1. Понятие информации, данных, базы данных, примеры БД.

Информация. (из презентации) Information is that which informs.
Данные. (из презентации) Data is a set of values of qualitative or quantitative variables.
Данные - это совокупность сведений, зафиксированных на определенном носителе в форме, пригодной для постоянного хранения, передачи и обработки. Преобразование и обработка данных позволяет получить информацию.
Информация - это результат преобразования и анализа данных. Отличие информации от данных состоит в том, что данные - это фиксированные сведения о событиях и явлениях, которые хранятся на определенных носителях, а информация появляется в результате обработки данных при решении конкретных задач. Например, в базах данных хранятся различные данные, а по определенному запросу система управления базой данных выдает требуемую информацию.

Как смотреть на данные?
Инфологический аспект – про смысловое содержание данных,
даталогический аспект – про представление данных в памяти ВМ.

База данных. (из презентации) A database is an organized collection of data.
База данных — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ)

Примеры БД:
Медицинская карта, XLSX-файл, чат в мессенджере, социальная сеть, товарный чек, файловая система, географическая карта

---

2. Типовые наборы операций над данными. Понятие, функции, классификация СУБД.

Типовые наборы операций над данными.

CRUD (сокр. от англ create, read, update, delete — «создать, прочесть, обновить, удалить») — акроним, обозначающий четыре базовые функции, используемые при работе с персистентными хранилищами данных:

- создание;
- чтение;
- редактирование;
- удаление.

Операции в SQL: create – INSERT, read – SELECT, update – UPDATE, delete – DELETE

СУБД.
A database-management system (DBMS) is a computer-software application that interacts with end-users, other applications, and the database itself to capture and analyze data.
Система управления базами данных (**СУБД**) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных
//(**База данных** — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ))

Задачи СУБД:
- Массовость,
- хранение (персистентное),
- поиск,
- безопасность,
- удобство,
- эффективность,
- надежность, восстановление после сбоев
- многопользовательский доступ

Кто работает с СУБД?
Разработчик СУБД, Разработчик БД, Разработчик ПО, Администратор БД, Системный администратор.

Основные функции СУБД:

1. Непосредственное управление данными во внешней памяти
Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным**.**

2. Управление буферами оперативной памяти
СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся (система будет работать со скоростью устройства внешней памяти.
Практически единственным (способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД

поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

3. Управление транзакциями

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется и СУБД фиксирует изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД

4. Журнализация

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии).

5. Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - язык определения схемы БД (SDL) и язык манипулирования данными (DML). SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных.

Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language), который сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Классификация СУБД. *Модель данных*

- Иерархические

- Реляционные
- Графовые
- Объектные
- Документоориентированные
- Ключ-значение
- Колоночные
- Дедуктивные
- Пространственные/геометрические

Исполнение

- Встраиваемые
- Выделенные
- Распределенные

Хранение данных

- В памяти
- На диске
- Хранилища
- В сети

Использование

- Общие задачи
- Сообщения
- Счетчики
- Логирование
- OLTP
- OLAP
- Entity enrichment
- Версионирование

Другие аспекты

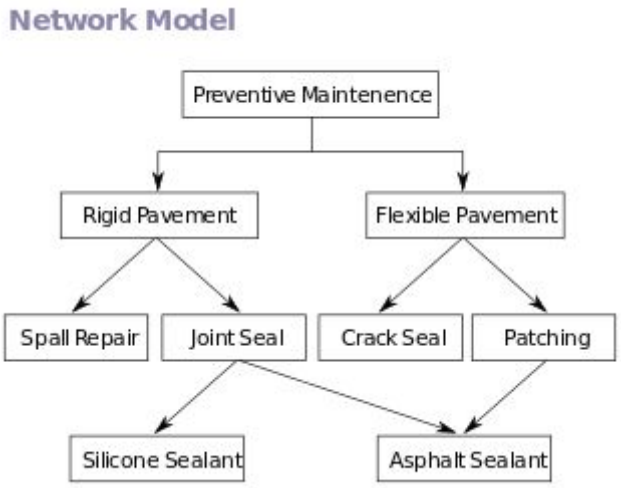
- Язык запросов
- Консистентность
- Транзакции
- Репликация
- Устойчивость к внешним воздействиям
- Резервное копирование

---

3. Модель данных. Составляющие модели данных.

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных[1].

Каждая БД и СУБД строится на основе некоторой явной или неявной модели данных. Все СУБД, построенные на одной и той же модели данных, относят к одному типу. Например, основой реляционных СУБД является реляционная модель данных (**Реляционная модель данных** (РМД) — логическая модель данных, прикладная теория построения баз данных, которая является приложением к задачам обработки данных таких разделов математики, как теория множеств и логика первого порядка.), сетевых СУБД — сетевая модель данных (**Сетевая модель данных** — логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных, см. картинку ниже), иерархических СУБД — иерархическая модель данных (**Иерархическая модель данных** — это модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.) и т. д.



Картинка для СМД:

В модели данных описывается некоторый набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, если они основываются на этой модели. Наличие модели данных позволяет сравнивать конкретные реализации, используя один общий язык.

Хотя **понятие модели данных** было введено Коддом, наиболее распространенная трактовка модели данных, по-видимому, **принадлежит Кристоферу Дейту**, который воспроизводит ее (с различными уточнениями) применительно к реляционным БД практически во всех своих книгах. Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: **структурной части, манипуляционной части и целостной части.**

В структурной части модели данных фиксируются основные логические структуры данных, которые могут применяться на уровне пользователя при организации БД, соответствующих данной модели. Например, в модели данных SQL основным видом структур базы данных являются таблицы, а в объектной модели данных – объекты ранее определенных типов.

Манипуляционная часть модели данных содержит спецификацию одного или нескольких языков, предназначенных для написания запросов к БД. Эти языки могут быть абстрактными, не обладающими точно проработанным синтаксисом (что

свойственно языками реляционной алгебры и реляционного исчисления, используемым в реляционной модели данных), или законченными производственными языками (как в случае модели данных SQL). Основное назначение манипуляционной части модели данных – обеспечить эталонный «модельный» язык БД, уровень выразительности которого должен поддерживаться в реализациях СУБД, соответствующих данной модели.
Наконец, **в целостной части модели данных** (которая явно выделяется не во всех известных моделях) специфицируются механизмы ограничений целостности, которые обязательно должны поддерживаться во всех реализациях СУБД, соответствующих данной модели. Например, в целостной части реляционной модели данных категорически требуется поддержка ограничения первичного ключа в любой переменной отношения, а аналогичное требование к таблицам в модели данных SQL отсутствует.

---

4. Реляционная модель данных. Структурная часть.

Реляционной моделью называется база данных, в которой все данные, доступные пользователю, организованы в виде таблиц, а все операции над данными сводятся к операциям над этими таблицами. Наглядной формой представления отношения является двумерная таблица.

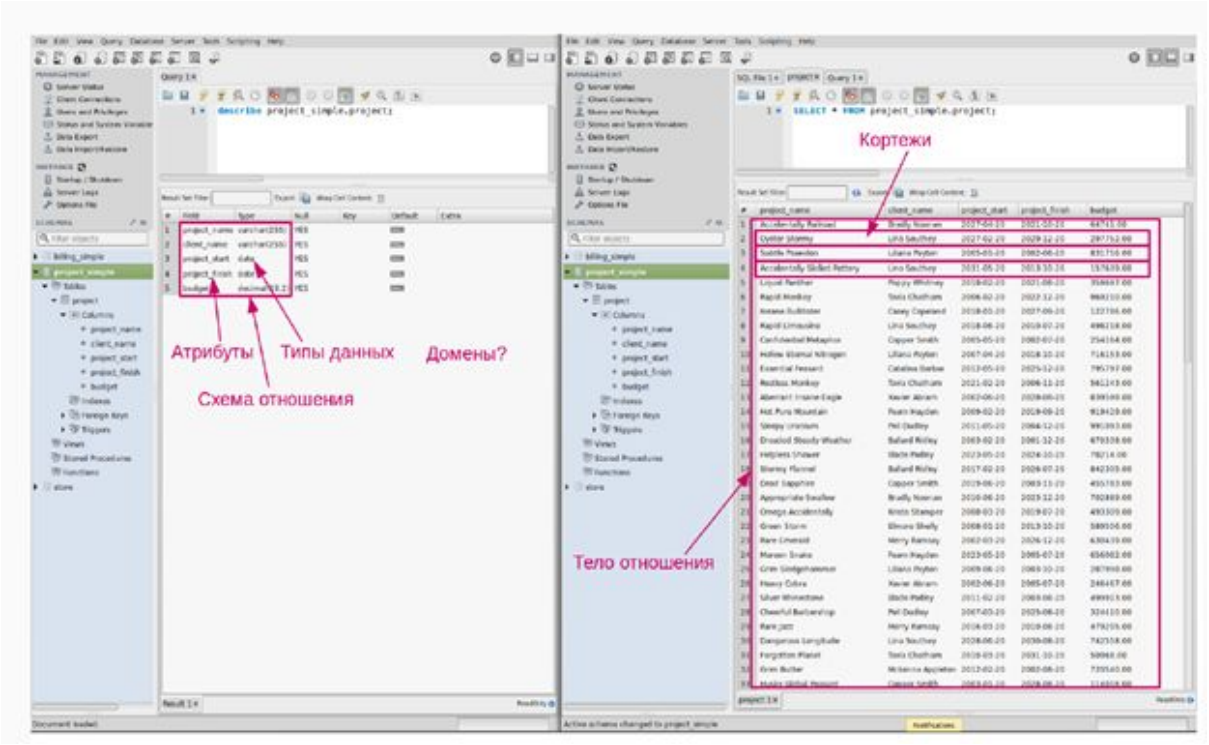
Реляционная база данных — это совокупность взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определенного типа.

На реляционной модели данных строятся реляционные базы данных.

Реляционная модель данных включает следующие компоненты:

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

Структурная часть:



1) Тип данных

Любые данные, используемые в программировании, имеют свои типы данных.

Важно! Реляционная модель требует, чтобы типы используемых данных были простыми.

Для уточнения этого утверждения рассмотрим, какие вообще типы данных обычно рассматриваются в программировании. Как правило, типы данных делятся на три группы:

- Простые типы данных. (Простые, или атомарные, типы данных не обладают внутренней структурой. Данные такого типа называют скалярами. К простым типам данных относятся следующие типы: Логический, Строковый, Численный)
- Структурированные типы данных. (Структурированные типы данных предназначены для задания сложных структур данных – массивы, записи\структуры)
- Ссылочные типы данных. (Ссылочный тип данных (указатели) предназначен для обеспечения возможности указания на другие данные. Указатели характерны для языков процедурного типа, в которых есть понятие области памяти для хранения данных.)

Собственно, для реляционной модели данных тип используемых данных не важен. Требование, чтобы тип данных был простым, нужно понимать так, что в реляционных операциях не должна учитываться внутренняя структура данных. Конечно, должны быть описаны действия, которые можно производить с данными как с единым целым, например, данные числового типа можно складывать, для строк возможна операция конкатенации и т.д.

С этой точки зрения, если рассматривать массив, например, как единое целое и не использовать поэлементных операций, то массив можно считать простым типом

данных. Более того, можно создать свой, сколь угодно сложных тип данных, описать возможные действия с этим типом данных, и, если в операциях не требуется знание внутренней структуры данных, то такой тип данных также будет простым с точки зрения реляционной теории. Например, можно создать новый тип - комплексные числа как запись вида $z=(x,y)$, где $x \in \mathbb{R}, y \in \mathbb{R}$. Можно описать функции сложения, умножения, вычитания и деления, и все действия с компонентами x и y выполнять только внутри этих операций. Тогда, если в действиях с этим типом использовать только описанные операции, то внутренняя структура не играет роли, и тип данных извне выглядит как атомарный.

Именно так в некоторых пост-реляционных СУБД реализована работа со сколь угодно сложными типами данных, создаваемых пользователями.

2) Домен

Домен в реляционной модели данных — тип данных, то есть множество допустимых значений.

Домен - это семантическое понятие. Домен можно рассматривать как подмножество значений некоторого типа данных имеющих определенный смысл. Домен характеризуется следующими свойствами:

- Домен имеет уникальное имя (в пределах базы данных).
- Домен определен на некотором простом типе данных или на другом домене.
- Домен может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для данного домена.
- Домен несет определенную смысловую нагрузку.

3) Атрибут\поле

Атрибут — свойство некоторой сущности. Часто называется полем таблицы.

Атрибут отношения *есть пара вида <Имя_атрибута : Имя_домена>.*

Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

4) Кортеж\строка

Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).

кортеж отношения представляет собой множество пар вида <Имя_атрибута : Значение_атрибута>:

$$(<A_1 : Val_1>, <A_2 : Val_2>, \ldots, <A_n : Val_n>)$$

таких что значение Val_i атрибута A_i принадлежит домену D_i

5) Схема отношения\заголовок таблицы

Схема отношения — конечное множество атрибутов, определяющих некоторую сущность. Иными словами, это структура таблицы, состоящей из конкретного набора полей.

6) Отношение\таблица

Отношение — конечное множество кортежей (таблица).

Отношение R , определенное на множестве доменов $D_1, D_2, \ldots, D_n$ (не обязательно различных), содержит две части: заголовок и тело.

Заголовок отношения содержит фиксированное количество атрибутов отношения:

$$(< A_1 : D_1 >, < A_2 : D_2 >, \ldots, < A_n : D_n >)$$

Тело отношения содержит множество кортежей отношения.

7) Схема БД

Схемой реляционной базы данных называется набор заголовков отношений, входящих в базу данных.

При табличной организации данных отсутствует иерархия элементов. Строки и столбцы могут быть просмотрены в любом порядке, поэтому высока гибкость выбора любого подмножества элементов в строках и столбцах. Любая таблица в реляционной базе состоит из строк, которые называют **записями**, и столбцов, которые называют **полями**. На пересечении строк и столбцов находятся конкретные значения данных. Для каждого поля определяется множество его значений.

В реляционной модели данных применяются разделы реляционной алгебры, откуда и была заимствованна соответствующая терминология.В реляционной алгебре поименованный столбец отношения называется **атрибутом**, а множество всех возможных значений конкретного атрибута – **доменом**. Строки таблицы со значениями разных атрибутов называют **кортежами**. Атрибут, значение которого однозначно идентифицирует кортежи, называется **ключевым** (или просто ключом). Так **ключевое поле** – это такое поле, значения которого в данной таблице не повторяется. В отличие от иерархической и сетевой моделей данных в реляционной отсутствует понятие группового отношения. Для отражения ассоциаций между кортежами разных отношений используется дублирование их ключей. Сложный ключ выбирается в тех случаях, когда ни одно поле таблицы однозначно не определяет запись.

Записи в таблице хранятся упорядоченными по ключу. Ключ может быть простым, состоящим из одного поля, и сложным, состоящим из нескольких полей. Сложный ключ выбирается в тех случаях, когда ни одно поле таблицы однозначно не определяет запись.

Кроме первичного ключа в таблице могут быть вторичные ключи, называемые еще внешними ключами, или индексами. **Индекс** – это поле или совокупность полей, чьи значения имеются в нескольких таблицах и которое является первичным ключом в одной из них. Значения индекса могут повторяться в некоторой таблице. Индекс обеспечивает логическую последовательность записей в таблице, а также прямой доступ к записи.

По первичному ключу всегда отыскивается только одна строка, а по вторичному – может отыскиваться группа строк с одинаковыми значениями первичного ключа. Ключи нужны для однозначной идентификации и упорядочения записей таблицы, а индексы для упорядочения и ускорения поиска.

Индексы можно создавать и удалять, оставляя неизменным содержание записей реляционной таблицы. Количество индексов, имена индексов, соответствие индексов полям таблицы определяется при создании схемы таблицы.

Индексы позволяют эффективно реализовать поиск и обработку данных, формирую дополнительные индексные файлы. При корректировке данных автоматически упорядочиваются индексы, изменяется местоположение каждого индекса согласно принятому условию (возрастанию или убыванию значений). Сами же записи

реляционной таблицы не перемещаются при удалении или включении новых экземпляров записей, изменении значений их ключевых полей.
С помощью индексов и ключей устанавливаются связи между таблицами. Связь устанавливается путем присвоения значений внешнего ключа одной таблицы значениям первичного ключа другой. Группа связанных таблиц называется схемой данных. Информация о таблицах, их полях, ключах и т.п. называется метаданными.

---

5. Реляционная алгебра.

РЕЛЯЦИОННАЯ АЛГЕБРА
•РА – это математический аппарат на основе теории множеств, специфицирующий операции, позволяющие создавать на основе одного или нескольких отношений другое отношение без изменения исходных отношений.
•РА является языком последовательного использования отношений, в котором все кортежи, возможно, даже взятые из различных отношений, обрабатываются одном командой, без организации циклов.
Операции РА
Пять основных операций:
•Выборка (selection)
•Проекция (projection)
•Декартово произведение (cartesian product)
•Объединение (union)
•Разность множеств (set difference)

Дополнительные:
•Соединение (join)
•Пересечение (intersection)
•Деление (division)

•Операции выборки и селекции являются *унарными* (работают с одним отношением).
•Остальные – *бинарные*.

Выборка (селекция, ограничение) применяется к одному отношению R и определяет результирующее отношение, которое содержит только те кортежи из отношения R, которые удовлетворяют заданному условию (предикату).

Проекция применяется к одному отношению R и определяет новое отношение, содержащее вертикальное подмножество отношения R, создаваемое посредством извлечения значений указанных атрибутов и исключения из результата строк-дублетов.
Объединение R = R1VR2 определяет новое отношение, которое включает все кортежи, которые принадлежат или одному из двух исходных отношений, или обоим одновременно, причем все дубликаты кортежей исключены.

Пересечение определяет новое отношение, которое включает все кортежи, которые принадлежат одновременно двум определенным отношениям.

Вычитание (разность) R = R1 – R2 состоит из кортежей, которые принадлежат первому из двух определенных отношений и не принадлежат второму.

Декартово произведение R1xR2 определяет новое отношение, которое является результатом конкатенации (сцепления) каждого кортежа из отношения R1с каждым кортежем из отношение R2.
•Т.е. результирующее отношение содержит все возможные пары кортежей обоих отношений.

Соединение
•*Тета-соединение*. Определяет отношение, в котором содержатся кортежи из декартова произведения отношений R1 и R2, удовлетворяющие предикату F.
•*Соединение по эквивалентности* - если предикат F содержит только операцию сравнения по равенству (=).
•***Естественное соединение*** – соединение по эквивалентности двух отношение R1 и R2, выполненное по всем общим атрибутам x, из результатов которого исключено по одному экземпляру каждого общего атрибута.
•Внешнее соединение
•Полусоединение

Деление
•Пусть отношение R1 определено на множестве атрибутов A, а отношение R2 определено на множестве атрибутов B, причем B является подмножеством A. Пусть C=A-B, т.е. C является подмножеством атрибутов отношения R1, которые не являются атрибутами отношения R2.
•Результатом операции деления R1/R2 является набор кортежей отношения R1, определенных на множестве атрибутов C, которые соответствуют комбинации всех кортежей отношения R2.

