Презентации для курса “Основы управления данными” (“Базы данных”)

ФИТУ, ПОВТ, 3-5, 3-5б, 3-6

По плану: 36 час. лекций,
36 час. лаб.работ,
курсовая работа,
экзамен

Лектор: ст. преподователь кафедры ПОВТ
Шлыков Павел Васильевич

Литература

- Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.: ил.

- Зеленков Ю.А. Ведение в базы данных: Центр Интернет ЯрГУ, 1997 г. *http://www.mstu.edu.ru/study/materials/zelenkov/toc.html*

- Пушников А.Ю. Введение в системы управления базами данных: Учебное пособие/Изд-е Башкирского ун-та. - Уфа, 1999. - 246 с. – в двух частях *http://citforum.ru/database/dblearn/index.shtml*

- Мартин Грубер Понимание SQL: Москва, 1993

ВВЕДЕНИЕ. Для чего нужны базы данных

- Компьютеры были созданы для решения вычислительных задач, однако со временем они все чаще стали использоваться для построения ***систем обработки документов***, а точнее, содержащейся в них информации. Такие системы обычно и называют ***информационными***. В качестве примера можно привести систему учета отработанного времени работниками предприятия и расчета заработной платы, систему учета продукции на складе, систему учета книг в библиотеке и т.д.

- ***З АМЕ Ч А Н И Е***

  Информационная система (Information System, IS) — это система, предназначенная для сбора, корректировки и распространения информации внутри организации и объединяющая в своем составе персонал, оборудование, базы данных и программное обеспечение.

ВВЕДЕНИЕ. Для чего нужны базы данных

- Все вышеперечисленные системы имеют следующие особенности:

- для обеспечения их работы нужны сравнительно низкие вычислительные мощности;
- данные, которые они используют, имеют сложную структуру;
- необходимы средства сохранения данных между последовательными запусками системы.

- Другими словами, информационная система требует создания в памяти ЭВМ ***динамически обновляемой*** модели внешнего мира с использованием единого хранилища - **базы данных**.

ВВЕДЕНИЕ. Для чего нужны базы данных

- Для дальнейшего обсуждения нам необходимо ввести понятие предметной области:
- **Предметная область** - часть реального мира, подлежащая изучению с целью организации управления и, в конечном счете, автоматизации. Предметная область представляется множеством *фрагментов*, например, предприятие - цехами, дирекцией, бухгалтерией и т.д. Каждый фрагмент предметной области характеризуется множеством *объектов* и *процессов*, использующих объекты, а также множеством *пользователей*, характеризуемых различными взглядами на предметную область.

ВВЕДЕНИЕ. Для чего нужны базы данных

- Словосочетание "***динамически обновляемая***" означает, что соответствие базы данных текущему состоянию предметной области обеспечивается не периодически, а в режиме реального времени. При этом одни и те же данные могут быть по-разному представлены в соответствии с потребностями различных групп пользователей.
- **Отличительной чертой** баз данных следует считать то, что данные хранятся совместно с их описанием, а в прикладных программах описание данных не содержится. Независимые от программ пользователя данные обычно называются **метаданными**. В ряде современных систем метаданные, содержащие также информацию о пользователях, форматы отображения, статистику обращения к данным и др. сведения, *хранятся в словаре базы данных*.

ВВЕДЕНИЕ. Для чего нужны базы данных

- Таким образом, **система управления базой данных** (*СУБД*) - важнейший компонент информационной системы. Для создания и управления информационной системой СУБД необходима в той же степени, как для разработки программы на алгоритмическом языке необходим транслятор. Основные функции СУБД:

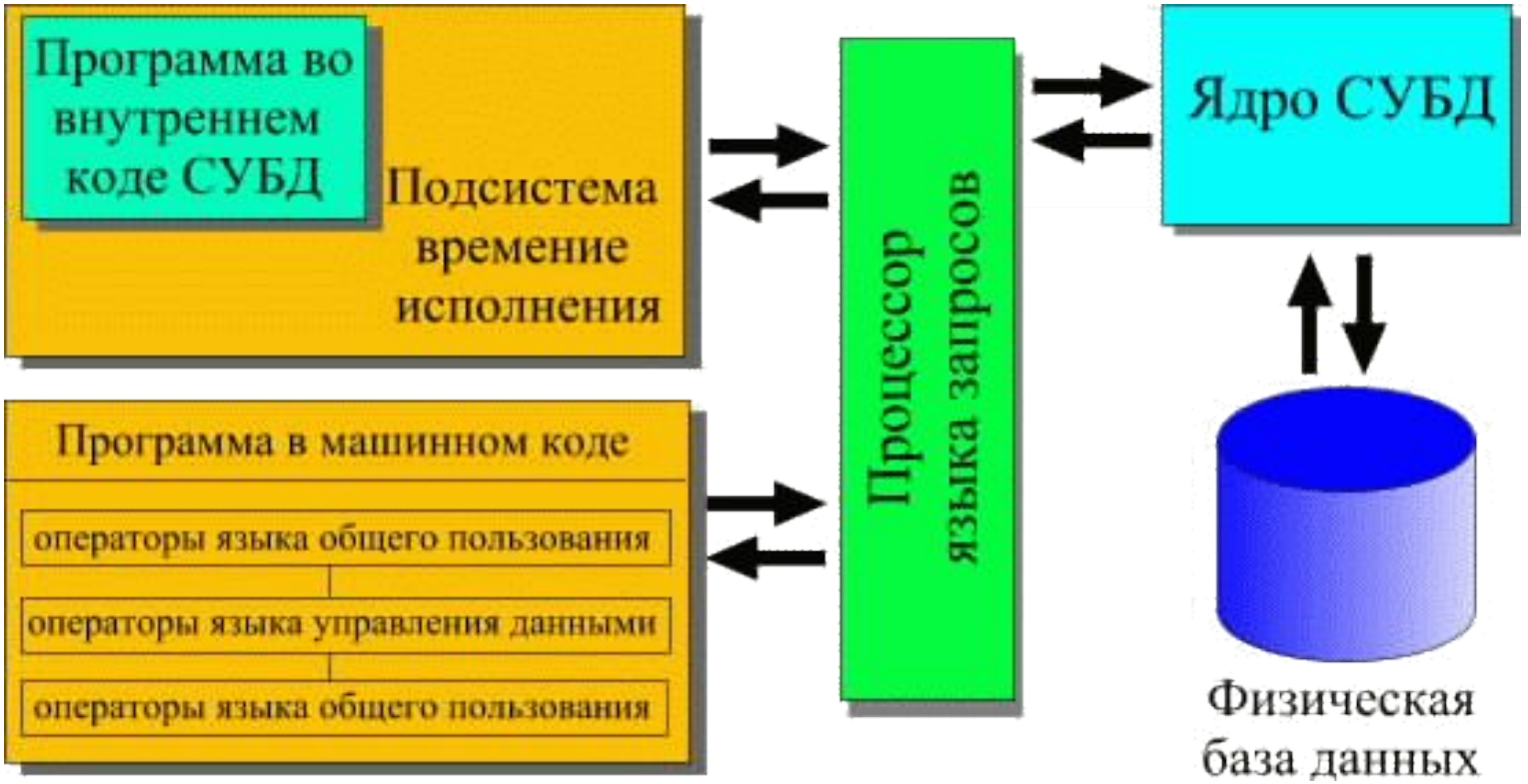
- управление данными во внешней памяти (на дисках);
- управление данными в оперативной памяти;
- журнализация изменений и восстановление базы данных после сбоев;
- поддержание языков БД (язык определения данных, язык манипулирования данными).

ВВЕДЕНИЕ. Для чего нужны базы данных

- Обычно современная СУБД содержит следующие компоненты (см. рис.):

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

ВВЕДЕНИЕ. Для чего нужны базы данных



Компоненты СУБД

ВВЕДЕНИЕ. Для чего нужны базы данных

- Создание первых баз данных и СУБД стало возможно лишь с появлением достаточно дешевых и производительных устройств внешней памяти, какими стали жесткие диски (винчестеры), появившиеся во второй половине 60-х годов. В 70-е годы шла интенсивная разработка теоретических вопросов построения баз данных. В результате в начале 80-х годов на рынке появились мощные инструментальные средства проектирования и построения информационных систем. Однако, развитие информационных технологий в 90-х привело к появлению новых, более широких требований к обработке и представлению данных. Таким образом, теория баз данных, хотя и располагает впечатляющими достижениями, еще далека от завершения.

Системы, основанные на файлах

- Рассмотрим в общих чертах идею построения систем, основанных на файлах.
- Допустим, что мы планируем создать программку, хранящую сведения об именах и днях рождений наших сотрудников. Получивший столь ответственное задание программист поступает следующим образом.
- Во-первых, он готовит структуру, подходящую для хранения указанных данных. Для этого программист просматривает список персонала и выясняет максимальное число символов в фамилии, имени и отчестве (допустим, это значения: 20, 15 и 15 байтов). Затем программист выделяет какое-то число байтов для хранения даты (пусть это будет 8 байтов). Узнав все необходимые размерности, разработчик описывает структуру непосредственно в коде программы.

Системы, основанные на файлах

Структура для хранения одной записи в файле

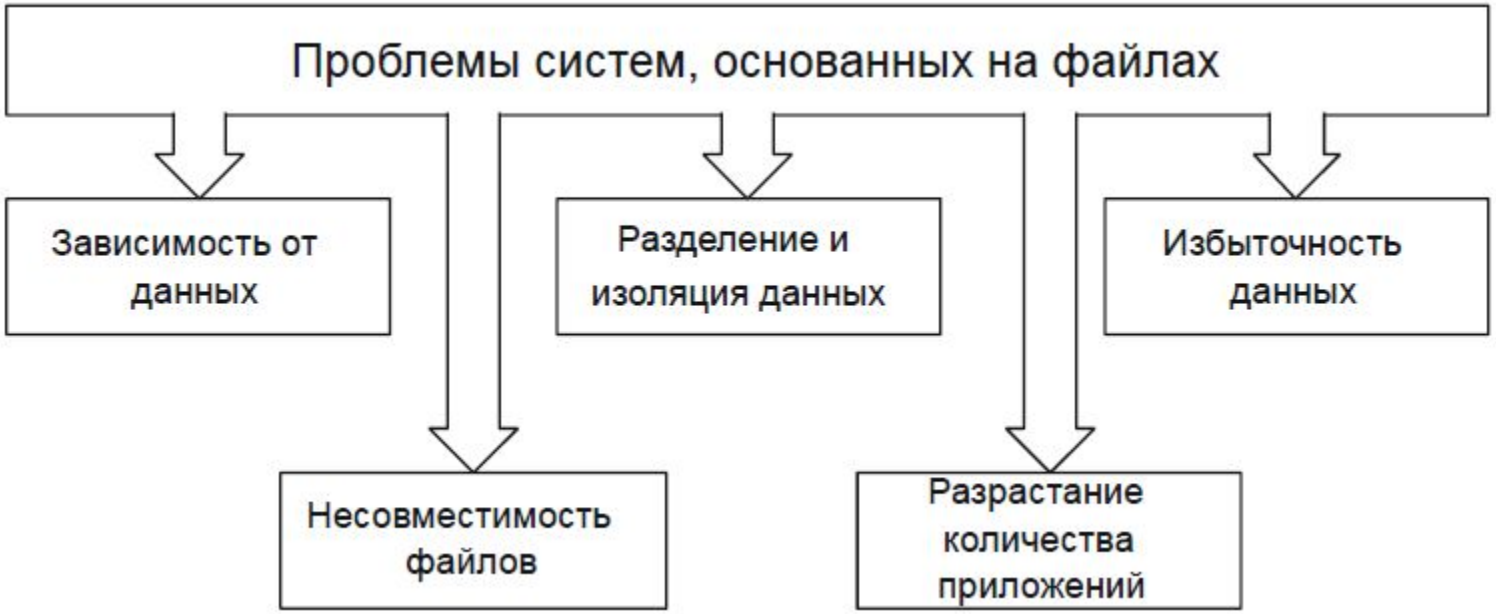
- Во-вторых, программист разрабатывает процедуры, осуществляющие основные операции по работе с файлом. Как минимум это *добавление новой записи, редактирование, удаление и чтение записи*. Ни одну из этих операций невозможно осуществить без знания размерности и состава полей исходной структуры.

Системы, основанные на файлах

- При вставке новой строки в файл к нему следует добавить 20+15+15+8=58 байтов, причем процедура добавления должна знать, что фамилия начинается с 1-го байта, имя с 21-го и т. д. При просмотре файла необходимо осуществлять последовательные операции чтения порциями по 58 байтов, опять же процедура чтения должна иметь информацию, сколько байт отводится тому или иному полю. Как вы догадываетесь, операции редактирования и удаления не являются исключением из правил и также нуждаются в знаниях об исходной структуре. К счастью, эти сведения искать не нужно — все данные о составе полей нашей программе хорошо известны, ведь описание структуры спрятано внутри нее.

Недостатки систем, основанных на файлах

- Поначалу у разработчиков систем, основанных на файлах, дела шли весьма неплохо. Пользователям очень нравилось, что работа с электронными картотеками была схожа с работой с бумажными архивами. Однако, очень скоро стали проявляться отрицательные стороны "лобового" подхода разработчиков первых прототипов баз данных. Из всего сонма недостатков особо выделяются пять проблем.



Недостатки систем, основанных на файлах

- *Зависимость от данных.*
- Уже первая проблема, с которой столкнулись разработчики файловых систем, не предвещала ничего хорошего. Допустим, что нам требуется осуществить элементарную операцию — увеличить на один символ размер поля, отвечающего за хранение фамилии. Оказалось, что даже незначительное усовершенствование структуры влечет за собой ком дополнительных трудностей. Судите сами, изменение размера или состава полей записи типизированного файла вынуждает нас не только перекомпилировать исполняемый файл, но и написать одноразовую программу-конвертор, которая должна преобразовать старые данные к новому формату. То же самое произойдет не только при изменении размерностей, но и в случае, если мы попытаемся добавить новые или удалить лишние поля с данными.
- Таким образом, первой болезнью, поразившей первые прототипы баз данных, стала зависимость системы, основанной на файлах от данных.

Недостатки систем, основанных на файлах

- *Разделение и изоляция данных.*
- Файловые системы объединяют в себе десятки отдельных файлов с данными. В одном файле хранится информация о сотрудниках фирмы, в другом — сведения о клиентах, в третьем — перечень предоставляемых услуг, в четвертом — список заказов и т. д. Для извлечения логически связанных данных (допустим о клиентах и их заказах) программисту приходилось писать сложные алгоритмы синхронного чтения из двух файлов.
- С увеличением числа файлов, вовлекаемых в итоговый отчет, сложность задачи возрастала в арифметической прогрессии. Задача извлечения взаимоувязанных данных из десятка файлов могла стать непосильной не только для программиста, но и для вычислительных машин тех времен. Кроме того, данные могли быть разделены между отделами и службами предприятия — в отделе кадров находились данные о сотрудниках, в отделе продаж — о заказах и т. п. В 1960-х годах удельный вес предприятий, чьи машины были объединены локальными вычислительными сетями, стремился к нулю, поэтому данные были еще и изолированы друг от друга.

Недостатки систем, основанных на файлах

- *Избыточность (дублирование) данных.*
- С усовершенствованием микропроцессорной техники многие руководители предприятий стали отказываться от покупок больших ЭВМ и начали отдавать свои предпочтения более дешевым мини-ЭВМ, расставляя их по отделам и службам своих организаций. Подобное, во многом правильное решение имело и свои отрицательные стороны, одна из них — вынужденный отказ от централизованного хранения данных. Децентрализованное хранение данных систем, основанных на файлах, на нескольких машинах приводила к тому, что одна и та же информация повторялась на магнитных носителях многих мини-ЭВМ, разбросанных по учреждению. В этом случае страшна не столько избыточность данных, сколько нарушение непротиворечивости данных предприятия — на всех машинах должны храниться идентичные копии данных.
- Избыточность данных порождает целый букет проблем и проблемок.

Недостатки систем, основанных на файлах

- *Противоречивость данных.*
- На одной из рабочих станций предприятия хранится устаревший номер телефона вашего контрагента. На второй этого номера вовсе нет. На третьем компьютере, за счет ошибки оператора, там находится телефон его бабушки. В результате ни один из звонков не достигает цели. Как следствие, фирма терпит убытки.

Недостатки систем, основанных на файлах

- *Аномалии данных.*
- Некорректные данные влекут за собой шлейф дополнительных неприятностей: *аномалий добавления, редактирования и удаления записей*. Какая из аномалий способна принести больше печали в наш офис, судите сами. Допустим, у нашей фирмы появился новый, не жалеющий денег, оптовый покупатель. Данные нового клиента следует ввести сразу в несколько систем файлов (отдел сбыта, личная картотека главного менеджера, бухгалтерия и т. д.). Если все сделано безошибочно, то порядок обеспечен… Но если таких покупателей несколько, то можно гарантировать, что где-нибудь, кто-нибудь спутает пару цифр в номерах счетов. В результате платеж в пару миллионов уходит на чужой расчетный счет. После долгого судебного разбирательства и уплаты неустоек вы, наконец, выясняете, в чем причина сбоя, но к этому времени вам уже все равно...

Недостатки систем, основанных на файлах

- *Аномалии данных.*
- С редактированием данных в избыточных системах дела также обстоят далеко не лучшим образом. В идеале следует нанять отдельного сотрудника, задачей которого станет регулярная "пробежка" по всем структурным подразделениям компании, чтобы исправить почтовый адрес (номер телефона, дату рождения, номер счета или что-нибудь в этом духе) главного спонсора фирмы. В результате поздравительная открытка, отправленная в канун очередного юбилея, не попадет к адресату, а в отместку "благодарный" юбиляр не перечислит вашей компании давно обещанные (и так необходимые сейчас) финансовые вливания.

Недостатки систем, основанных на файлах

- *Аномалии данных.*
- Аномалия удаления в состоянии принести не меньшие неприятности. Как вы думаете, как скажется на финансовом состоянии фирмы тот факт, что она станет выплачивать ежегодные премии давно уволенному менеджеру? Это печальное событие произойдет только потому, что в бухгалтерии забудут вычеркнуть всего одну строку с данными.
- В "доисторических" системах файлов избыточность данных вынужденно присутствовала и в рамках одного-единственного проекта. Допустим, что от нас потребуют дополнить программу "Дни рождений сотрудников" еще одним информационным полем — местом работы. В результате в файле появятся многократно дублирующиеся данные, например Петров — Бухгалтерия, Иванов — Бухгалтерия и т. д. Исследования файловых систем тех времен показали, что до 60% хранящихся в них данных избыточны.

Недостатки систем, основанных на файлах

- *Несовместимость файлов.*
- Структура файлов с данными определялась не только разработчиками программного обеспечения, но и языками программирования, применяемыми в тех или иных организациях. Построение файла, описанного на языке Algol, могло принципиально отличаться от структур, генерируемых программами на PL/1, ADA или каким-нибудь еще средством разработки приложений тех времен. Более того, дополнительные ограничения вносились из-за особенностей архитектурных решений тех или иных ЭВМ. Помножьте это на специфичные черты различных операционных систем. В результате файлы, выходящие из-под "пера" программистов, зачастую становились несовместимыми, хотя и содержали практически одно и то же описание данных.

Недостатки систем, основанных на файлах

- *Разрастание количества приложений.*
- Сами по себе данные не представляют никакого интереса. Представьте, что у вас имеется файл с несортированными телефонными номерами жителей миллионного города, но у вас нет средств упорядочивания и поиска данных. В результате цена таким данным — ломаный грош, ну-ка найдите номер телефона гражданина Иванова, затерявшегося где-то среди сотен тысяч других номеров… Данные нужно не только хранить, но и уметь представлять пользователю в удобном формате. А пожеланий у пользователей ни счесть, одним требуется, чтобы списки заказов упорядочивались по алфавиту, другим по дате заказа, третьим хотелось бы, чтобы имелась возможность сортировки записей по денежной сумме. Идеи и пожелания пользователей сыплются на программиста как из рога изобилия и никто кроме него не ведает, что каждый новый запрос приводит к цепной реакции — бесконечной переработке исходного приложения. В конце концов, головная программа обрастает скопищем утилит и "утилиток", что, в свою очередь, необратимо ведет к хаосу.

Пути устранения недостатков систем, основанных на файлах

- Сегодня системы, основанные на файлах, практически не используются, исключение составляют небольшие по числу записей хранилища, состоящие из одного-двух файлов данных. Чтобы не повторять прошлые ошибки, проектировщики БД сделали нужные выводы:
- **Во-первых**, разработчики в принципе отказались от хранения физической структуры данных в коде приложений. Вместо этого описание данных стали выносить в отдельное хранилище, называемое ***системным каталогом*** *(system catalog).*
- Таким образом, во всех современных БД помимо собственно хранимых в них данных еще имеются **метаданные** (данные о данных). Если внешняя программа обладает возможностью чтения метаданных, то она без труда сможет получить доступ к хранимой в БД информации.

Пути устранения недостатков систем, основанных на файлах

- **Во-вторых**, стали предпринимать активные попытки **стандартизировать** способы описания и хранения данных. Наличие стандарта, единого для всех разработчиков, значительно упростило доступ к данным.
- **В-третьих**, возникла необходимость создания единого **универсального языка**, позволяющего производить с данными наиболее важные операции: (вставки, редактирования, удаления и просмотра).
- В результате на смену морально устаревшим системам файлов пришли свободные от недостатков своих предшественников базы данных.

Типы и структуры данных

- **Основные типы данных.**
- Данные, хранящиеся в памяти ЭВМ, представляют собой совокупность нулей и единиц (битов). Биты объединяются в последовательности: байты, слова и т.д. Каждому участку оперативной памяти, который может вместить один байт или слово, присваивается порядковый номер (адрес).
- Какой смысл заключен в данных, какими символами они выражены - буквенными или цифровыми, что означает то или иное число - все это определяется программой обработки. Все данные необходимые для решения практических задач подразделяются на несколько типов, причем понятие **тип** связывается не только с представлением данных в адресном пространстве, но и со **способом их обработки**.

Типы и структуры данных

- Любые данные могут быть отнесены к одному из двух типов: **основному** (**простому**), форма представления которого определяется архитектурой ЭВМ, или **сложному**, конструируемому пользователем для решения конкретных задач.
- Данные простого типа это - символы, числа и т.п. элементы, дальнейшее дробление которых не имеет смысла. Из элементарных данных формируются структуры (сложные типы) данных.

Типы и структуры данных

- Некоторые структуры:
- **Массив** (функция с конечной областью определения) - простая совокупность элементов данных одного типа, средство оперирования группой данных одного типа. Отдельный элемент массива задается индексом. Массив может быть одномерным, двумерным и т.д. Разновидностями одномерных массивов переменной длины являются структуры типа *кольцо, стек, очередь и двухсторонняя очередь*.
- **Запись** (декартово произведение) - совокупность элементов данных разного типа. В простейшем случае запись содержит постоянное количество элементов, которые называют *полями*. Совокупность записей одинаковой структуры называется *файлом*. (Файлом называют также набор данных во внешней памяти, например, на магнитном диске). Для того, чтобы иметь возможность извлекать из файла отдельные записи, каждой записи присваивают уникальное имя или номер, которое служит ее идентификатором и располагается в отдельном поле. Этот идентификатор называют *ключом*.

Типы и структуры данных

- Такие структуры данных как массив или запись занимают в памяти ЭВМ постоянный объем, поэтому их называют статическими структурами. К статическим структурам относится также *множество*.
- Имеется ряд структур, которые могут изменять свою длину - так называемые ***динамические структуры***. К ним относятся дерево, список, ссылка.

- Важной структурой, для размещения элементов которой требуется нелинейное адресное пространство, является **дерево**. Существует большое количество структур данных, которые могут быть представлены как деревья. Это, например, классификационные, иерархические, рекурсивные и др. структуры.

Типы и структуры данных

Классификация типов данных

Обобщенные структуры или модели данных

- Выше мы рассмотрели несколько типов структур, являющихся совокупностями элементов данных: массив, дерево, запись. Более сложный тип данных может включать эти структуры в качестве элементов. Например, элементами записи может быть массив, стек, дерево и т.д.
- Существует большое разнообразие сложных типов данных, но исследования, проведенные на большом практическом материале, показали, что среди них можно выделить несколько наиболее общих. Обобщенные структуры называют также **моделями данных**, т.к. они *отражают представление пользователя о данных реального мира*.

Обобщенные структуры или модели данных

- Любая модель данных должна содержать три компоненты:

- ***структура данных*** - описывает точку зрения пользователя на представление данных.
- ***набор допустимых операций***, выполняемых на структуре данных. Модель данных предполагает, как минимум, наличие языка определения данных (ЯОД), описывающего структуру их хранения, и языка манипулирования данными (ЯМД), включающего операции извлечения и модификации данных.
- ***ограничения целостности*** - механизм поддержания соответствия данных предметной области на основе формально описанных правил.

