняемых при базе данных предикатов целостности. Эти предикаты, называемые в System R условиями целостности (assertions), задаются также средствами языка SQL.

В System R - журнал, в который помещаются записи обо всех меняющих состояние базы данных операциях всех транзакций.

В языке SQL имеется средство определения так называемых условных воздействий (triggers), позволяющих автоматически поддерживать целостность базы данных при модификациях ее объектов. Условное воздействие - это каталогизированная операция модификации, для которой задано условие ее автоматического выполнения.

Язык SQL содержит средства определения представлений. Представление - это запомненный именованный запрос на выборку данных (из одной или нескольких таблиц).

Авторизация доступа к базе данных основана также на средствах SQL. При создании любого объекта базы данных выполняющий эту операцию пользователь становится полновластным владельцем этого объекта, т.е. может выполнять по отношению к этому объекту любую функцию из предопределенного набора.

Что касается естественных требований к эффективности системы, то здесь основные решения связаны со спецификой физической организации баз данных на внешней памяти, буферизацией используемых страниц базы данных в оперативной памяти и развитой техникой оптимизации запросов, сформулированных на SQL, производимой на стадии их компиляции.

Структурная организация System R вполне согласуется с поставленными при ее разработке целями и выбранными решениями. Основными структурными компонентами System R являются система управления реляционной памятью (Relational Storage System - RSS) и компилятор запросов языка SQL. RSS обеспечивает интерфейс довольно низкого, но достаточного для реализации SQL, уровня для доступа к хранимым в базе данным. Синхронизация транзакций, журнализация изменений и восстановление баз данных после сбоев также относятся к числу функций RSS.

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

Разновидности объектов во внешней памяти базы данных:

строки отношений – основная часть базы данных, большей частью непосредственно видимая пользователям;

управляющие структуры - индексы, создаваемые по инициативе пользователя (админа) или верхнего уровня системы из соображений повышения эффективности выполнения запросов и обычно автоматически поддерживаемые нижним уровнем системы;

журнальная информация, поддерживаемая для удовлетворения потребности в надежном хранении данных;

служебная информация, поддерживаемая для удовлетворения внутри потребностей нижнего уровня системы (например, информация о свободной памяти).

Существуют два принципиальных подхода к физическому хранению отношений:

1) покортежное хранение отношений (кортеж является единицей физического хранения). Естественно, это обеспечивает быстрый доступ к целому кортежу, но при этом во внешней памяти дублируются общие значения разных кортежей одного отношения

2) альтернативным (менее распространённым) подходом является хранение отношения по столбцам, т.е. единицей хранения является столбец отношения с исключенными дубликатами. При такой организации суммарно в среднем тратится меньше внешней памяти, поскольку дубликаты значений не хранятся; за один обмен с внешней памятью в общем случае считается больше полезной информации. Дополнительным преимуществом является возможность использования значений столбца отношения для оптимизации выполнения операций соединения. Но при этом требуются существенные дополнительные действия для сборки целого кортежа (или его части).

Подходы к организации индексов:

1. В-деревья

С точки зрения внешнего логического представления B-дерево — это сбалансированное сильно ветвистое дерево во внешней памяти. Сбалансированность означает, что длина пути от корня дерева к любому его листу одна и та же. Ветвистость дерева — это свойство каждого узла дерева ссылаться на большое число узлов-потомков. С точки зрения физической организации B-дерево представляется как мульти песочная структура страниц внешней памяти, т.е. каждому узлу дерева соответствует блок внешней памяти (страница). Внутренние и листовые страницы обычно имеют разную структуру. Поиск в B-дереве - это прохождение от корня к листу в соответствии с заданным значением ключа. Заметим, что поскольку деревья сильно ветвистые и сбалансированные, то для выполнения поиска по любому значению ключа потребуется одно и то же (и обычно небольшое) число обменов с внешней памятью. Основной "изюминкой" B-деревьев является автоматическое поддержание свойства сбалансированности.

2. хэширование

Общая идея - применение к значению ключа некоторой функции свертки (хэш-функции), вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи. Свертка ключа используется как адрес в таблице, содержащей ключи и записи. Основным требованием к хэш-функции является равномерное распределение значение свертки.

# 12. Транзакции и целостность баз данных. Сериализация транзакций.

Транзакция – неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), что, либо результаты всех операторов, входящих в транзакцию, отображаются в БД, либо воздействие всех этих операторов полностью отсутствует. Транзакция – единица активности пользователя. Для многопользовательских систем пока первый пользователь обращается к БД, другому доступ закрыт. Девиз «Всё или ничего»: при завершении операций оператором «COMMIT» результат фиксируется в ВЗУ, «ROLLBACK» – всё отменяется.

Бывают ситуации, при которых целостность БД невозможно не нарушить, выполняя только один оператор изменения БД. Поэтому для поддержания целостности допускается ее нарушение внутри транзакции с условием, чтобы к завершению транзакции целостность была соблюдена. Несоблюдение этого условия приводит к тому, что вместо фиксации результатов транзакции происходит её откат (вместо оператора COMMIT выполняется ROLLBACK), и БД остается в таком состоянии, в котором находилась к моменту начала транзакции, то есть в целостном состоянии.

Различают два вида ограничений целостности: немедленно проверяемые и откладываемые.

НПОЦ – такие ограничения, проверку которых бессмысленно или невозможно откладывать. Они реализуются отдельным операторами языкового уровня СУБД. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор.

ООЦ – ограничения на БД, а не на отдельные операции. Эти ограничения проверяются в конце транзакции, и их нарушение вызывает автоматическую замену оператора COMMIT на оператор ROLLBACK. В некоторых системах существует оператор насильственной проверки ограничения целостности внутри транзакции.

Чтобы добиться изолированности, в СУБД есть методы совместного выполнения.

Сериализация – механизм выполнения транзакций по некоему сериальному плану. Результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций. Методы сериализации основываются на трёх конфликтах:

* W-W: Т2 пытается изменить объект, измененный Т1.
* R-W: Т2 пытается изменить объект, прочитанный Т1.
* W-R: Т2 пытается прочитать объект, измененный Т1.

Методы сериализации:

Метод синхронизационных захватов (СЗ) объектов БД.

Метод временных меток.

Перед выполнением операции от имени транзакции запрашивается синхронизационный захват в каком-то из режимов:

Совместный режим S (Share) - это разделяемый захват, требуется для чтения/

Монопольный режим X (eXclusive) - для операций занесения данных, удаления, модификацией и не только. Захваты по чтению совместимы; захваты по изменению несовместимы с захватами по чтению; захваты по изменению несовместимы.

Что считать объектом для захвата:

Файл - физический объект, где хранятся несколько отношений.

Отношение - логический объект с множеством кортежей.

Кортеж - элементарный физический объект базы данных.