Пример 1
Получить имена поставщиков, которые поставляют деталь P2
•((SP JOIN S) WHERE P# = 'P2') [SNAME]

1.Соединение SP и S по общему атрибуту S# – «которые поставляют»
2.Выборка кортежей, где P# = 'P2' - «деталь P2»
3.Проекция на SNAME – «имена поставщиков»

Пример 3
Получить имена поставщиков, которые поставляют все детали
((SP [S#, P#] DIVIDE BY P [P#]) JOIN S) [SNAME]
1.Проекция SP на S#, P# - получаем бинарное делимое для операции деления

2.Проекция P на P# - получаем унарный делитель для операции деления (номера всех деталей, которые поставляются)
3.Деление – «которые поставляют **все** детали». В результате – унарное отношение с атрибутом S#.
4.Соединение частного с S – для получения информации об именах поставщиков
5.Проекция на SNAME - имена поставщиков

Пример 4
Получить номера поставщиков, поставляющих по крайней мере все те детали, которые поставляет поставщик S2
•SP [S#, P#] DIVIDE BY

(SP WHERE S# = 'S2') [P#]
1.Проекция SP на S#, P# - получаем бинарное делимое для операции деления
2.Выбора кортежей SP, где номер поставщика S2 - получаем сведения обо всех поставках, которые выполнял поставщик S2
3.Проекция на P# - получаем унарный делитель для операции деления (номера всех деталей, которые поставлял поставщик S2)
4.Деление – «поставляющих **все** те детали»
5.В результате – унарное отношение с атрибутом S# - «получить номера поставщиков»

---

6. Реляционная модель данных. Понятие ключа. Внешний ключ.

Реляционная модель данных (РМД) — логическая модель данных, прикладная теория построения баз данных, которая является приложением к задачам обработки данных таких разделов математики как теории множеств и логика первого порядка. На реляционной модели данных строятся реляционные базы данных.

Структурный аспект (составляющая) — данные в базе данных представляют собой набор отношений.

•***Отношение*** – таблица с колонками и столбцами (относится к концептуальному и внешнему уровням модели ANSI/SPARC)
•***Атрибут*** – именованная колонка в отношении.
•***Домен*** – набор допустимых значений одного или более атрибутов.

•***Кортеж*** – строка таблицы (одно значение для каждого атрибута)
•***Степень*** – число атрибутов в кортеже
•***Кардинальность*** – количество кортежей в отношении
•***РБД*** – набор нормализованных отношений

•В отношении выделяют две части: ***тело отношения и заголовок отношения***. Заголовок – это схема отношения. Тело – это множество всех кортежей, присутствующих в данный момент в отношении.

Свойства отношений

1.Отношение имеет имя, отличное от имен всех других отношений БД.
2.Значения атрибутов являются атомарными (т.е. не допускаются повторяющиеся группы).
3.Имена атрибутов уникальны.
4.Все значения атрибута принадлежат одному и тому же домену.
5.Порядок атрибутов несущественен.
6.Каждый кортеж уникален, нет повторяющихся кортежей.
7.Порядок кортежей несущественен (теоретически).

Потенциальный ключ

Пусть *R* — некоторое отношение. Тогда **потенциальный ключ** *K* для *R* — это подмножество множества атрибутов *R,*обладающее следующими свойствами:
1.Свойством уникальности.
Нет двух различных кортежей в отношении *R с* одинаковым значением *K.*
2.Свойством неизбыточности.
Никакое из подмножеств *K* не обладает свойством уникальности.

Первичные и альтернативные ключи

Один из потенциальных ключей должен быть выбран в качестве ***первичного ключа*** в базовом отношении, а остальные потенциальные ключи, если они есть, будут называться ***альтернативными ключами***

Внешние ключи

Пусть *R2* — базовое отношение. Тогда **внешний ключ** *FK* в отношении *R2* — это подмножество множества атрибутов *R2,* такое что:
•существует базовое отношение *R1 (R1* и *R2* не обязательно различны) с потенциальным ключом *CK;*
•каждое значение *FK* в текущем значении *R2* всегда совпадает со значением *CK* некоторого кортежа в текущем значении *R1.*

Часто в определении внешнего ключа используют понятие первичного, а не потенциального ключа. В большинстве практических случаев это справедливо, однако с теоретических позиций это частный случай.

Значение внешнего ключа представлено **ссылкой** к кортежу, содержащему соответствующее значение потенциального ключа **(ссылочный кортеж** или **целевой кортеж).**
Проблема обеспечения того, что база данных не включает никаких неверных значений внешних ключей, известна как проблема **ссылочной целостности.**
Ограничение, по которому значения данного внешнего ключа должны быть адекватны значениям соответствующих потенциальных ключей, называют **ссылочным ограничением.**

Отношение, которое содержит внешний ключ, называется **ссылающимся отношением,** а отношение, которое содержит соответствующий потенциальный ключ, — **ссылочным отношением** или **целевым отношением (target relation).**

---

7. Модель данных SQL.

Модель данных SQL в относительно законченном виде сложилась к 1999 г., когда был принят и опубликован стандарт SQL:1999.

Типы и структуры данных SQL
SQL-ориентированная база данных представляет собой набор таблиц, каждая из которых в любой момент времени содержит некоторое мультимножество строк, соответствующих заголовку таблицы. В этом состоит первое и наиболее важное отличие модели данных SQL от реляционной модели данных. Вторым существенным отличием является то, что для таблицы поддерживается порядок столбцов, соответствующий порядку их определения. Другими словами, таблица – это вовсе не отношение, хотя во многом они похожи.

Имеется две основных разновидности таблиц, хранимых в базе данных:

1. Традиционная таблица
Множество столбцов с указанными типами данных.
В SQL поддерживаются следующие категории типов данных:

- точные числовые типы;
- приближенные числовые типы;
- типы символьных строк;
- типы битовых строк;
- типы даты и времени;
- типы временных интервалов;
- булевский тип - содержит три значения – *true*, *false* и *uknown*. Это связано с интенсивным использованием в SQL так называемого неопределенного значения (NULL), которое разрешается использовать вместо значения любого типа данных
- типы коллекций;
- анонимный строчный тип – это безымянный структурный тип, значения которого являются строками, состоящими из элементов ранее

определенных типов. Поддерживается два вида типов данных, определяемых пользователями: *индивидуальные* и *структурные* типы. *Индивидуальный тип* – это именованный тип данных, основанный на единственном предопределенном типе. Индивидуальный тип не наследует от своего опорного типа набор операций над значениями. Чтобы выполнить некоторую операцию базового типа над значениями определенного над ним индивидуального типа, требуется явно сообщить системе, что с этими значениями нужно обращаться как со значениями базового типа. Имеется также возможность явного определения методов, функций и процедур, связанных с данным индивидуальным типом. *Структурный тип данных* – это именованный тип данных, включающий один или более атрибутов любого из допустимых в SQL типа данных, в том числе другого структурного типа, типа коллекций, анонимного строчного типа и т. д. Дополнительные механизмы определяемых пользователями методов, функций и процедур позволяют определить поведенческие аспекты структурного типа. При определении структурного типа можно использовать механизм наследования от ранее определенного структурного типа.

- типы, определяемые пользователем;
- ссылочные типы.

2. Типизированная таблица.

При определении типизированной таблицы указывается ранее определенный структурный тип, и если в нем содержится *n* атрибутов, то в таблице образуется *n*+1 столбец, из которых последние *n* столбцов с именами и типами данных, совпадающими именам и типам атрибутов структурного типа. Первый же столбец, имя которого явно задается, называется «самоссылающимся» и содержит типизированные уникальные идентификаторы строк, которые могут генерироваться системой при вставке строк в типизированную таблицу, явно указываться пользователями или состоять из комбинации значений других столбцов. Типом «самоссылающегося» столбца является ссылочный тип, ассоциированный со структурным типом типизированной таблицы. Способ генерации значений ссылочного типа указывается при определении соответствующего структурного типа и подтверждается при определении типизированной таблицы.

При определении типизированных таблиц можно использовать механизм наследования. Можно определить подтаблицу типизированной супертаблицы, если структурный тип подтаблицы является непосредственным подтипом структурного типа супертаблицы. Подтаблица наследует у супертаблицы способ генерации значений ссылочного типа и все ограничения целостности, которые были специфицированы в определении супертаблицы. Дополнительно можно определить ограничения, затрагивающие новые столбцы.

С типизированной таблицей можно обращаться, как с традиционной таблицей, считая, что у нее имеются неявно определенные столбцы, а можно относиться к строкам типизированной таблицы, как к объектам структурного типа, OID которых содержатся в «самоссылающемся» столбце. Ссылочный тип можно использовать для типизации столбцов традиционных таблиц и атрибутов структурных типов, на которых потом определяются типизированные таблицы. В последнем случае можно считать, что значениями атрибутов соответствующих объектов являются объекты структурного типа, с которыми ассоциирован данный ссылочный тип.

Манипулирование данными в SQL

Оператор SQL **SELECT**, предназначенного для выборки данных и имеет следующий синтаксис:

```
SELECT [ ALL | DISTINCT ] select_item_commalist
FROM table_reference_commalist
[ WHERE conditional_expression ]
[ GROUP BY column_name_commalist ]
[ HAVING conditional_expression ]
[ ORDER BY order_item_commalist ]
```

Выборка данных производится из одной или нескольких таблиц, указываемых в разделе **FROM** запроса. В последнем случае на первом этапе выполнения оператора **SELECT** образуется одна общая таблица, получаемая из исходных таблиц путем применения операции расширенного декартова умножения.

Таблицы могут быть как базовыми, реально хранимыми в базе данных (традиционными или типизированными), так и порожденными, т.е. задаваемыми в виде некоторого оператора **SELECT.** Это допускается, поскольку результатом выполнения оператора **SELECT** в его базовой форме является традиционная таблица. Кроме того, в разделе **FROM** можно указывать выражения соединения базовых и/или порожденных таблиц, результатами которых опять же являются традиционные таблицы.
На следующем шаге общая таблица, полученная после выполнения раздела, подвергается фильтрации путем вычисления для каждой ее строки логического выражения, заданного в разделе **WHERE** запроса. В отфильтрованной таблице остаются только те строки общей таблицы, для которых значением логического выражения является *true*.
Если в операторе отсутствует раздел **GROUP BY**, то после этого происходит формирование результирующей таблицы запроса путем вычисления выражений, заданных в списке выборки оператора **SELECT**. В этом случае список выборки вычисляется для каждой строки отфильтрованной таблицы, и в результирующей таблице появится ровно столько же строк.
При наличии раздела **GROUP BY** из отфильтрованной таблицы получается сгруппированная таблица, в которой каждая группа состоит из кортежей отфильтрованной таблицы с одинаковыми значениями столбцов группировки, задаваемых в разделе **GROUP BY**. Если в запросе отсутствует раздел HAVING, то результирующая таблица строится прямо на основе сгруппированной таблицы. Иначе образуется отфильтрованная сгруппированная таблица, содержащая только те группы, для которых значением логического выражения, заданного в разделе **HAVING,** является *true*.
Результирующая таблица на основе сгруппированной или отфильтрованной сгруппированной таблицы строится путем вычисления списка выборки для каждой группы. Тем самым, в результирующей таблице появится ровно столько строк, сколько групп содержалось в сгруппированной или отфильтрованной сгруппированной таблице.
Если в запросе присутствует ключевое слово **DISTINCT**, то из результирующей таблицы устраняются строки-дубликаты, т.е. запрос вырабатывает не мультимножество, а множество строк.

Наконец, в запросе может присутствовать еще и раздел **ORDER BY**. В этом случае результирующая таблица сортируется в порядке возрастания или убывания в соответствии со значениями ее столбцов, указанных в разделе **ORDER BY**. Результатом такого запроса является не таблица, а отсортированный список, который нельзя сохранить в базе данных. Сам же запрос, содержащий раздел **ORDER BY**, нельзя использовать в разделе **FROM** других запросов.
Приведенная характеристика средств манипулирования данными языка SQL является не вполне точной и полной. Кроме того, она отражает *семантику* оператора SQL, а не то, как он обычно исполняется в SQL-ориентированных СУБД.

Ограничения целостности в модели SQL

Наиболее важным отличием модели данных SQL от реляционной модели данных является то, что таблицы SQL могут содержать мультимножества строк. Из этого, в частности, следует, что в модели SQL отсутствует обязательное предписание об ограничении целостности сущности. В базе данных могут существовать таблицы, для которых не определен первичный ключ. С другой стороны, если для таблицы определен первичный ключ, то для нее ограничение целостности сущности поддерживается точно так же, как это требуется в реляционной модели данных.

Ссылочная целостность в модели данных SQL поддерживается в обязательном порядке, но в трех разных вариантах, лишь один из которых полностью соответствует реляционной модели.
Кроме того, в SQL имеются развитые возможности явного определения ограничений целостности на уровне столбцов таблиц, на уровне таблиц целиком и на уровне базы данных.

---

8. Функциональные зависимости. Аксиомы.

Функциональные зависимости

Наиболее важные с практической точки зрения нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости.

Общие определения

Пусть задана переменная отношения r, и X и Y являются произвольными подмножествами заголовка r ("составными" атрибутами).

В значении переменной отношения r **атрибут Y функционально зависит от атрибута X** в том и только в том случае, если каждому значению X соответствует в точности одно значение Y. В этом случае говорят также, что атрибут X **функционально определяет** атрибут Y (X является детерминантом (**определителем**) для Y, а Y является зависимым от X). Будем обозначать это как r.X->r.Y .

Для примера будем использовать отношение СЛУЖАЩИЕ_ПРОЕКТЫ {СЛУ_НОМ, СЛУ_ИМЯ, СЛУ_ЗАРП, ПРО_НОМ, ПРОЕКТ_РУК} (рис. 6.1). Очевидно, что если СЛУ_НОМ является первичным ключом отношения СЛУЖАЩИЕ, то для этого отношения справедлива функциональная зависимость (Functional Dependency – FD) СЛУ_НОМ->СЛУ_ИМЯ.

На самом деле, для тела отношения СЛУЖАЩИЕ_ПРОЕКТЫ в том виде, в котором оно показано на рис. 6.1, выполняются еще и следующие FD (1):

СЛУ_НОМ	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ	ПРОЕКТ_РУК
2934	Иванов	22400.00	1	Иванов
2935	Петров	29600.00	1	Иванов
2936	Сидоров	18000.00	1	Иванов
2937	Федоров	20000.00	1	Иванов
2938	Иванова	22000.00	1	Иванов
2939	Сидоренко	18400.00	2	Иваненко
2940	Федоренко	20400.00	2	Иваненко
2941	Иваненко	22600.00	2	Иваненко

Рис. 6.1. Пример возможного тела отношения СЛУЖАЩИЕ_ПРОЕКТЫ

СЛУ_НОМ->СЛУ_ИМЯ
СЛУ_НОМ->СЛУ_ЗАРП
СЛУ_НОМ->ПРО_НОМ
СЛУ_НОМ->ПРОЕКТ_РУК
{СЛУ_НОМ, СЛУ_ИМЯ}->СЛУ_ЗАРП
{СЛУ_НОМ, СЛУ_ИМЯ}->ПРО_НОМ
{СЛУ_НОМ, СЛУ_ИМЯ}->{СЛУ_ЗАРП, ПРО_НОМ}
…
ПРО_НОМ->ПРОЕКТ_РУК и т.д.

Поскольку имена всех служащих различны, то выполняются и такие FD (2):

СЛУ_ИМЯ->СЛУ_НОМ
СЛУ_ИМЯ->СЛУ_ЗАРП
СЛУ_ИМЯ->ПРО_НОМ и т.д.

Более того, для примера на рис. 6.1 выполняется и FD (3):

СЛУ_ЗАРП->ПРО_НОМ

Однако заметим, что природа FD группы (1) отличается от природы FD групп (2) и (3). Логично предположить, что идентификационные номера служащих должны быть всегда различны, а у каждого проекта имеется только один руководитель. Поэтому FD группы (1) должны быть верны для любого допустимого значения переменной

отношения СЛУЖАЩИЕ_ПРОЕКТЫ и могут рассматриваться как **инварианты**, или **ограничения целостности** этой переменной отношения.
FD группы (2) базируются на менее естественном предположении о том, что имена всех служащих различны. Это соответствует действительности для примера из рис. 6.1, но возможно, что с течением времени FD группы (2) не будут выполняться для какого-либо значения переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ.
Наконец, FD группы (3) основана на совсем неестественном предположении, что никакие двое служащих, участвующие в разных проектах, не получают одинаковую зарплату. Опять же, данное предположение верно для примера из рис. 6.1, но, скорее всего, это случайное совпадение.
В дальнейшем нас будут интересовать только те функциональные зависимости, которые должны выполняться для всех возможных значений переменных отношений. Заметим, что если атрибут A отношения r является возможным ключом, то для любого атрибута B этого отношения всегда выполняется FD A->B (в группе (1) к этим FD относятся все FD, детерминантом которых является СЛУ_НОМ). Обратите внимание, что наличие в отношении СЛУЖАЩИЕ_ПРОЕКТЫ FD ПРО_НОМ->ПРОЕКТ_РУК приводит к некоторой **избыточности** этого отношения. Имя руководителя проекта является характеристикой проекта, а не служащего, но в нашем случае содержится в теле отношения столько раз, сколько служащих работает над проектом.
Итак, мы будем иметь дело с FD, которые выполняются для всех возможных состояний тела соответствующего отношения и могут рассматриваться как ограничения целостности. Как показывает (неполный) список (1), таких зависимостей может быть очень много. Поскольку они трактуются как ограничения целостности, за их соблюдением должна следить СУБД. Поэтому важно уметь сократить набор FD до минимума, поддержка которого гарантирует выполнение всех зависимостей. Мы займемся этим в следующих подразделах.

FD A->B называется **тривиальной**, если $A \supseteq B$ (т. е. множество атрибутов A включает множество B или совпадает с множеством B).
Очевидно, что любая тривиальная FD всегда выполняется. Например, в отношении СЛУЖАЩИЕ_ПРОЕКТЫ всегда выполняется FD {СЛУ_ЗАРП, ПРО_НОМ}->СЛУ_ЗАРП. Частным случаем тривиальной FD является A->A.
Поскольку тривиальные FD выполняются всегда, их нельзя трактовать как ограничения целостности, и поэтому они не представляют интереса с практической точки зрения. Однако в теоретических рассуждениях их наличие необходимо учитывать.