Обобщенные структуры или модели данных

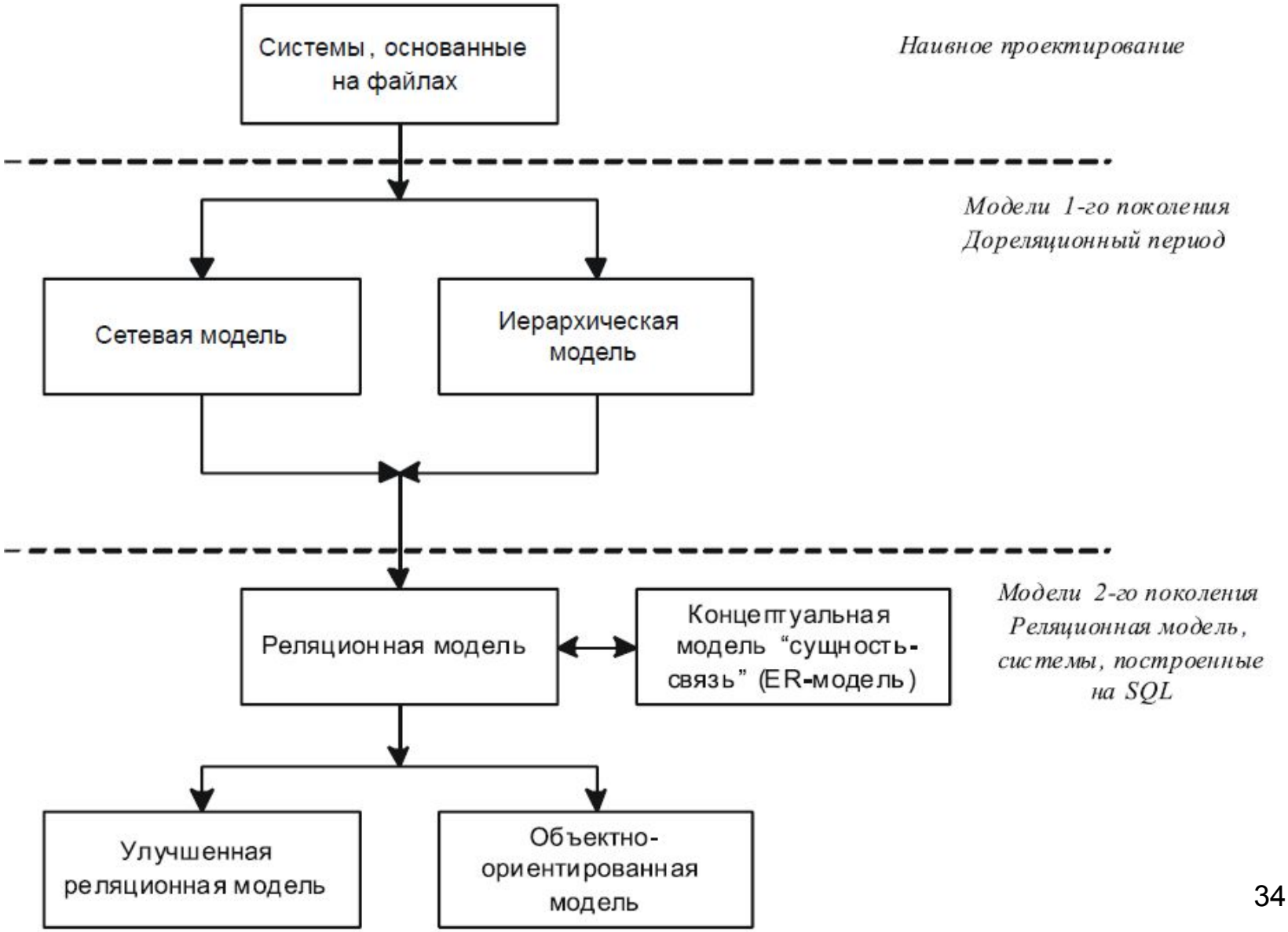
- В процессе исторического развития в СУБД использовалось следующие модели данных:

- иерархическая;
- сетевая;
- реляционная.

- В последнее время все большее значение приобретает объектно-ориентированный подход к представлению данных.

- Следует заметить, что ведущие специалисты в области баз данных до сегодняшнего дня не пришли к единому мнению даже в признании факта существования представленной на рис. объектно-ориентированной модели.

Обобщенные структуры или модели данных



Обобщенные структуры или модели данных

- Это так называемые ***модели реализации***, т. е. модели, ориентированные на получение ответа на вопрос: "Каким образом следует описывать структуры данных?“.
- Единственное исключение составляет ***понятийная модель*** "сущность-связь", это ближайший союзник реляционной модели, но отвечающий не за реализацию, а за логику будущей БД.

Методы доступа к данным

- Вопросы представления данных тесно связаны с операциями, при помощи которых эти данные обрабатываются. К числу таких операций относятся: *выборка, изменение, включение* и *исключение* данных. В основе всех перечисленных операций лежит операция **доступа**, которую нельзя рассматривать независимо от способа представления.
- В задачах поиска предполагается, что все данные хранятся в памяти с определенной идентификацией и, говоря о доступе, имеют в виду прежде всего доступ к данным (называемым ключами), однозначно идентифицирующим связанные с ними совокупности данных.

Методы доступа к данным

- Пусть нам необходимо организовать доступ к файлу, содержащему набор одинаковых записей, каждая из которых имеет уникальное значение ключевого поля. Самый простой способ поиска - последовательно просматривать каждую запись в файле до тех пор, пока не будет найдена та, значение ключа которой удовлетворяет критерию поиска. Очевидно, этот способ весьма неэффективен, поскольку записи в файле не упорядочены по значению ключевого поля. Сортировка записей в файле также неприменима, поскольку требует еще больших затрат времени и должна выполняться после каждого добавления записи. Поэтому, поступают следующим образом - ключи вместе с указателями на соответствующие записи в файле копируют в другую структуру, которая позволяет быстро выполнять операции сортировки и поиска. При доступе к данным вначале в этой структуре находят соответствующее значение ключа, а затем по хранящемуся вместе с ним указателю получают запись из фала.

Методы доступа к данным

- Существуют два класса методов, реализующих доступ к данным по ключу:

- методы поиска по дереву;
- методы хеширования.

Методы поиска по дереву

- ***Определение:*** *Деревом называется конечное множество, состоящее из одного или более элементов, называемых узлами, таких, что:*

- *между узлами имеет место отношение типа "исходный-порожденный " (“родитель-потомок”);*
- *есть только один узел, не имеющий исходного. Он называется корнем;*
- *все узлы за исключением корня имеют только один исходный; каждый узел может иметь несколько порожденных;*
- *отношение "исходный-порожденный" действует только в одном направлении, т.е. ни один потомок некоторого узла не может стать для него предком.*

Методы поиска по дереву

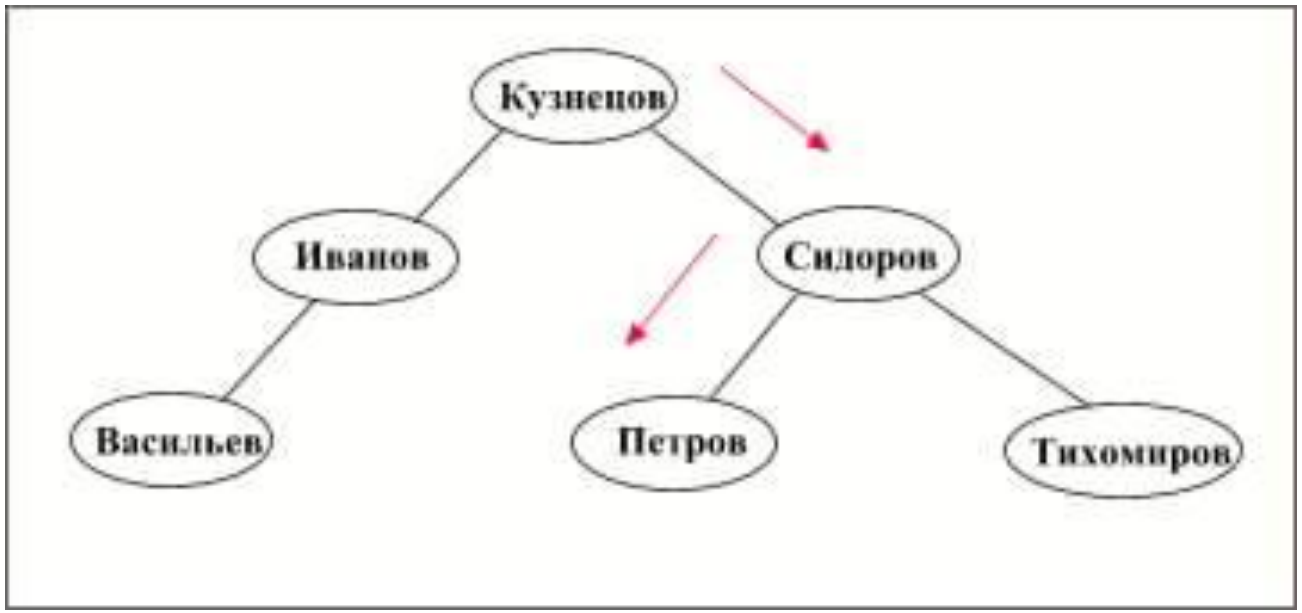
- Число порожденных отдельного узла (число поддеревьев данного корня) называется его *степенью*. Узел с нулевой степенью называют листом или концевым узлом. Максимальное значение степени всех узлов данного дерева называется *степенью дерева*.
- Если в дереве между порожденными узлами, имеющими общий исходный, считается существенным их порядок, то дерево называется *упорядоченным*. В задачах поиска почти всегда рассматриваются упорядоченные деревья.
- Упорядоченное дерево, степень которого не больше 2 называется **бинарным** деревом. Бинарное дерево особенно часто используется при поиске в оперативной памяти. Алгоритм поиска: вначале аргумент поиска сравнивается с ключом, находящимся в корне. Если аргумент совпадает с ключом, поиск закончен, если же не совпадает, то в случае, когда аргумент оказывается меньше ключа, поиск продолжается в левом поддереве, а в случае когда больше ключа - в правом поддереве. Увеличив уровень на 1 повторяют сравнение, считая текущий узел корнем.

Методы поиска по дереву

- **Пример:** *Пусть дан список студентов, содержащий их фамилии и средний бал успеваемости (см. таблицу). В качестве ключа используется фамилия студента. Предположим, что все записи имеют фиксированную длину, тогда в качестве указателя можно использовать номер записи. Смещение записи в файле в этом случае будет вычисляться как ([номер_записи] -1) * [длина_записи]. Пусть аргумент поиска "Петров". На рисунке показаны одно из возможных для этого набора данных бинарных деревьев поиска и путь поиска.*

Студент	Балл
Васильев	4,2
Иванов	3,4
Кузнецов	3,5
Петров	3,2
Сидоров	4,6
Тихомиров	3,8

Методы поиска по дереву



Поиск по бинарному дереву

Заметим, что здесь используется следующее правило сравнения строковых переменных: считается, что значение символа соответствует его порядковому номеру в алфавите. Поэтому "И" меньше "К", а "К" меньше "С". Если текущие символы в сравниваемых строках совпадают, то сравниваются символы в следующих позициях.

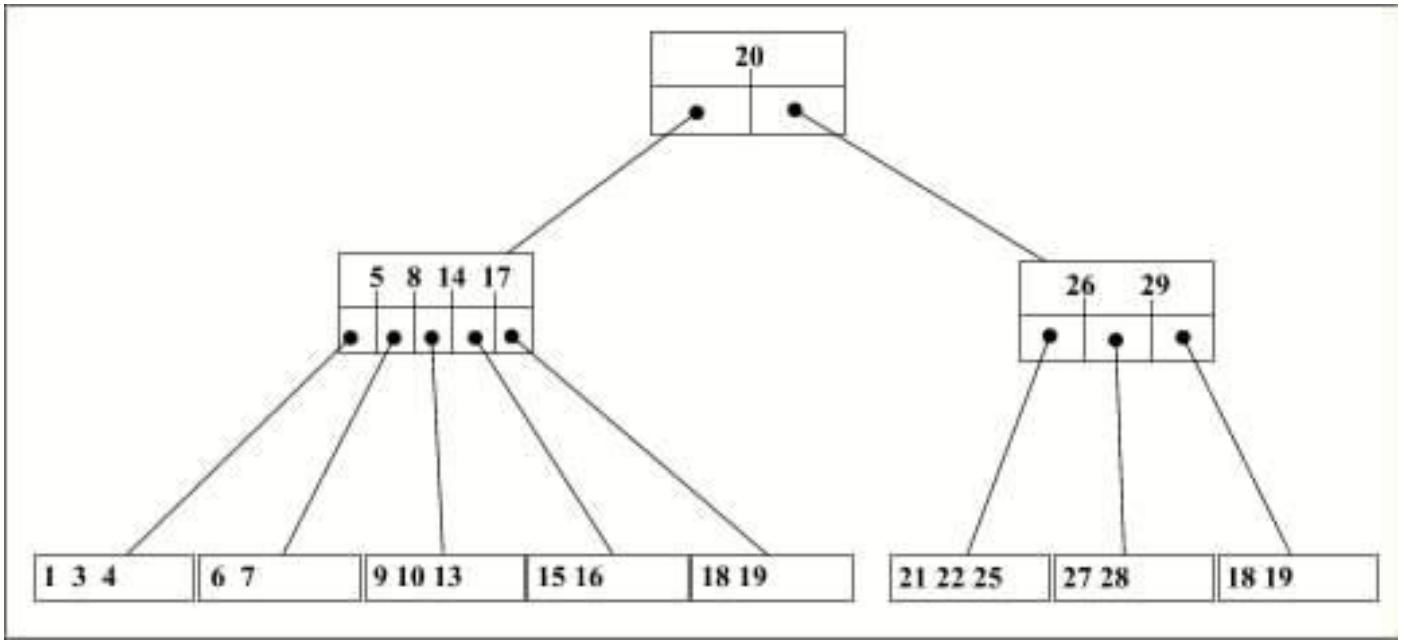
Методы поиска по дереву

- Бинарные деревья особенно эффективны в случае когда множество ключей заранее неизвестно, либо когда это множество интенсивно изменяется. Очевидно, что при переменном множестве ключей лучше иметь **сбалансированное дерево**.
- ***Определение:*** *Бинарное дерево называют сбалансированным (balanced), если высота левого поддерева каждого узла отличается от высоты правого поддерева не более чем на 1.*
- При поиске данных во внешней памяти очень важной является проблема сокращения числа перемещений данных из внешней памяти в оперативную. Поэтому, в данном случае по сравнению с бинарными деревьями более выгодными окажутся сильно ветвящиеся деревья - т.к. их высота меньше, то при поиске потребуется меньше обращений к внешней памяти. Наибольшее применение в этом случае получили B-деревья (B - balanced)

Методы поиска по дереву

- ***Определение:*** *B-деревом порядка n называется сильно ветвящееся дерево степени 2n+1, обладающее следующими свойствами:*
- *Каждый узел, за исключением корня, содержит не менее n и не более 2n ключей;*
- *Корень содержит не менее одного и не более 2n ключей;*
- *Все листья расположены на одном уровне;*
- *Каждый нелистовой узел содержит два списка: упорядоченный по возрастанию значений список ключей и соответствующий ему список указателей (для листовых узлов список указателей отсутствует).*
- Для такого дерева:
- сравнительно просто может быть организован последовательный доступ, т.к. все листья расположены на одном уровне;
- при добавлении и изменении ключей все изменения ограничиваются, как правило, одним узлом.

Методы поиска по дереву



Сбалансированное дерево

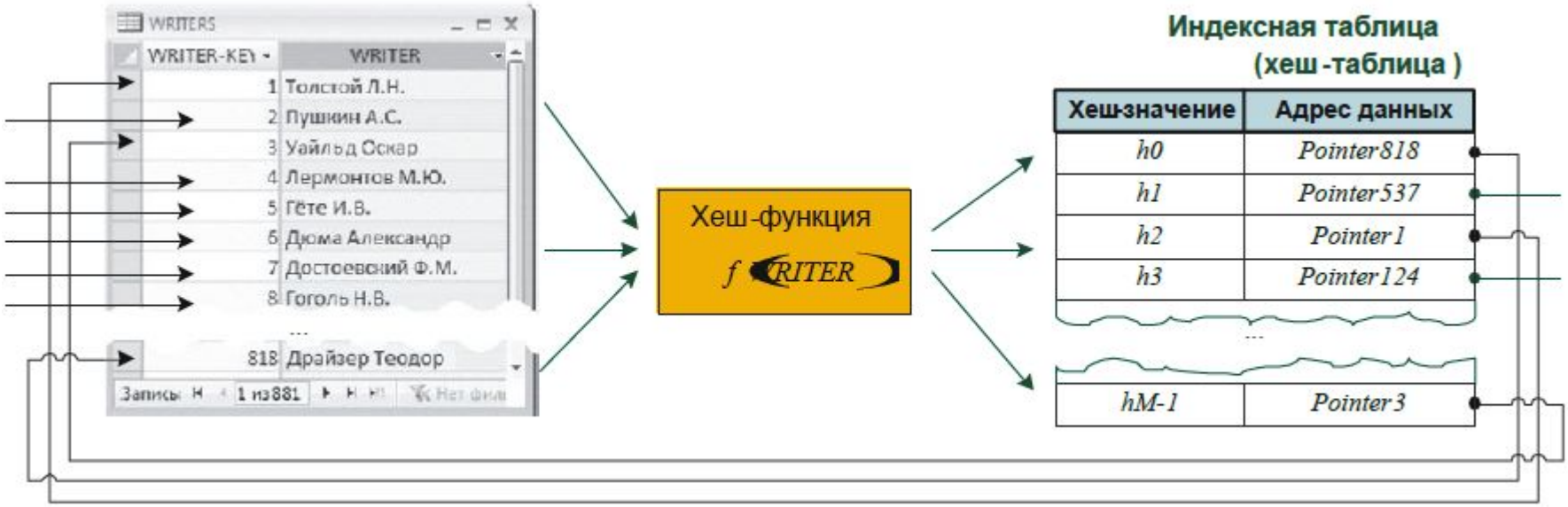
Методы поиска по дереву

- *B*-дерево, в котором истинные значения содержатся только в листьях (концевых узлах), называется *B*+-деревом. Во внутренних узлах такого дерева содержатся ключи-разделители, задающие диапазон изменения ключей для поддеревьев.
- Подробнее о различных видах сбалансированных деревьев, а также методах их реализации можно прочитать в соответствующей литературе. Следует отметить, что *B*- деревья наилучшим образом подходят только для организации доступа к достаточно простым (одномерным) структурам данных. Для доступа к более сложным структурам, таким, например, как пространственные (многомерные) данные в последнее время все чаще используют *R*-деревья.
- *R*-дерево (*R*-Tree) это индексная структура для доступа к пространственным данным, предложенная А.Гуттманом (Калифорнийский университет, Беркли). R-дерево допускает произвольное выполнение операций добавления, удаления и поиска данных без периодической переиндексации.

Хеширование

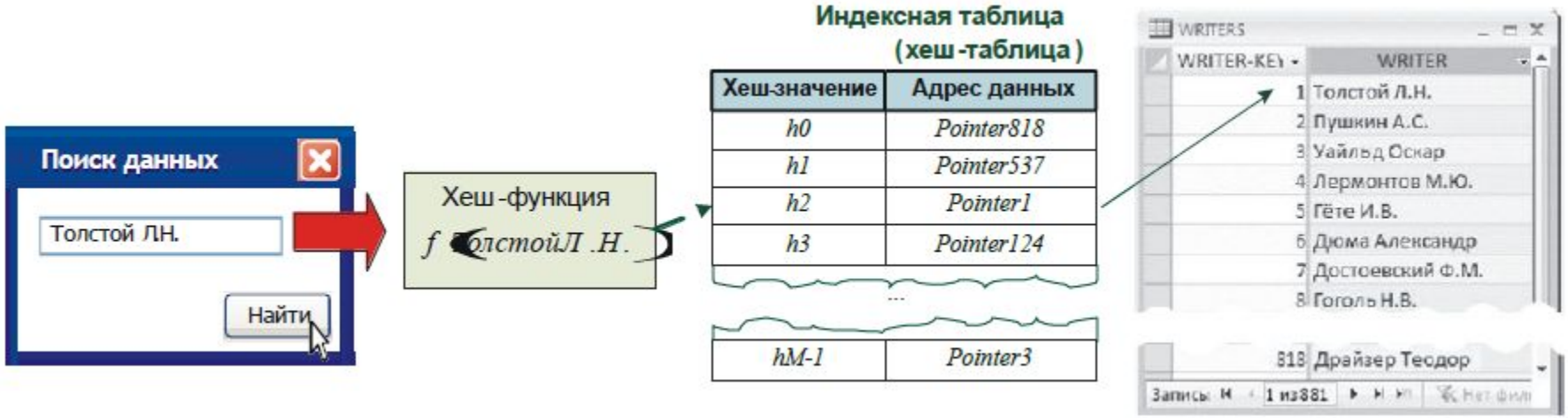
- Этот метод используется тогда, когда все множество ключей заранее известно и на время обработки может быть размещено в оперативной памяти. В этом случае строится специальная функция, однозначно отображающая множество ключей на множество указателей, называемая хеш-функцией (от английского "to hash" - резать, измельчать). Имея такую функцию можно вычислить адрес записи в файле по заданному ключу поиска. В общем случае ключевые данные, используемые для определения адреса записи организуются в виде таблицы, называемой хеш-таблицей.
- Если множество ключей заранее неизвестно или очень велико, то от идеи однозначного вычисления адреса записи по ее ключу отказываются, а хеш-функцию рассматривают просто как функцию, рассеивающую множество ключей во множество адресов.

Хеширование



Построение индексной таблицы на основе хеширования

Хеширование



Поиск данных по индексу

Модель "сущность-связь"

Назначение модели

- Прежде, чем приступать к созданию системы автоматизированной обработки информации, разработчик должен сформировать понятия о предметах, фактах и событиях, которыми будет оперировать данная система. Для того, чтобы привести эти понятия к той или иной модели данных, необходимо заменить их информационными представлениями. Одним из наиболее удобных инструментов унифицированного представления данных, независимого от реализующего его программного обеспечения, является модель "сущность-связь" (entity - relationship model, ER - model).

Модель "сущность-связь"

- Модель "сущность-связь" основывается на некой важной *семантической* информации о реальном мире и предназначена для *логического* представления данных. Она определяет значения данных в контексте их взаимосвязи с другими данными. Важным для нас является тот факт, что из **модели "сущность-связь"** могут быть порождены все существующие **модели данных** (иерархическая, сетевая, реляционная, объектная), поэтому она является наиболее общей.
- *Отметим, что модель "сущность-связь" не является моделью данных в том смысле, как это было определено ранее, поскольку не определяет операций над данными и ограничивается описанием только их логической структуры (т.е это **понятийная модель**).*
- Модель "сущность-связь" была предложена в 1976 г. Питером Пин-Шэн Ченом (Peter Pin-Shan Chen), когда П. Чен опубликовал работу "The Entity-Relationship Model. Toward a Unified View of Data" в научном журнале "Transactions on Database Systems (TODS)".

Элементы модели "сущность-связь"

- Любой фрагмент предметной области может быть представлен как *множество сущностей*, между которыми существует некоторое *множество связей*. Дадим определения:
- **Сущность** (entity) - это объект, который может быть идентифицирован неким способом, отличающим его от других объектов. Примеры: *конкретный человек, предприятие, событие и т.д.*
- **Набор сущностей** (entity set) - множество сущностей одного типа (обладающих одинаковыми свойствами). Примеры: *все люди, предприятия, праздники и т.д.* Наборы сущностей не обязательно должны быть непересекающимися. Например, сущность, принадлежащая к набору МУЖЧИНЫ, также принадлежит набору ЛЮДИ.

Элементы модели "сущность-связь"

- Сущность фактически представляет из себя множество **атрибутов**, которые описывают свойства всех членов данного набора сущностей.
- **Пример:**
  рассмотрим множество работников некого предприятия. Каждого из них можно описать с помощью характеристик *табельный номер, имя, возраст*. Поэтому, сущность СОТРУДНИК имеет атрибуты ТАБЕЛЬНЫЙ_НОМЕР, ИМЯ, ВОЗРАСТ. Используя нотацию языка Pascal этот факт можно представить как:

```
type employe = record
      number : string[6];
      name   : string[50];
      age    : integer;
end;
```

Элементы модели "сущность-связь"

- В дальнейшем для определения сущности и ее атрибутов будем использовать обозначение вида

  СОТРУДНИК (ТАБЕЛЬНЫЙ_НОМЕР, ИМЯ, ВОЗРАСТ).
- Например отделы, на которые подразделяется предприятие, и в которых работают сотрудники, можно описать как ОТДЕЛ (НОМЕР_ОТДЕЛА, НАИМЕНОВАНИЕ).
- Множество значений (область определения) атрибута называется **доменом**. Например, для атрибута ВОЗРАСТ домен (назовем его ЧИСЛО_ЛЕТ) задается интервалом целых чисел больших нуля, поскольку людей с отрицательным возрастом не бывает.
- В упомянутой статье П.Чена атрибут определяется как *функция, отображающая набор сущностей в набор значений или в декартово произведение наборов значений*. Так атрибут ВОЗРАСТ производит отображение в набор значений (домен) ЧИСЛО_ЛЕТ. Атрибут ИМЯ производит отображение в декартово произведение наборов значений ИМЯ, ФАМИЛИЯ и ОТЧЕСТВО.