Страница данных - физический объект, который хранит кортежи одного или нескольких отношений, индексы, служебную информацию.

Операция над кортежем - это операция над страницей, где он находится, и над отношением, и над файлом. Чем крупнее объект, тем меньше СЗ для него будет подерживаться и меньше расходов, но возрастает вероятность конфликтов между транзакциями. Сегодня реализованы кортежные захваты. Есть гранулированные СЗ.

Выполнение СЗ иногда приводит к тупикам между транзакциями.

Пример: Т1 и Т2 устанавливают монопольный захват объектов А1 и А2 соответственно. После этого Т1 требует совместный захват А2, а Т2 - совместный захват А1. Ни одна из транзакций не продолжится: Монопольные захваты не будут сняты, а совместные не будут удовлетворены.

Обнаруживают тупики через граф ожидания транзакции.

По этому графу обнаруживается цикл, где и выбирается транзакция-жертва, и её откатывают.

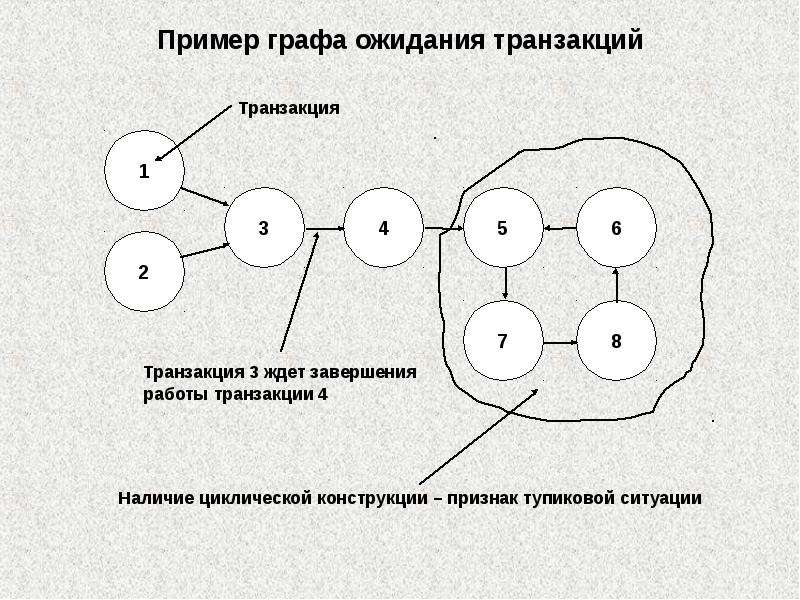


Рис. . Граф ожидания транзакции

# 13. Изолированность пользователей.

Транзакция – неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), что, либо результаты всех операторов, входящих в транзакцию, отображаются в БД, либо воздействие всех этих операторов полностью отсутствует. Транзакция – единица активности пользователя. Для многопользовательских систем пока первый пользователь обращается к БД, другому доступ закрыт. Девиз «Всё или ничего»: при завершении операций оператором «COMMIT» результат фиксируется в ВЗУ, «ROLLBACK» – всё отменяется.

С одной БД могут одновременно работать несколько пользователей или прикладных программ. Система должна обеспечивать изолированность пользователей, то есть создание иллюзии работы с БД в одиночку. Тогда транзакция выступает единицей изолированности пользователей.

Бывают ситуации, при которых целостность БД невозможно не нарушить, выполняя только один оператор изменения БД. Поэтому для поддержания целостности допускается ее нарушение внутри транзакции с условием, чтобы к завершению транзакции целостность была соблюдена. Несоблюдение этого условия приводит к тому, что вместо фиксации результатов транзакции происходит её откат (вместо оператора COMMIT выполняется ROLLBACK), и БД остается в таком состоянии, в котором находилась к моменту начала транзакции, то есть в целостном состоянии.

Различают два ограничения целостности: немедленно проверяемые и откладываемые.

Немедленно проверяемые ограничения целостности – такие ограничения, проверку которых бессмысленно или невозможно откладывать. Пример: возраст сотрудника - 150 лет. Они реализуются отдельным операторами языкового уровня СУБД. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор.

Откладываемые ограничения целостности – ограничения на БД, а не на отдельные операции. Эти ограничения проверяются в конце транзакции, и их нарушение вызывает автоматическую замену оператора COMMIT на ROLLBACK. В некоторых системах существует оператор насильственной проверки ограничения целостности внутри ТР.

Четыре уровня изолированности:

* Отсутствие потерянных изменений. Т1 (первая транзакция) изменяет объект А. До завершения Т1 вторая транзакция Т2 также пытается изменить объект А. Т2 завершается операцией ROLLBACK. Тогда при повторном чтении объекта Т1 не видит изменений объекта. Это противоречит изолированности пользователей.

Проблема: ситуация потерянных изменений.

Решение: до завершения Т1 транзакции, никакая другая транзакция не может изменить объект.

Профилактика: отсутствие потерянных изменений – минимальное требование к СУБД при синхронизации параллельно выполняемых транзакций.

* Отсутствие чтения «грязных данных». Т1 изменяет объект, параллельно с этим Т2 читает объект A. Так как операция изменения не завершена, Т2 видит несогласованные данные.

Проблема: несоответствие требованию изолированности пользователей, где каждый пользователь начинает свою транзакцию при согласованном состоянии базы данных и вправе видеть согласованные данные.

Решение: до согласования транзакции объекта Т1, другая транзакция не должна читать объект, где минимальным требованием является блокировка чтения объекта А до завершения операции его изменения в Т1.

* Отсутствие неповторяющихся чтений. Т1 читает объект БД А. До завершения первой транзакции Т2 изменяет объект А и успешно завершается COMMIT. Т1 повторно читает объект А и видит его изменённое состояние.

Проблема: противоречие идеи изолированности.

Решение: до завершения первой транзакции никакая другая транзакция не должна изменять объект

* Проблема кортежей-фантомов. Т1 выполняет оператор А выборки кортежа отношения R («Relationship») c условием выборки S («Share»). До завершения первой транзакции Т2 вставляет в R новый кортеж, удовлетворяющий S, и совершает COMMIT. Т1 повторно также выполняет этот оператор и в результате видит кортеж, которого не было.

Проблема: противоречие идеи изолированности.

Решение: требуется более высокий уровень синхронизации транзакции.

Рис. ХХХ. Уровни изоляции

# 14. Журнализация изменений БД. Журнализация и буферизация.

Журнализация необходима для восстановления БД в случае сбоев. Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя.

Журнализация изменений базы данных (БД) — это процесс записи всех изменений, вносимых в данные, в специальный журнал (лог), который позволяет восстановить состояние БД в случае сбоя или ошибки. Журнал содержит информацию о всех операциях, таких как вставка, обновление и удаление данных, а также изменения структуры БД.

Буферизация — данные временно хранятся в буфере (памяти), прежде чем быть записанными на диск.