Замыканием множества FD S является множество FD S^+, включающее все FD, логически выводимые из FD множества S .
Для начала приведем два примера FD, из которых следуют (или **выводятся**) другие FD. Будем снова пользоваться отношением СЛУЖАЩИЕ_ПРОЕКТЫ. Для этого отношения выполняется, например, FD СЛУ_НОМ->{СЛУ_ЗАРП, ПРО_НОМ}. Из этой FD выводятся FD СЛУ_НОМ->СЛУ_ЗАРП и СЛУ_НОМ->ПРО_НОМ.
В отношении СЛУЖАЩИЕ_ПРОЕКТЫ имеется также пара FD СЛУ_НОМ->ПРО_НОМ и ПРО_НОМ->ПРОЕКТ_РУК. Из них выводится FD СЛУ_НОМ->ПРОЕКТ_РУК. Заметим, что FD вида СЛУ_НОМ->ПРОЕКТ_РУК называются транзитивными, поскольку ПРОЕКТ_РУК зависит от СЛУ_НОМ "транзитивно", через ПРО_НОМ.

FD A->C называется **транзитивной**, если существует такой атрибут B, что имеются функциональные зависимости A->B и B->C и отсутствует функциональная зависимость C->A .
Подход к решению проблемы поиска замыкания S^+ множества FD S впервые предложил Вильям Армстронг[1]. Им был предложен набор правил вывода новых FD из существующих (эти правила обычно называют аксиомами Армстронга, хотя справедливость правил доказывается на основе определения FD). Обычно принято формулировать эти правила вывода в следующей форме. Пусть A, B и C являются (в общем случае, составными) атрибутами отношения r. Множества A, B и C могут иметь непустое пересечение. Для краткости будем обозначать через AB A UNION B. Тогда:

1. если $B \in A$, то A->B (рефлексивность);
2. если A->B, то AC->BC (дополнение);
3. если A->B и B->C, то A->C (транзитивность).

Истинность первой аксиомы Армстронга следует из того, что при $B \in A$ FD A->B является тривиальной.
Справедливость второй аксиомы докажем от противного. Предположим, что FD AC->BC не соблюдается. Это означает, что в некотором допустимом теле отношения найдутся два кортежа t1 и t2, такие, что t1 {AC} = t2 {AC} (a), но $t1\{BC\} \neq t2\{BC\}$ (b) (здесь t {A} обозначает проекцию кортежа t на множество атрибутов A). По аксиомерефлексивности из равенства (a) следует, что t1 {A} = t2 {A}. Поскольку имеется FD A->B, должно соблюдаться равенство t1 {B} = t2 {B}. Тогда из неравенства (b) следует, что $t1\{C\} \neq t2\{C\}$, что противоречит наличию тривиальной FD AC->C. Следовательно, предположение об отсутствии FD AC->BC не является верным, и справедливость второй аксиомы доказана.
Аналогично докажем истинность третьей аксиомы Армстронга. Предположим, что FD A->C не соблюдается. Это означает, что в некотором допустимом теле отношения найдутся два кортежа t1 и t2, такие, что t1 {A} = t2 {A}, но $t1\{C\} \neq t2\{C\}$. Но из наличия FD A->B следует, что t1 {B} = t2 {B}, а потому из наличия FD B->C следует, что t1 {C} = t2 {C}. Следовательно, предположение об отсутствии FD A->C не является верным, и справедливость третьей аксиомы доказана.
Можно доказать, что система правил вывода Армстронга **полна** и **совершенна (sound and complete)** в том смысле, что для данного множества FD S любая FD, потенциально выводимая из S, может быть выведена на основе аксиом Армстронга, и применение этих аксиом не может привести к выводу лишней FD. Тем не менее Дейт по практическим соображениям предложил расширить базовый набор правил вывода еще пятью правилами:

1. A->A (самодетерминированность) – прямо следует из правила (1);
2. если A->BC, то A->B и A->C (декомпозиция) – из правила (1) следует, что BC->B ; по правилу (3) A->B ; аналогично, из BC->C и правила (3) следует A->C ;
3. если A->B и A->C, то A->BC (объединение) – из правила (2) следует, что A->AB и AB->BC ; из правила (3) следует, что A->BC ;
4. если A->B и C->D, то AC->BD (композиция) – из правила (2) следует, что AC->BC и BC->BD ; из правила (3) следует, что AC->BD ;

5. если A->BC и B->D, то A->BCD (накопление) – из правила (2) следует, что BC->BCD ; из правила (3) следует, что A->BCD.

Пусть заданы отношение r, множество Z атрибутов этого отношения (подмножество заголовка r, или составной атрибут r) и некоторое множество FD S, выполняемых для r. Тогда **замыканием Z над S** называется наибольшее множество Z^+ таких атрибутов Y отношения r, что FD Z->Y входит в S^+ .
Алгоритм вычисления Z^+ очень прост. Один из его вариантов показан на рис. 6.2.

```
K := 0; Z[0] := Z;
DO
   K := K+1;
   Z[K] := Z[K-1];
   FOR EACH FD A→B IN S DO
      IF A⊆Z[K] THEN Z[K] := (Z[K] UNION B) END DO;
UNTIL Z[K] = Z[K-1];
Z+ := Z[K];
```

Рис. 6.2. Алгоритм построения замыкания атрибутов над заданным множеством FD
Докажем корректность алгоритма по индукции. На нулевом шаге Z[0] = Z, FD Z->Z[I], очевидно, принадлежит S^+ (тривиальная FD "выводится" из любого множества FD). Пусть для некоторого K выполняется FD Z->Z[K], и пусть мы нашли в S такую FD A->B, что $A \in Z[K]$. Тогда можно представить Z[K] в виде AC, и, следовательно, выполняется FD Z->AC. Но по правилу (8) мы имеем FD Z->ACB, т.е. FD Z->(Z[K] UNION B) входит во множество S^+, что переводит нас на следующий шаг индукции.
Пусть для примера имеется отношение с заголовком {A, B, C, D, E, F} и заданным множеством FD S = {A->D, AB->E, BF->E, CD->F, E->C}. Пусть требуется найти $\{AE\}^+$ над S. На первом проходе тела цикла DO Z[1] равно AE. В теле цикла FOR EACHбудут найдены FD A->D и E->C, и в конце цикла Z[1] станет равным ACDE. На втором проходе тела цикла DO при Z[2], равном ACDE, в теле цикла FOR EACH будет найдена FD CD->F, и в конце цикла Z[2] станет равным ACDEF. Следующий проход тела цикла DO не изменит Z[3], и Z^+ ($\{AE\}^+$) будет равно ACDEF.
Алгоритм построения замыкания множества атрибутов Z над заданным множеством FD S помогает легко установить, входит ли заданная FD Z->B в замыкание S^+.
Очевидно, что необходимым и достаточным условием для этого является $B \subset Z^+$, т. е. вхождение составного атрибута B в замыкание Z^2.
Суперключом отношения r называется любое подмножество K заголовка r, включающее, по меньшей мере, хотя бы один возможный ключ r .
Одно из следствий этого определения состоит в том, что подмножество K заголовка отношения r является суперключом тогда и только тогда, когда для любого атрибута A (возможно, составного) заголовка отношения r выполняется FD K->A. В терминах замыкания множества атрибутов K является суперключом тогда и только тогда, когда K^+ совпадает с заголовком r.

---

9. Минимальное множество функциональных зависимостей.

Минимальное множество ФЗ (ФЗ – функциональные зависимости)
Функциональная зависимость между атрибутами (множествами атрибутов) X и Y означает, что для любого допустимого набора кортежей в данном отношении: если два кортежа совпадают по значению X, то они совпадают по значению Y.
Например, если значение атрибута «Название компании» — Canonical Ltd, то значением атрибута «Штаб-квартира» в таком кортеже всегда будет Millbank Tower, London, United Kingdom.
Обозначение: {X} -> {Y}.
Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).
Атрибут — свойство некоторой сущности. Часто называется полем таблицы.

Множество ФЗ X называется минимальным тогда и только тогда, когда удовлетворяет следующим свойствам:

- правая часть любой ФЗ из X является множеством из одного атрибута (простым атрибутом);
- детерминант каждой ФЗ из X обладает свойством минимальности: удаление любого атрибута из детерминанта приводит к невозможности построить набор ФЗ, эквивалентный исходному;

Y зависит от X
X - детерминант для Y

- удаление любой ФЗ из X приводит к невозможности построения множества зависимостей, эквивалентных X.

СЛУ_НОМ	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ	ПРОЕКТ_РУК
18	Кузнецов	14160.00	1	Кузнецов
19	Афонин	16467.00	1	Кузнецов
20	Родригес	18160.00	1	Кузнецов
21	Альваро	34150.00	1	Кузнецов
22	Мозгов	24160.00	1	Кузнецов
23	Игнатьев	16560.00	2	Петроченко
24	Медведев	74054.00	2	Петроченко
25	Петроченко	53200.00	2	Петроченко

Минимальное множество ФЗ для примера
Пример ФЗ:

- СЛУ_НОМ→СЛУ_ИМЯ
- СЛУ_НОМ→СЛУ_ЗАРП
- СЛУ_НОМ→ПРО_НОМ
- ПРО_НОМ→ПРОЕКТ_РУК

Важные замечания

- если атрибут A отношения является возможным ключом, то для любого атрибута B всегда выполняется ФЗ A→B
- если в отношении есть ФЗ, детерминантом которой является неключевой атрибут, то отношение избыточно (ПРО_НОМ→ПРОЕКТ_РУК: руководитель - свойство проекта, но не атрибут сотрудника)

Нахождение минимального множества ФЗ

- Используя декомпозицию, исходное множество S приводится к эквивалентному множеству S1, правые части ФЗ которого содержат только одноэлементные множества (простые атрибуты)

Декомпозиция — научный метод, использующий структуру задачи и позволяющий заменить решение одной большой задачи решением серии меньших задач.

- Для каждой ФЗ, где в детерминанте более одного атрибута последовательно производится попытка удаления каждого атрибута с проверкой соблюдения эквивалентности
- Производится попытка удаления каждой ФЗ из S с проверкой эквивалентности

Декомпозиция отношений

- Проверка - соединением
- Не должно быть потерянных кортежей

---

10. Задача проектирования БД. Процесс проектирования на основе нормализации.

Задачи проектирования

- Отображение объектов предметной области в объекты модели данных с сохранением семантики
- Обеспечение эффективного доступа к данным

Процесс проектирования на основе нормализации

- Неформальное описание предметной области
- Формирование (интуиция) некоторого набора отношений для отображения объектов предметной области
- Преобразование исходных отношений в схему, отвечающую некоторому набору требований

Отношение — конечное множество кортежей (таблица).

Схема отношения — конечное множество атрибутов, определяющих некоторую сущность. Иными словами, это структура таблицы, состоящей из конкретного набора полей.
Атрибут — свойство некоторой сущности. Часто называется полем таблицы.

Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).

Процесс проектирования БД с использование метода НФ является итерационным и заключается в последовательном переводе отношения из 1НФ в НФ более высокого порядка по определенным правилам. Каждая следующая НФ ограничивается определенным типом функциональных зависимостей и устранением соответствующих аномалий при выполнении операций над отношениями БД, а также сохранении свойств предшествующих НФ.

Нормализация
Преобразование схемы БД к виду, удовлетворяющему требованиям нормальных форм.
Зачем:

- исключение избыточности
- устранение аномалий обновления
- "хорошее" отображение требований предметной области
- "простое" использование ограничений целостности БД

В частности, в процессе нормализации из отношений удаляются ФЗ детерминантами которых не являются возможные ключи.

---

11. Аномалии добавления, изменения, удаления. Первая, вторая нормальная форма.

Нормализация
Преобразование схемы БД к виду, удовлетворяющему требованиям нормальных форм.
Зачем:

- исключение избыточности
- устранение аномалий обновления
- "хорошее" отображение требований предметной области
- "простое" использование ограничений целостности БД

В частности, в процессе нормализации из отношений удаляются ФЗ детерминантами которых не являются возможные ключи.
Нормальная форма — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

Метод нормальных форм (НФ) состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений.

Цель нормализации: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей(строк таблицы). В идеале при нормализации надо добиться, чтобы любое значение хранилось в базе в одном экземпляре, причем значение это не должно быть получено расчетным путем из других данных, хранящихся в базе.

Аномалией называется такая ситуация в таблице БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.

Аномалии-модификации проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.

Аномалии-удаления — при удалении какого либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью.

Аномалии-добавления возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.

Атрибут — свойство некоторой сущности. Часто называется полем таблицы.
Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).

Первая нормальная форма

В любом кортеже только значения атрибутов атомарны. Выполняется для реляционных БД.
Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Первая нормальная форма:

- запрещает повторяющиеся столбцы (содержащие одинаковую по смыслу информацию)
- запрещает множественные столбцы (содержащие значения типа списка и т.п.)
- требует определить первичный ключ для таблицы, то есть тот столбец или комбинацию столбцов, которые однозначно определяют каждую строку

Пример для нормализации

Таблица 2: Служащие, проекты, задачи

СЛУ_НОМ	СЛУ_УРОВ	СЛУ_ЗАРП	ПРО_НОМ	СЛУ_ЗАДАН
18	2	16467.00	1	A
19	3	34150.00	1	B
20	1	14160.00	1	C
21	1	14160.00	1	D
18	2	16467.00	2	D
19	3	34150.00	2	C
20	1	14160.00	2	B
21	1	14160.00	2	A

Ограничения в примере:

- уровень служащего определяет размер его заработной платы
- можно участвовать в нескольких проектах
- не более одного задания на сотрудника в проекте

Зависимости в примере:
Ключ:СЛУ_НОМ, ПРО_НОМ

Зависимости:

- СЛУ_НОМ→СЛУ_ЗАРП
- СЛУ_НОМ→СЛУ_УРОВ
- СЛУ_УРОВ→СЛУ_ЗАРП
- СЛУ_НОМ, ПРО_НОМ→СЛУ_ЗАДАН

Аномалии обновления

- Добавление: нельзя добавить служащего без проекта
- Удаление: нельзя оставить служащего без проекта
- Изменение: при изменении уровня или зарплаты требует обновления набора записей

Декомпозиция

Таблица 3: Служащие

СЛУ_НОМ	СЛУ_УРОВ	СЛУ_ЗАРП
18	2	16467.00
19	3	34150.00
20	1	14160.00
21	1	14160.00

Таблица 4: Служ, проекты, задачи

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
18	1	A
19	1	B
20	1	C
21	1	D
18	2	D
19	2	C
20	2	B
21	2	A

Что с аномалиями теперь?

- Добавление: можно добавить служащего без проекта
- Удаление: можно удалить запись об участии в проекте, не удаляя запись о сотруднике
- Изменение: при изменении уровня или зарплаты нужно изменить только одну запись

Еще пример

Например, есть таблица «Автомобили»:

Фирма	Модели
BMW	M5, X5M, M1
Nissan	GT-R

Нарушение нормализации 1НФ происходит в моделях BMW, т.к. в одной ячейке содержится список из 3 элементов: M5, X5M, M1, т.е. он не является атомарным. Преобразуем таблицу к 1НФ:

Фирма	Модели
BMW	M5
BMW	X5M
BMW	M1
Nissan	GT-R

Вторая нормальная форма

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме и каждый неключевой атрибут функционально полно зависит от ключа. Неключевой атрибут функционально полно зависит от составного ключа если он функционально зависит от всего ключа в целом, но не находится в функциональной зависимости от какого-либо из входящих в него атрибутов.

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Вторая нормальная форма требует, чтобы неключевые столбцы таблиц зависили от первичного ключа в целом, но не от его части. Маленькая ремарочка: если таблица находится в первой нормальной форме и первичный ключ у нее состоит из одного столбца, то она автоматически находится и во второй нормальной форме.

К 1 примеру из первой нормальной формы!
Снова аномалии?

- Добавление: нельзя добавить уровень без сотрудника
- Удаление: если уволить всех сотрудников с некоторым разрядом, то пропадет информация о разряде
- Изменение: при изменении зарплаты разряда нужно изменить несколько записей

Снова декомпозиция

Таблица 5: Служащие1

СЛУ_НОМ	СЛУ_УРОВ
18	2
19	3
20	1
21	1

Таблица 6: Уровни

СЛУ_УРОВ	СЛУ_ЗАРП
2	16467.00
3	34150.00
1	14160.00

Еще пример

Например, дана таблица:

Модель	Фирма	Цена	Скидка
M5	BMW	5500000	5%
X5M	BMW	6000000	5%
M1	BMW	2500000	5%
GT-R	Nissan	5000000	10%

Таблица находится в первой нормальной форме, но не во второй. Цена машины зависит от модели и фирмы. Скидка зависит от фирмы, то есть зависимость от первичного ключа неполная. Исправляется это путем декомпозиции на два отношения, в которых не ключевые атрибуты зависят от ПК.

Модель	Фирма	Цена
M5	BMW	5500000
X5M	BMW	6000000
M1	BMW	2500000
GT-R	Nissan	5000000

Фирма	Скидка
BMW	5%
Nissan	10%

---

12. Третья нормальная форма. Нормальная форма Бойса-Кодда.

Аномалией называется такая ситуация в таблице БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.

Аномалии-модификации проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.

Аномалии-удаления — при удалении какого либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью.

Аномалии-добавления возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.

Атрибут — свойство некоторой сущности. Часто называется полем таблицы.
Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).

Третья нормальная форма
Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Аномалии в третьей нормальной форме

Таблица 7: Служащие, проекты, задачи1

СЛУ_НОМ	СЛУ_ИМЯ	ПРО_НОМ	СЛУ_ЗАДАН
18	Кузнецов	1	A
19	Родригес	1	B
18	Кузнецов	2	D
19	Родригес	2	C

Особенности примера

- Имена уникальны, а значит: СЛУ_НОМ→СЛУ_ИМЯ и СЛУ_ИМЯ→СЛУ_НОМ
- Два возможных ключа: СЛУ_НОМ, ПРО_НОМ и СЛУ_ИМЯ, ПРО_НОМ
- Аномалия обновления: при изменении фамилии нужно менять несколько записей

Еще пример

Рассмотрим таблицу:

Модель	Магазин	Телефон
BMW	Риал-авто	87-33-98
Audi	Риал-авто	87-33-98
Nissan	Некст-Авто	94-54-12

Таблица находится во 2НФ, но не в 3НФ.
В отношении атрибут «Модель» является первичным ключом. Личных телефонов у автомобилей нет, и телефон зависит исключительно от магазина.
Таким образом, в отношении существуют следующие функциональные зависимости: Модель → Магазин, Магазин → Телефон, Модель → Телефон.
Зависимость Модель → Телефон является транзитивной, следовательно, отношение не находится в 3НФ.
В результате разделения исходного отношения получаются два отношения, находящиеся в 3НФ:

Риал-авто 87-33-98
Риал-авто 87-33-98
Некст-Авто 94-54-12

Модель	Магазин
BMW	Риал-авто
Audi	Риал-авто
Nissan	Некст-Авто

Нормальная форма Бойса-Кодда
Отношение находится в нормальной форме Бойса-Кодда тогда и только тогда, когда любая выполняемая для этой переменной отношения нетривиальная и минимальная ФЗ имеет в качестве детерминанта некоторый потенциальный ключ.
Y зависит от X
X - детерминант для Y

Нормальная форма Бойса-Кодда требует, чтобы в таблице был только один потенциальный первичный ключ. Чаще всего у таблиц, находящихся в третьей нормальной форме, так и бывает, но не всегда. Если обнаружился второй столбец

(комбинация столбцов), позволяющий однозначно идентифицировать строку, то для приведения к нормальной форме Бойса-Кодда такие данные надо вынести в отдельную таблицу.