Элементы модели "сущность-связь"

- Отсюда определяется **ключ сущности** - группа атрибутов, такая, что отображение набора сущностей в соответствующую группу наборов значений является взаимно-однозначным отображением. Другими словами: ключ сущности - это один или более атрибутов уникально определяющих данную сущность. В нашем примере ключом сущности СОТРУДНИК является атрибут ТАБЕЛЬНЫЙ_НОМЕР (конечно, только в том случае, если все табельные номера на предприятии уникальны).

Элементы модели "сущность-связь"

- **Связь** (relationship) - это ассоциация, установленная между несколькими сущностями. Примеры:

- поскольку каждый сотрудник работает в каком-либо отделе, между сущностями СОТРУДНИК и ОТДЕЛ существует связь "работает в" или ОТДЕЛ-РАБОТНИК;
- так как один из работников отдела является его руководителем, то между сущностями СОТРУДНИК и ОТДЕЛ имеется связь "руководит" или ОТДЕЛ-РУКОВОДИТЕЛЬ;
- могут существовать и связи между сущностями одного типа, например связь РОДИТЕЛЬ - ПОТОМОК между двумя сущностями ЧЕЛОВЕК.

- *(В скобках здесь следует отметить, что в методике проектирования данных есть своеобразное правило хорошего тона, согласно которому сущности обозначаются с помощью имен существительных, а связи - глагольными формами. Данное правило, однако, не является обязательным)*.

Элементы модели "сущность-связь"

- К сожалению, не существует общих правил определения, что считать сущностью, а что связью. В рассмотренном выше примере мы положили, что "руководит" - это связь. Однако, можно рассматривать сущность "руководитель", которая имеет связи "руководит" с сущностью "отдел" и "является" с сущностью "сотрудник".
- Связь также может иметь атрибуты. Например, для связи ОТДЕЛ-РАБОТНИК можно задать атрибут СТАЖ_РАБОТЫ_В_ОТДЕЛЕ.
- **Роль сущности в связи** - функция, которую выполняет сущность в данной связи. Например, в связи РОДИТЕЛЬ-ПОТОМОК сущности ЧЕЛОВЕК могут иметь роли "родитель" и "потомок". Указание ролей в модели "сущность-связь" не является обязательным и служит для уточнения семантики связи.

Элементы модели "сущность-связь"

- **Набор связей** (relationship set) - это отношение между *n* (причем *n* не меньше 2) сущностями, каждая из которых относится к некоторому набору сущностей.
- **Пример:**

сущности		наборы сущностей
e1	принадлежит	E1
e2	принадлежит	E2
. . .		
en	принадлежит	En

тогда [e1,e2,...,en] - набор связей R

Элементы модели "сущность-связь"

- Хотя, строго говоря, понятия "связь" и "набор связей" различны (первая является элементом второго), их, тем не менее, очень часто смешивают. Поэтому, мы, не претендуя на академическую строгость, в дальнейшем также будем часто пользоваться терминами "связь" имея в виду "набор связей" и "сущность" имея в виду "набор сущностей".
- В случае $n=2$, т.е. когда связь объединяет две сущности, она называется **бинарной**. Доказано, что n-арный набор связей ($n>2$) всегда можно заменить множеством бинарных, однако первые лучше отображают семантику предметной области.

Элементы модели "сущность-связь"

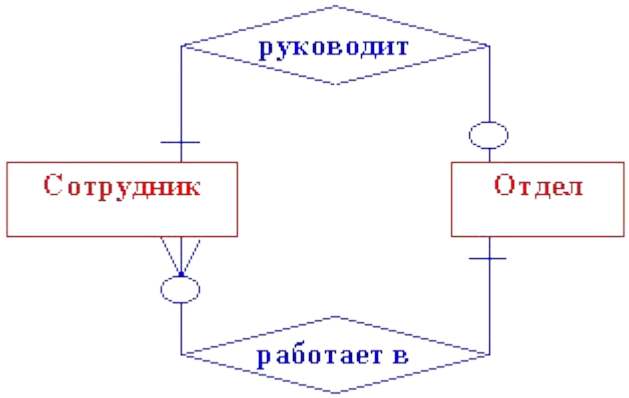
- То число сущностей, которое может быть ассоциировано через набор связей с другой сущностью, называют **степенью связи**. Рассмотрение степеней особенно полезно для бинарных связей. Могут существовать следующие степени бинарных связей:
- ***один к одному*** (обозначается *1 : 1*). Это означает, что в такой связи сущности с одной ролью всегда соответствует не более одной сущности с другой ролью. В рассмотренном нами примере это связь "руководит", поскольку в каждом отделе может быть только один начальник, а сотрудник может руководить только в одном отделе. Данный факт представлен на следующем рисунке, где прямоугольники обозначают сущности, а ромб - связь. Так как степень связи для каждой сущности равна 1, то они соединяются одной линией.

Элементы модели "сущность-связь"

- Другой важной характеристикой связи помимо ее степени является **класс принадлежности** входящих в нее сущностей или **кардинальность** связи. Так как в каждом отделе обязательно должен быть руководитель, то каждой сущности "ОТДЕЛ" непременно должна соответствовать сущность "СОТРУДНИК". Однако, не каждый сотрудник является руководителем отдела, следовательно в данной связи не каждая сущность "СОТРУДНИК" имеет ассоциированную с ней сущность "ОТДЕЛ".
- Таким образом, говорят, что сущность "СОТРУДНИК" имеет *обязательный класс принадлежности* (этот факт обозначается также указанием интервала числа возможных вхождений сущности в связь, в данном случае это 1,1), а сущность "ОТДЕЛ" имеет *необязательный класс принадлежности* (0,1). Теперь данную связь мы можем описать как *0,1:1,1*. В дальнейшем кардинальность бинарных связей степени 1 будем обозначать следующим образом:

Элементы модели "сущность-связь"

- ***один ко многим*** (*1 : n*). В данном случае сущности с одной ролью может соответствовать любое число сущностей с другой ролью. Такова связь ОТДЕЛ-СОТРУДНИК. В каждом отделе может работать произвольное число сотрудников, но сотрудник может работать только в одном отделе. Графически степень связи *n* отображается "древообразной" линией, так это сделано на следующем рисунке.
- Данный рисунок дополнительно иллюстрирует тот факт, что между двумя сущностями может быть определено несколько наборов связей.



Элементы модели "сущность-связь"

- Здесь также необходимо учитывать класс принадлежности сущностей. Каждый сотрудник должен работать в каком-либо отделе, но не каждый отдел (например, вновь сформированный) должен включать хотя бы одного сотрудника. Поэтому сущность "ОТДЕЛ" имеет обязательный, а сущность "СОТРУДНИК" необязательный классы принадлежности. Кардинальность бинарных связей степени *n* будем обозначать так:

Элементы модели "сущность-связь"

- ***много к одному*** (*n : 1*). Эта связь аналогична отображению *1 : n*. Предположим, что рассматриваемое нами предприятие строит свою деятельность на основании контрактов, заключаемых с заказчиками. Этот факт отображается в модели "сущность-связь" с помощью связи КОНТРАКТ-ЗАКАЗЧИК, объединяющей сущности КОНТРАКТ(НОМЕР, СРОК_ИСПОЛНЕНИЯ, СУММА) и ЗАКАЗЧИК (НАИМЕНОВАНИЕ, АДРЕС). Так как с одним заказчиком может быть заключено более одного контракта, то связь КОНТРАКТ-ЗАКАЗЧИК между этими сущностями будет иметь степень *n : 1*.
- В данном случае, по совершенно очевидным соображениям (каждый контракт заключен с конкретным заказчиком, а каждый заказчик имеет хотя бы один контракт, иначе он не был бы таковым), каждая сущность имеет обязательный класс принадлежности.

Элементы модели "сущность-связь"

- ***многие ко многим*** (*n : m*). В этом случае каждая из ассоциированных сущностей может быть представлена любым количеством экземпляров. Пусть на рассматриваемом нами предприятии для выполнения каждого контракта создается рабочая группа, в которую входят сотрудники разных отделов. Поскольку каждый сотрудник может входить в несколько (в том числе и ни в одну) рабочих групп, а каждая группа должна включать не менее одного сотрудника, то связь между сущностями СОТРУДНИК и РАБОЧАЯ_ГРУППА имеет степень *n : m*.

Элементы модели "сущность-связь"

- Если существование сущности *x* зависит от существования сущности *y*, то *x* называется **зависимой сущностью** (иногда сущность *x* называют "слабой", а "сущность" *y - сильной). В качестве примера рассмотрим связь между ранее описанными сущностями РАБОЧАЯ_ГРУППА и КОНТРАКТ. Рабочая группа создается только после того, как будет подписан контракт с заказчиком, и прекращает свое существование по выполнению контракта. Таким образом, сущность РАБОЧАЯ_ГРУППА является зависимой от сущности КОНТРАКТ. Зависимую сущность будем обозначать двойным прямоугольником, а ее связь с сильной сущностью линией со стрелкой:*

Элементы модели "сущность-связь"

- *Заметим, что кардинальность связи для сильной сущности всегда будет (1,1). Класс принадлежности и степень связи для зависимой сущности могут быть любыми. Предположим, например, что рассматриваемое нами предприятие пользуется несколькими банковскими кредитами, которые представляются набором сущностей КРЕДИТ(НОМЕР_ДОГОВОРА,СУММА, СРОК_ПОГАШЕНИЯ, БАНК). По каждому кредиту должны осуществляться выплаты процентов и платежи в счет его погашения. Этот факт представляется набором сущностей ПЛАТЕЖ(ДАТА, СУММА) и набором связей "осуществляется по". В том случае, когда получение запланированного кредита отменяется, информация о нем должна быть удалена из базы данных. Соответственно, должны быть удалены и все сведения о плановых платежах по этому кредиту. Таким образом, сущность ПЛАТЕЖ зависит от сущности КРЕДИТ.*

Диаграмма "сущность-связь".

- Очень важным свойством модели "сущность-связь" является то, что она может быть представлена в виде графической схемы. Это значительно облегчает анализ предметной области. Существует несколько вариантов обозначения элементов диаграммы "сущность-связь", каждый из которых имеет свои положительные черты. Краткий обзор некоторых из этих нотаций будет сделан далее. Здесь мы будем использовать некий гибрид нотаций Чена (обозначение сущностей, связей и атрибутов) и Мартина (обозначение степеней и кардинальностей связей). В таблице приводится список используемых здесь обозначений.

Диаграмма "сущность-связь".

Используемые обозначения модели "сущность-связь"

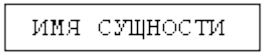
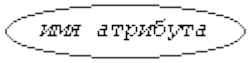
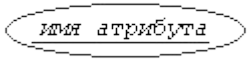
Обозначение	Значение
	Набор независимых сущностей
	Набор зависимых сущностей
	Атрибут
	Ключевой атрибут
	Набор связей

Диаграмма "сущность-связь".

- Атрибуты с сущностями и сущности со связями соединяются прямыми линиями. При этом для указания кардинальностей связей используются обозначения, введенные в предыдущем параграфе.
- В процессе построения диаграммы можно выделить несколько очевидных этапов:
  1. Идентификация представляющих интерес сущностей и связей.
  2. Идентификация семантической информации в наборах связей (например, является ли некоторый набор связей отображением *1:n*).
  3. Определение кардинальностей связей.
  4. Определение атрибутов и наборов их значений (доменов).
  5. Организация данных в виде отношений "сущность-связь".

В качестве примера построим диаграмму, отображающую связь данных для подсистемы учета персонала предприятия.

Диаграмма "сущность-связь".

- **Выделим интересующие нас сущности и связи:**

1. Прежде всего предприятие состоит из отделов, в которых работают сотрудники. Оклад каждого сотрудника зависит от занимаемой им должности (инженер, ведущий инженер, бухгалтер, уборщик и т.д.). Далее предположим, что на нашем предприятии допускается совместительство должностей, т.е. каждый сотрудник может иметь более чем одну должность (и работать более чем в одном отделе), причем может занимать неполную ставку. В то же время, одну и ту же должность могут занимать одновременно несколько сотрудников. В результате этих рассуждений мы должны ввести наборы сущностей
   - ОТДЕЛ(ИМЯ_ОТДЕЛА),
   - СОТРУДНИК(ТАБЕЛЬНЫЙ_НОМЕР, ИМЯ),
   - ДОЛЖНОСТЬ(ИМЯ_ДОЛЖНОСТИ, ОКЛАД),

   и набор связей РАБОТАЕТ_В с атрибутом ставка между ними. Атрибут ставка может принимать значения из интервала [0,1] (больше нуля, но меньше или равен единице), он определяет какую часть должностного оклада получает данный сотрудник.

Диаграмма "сущность-связь".

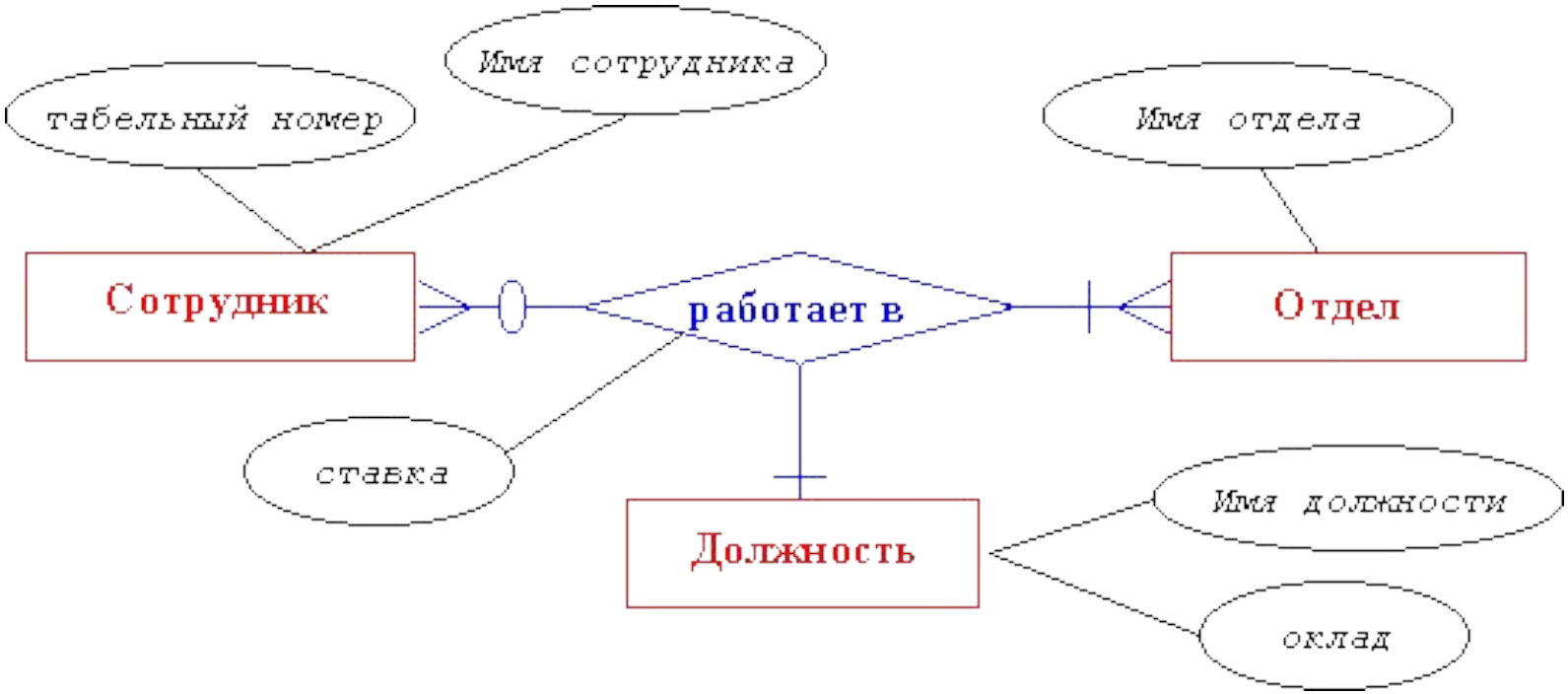


Диаграмма "сущность-связь".

- Как уже отмечалось выше, каждый *n*-арный набор связей можно заменить несколькими бинарными наборами. Сейчас как раз представляется удобный случай, чтобы оценить преимущества каждого из этих способов представления связей.
  - Тренарная связь, показанная здесь, безусловно несет более полную информацию о предметной области. Действительно, она однозначно отображает тот факт, что оклад сотрудника зависит от его должности, отдела, где он работает, и ставки. Однако, в этом случае возникают некоторые проблемы с определением степени связи. Хотя, как было сказано, каждый работник может занимать несколько должностей, а в штате каждого отдела существуют вакансии с различными должностями, тем не менее класс принадлежности сущности ДОЛЖНОСТЬ на приведенном рисунке установлен в (1,1). Это объясняется тем, что ДОЛЖНОСТЬ ассоциируется фактически не с сущностями СОТРУДНИК и ОТДЕЛ, а со связью между ними. Обозначать этот факт предлагается так, как это показано на следующей диаграмме:

Диаграмма "сущность-связь".

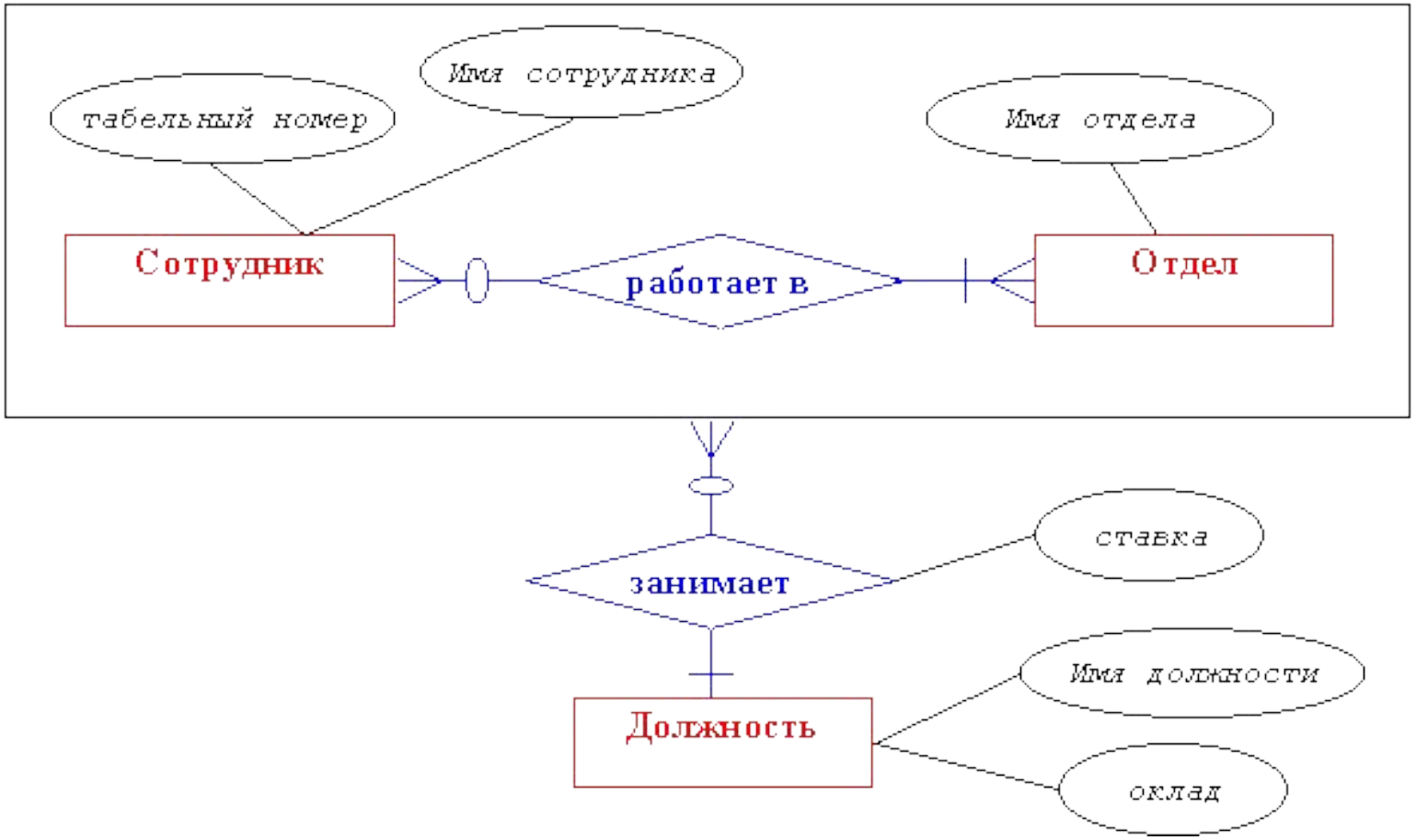


Диаграмма "сущность-связь".

- Здесь сущности СОТРУДНИК, ОТДЕЛ и связь РАБОТАЕТ_В агрегируются в некую новую абстрактную сущность, которая ассоциируется с сущностью ДОЛЖНОСТЬ с помощью связи степени n:1.
  - Попытаемся отобразить ассоциации сотрудников, отделов и должностей с помощью бинарных связей.

Диаграмма "сущность-связь".

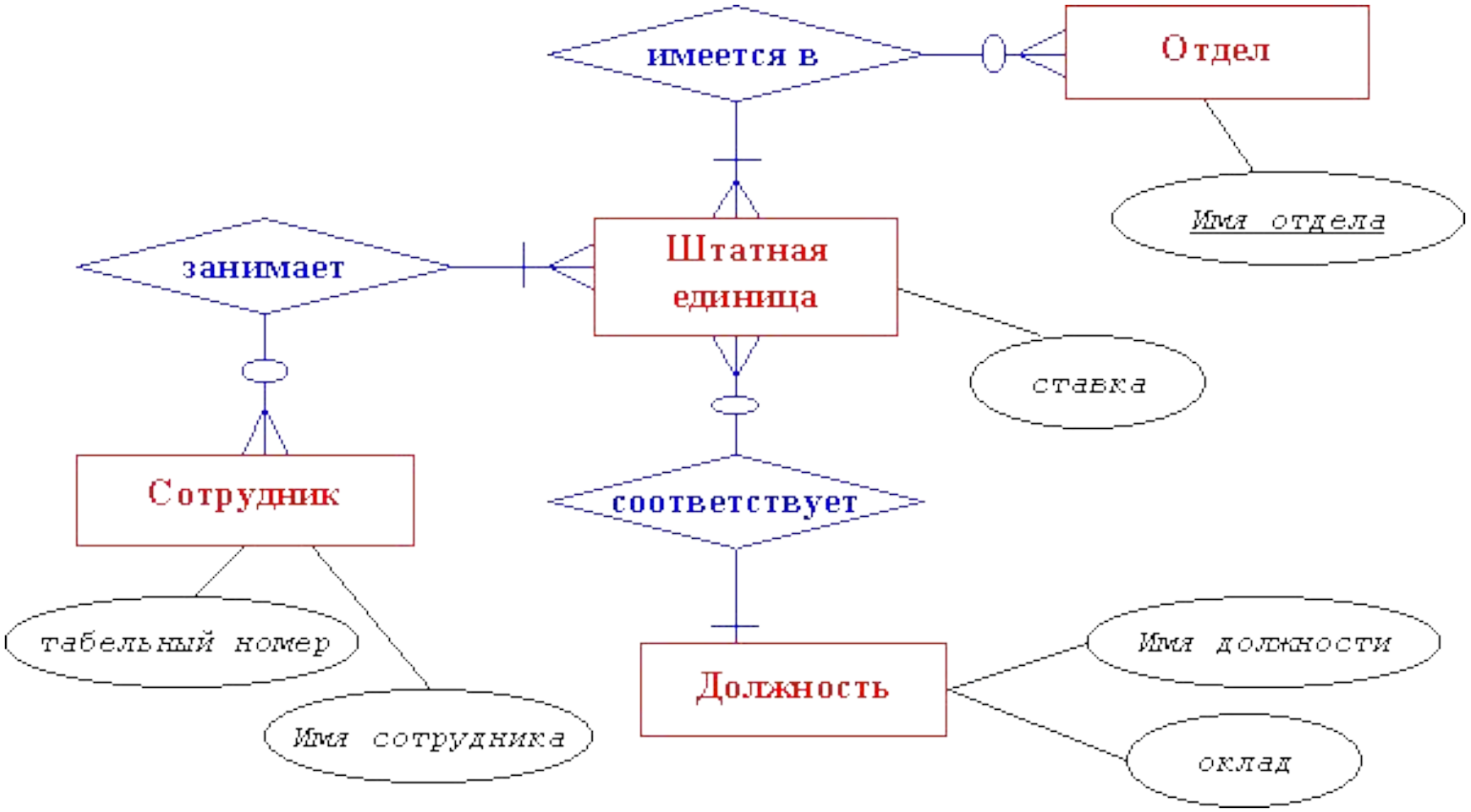


Диаграмма "сущность-связь".