Существует два вида аппаратных сбоев:

• мягкие сбои или внезапная остановка работы компьютера (например, аварийное выключение питания).

• жёсткие сбои, характеризуемые потерей информации на носителях внешней памяти.

Возможны следующие ситуации, при которых требуется производить восстановление состояния базы данных:

* Индивидуальный откат транзакции: 1) завершение оператором ROLLBACK; 2) откат транзакции инициируется системой; 3) исключительной ситуации в прикладной программе (например, деление на ноль); 4) выбор транзакции в качестве жертвы при обнаружении синхронизационного тупика.
* Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой): 1) выключении электрического питания; 2) возникновении неустранимого сбоя процессора.
* Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой).

Основой восстановления является архивная копия и журнал изменений базы данных. Возможны два основных варианта ведения журнальной информации:

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

Журнализация и буферизация.

Основным принципом согласованной политики записи изменений в журнал и непосредственно в базу данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется Write Ahead Log (WAL) – «пиши сначала в журнал», и состоит в том, что если требуется сохранить во внешней памяти измененный объект базы данных, то перед этим нужно гарантировать сохранение во внешней памяти журнала записи о его изменении.

Проблема не возникает при индивидуальных откатах транзакций, поскольку в этих случаях содержимое оперативной памяти не утрачено и можно пользоваться содержимым как буфера журнала, так и буферов страниц базы данных. Но если произошел мягкий сбой, и содержимое буферов утрачено, для проведения восстановления базы данных необходимо иметь некоторое согласованное состояние журнала и базы данных во внешней памяти.

В журнале отмечаются точки физической согласованности базы данных - моменты времени, в которые во внешней памяти содержатся согласованные результаты операций, завершившихся до соответствующего момента времени, и отсутствуют результаты операций, которые не завершились, а буфер журнала вытолкнут во внешнюю память.

Общими принципами восстановления являются следующие:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;

- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

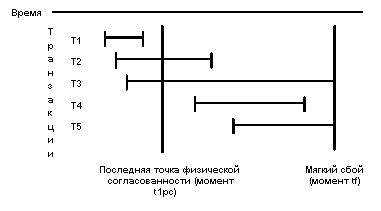


Рис. . Восстановление после магкого сбоя

Т1: Ничего не надо делать, т.к. закончилась до ТФС, все результаты отображены в ВЗУ.

Т2: Необходимо повторно выполнить оставшуюся часть транзакции (redo).

Т3: Выполнить в обратном направлении первую часть операции (undo): во внешней памяти нет результатов Т3, которые выполнены после ТФС, но есть результаты до ТФС.

Т4: Необходимо выполнить полную повторную прямую интерпретацию команд этой транзакции (redo).

Т5: Ничего не поможет, необходимо заново выполнять все команды транзакции.

# 15. Восстановление баз данных в различных ситуациях.

* Индивидуальный откат транзакции: 1) выбирается очередная запись из списка данной транзакции; 2) выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE; 3) любая из этих обратных операций также журнализуются; 4) при успешном завершении отката в журнал заносится запись о конце транзакции.
* Восстановление после мягкого сбоя. 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 журнале отмечаются точки физической согласованности базы данных - моменты времени, в которые во внешней памяти содержатся согласованные результаты операций, завершившихся до соответствующего момента времени, и отсутствуют результаты операций, которые не завершились, а буфер журнала вытолкнут во внешнюю память - точки tpc (time of physical consistency).

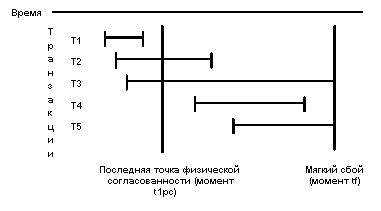


Рис. . Восстановление после магкого сбоя

Т1: Ничего не надо делать, т.к. закончилась до ТФС, все результаты отображены в ВЗУ.

Т2: Необходимо повторно выполнить оставшуюся часть транзакции (redo).

Т3: Выполнить в обратном направлении первую часть операции (undo): во внешней памяти нет результатов Т3, которые выполнены после ТФС, но есть результаты до ТФС.

Т4: Необходимо выполнить полную повторную прямую интерпретацию команд этой транзакции (redo).

Т5: Ничего не поможет, необходимо заново выполнять все команды транзакции.

Чтобы восстановить состояние базы данных в момент tpc используются два основных подхода: подход, основанный на использовании теневого механизма, и подход, в котором применяется журнализация постраничных изменений базы данных.

Если во время работы с открытым файлом происходит сбой, во внешней памяти автоматически сохраняется состояние файла до его открытия. Для явного восстановления файла достаточно повторно считать в оперативную память теневую таблицу отображения.

Восстановление к tpc происходит мгновенно: текущая таблица отображения заменяется на теневую. В пределе может потребоваться вдвое больше внешней памяти, чем реально нужно для хранения базы данных. Теневой механизм — это надежное, но слишком грубое средство. Обеспечивается согласованное состояние внешней памяти в один общий для всех объектов момент времени. На самом деле достаточно иметь совокупность согласованных наборов страниц, каждому из которых может соответствовать свои временные отсчеты.

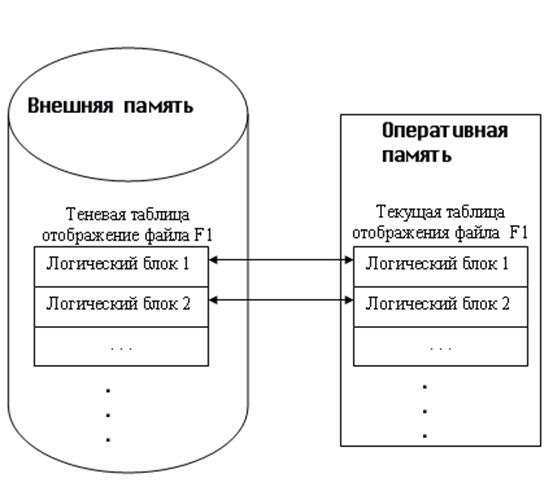


Рис. . Восстановление с помощью теневой таблицы

* Восстановление после жесткого сбоя. Для восстановления последнего согласованного состояния БД 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: 1) по журналу в прямом направлении выполняются все операции; 2) для транзакций, которые не закончились к моменту сбоя, выполняется откат.

На самом деле, поскольку жесткий сбой не сопровождается утратой буферов оперативной памяти, можно восстановить базу данных до такого уровня, чтобы можно было продолжить даже выполнение не закончившихся транзакций. Но обычно это не делается, потому что восстановление после жесткого сбоя — это достаточно длительный процесс.

Хотя к ведению журнала предъявляются особые требования по части надежности, в принципе возможна и его утрата. Тогда единственным способом восстановления базы данных является возврат к архивной копии. Конечно, в этом случае не удастся получить последнее согласованное состояние базы данных, но это лучше, чем ничего.