Определение 3НФ не совсем подходит для следующих отношений:
1) отношение имеет две или более потенциальных ключа;
2) два и более потенциальных ключа являются составными;
3) они пересекаются, т.е. имеют хотя бы один атрибут.

Для отношений, имеющих один потенциальный ключ (первичный), НФБК является 3НФ.

Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.

Предположим, рассматривается отношение, представляющее данные о бронировании стоянки на день:

Номер стоянки	Время начала	Время окончания	Тариф
1	09:30	10:30	Бережливый
1	11:00	12:00	Бережливый
1	14:00	15:30	Стандарт
2	10:00	12:00	Премиум-B
2	12:00	14:00	Премиум-B
2	15:00	18:00	Премиум-A

Тариф имеет уникальное название и зависит от выбранной стоянки и наличии льгот, в частности:

- «Бережливый»: стоянка 1 для льготников
- «Стандарт»: стоянка 1 для не льготников
- «Премиум-А»: стоянка 2 для льготников
- «Премиум-В»: стоянка 2 для не льготников.

Таким образом, возможны следующие составные первичные ключи: {Номер стоянки, Время начала}, {Номер стоянки, Время окончания}, {Тариф, Время начала}, {Тариф, Время окончания}.

Отношение находится в 3НФ. Требования второй нормальной формы выполняются, так как все атрибуты входят в какой-то из потенциальных ключей, а неключевых атрибутов в отношении нет. Также нет и транзитивных зависимостей, что соответствует требованиям третьей нормальной формы. Тем не менее, существует функциональная зависимость Тариф → Номер стоянки, в которой левая часть (детерминант) не является потенциальным ключом отношения, то есть отношение не находится в нормальной форме Бойса — Кодда.

Недостатком данной структуры является то, что, например, по ошибке можно приписать тариф «Бережливый» к бронированию второй стоянки, хотя он может относиться только к первой стоянки.

Можно улучшить структуру с помощью декомпозиции отношения на два и добавления атрибута Имеет льготы, получив отношения, удовлетворяющие НФБК (подчёркнуты атрибуты, входящие в первичный ключ.):

Тарифы

Тариф	Номер стоянки	Имеет льготы
Бережливый	1	Да
Стандарт	1	Нет
Премиум-А	2	Да
Премиум-В	2	Нет

Бронирование

Тариф	Время начала	Время окончания
Бережливый	09:30	10:30
Бережливый	11:00	12:00
Стандарт	14:00	15:30
Премиум-В	10:00	12:00
Премиум-В	12:00	14:00
Премиум-А	15:00	18:00

Еще пример

Опять декомпозиция

Декомпозиция — научный метод, использующий структуру задачи и позволяющий заменить решение одной большой задачи решением серии меньших задач.

Таблица 8: Служащие2

СЛУ_НОМ	СЛУ_ИМЯ
18	Кузнецов
19	Родригес

Таблица 9: Служащие, проекты, задания2

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
18	1	A
19	1	B
18	2	D
19	2	C

---

13. Обоснование необходимости использования семантических моделей для проектирования БД. Концептуальная модель данных. Требования к концептуальной модели данных.

Чего хотим добиться используя семантическую модель:

- Отображение объектов предметной области
- Отображение связей объектов

- Достаточность представления предметной области для пользователей
- Формирование ограничений целостности
- Минимизация аномалий

Ожидаемый результат:

- Формальное описание предметной области в какой-либо нотации

(Для таких, как я: Нотация - система условных обозначений, принятая в какой-либо области знаний или деятельности. Включает множество символов, используемых для представления понятий и их взаимоотношений, составляющее *алфавит нотации*, а также правила их применения.)

- Автоматическое преобразование модели в РБД
- Проверка противоречий в модели
- Реверс-инжиниринг
- Наличие средств отображения для модели, хорошо передающих семантику

Реляционная модель данных ограничена в следующих аспектах:

- Модель не предоставляет достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к упоминавшейся нами проблеме представления ограничений целостности.
- Для многих приложений трудно моделировать предметную область на основе плоских таблиц. В ряде случаев на самой начальной стадии проектирования проектировщику приходится производить насилие над собой, чтобы описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.
- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.
- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области ("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

Концептуальной моделью будем называть некоторое формальное описание предметной области на основе некоторой семантической МД.

Зачем нужно:

- Описание БП предметной области
- Формирование выводов и рекомендаций по изменению БП и развитию предметной области
- Основа для логического проектирования БД
- Средство документирования проекта

Требования к концептуальной модели:

- Корректное отображение предметной области
- Отсутствие внутренних противоречий

- Восприятие и однозначная интерпретация пользователями
- Развитие
- Декомпозиция

---

14. Этапы концептуального проектирования.

Заметка: в целом проектирование может вестись на основе семантической модели данных. концептуальное проектирование лишь один пункт из проектирования на основе семантической МД.

Так для проектирования на основе семантической МД, мы:

- Формируем концептуальную схему данных
- Преобразуем ее к логической схеме данных на основе некоторой модели данных
- Преобразуем ее к физической схеме данных

Чтобы собрать мысли в одну кучу:

Концептуальная модель хранилища данных представляет собой описание главных (основных) сущностей и отношений между ними. Концептуальная модель является отражением предметных областей, в рамках которых планируется построение хранилища данных.

Логическая модель расширяет концептуальную путем определения для сущностей их атрибутов, описаний и ограничений, уточняет состав сущностей и взаимосвязи между ними.

Физическая модель данных описывает реализацию объектов логической модели на уровне объектов конкретной базы данных.

Этапы концептуального проектирования:

- Выделение сущностей
- Определение связей
- Атрибуты, типы данных, домены
- Ключи
- Ограничения целостности:
    - Допустимые значения
    - Разрешенные значения
    - Существующие значения

Выделение сущностей:

- Выяснение потребностей и представления предметной области для каждой роли пользователя
- Выделяются независимые типы сущностей
- Отбрасываются “ненужные”
- Именование типов сущностей

Заметка: Необходимо различать такие понятия, как тип сущности и экземпляр сущности. Понятие тип сущности относится к набору однородных личностей, предметов, событий или идей, выступающих как целое. Экземпляр сущности относится к конкретной вещи в наборе. Например, типом сущности может быть ГОРОД, а экземпляром – Москва, Киев и т.д. Предполагается, что гарантировано отличие экземпляров одного типа сущности друг от друга.

Определение связей:

- Определяются типы связей:
    - Один ко многим (У одного клиента может быть несколько телефонов)
    - Многие к одному
    - Многие ко многим (Одна книга могла быть написана несколькими авторами. Автор мог написать несколько книг)
    - Один к одному
- Мощность
    - 1
    - 0..1
    - 0..n
    - 1..n

Связи характеризуются степенью связи и классом принадлежности сущности к связи. Степень (мощность) связи - это отношение числа сущностей, участвующих в образовании связи. Например, "один-к-одному", "один-ко-многим", "многие-ко-многим". На уровне информационной модели допускается неопределенная или неразрешенная связь. Класс принадлежности сущности - это характер участия сущности в связи. Различают обязательные и необязательные классы принадлежности сущности к связи. Обязательным является такой класс принадлежности, когда экземпляры сущности участвуют в установлении связи в обязательном порядке. В противном случае сущность принадлежит к необязательному классу принадлежности. Для необязательного класса принадлежности сущности степень связи может быть равна нулю, т.е. экземпляр сущности можно связать с 0, 1 или несколькими экземплярами другой сущности. Для обязательного класса принадлежности степень связи не может равняться нулю.

Заметка: кажется, что типы связи и мощность в его презентации - это одно и то же. Снизу мелким текстом прикрепил некоторые пояснения. По сути, Мощность - это то, сколько сущностей зависит от выбранной сущности.(Грубо: если удалить какую-то строку из таблицы, необходимо из другой таблицы удалить лишь одну строку или n?)

Определение атрибутов:

- Выделяются атрибуты типов сущностей (грубо: атрибуты = поля у классов в Java, а типы сущностей = класс в Java)
- Выделяются атрибуты связей
- Простые и составные атрибуты (простой атрибут принадлежит одному типу данных и является неделимым, а составной состоит из нескольких, которые могут иметь разные типы)
- Определение типов данных, доменов (домен - область допустимых значений)

- Именование атрибутов

Выбор ключей:

- Определяются ключевые атрибуты
- Выбирается первичный ключ
    - Простота
    - Минимальность
    - Редкие изменения
- Для слабых сущностей вводятся синтетические ключевые атрибуты (Слабая сущность - сущность удовлетворяющая след условиям: 1) зависит от существования другой сущности, т.е не может существовать без сущности, с которой она связана; 2) ее первичный ключ частично или целиком произведен из порождающей сущности данной связи)

Финалом любого проектирования должна быть проверка. Так и здесь в концептуальном проектировании она есть.

Проверка:

- Наличие связей 1 к 1
- Наличие избыточных связей
- Проверка выполнимости пользовательских операций
- Обсуждение с пользователями

---

15. Модель Сущность-Связь.

Модель "Сущность-Связь" - одной из наиболее популярных семантических МД.

Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных. Среди множества разновидностей ER-моделей одна из наиболее развитых применяется в системе CASE фирмы ORACLE.

ER-модель оперирует следующими определениями:
Сущность - это 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 и может существовать между

двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

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.

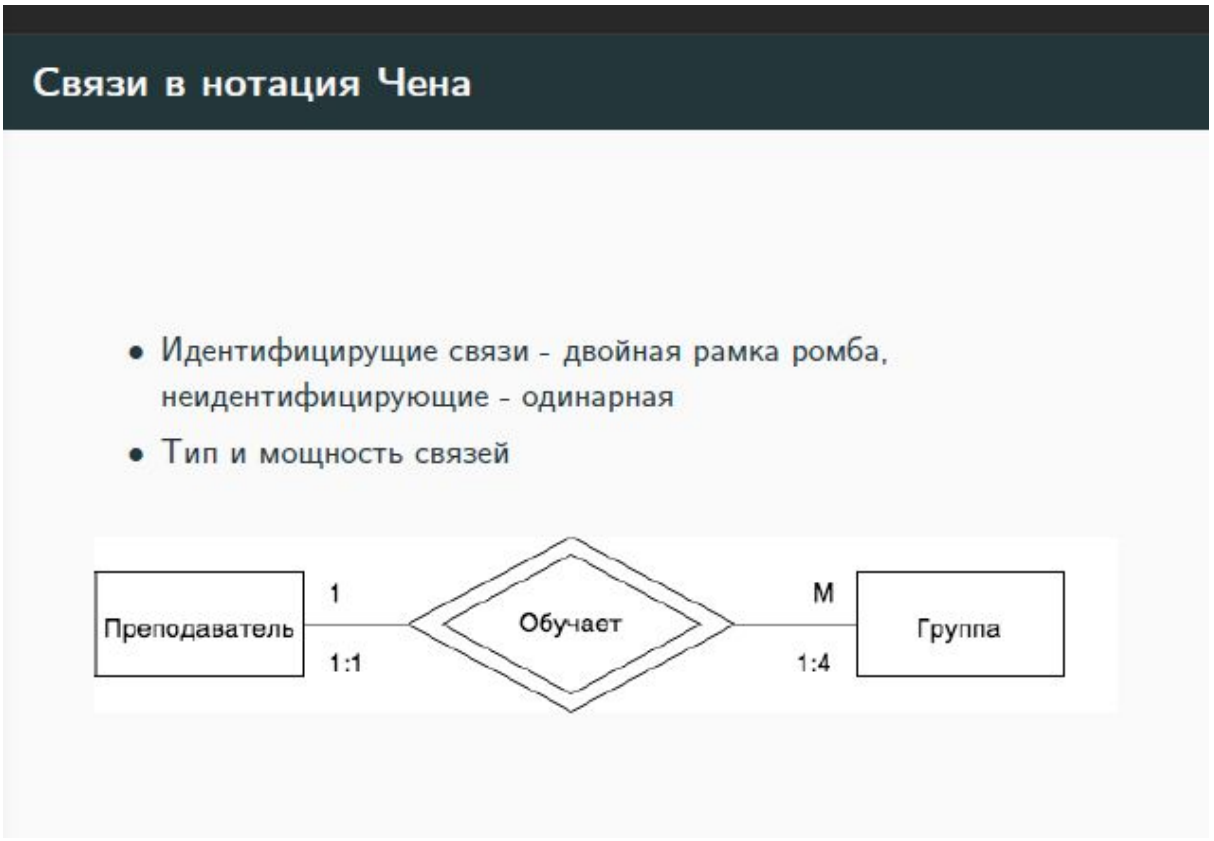
Домен - область допустимых значений.

Ограничения модели:

- сущности должны быть отличимы
- связи бинарные, типизированные

Независимая сущность - одинарная рамка, зависимая - двойная.
Многозначные атрибуты изображаются с двойным контуром.
Составные атрибуты отображаются в виде связанных атрибутов.
Производные атрибуты отображаются пунктирной линией.
Первичный ключ - двойное подчеркивание.

Связи:



Опять же {

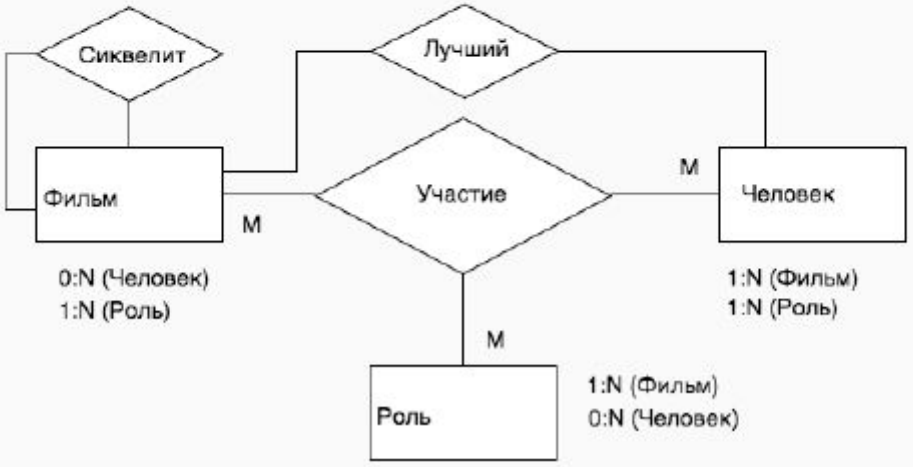
Связи характеризуются степенью связи и классом принадлежности сущности к связи. Степень (мощность) связи - это отношение числа сущностей, участвующих в образовании связи. Например, "один-к-одному", "один-ко-многим", "многие-ко-многим". На уровне информационной модели допускается неопределенная или неразрешенная связь. Класс принадлежности сущности - это характер участия сущности в связи. Различают обязательные и необязательные классы принадлежности сущности к связи. Обязательным является такой класс принадлежности, когда экземпляры сущности участвуют в установлении связи в обязательном порядке. В противном случае сущность принадлежит к необязательному классу принадлежности. Для необязательного класса принадлежности сущности степень связи может быть равна нулю, т.е. экземпляр сущности можно связать с 0, 1 или несколькими экземплярами другой сущности. Для обязательного класса принадлежности степень связи не может равняться нулю.

Заметка: кажется, что типы связи и мощность в его презентации - это одно и то же. Сверху мелким текстом прикрепил некоторые пояснения. По сути, Мощность - это то, сколько сущностей зависит от выбранной сущности.(Грубо: если удалить какую-то строку из таблицы, необходимо из другой таблицы удалить лишь одну строку или n?)

}

Степень связи определяется числом ассоциированных сущностей

- Унарные
- Бинарные
- Тернарные
- ...



Про степень связи - имеется в виду не мощность, а именно количество типов сущностей завязанных друг на друге. Так, на картинке “Роль” связана с “Фильм” и “Человек” (таким образом, степень связи - бинарная), “Фильм” связан с “Фильм” (рекурсивно, возможно какой-то фильм является сиквелом к выбранному фильму), с “Человек” и с “Роль” (т.о. степень связи - тернарная).

---

16. Супертипы и подтипы. Взаимоисключающие связи.

Моделирование взаимоисключающих классов объектов.

Взаимоисключающие классы объектов моделируются с помощью супертипов и подтипов.

Супертип – это класс объектов, который делится на взаимоисключающие подгруппы меньшего размера.

Супертип может иметь собственные свойства или просто использоваться как имя группы.

Подтип – это класс объектов, представляющий разбитую группу в рамках супертипа.

Каждый подтип неявно наследует все свойства и связи супертипа, кроме того, он может иметь свои собственные свойства и связи.

Подтипы должны быть **взаимоисключаемыми**, то есть объект одного подтипа **не может быть объектом другого подтипа**.

Свойства и связи, общие для подтипов можно описывать на уровне супертипов.

Уникальные идентификаторы, выявленные для супертипа, наследуются подтипом.

Замечание: **подтип – это не наследование, это эксклюзивность!**

При чтении ER-диаграммы подтипы читаются «или – или»: «в каждой ПОЗИЦИИ ДОКУМЕНТА может быть отражен или ФИЛЬМ или ИГРА или ПРОЧИЙ продукт проката».

Подтипы могут быть вложенными. Для ясности модели рекомендуется ограничиваться 2—3 уровнями вложения.

ПРИМЕР:

Рассмотрим описание фрагмента предметной области «Прокат видеофильмов и видеоигр». Необходимо хранить информацию о продуктах проката. Каждый продукт характеризуется кодом, названием, текстовым описанием. Для фильмов необходимо хранить такие свойства как категория фильма (код, название), длительность. Для игры – возраст ограничения, количество игроков. Кроме того, необходимо учитывать прочие продукты проката, но пока свойства их неизвестны.

Для описания фрагмента данной предметной области можно использовать супертип, объединяющий общие атрибуты и подтипы, отражающие эксклюзивность отдельных продуктов проката.

На рисунке представлено отображение предметной области в виде супертипа «ПРОДУКТ ПРОКАТА» и подтипов «ФИЛЬМ», «ИГРА», «ПРОЧЕЕ». Подтип «ФИЛЬМ» имеет собственную связь с классом объектов «КАТЕГОРИЯ». Каждый подтип, входящий в супертип может быть указан в позиции документа, с помощью которого фиксируется выдача продуктов проката.