- В этом случае для адекватного описания семантики предметной области необходимо ввести еще одну сущность ШТАТНАЯ_ЕДИНИЦА, которая фактически заменяет собой связь РАБОТАЕТ_В в абстрактной сущности и поэтому имеет атрибут *ставка*.
- Переход от *n*-арной связи через агрегацию сущностей к набору бинарных связей можно рассматривать как последовательные этапы одного процесса, который приводит к однозначному порождению реляционной модели данных. При построении диаграммы "сущность- связь" можно использовать любой из этих трех способов представления данных.

Диаграмма "сущность-связь".

2. Перечислим ряд объектов, описанных в предыдущем параграфе, которые будут полезны при моделировании данных рассматриваемого предприятия. Им соответствуют следующие сущности:
   - ЗАКАЗЧИК(ИМЯ_ЗАКАЗЧИКА,АДРЕС)
   - КОНТРАКТ(НОМЕР,СРОК_НАЧАЛА,СРОК_ОКОНЧАНИЯ,СУММА)
   - РАБОЧАЯ ГРУППА(ПРОЦЕНТ_ВОЗНАГРАЖДЕНИЯ)

- Атрибут *"процент_вознаграждения"* отражает ту долю стоимости контракта, которая предназначена для оплаты труда членов соответствующей рабочей группы. Смысл остальных атрибутов понятен без дополнительных пояснений. Связи между перечисленными сущностями также описаны в предыдущем параграфе.

Диаграмма "сущность-связь".

- Как правило, один из членов рабочей группы является руководителем по отношению к другим сотрудникам, входящим в ее состав. Для отражения этого факта мы должны ввести связь "руководит" с кардинальностью *1,1:0,n* между сущностями СОТРУДНИК и РАБОЧАЯ_ГРУППА (сотрудник может руководить в произвольном числе рабочих групп, но каждая рабочая группа имеет одного и только одного руководителя).

3. Рассмотрим теперь более внимательно информационный объект "заказчик". На практике очень часто возникает необходимость различать национальную принадлежность юридических лиц, с которыми предприятие вступает в договорные отношения. Это связано с тем, что для зарубежных фирм необходимо хранить, например, сведения о валюте, в которой осуществляются расчеты, языке, на котором подписан контракт и т.д. В свою очередь, для отечественных компаний необходимо иметь сведения о их форме собственности (частная или государственная), поскольку от этого может зависеть порядок налогообложения средств, полученных за выполнение работ по контракту.

Диаграмма "сущность-связь".

- Таким образом, мы приходим к выводу, что необходимо ввести в рассмотрение еще два непересекающихся множества ЗАРУБЕЖНОЕ_ПРЕДПРИЯТИЕ(ВАЛЮТА, ЯЗЫК) и ОТЕЧЕСТВЕННОЕ_ПРЕДПРИЯТИЕ(ФОРМА_СОБСТВЕННОСТИ), объединение которых составляет полное множество ЗАКАЗЧИК. Ассоциацию между этими объектами называют **отношением наследования** или **иерархической связью**, так как сущности ЗАРУБЕЖНОЕ_ПРЕДПРИЯТИЕ и ОТЕЧЕСТВЕННОЕ_ПРЕДПРИЯТИЕ наследуют атрибуты сущности ЗАКАЗЧИК(ИМЯ_ЗАКАЗЧИКА, АДРЕС). Для того, чтобы определить к какому подмножеству относится конкретная сущность из набора ЗАКАЗЧИК (и, соответственно, какой набор атрибутов она имеет) необходимо ввести атрибут *"национальная принадлежность", называемый **дискриминантом**. Этот тип связи предлагается отображать на диаграмме следующим образом:*

Диаграмма "сущность-связь".

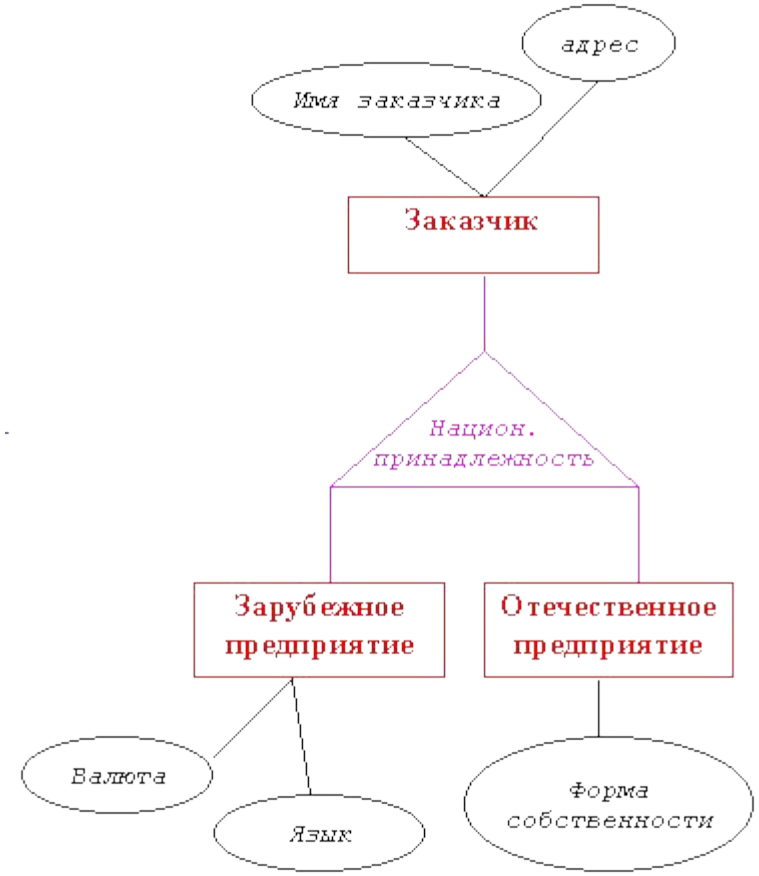
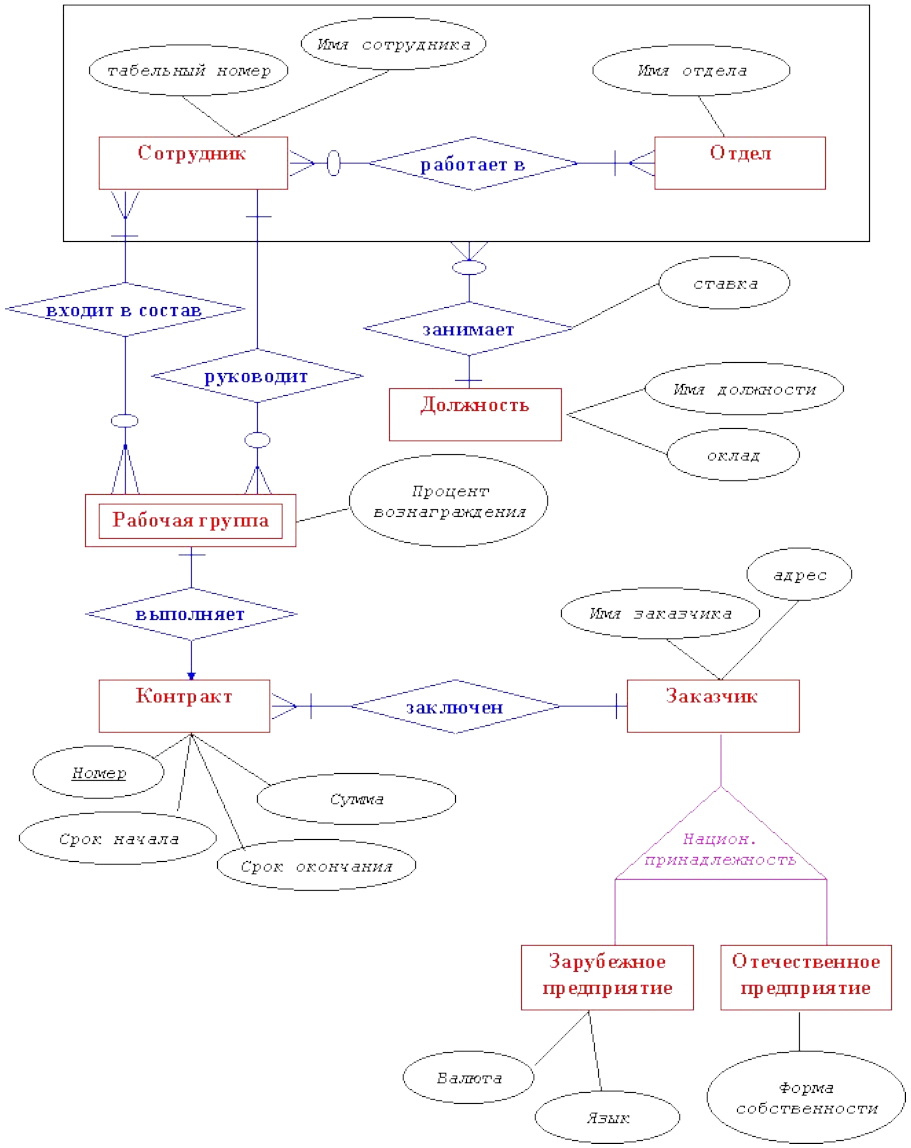


Диаграмма "сущность-связь".

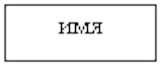
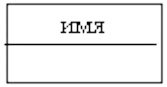
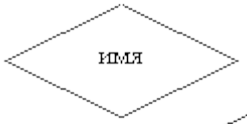
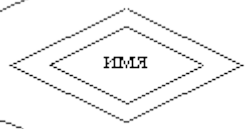


Целостность данных

- Модель "сущность-связь" также полезна для понимания и спецификации ограничений, направленных на поддержание целостности данных. В модели имеется три типа ограничений на значения:
  1. ограничения на *допустимые* значения в *наборе значений (домене)*. Домен можно трактовать как область определения атрибута, которая может быть задана либо непрерывным или дискретным интервалом, либо фиксированным списком значений.
  2. ограничения на *разрешенные* значения для *каждого атрибута*. Например, возраст сотрудников может быть ограничен интервалом от 18 до 65 лет.
  3. ограничения на *существующие* значения в *базе данных*. Например, сумма отчислений с зарплаты сотрудника не должна превышать самой зарплаты.
- К сожалению, указанные ограничения невозможно представить на диаграмме "сущность - связь".

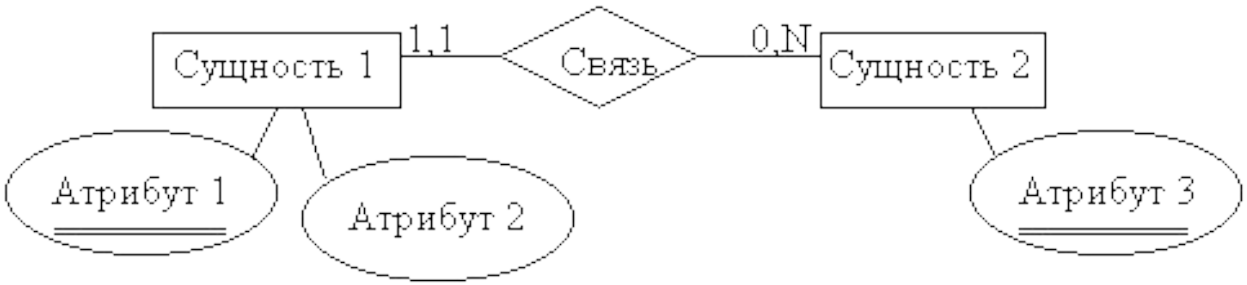
Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация Чена

Элемент диаграммы	Обозначает
	независимая сущность
	зависимая сущность
	родительская сущность в иерархической связи
	Связь
	идентифицирующая связь
	Атрибут
	первичный ключ
	внешний ключ (понятие внешнего ключа вводится в реляционной модели данных)
	многозначный атрибут
	получаемый (наследуемый) атрибут в иерархических связях

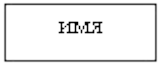
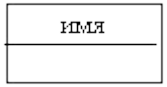
Обзор нотаций, используемых при построении диаграмм "сущность-связь"

- Связь соединяется с ассоциируемыми сущностями линиями. Возле каждой сущности на линии, соединяющей ее со связью, цифрами указывается класс принадлежности. Пример:



Обзор нотаций, используемых при построении диаграмм "сущность-связь"

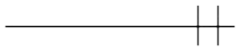
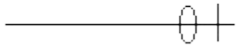
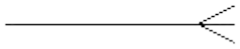
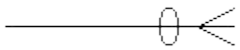
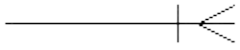
Нотация Мартина

Элемент диаграммы	Обозначает
	независимая сущность
	зависимая сущность
	родительская сущность в иерархической связи

Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация Мартина

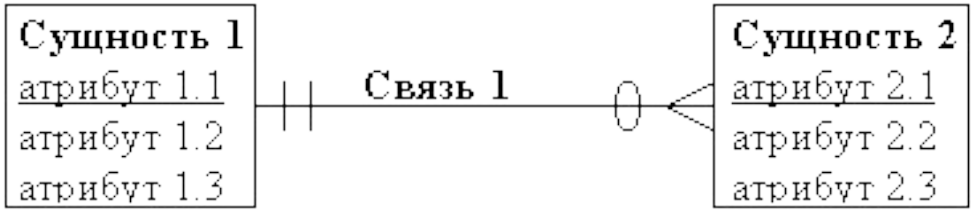
- Список атрибутов приводится внутри прямоугольника, обозначающего сущность. Ключевые атрибуты подчеркиваются. Связи изображаются линиями, соединяющими сущности, вид линии в месте соединения с сущностью определяет кардинальность связи:

Обозначение	Кардинальность
	нет
	1,1
	0,1
	M,N
	0,N
	1,N

Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация Мартина

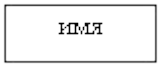
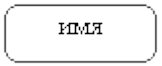
- Имя связи указывается на линии ее обозначающей. Пример:



Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация IDEF1X

- Обозначения сущностей:

Элемент диаграммы	Обозначает
	независимая сущность
	зависимая сущность

Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация IDEF1X

- Список атрибутов приводится внутри прямоугольника, обозначающего сущность. Атрибуты, составляющие ключ сущности, группируются в верхней части прямоугольника и отделяются горизонтальной чертой.
- Обозначения связей:

Элемент диаграммы	Обозначает
———————	идентифицирующая связь
- - - - - - - - - - - -	Не идентифицирующая связь

Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация IDEF1X

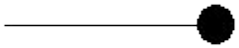
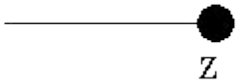
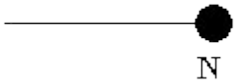
- Список атрибутов приводится внутри прямоугольника, обозначающего сущность. Атрибуты, составляющие ключ сущности, группируются в верхней части прямоугольника и отделяются горизонтальной чертой.
- Обозначения связей:

Элемент диаграммы	Обозначает
———————	идентифицирующая связь
- - - - - - - - - - - -	Не идентифицирующая связь

Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация IDEF1X

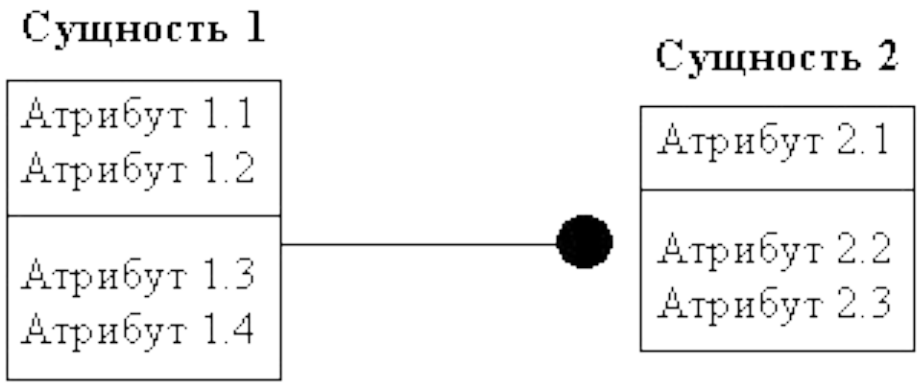
- Обозначение кардинальности связей:

Элемент диаграммы	Обозначает
	1,1
	0,M
	0,1
	1,M
	точно *N* (*N* - произвольное число)

Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация IDEF1X

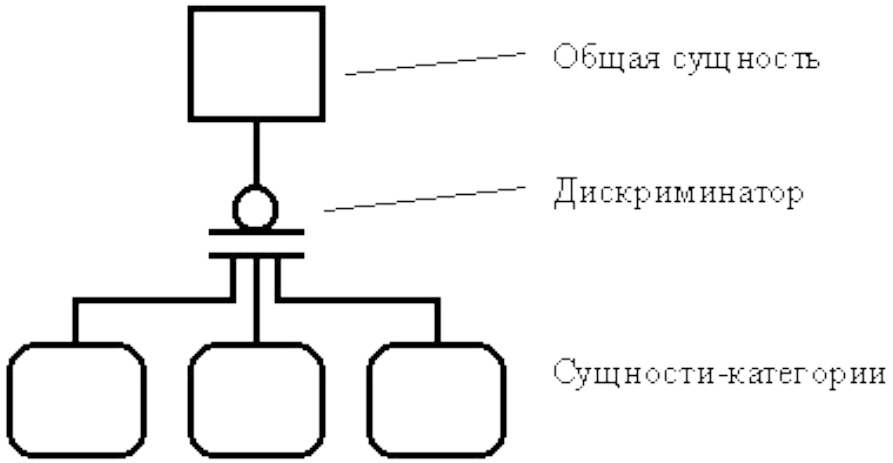
- Пример:



Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация IDEF1X

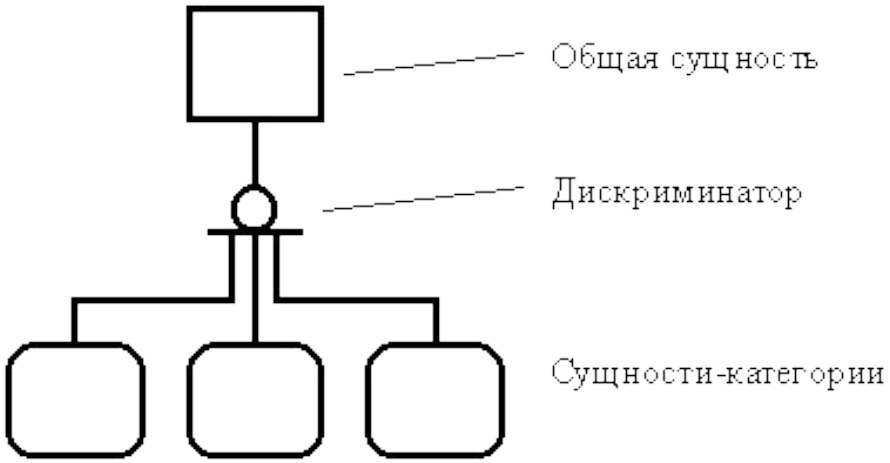
- Кроме того, в IDEF1X вводится понятие “отношение категоризации”, по смыслу эквивалентное рассмотренной нами иерархической связи. Отношение полной категоризации (сущности-категории составляют полное множество потомков родительской сущности) обозначается:



Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация IDEF1X

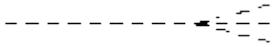
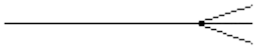
- Также может существовать отношение неполной категоризации (сущности-категории составляют неполное множество потомков общей сущности):



Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация Баркера

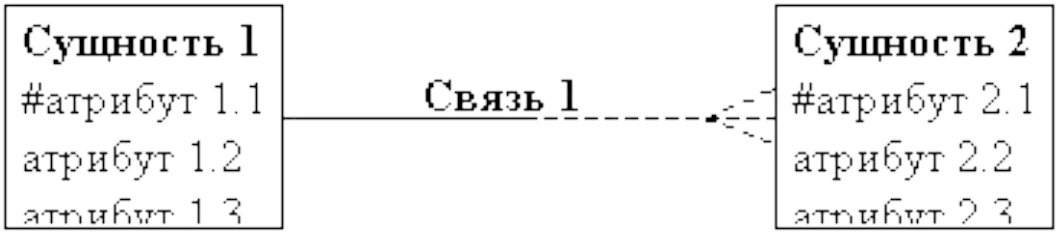
- Сущности обозначаются прямоугольниками, внутри которых приводится список атрибутов. Ключевые атрибуты отмечаются символом # (решетка). Связи обозначаются линиями с именами, место соединения связи и сущности определяет кардинальность связи:

Обозначение	Кардинальность
	0,1
	1,1
	0,N
	1,N

Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация Баркера

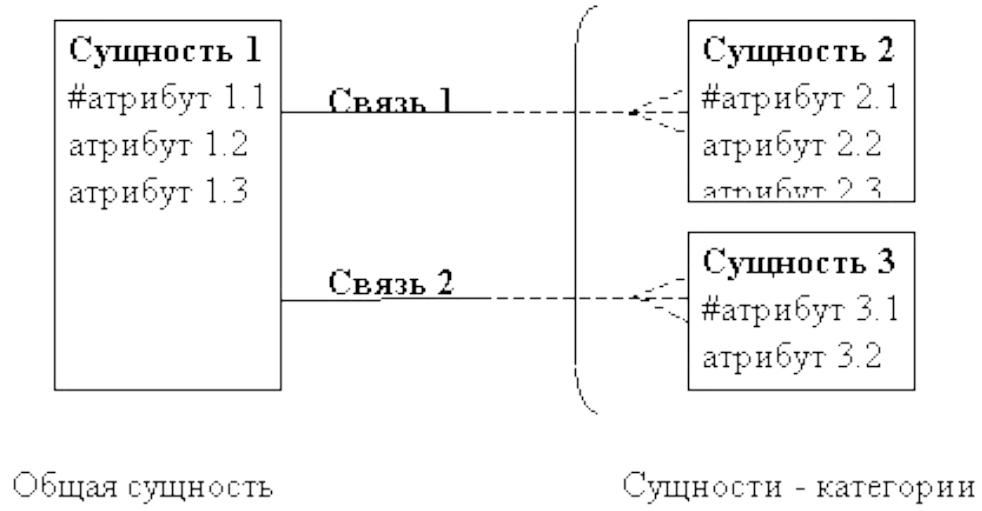
- Пример:



Обзор нотаций, используемых при построении диаграмм "сущность-связь"

Нотация Баркера

- Для обозначения отношения категоризации вводится элемент "дуга":



Иерархическая модель данных

Структура данных

- Организация данных в СУБД иерархического типа определяется в терминах: *элемент, агрегат, запись (группа), групповое отношение, база данных.*

- ***Атрибут (элемент данных)*** - наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем.
- ***Запись*** - именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. Экземпляр записи - конкретная запись с конкретным значением элементов.

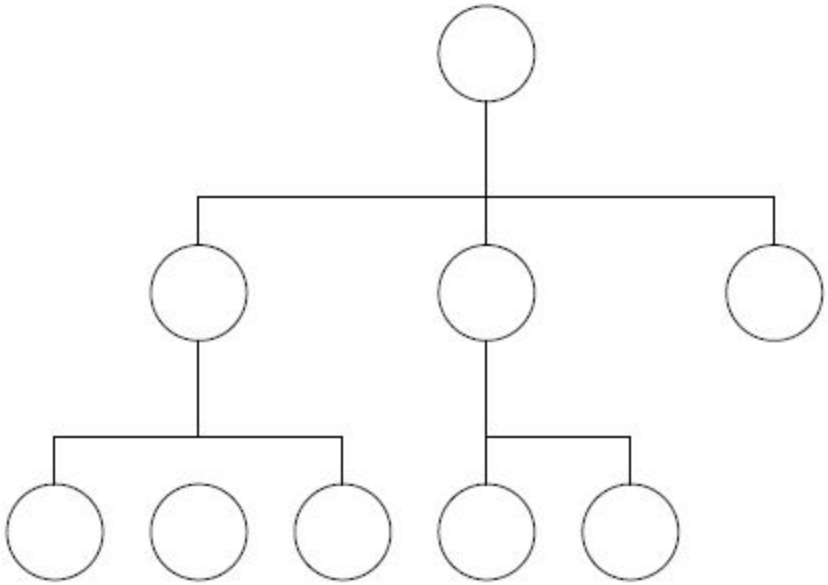
Иерархическая модель данных

- ***Групповое отношение*** - иерархическое отношение между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) - подчиненными. Иерархическая база данных может хранить только такие древовидные структуры.