# 16. Язык SQL, функции и основные возможности. Стандартизация SQL.

1. Язык SQL - язык непроцедурный, подразумевает постановку цели, так как не предоставляет средств для построения циклов, ветвлений и прочее. SQL — это декларативный язык программирования. В нём человек формулирует запрос на извлечение или модификацию данных, а алгоритм его выполнения почти полностью ложится на плечи конкретной системы управления базами данных (СУБД).

SQL (Structured Query Language) - язык структурированных запросов **= DDL + DML + DQL**  
DDL (Data Definition Language) - язык описания данных  
DML (Data Manipulation Language) - язык манипуляции данными  
DQL (Data Query Language) - язык запросов данных

SQL появился в середине 70-х г. в рамках "System R". Уже тогда был полным языком БД.

2. Стандартизация SQL

Цель - сделать едиными требования для участников процесса развития языка.

* SQL-86, SQL-87, Первый вариант стандарта, принятый институтом ANSI и одобренный ISO в 1987 году.
* SQL1 (SQL89) - появился в 1989 году, доработанный вариант предыдущего стандарта.
* SQL2 (SQL92) - 1992 год, Значительные изменения (ISO 9075); уровень Entry Level стандарта SQL-92 был принят как стандарт FIPS 127-2.
* SQL3 - 1999 год, Добавлена поддержка регулярных выражений, рекурсивных запросов, поддержка триггеров, базовые процедурные расширения, нескалярные типы данных и некоторые объектно-ориентированные возможности.
* SQL:2003 - Введены расширения для работы с XML-данными, оконные функции (применяемые для работы с OLAP-базами данных), генераторы последовательностей и основанные на них типы данных.
* SQL:2006 - Функциональность работы с XML-данными значительно расширена. Появилась возможность совместно использовать в запросах SQL и XQuery.
* SQL:2008 - Улучшены возможности оконных функций, устранены некоторые неоднозначности стандарта SQL:2003
* SQL:2011 - Реализована поддержка хронологических БД (PERIOD FOR), поддержка конструкции FETCH.
* SQL:2016 - Защита на уровне строк, полиморфные табличные функции, JSON.
* SQL:2023 - Операции над графами. Агрегатная функция ANY\_VALUE(). Поддержка шестнадцатеричных/двоичных/восьмеричных литералов, улучшение поддержки JSON.

3. SQL в прикладном программировании

В SQL89 определены основные способы взаимодействия БД и прикладных программ, написанных на традиционных ЯПВУ. Fortran, PL\_1, Pascal etc.

3.1. Способы взаимодействия

1) модульный SQL - Операторы собраны в модуль, используются как процедуры. Прикладная программа вызывает процедуру модуля, передает ей фактические параметры. Модуль компилируется отдельно от прикладной программы, при этом порождается набор хранимых процедур.

2) Встроенный SQL - непосредственно на ЯП имеется SQL. Обмен производится через переменные. Операторы компилируется каждый раз при их использовании, точнее, при каждом первом использовании оператора при данном запуске программы.

* Плюсы модульной организации: 1) Прикладная программа имеет меньший размер, 2) Взаимодействие с СУБД локализованы.
* Недостатки модульной организации: 1) Для понимания смысла прикладной программы требуется чтение двух текстов, 2) Синтаксис модулей бывает разным
* Достоинства встроенной организации: 1) Прикладная программа более самосодержащаяся, 2) Стандарт встраивания единообразен.
* Недостатки встроенной организации: 1) Синтаксис SQL операторов контрастирует с синтаксисом базового языка.

# 17. Типы данных языка SQL. Средства определения схемы.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Ключевое слово | Размер | Диапозон |
| целые | int (integer) | 4 байта | от -231 до 231 |
| smallint | 2 байта | от -32768 до 32767 |
| tinyint | 1 байт | от 0 до 255 |
| упакованные десятичные (точные числовые значения) | numeric (p, s)  (decimal) | в зависимости от заданной точности p которая может быть до 38 знаков требуется от 2 до 17 байт | В зависимости от точности (до 38 знаков) |
| числа с плавающей точкой (приближенные числовые значения | float (float(n)) | 8 байт | Точность 15 знаков, n принимает значения от 8 до 15 |
| real (float(n)) | 4 байта | точность 7 знаков, n принимает значения от 1 до 7 |
| символьный фиксированной длины | char[(n)]  (character[(n)]) | занимает n байт даже если строка короче | n принимает значения от 0 до 255, если n не указать, то размерность - 1 |
| символьный переменной длины | varchar (n)  (character varying (n)) | по 1 байту на каждый символ, то есть в соответствии с фактической длиной строки | n принимает значения от 1 до 255 |
| дата-время | datetime | 8 байт | от 01.01.1753 до 31.01.9999 |
| smalldatetime | 4 байта | от 01.01.1900 до 06.06.2079 |
| денежные суммы | money | 8 байт | 4 знака после запятой  от -922 трлн. до +922 трлн. |
| smallmoney | 4 байта | 4 знака после запятой  от -214 тыс. до +214 тыс. |
| двоичный фиксированной длины | binary (n) | n байт, даже если больше длина | любые двоичные данные до 255 байт |
| двоичные числа переменной длины | varbinary (n) | фактическое число хранящихся байт | любые двоичные данные до 255 байт |
| длинный текст | text (image) | до 2 Гб | text - для хранения символьных данных, image - для хранения двоичных данных |
| булевые значения | bit | поле размещается в 1 байте вместе с другими полями bit той же таблицы | 0 или 1 |

DDL - язык определения структур и ограничений целостности баз данных. DDL – Data Definition Language (язык описания данных). Сюда относятся команды создания и удаления баз данных; создания, изменения и удаления таблиц; управления пользователями и т.д.

**Оператор определения схемы**

Операторы базы данных:

Создание таблицы:

CREATE DATABASE <имя\_базы\_данных>

Удаление таблицы:

DROP DATABASE <имя\_базы\_данных>

Создание и удаление таблиц

Создание таблицы:

CREATE TABLE <имя\_таблицы>

(<имя\_столбца> <тип\_столбца>

[NOT NULL]

[UNIQUE | PRIMARY KEY]

[REFERENCES <имя\_мастер\_таблицы> [<имя\_столбца>]

, ...)

* NOT NULL - в этом случае элементы столбца всегда должны иметь определенное значение (не NULL)
* один из взаимоисключающих параметров UNIQUE - значение каждого элемента столбца должно быть уникальным или PRIMARY KEY - столбец является первичным ключом.
* REFERNECES <имя\_мастер\_таблицы> [<имя\_столбца>] - эта конструкция определяет, что данный столбец является внешним ключом, и указывает, на ключ какой мастер\_таблицы он ссылается.