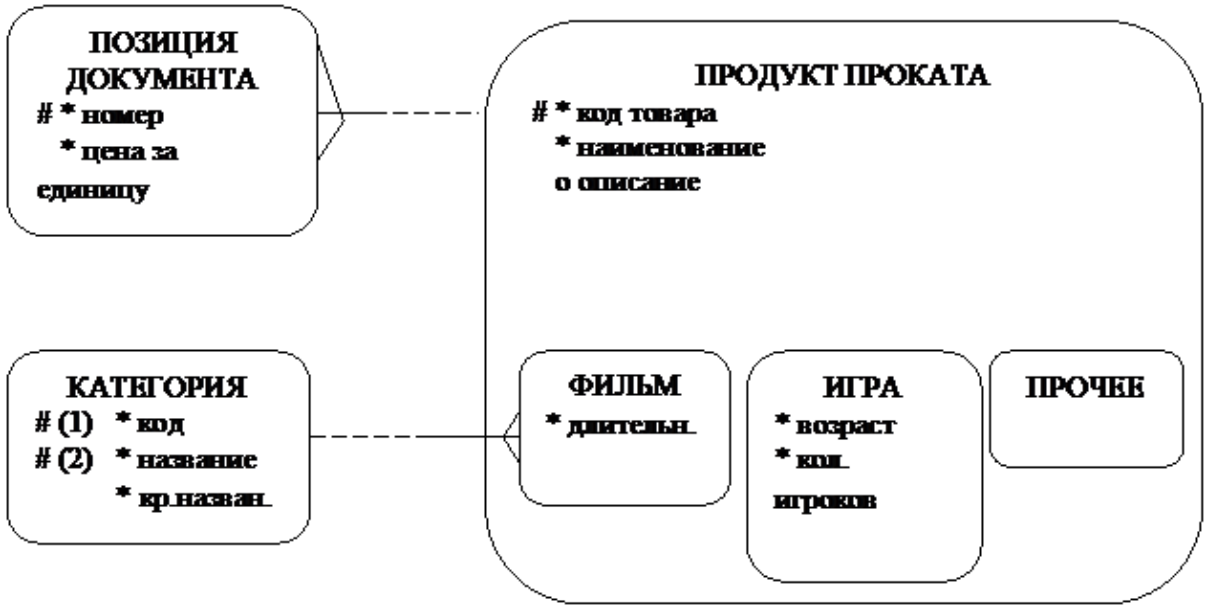


Рисунок — Пример супертипа и подтипов

Примеры фрагментов предметных областей с подтипами:

1) Супертип: ПИТОМЕЦ ЗООПАРКА, подтипы: МЛКОПИТАЮЩЕЕ, РЕПТИЛИЯ, РЫБЫ, ПТИЦЫ

2) ДОГОВОР НА ОБУЧЕНИЕ; ДОГОВОР ДВУХСТОРОННИЙ, ДОГОВОР ТРЕХСТОРОННИЙ

3) ЗАРПЛАТА ЗА МЕСЯЦ; НАЧИСЛЕНИЯ; УДЕРЖАНИЯ

2 Моделирование взаимоисключающих связей.

Взаимоисключающая связь – это такая ситуация, когда класс объектов имеет связь либо с классом объектов А, либо с классом объектов В. Обе связи могут быть действительными, но в разные моменты времени.

Взаимоисключаемость связей моделируется с помощью арка.

Арк – это элемент ER—диаграммы, построенной по методологии Ричарда Баркера, изображается в виде дуги, пересекающей входящие в арк взимоисключающие связи.

Связи, входящие в арк, помечаются кружочком. CASE-средство это делает автоматически.

Пример использования арка приведен на рисунке.

На рисунке изображено типовое представление адреса в предметной области. Связи, входящие в арк читаются с использованием союзов или—или, либо—либо. Например: «каждый АДРЕС должен относиться либо к ФИЗИЧЕСКОМУ, либо к ЮРИДИЧЕСКОМУ ЛИЦУ».

Правила использования арка:

— все концы связей в арке должны иметь одну и ту же опциональность, если это не так, то эксклюзивность связей не различается;

— связь может входить только в один арк;

— количество связей в арке может быть любым;

— связи в арке часто имеют одинаковые имена;

— связи, входящие в арк, должны идти от одного и того же класса объектов.

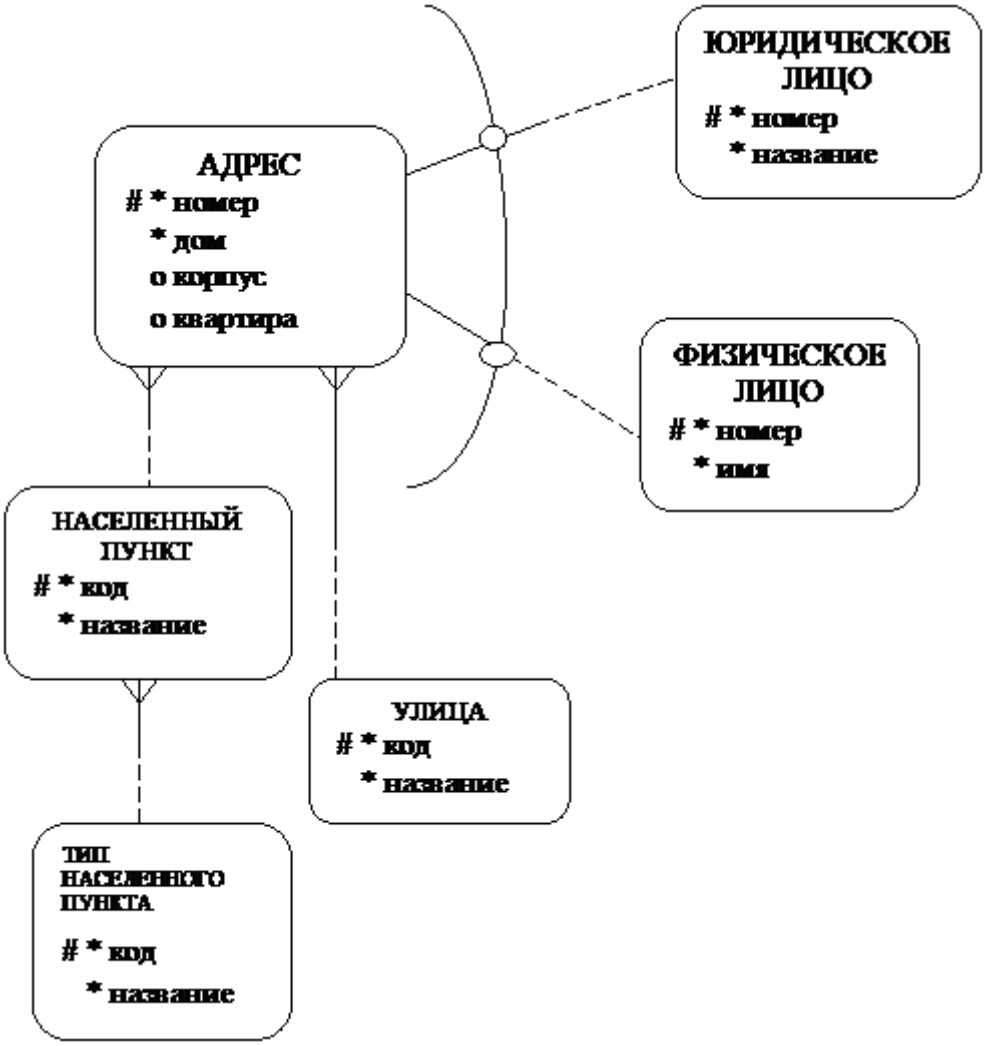


Рисунок — Пример использования арка.

Примеры фрагметов предметных областей

1) ДОГОВОР относится либо к ЮЛ, либо к ФЛ.

2) ЧЛЕНСКИЙ БИЛЕТ относится либо к ЮЛ, либо к ФЛ

3)

Для моделирования взаимоисключаемости могут быть использованы арки, подтипы, сочетание первого и второго. Например, бизнес правило «каждый Членский БИЛЕТ должен принадлежать либо ОРГАНИЗАЦИИ, либо ФИЗИЧЕСКОМУ ЛИЦУ», отображаемое с помощью арка, может быть несколько интерпретировано и отображено с помощью супертипа и подтипов: «каждый ЧЛЕНСКИЙ БИЛЕТ выдается КЛИЕНТУ, который может быть либо ФИЗИЧЕСКИМ ЛИЦОМ, либо ОГРАНИЗАЦИЕЙ».

Можно пользоваться любым удобным методом при условии, что он позволит правильно отразить потребности автоматизируемого предприятия.

17. Логическое проектирование. Преобразование ER-модели в схему реляционной БД.

Логическое проектирование БД - это процесс конструирования общей информационной модели предприятия на основе отдельных моделей данных пользователей, которая является независимой от особенностей реально используемой СУБД и других физических условий.

Этапы логического проектирования:

1. Преобразование локальной концептуальной модели данных в локальную логическую модель. (Удаление связей M: H, сложных связей, рекурсивных связей, связей с атрибутами, удаление множественных атрибутов.)
2. Определение набора отношений исходя из структуры локальной логической модели данных.
3. Проверка модели с помощью правил нормализации.
4. Проверка модели в отношении транзакций пользователей.
5. Создание диаграммы сущность-связь.
6. Определение требований поддержки целостности данных. (Обязательные данные, ограничения для доменов атрибутов, целостность сущностей (PK не может быть NULL), требования данного предприятия (бизнес-правила)).
7. Обсуждение разработанных локальных логических моделей данных с конечными пользователями.

Этап 2:

1. Слияние локальных моделей в единую глобальную модель данных (анализ имен сущностей и связей,PK).
2. Проверка глобальной логической модели данных (нормализация и транзакции).
3. Проверка возможностей расширения модели в будущем.
4. Создание окончательного варианта диаграммы сущность-связь
5. Обсуждение глобальной модели данных с пользователем.

Логическая модель описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью. Примеры понятий – «сотрудник», «отдел», «проект», «зарплата». Примеры взаимосвязей между понятиями – «сотрудник числится ровно в одном отделе», «сотрудник может выполнять несколько проектов».

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но *без привязки к конкретной СУБД*. Более того, логическая модель данных необязательно должна быть выражена средствами именно *реляционной* модели данных. Основным средством разработки логической модели данных в настоящий момент являются различные варианты ***ER-диаграмм*** (***Entity-Relationship***, ***диаграммы сущность-связь***). Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель даны.

Преобразование ER-модели в схему реляционной БД.

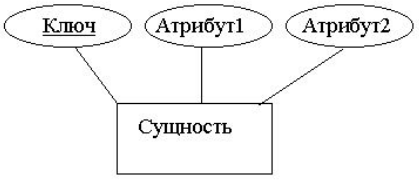
Преобразование ER-модели в реляционную модель

Дано: ER-модель.
Получить: набор таблиц (отношений) вида
Таблица (Ключ, Атрибут1, Атрибут2, …, АтрибутN)

Ключ может быть составным.
Удобно имена атрибутов в масштабе ER-модели сделать уникальными, тогда при построении реляционной модели их (почти никогда) не придется переименовывать.
Для краткости в примерах пропущены несущественные неключевые атрибуты.

I. Преобразование множеств сущностей (для краткости - сущностей)

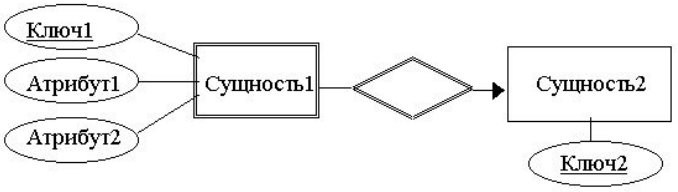
1. Преобразование обычной сущности



Обычная сущность преобразуется в отдельную таблицу, полями таблицы будут все атрибуты сущности:

Сущность (Ключ, Атрибут1, Атрибут2)

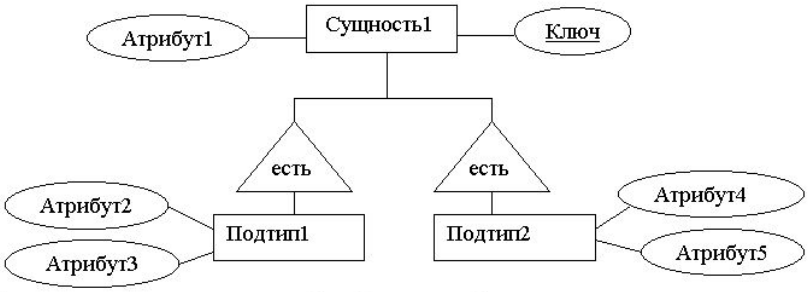
2. Преобразование слабой сущности



Слабая сущность преобразуется в отдельную таблицу, полями таблицы будут все атрибуты сущности плюс ключевые атрибуты всех сущностей, с помощью которых данная слабая сущность идентифицируется.
Ключевые поля всех родительских таблиц войдут в ключ дочерней таблицы. Для дочерней таблицы они будут называться **внешним ключом**.

Сущность1 (Ключ1, Ключ2, Атрибут1, Атрибут2)

3. Преобразование подтипов сущностей.



1 способ) Создается одна таблица, в которую помещают все атрибуты. Для того, чтобы указать, к какому подтипу относится объект, приходится вводить дополнительное поле-признак.

Сущность1(Ключ, Атрибут1, Атрибут2, Атрибут3, Атрибут4, Атрибут4, Признак)

Недостатком этого способа является то, что в таблице остается много незаполненных полей: для объекта подтипа 1 атрибуты 4 и 5, а для объекта подтипа 2 - атрибуты 2 и 3 останутся пустыми.

2 способ) Создается отдельная таблица для каждого подтипа. В нее включаются все атрибуты этого подтипа и все атрибуты надтипа.

Подтип1(Ключ, Атрибут1, Атрибут2, Атрибут3)
Подтип2(Ключ, Атрибут1, Атрибут4, Атрибут5)

Недостатком этого подхода является то, что подтипы теперь никак не связаны друг с другом.

3 способ) Создается одна таблица для надтипа, и по одной таблице для каждого подтипа, в которую включаются ключевые поля надтипа:

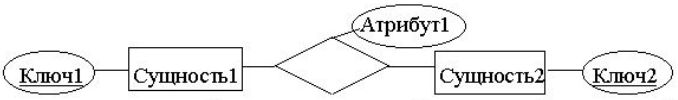
Сущность1(Ключ, Атрибут1)
Подтип1(Ключ, Атрибут2, Атрибут3)
Подтип2(Ключ, Атрибут4, Атрибут5)

Недостатком этого подхода является то, что информация о каждом объекте теперь раскидана по двум таблицам.

II. Преобразование связей

Для связей-двойных ромбов ничего делать не нужно, вся информация уже хранится в таблице слабой сущности.

1. Связь M:M

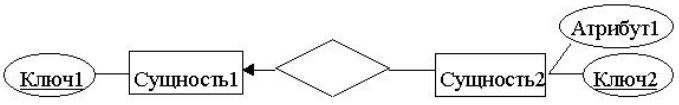


Создается новая таблица, содержащая ключевые поля каждой сущности, участвующей в связи, и собственные атрибуты связи, если таковые имеются. В названии обычно отражают, какие именно сущности связываются, или называют новую таблицу именем связи.

Сущ1Сущ2(Ключ1, Ключ2, Атрибут1).

ВНИМАНИЕ! Если собственный атрибут связи является ключевым, то на самом деле это уже не бинарная связь, а связь большей арности, т.е. этот атрибут может быть заменен на новую сущность, по направлению к которой существует связь "к одному". В этом случае нужно применять правила для тернарных и др. связей.

2. Связь 1:M



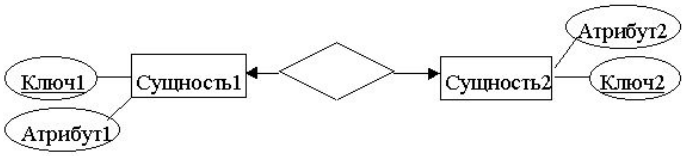
1 способ) Точно так же, как и в случае M:M, создается новая таблица, содержащая ключевые поля каждой сущности, участвующей в связи. В названии обычно отражают, какие именно сущности связываются, или называют новую таблицу именем связи.
Ключом будет ключ второй сущности.
[Этот способ предпочтительнее использовать в том случае, если связь не является "ровно к одному", то есть не все экземпляры сущностей участвуют в связи]

Сущ1Сущ2(Ключ1, Ключ2).

2 способ) Новая таблица не создается, а в таблицу дочерней сущности добавляют ключевые поля родительской сущности (в ключ дочерней сущности они входить не будут!). Ключевые поля родительской сущности представляют собой **внешний ключ** (foreign key) для дочерней сущности.
[Этот способ предпочтительнее использовать в том случае, если связь является связью "ровно к одному", то есть все экземпляры сущностей участвуют в связи. В этом случае поле внешнего ключа никогда не будет пустым]

Таблица дочерней сущности: **Сущность2(Ключ2, Атрибут1, Ключ1).**

3. Связь 1:1



1 способ) Точно так же, как и в случае M:M, создается новая таблица, содержащая ключевые поля каждой сущности, участвующей в связи. В названии обычно отражают, какие именно сущности связываются, или называют новую таблицу именем связи.
Ключом будет ключ любой сущности.
[Этот способ предпочтительнее использовать в том случае, если связь не является "ровно к одному", то есть не все экземпляры сущностей участвуют в связи]

Сущ1Сущ2(Ключ1, Ключ2) или **Сущ1Сущ2(Ключ1, Ключ2)**.

2 способ) Точно так же, как и во 2 случае 1:M, новая таблица не создается, а в таблицу одной из сущностей (будем считать ее дочерней) добавляют ключевые поля другой сущности (будем считать ее родительской).
[Если связь не является связью "ровно к одному" по отношению к родительской таблице, то есть не все экземпляры сущностей участвуют в связи, поле внешнего ключа в некоторых записях может быть пустым]

Таблица дочерней сущности: **Сущность1(Ключ1, Атрибут1, Ключ2)**,
или **Сущность2(Ключ2, Атрибут2, Ключ1)**.

3 способ) Две таблицы для сущностей, связанных 1:1, объединяются в одну. Ключом новой таблицы может быть комбинация ключей обеих таблиц. Если хотя бы в одном направлении связь "ровно к одному", то ключ этой сущности можно считать ключом объединенной таблицы.

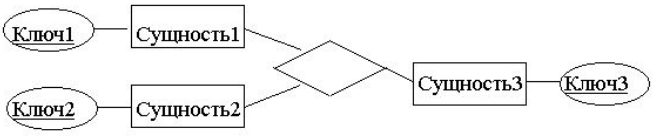
Таблицы **Сущность1(Ключ1, Атрибут1)** и **Сущность2(Ключ2, Атрибут2)** заменяются на
Сущ1Сущ2(Ключ1, Атрибут1, Ключ2, Атрибут2)
[или, возможно, **Сущ1Сущ2(Ключ1, Атрибут1, Ключ2, Атрибут2)**,
или **Сущ1Сущ2(Ключ1, Атрибут1, Ключ2, Атрибут2)**].

Примечание 1: Для связи сущности с самой собой применяются те же правила, но, так как одна и та же сущность участвует в связи дважды, ключевые поля должны войти в одну и ту же таблицу два раза. Поэтому приходится переименовывать один из ключей.

Рассмотрим связь 1:M, способ 2. Переименован внешний ключ.

Сущность1(Ключ1, Атрибут1, ЕщеОдинКлюч1).