Иерархическая модель данных

- Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения. Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой по иерархическому пути.
- При графическом изображении групповые отношения изображают дугами ориентированного графа, а типы записей - вершинами (диаграмма Бахмана).
- Для групповых отношений в иерархической модели обеспечивается *автоматический режим включения* и *фиксированное членство*. Это означает, что для запоминания любой некорневой записи в БД должна существовать ее родительская запись (см. сетевую модель). При удалении родительской записи автоматически удаляются все подчиненные.

Иерархическая модель данных



Иерархическая модель данных

Иерархическая модель данных

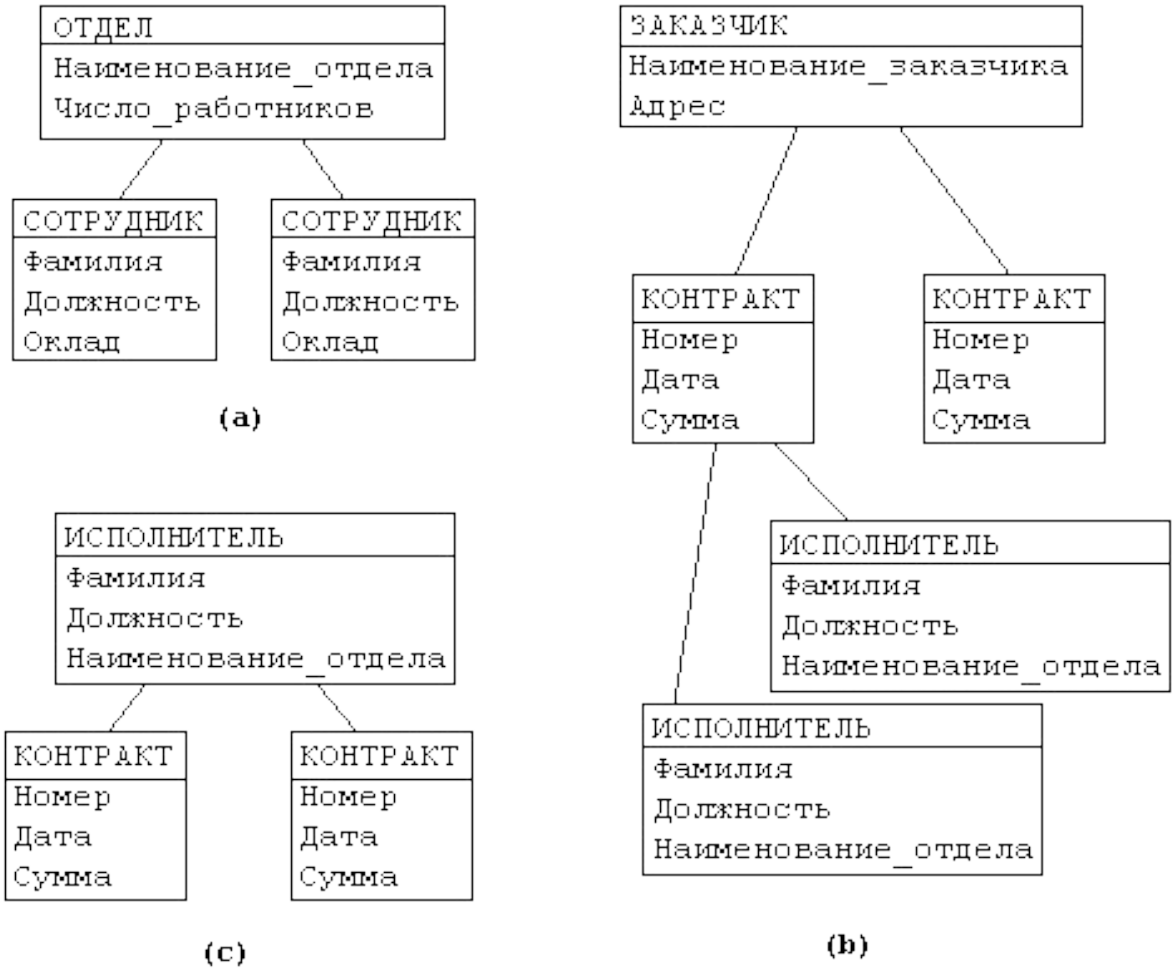
- **Пример:**
- *Рассмотрим следующую модель данных предприятия (см. **рис.**): предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников, но сотрудник не может работать более чем в одном отделе.*
- *Поэтому, для информационной системы управления персоналом необходимо создать групповое отношение, состоящее из родительской записи ОТДЕЛ (НАИМЕНОВАНИЕ_ОТДЕЛА, ЧИСЛО_РАБОТНИКОВ) и дочерней записи СОТРУДНИК (ФАМИЛИЯ, ДОЛЖНОСТЬ, ОКЛАД). Это отношение показано на рис. **(а)** (Для простоты полагается, что имеются только две дочерние записи).*

Иерархическая модель данных

- **Пример:**
- *Для автоматизации учета контрактов с заказчиками необходимо создание еще одной иерархической структуры : заказчик - контракты с ним - сотрудники, задействованные в работе над контрактом. Это дерево будет включать записи ЗАКАЗЧИК (НАИМЕНОВАНИЕ_ЗАКАЗЧИКА, АДРЕС), КОНТРАКТ(НОМЕР, ДАТА,СУММА), ИСПОЛНИТЕЛЬ (ФАМИЛИЯ, ДОЛЖНОСТЬ, НАИМЕНОВАНИЕ_ОТДЕЛА) (рис. **(b)**)*.

Иерархическая модель данных

- **Пример:**



Иерархическая модель данных

- Из этого примера видны недостатки иерархических БД:
  - Частично дублируется информация между записями СОТРУДНИК и ИСПОЛНИТЕЛЬ (такие записи называют парными), причем в иерархической модели данных не предусмотрена поддержка соответствия между парными записями.
  - Иерархическая модель реализует отношение между исходной и дочерней записью по схеме **1:N**, то есть одной родительской записи может соответствовать любое число дочерних. Допустим теперь, что исполнитель может принимать участие более чем в одном контракте (т.е. возникает связь типа **M:N**). В этом случае в базу данных необходимо ввести еще одно групповое отношение, в котором ИСПОЛНИТЕЛЬ будет являться исходной записью, а КОНТРАКТ - дочерней (рис. **(c)**). Таким образом, мы опять вынуждены дублировать информацию.

Иерархическая модель данных

Операции над данными, определенные в иерархической модели:

- **ДОБАВИТЬ** в базу данных новую запись. Для корневой записи обязательно формирование значения ключа.
- **ИЗМЕНИТЬ** значение данных предварительно извлеченной записи. Ключевые данные не должны подвергаться изменениям.
- **УДАЛИТЬ** некоторую запись и все подчиненные ей записи.
- **ИЗВЛЕЧЬ**:
  - извлечь корневую запись по ключевому значению, допускается также последовательный просмотр корневых записей
  - извлечь следующую запись (следующая запись извлекается в порядке левостороннего обхода дерева)

- В операции ИЗВЛЕЧЬ допускается задание условий выборки (например, извлечь сотрудников с окладом более 10 тысяч руб.)
- Как видим, все операции изменения применяются только к одной "текущей" записи (которая предварительно извлечена из базы данных). Такой подход к манипулированию данных получил название "**навигационного**".

Иерархическая модель данных

Ограничения целостности.

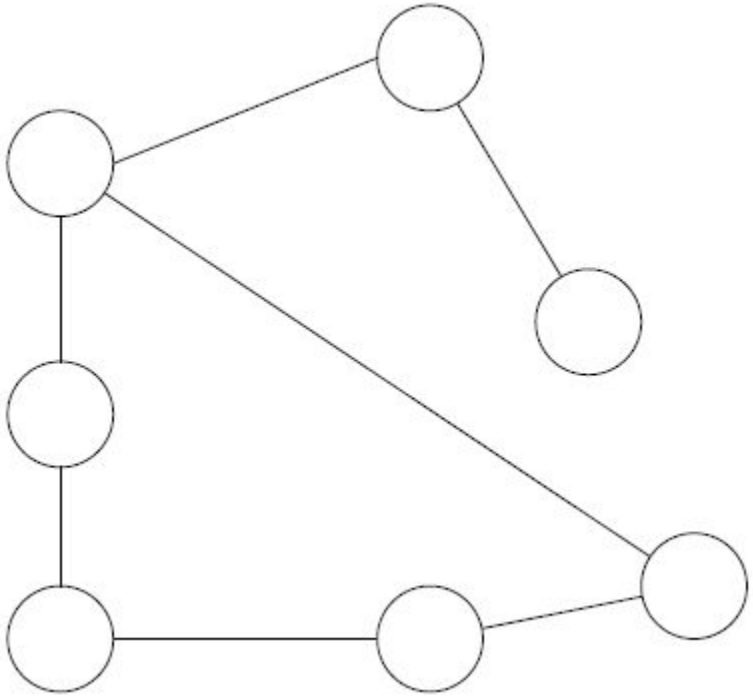
- Поддерживается только целостность связей между владельцами и членами группового отношения (никакой потомок не может существовать без предка). Как уже отмечалось, не обеспечивается автоматическое поддержание соответствия парных записей, входящих в разные иерархии.

Сетевая модель данных

Структура данных.

- Сетевая модель данных определяется в тех же терминах, что и иерархическая. Она состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид **1:N**.
- Основное различие этих моделей состоит в том, что в сетевой модели запись может быть членом *более чем одного* группового отношения. Согласно этой модели каждое групповое отношение именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При этом имеется следующее ограничение: экземпляр записи не может быть членом двух экземпляров групповых отношений одного типа (т.е. сотрудник, например, не может работать в двух отделах).

Сетевая модель данных



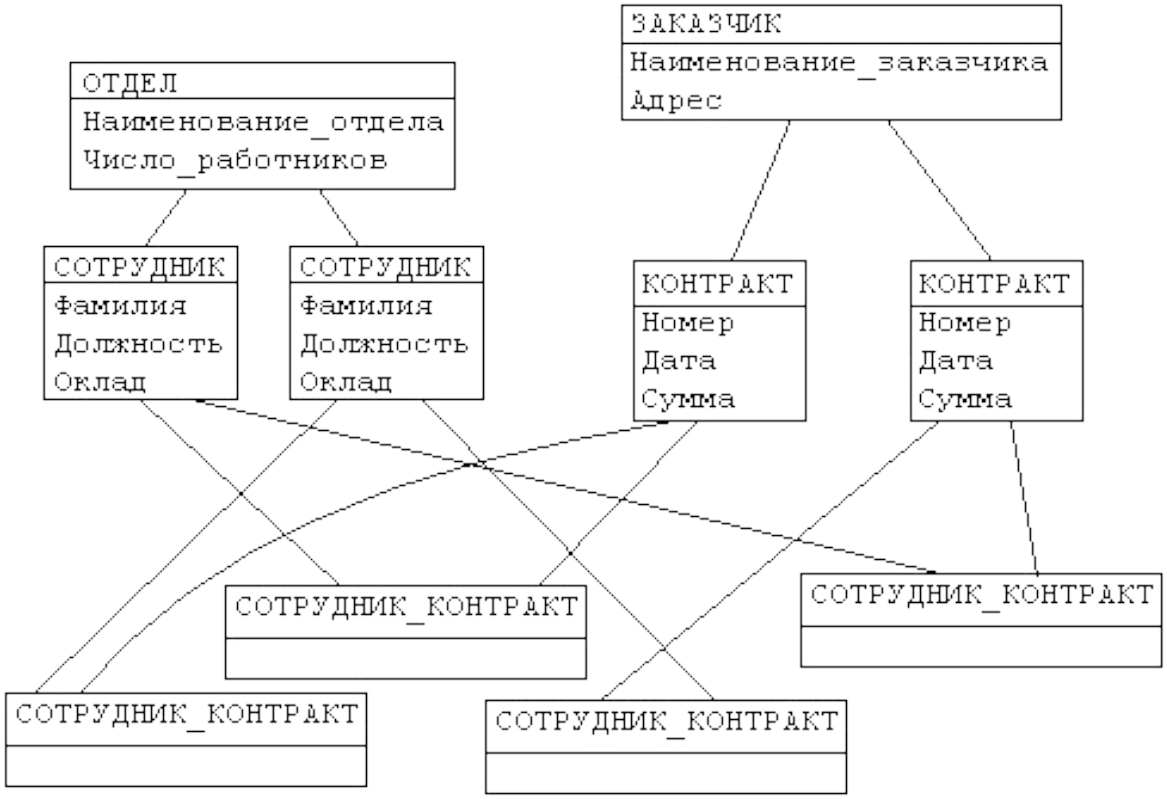
Сетевая модель данных

Сетевая модель данных

- Иерархическая структура преобразовывается в сетевую следующим образом (см. рис.):
- деревья (a) и (b), показанные в предыдущем примере, заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два групповых отношения;
- для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК, см. рис. (Отметим, что в этой записи может храниться и полезная информация, например, доля данного сотрудника в общем вознаграждении по данному контракту.)

Сетевая модель данных

- **Пример:**



Сетевая модель данных

- Каждый экземпляр группового отношения характеризуется следующими признаками:
- **способ упорядочения подчиненных записей**:
  - произвольный,
  - хронологический /очередь/,
  - обратный хронологический /стек/,
  - сортированный.
- Если запись объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой способ упорядочивания.

Сетевая модель данных

- **режим включения подчиненных записей**:
  - автоматический - невозможно занести в БД запись без того, чтобы она была сразу же закреплена за неким владельцем;
  - ручной - позволяет запомнить в БД подчиненную запись и не включать ее немедленно в экземпляр группового отношения. Эта операция позже инициируется пользователем).

- **режим исключения** Принято выделять три класса членства подчиненных записей в групповых отношениях:
  - *Фиксированное.* Подчиненная запись жестко связана с записью владельцем и ее можно исключить из группового отношения только удалив. При удалении записи-владельца все подчиненные записи автоматически тоже удаляются. В рассмотренном выше примере фиксированное членство предполагает групповое отношение "ЗАКЛЮЧАЕТ" между записями "КОНТРАКТ" и "ЗАКАЗЧИК", поскольку контракт не может существовать без заказчика.

Сетевая модель данных

- *Обязательное.* Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для удаления записи-владельца необходимо, чтобы она не имела подчиненных записей с обязательным членством. Таким отношением связаны записи "СОТРУДНИК" и "ОТДЕЛ". Если отдел расформировывается, все его сотрудники должны быть либо переведены в другие отделы, либо уволены.
- *Необязательное.* Можно исключить запись из группового отношения, но сохранить ее в базе данных не прикрепляя к другому владельцу. При удалении записи-владельца ее подчиненные записи - необязательные члены сохраняются в базе, не участвуя более в групповом отношении такого типа. Примером такого группового отношения может служить "ВЫПОЛНЯЕТ" между "СОТРУДНИКИ" и "КОНТРАКТ", поскольку в организации могут существовать работники, чья деятельность не связана с выполнением каких-либо договорных обязательств перед заказчиками.

Сетевая модель данных

Операции над данными.

- **ДОБАВИТЬ** - внести запись в БД и, в зависимости от режима включения, либо включить ее в групповое отношение, где она объявлена подчиненной, либо не включать ни в какое групповое отношение.
- **ВКЛЮЧИТЬ В ГРУППОВОЕ ОТНОШЕНИЕ** - связать существующую подчиненную запись с записью-владельцем.
- **ПЕРЕКЛЮЧИТЬ** - связать существующую подчиненную запись с другой записью-владельцем в том же групповом отношении.
- **ОБНОВИТЬ** - изменить значение элементов предварительно извлеченной записи.
- **ИЗВЛЕЧЬ** - извлечь записи последовательно по значению ключа, а также используя групповые отношения - от владельца можно перейти к записям - членам, а от подчиненной записи к владельцу набора.

Сетевая модель данных

Операции над данными.

- **УДАЛИТЬ** - убрать из БД запись. Если эта запись является владельцем группового отношения, то анализируется класс членства подчиненных записей. Обязательные члены должны быть предварительно исключены из группового отношения, фиксированные удалены вместе с владельцем, необязательные останутся в БД.
  ИСКЛЮЧИТЬ ИЗ ГРУППОВОГО ОТНОШЕНИЯ - разорвать связь между записью-владельцем и записью-членом.

Сетевая модель данных

Ограничения целостности.

- Как и в иерархической модели обеспечивается только поддержание целостности по ссылкам (владелец отношения - член отношения).

Реляционная модель данных

- Создателем реляционной модели считается математик Эдгар Фрэнк Кодд (Edgar Frank Codd, 1923 - 2003). Однако работа талантливого ученого начиналась не с чистого листа. Научный базис теории отношений, положенной Коддом в основу реляционной модели, закладывался еще на рубеже XIX и XX веков. Авторами основ будущей реляционной теории являются два исследователя: Чарльз Содерс Пирс (1839 -1914) и Эрнст Шредер (1841-1902). Несмотря на столь глубокие корни реляционной теории, первая реляционная база данных появилась на свет спустя семьдесят лет. Датой рождения реляционной БД можно считать июнь 1970 года. Именно тогда Кодд (на тот момент времени сотрудник одной из лабораторий корпорации IBM) опубликовал свою знаменитую статью "Реляционная модель данных для больших совместно используемых банков данных" (Codd E. F., A Relational Model of Data for Large Shared Data Banks. CACM 13: 6), в которой впервые прозвучал столь популярный сегодня термин "реляционная модель".

Реляционная модель данных

- Первопричиной возникновения нового по тем временам подхода к проектированию баз данных послужили существенные ограничения предыдущих моделей.
- Ни сетевая, ни иерархическая модели не были способны просто и доступно описывать подлежащие учету данные. Кодд сумел объединить, на первый взгляд, несовместимые вещи — с одной стороны, реляционная модель опиралась на математические выкладки, а с другой стороны, была понятна рядовому пользователю, состоящему в конфронтации даже с таблицей умножения.

Реляционная модель данных

- Модель данных не представляет существенного интереса для обычного пользователя, но без нее не обойтись разработчику БД. Ведь это не что иное, как чертеж будущей БД. Ни один строитель не начнет строить дом без чертежа, так и ни один программист не начнет создание физической БД без ее подробной схемы. Фундамент реляционной модели построен на понятии сущности и связей между сущностями.
- Реляционной модели данных посвящено много фундаментальных трудов, в которых подробно изложены все ключевые аспекты.
- Для лучшего понимания реляционной модели рекомендуется хотя бы в обзорном порядке ознакомиться с работами ведущих специалистов в этой области.
- Далее мы попытаемся сформулировать тот минимум информации, без которого понять реляционную модель будет просто невозможно.

Реляционная модель данных

- Любая область знаний, будь то математика, радиоэлектроника, физика, химия или информатика по мере своего развития формирует свой набор терминов и определений. Без этого никак не обойтись, ведь в своих ученых беседах специалисты оперируют понятиями, не имеющими аналогов в окружающем мире. Зачастую размерность специфичных понятий разрастается до такой степени, что существенная часть времени отводимого на изучение интересующего нас предмета затрачивается только на формирование словарного запаса.
- С точки зрения сложности семантики, системы управления базами данных (в общем) и реляционная модель (в частности) не являются исключением из правил.
- Эта сравнительно новая область знаний практически мгновенно успела обзавестись специфичным набором слов. Многие термины перекочевали в БД из словарного запаса программистов, часть определений позаимствована из теории множеств, в тезаурусе разработчиков моделей данных даже нашлось место для понятий, обычно встречающихся в справочниках по философии.

Реляционная модель данных

Структура данных

- В реляционной модели достигается гораздо более высокий уровень абстракции данных, чем в иерархической или сетевой. В упомянутой статье Е.Ф.Кодда утверждается, что "*реляционная модель предоставляет средства описания данных на основе только их естественной структуры, т.е. без потребности введения какой-либо дополнительной структуры для целей машинного представления*". Другими словами, представление данных не зависит от способа их физической организации. Это обеспечивается за счет использования математической теории отношений (само название "реляционная" происходит от английского relation - "отношение").

Реляционная модель данных

- Перейдем к рассмотрению структурной части реляционной модели данных. Прежде всего необходимо дать несколько определений.

- **Определения:**

- **Декартово произведение**: Для заданных конечных множеств $D_1,D_2,\ldots,D_n$ (не обязательно различных) декартовым произведением $D_1*D_2*\ldots*D_n$ называется множество произведений вида: $d_1*d_2*\ldots*d_n$, где $d_1 \in D_1$, $d_2 \in D_2,\ldots$, $d_n \in D_n$

- Пример: *если даны два множества A (a1,a2,a3) и B (b1,b2), их декартово произведение будет иметь вид C=A*B (a1*b1, a2*b1, a3*b1, a1*b2, a2*b2, a3*b2)*

Реляционная модель данных

- **Отношение:** Отношением *R*, определенным на множествах $D_1,D_2,\ldots,D_n$ называется подмножество декартова произведения $D_1*D_2*\ldots*D_n$. При этом:
  - множества $D_1,D_2,\ldots,D_n$ называются **доменами** отношения;
  - элементы декартова произведения $d_1*d_2*\ldots*d_n$ называются **кортежами**;
  - число *n* определяет **степень отношения** (*n=1* - унарное, *n=2* - бинарное, ..., *n*-арное);
  - количество кортежей называется **мощностью отношения**.

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

Реляционная модель данных

- Отношения удобно представлять в виде таблиц. На рисунке представлена таблица (отношение степени 5), содержащая некоторые сведения о работниках гипотетического предприятия.

Реляционная модель данных

- Строки таблицы соответствуют **кортежам**. Каждая строка фактически представляет собой описание одного объекта реального мира (в данном случае работника), характеристики которого содержатся в столбцах. Можно провести аналогию между элементами реляционной модели данных и элементами модели "сущность-связь". Реляционные отношения соответствуют наборам сущностей, а кортежи - сущностям. Поэтому, также как и в модели "сущность-связь" столбцы в таблице, представляющей реляционное отношение, называют **атрибутами**.
- Каждый атрибут определен на **домене**, поэтому домен можно рассматривать как множество допустимых значений данного атрибута.

Реляционная модель данных

- Несколько атрибутов одного отношения и даже атрибуты разных отношений могут быть определены на одном и том же домене. В примере, показанном на рисунке атрибуты "Оклад" и "Премия" определены на домене "Деньги". Поэтому, понятие домена имеет семантическую нагрузку: данные можно считать сравнимыми только тогда, когда они относятся к одному домену. Таким образом, в рассматриваемом нами примере сравнение атрибутов "Табельный номер" и "Оклад" является семантически некорректным, хотя они и содержат данные одного типа.
- Именованное множество пар "имя атрибута - имя домена" называется **схемой отношения**. Мощность этого множества - называют **степенью** или *"арностью"* отношения. Набор именованных схем отношений представляет из себя **схему базы данных**.

Реляционная модель данных

- Атрибут, значение которого однозначно идентифицирует кортежи, называется **ключевым** (или просто **ключом**). В нашем случае ключом является атрибут "Табельный номер", поскольку его значение уникально для каждого работника предприятия. Если кортежи идентифицируются только сцеплением значений нескольких атрибутов, то говорят, что отношение имеет составной ключ.
- Отношение может содержать несколько ключей. Всегда один из ключей объявляется **первичным**, его значения не могут обновляться. Все остальные ключи отношения называются *возможными (потенциальными) ключами*.

Реляционная модель данных

- Хотя любое отношение можно изобразить в виде таблицы, нужно четко понимать, что *отношения не являются таблицами*. Это близкие, но не совпадающие понятия.
- Термины, которыми оперирует реляционная модель данных, имеют соответствующие "табличные" синонимы:

Реляционный термин	**Соответствующий "табличный" термин**
База данных	Набор таблиц
Схема базы данных	Набор заголовков таблиц
Отношение	Таблица
Заголовок отношения	Заголовок таблицы
Тело отношения	Тело таблицы
Атрибут отношения	Наименование столбца таблицы
Кортеж отношения	Строка таблицы
Степень (-арность) отношения	Количество столбцов таблицы
Мощность отношения	Количество строк таблицы
Домены и типы данных	Типы данные в ячейках таблицы

Реляционная модель данных

- Приведем пример таблицы произвольной структуры, не являющейся отношением:

Отдел	**Сотрудники**	**Дети сотрудников (интересы)**		
Цех	Иванов И.И.	Маша	ЛЕГО	
		Петя	Книги	Видео
		Саша	Компьютеры	
		Дима	Спорт	
	Петров П.П.	Артур	Ничем не интересуется	
	Сидоров С. С.	Сергей	Компьютеры Книги	
		Валерий	Книги	
		Станислав	Видео	
Бухгалтерия	…	…		

Таблица произвольной формы

Реляционная модель данных

Свойства отношений

Свойства отношений непосредственно следуют из приведенного выше определения отношения. В этих свойствах в основном и состоят различия между отношениями и таблицами.