Контроль за выполнением указанных условий осуществляет СУБД.

Удаление таблицы:

DROP TABLE <имя\_таблицы>

Модификация таблицы

Добавить столбцы:

ALTER TABLE <имя\_таблицы> ADD   
 (<имя\_столбца> <тип\_столбца>   
 [NOT NULL]   
 [UNIQUE | PRIMARY KEY]   
 [REFERENCES <имя\_мастер\_таблицы> [<имя\_столбца>]]   
 ,...)

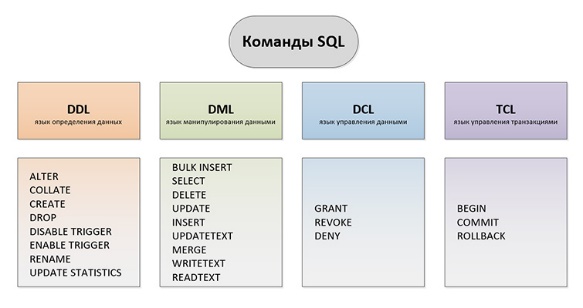
Удалить столбцы:

ALTER TABLE <имя\_таблицы>

DROP (<имя\_столбца>,...)

Модификация типа столбцов:  
ALTER TABLE <имя\_таблицы> MODIFY   
 (<имя\_столбца> <тип\_столбца>   
 [NOT NULL]   
 [UNIQUE | PRIMARY KEY]   
 [REFERENCES <имя\_мастер\_таблицы> <имя\_столбца>]]   
 ,...)

# 18. Язык SQL, средства манипулирования данными. Структура запросов.



## Операторы DML

insert into имя\_табл  
[(список\_полей)]  
values (список\_значений)

insert into имя\_табл [(список\_полей)]  
select...

Для ввода записей из таблицы, отобранных через команду select

update имя\_табл  
 set имя\_поля = выражение  
 set имя\_поля2 = выражение  
 ...  
   
 where условие

изменяет значение в полях, указанных операндом set для записи, удовлетворяющей слову-условию where

delete from имя\_таблицы  
[where условие]

удаляет записи, удовлетворяющие условию (или безусловно)

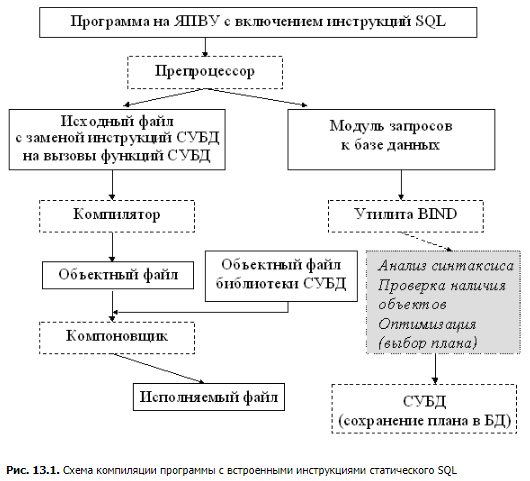
## Операторы DQL

select [all | distinct]  
список  
from список\_таблиц  
[where условие]  
[group by имя\_поля1, имя\_поля2, ...]  
[having условие\_поиска]  
[order by спецификация]

select производит выборку и вычисление над данными из список\_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, а также выражения над полями. В списке полей из разных таблиц, перед именем поля указывается имя таблицы, отделяемое точкой (имя*\_*таблицы.имя\_поля).  
В отношении остальных разделов.

Для обеспечения связи программы-редактора SQL команд и базы данных какого-то формата обращаются к ***услугам*** ODBC (Open Data Base Connectivity - Открытый Интерфейс взаимодействия с Базами Данных (made by Microsoft))

# 19. Использование SQL при прикладном программировании.



В стандарте SQL/89 определен очень ограниченный набор операторов манипулирования данными. Их можно классифицировать на группы операторов, связанных с курсором; одиночных операторов манипулирования данными; и операторов завершения транзакции. Все эти операторы можно использовать как в модулях SQL, так и во встроенном SQL. Заметим, что в SQL/89 не определен набор операторов интерактивного SQL.

B SQL-89 определены способы взаимодействия из БД с прикладными программами, написанных на традиционных языках программирования (Fortran, PL/I, Pascal, C)

Способы взаимодействия:

1) модульный ( операторы собраны в модуль) - Прикладная программа вызывает процедуру модуля, передаёт ей фактические параметры в тело процедуры и получает ответные параметры. Модуль компилируется отдельно от прикладной программы.

2) Встроенный SQL - Непосредственно в программы на языке высокого уровня в SQL оператоpe обмен данными через переменные.

SQL операторы компилируются каждый раз при их использовании, точнее при каждом первом использовании оператора при данном запуске программы.

Достоинства модульного взаимодействия:

* Прикладная программа имеет меньший размер;
* Взаимодействия с БД локализованы.

Недостатки модульного взаимодействия:

* Для понимания смысла программы требуется чтение двух текстов;
* Синтаксис модуля в разных программах бывает разный.

Достоинства встроенного SQL:

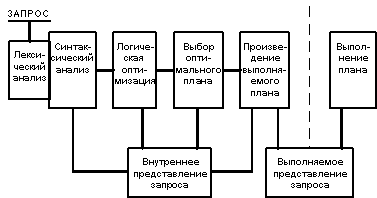
* Прикладная программа более само содержащаяся ;
* Стандарт встраивания единообразен.

Недостатки встроенного SQL :

* Синтаксис оператора SQL контрастирует с базовым.

# 20. Проблемы оптимизации. Схема обработки запроса.

Оптимизация запросов в реляционных СУБД - такой способ обработки запросов, когда по начальному представлению запроса путем его преобразований вырабатывается процедурный план его выполнения, наиболее оптимальный при существующих в базе данных управляющих структурах. Соответствующие преобразования начального представления запроса выполняются специальным компонентом СУБД - оптимизатором, и оптимальность производимого им плана запроса носит условный характер: план оптимален в соответствии с критериями, заложенными в оптимизатор.



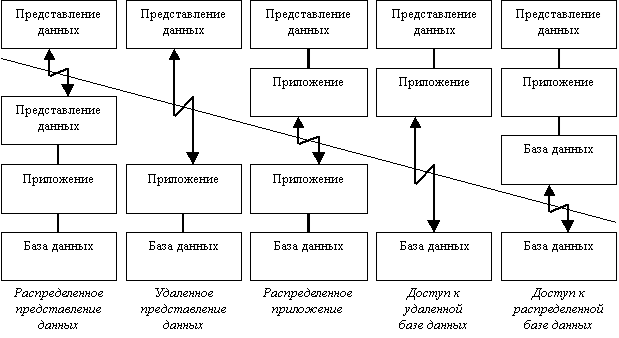
Обработка поступившего в систему запроса

* запрос подвергается лексическому и синтаксическому анализу. При этом вырабатывается его внутреннее представление, отражающее структуру запроса и содержащее информацию, которая характеризует объекты базы данных. Информация о хранимых в базе данных объектах выбирается из каталогов базы данных. Внутреннее представление запроса используется и преобразуется на следующих стадиях обработки запроса.
* запрос во внутреннем представлении подвергается логической оптимизации. Преобразования:
* - э\квивалентные, результат внутреннее представление, семантически эквивалентное начальному
* Семантические: получаемое представление не является семантически эквивалентным начальному, но гарантируется, что результат выполнения преобразованного запроса совпадает с результатом запроса в начальной форме при соблюдении ограничений целостности, существующих в базе данных.