Примечание 2: Для связей с арностью более 2 обычно применяется тот же способ, что и для бинарной связи M:M - создается новая таблица, содержащая ключевые поля всех связанных таблиц.

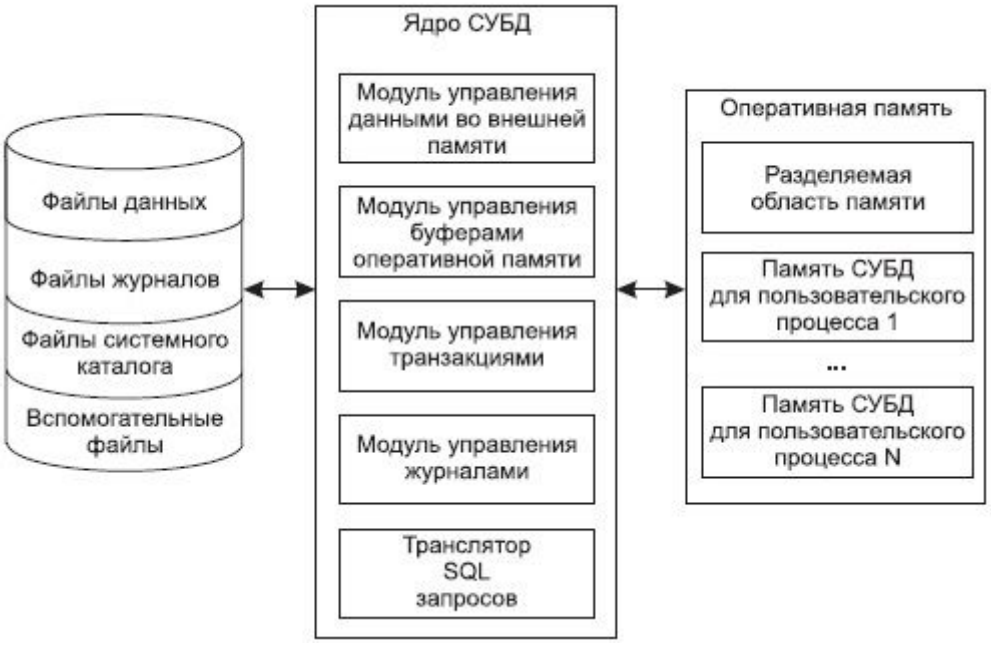


Сущ1Сущ2Сущ3(Ключ1, Ключ2, Ключ3).

Внутрь какой-то таблицы такие связи лучше не включать.

http://kek.ksu.ru/eos/bd/ER_to_RDB.html

18. Обобщенная архитектура многопользовательской SQL-СУБД. Компоненты и их назначение.



На рисунке выше изображена обобщенная структура. Здесь условно показано, что СУБД должна управлять внешней памятью, в которой расположены файлы с данными, файлы журналов и файлы системного каталога.

С другой стороны, СУБД управляет и оперативной памятью, в которой располагаются буфера с данными, буфера журналов, данные системного каталога, которые необходимы для поддержки целостности и проверки привилегий пользователей. Кроме того, в оперативной памяти во время работы СУБД располагается информация, которая соответствует текущему состоянию обработки запросов, там хранятся планы выполнения скомпилированных запросов и т. д.

Модуль управления внешней памятью обеспечивает создание необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например для ускорения доступа к данным в некоторых случаях (обычно для этого используются индексы). Как мы рассматривали ранее, в некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

Модуль управления буферами оперативной памяти предназначен для решения задач эффективной буферизации, которая используется практически для выполнения всех остальных функций СУБД.
Условно оперативную память, которой управляет СУБД, можно представить как совокупность буферов, хранящих страницы данных, буферов, хранящих страницы журналов транзакций и область совместно используемого пула (см. рисунок ниже). Последняя область содержит фрагменты системного каталога, которые необходимо постоянно держать в оперативной памяти, чтобы ускорить обработку запросов пользователей, и область операторов SQL с курсорами. Фрагменты системного каталога в некоторых реализациях называются словарем данных. В стандарте SQL2 определены общие требования к системному каталогу.

Оперативная память, управляемая СУБД

Системный каталог в реляционных СУБД представляет собой совокупность специальных таблиц, которыми владеет сама СУБД. Таблицы системного каталога создаются автоматически при установке программного обеспечения сервера БД. Все системные таблицы обычно объединяются некоторым специальным "системным идентификатором пользователя". При обработке SQL-запросов СУБД постоянно обращается к этим таблицам. В некоторых СУБД разрешен ограниченный доступ пользователей к ряду системных таблиц, однако только в режиме чтения. Только системный администратор имеет некоторые права на модификацию данных в некоторых системных таблицах.

---

19. Обработка запросов в СУБД.

Из презентации:

- Синтаксический анализ
- Разрешение имен
  - Канонизация имен (server.database.schema.table; database.schema.table; schema.table)
  - Проверка наличия (системный каталог)
- Преобразование во внутренний формат, с которым может работать оптимизатор
- Проверка ограничений (CHECK)
- Проверка прав доступа (тут сложно)

По факту: общая схема обработки запросов

1. На первой фазе запрос, заданный на языке запросов SQL, подвергается лексическому и синтаксическому анализу. При этом вырабатывается его внутреннее представление, отражающее структуру запроса и содержащее информацию, которая характеризует объекты базы данных, упомянутые в запросе (отношения, поля и константы). Информация о хранимых в базе данных объектах выбирается из каталогов базы данных. Внутреннее представление запроса используется и преобразуется на следующих стадиях обработки запроса. Форма внутреннего представления должна быть достаточно удобной для последующих оптимизационных преобразований.

2. На второй фазе запрос во внутреннем представлении подвергается логической оптимизации. Могут применяться различные преобразования, "улучшающие" начальное представление запроса. Среди преобразований могут быть эквивалентные, после проведения которых получается внутреннее представление, семантически эквивалентное начальному (например, приведение запроса к некоторой канонической форме), Преобразования могут быть и семантическими: 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.

3. Третий этап обработки запроса состоит в выборе на основе информации, которой располагает оптимизатор, набора альтернативных процедурных планов выполнения данного запроса в соответствии с его внутреннем представлением, полученным на второй фазе. Для каждого плана оценивается предполагаемая стоимость выполнения запроса. При оценках используется статистическая информация о состоянии базы данных, доступная оптимизатору. Из полученных альтернативных планов выбирается наиболее дешевый, и его внутреннее представление теперь соответствует обрабатываемому запросу.

4. На четвертом этапе по внутреннему представлению наиболее оптимального плана выполнения запроса формируется выполняемое представление плана. Выполняемое представление плана может быть программой в машинных кодах, если, как в случае System R, система ориентирована на компиляцию запросов в машинные коды, или быть машинно-независимым, но более удобным для интерпретации, если, как в случае INGRES, система ориентирована на интерпретацию запросов. В нашем случае это не принципиально, поскольку четвертая фаза обработки запроса уже не связана с оптимизацией.

5. Наконец, на пятом этапе обработки запроса происходит его реальное выполнение. Это либо выполнение соответствующей подпрограммы, либо вызов интерпретатора с передачей ему для интерпретации выполняемого плана.

---

20. Оптимизация запросов. Планировщик запросов.

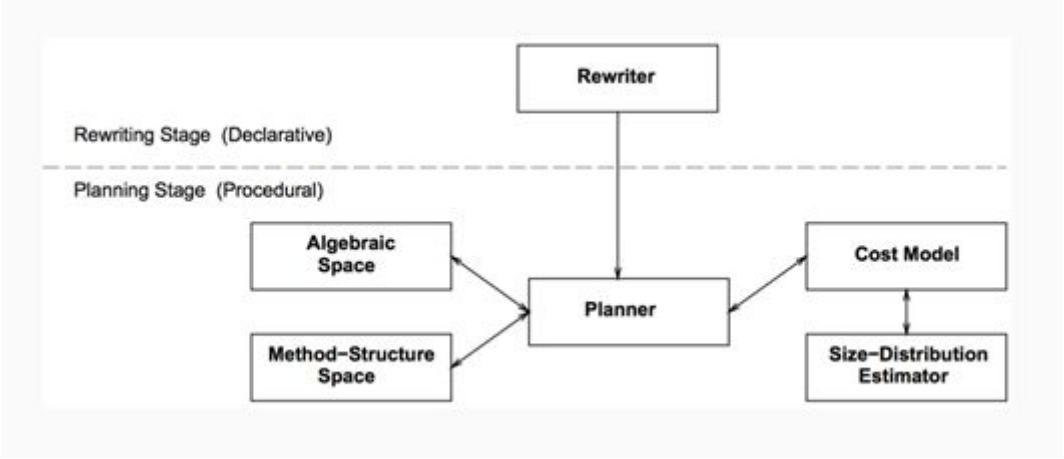
Оптимизация запросов

- Синтаксические оптимизации
- Разрешение представлений
- Вычисление констант
- Преобразования по транзитивности предикатов
  $R.X < 10$ AND $R.X = S.Y \rightarrow \ldots$ AND $S.Y < 10$
- Семантические преобразования (исключение лишних соединений)
- Разворачивание подзапросов
- Эвристики

Оптимизация запросов — это
1) функция СУБД, осуществляющая поиск оптимального плана выполнения запросов из всех возможных для заданного запроса,
2) процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса.
Один и тот же результат может быть получен СУБД различными способами (планами выполнения запросов), которые могут существенно отличаться как по затратам

ресурсов, так и по времени выполнения. Задача оптимизации заключается в нахождении оптимального способа.
В реляционной СУБД оптимальный план выполнения запроса — это такая последовательность применения операторов реляционной алгебры к исходным и промежуточным отношениям, которая для конкретного текущего состояния БД (её структуры и наполнения) может быть выполнена с минимальным использованием вычислительных ресурсов.

Структура оптимизатора запросов



Rewriter

Построение некоторой эквивалентной формы, называемой иногда канонической формой, которая требует меньших затрат на выполнение запроса, но дает результат, полностью эквивалентный исходному запросу.

- Нет взаимодействия с данными
- Раскрытие представлений
- Логические преобразования запросов. Прежде всего относится к преобразованию предикатов, входящих в условие выборки
- Преобразования запросов с изменением порядка реляционных операций
- Приведение запросов с вложенными подзапросами к запросам с соединениями

Логические преобразования

- Предикаты, содержащие операции сравнения простых значений:

  арифм. выражение ОС арифм. выражение

  - ОС — операция сравнения
  - Арифметические выражения в общем случае содержат имена полей отношений и константы
- Каноническое представление

  имя поля ОС константное арифм. выражение

Пример на логическое преобразование:

- Начальное представление предиката:

  *(n+12)*R.B OC 100*

  - n — переменная языка
  - R.B — имя столбца B отношения R
  - OC - допустимая операция сравнения
- Каноническое представление

  R.B OC 100/(n+12)

Логические преобразования. Два поля

Если предикат включает в точности два имени поля разных отношений (или двух разных вхождений одного отношения), то его каноническое представление имеет вид:

имя поля OC арифметическое выражение

- Начальное представление предиката:
  12(R1.A)-n*(R2.B) OC m*
- Каноническое представление
  R1.A OC (m+n(R2.B))/12*

Логические преобразования. Общий случай

- В общем случае желательно приведение предиката к каноническому представлению вида:
  арифметическое выражение OC константное арифметическое выражение
- В дальнейшем можно произвести поиск общих арифметических выражений в разных предикатах запроса

Изменение порядка реляционных операций

- Исходный запрос

```
R1 NATURAL JOIN R2
    WHERE R1.A > a AND R2.B < b
```

- Измененный запрос

```
R3 = R1[R1.A > a]
R4 = R2[R2.B < b]
R5 = R3*[ ]*R4
```

Приведение запросов с вложенными подзапросами

- Предикаты с вложенными подзапросами при наличии общего синтаксиса могут обладать весьма различной семантикой
- Единственным общим для всех возможных семантик вложенных подзапросов алгоритмом выполнения запроса является вычисление вложенного подзапроса всякий раз при вычислении значения предиката
- Каноническим представлением запроса на n отношениях называется запрос, содержащий n–1 предикат соединения и не содержащий предикатов с вложенными подзапросами

Планировщик

- Составление «всех» возможных планов выполнения запросов

· Выбора наиболее «дешевого»
· Использует стратегию поиска на множестве планов, которое определяется модулями Algebraic Space, Method-Structure Space

Algebraic Space

· Определяет набор действий и порядок их выполнения для каждого плана
· Все наборы действий должны давать одинаковый результат
· Чаще всего используются сущности реляционной алгебры

Method-Structure Space

· Определяет способ выполнения действий
· Учитывает особенности организации хранения данных в СУБД (nested loops, merge scan, and hash join)
· Использует данные об индексных конструкциях
· Полученная формула в виде дерева от предыдущего модуля компилируется в конечный план исполнения

Cost Model

· Оценивает «стоимость» выполнения плана
· Использует усредненные данные по скорости доступа к диску, работе с индексами и пр.
· Определяет формулы для любого метода соединения, использования индексов и пр.

Size-Distribution Estimator

· Определяет размеры выборок и индексов
· Продуцируемые данные используются в оценке «стоимости» запросов

Статистика

Виды:

· Количество строк/страниц в таблицах
· По столбцам:
· количество уникальных значений
· длина/диапазон значений

Ограничения:

· Нужно периодически обновлять
· Что делать на большом количестве записей?

Пример 1

emp(name,age,sal,dno)
dept(dno,dname,oor,budget,mgr,ano)
acnt(ano,type,balance,bno)
bank(bno,bname,address)

Parameter Description	Parameter Value
Number of emp pages	20000
Number of emp tuples	100000
Number of emp tuples with sal>100K	10
Number of dept pages	10
Number of dept tuples	100
Indices of emp	Clustered B+-tree on emp.sal (3-levels deep)
Indices of dept	Clustered hashing on dept.dno (average bucket length of 1.2 pages)
Number of buffer pages	3
Cost of one disk page access	20ms

Запрос в примере 1:

```
SELECT NAME, FLOOR
        FROM emp, dept
        WHERE emp.dno = dept.dno AND sal > 100K;
```

Пример 1. Планы выполнения:

- По бинарному индексу определить выборку сотрудников с удовлетворяющим условием оплаты. Для каждого результата выборки используя hashing index найти соответствующий департамент.
- Для каждой страницы департаментов производится поиск сотрудников. Если запись сотрудника соответствует заданным условиям (принадлежит к подразделению, ЗП), строка сотрудника и строка подразделения попадают в результирующую выборку.

Для каждой записи подразделения производится выборка данных, сотрудников, сохраняется m*n объединений. Производится фильтрация полученного списка по равенству значению атрибута внешнего ключа и значению ЗП.

---

21. Алгоритмы выбора плана запроса.

Алгоритмы поиска оптимального плана
Как искать?

- Dynamic Programming Algorithms (экспотенциальная сложность) - алгоритмы динамического программирования
- Randomized Algorithms – рандомизированные (??) алгоритмы
- Other Search Strategies (комбинированные, эвристические) - другие стратегии поиска

Semantic Query Optimization (Оптимизация семантического запроса)

- `sal > 100K WHERE job = 'Sr. Programmer'`
- Исходный запрос

```
SELECT name, floor
        FROM emp, dept
WHERE emp.dno = dept.dno
        AND job = 'Sr. Programmer'
```

- Запрос после преобразования

```
SELECT name, floor
        FROM emp, dept
WHERE emp.dno = dept.dno
        AND job = 'Sr. Programmer' AND sal > 100K
```

Global Query Optimization (Глобальная оптимизация запросов)

- Запрос 1

```
SELECT name, floor
        FROM emp, dept
WHERE emp.dno = dept.dno
        AND job = 'Sr. Programmer'
```

- Запрос 2

```
SELECT name
        FROM emp, dept
WHERE emp.dno = dept.dno AND budget > 1M
```

Parametric/Dynamic Query Optimization (оптимизация параметрических/динамических запросов)

· Необходим учет значений параметров
· Да и все меняется постоянно...
· Хорошо бы на этапе компиляции выбирать план
· Randomized algorithms / Volcano

А можно динамически собирать статистику, насколько часто меняется стоимость плана в процессе выполнения

---

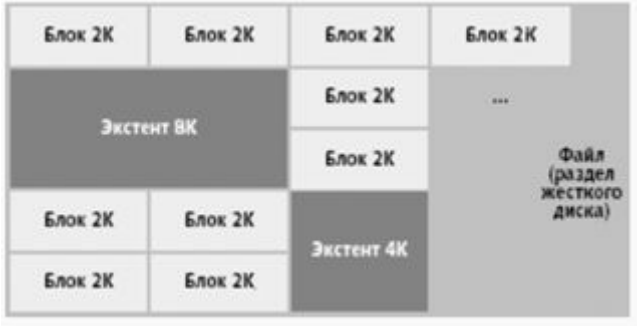
22. Организация данных во внешней памяти в СУБД. Табличное пространство.

Хранение данных во внешней памяти в известных СУБД (Oracle, IBM DB2, Microsoft SQL Server, CA-OpenIngres, решения от Sybase и Informix и др.) организовано очень похожим образом.

Основные единицы физического хранения:

1. блок данных
2. экстент
3. файл (либо раздел жесткого диска).

Взаимосвязь блоков, экстентов и файлов баз данных:



Блок — наименьшая единица выделения пространства в СУБД, минимальная единица обмена с внешней памятью. В блоках и хранятся строки данных, индексов или промежуточные результаты сортировок. Именно блоками сервер СУБД обычно выполняет чтение и запись на диск. Блок данных содержит фиксированное число байт физического пространства на диске:

- 16 – 64 кб лучше всего для длинных строк, текстов, blob'ов
- 2-4 кб для чисел, timestamp, коротких строк

Размер блока данных устанавливается для каждой базы данных либо для каждого типа структур. Очень важно сразу правильно выбрать размер блока: в работающей базе изменить его практически невозможно. Этот размер кратен размеру блока операционной системы.

Размер блока оказывает большое влияние на производительность базы данных — при больших размерах скорость операций чтения/записи растет (особенно это характерно для полных просмотров таблиц и операций интенсивной загрузки данных), однако возрастают накладные расходы на хранение (база увеличивается) и снижается эффективность индексных просмотров. Меньший размер блока позволяет более экономно расходовать память, но вместе с тем относительно дорог.

Следует также учитывать размер блока ОС, он должен быть кратен размеру блока базы данных. Малый размер блока лучше подходит для систем оперативной обработки транзакций, потому что, если сервер блокирует данные на уровне блоков, то это позволяет большему числу пользователей работать, не мешая друг другу. В системах поддержки принятия решений, для которых более критичным является не общая пропускная способность (количество транзакций в единицу времени), а среднее время отклика (response time), больший блок предпочтительнее.

Все блоки данных можно разделить на две основные части: часть строк данных и часть свободного пространства. (Есть и другие, более мелкие области, такие как пространство заголовков и пространство для накладных расходов, связанных с обслуживанием базы.) Раздел строк данных содержит собственно данные, хранимые в таблицах и их индексах. Раздел свободного пространства представляет собой место, оставшееся в блоке данных СУБД для вставки новых данных или расширения существующих строк в блоке.