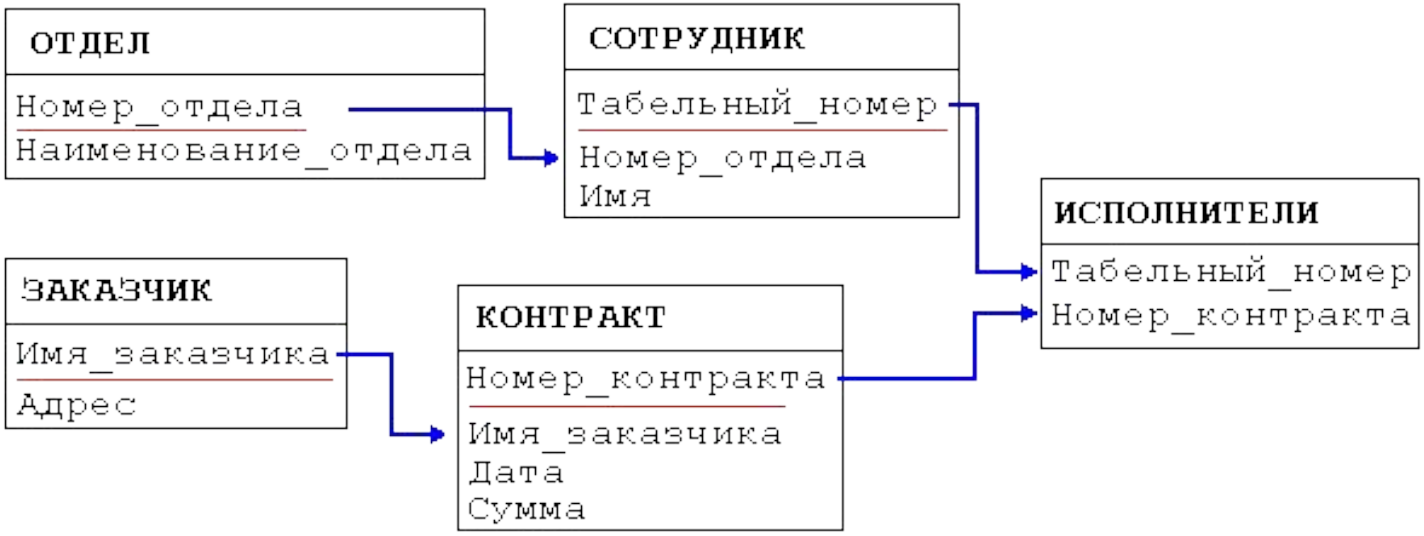
1. **Отсутствие кортежей-дубликатов**. *Из этого свойства вытекает наличие у каждого кортежа первичного ключа. Для каждого отношения, по крайней мере, полный набор его атрибутов является первичным ключом. Однако, при определении первичного ключа должно соблюдаться требование "минимальности", т.е. в него не должны входить те атрибуты, которые можно отбросить без ущерба для основного свойства первичного ключа - однозначно определять кортеж.*
2. **Отсутствие упорядоченности кортежей**.
3. **Отсутствие упорядоченности атрибутов**. Для ссылки на значение атрибута всегда используется имя атрибута.
4. **Атомарность значений атрибутов**, т.е. среди значений домена не могут содержаться множества значений (отношения).

Реляционная модель данных

- Замечание. Из свойств отношения следует, что *не каждая* таблица может задавать отношение. Для того, чтобы некоторая таблица задавала отношение, необходимо, чтобы таблица имела простую структуру (содержала бы только строки и столбцы, причем, в каждой строке было бы одинаковое количество полей), в таблице не должно быть одинаковых строк, любой столбец таблицы должен содержать данные только одного типа, все используемые типы данных должны быть простыми.

Реляционная модель данных

- 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. Рассмотренный в предыдущих разделах пример базы данных, содержащей сведения о подразделениях предприятия и работающих в них сотрудниках, применительно к реляционной модели будет иметь вид:



База данных о подразделениях и сотрудниках предприятия

Реляционная модель данных

- Например, связь между отношениями ОТДЕЛ и СОТРУДНИК создается путем копирования первичного ключа *"Номер_отдела"* из первого отношения во второе. Таким образом:
- для того, чтобы получить список работников данного подразделения, необходимо
  - из таблицы ОТДЕЛ установить значение атрибута *"Номер_отдела"*, соответствующее данному "Наименованию_отдела"
  - выбрать из таблицы СОТРУДНИК все записи, значение атрибута *"Номер_отдела"* которых равно полученному на предыдущем шаге.
- для того, чтобы узнать в каком отделе работает сотрудник, нужно выполнить обратную операцию:
  - определяем *"Номер_отдела"* из таблицы СОТРУДНИК
  - по полученному значению находим запись в таблице ОТДЕЛ.
- Атрибуты, представляющие собой копии ключей других отношений, называются **внешними ключами**.

Реляционная модель данных

Целостность реляционных данных

- Во второй части реляционной модели данных определяются два ограничения, которые должны выполняться в *любой* реляционной базе данных. Это:

  - Целостность сущностей.

  - Целостность внешних ключей.

- Прежде, чем говорить о целостности сущностей, опишем использование null-значений в реляционных базах данных.

Реляционная модель данных

- **Null-значения**
- Основное назначение баз данных состоит в том, чтобы хранить и предоставлять информацию о реальном мире. Для представления этой информации в базе данных используются привычные для программистов типы данных - строковые, численные, логические и т.п. Однако в реальном мире часто встречается ситуация, когда данные неизвестны или не полны. Например, место жительства или дата рождения человека могут быть неизвестны. Если вместо неизвестного адреса уместно было бы вводить пустую строку, то что вводить вместо неизвестной даты? Ответ - пустую дату - не вполне удовлетворителен, т.к. простейший запрос "выдать список людей в порядке возрастания дат рождения" даст заведомо неправильных ответ.
- Для того чтобы обойти проблему неполных или неизвестных данных, в базах данных могут использоваться типы данных, пополненные так называемым ***null-значением***. Null-значение - это, собственно, не значение, а некий маркер, показывающий, что значение неизвестно.

Реляционная модель данных

- Таким образом, в ситуации, когда возможно появление неизвестных или неполных данных, разработчик имеет на выбор два варианта.
- Первый вариант состоит в том, чтобы ограничиться использованием обычных типов данных и не использовать null-значения, а вместо неизвестных данных вводить либо нулевые значения, либо значения специального вида - например, договориться, что строка "АДРЕС НЕИЗВЕСТЕН" и есть те данные, которые нужно вводить вместо неизвестного адреса. В любом случае на пользователя (или на разработчика) ложится ответственность на правильную трактовку таких данных. В частности, может потребоваться написание специального программного кода, который в нужных случаях "вылавливал" бы такие данные. Проблемы, возникающие при этом очевидны - не все данные становятся равноправны, требуется дополнительный программный код, "отслеживающий" эту неравноправность, в результате чего усложняется разработка и сопровождение приложений.

Реляционная модель данных

- Второй вариант состоит в использовании null-значений вместо неизвестных данных. За кажущейся естественностью такого подхода скрываются менее очевидные и более глубокие проблемы. Наиболее бросающейся в глаза проблемой является необходимость использования трехзначной логики при оперировании с данными, которые могут содержать null-значения. В этом случае при неаккуратном формулировании запросов, даже самые естественные запросы могут давать неправильные ответы. Есть более фундаментальные проблемы, связанные с теоретическим обоснованием корректности введения null-значений, например, непонятно вообще, входят ли null-значения в домены или нет.
- Подробное обсуждение проблем использования null-значений выходит за пределы лекций. Можно только сказать о том, что этот вопрос в теории реляционных баз данных окончательно не решен. Основоположник реляционного подхода Кодд считал null-значения неотъемлемой частью реляционной модели. К.Дейт, один из крупнейших теоретиков реляционной модели выступает категорически против null-значений.

Реляционная модель данных

- **Трехзначная логика (3VL)**
- Т.к. null-значение обозначает на самом деле тот факт, что значение неизвестно, то любые алгебраические операции (сложение, умножение, конкатенация строк и т.д.) должны давать также неизвестное значение, т.е. null. Действительно, если, например, вес детали неизвестен, то неизвестно также, сколько весят 10 таких деталей.
- При сравнении выражений, содержащих null-значения, результат также может быть неизвестен, например, значение истинности для выражения есть null, если один или оба аргумента есть null. Таким образом, определение истинности логических выражений базируется на ***трехзначной логике*** (***three-valued logic***, ***3VL***), в которой кроме значений T – (True) ИСТИНА и F – (False) ЛОЖЬ, введено значение U – (Unknown) НЕИЗВЕСТНО. Логическое значение U - это то же самое, что и null-значение.

Реляционная модель данных

- Трехзначная логика базируется на следующих таблицах истинности:

AND	F	T	U
F	F	F	F
T	F	T	U
U	F	U	U

Таблица истинности AND

OR	F	T	U
F	F	T	U
T	T	T	T
U	U	T	U

Таблица истинности OR

NOT	
F	T
T	F
U	U

Таблица истинности NOT

Реляционная модель данных

- Имеется несколько парадоксальных следствий применения трехзначной логики.
- Парадокс 1. Null-значение не равно самому себе. Действительно, выражение null = null дает значение не ИСТИНА, а НЕИЗВЕСТНО. Значит выражение *x=x* не обязательно ИСТИНА!
- Парадокс 2. Неверно также, что null-значение не равно самому себе! Действительно, выражение null ≠ null также принимает значение не ИСТИНА, а НЕИЗВЕСТНО! Значит также, что и выражение *x ≠ x* тоже не обязательно ЛОЖЬ!
- Парадокс 3. *a or not (a)* не обязательно ИСТИНА. Значит, в трехзначной логике не работает принцип исключенного третьего (любое высказывание либо истинно, либо ложно).
- Таких парадоксов можно построить сколько угодно. Конечно, это на самом деле не парадоксы, а просто следствия из аксиом трехзначной логики.

Реляционная модель данных

- **Потенциальные ключи**
- По определению, тело отношения есть *множество* кортежей, поэтому отношения не могут содержать одинаковые кортежи. Это значит, что каждый кортеж должен обладать *свойством уникальности*. На самом деле, свойством уникальности в пределах отношения могут обладать отдельные атрибуты кортежей или группы атрибутов. Такие уникальные атрибуты удобно использовать для идентификации кортежей.
- *Определение 1*. Пусть дано отношение *R*. Подмножество атрибутов *K* отношения *R* будем называть ***потенциальным ключом***, если *K* обладает следующими свойствами:

  - Свойством уникальности - в отношении не может быть двух различных кортежей, с одинаковым значением *K.*

  - Свойством неизбыточности - никакое подмножество в *K* не обладает свойством уникальности.

Реляционная модель данных

- Любое отношение имеет по крайней мере один потенциальный ключ. Действительно, если никакой атрибут или группа атрибутов не являются потенциальным ключом, то, в силу уникальности кортежей, все атрибуты вместе образуют потенциальный ключ.
- Потенциальный ключ, состоящий из одного атрибута, называется ***простым***. Потенциальный ключ, состоящий из нескольких атрибутов, называется ***составным***.
- Отношение может иметь несколько потенциальных ключей. Традиционно, один из потенциальных ключей объявляется ***первичным***, а остальные - ***альтернативными***. Различия между первичным и альтернативными ключами могут быть важны в конкретной реализации реляционной СУБД, но с точки зрения реляционной модели данных, нет оснований выделять таким образом один из потенциальных ключей.

Реляционная модель данных

- Замечание. Понятие потенциального ключа является *семантическим* понятием и отражает некоторый смысл (трактовку) понятий из конкретной предметной области. Для того чтобы проиллюстрировать этот факт рассмотрим следующее отношение "Сотрудники":

Табельный номер	**Фамилия**	**Зарплата**
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

Реляционная модель данных

- При первом взгляде на таблицу, изображающую это отношение, может показаться, что в таблице имеется три потенциальных ключа - в каждой колонке таблицы содержатся уникальные данные. Однако среди сотрудников могут быть однофамильцы и сотрудники с одинаковой зарплатой. Табельный же номер по сути свой уникален для каждого сотрудника. Какие же соображения привели нас к пониманию того, что в данном отношении только один потенциальный ключ - "Табельный номер"? Именно *понимание смысла данных*, содержащихся в отношении.
- Попробуем представить это отношение в другом виде, изменив наименования атрибутов:

A	**B**	**C**
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

Реляционная модель данных

- Предъявим кому-нибудь эту таблицу и не сообщим смысл наименований атрибутов. Очевидно, что невозможно судить, не понимая смысла данных, может или не может в этом отношении появиться, например, кортеж (1, Петров, 3000). Если бы, кстати, такой кортеж появился (что, на первый взгляд, вполне возможно, т. к. не нарушается уникальность кортежей), то мы точно смогли бы сказать, что *не является* альтернативным ключом - ни один из атрибутов по отдельности. Но мы не сможем сказать, что же *является* первичным ключом.
- Замечание. Потенциальные ключи служат ***средством идентификации*** объектов предметной области, данные о которых хранятся в отношении. Объекты предметной области должны быть различимы.
- Замечание. Потенциальные ключи служат единственным ***средством адресации на уровне кортежей*** в отношении. Точно указать какой-нибудь кортеж можно только зная значение его потенциального ключа.

Реляционная модель данных

- **Целостность сущностей**
- Т.к. потенциальные ключи фактически служат идентификаторами объектов предметной области (т.е. предназначены для *различения* объектов), то значения этих идентификаторов не могут содержать неизвестные значения. Действительно, если бы идентификаторы могли содержать null-значения, то мы не могли бы дать ответ "да" или "нет" на вопрос, совпадают или нет два идентификатора.
- Это определяет следующее *правило целостности сущностей*:
- ***Правило целостности сущностей***. Атрибуты, входящие в состав некоторого потенциального ключа не могут принимать null-значений.

Реляционная модель данных

- **Внешние ключи**
- Различные объекты предметной области, информация о которых хранится в базе данных, всегда взаимосвязаны друг с другом. Например, накладная на поставку товара *содержит* список товаров с количествами и ценами, сотрудник предприятия *имеет* детей, *числится* в подразделении и т.д. Термины "содержит", "имеет", "числится" отражают взаимосвязи между понятиями "накладная" и "список товаров", "сотрудник" и "дети", "сотрудник" и "подразделение". Такие взаимосвязи отражаются в реляционных базах данных при помощи ***внешних ключей***, связывающих несколько отношений.

Реляционная модель данных

- Рассмотрим пример с поставщиками и поставками деталей. Предположим, что нам требуется хранить информацию о наименовании поставщиков, наименовании и количестве поставляемых ими деталей, причем каждый поставщик может поставлять несколько деталей и каждая деталь может поставляться несколькими поставщиками. Можно предложить хранить данные в следующем отношении:

Номер поставщика	**Наименование поставщика**	***Номер детали***	**Наименование детали**	**Поставляемое количество**
1	Иванов	*1*	Болт	100
1	Иванов	*2*	Гайка	200
1	Иванов	*3*	Винт	300
2	Петров	*1*	Болт	150
2	Петров	*2*	Гайка	250
3	Сидоров	*3*	Винт	1000

Реляционная модель данных

- Потенциальным ключом этого отношения может выступать пара атрибутов {"Номер поставщика", "Номер детали"} - в таблице они выделены курсивом.
- Приведенный способ хранения данных обладает рядом недостатков.
- Что произойдет, если изменилось наименование поставщика? Т.к. наименование поставщика повторяется во многих кортежах отношения, то это наименование нужно *одновременно* изменить *во всех* кортежах, где оно встречается, иначе данные станут противоречивыми. То же самое с наименованиями деталей. Значит, данные хранятся в нашем отношении с большой избыточностью.

Реляционная модель данных

- Далее, как отразить факт, что некоторый поставщик, например Петров, временно прекратил поставки деталей? Если мы удалим все кортежи, в которых хранится информация о поставках этого поставщика, то мы *потеряем данные* о самом Петрове как потенциальном поставщике. Выйти из этого положения, оставив в отношении кортеж типа (2, Петров, NULL, NULL, NULL) мы не можем, т.к. атрибут "Номер детали" входит в состав потенциального ключа и не может содержать null-значений. То же самое произойдет, если некоторая деталь временно не поставляется никаким поставщиком. Получается, что мы не можем хранить информацию о том, что есть некий поставщик, если он не поставляет хотя бы одну деталь, и не можем хранить информацию о том, что есть некоторая деталь, если она никем не поставляется.
- Подобные проблемы возникают потому, что мы смешали в одном отношении различные объекты предметной области - и данные о поставщиках, и данные о деталях, и данные о поставках деталей. Говорят, что это отношение *плохо нормализовано.*

Реляционная модель данных

- О том, как правильно нормализовать отношения, будет сказано в следующих главах, сейчас же предложим разнести данные по трем отношениям - "Поставщики", "Детали", "Поставки". Для нас важно выяснить, каким образом данные, хранящиеся в этих отношениях взаимосвязаны друг с другом. Эта связь определяется семантикой предметной области и описывается фразами: "Поставщики *выполняют* Поставки", "Детали *поставляются через* Поставки". Эти две взаимосвязи косвенно определяют новую взаимосвязь между "Поставщиками" и "Деталями": "Детали *поставляются* Поставщиками".
- Эти фразы отражают различные типы взаимосвязей. Чтобы более точно отразить предметную область, можно иначе переформулировать фразы: "*Один* Поставщик может выполнять *несколько* Поставок", "*Одна* Деталь может поставляться *несколькими* Поставками". Это пример взаимосвязи типа "***один-ко-многим***".

Реляционная модель данных

- Взаимосвязь между "Поставщиками" и "Деталями" можно переформулировать так: "*Несколько* Деталей может поставляться *несколькими* Поставщиками". Это пример взаимосвязи типа "***много-ко-многим***".
- В реляционных базах данных основными являются взаимосвязи типа "один-ко-многим". Взаимосвязи типа "много-ко-многим" реализуются использованием нескольких взаимосвязей типа "один-ко-многим". Отношение, входящее в связь со стороны "один" (например, "Поставщики"), называют ***родительским отношением***. Отношение, входящее в связь со стороны "много" (например, "Поставки"), называется ***дочернем отношением***.
- Механизм реализации взаимосвязи "один-ко-многим" состоит в том, что в дочернее отношение добавляются атрибуты, являющиеся ссылками на ключевые атрибуты родительского отношения. Эти атрибуты и являются внешними ключами, определяющими, с какими кортежами родительского отношения связаны кортежи дочернего отношения. Такие атрибуты еще называют ***мигрирующими*** из родительского отношения.

Реляционная модель данных

- Таким образом, наш пример с поставщиками и поставляемыми деталями должен выглядеть следующим образом:

Номер поставщика	**Наименование поставщика**
1	Иванов
2	Петров
3	Сидоров

Отношение "Поставщики"

Номер детали	**Наименование детали**
1	Болт
2	Гайка
3	Винт

Отношение "Детали"

Номер поставщика	***Номер детали***	**Поставляемое количество**
1	*1*	100
1	*2*	200
1	*3*	300
2	*1*	150
2	*2*	250
3	*3*	1000

Отношение "Поставки"

Реляционная модель данных

- Дадим точное определение.
- ***Определение 2.*** Пусть дано отношение *R*. Подмножество атрибутов *FK* отношения *R* будем называть ***внешним ключом***, если:

  1. Существует отношение *S*(*R* и *S* не обязательно различны) с потенциальным ключом *K*.

  2. Каждое значение *FK* в отношении *R* всегда совпадает со значением *K* для некоторого кортежа из *S*, либо является null-значением.
- Отношение *S* называется ***родительским отношением***, отношение *R* называется ***дочерним отношением***.

- Замечание. Внешний ключ, также как и потенциальный, может быть простым и составным.
- Замечание. Внешний ключ должен быть определен на тех же доменах, что и соответствующий первичный ключ родительского отношения.

Реляционная модель данных

- Замечание. Внешний ключ, как правило, *не обладает свойством уникальности*. Так и должно быть, т.к. в дочернем отношении может быть несколько кортежей, ссылающихся на один и тот же кортеж родительского отношения. Это, собственно, и дает тип отношения "один-ко-многим".

- Замечание. Если внешний ключ все-таки обладает свойством уникальности, то связь между отношениями имеет тип "***один-к-одному***". Чаще всего такие отношения объединяются в одно отношение, хотя это и не обязательно.

- Замечание. Хотя каждое значение внешнего ключа обязано совпадать со значениями потенциального ключа в некотором кортеже родительского отношения, то обратное, вообще говоря, неверно. Например, могут существовать поставщики, не поставляющие никаких деталей.

Реляционная модель данных

- Замечание. Для внешнего ключа не требуется, чтобы он был компонентом некоторого потенциального ключа (как получилось в примере с поставщиками и деталями). (Идентифицирующая и неидентифицирующая связи из IDEF1X).

- Замечание. Null-значения для атрибутов внешнего ключа допустимы только в том случае, когда атрибуты внешнего ключа не входят в состав никакого потенциального ключа

Реляционная модель данных

- **Целостность внешних ключей**
- Т.к. внешние ключи фактически служат ссылками на кортежи в другом (или в том же самом) отношении, то эти ссылки не должны указывать на несуществующие объекты. Это определяет следующее *правило целостности внешних ключей*:
- ***Правило целостности внешних ключей***. Внешние ключи не должны быть несогласованными, т.е. для каждого значения внешнего ключа должно существовать соответствующее значение первичного ключа в родительском отношении.

Реляционная модель данных

- **Операции, могущие нарушить ссылочную целостность**

- Ссылочная целостность может нарушиться в результате операций, изменяющих состояние базы данных. Таких операций три - *вставка*, *обновление* и *удаление* кортежей в отношениях. Т.к. в определении ссылочной целостности участвуют два отношения - родительское и дочернее, а в каждом из них возможны три операции - вставка, обновление, удаление, то нужно рассмотреть шесть различных вариантов.

Реляционная модель данных

- **Операции, могущие нарушить ссылочную целостность**
- **Для родительского отношения**
- *Вставка кортежа в родительском отношении*. При вставке кортежа в родительское отношение возникает новое значение потенциального ключа. Т.к. допустимо существование кортежей в родительском отношении, на которые нет ссылок из дочернего отношения, то вставка кортежей в родительское отношение *не нарушает ссылочной целостности*.
- *Обновление кортежа в родительском отношении*. При обновлении кортежа в родительском отношении может измениться значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на обновляемый кортеж, то значения их внешних ключей станут некорректными. Обновление кортежа в родительском отношении *может привести к нарушению ссылочной целостности*, если это обновление затрагивает значение потенциального ключа.

Реляционная модель данных

- **Операции, могущие нарушить ссылочную целостность**
- **Для родительского отношения**

- *Удаление кортежа в родительском отношении*. При удалении кортежа в родительском отношении удаляется значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на удаляемый кортеж, то значения их внешних ключей станут некорректными. Удаление кортежей в родительском отношении *может привести к нарушению ссылочной целостности*.

Реляционная модель данных

- **Операции, могущие нарушить ссылочную целостность**
- **Для дочернего отношения**

- *Вставка кортежа в дочернее отношение*. Нельзя вставить кортеж в дочернее отношение, если вставляемое значение внешнего ключа некорректно. Вставка кортежа в дочернее отношение *может привести к нарушению ссылочной целостности*.
- *Обновление кортежа в дочернем отношении*. При обновлении кортежа в дочернем отношении можно попытаться некорректно изменить значение внешнего ключа. Обновление кортежа в дочернем отношении *может привести к нарушению ссылочной целостности*.
- *Удаление кортежа в дочернем отношении*. При удалении кортежа в дочернем отношении *ссылочная целостность не нарушается*.

Реляционная модель данных

- **Операции, могущие нарушить ссылочную целостность**

- Таким образом, ссылочная целостность в принципе может быть нарушена при выполнении одной из четырех операций:
  - Обновление кортежа в родительском отношении.
  - Удаление кортежа в родительском отношении.
  - Вставка кортежа в дочернее отношение.
  - Обновление кортежа в дочернем отношении.

Реляционная модель данных

- **Стратегии поддержания ссылочной целостности**

- Существуют ***две основные стратегии поддержания ссылочной целостности***:

- ***RESTRICT (ОГРАНИЧИТЬ)***- не разрешать выполнение операции, приводящей к нарушению ссылочной целостности. Это самая простая стратегия, требующая только проверки, имеются ли кортежи в дочернем отношении, связанные с некоторым кортежем в родительском отношении.

Реляционная модель данных

- ***CASCADE (КАСКАДИРОВАТЬ)***- разрешить выполнение требуемой операции, но внести при этом необходимые поправки в других отношениях так, чтобы не допустить нарушения ссылочной целостности и сохранить все имеющиеся связи. Изменение начинается в родительском отношении и каскадно выполняется в дочернем отношении. В реализации этой стратегии имеется одна тонкость, заключающаяся в том, что дочернее отношение само может быть родительским для некоторого третьего отношения. При этом может дополнительно потребоваться выполнение какой-либо стратегии и для этой связи и т.д. Если при этом какая-либо из каскадных операций (любого уровня) не может быть выполнена, то необходимо отказаться от первоначальной операции и вернуть базу данных в исходное состояние. Это самая сложная стратегия, но она хороша тем, что при этом не нарушается связь между кортежами родительского и дочернего отношений.
- Эти стратегии являются стандартными и присутствуют во всех СУБД, в которых имеется поддержка ссылочной целостности.