После выполнения второй фазы обработки запроса его внутреннее представление остается непроцедурным, хотя и является в некотором смысле более эффективным, чем начальное.

* выбор альтернативных процедурных планов. Для каждого плана оценивается предполагаемая стоимость выполнения запроса, на основе статистическая информация о состоянии БД. Из них выбирается наиболее дешевый.
* из оптимального плана формируется выполняемое представление плана в машинных кодах или машинно-независимым, но более удобным для интерпретации.
* реальное выполнение - либо выполнение соответствующей подпрограммы, либо вызов интерпретатора с передачей ему для интерпретации выполняемого плана.

# 21. Архитектура "клиент-сервер". Серверы баз данных.



Двухзвенная модель распределенной функции.

Прикладные программы сосредоточены на сервере приложений, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере.

Функции распределяются между Сервером и Клиентом.

Распределенное представление: Сервер (управление, обработка и представление), Клиент (представление).

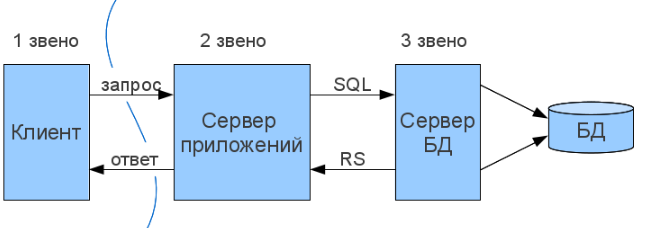
Удаленное представление: Сервер (управление, обработка), Клиент (представление).

Распределенная функция: Сервер (управление, обработка), Клиент (обработка, представление).

Удаленный доступ к данным: Сервер (управление), Клиент (обработка, представление).

Распределенные БД: Сервер (управление), Клиент (управление, обработка, представление).

Трехзвенная модель архитектуры.



Трехзвенная модель распределения функций:

Представление данных - Клиент

Обработка данных - сервер приложения

Управление данными - сервер БД

Каждая из 3-х функций реализована на отдельном компьютере.

Сервер приложения решает прикладную задачу, как службу предоставления услуг клиенту. Серверов приложения может быть несколько. Клиент трактуется широко, может быть реализован в виде датчиков. В более сложных схемах взаимодействия сервер некоторого клиента сам может становиться клиентом по отношению к другому серверу.

Сущность системы со множественными связями: компьютер по отношению к одним - клиент, по отношению к другим - сервер. Во времени их роли могут меняться.

Сервер – компьютер/программа, управляющая некоторым ресурсом (сервер телекоммуникаций, вычислительный сервер, дисковый сервер, файловый сервер, сервер баз данных).

Клиент – компьютер/программа, которые используют этот ресурс.

Ресурсы – БД, файловый сервер, файловая система, служба печати, почтовая служба, вычислительный сервер и т.д.

Реальное распространение архитектуры "клиент-сервер" стало возможным благодаря развитию и широкому внедрению в практику концепции открытых систем.

Ключевой фразой открытых систем является независимость от конкретного поставщика.

В общем случае, чтобы прикладная программа, выполняющаяся на рабочей станции, могла запросить услугу у некоторого сервера, как минимум требуется некоторый интерфейсный программный слой, поддерживающий такого рода взаимодействие. Система разбивается на две части, которые могут выполняться в разных узлах сети, - клиентскую и серверную части.

Прикладная программа или конечный пользователь взаимодействуют с клиентской частью системы. Клиентская часть системы при потребности обращается по сети к серверной части.

Интерфейс серверной части определен и фиксирован.

Каждая функция реализована на отдельной ЭВМ: сервер БД (управление, решает как служба представляет услугу Клиенту), сервер приложений (обработка, может быть много, у каждой свой вид сервиса), Клиент (представление, прием данных).

Доступ к базе данных от прикладной программы или пользователя производится путем обращения к клиентской части системы. В качестве основного интерфейса между клиентской и серверной частями выступает язык баз данных SQL.

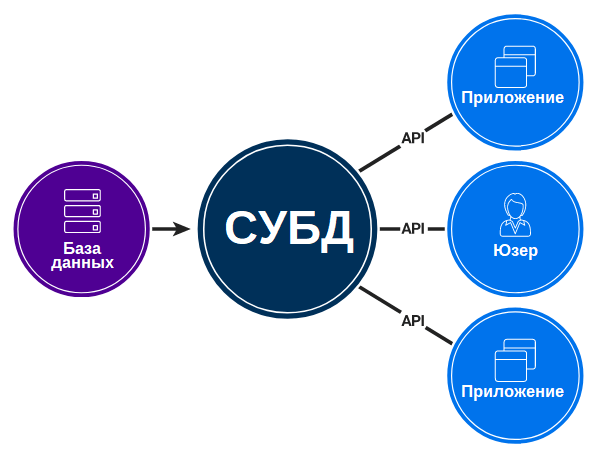
Очевидное преимущество - стандартность интерфейса. Недостаток тоже довольно очевиден. При таком высоком уровне интерфейса между клиентской и серверной частями системы на стороне клиента работает слишком мало программ СУБД.

Типичное разделение функций между клиентами и серверами.

На стороне клиента СУБД работает только такое программное обеспечение, которое не имеет непосредственного доступа к базам данных, а обращается для этого к серверу с использованием языка SQL.

Требования к аппаратным возможностям и базовому программному обеспечению клиентов и серверов.

Если разделение между клиентом и сервером достаточно жесткое, то пользователям, работающим на рабочих станциях или персональных компьютерах, абсолютно все равно, какая аппаратура и операционная система работают на сервере, лишь бы он справлялся с возникающим потоком запросов.

* 

# 22. Распределенные БД.

Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных баз данных, располагающихся в некоторых узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных как к единой базе данных.

При этом должны обеспечиваться:

* простота использования системы;
* возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
* высокая степень эффективности.

Для решения этих проблем было необходимо принять ряд проектных решений, касающихся декомпозиции исходного запроса, оптимального выбора способа выполнения запроса, согласованного выполнения транзакций, обеспечения синхронизации, обнаружения и разрешения распределенных тупиков, восстановления состояния баз данных после разного рода сбоев узлов сети.