Формат блока:

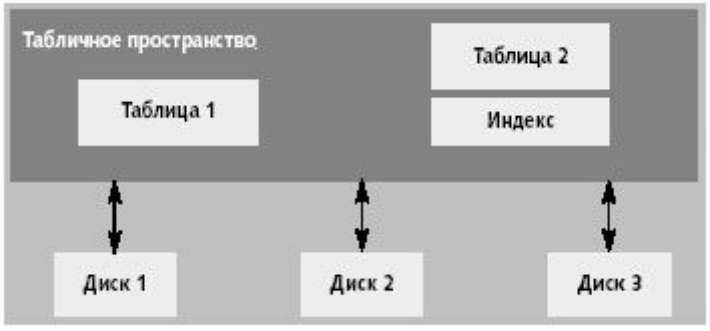
Заголовок блока содержит информацию о типе блока (блок таблицы, блок индекса и т.д.), информацию о текущих и прежних транзакциях, затронувших блок, а также адрес (местонахождение) блока на диске. Каталог таблиц содержит информацию о таблицах, строки которых хранятся в этом блоке (в блоке могут храниться данные нескольких таблиц). Каталог строк содержит описание хранящихся в блоке строк (включая адреса каждой порции строки в области данных строк). Это массив указателей на строки, хранящиеся в области данных блока. Вместе эти три части блока называются служебным пространством блока. Это пространство недоступно для данных и используется сервером Oracle для управления блоком. Остальные две части блока вполне понятны: в блоке имеется занятое пространство, в котором хранятся данные, и может быть свободное пространство.

Администратор отводит пространство для базы данных на внешних устройствах большими фрагментами: **файлами и разделами диска**. В первом случае доступ к диску осуществляется операционной системой, что дает определенные преимущества, например, работа с файлами средствами ОС. Во втором случае с внешним устройством работает сам сервер. При этом достигается более высокая производительность; кроме того, использование дисков необходимо в случае, если кэш ОС не может работать в режиме сквозной (write-through) записи. Диски особенно эффективны для ускорения операций записи данных (подобный механизм поддерживается Oracle, DB2 и Informix; например, в DB2 данная единица размещения называется контейнером).

Пространством внешней памяти, отведенным ему администратором, сервер управляет с помощью **экстентов** (extent), т.е. непрерывных последовательностей блоков. Информация о наличии экстентов для объекта схемы данных находится в специальных управляющих структурах, реализация которых зависит от СУБД. На управление экстентами (выделение пространства, освобождение, слияние) тратятся определенные ресурсы, поэтому для достижения эффективности нужно правильно определять их параметры. СУБД от Oracle, IBM, Informix позволяют определять параметры этих структур, а в Sybase экстенты имеет постоянный размер, равный 8 страницам. Уменьшение размера экстента будет способствовать более эффективному использованию памяти, однако при этом возрастают накладные расходы на управление большим количеством экстентов, что может замедлить операции вставки большого количества строк в таблицу. Кроме того, сервер может иметь ограничение на

максимальное количество экстентов для таблицы. При слишком большом размере экстентов могут возникнуть проблемы с выделением для них необходимого количества памяти. Обычно определяется размер начального экстента, размер второго и правило определения размеров следующих экстентов.

Общим для СУБД Oracle, DB2 и Informix является понятие **пространства** (для Oracle и DB2 это табличное пространство) – единицы логического хранения. Различные логические структуры данных, такие как таблицы и индексы, временные таблицы и словарь данных размещены в табличных пространствах. В DB2 и Informix дополнительно можно устанавливать размер страницы отдельно для каждой из этих структур. Группировка хранимых данных по пространствам производится по ряду признаков: частота изменения данных, характер работы с данными (преимущественно чтение или запись и т.п.), скорость роста объема данных, важность и т.п. Таким образом, например, только читаемые таблицы помещаются в одно пространство, для которого установлены одни параметры хранения, таблицы транзакций размещаются в пространстве с другими параметрами.



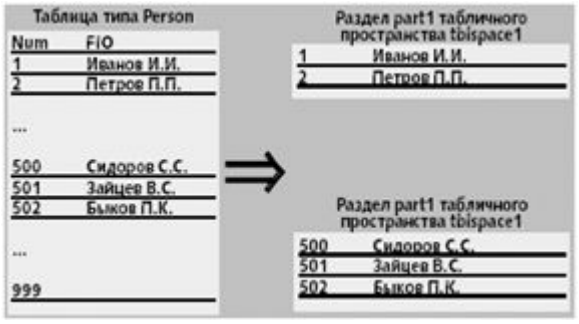
Одна логическая единица данных (таблица или индекс) размещается точно в одном пространстве, которое может быть отображено на несколько физических устройств или файлов. При этом физически разнесены (располагаться на разных дисках) могут не только логические единицы данных (таблицы отдельно от индексов), но и данные одной логической структуры (таблица на нескольких дисках). Такой способ хранения называется горизонтальной **фрагментацией**: таблица делится на фрагменты по строкам. В Oracle вместо термина «фрагментация» используется «секционирование» (partitioning). Фрагментация — один из способов повышения производительности. Могут применяться различные схемы записи данных во фрагментированные таблицы. Одна из них — круговая (round-robin), когда некоторая часть вставляемых в таблицу строк записывается в первый фрагмент, другая часть — в следующий и так далее по кругу. В данном случае за счет распараллеливания может быть увеличена производительность операций модификации данных и запросов.

Существует и другая схема, включающая логическое разделение строк таблицы по первичному ключу - **кластеризация**. Данная схема позволяет избежать перерасхода процессорного времени и уменьшить общий объем операций ввода/вывода. Ее суть в том, что при создании таблицы все пространство значений ключа таблицы разбивается на несколько интервалов, а строкам с ключами, принадлежащими разным интервалам, назначаются различные месторасположения. Впоследствии, при обработке запроса, данная информация учитывается оптимизатором. Если

производится поиск по ключу, то оптимизатор может удалять из рассмотрения фрагменты таблицы, не удовлетворяющие условию выборки. Например, создание кластеризованной таблицы будет выглядеть следующим образом (этот и все остальные SQL-скрипты приведены для Oracle):

```
CREATE TABLE person
( num INTEGER,
  fio VARCHAR2 (30) )
PARTITION BY RANGE (num)
( PARTITION part1 VALUES LESS THAN ( 500 )
  TABLESPACE tblspace1,
PARTITION part2 VALUES LESS THAN ( 1000 )
  TABLESPACE tblspace2
)
```

Здесь создаются два раздела part1 и part2, каждый из которых размещен в своем табличном пространстве (tblspace1 и tblspace2). Записи со значением поля num от 1 до 499 будут располагаться в первом разделе, а записи с номерами от 500 до 1000 — во втором.



Тогда при запросе:

```
SELECT fio FROM person
 WHERE num BETWEEN 10 AND 40
```

оптимизатор будет производить поиск только в разделе part1, что может дать ощутимый выигрыш в производительности в таблице с десятками тысяч строк.

---

23. Методы доступа к данным. Назначение и типы индексов.

Набор операций над данными, выполняемых сервером для получения результата заданной выборки, называется **путем доступа**. Путь доступа можно представить в виде дерева с корнем, представляющим собой конечный результат. Каждый узел этого дерева называется **методом доступа** или **источником данных**. Объектами операций в методах доступа являются **потоки данных**. Каждый метод доступа либо формирует поток данных, либо трансформирует его по определенным правилам.

С точки зрения видов выполняемых операций существует три класса источников данных:

- первичный метод доступа – выполняет чтение из таблицы или хранимой процедуры

- фильтр – трансформирует один входной поток данных в один выходной поток
- слияние – преобразует два или более входных потоков данных в один выходной поток

Источники данных могут быть конвейерными и буферизированными. Конвейерный источник данных выдает записи в процессе чтения своих входных потоков, в то время как буферизированный источник сначала должен прочитать все записи из своих входных потоков и только потом сможет выдать первую запись на свой выход.

С точки зрения оценки производительности, каждый метод доступа имеет два обязательных атрибута – **кардинальность (cardinality) и стоимость (cost)**. Первый отражает, сколько записей будет выбрано из источника данных. Второй оценивает стоимость выполнения метода доступа. Величина стоимости напрямую зависит от кардинальности и механизма выборки или трансформации потока данных.

Методы доступа:

1. Полное сканирование

Этот метод также известен как естественный или последовательный перебор (full table scan, natural scan, sequential scan).

В данном методе доступа сервер осуществляет последовательное чтение всех страниц, выделенных для данной таблицы, в порядке их физического расположения на диске. Очевидно, что этот метод обеспечивает наиболее высокую производительность с точки зрения пропускной способности. Также достаточно очевидно, что прочитаны будут все записи данной таблицы независимо от того, нужны они нам или нет.

Так как ожидаемый объем данных довольно велик, то в процессе чтения страниц таблицы с диска существует проблема вытеснения читаемыми страницами других, потенциально нужных конкурирующим сессиям. Для этого логика работы страничного кеша меняется – текущая страница скана помечается как MRU (most recently used) в течение чтения всех записей с данной страницы. Как только на странице нет больше данных и надо запрашивать следующую, то текущая страница освобождается с признаком LRU (least recently used), уходя в "хвост" очереди и будучи таким образом первым кандидатом на удаление из кеша.

Записи читаются из таблицы по одной, сразу выдаваясь на выход.

Оптимизатор при выборе данного метода доступа использует стратегию на основе правил (rule based) – полное сканирование осуществляется только в случае отсутствия индексов, применимых к предикату (true/false) запроса. Стоимость этого метода доступа равна количеству записей в таблице (оценивается приближенно по количеству страниц таблицы, размеру записи и среднему коэффициенту сжатия страниц данных).

В статистике выполнения запроса записи, прочитанные полным сканированием, отражаются как неиндексированные чтения (non indexed reads).

2. Сканирование диапазона

Поиск по индексу осуществляется с использованием верхней и нижней границы. То есть, если указана нижняя граница сканирования, то сначала находится соответствующий ей ключ и только потом начинается последовательный перебор ключей с занесением номеров записей в битовую карту. Если указана верхняя граница, то каждый ключ сравнивается с ней и при ее достижении сканирование прекращается. Такой механизм называется сканированием диапазона (range scan).

В случае, когда нижняя граница равна верхней, говорят о сканировании на равенство (equality scan). Если сканирование на равенство выполняется для уникального индекса, то это называется уникальным сканированием (unique scan).

Данный вид сканирования имеет особое значение, так как может вернуть не более одной записи и, следовательно, по определению является самым дешевым. Если у нас не задана ни одна из границ, то мы имеем дело с полным сканированием (full scan).

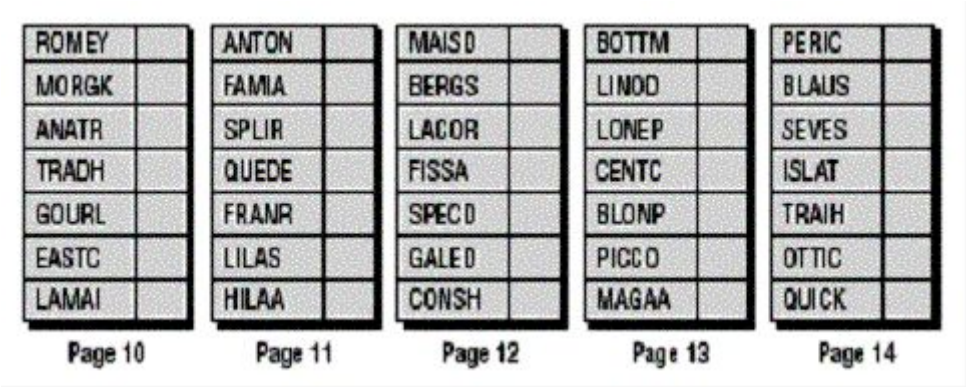
3. Сканирование по уникальным значениям

Используется в тех случаях, когда вам нужно получить из индекса только какое-то одно значение.

Индекс— объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска.

Простейшим примером применения индексов в реальной жизни является оглавление книги. Вместо перебора всех страниц книги, читатель может открыть конкретную главу книги, получив номер ее начальной страницы из оглавления. Той же самой цели служат и индексы в таблице. Они позволяют получить данные из определенной записи без перебора всех записей в таблице.

Физически данные хранятся на 8Кб страницах. Сразу после создания, пока таблица не имеет индексов, таблица выглядит как куча (heap) данных. Записи не имеют определенного порядка хранения.



Когда вы хотите получить доступ к данным, SQL Server будет производить сканирование таблицы (table scan). SQL Server сканирует всю таблицу чтобы найти искомые записи. Например мы хотим найти запись, удовлетворяющую условию:

```
SELECT * FROM Customers
WHERE CustomerID = 'ROMEY'
```

SQL Server прочитает все записи начиная с первой и заканчивая последней и выберет те, которые будут удовлетворять указанному условию. SQL Server не знает что в таблице существует только одна запись, удовлетворяющая условию, пока в таблице не существует unique constraint, unique index или primary key. Во всех трёх перечисленных случаях создается индекс для поддержания ограничения.

Приведенный пример иллюстрирует две базовые функции индексов:

- увеличение скорости доступа к данным
- поддержка уникальности данных

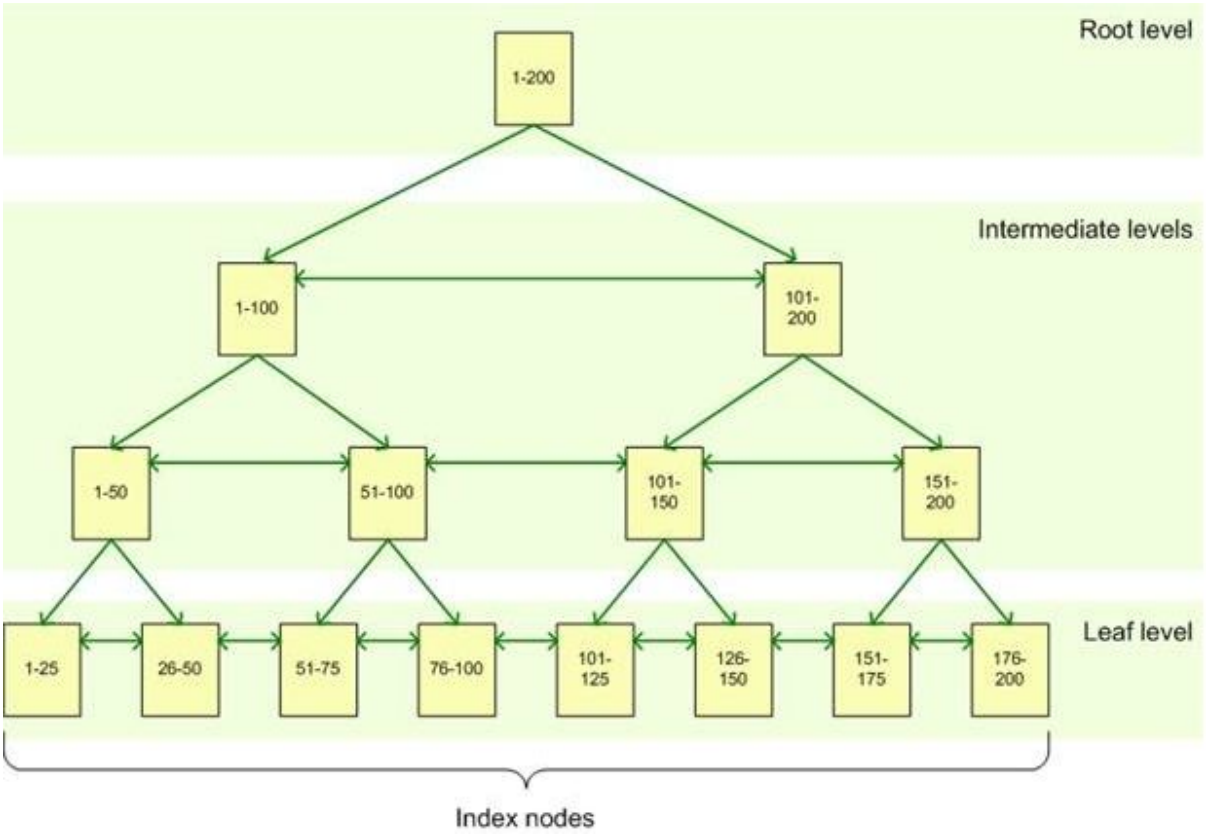
Несмотря на достоинства, индексы так же имеют и ряд недостатков. Первый из них – индексы занимают дополнительное место на диске и в оперативной памяти. Каждый раз когда вы создаете индекс, вы сохраняете ключи в порядке убывания или возрастания, которые могут иметь многоуровневую структуру. И чем больше/длиннее ключ, тем больше размер индекса. Второй недостаток – замедляются операции вставки, обновления и удаления записей. Однако алгоритмы построения индексов разработаны таким образом что бы иметь как можно меньший негативный эффект для указанных операций и даже позволяет выполнять их быстрее.

Типы индексов

1. B-tree

В SQL Server индексы хранятся в виде B-деревьев (B-tree). B-tree означает balanced tree, «сбалансированное дерево». Индекс содержит ровно одну корневую страницу, которая является точкой входа для поиска. Корневая страница содержит значения ключей и ссылки на страницы следующего уровня для данных значений индекса. При поиске по индексу находится последнее значение, которое не превосходит искомое, и происходит переход на соответствующую страницу. На последнем, листьевом уровне дерева перечислены все ключи, и для каждого из них указана ссылка (bookmark) на данные таблицы.