Реляционная модель данных

- Можно рассмотреть ***дополнительные стратегии поддержания ссылочной целостности***:
- ***SET NULL (УСТАНОВИТЬ В NULL)*** - разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на null-значения. Эта стратегия имеет два недостатка. Во-первых, для нее требуется допустить использование null-значений. Во-вторых, кортежи дочернего отношения теряют всякую связь с кортежами родительского отношения. Установить, с каким кортежем родительского отношения были связаны измененные кортежи дочернего отношения, после выполнения операции уже нельзя.
- ***SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ)*** - разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию. Достоинство этой стратегии по сравнению с предыдущей в том, что она позволяет не пользоваться null-значениями. Недостатки заключаются в следующем:

Реляционная модель данных

- Во-первых, в родительском отношении должен быть некий кортеж, потенциальный ключ которого принят как значение по умолчанию для внешних ключей. В качестве такого "кортежа по умолчанию" обычно принимают специальный кортеж, заполненный нулевыми значениями (не null-значениями!). Этот кортеж *нельзя удалять* из родительского отношения, и в этом кортеже *нельзя изменять* значение потенциального ключа. Таким образом, не все кортежи родительского отношения становятся равнозначными, поэтому приходится прилагать дополнительные усилия для отслеживания этой неравнозначности. Это плата за отказ от использования null-значений. Во-вторых, как и в предыдущем случае, кортежи дочернего отношения теряют всякую связь с кортежами родительского отношения. Установить, с каким кортежем родительского отношения были связаны измененные кортежи дочернего отношения, после выполнения операции уже нельзя.

Реляционная модель данных

- В некоторых реализация СУБД рассматривается ***еще одна стратегия поддержания ссылочной целостности***:
- ***IGNORE (ИГНОРИРОВАТЬ)*** - выполнять операции, не обращая внимания на нарушения ссылочной целостности.
- Конечно, это не стратегия, а отказ от поддержки ссылочной целостности. В этом случае в дочернем отношении могут появляться некорректные значения внешних ключей, и вся ответственность за целостность базы данных ложится на пользователя.
- В дополнение к приведенным стратегиям пользователь может придумать свою уникальную стратегию поддержания ссылочной целостности.

Нормальные формы отношений

Этапы разработки базы данных

- Целью разработки любой базы данных является хранение и использование информации о какой-либо предметной области. Для реализации этой цели имеются следующие инструменты:

  - *Реляционная модель данных - удобный способ представления данных предметной области.*

  - Язык SQL - универсальный способ манипулирования такими данными.
- Однако очевидно, что для одной и той же предметной области реляционные отношения можно спроектировать множеством различных способов. Например, можно спроектировать несколько отношений с большим количеством атрибутов, или наоборот, разнести все атрибуты по большому числу мелких отношений. Как определить, по каким признакам нужно помещать атрибуты в те или иные отношения?

Нормальные формы отношений

- При разработке базы данных обычно выделяется несколько уровней моделирования, при помощи которых происходит переход от предметной области к конкретной реализации базы данных средствами конкретной СУБД. Можно выделить следующие уровни:

  - Сама предметная область;

  - Модель предметной области (Концептуальная модель);

  - Логическая модель данных;

  - Физическая модель данных;

  - Собственно база данных и приложения.

- Ясно, что решения, принятые на каждом этапе моделирования и разработки базы данных, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений *на ранних этапах моделирования*.

Нормальные формы отношений

Критерии оценки качества логической модели данных

- Опишем некоторые принципы построения *хороших логических моделей данных*. Хороших в том смысле, что решения, принятые в процессе логического проектирования приводили бы к хорошим физическим моделям и в конечном итоге к хорошей работе базы данных.
- Для того чтобы оценить качество принимаемых решений на уровне логической модели данных, необходимо сформулировать некоторые критерии качества в терминах физической модели и конкретной реализации и посмотреть, как различные решения, принятые в процессе *логического* моделирования, влияют на качество *физической* модели и на скорость работы базы данных.

Нормальные формы отношений

- Конечно, таких критериев может быть очень много и выбор их в достаточной степени произволен. Мы рассмотрим некоторые из таких критериев, которые являются безусловно важными с точки зрения получения качественной базы данных:

  - Адекватность базы данных предметной области;

  - Легкость разработки и сопровождения базы данных;

  - Скорость выполнения операций обновления данных (вставка, обновление, удаление кортежей);

  - Скорость выполнения операций выборки данны.

Нормальные формы отношений

Адекватность базы данных предметной области

- База данных должна адекватно отражать предметную область. Это означает, что должны выполняться следующие условия:

1. Состояние базы данных в каждый момент времени должно соответствовать состоянию предметной области.
2. Изменение состояния предметной области должно приводить к соответствующему изменению состояния базы данных
3. Ограничения предметной области, отраженные в модели предметной области, должны некоторым образом отражаться и учитываться базе данных.

Нормальные формы отношений

- **Легкость разработки и сопровождения базы данных**
- Практически любая база данных, за исключением совершенно элементарных, содержит некоторое количество программного кода в виде триггеров и хранимых процедур.
- ***Хранимые процедуры*** - это процедуры и функции, хранящиеся непосредственно в базе данных в откомпилированном виде и которые могут запускаться пользователями или приложениями, работающими с базой данных. Хранимые процедуры обычно пишутся либо на специальном процедурном расширении языка SQL (например, PL/SQL для ORACLE или Transact-SQL для MS SQL Server), или на некотором универсальном языке программирования, например, C++, с включением в код операторов SQL в соответствии со специальными правилами такого включения. Основное назначение хранимых процедур - реализация бизнес-процессов предметной области.

Нормальные формы отношений

- ***Триггеры*** - это хранимые процедуры, связанные с некоторыми событиями, происходящими во время работы базы данных. В качестве таких событий выступают операции вставки, обновления и удаления строк таблиц. Если в базе данных определен некоторый триггер, то он запускается *автоматически* всегда при возникновении события, с которым этот триггер связан. Очень важным является то, что пользователь не может обойти триггер. Триггер срабатывает независимо от того, кто из пользователей и каким способом инициировал событие, вызвавшее запуск триггера. Таким образом, основное назначение триггеров - автоматическая поддержка целостности базы данных. Триггеры могут быть как достаточно простыми, например, поддерживающими ссылочную целостность, так и довольно сложными, реализующими какие-либо сложные ограничения предметной области или сложные действия, которые должны произойти при наступлении некоторых событий.

Нормальные формы отношений

- Например, с операцией вставки нового товара в накладную может быть связан триггер, который выполняет следующие действия - проверяет, есть ли необходимое количество товара, при наличии товара добавляет его в накладную и уменьшает данные о наличии товара на складе, при отсутствии товара формирует заказ на поставку недостающего товара и тут же посылает заказ по электронной почте поставщику.
- Очевидно, что чем больше программного кода в виде триггеров и хранимых процедур содержит база данных, тем сложнее ее разработка и дальнейшее сопровождение.

Нормальные формы отношений

Скорость операций обновления данных (вставка, обновление, удаление)

- На уровне логического моделирования мы определяем реляционные отношения и атрибуты этих отношений. На этом уровне мы не можем определять какие-либо физические структуры хранения (индексы, хеширование и т.п.). Единственное, чем мы можем управлять - это распределением атрибутов по различным отношениям. Можно описать мало отношений с большим количеством атрибутов, или много отношений, каждое из которых содержит мало атрибутов. Таким образом, необходимо попытаться ответить на вопрос - влияет ли количество отношений и количество атрибутов в отношениях на скорость выполнения операций обновления данных. Такой вопрос, конечно, не является достаточно корректным, т.к. скорость выполнения операций с базой данных сильно зависит от физической реализации базы данных. Тем не менее, попытаемся *качественно* оценить это влияние при *одинаковых подходах к физическому моделированию*.

Нормальные формы отношений

Основными операциями, изменяющими состояние базы данных, являются *операции вставки, обновления и удаления записей*. В базах данных, требующих постоянных изменений (складской учет, системы продаж билетов и т.п.) производительность определяется скоростью выполнения большого количества небольших операций вставки, обновления и удаления.

- Рассмотрим *операцию вставки записи* в таблицу. Вставка записи производится в одну из свободных страниц памяти, выделенной для данной таблицы. СУБД постоянно хранит информацию о наличии и расположении свободных страниц. Если для таблицы не созданы индексы, то операция вставки выполняется фактически с одинаковой скоростью независимо от размера таблицы и от количества атрибутов в таблице. Если в таблице имеются индексы, то при выполнении операции вставки записи индексы должны быть перестроены. Таким образом, скорость выполнения операции вставки *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице.

Нормальные формы отношений

Рассмотрим *операции обновления и удаления записей из таблицы*. Прежде, чем обновить или удалить запись, ее необходимо найти. Если таблица не индексирована, то единственным способом поиска является последовательное сканирование таблицы в поиске нужной записи. В этом случае, скорость операций обновления и удаления существенно увеличивается с увеличением количества записей в таблице и не зависит от количества атрибутов. Но на самом деле неиндексированные таблицы практически никогда не используются. Для каждой таблицы обычно объявляется один или несколько индексов, соответствующий потенциальным ключам. При помощи этих индексов поиск записи производится очень быстро и практически не зависит от количества строк и атрибутов в таблице (хотя, конечно, некоторая зависимость имеется). Если для таблицы объявлено несколько индексов, то при выполнении операций обновления и удаления эти индексы должны быть перестроены, на что тратится дополнительное время. Таким образом, скорость выполнения операций обновления и удаления также *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице.

Нормальные формы отношений

- Можно предположить, что чем больше атрибутов имеет таблица, тем больше для нее будет объявлено индексов. Эта зависимость, конечно, не прямая, но *при одинаковых подходах* к физическому моделированию обычно так и происходит. Таким образом, можно принять допущение, что *чем больше атрибутов имеют отношения*, разработанные в ходе логического моделирования, *тем медленнее будут выполняться операции обновления данных*, за счет затраты времени на перестройку большего количества индексов.

Нормальные формы отношений

Скорость операций выборки данных

- Одно из назначений базы данных - предоставление информации пользователям. Информация извлекается из реляционной базы данных при помощи оператора SQL - SELECT. Одной из наиболее дорогостоящих операций при выполнении оператора SELECT является операция соединение таблиц. Таким образом, чем больше взаимосвязанных отношений было создано в ходе логического моделирования, тем больше вероятность того, что при выполнении запросов эти отношения будут соединяться, и, следовательно, тем медленнее будут выполняться запросы. Таким образом, *увеличение количества отношений приводит к замедлению выполнения операций выборки данных*, особенно, если запросы заранее неизвестны.

Нормальные формы отношений

Основной пример

- Рассмотрим в качестве предметной области некоторую организацию, выполняющую некоторые проекты. Модель предметной области опишем следующим неформальным текстом:

1. Сотрудники организации выполняют проекты.
2. Проекты состоят из нескольких заданий.
3. Каждый сотрудник может участвовать в одном или нескольких проектах, или временно не участвовать ни в каких проектах.
4. Над каждым проектом может работать несколько сотрудников, или временно проект может быть приостановлен, тогда над ним не работает ни один сотрудник.
5. Над каждым заданием в проекте работает ровно один сотрудник.
6. Каждый сотрудник числится в одном отделе.
7. Каждый сотрудник имеет телефон, находящийся в отделе сотрудника.

Нормальные формы отношений

Основной пример

- В ходе дополнительного уточнения того, какие данные необходимо учитывать, выяснилось следующее:

1. О каждом сотруднике необходимо хранить табельный номер и фамилию. Табельный номер является уникальным для каждого сотрудника.
2. Каждый отдел имеет уникальный номер.
3. Каждый проект имеет номер и наименование. Номер проекта является уникальным.
4. Каждая работа из проекта имеет номер, уникальный в пределах проекта. Работы в разных проектах могут иметь одинаковые номера.

Нормальные формы отношений

- В ходе логического моделирования на первом шаге предложено хранить данные в одном отношении, имеющем следующие атрибуты:
- **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** (***Н_СОТР***, **ФАМ**, **Н_ОТД**, **ТЕЛ**, ***Н_ПРО***, **ПРОЕКТ**, **Н_ЗАДАН**)
- где:
- ***Н_СОТР*** - табельный номер сотрудника
- **ФАМ** - фамилия сотрудника
- **Н_ОТД** - номер отдела, в котором числится сотрудник
- **ТЕЛ** - телефон сотрудника
- ***Н_ПРО*** - номер проекта, над которым работает сотрудник
- **ПРОЕКТ** - наименование проекта, над которым работает сотрудник
- **Н_ЗАДАН** - номер задания, над которым работает сотрудник
- Т.к. каждый сотрудник в каждом проекте выполняет ровно одно задание, то в качестве потенциального ключа отношения необходимо взять пару атрибутов {***Н_СОТР***, ***Н_ПРО***}.

Нормальные формы отношений

- В текущий момент состояние предметной области отражается следующими фактами:
- Сотрудник Иванов, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 1 и во втором проекте "Климат" задание 1.
- Сотрудник Петров, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 2.
- Сотрудник Сидоров, работающий во 2 отделе, выполняет в первом проекте "Космос" задание 3 и во втором проекте "Климат" задание 2.

Нормальные формы отношений

- Это состояние отражается в таблице (курсивом выделены ключевые атрибуты):

Н_СОТР	**ФАМ**	**Н_ОТД**	**ТЕЛ**	***Н_ПРО***	**ПРОЕКТ**	**Н_ЗАДАН**
1	Иванов	1	11-22-33	*1*	Космос	1
1	Иванов	1	11-22-33	*2*	Климат	1
2	Петров	1	11-22-33	*1*	Космос	2
3	Сидоров	2	33-22-11	*1*	Космос	3
3	Сидоров	2	33-22-11	*2*	Климат	2

Отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ

Нормальные формы отношений

- Для примера показано, что не все таблицы, аналогично приведенному отношению отражающие предметную область, могут быть отношениями:

Н_СОТР	**ФАМ**	**Н_ОТД**	**ТЕЛ**	***Н_ПРО***	**ПРОЕКТ**	**Н_ЗАДАН**
1	Иванов	1	11-22-33	*1*	Космос	1
				2	Климат	1
2	Петров	1	11-22-33	*1*	Космос	2
3	Сидоров	2	33-22-11	*1* *2*	Космос Климат	3 2

Таблица, не являющаяся отношением
СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ

Нормальные формы отношений

1НФ (Первая Нормальная Форма)

- ***Первая нормальная форма*** (***1НФ***) - это обычное отношение. Согласно нашему определению отношений, любое отношение автоматически уже находится в 1НФ. Напомним кратко свойства отношений (это и будут свойства 1НФ):

1. *В отношении нет одинаковых кортежей.*
2. *Кортежи не упорядочены.*
3. *Атрибуты не упорядочены и различаются по наименованию.*
4. *Все значения атрибутов атомарны.*

Нормальные формы отношений

Аномалии обновления

- Даже одного взгляда на таблицу отношения **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** достаточно, чтобы увидеть, что данные хранятся в ней с *большой избыточностью*. Во многих строках повторяются фамилии сотрудников, номера телефонов, наименования проектов. Кроме того, в данном отношении хранятся вместе независимые друг от друга данные - и данные о сотрудниках, и об отделах, и о проектах, и о работах по проектам. Пока никаких действий с отношением не производится, это не страшно. Но как только состояние предметной области изменяется, то, при попытках соответствующим образом изменить состояние базы данных, возникает большое количество проблем.
- Исторически эти проблемы получили название ***аномалии обновления***.
- ***Аномалии*** есть либо *неадекватность модели данных* предметной области, либо некоторые *дополнительные трудности в реализации ограничений* предметной области средствами СУБД.

Нормальные формы отношений

- Т.к. аномалии проявляют себя при выполнении операций, изменяющих состояние базы данных, то различают следующие виды аномалий:
- Аномалии вставки (INSERT)
- Аномалии обновления (UPDATE)
- Аномалии удаления (DELETE)
- В отношении **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** можно привести примеры следующих аномалий:

Нормальные формы отношений

Аномалии вставки (INSERT)

- В отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** нельзя вставить данные о сотруднике, который пока не участвует ни в одном проекте. Действительно, если, например, во втором отделе появляется новый сотрудник, скажем, Пушников, и он пока не участвует ни в одном проекте, то мы должны вставить в отношение кортеж (4, Пушников, 2, 33-22-11, null, null, null). Это сделать невозможно, т.к. атрибут ***Н_ПРО*** (номер проекта) входит в состав потенциального ключа, и, следовательно, не может содержать null-значений.
- Точно также нельзя вставить данные о проекте, над которым пока не работает ни один сотрудник.
- Причина аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту).
- Вывод - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

Нормальные формы отношений

Аномалии обновления (UPDATE)

- Фамилии сотрудников, наименования проектов, номера телефонов повторяются во многих кортежах отношения. Поэтому если сотрудник меняет фамилию, или проект меняет наименование, или меняется номер телефона, то такие изменения необходимо *одновременно* выполнить во всех местах, где эта фамилия, наименование или номер телефона встречаются, иначе отношение станет некорректным (например, один и тот же проект в разных кортежах будет называться по-разному). Таким образом, обновление базы данных одним действием реализовать невозможно. Для поддержания отношения в целостном состоянии необходимо написать триггер, который при обновлении одной записи корректно исправлял бы данные и в других местах.
- Причина аномалии - избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация.
- Вывод - увеличивается сложность разработки базы данных. БД, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Нормальные формы отношений

Аномалии удаления (DELETE)

- При удалении некоторых данных может произойти потеря другой информации. Например, если закрыть проект "Космос" и удалить все строки, в которых он встречается, то будут потеряны все данные о сотруднике Петрове. Если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11. Если по проекту временно прекращены работы, то при удалении данных о работах по этому проекту будут удалены и данные о самом проекте (наименование проекта). При этом если был сотрудник, который работал только над этим проектом, то будут потеряны и данные об этом сотруднике.
- Причина аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту).
- Вывод - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

Нормальные формы отношений

Определение функциональной зависимости

- Для устранения указанных аномалий (а на самом деле *для правильного проектирования модели данных*!) применяется метод нормализации отношений. Нормализация основана на понятии функциональной зависимости атрибутов отношения.

- *Определение 1*. Пусть R - отношение. Множество атрибутов Y ***функционально зависимо*** от множества атрибутов X (X ***функционально определяет Y***) тогда и только тогда, когда *для любого состояния* отношения R для любых кортежей $r_1, r_2 \in R$ из того, что $r_1X = r_2X$ следует что $r_1Y = r_2Y$ (т.е. во всех кортежах, имеющих одинаковые значения атрибутов X, значения атрибутов Y также совпадают *в любом состоянии* отношения R). Символически функциональная зависимость записывается $X \rightarrow Y$.

- Множество атрибутов X называется ***детерминантом функциональной зависимости***, а множество атрибутов Y называется ***зависимой частью***.

Нормальные формы отношений

- Замечание. Если атрибуты X составляют потенциальный ключ отношения R, то *любой* атрибут отношения R функционально зависит от X.

- **Пример 1**. В отношении **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** можно привести следующие примеры функциональных зависимостей:
- Зависимость атрибутов от ключа отношения:
- **{*Н_СОТР*, *Н_ПРО*}** → **ФАМ**
- **{*Н_СОТР*, *Н_ПРО*}** → **Н_ОТД**
- **{*Н_СОТР*, *Н_ПРО*}** → **ТЕЛ**
- **{*Н_СОТР*, *Н_ПРО*}** → **ПРОЕКТ**
- **{*Н_СОТР*, *Н_ПРО*}** → **Н_ЗАДАН**

Нормальные формы отношений

- Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:
- ***Н_СОТР*** → **ФАМ**
- ***Н_СОТР*** → **Н_ОТД**
- ***Н_СОТР*** → **ТЕЛ**
- Зависимость наименования проекта от номера проекта:
- ***Н_ПРО*** → **ПРОЕКТ**
- Зависимость номера телефона от номера отдела:
- **Н_ОТД** → **ТЕЛ**

Нормальные формы отношений

- Замечание. Приведенные функциональные зависимости *не выведены* из внешнего вида отношения, приведенного в таблице 1. Эти зависимости отражают взаимосвязи, обнаруженные между объектами предметной области и являются дополнительными ограничениями, определяемыми предметной областью. Таким образом, функциональная зависимость - *семантическое понятие*. Она возникает, когда по значениям одних данных в предметной области можно определить значения других данных. Например, зная табельный номер сотрудника, можно определить его фамилию, по номеру отдела можно определить телефон. Функциональная зависимость *задает дополнительные ограничения* на данные, которые могут храниться в отношениях. Для корректности базы данных (адекватности предметной области) необходимо при выполнении операций модификации базы данных проверять все ограничения, определенные функциональными зависимостями.

Нормальные формы отношений

2НФ (Вторая Нормальная Форма)

- *Определение 3*. Отношение R находится во ***второй нормальной форме*** (***2НФ***) тогда и только тогда, когда отношение находится в 1НФ и *нет неключевых атрибутов, зависящих от части сложного ключа*. (***Неключевой атрибут*** - это атрибут, не входящий в состав никакого потенциального ключа).

- Замечание. Если потенциальный ключ отношения является простым, то отношение автоматически находится в 2НФ.

Нормальные формы отношений

- Отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** не находится в 2НФ, т.к. есть атрибуты, зависящие от части сложного ключа:
- Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника является зависимостью от части сложного ключа:
- ***Н_СОТР*** → **ФАМ**
- ***Н_СОТР*** → **Н_ОТД**
- ***Н_СОТР*** → **ТЕЛ**
- Зависимость наименования проекта от номера проекта является зависимостью от части сложного ключа:
- ***Н_ПРО*** → **ПРОЕКТ**
- Для того, чтобы устранить зависимость атрибутов от части сложного ключа, нужно произвести ***декомпозицию*** отношения на несколько отношений. При этом *те атрибуты, которые зависят от части сложного ключа,* выносятся в отдельное отношение.

Нормальные формы отношений

- Отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** декомпозируем на три отношения - **СОТРУДНИКИ_ОТДЕЛЫ**, **ПРОЕКТЫ**, **ЗАДАНИЯ**.
- Отношение **СОТРУДНИКИ_ОТДЕЛЫ** (***Н_СОТР***, **ФАМ**, **Н_ОТД**, **ТЕЛ**):
- Функциональные зависимости:
- Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:
- ***Н_СОТР*** → **ФАМ**
- ***Н_СОТР*** → **Н_ОТД**
- ***Н_СОТР*** → **ТЕЛ**
- Зависимость номера телефона от номера отдела:
- **Н_ОТД** → **ТЕЛ**

Нормальные формы отношений

- Отношение **СОТРУДНИКИ_ОТДЕЛЫ** (***Н_СОТР***, **ФАМ**, **Н_ОТД**, **ТЕЛ**):

Н_СОТР	**ФАМ**	**Н_ОТД**	**ТЕЛ**
1	Иванов	1	11-22-33
2	Петров	1	11-22-33
3	Сидоров	2	33-22-11

Нормальные формы отношений

- Отношение **ПРОЕКТЫ** (***Н_ПРО***, **ПРОЕКТ**):
- Функциональные зависимости:
- ***Н_ПРО*** → **ПРОЕКТ**

Н_ПРО	**ПРОЕКТ**
1	Космос
2	Климат

Нормальные формы отношений

- Отношение **ЗАДАНИЯ** (***Н_СОТР***, ***Н_ПРО***, **Н_ЗАДАН**):
- Функциональные зависимости:
- **{*Н_СОТР*, *Н_ПРО*}** → **Н_ЗАДАН**

Н_СОТР	***Н_ПРО***	**Н_ЗАДАН**
1	*1*	1
1	*2*	1
2	*1*	2
3	*1*	3
3	*2*	2

Нормальные формы отношений

Анализ декомпозированных отношений

- Отношения, полученные в результате декомпозиции, находятся в 2НФ. Действительно, отношения **СОТРУДНИКИ_ОТДЕЛЫ** и **ПРОЕКТЫ** имеют простые ключи, следовательно автоматически находятся в 2НФ, отношение **ЗАДАНИЯ** имеет сложный ключ, но единственный неключевой атрибут **Н_ЗАДАН** функционально зависит от всего ключа **{*Н_СОТР*, *Н_ПРО*}**.
- Часть аномалий обновления устранена. Так, данные о сотрудниках и проектах теперь хранятся в различных отношениях, поэтому при появлении сотрудников, не участвующих ни в одном проекте просто добавляются кортежи в отношение **СОТРУДНИКИ_ОТДЕЛЫ**. Точно также, при появлении проекта, над которым не работает ни один сотрудник, просто вставляется кортеж в отношение **ПРОЕКТЫ**.
- Фамилии сотрудников и наименования проектов теперь хранятся без избыточности. Если сотрудник сменит фамилию или проект сменит наименование, то такое обновление будет произведено в одном месте.