Легкость использования системы достигается за счет того, что пользователи System R\* (разработчики прикладных программ и конечные пользователи) остаются в среде языка SQL, т.е. могут продолжать работать в тех же внешних условиях, что и в System R (и SQL/DS и DB2). Возможность использования SQL основывается на обеспечении System R\* прозрачности местоположения данных. Система автоматически обнаруживает текущее местоположение упоминаемых в запросе пользователя объектов данных; одна и та же прикладная программа, включающая предложения SQL, может быть выполнена в разных узлах сети. При этом в каждом узле сети на этапе компиляции запроса выбирается наиболее оптимальный план выполнения запроса в соответствии с расположением данных в распределенной системе.

Обеспечению автономности узлов сети в System R\* уделяется очень большое внимание. Каждая локальная база данных администрируется независимо от других. Возможны автономное подключение новых пользователей, смена версии автономной части системы и т.д. Система спроектирована таким образом, что в ней не требуются централизованные службы именования объектов или обнаружения тупиков. В индивидуальных узлах не требуется наличие глобального знания об операциях, выполняющихся в других узлах сети; работа с доступными базами данных может продолжаться при выходе из строя отдельных узлов сети или линий связи.

Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным системам управления базами данных вообще и к System R\* в частности. Для достижения этой цели используются два основных приема.

Виды блокировок:

1. Полная

2. Блокировка от записи.

3. Предохраняющая блокировка от записи

4. Предохраняющая полная блокировка.

Первая распределенная СУБД System R\*.

1. Пользователи остаются в среде языка SQL. Это возможно благодаря обеспечению прозрачности местоположения данных.
2. всякая лок. БД администрируется независимо от других: нет центральной службы именования объектов, в узлах не треб-ся наличие глобального знания об операциях, выполняемых в других узлах.
3. Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным системам управления базами данных вообще и к System R\* в частности.

Для достижения этой цели используются 2 основных приема:

1) выполнению запроса предшествует его компиляция;

2) возможность перемещения удаленных отношений в локальную базу данных.

* - пользователи узлов всегда обеспечены актуальной информацией, если каналы связи исправны
* - экономность использования памяти

- - высокие требования к каналам связи

Идентичность копий обеспечивает репликатор, он передает отдельные изменения

* - более высокая скорость доступа к данным
* - малый объем передаваемой информации
* - надежность

- - различие(расхождение) между копиями во времени

Распределенная БД состоит из фрагментов, размещенных на разных узлах сети и управляются (иногда) разными СУБД

# 23. Системы управления базами данных следующего поколения.

Направления развития:

1)Поиск совершенных моделей представления данных и типов данных

2)Разработка новых архитектур СУБД

3)Расширения области применения БД

Информационные системы наблюдения земли (EOS), Гридсистемы (GridSyStem)

4)Улучшение сервиса пользователей, администратора и разработчиков.

Перспективы. Лучшее качество СУБД. Совершенствование элементов базы: объем носителей, скорость доступа к ним, скорость обработки запросов, более совершенное ПО, более гибкий удобный интерфейс. Современные БД позволяют хранить информацию о предметной области и информацию о разработчиках. Обеспечена безостановочная работа – режим «7\*24 - работа» и «24\*356 - работа»

5)БД для мобильных пользователей.

Направления в новых СУБД

Направление Postgres. Основная характеристика: максимальное следование (насколько это возможно с учетом новых требований) известным принципам организации СУБД (если не считать коренной переделки системы управления внешней памятью).   
Направление Exodus/Genesis. Основная характеристика: создание собственно не системы, а генератора систем, наиболее полно соответствующих потребностям приложений. Решение достигается путем создания наборов модулей со стандартизованными интерфейсами, причем идея распространяется вплоть до самых базисовых слоев системы.   
Направление Starburst. Основная характеристика: достижение расширяемости системы и ее приспосабливаемости к нуждам конкретных приложений путем использования стандартного механизма управления правилами. По сути дела, система представляет собой некоторый интерпретатор системы правил и набор модулей-действий, вызываемых в соответствии с этими правилами. Можно изменять наборы правил (существует специальный язык задания правил) или изменять действия, подставляя другие модули с тем же интерфейсом.

Ориентация на расширенную реляционную модель. Основной смысл этого направления состоит в том, что в руки проектировщиков даются настолько же мощные и гибкие средства структуризации данных, как те, которые были присущи иерархическим и сетевым системам базам данных.

Абстрактные типы данных. Одной из наиболее известных СУБД третьего поколения является система Postgres. Одно свойство системы Postgres сближает ее со свойствами объектно-ориентированных СУБД. Допускается хранение в полях отношений данных абстрактных, определяемых пользователями типов. Если с использованием такой системы программирования определяются типы данных, хранимых в базе данных, то СУБД оказывается не в состоянии контролировать безопасность этих определений, т.е. то отсутствует гарантия, что при выполнении процедур абстрактных типов данных не будет разрушена сама база данных.

Заметим, что в середине 1995 г. компания Sun Microsystems объявила о выпуске нового продукта - языка и семейства интерпретаторов под названием Java. Язык Java является расширенным подмножеством языка Си++. Основные изменения касаются того, что язык является пооператорно интерпретируемым (в стиле языка Бейсик), а программы, написанные на языке Java, гарантированно безопасны (в частности, при выполнении любой программы не может быть поврежден интерпретатор). Для этого, в частности, из языка удалена арифметика над указателями. В то же время Java остается мощным объектно-ориентированным языком, включающим развитые средства определения абстрактных ТД.

Генерация систем баз данных, ориентированных на приложения. Идея очень проста: никогда не станет возможно создать универсальную систему управления базами данных, которая будет достаточна и не избыточна для применения в любом приложении. Поэтому очень заманчиво производить не законченные универсальные СУБД, а нечто вроде компиляторов компиляторов, позволяющих собрать систему баз данных, ориентированную на конкретное приложение.

Оптимизация запросов, управляемая правилами. Главная неприятность, связанная с оптимизаторами запросов, состоит в том, что отсутствует принятая технология их программирования. Наиболее революционный подход среди известных автору был применен в экспериментальной постреляционной системе компании IBM Starburst. Система Starburst основана на применении продукционной системы. Эта система является, по существу, виртуальной машиной, в которой выполняются все компоненты СУБД, начиная от компилятора языка баз данных и заканчивая подсистемой непосредственного исполнения запросов. Очевидно, что такая организация системы обеспечивает максимальную гибкость. Тем самым, Starburst может использоваться (и реально используется в научно-исследовательских лабораториях компании IBM) как мощное и гибкое средство исследования методов оптимизации запросов. Конечно, сомнительно, что технология, положенная в основу Starburst, позволит этой системе конкурировать с такими выполненными в традиционной манере коммерческими СУБД, как DB2, Oracle, Informix.