Поиск



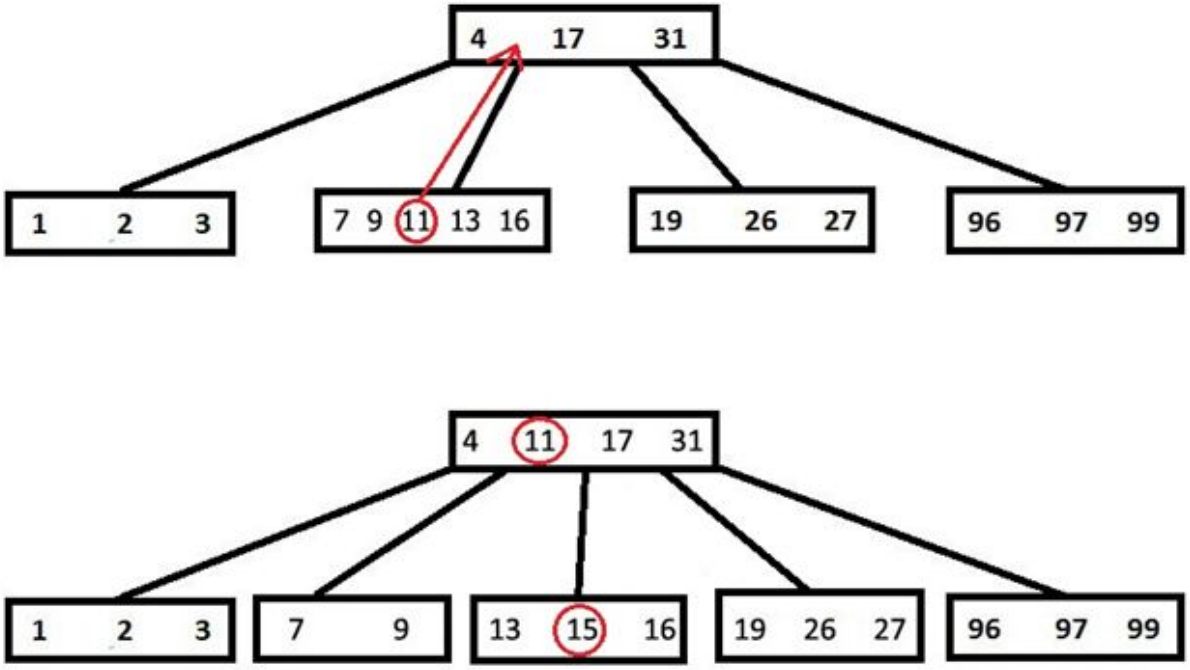
Когда вы формируете запрос на индексированный столбец, подсистема запросов начинает идти сверху от корневого узла и постепенно двигается вниз через промежуточные узлы, при этом каждый слой промежуточного уровня содержит более детальную информацию о данных. Подсистема запросов продолжает двигаться по узлам индекса до тех пор, пока не достигнет нижнего уровня с листьями индекса. К примеру, если вы ищете значение 123 в индексированном столбе, то подсистема запросов сначала на корневом уровне определит страницу на первом промежуточном (intermediate) уровне. В данном случае первой страница указывает на значение от 1 до 100, а вторая от 101 до 200, таким образом подсистема запросов обратится ко второй странице этого промежуточного уровня. Далее будет выяснено, что следует обратиться к третьей странице следующего промежуточного уровня. Отсюда подсистема запросов прочитает на нижнем уровне значение самого индекса. Листья индекса могут содержать как сами данные таблицы, так и просто указатель на строки с данными в таблице, в зависимости от типа индекса: кластеризованный индекс или некластеризованный.

Добавление

В отличие от поиска, операция добавления существенно сложнее, чем в бинарном дереве, так как просто создать новый лист и вставить туда ключ нельзя, поскольку будут нарушаться свойства B-дерева. Также вставить ключ в уже заполненный лист невозможно => необходима операция разбиения узла на 2. Если лист был заполнен, то в нем находилось 2t-1 ключей => разбиваем на 2 по t-1, а средний элемент (для которого t-1 первых ключей меньше его, а t-1 последних больше) перемещается в

родительский узел. Соответственно, если родительский узел также был заполнен – то нам опять приходится разбивать. И так далее до корня (если разбивается корень – то появляется новый корень и глубина дерева увеличивается). Как и в случае обычных бинарных деревьев, вставка осуществляется за один проход от корня к листу. На каждой итерации (в поисках позиции для нового ключа – от корня к листу) мы разбиваем все заполненные узлы, через которые проходим (в том числе лист). Таким образом, если в результате для вставки потребуется разбить какой-то узел – мы уверены в том, что его родитель не заполнен!

На рисунке ниже проиллюстрировано то же дерево, что и в поиске (t=3). Только теперь добавляем ключ «15». В поисках позиции для нового ключа мы натыкаемся на заполненный узел (7, 9, 11, 13, 16). Следуя алгоритму, разбиваем его – при этом «11» переходит в родительский узел, а исходный разбивается на 2. Далее ключ «15» вставляется во второй «отколовшийся» узел. Все свойства B-дерева сохраняются!



Операция добавления происходит также за время O(t logt n). Важно опять же, что дисковых операций мы выполняем всего лишь O(h), где h – высота дерева.

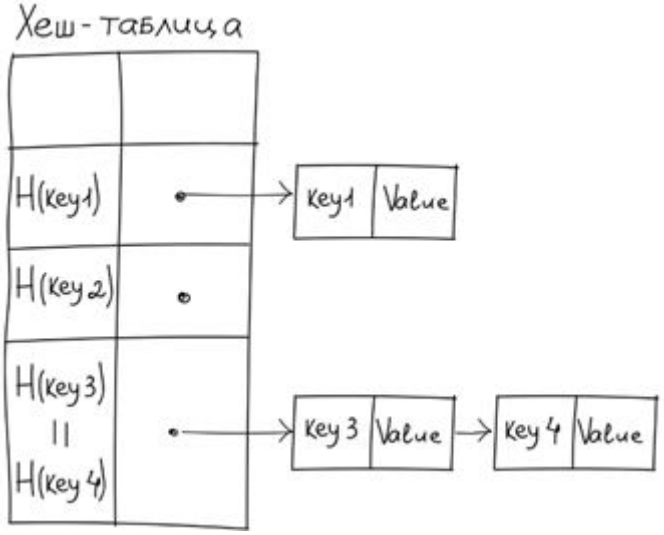
1. Хэш-табли́ца

Это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Пожалуй, главное свойство hash-таблиц — все три операции вставка, поиск и удаление в среднем выполняются за время *O*(1), среднее время поиска по ней также равно *O*(1) и *O*(n) в худшем случае.
Минусы:

· коллизии/переполнение

Для борьбы обычно применяются методы цепочек и открытой индексации.

Метод цепочек часто называют открытым хешированием. Его суть проста — элементы с одинаковым хешем попадают в одну ячейку в виде связного списка.



Второй распространенный метод — открытая индексация. Это значит, что пары ключ-значение хранятся непосредственно в хеш-таблице. А алгоритм вставки проверяет ячейки в некотором порядке, пока не будет найдена пустая ячейка. Порядок вычисляется на лету.
Самая простая в реализации последовательность проб — линейное пробирование (или линейное исследование). Здесь все просто — в случае коллизии, следующие ячейки проверяются линейно, пока не найдет нужный элемент или пустую ячейку. В случае, если таблица будет заполнена, ее придется динамически расширять.

· Сложно выбрать хорошую хэш-функцию для сложных типов данных
· При использовании внешней памяти скорость ухудшается

3. Bitmap

Bitmap index – метод битовых индексов заключается в создании отдельных битовых карт (последовательность 0 и 1) для каждого возможного значения столбца, где каждому биту соответствует строка с индексируемым значением, а его значение равное 1 означает, что запись, соответствующая позиции бита содержит индексируемое значение для данного столбца или свойства.

EmpID	Пол
1	Мужской
2	Женский
3	Женский
4	Мужской
5	Женский

Битовые карты

Значение	Начало	Конец	Битовая маска
Мужской	Адрес первой строки	Адрес последней строки	10010
Женский	Адрес первой строки	Адрес последней строки	01101

Основное преимущество битовых индексов в том, что на больших множествах с низкой мощностью и хорошей кластеризацией по их значениям индекс будет меньше чем B*-tree.

4. Инвертированный список

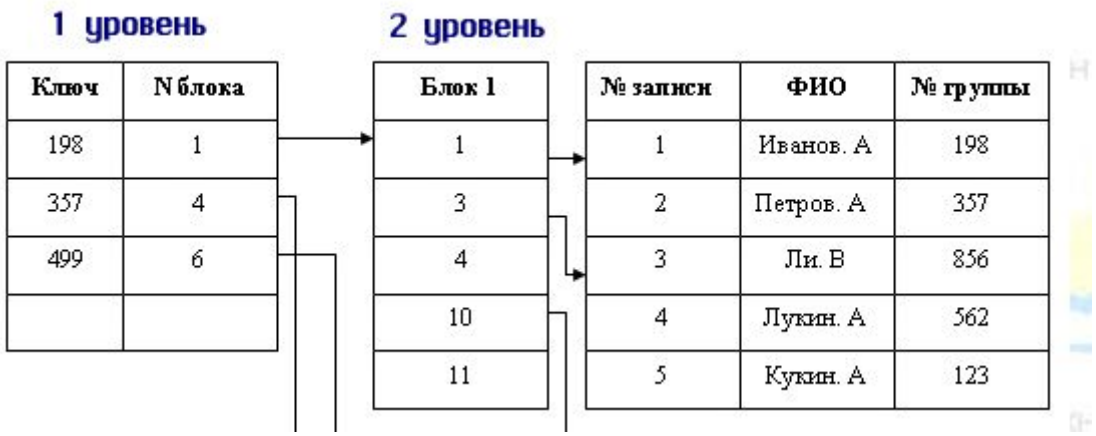
достаточно часто в базах данных требуется проводить операции доступа по вторичным ключам. Напомним, что вторичным ключом является набор атрибутов, которому соответствует набор искомых записей. Это означает, что существует множество записей, имеющих одинаковые значения вторичного ключа. Например, в случае нашей БД «Библиотека» вторичным ключом может служить место издания, год издания. Множество книг могут быть изданы в одном месте, и множество книг могут быть изданы в один год.

Для обеспечения ускорения доступа по вторичным ключам используются структуры, называемые инвертированными списками, которые послужили основой организации индексных файлов для доступа по вторичным ключам.

Инвертированный список в общем случае — это двухуровневая индексная структура. Здесь на первом уровне находится файл или часть файла, в которой упо-рядочепно расположены значения вторичных ключей. Каждая запись с вторичным ключом имеет ссылку на номер первого блока в цепочке блоков, содержащих номера записей с данным значением вторичного ключа. На втором уровне находится цепочка блоков, содержащих номера записей, содержащих одно и то же значение вторичного ключа. При этом блоки второго уровня упорядочены по значениям вторичного ключа.

И наконец, на третьем уровне находится собственно основной файл.

Механизм доступа к записям по вторичному ключу при подобной организации записей весьма прост. На первом шаге мы ищем в области первого уровня заданное значение вторичного ключа, а затем по ссылке считываем блоки второго уровня, содержащие номера записей с заданным значением вторичного ключа, а далее уже прямым доступом загружаем в рабочую область пользователя содержимое всех записей, содержащих заданное значение вторичного ключа.



---

24. Способы слияния наборов данных.

Главной задачей технологии слияния данных (Data Fusion) является объединение данных из разных источников в интересах решения последующих содержательных задач: принятие решений, классификация, определение состояния объектов, оценка ситуации и т.д.

Зависимостью может быть:

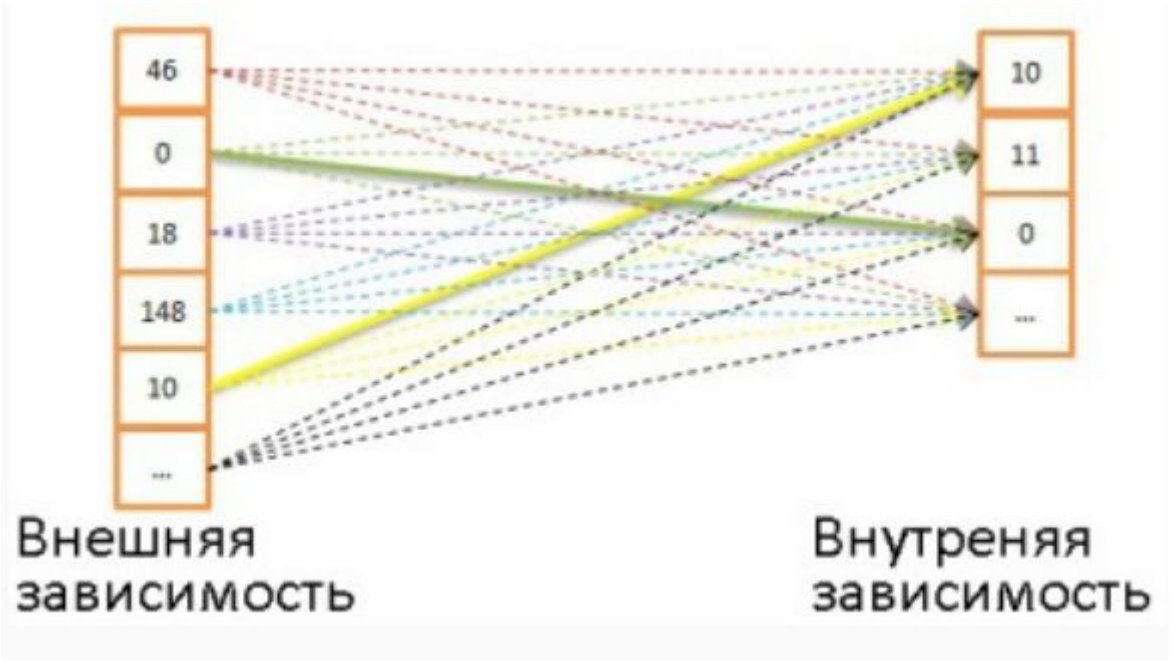
- таблица
- индекс
- результат предыдущей операции

Когда мы объединяем две зависимости, алгоритмы объединения управляют ими по-разному. Допустим, A JOIN B является объединением A и B, где A является внешней зависимостью, а B — внутренней.

Чаще всего стоимость A JOIN B не равна стоимости B JOIN A.

Nested loops

Суть очень простая — для каждого элемента первого списка пройдемся по всем элементам второго списка; если ключи элементов вдруг оказались равны — запишем совпадение в результирующую таблицу.



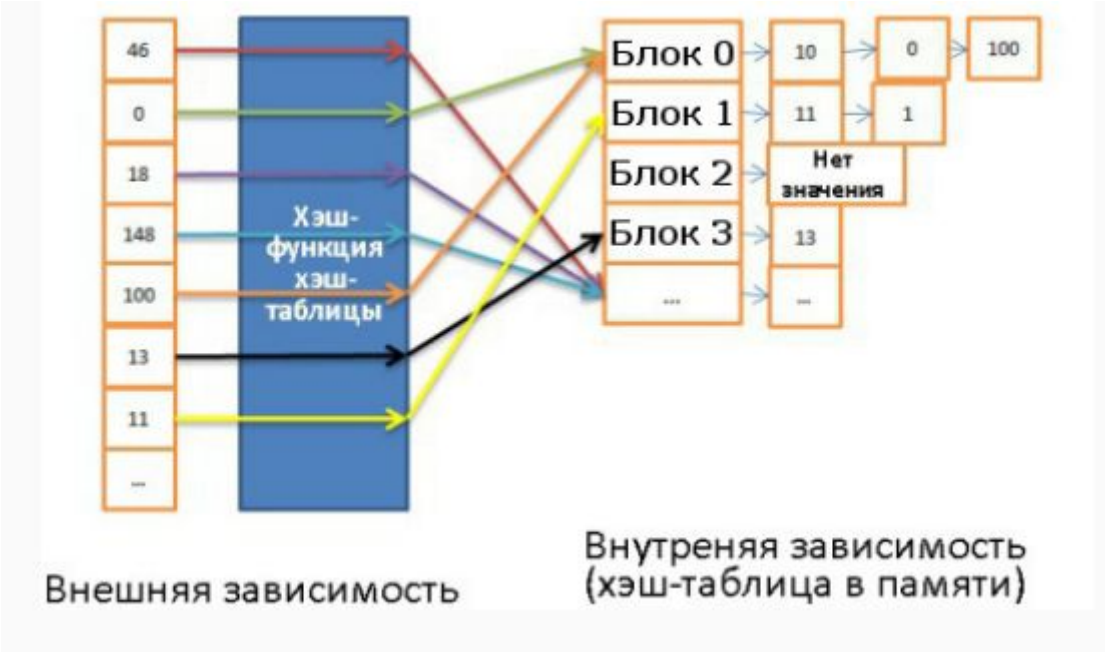
- Временная сложность определяется как O(N*M)
- Чтений с диска N + N*M
- Можно внутреннюю зависимость считать в память
- Или заменить индексом
- Можно считывать наборами данных (временная сложность та же, но экономим дисковые операции)

Основной плюс этого метода — полное безразличие к входным данным. Алгоритм работает для любых двух таблиц, не требует никаких индексов и перекладываний данных в памяти, а также прост в реализации. На практике это означает, что достаточно просто бежать по диску двумя курсорами и периодически выплёвывать в сокет совпадения.

Однако ж, фатальный минус алгоритма — высокая временная сложность O(N*M) (квадратичная асимптотика, если вы понимаете, о чем я). Например, для джойна пары небольших таблиц в 100к и 500к записей потребуется сделать аж 100.000 * 500.000 = 50.000.000.000 (50 млрд) операций сравнения. Запросы с таким джойном будут выполняться невежливо долго, часто именно они — причина беспощадных тормозов кривеньких самописных CMS'ок.

Современные РСУБД используют nested loops join в самых безнадежных случаях, когда не удается применить никакую оптимизацию.

Hash join
Если размер одной из таблиц позволяет засунуть ее целиком в память, значит, на ее основе можно сделать хеш-таблицу и быстренько искать в ней нужные ключи.



- Считывается внутренняя зависимость, для нее создается хэш-таблица в основной памяти
- Последовательно считываются элементы внешней зависимости
- Для каждого считается хэш, проверяется наличие в таблице для внутренней зависимости
- Если есть соответствие - сохраняется

Проверим размер обоих списков. Возьмем меньший из списков, прочтем его полностью и загрузим в память, построив HashMap. Теперь вернемся к большему списку и пойдем по нему курсором с начала. Для каждого ключа проверим, нет ли такого же в хеш-таблице. Если есть — запишем совпадение в результирующую таблицу.

Временная сложность этого алгоритма падает до линейной O(N+M), но требуется дополнительная память.

Hash join. Особенности

- (M / X) * (N / X) + накладные расходы (?)
- Если хорошая функция - то O(M + N)
- Должно быть условие равенства
- Плохо на больших таблицах

Merge join

А теперь представим, что данные в обоих списках заранее отсортированы, например, по возрастанию. Так бывает, если у нас были индексы по этим таблицам, или же если мы отсортировали данные на предыдущих стадиях запроса. Как вы наверняка помните, два отсортированных списка можно склеить в один отсортированный за линейное время — именно на этом основан алгоритм сортировки слиянием. Здесь нам

предстоит похожая задача, но вместо того, чтобы склеивать списки, мы будем искать в них общие элементы.

Итак, ставим по курсору в начало обоих списков. Если ключи под курсорами равны, записываем совпадение в результирующую таблицу. Если же нет, смотрим, под каким из курсоров ключ меньше. Двигаем курсор над меньшим ключом на один вперед, тем самым “догоняя” другой курсор.

- Нужны отсортированные данные
    - Таблица (или результат предыдущего объединения) отсортирована
    - Зависимость является индексом
- Учитывается планировщиком запросов

Если данные отсортированы, то временная сложность алгоритма линейная O(M+N) и не требуется никакой дополнительной памяти. Если же данные не отсортированы, то нужно сначала их отсортировать. Из-за этого временная сложность возрастает до O(M log M + N log N), плюс появляются дополнительные требования к памяти.

Merge join. Особенности

- Временная сложность
    - O(N * Log(N) + M * Log(M))
    - Если все отсортировано: O(N + M)
- На самом деле все немного сложнее, поскольку могут быть повторения

Выбор алгоритма слияния

Значимые факторы

- Объем доступной памяти
- Размер наборов данных
- Наличие индексов