# 24. Объектно-ориентированные СУБД.

Объектно-ориентированные СУБД (Системы управления базами данных) используют концепции объектно-ориентированного программирования для хранения данных в виде объектов с методами и свойствами. Они позволяют более естественно моделировать сложные структуры данных и их взаимосвязи. Применяются в приложениях, где данные имеют сложную структуру, таких как графические системы, CAD/CAM, системы управления проектами и другие. Они отличаются от реляционных СУБД, таких как MySQL или PostgreSQL, тем, что они ориентированы на объекты, а не на таблицы и запросы.

Объектно-ориентированные базы данных применяются с конца 1980-х для обеспечения управления [БД](https://www.tadviser.ru/index.php/%D0%91%D0%94), построенными в соответствии с концепцией объектно-ориентированного программирования. Объектная технология расширяет традиционную методику разработки приложений новым моделированием данных и методами программирования. Для повторного использования кода и улучшения сохранности целостности данных в объектном программировании данные и код для их обработки организованы в объекты. Таким образом, практически полностью снимаются ограничения на типы данных. Подходят ООСУБД и для организации распределенных вычислений. Традиционные базы данных построены вокруг центрального сервера, выполняющего все операции над базой. По существу, эта модель мало отличается от мэйнфреймовой организации 60-х годов с центральной [ЭВМ](https://www.tadviser.ru/index.php/%D0%AD%D0%92%D0%9C) — мэйнфреймом (mainframe), выполняющей все вычисления, и пассивных терминалов. Такая архитектура имеет ряд недостатков, главным из которых является вопрос масштабируемости. В настоящее время рабочие станции (клиенты) имеют вычислительную мощность порядка 30 — 50 % мощности сервера базы данных, то есть большая часть ресурсов распределена среди клиентов. Поэтому все больше приложений, и в первую очередь базы данных и средства принятия решений, работают в распределенных средах, в которых объектные программные компоненты распределены по многим рабочим станциям и серверам и где любой пользователь может получить доступ к любому объекту. Благодаря стандартам межкомпонентного взаимодействия все эти фрагменты кода комбинируются друг с другом независимо от аппаратного, программного обеспечения, операционных систем, сетей, компиляторов, языков программирования, различных средств организации запросов и формирования отчетов и динамически изменяются при манипулировании объектами без потери работоспособности.

Достоинства

В отличие от реляционных, ООСУБД полностью поддерживают объектно-ориентированные языки программирования. Разработчики, применяющие [С++](https://www.tadviser.ru/index.php?title=%D0%A1%2B%2B&action=edit&redlink=1) или [Smalltalk](https://www.tadviser.ru/index.php?title=Smalltalk&action=edit&redlink=1), имеют дело с одним набором правил, позволяющих использовать такие преимущества объектной технологии, как наследование, инкапсуляция и полиморфизм. Разработчик не должен прибегать к трансляции объектной модели в реляционную и обратно. Прикладные программы обращаются и функционируют с объектами, сохраненными в базе данных, которая использует стандартную объектно-ориентированную семантику языка и операции.

Примеры:

* [Caché](https://www.tadviser.ru/index.php?title=Cach%C3%A9&action=edit&redlink=1)
* [Cerebrum](https://www.tadviser.ru/index.php?title=Cerebrum&action=edit&redlink=1)
* [IBM Lotus Notes/Domino](https://www.tadviser.ru/index.php/IBM_Lotus_Notes/Domino)
* [Jasmine](https://www.tadviser.ru/index.php?title=Jasmine&action=edit&redlink=1)
* [ObjectStore](https://www.tadviser.ru/index.php?title=ObjectStore&action=edit&redlink=1)
* [ODANT Object Data Access Network Technology Распределенная объектно-сервисная СУБД](https://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:ODANT_Object_Data_Access_Network_Technology_%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94)

В наиболее общей и классической постановке объектно-ориентированный подход базируется на следующих концепциях: объекта и идентификатора объекта; атрибутов и методов; классов; иерархии и наследования классов.

Сущность реального мира моделируется в виде объекта, объект имеет идентификатор, состояние и поведение.

* Состояние – набор значений его атрибутов.
* Поведение – набор методов, оперирующих над состоянием.
* Множество объектов с одним набором атрибутов и методов образует класс объектов.

В наиболее общей и классической постановке объектно-ориентированный подход базируется на следующих концепциях:

* объекта и идентификатора объекта;
* атрибутов и методов;
* классов;
* иерархии и наследования классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом все время его существования и не меняется при изменении состояния объекта.

Каждый объект имеет состояние и поведение. Состояние объекта - набор значений его атрибутов. Поведение объекта - набор методов (программный код), оперирующих над состоянием объекта. Значение атрибута объекта — это тоже некоторый объект или множество объектов. Состояние и поведение объекта инкапсулированы в объекте; взаимодействие объектов производится на основе передачи сообщений и выполнении соответствующих методов.

Допускается порождение нового класса на основе уже существующего класса - наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса), наследует все атрибуты и методы суперкласса. Наиболее важным новым качеством ООБД, которого позволяет достичь объектно-ориентированный подход, является поведенческий аспект объектов.

Наиболее важным новым качеством ООБД, которого позволяет достичь объектно-ориентированный подход, является поведенческий аспект объектов.

Возможны два вида организации данных: классы, экземплярами которых являются объекты, инкапсулирующие данные и поведение, и типы, экземплярами которых являются значения.

Примеры объектно-ориентированных СУБД.

1. Проект ORION осуществлялся с 1985 по 1989 г. фирмой MCC. Под названием ORION на самом деле скрывается семейство трех СУБД: ORION-1 - однопользовательская система; ORION-1SX, предназначенная для использования в качестве сервера в локальной сети рабочих станций; ORION-2 - полностью распределенная объектно-ориентированная СУБД.

Основными функциональными компонентами системы являются подсистемы управления памятью, объектами и транзакциями. В число функций подсистемы управления памятью входит распределение внешней памяти, перемещение страниц из буферов оперативной памяти во внешнюю память и наоборот, поиск и размещение объектов в буферах оперативной памяти. Подсистема управления объектами включает подкомпоненты обработки запросов, управления схемой и версиями объектов. Подсистема управления транзакциями обеспечивает традиционную сериализуемость транзакций, а также поддерживает средства журнализации изменений и восстановления БД после сбоев.

1. Проект O2 выполнялся французской компанией Altair, образованной специально для целей проектирования и реализации объектно-ориентированной СУБД. Начало проекта датируется сентябрем 1986 г. Прототип системы функционировал в режиме клиент/сервер в локальной сети рабочих станций SUN c соответствующим разделением функций между сервером и клиентами. Основными компонентами системы являются интерпретатор запросов и подсистемы управления схемой, объектами и дисками.