Нормальные формы отношений

- Если по проекту временно прекращены работы, но требуется, чтобы сам проект сохранился, то для этого проекта удаляются соответствующие кортежи в отношении **ЗАДАНИЯ**, а данные о самом проекте и данные о сотрудниках, участвовавших в проекте, остаются в отношениях **ПРОЕКТЫ** и **СОТРУДНИКИ_ОТДЕЛЫ**.
- Тем не менее, часть аномалий разрешить не удалось.

Нормальные формы отношений

Оставшиеся аномалии вставки (INSERT)

- В отношение **СОТРУДНИКИ_ОТДЕЛЫ** нельзя вставить кортеж (4, Пушников, 1, 33-22-11), т.к. при этом получится, что два сотрудника из 1-го отдела (Иванов и Пушников) имеют разные номера телефонов, а это противоречит модели предметной области. В этой ситуации можно предложить два решения, в зависимости от того, что реально произошло в предметной области. Другой номер телефона может быть введен по двум причинам - по ошибке человека, вводящего данные о новом сотруднике, или потому что номер в отделе действительно изменился. Тогда можно написать триггер, который при вставке записи о сотруднике проверяет, совпадает ли телефон с уже имеющимся телефоном у другого сотрудника этого же отдела. Если номера отличаются, то система должна задать вопрос, оставить ли старый номер в отделе или заменить его новым. Если нужно оставить старый номер (новый номер введен ошибочно), то кортеж с данными о новом сотруднике будет вставлен, но номер телефона будет у него будет тот, который уже есть в отделе (в данном случае, 11-22-33).

Нормальные формы отношений

- Если же номер в отделе действительно изменился, то кортеж будет вставлен с новым номером, и одновременно будут изменены номера телефонов у всех сотрудников этого же отдела. И в том и в другом случае не обойтись без разработки громоздкого триггера.
- Причина аномалии - избыточность данных, порожденная тем, что в одном отношении хранится разнородная информация (о сотрудниках и об отделах).
- Вывод - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Нормальные формы отношений

Оставшиеся аномалии обновления (UPDATE)

- Одни и те же номера телефонов повторяются во многих кортежах отношения. Поэтому если в отделе меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где этот номер телефона встречаются, иначе отношение станет некорректным. Таким образом, обновление базы данных одним действием реализовать невозможно. Необходимо написать триггер, который при обновлении одной записи корректно исправляет номера телефонов в других местах.
- Причина аномалии - избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация.
- Вывод - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Нормальные формы отношений

Оставшиеся аномалии удаления (DELETE)

- При удалении некоторых данных по-прежнему может произойти потеря другой информации. Например, если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11.
- Причина аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и об отделах).
- Вывод - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

- Заметим, что при переходе ко второй нормальной форме отношения стали *почти* адекватными предметной области. Остались также трудности в разработке базы данных, связанные с необходимостью написания триггеров, поддерживающих целостность базы данных. Эти трудности теперь связаны только с одним отношением **СОТРУДНИКИ_ОТДЕЛЫ**.

Нормальные формы отношений

3НФ (Третья Нормальная Форма)

- *Определение 4*. Атрибуты называются ***взаимно независимыми***, если ни один из них не является функционально зависимым от другого.

- *Определение 5*. Отношение R находится в ***третьей нормальной форме*** (***3НФ***) тогда и только тогда, когда отношение находится в 2НФ и *все неключевые атрибуты взаимно независимы*.

Нормальные формы отношений

- Отношение **СОТРУДНИКИ_ОТДЕЛЫ** не находится в 3НФ, т.к. имеется функциональная зависимость неключевых атрибутов (зависимость номера телефона от номера отдела):
- **Н_ОТД → ТЕЛ**
- Для того, чтобы устранить зависимость неключевых атрибутов, нужно произвести декомпозицию отношения на несколько отношений. При этом *те неключевые атрибуты, которые являются зависимыми,* выносятся в отдельное отношение.
- Отношение **СОТРУДНИКИ_ОТДЕЛЫ** декомпозируем на два отношения - **СОТРУДНИКИ**, **ОТДЕЛЫ**.

Нормальные формы отношений

- Отношение **СОТРУДНИКИ** (***Н_СОТР***, **ФАМ**, **Н_ОТД**):
- Функциональные зависимости:
- Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:
- ***Н_СОТР*** → **ФАМ**
- ***Н_СОТР*** → **Н_ОТД**

Н_СОТР	**ФАМ**	**Н_ОТД**
1	Иванов	1
2	Петров	1
3	Сидоров	2

Нормальные формы отношений

- Отношение **ОТДЕЛЫ** (***Н_ОТД***, **ТЕЛ**):
- Функциональные зависимости:
- Зависимость номера телефона от номера отдела:
- ***Н_ОТД*** → **ТЕЛ**

Н_ОТД	**ТЕЛ**
1	11-22-33
2	33-22-11

Нормальные формы отношений

- Обратим внимание на то, что атрибут **Н_ОТД**, *не являвшийся ключевым* в отношении **СОТРУДНИКИ_ОТДЕЛЫ**, *становится потенциальным ключом* в отношении **ОТДЕЛЫ**. Именно за счет этого устраняется избыточность, связанная с многократным хранением одних и тех же номеров телефонов.
- Вывод. Таким образом, все обнаруженные аномалии обновления устранены. Реляционная модель, состоящая из четырех отношений **СОТРУДНИКИ**, **ОТДЕЛЫ**, **ПРОЕКТЫ**, **ЗАДАНИЯ**, находящихся в третьей нормальной форме, является адекватной описанной модели предметной области, и требует наличия только тех триггеров, которые поддерживают ссылочную целостность. Такие триггеры являются стандартными и не требуют больших усилий в разработке.

Нормальные формы отношений

Алгоритм нормализации (приведение к 3НФ)

- Итак, алгоритм нормализации (т.е. алгоритм приведения отношений к 3НФ) описывается следующим образом.
- *Шаг 1 (Приведение к 1НФ)*. На первом шаге задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области (не по внешнему виду полученных отношений!) выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1НФ.
- *Шаг 2 (Приведение к 2НФ)*. Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводим декомпозицию этих отношений на несколько отношений следующим образом: те атрибуты, которые зависят от части сложного ключа выносятся в отдельное отношение вместе с этой частью ключа. В исходном отношении остаются все ключевые атрибуты.

Нормальные формы отношений

- *Шаг 3 (Приведение к 3НФ)*. Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов от других неключевых атрибутов, то проводим декомпозицию этих отношений следующим образом: те неключевые атрибуты, которые зависят от других неключевых атрибутов выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости.

Нормальные формы отношений

- Замечание. На практике, при создании логической модели данных, как правило, не следуют прямо приведенному алгоритму нормализации. Опытные разработчики обычно сразу строят отношения в 3НФ. Кроме того, основным средством разработки логических моделей данных являются различные варианты ER-диаграмм. Особенность этих диаграмм в том, что они сразу позволяют создавать отношения в 3НФ. Тем не менее, приведенный алгоритм важен по двум причинам. *Во-первых*, этот алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений. *Во-вторых*, как правило, модель предметной области никогда не бывает правильно разработана с первого шага. Эксперты предметной области могут забыть о чем-либо упомянуть, разработчик может неправильно понять эксперта, во время разработки могут измениться правила, принятые в предметной области, и т.д. Все это может привести к появлению новых зависимостей, которые отсутствовали в первоначальной модели предметной области. Тут как раз и необходимо использовать алгоритм нормализации хотя бы для того, чтобы убедиться, что отношения остались в 3НФ и логическая модель не ухудшилась.

Нормальные формы отношений

Сравнение нормализованных и ненормализованных моделей

- Соберем воедино результаты анализа критериев, по которым мы хотели оценить влияние логического моделирования данных на качество физических моделей данных и производительность базы данных:

Критерий	**Отношения слабо нормализованы (1НФ, 2НФ)**	**Отношения сильно нормализованы (3НФ)**
Адекватность базы данных предметной области	ХУЖЕ (-)	ЛУЧШЕ (+)
Легкость разработки и сопровождения базы данных	СЛОЖНЕЕ (-)	ЛЕГЧЕ (+)
Скорость выполнения вставки, обновления, удаления	МЕДЛЕННЕЕ (-)	БЫСТРЕЕ (+)
Скорость выполнения выборки данных	БЫСТРЕЕ (+)	МЕДЛЕННЕЕ (-)

Нормальные формы отношений

- Как видно из таблицы, более сильно нормализованные отношения оказываются лучше спроектированы (три плюса, один минус). Они больше соответствуют предметной области, легче в разработке, для них быстрее выполняются операции модификации базы данных. Правда, это достигается ценой некоторого замедления выполнения операций выборки данных.
- У слабо нормализованных отношений единственное преимущество - если к базе данных обращаться только с запросами на выборку данных, то для слабо нормализованных отношений такие запросы выполняются быстрее. Это связано с тем, что в таких отношениях уже как бы произведено соединение отношений и на это не тратится время при выборке данных.
- Таким образом, выбор степени нормализации отношений зависит от характера запросов, с которыми чаще всего обращаются к базе данных.

Нормальные формы отношений

OLTP и OLAP-системы

- Можно выделить некоторые классы систем, для которых больше подходят сильно или слабо нормализованные модели данных.
- Сильно нормализованные модели данных хорошо подходят для так называемых ***OLTP-приложений*** (***On-Line Transaction Processing*** (***OLTP***)- ***оперативная обработка транзакций***). Типичными примерами OLTP-приложений являются системы складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег, и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Сами транзакции выглядят относительно просто, например, "снять сумму денег со счета А, добавить эту сумму на счет В". Проблема заключается в том, что, во-первых, транзакций очень много, во-вторых, выполняются они одновременно (к системе может быть подключено несколько тысяч одновременно работающих пользователей), в-третьих, при возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции

Нормальные формы отношений

- (не должно быть ситуации, когда деньги сняты со счета A, но не поступили на счет B). Практически все запросы к базе данных в OLTP-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для предоставления пользователям возможности выбора из различных справочников. Большая часть запросов, таким образом, известна заранее еще на этапе проектирования системы. Таким образом, критическим для OLTP-приложений является скорость и надежность выполнения коротких операций обновления данных. Чем выше уровень нормализации данных в OLTP-приложении, тем оно, как правило, быстрее и надежнее. Отступления от этого правила могут происходить тогда, когда уже на этапе разработки известны некоторые часто возникающие запросы, требующие соединения отношений и от скорости выполнения которых существенно зависит работа приложений. В этом случае можно пожертвовать нормализацией для ускорения выполнения подобных запросов.

Нормальные формы отношений

- Другим типом приложений являются так называемые ***OLAP-приложения*** (***On-Line Analitical Processing*** (***OLAP***) - ***оперативная аналитическая обработка данных***). Это обобщенный термин, характеризующий принципы построения ***систем поддержки принятия решений*** (***Decision Support System*** - ***DSS***), ***хранилищ данных*** (***Data Warehouse***), ***систем интеллектуального анализа данных*** (***Data Mining***). Такие системы предназначены для нахождения зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей), для проведения анализа "что если…". OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми их электронных таблиц или из других источников данных. Такие системы характеризуются следующими признаками:
- Добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложения).

Нормальные формы отношений

- Данные, добавленные в систему, обычно никогда не удаляются.
- Перед загрузкой данные проходят различные процедуры "очистки", связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, ошибочны.
- Запросы к системе являются нерегламентированными и, как правило, достаточно сложными. Очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса.
- Скорость выполнения запросов важна, но не критична.
- Данные OLAP-приложений обычно представлены в виде одного или нескольких гиперкубов, измерения которого представляют собой справочные данные, а в ячейках самого гиперкуба хранятся собственно данные. Например, можно построить гиперкуб, измерениями которого являются: время (в кварталах, годах), тип товара и отделения компании, а в ячейках хранятся объемы продаж.

Нормальные формы отношений

- Такой гиперкуб будет содержать данных о продажах различных типов товаров по кварталам и подразделениям. Основываясь на этих данных, можно отвечать на вопросы вроде "у какого подразделения самые лучшие объемы продаж в текущем году?", или "каковы тенденции продаж отделений Юго-Западного региона в текущем году по сравнению с предыдущим годом?"
- Физически гиперкуб может быть построен на основе специальной ***многомерной модели данных*** (***MOLAP*** - ***Multidimensional OLAP***) или построен средствами реляционной модели данных (***ROLAP*** - ***Relational OLAP***).
- Возвращаясь к проблеме нормализации данных, можно сказать, что в системах OLAP, использующих реляционную модель данных (ROLAP), данные целесообразно хранить в виде слабо нормализованных отношений, содержащих заранее вычисленные основные итоговые данные. Большая избыточность и связанные с ней проблемы тут не страшны, т.к. обновление происходит только в момент загрузки новой порции данных. При этом происходит как добавление новых данных, так и пересчет итогов.

Нормальные формы более высоких порядков

- Ранее были рассмотрены нормальные формы вплоть до третьей нормальной формы (3НФ). В большинстве случаев этого вполне достаточно, чтобы разрабатывать вполне работоспособные базы данных. Далее рассматриваются нормальные формы более высоких порядков, а именно, нормальная форма Бойса-Кодда (НФБК), четвертая нормальная форма (4НФ), пятая нормальная форма (5НФ).

НФБК (Нормальная Форма Бойса-Кодда)

- При приведении отношений при помощи алгоритма нормализации к отношениям в 3НФ неявно предполагалось, что все отношения содержат один потенциальный ключ. Это не всегда верно. Рассмотрим следующий пример отношения, содержащего два ключа.

Нормальные формы более высоких порядков

- **Пример 1**. Пусть требуется хранить данные о поставках деталей некоторыми поставщиками. Предположим, что наименования поставщиков являются уникальными. Кроме того, каждый поставщик имеет свой уникальный номер. Данные о поставках можно хранить в следующем отношении "Поставки»:

Номер поставщика PNUM	**Наименование поставщика PNAME**	**Номер детали DNUM**	**Поставляемое количество VOLUME**
1	Фирма 1	1	100
1	Фирма 1	2	200
1	Фирма 1	3	300
2	Фирма 2	1	150
2	Фирма 2	2	250
3	Фирма 3	1	1000

Нормальные формы более высоких порядков

- Данное отношение содержит два потенциальных ключа - **{*PNUM*, *DNUM*}** и **{*PNAME*, *DNUM*}**. Видно, что данные хранятся в отношении с избыточностью - при изменении наименования поставщика, это наименование нужно изменить во всех кортежах, где оно встречается. Можно ли эту аномалию устранить при помощи алгоритма нормализации, описанного в предыдущей главе? Для этого нужно выявить имеющиеся функциональные зависимости (как обычно, курсивом выделены ключевые атрибуты):
- ***PNUM*** → ***PNAME*** - наименование поставщика зависит от номера поставщика.
- ***PNAME*** → ***PNUM*** - номер поставщика зависит от наименования поставщика.
- **{*PNUM*, *DNUM*}** → **VOLUME** - поставляемое количество зависит от первого ключа отношения.
- **{*PNUM*, *DNUM*}** → ***PNAME*** - наименование поставщика зависит от первого ключа отношения.
- **{*PNAME*, *DNUM*}** → **VOLUME** - поставляемое количество зависит от второго ключа отношения.

Нормальные формы более высоких порядков

- ***{PNAME, DNUM}*** → ***PNUM*** - номер поставщика зависит от второго ключа отношения.
- Данное отношение не содержит *неключевых* атрибутов, зависящих от части сложного ключа (см. определение 2НФ). Действительно, от части сложного ключа зависят атрибуты ***PNAME*** и ***PNUM***, но они сами являются ключевыми. Таким образом, отношение находится в 2НФ.
- Кроме того, отношение не содержит зависимых друг от друга *неключевых* атрибутов, т.к. неключевой атрибут всего один - **VOLUME** (см. определение 3НФ). Таким образом, показано, что отношение "Поставки" находится в 3НФ.
- Таким образом, описанный ранее алгоритм нормализации неприменим к данному отношению. Очевидно, однако, что аномалия данного отношения устраняется путем декомпозиции его на два следующих отношения:

Нормальные формы более высоких порядков

- **Отношение "Поставщики"**

Номер поставщика PNUM	**Наименование поставщика PNAME**
1	Фирма 1
2	Фирма 2
3	Фирма 3

- **Отношение "Поставки-2"**

Номер поставщика PNUM	**Номер детали DNUM**	**Поставляемое количество VOLUME**
1	1	100
1	2	200
1	3	300
2	1	150
2	2	250
3	1	1000

Нормальные формы более высоких порядков

- *Определение 1*. Отношение находится в ***нормальной форме Бойса-Кодда*** (***НФБК***) тогда и только тогда, когда *детерминанты всех функциональных зависимостей являются потенциальными ключами*.
- Замечание. Если отношение находится в НФБК, то оно автоматически находится и в 3НФ. Действительно, это сразу следует из определения 3НФ.

Нормальные формы более высоких порядков

- Отношение "Поставки" не находится в НФБК, т.к. имеются зависимости (***PNUM*** → ***PNAME*** и ***PNAME*** → ***PNUM***), детерминанты которых не являются потенциальными ключами.
- Для того чтобы устранить зависимость от детерминантов, не являющихся потенциальными ключами, необходимо провести декомпозицию, вынося эти детерминанты и зависимые от них части в отдельное отношение. Отношения "Поставщики" и "Поставки-2", полученные в результате декомпозиции находятся в НФБК.

- Замечание. Приведенная декомпозиция отношения "Поставки" на отношения "Поставщики" и "Поставки-2" не является единственно возможной.

Нормальные формы более высоких порядков

- **4НФ (Четвертая Нормальная Форма)**
- Рассмотрим следующий пример. Пусть требуется учитывать данные об абитуриентах, поступающих в ВУЗ. При анализе предметной области были выделены следующие требования:

  - Каждый абитуриент имеет право сдавать экзамены на несколько факультетов одновременно.

  - Каждый факультет имеет свой список сдаваемых предметов.

  - Один и тот же предмет может сдаваться на нескольких факультетах.

  - Абитуриент обязан сдавать все предметы, указанные для факультета, на который он поступает, несмотря на то, что он, может быть, уже сдавал такие же предметы на другом факультете.
- Предположим, что нам требуется хранить данные о том, какие предметы должен сдавать каждый абитуриент. Попытаемся хранить данные в одном отношении "Абитуриенты-Факультеты-Предметы":

Нормальные формы более высоких порядков

- В данный момент в отношении хранится информация о том, что абитуриент Иванов поступает на два факультета (математически и физический), а абитуриент Петров - только на математический. Кроме того, можно сделать вывод, что на математическом факультете нужно сдавать математику и информатику, а на физическом - математику и физику.

Абитуриент	**Факультет**	**Предмет**
Иванов	Математический	Математика
Иванов	Математический	Информатика
Иванов	Физический	Математика
Иванов	Физический	Физика
Петров	Математический	Математика
Петров	Математический	Информатика

Отношение "Абитуриенты-Факультеты-Предметы"

Нормальные формы более высоких порядков

- Кажется, что в отношении имеется аномалия обновления, связанная с тем, что дублируются фамилии абитуриентов, наименования факультетов и наименования предметов. Однако эта аномалия легко устраняется стандартным способом - вынесением всех наименований в отдельные отношения, оставляя в исходном отношении только соответствующие номера:

Номер Абитуриента	**Номер Факультета**	**Номер Предмета**
1	1	1
1	1	2
1	2	1
1	2	3
2	1	1
2	1	2

Модифицированное отношение "Абитуриенты-Факультеты-Предметы"

Нормальные формы более высоких порядков

Номер Абитуриента	**Абитуриент**
1	Иванов
2	Петров

Отношение "Абитуриенты"

Номер Факультета	**Факультет**
1	Математический
2	Физический

Отношение "Факультеты"

Номер Предмета	**Предмет**
1	Математика
2	Информатика
3	Физика

Отношение "Предметы"

Нормальные формы более высоких порядков

- Теперь каждое наименование встречается только в одном месте.
- И все-таки как в исходном, так и в модифицированном отношении имеются аномалии обновления, возникающие при попытке вставить или удалить кортежи.
- *Аномалия вставки*. При попытке добавить в отношение "Абитуриенты-Факультеты-Предметы" новый кортеж, например (Сидоров, Математический, Математика), мы *обязаны* добавить также и кортеж (Сидоров, Математический, Информатика), т.к. все абитуриенты математического факультета *обязаны* иметь один и тот же список сдаваемых предметов. Соответственно, при попытке вставить в модифицированное отношении кортеж (3, 1, 1), мы обязаны вставить в него также и кортеж (3, 1, 2).
- Аномалия удаления. При попытке удалить кортеж (Иванов, Математический, Математика), мы *обязаны* удалить также и кортеж (Иванов, Математический, Информатика) по той же самой причине.
- Таким образом, вставка и удаление кортежей не может быть выполнена независимо от других кортежей отношения.

Нормальные формы более высоких порядков

- Кроме того, если мы удалим кортеж (Иванов, Физический, Математика), а вместе с ним и кортеж (Иванов, Физический, Физика), то будет потеряна информация о предметах, которые должны сдаваться на физическом факультете.
- Декомпозиция отношения "Абитуриенты-Факультеты-Предметы" для устранения указанных аномалий не может быть выполнена на основе функциональных зависимостей, т.к. это отношение *не содержит никаких функциональных зависимостей*. Это отношение является полностью ключевым, т.е. ключом отношения является все множество атрибутов. Но ясно, что какая-то взаимосвязь между атрибутами имеется. Эта взаимосвязь описывается понятием ***многозначной зависимости***.

Нормальные формы более высоких порядков

- *Определение 2*. Пусть R - отношение, и X, Y, Z - некоторые из его атрибутов (или *непересекающиеся* множества атрибутов).
- Тогда атрибуты (множества атрибутов) Y и Z ***многозначно зависят*** от X (обозначается $X \rightarrow\rightarrow Y \mid Z$), тогда и только тогда, когда из того, что в отношении R содержатся кортежи $r_1=(x,y,z_1)$ и $r_2=(x,y_1,z)$ следует, что в отношении R содержится также и кортеж $r_3=(x,y,z)$.
- Замечание. Меняя местами кортежи r_1 и r_2 в определении многозначной зависимости, получим, что в отношении R должен содержаться также и кортеж $r_4=(x,y_1,z_1)$. Таким образом, атрибуты Y и Z, многозначно зависящие от X, ведут себя "симметрично" по отношению к атрибуту X.

Нормальные формы более высоких порядков

- В отношении "Абитуриенты-Факультеты-Предметы" имеется многозначная зависимость Факультет →→ Абитуриент|Предмет.
- Словами это можно выразить так - для каждого факультета (для каждого значения из X) каждый поступающий на него абитуриент (значение из Y) сдает *один и тот же список* предметов (набор значений из Z), и для каждого факультета (для каждого значения из X) каждый сдаваемый на факультете экзамен (значение из Z) сдается *одним и тем же списком* абитуриентов (набор значений из Y). Именно наличие этой зависимости не позволяет независимо вставлять и удалять кортежи. Кортежи обязаны вставляться и удаляться одновременно *целыми наборами*.

Нормальные формы более высоких порядков

- Замечание. Если в отношении R имеется не менее трех атрибутов X, Y, Z и есть *функциональная* зависимость $X \rightarrow Y$, то есть и *многозначная* зависимость $X \rightarrow\rightarrow Y|Z$.
- Действительно, действуя формально в соответствии с определением многозначной зависимости, предположим, что в отношении R содержатся кортежи $r_1=(x,y,z_1)$ и $r_2=(x,y_1,z)$. В силу функциональной зависимости $X \rightarrow Y$ отсюда следует, что $y=y_1$. Но тогда кортеж $r_3=(x,y,z)$ в точности совпадает с кортежем $r_2=(x,y_1,z)$ и, следовательно, содержится в отношении R. Таким образом, имеется многозначная зависимость $X \rightarrow\rightarrow Y|Z$.
- Таким образом, *понятие многозначной зависимости является обобщением понятия функциональной зависимости*.

Нормальные формы более высоких порядков

- *Определение 3*. Многозначная зависимость X →→ Y|Z называется ***нетривиальной многозначной зависимостью***, если *не существует функциональных зависимостей* X → Y и X → Z.
- В отношении "Абитуриенты-Факультеты-Предметы" имеется именно нетривиальная многозначная зависимость Факультет →→ Абитуриент|Предмет.

- *Определение 4*. Отношение находится в ***четвертой нормальной форме*** (***4НФ***) тогда и только тогда, когда отношение находится в НФБК и *не содержит нетривиальных многозначных зависимостей*.

Нормальные формы более высоких порядков

- Отношение "Абитуриенты-Факультеты-Предметы" находится в НФБК, но не в 4НФ. Это отношение можно без потерь декомпозировать на отношения:

Факультет	**Абитуриент**
Математический	Иванов
Физический	Иванов
Математический	Петров

Отношение "Факультеты-Абитуриенты"

Факультет	**Предмет**
Математический	Математика
Математический	Информатика
Физический	Математика
Физический	Физика

Отношение "Факультеты-Предметы"

Нормальные формы более высоких порядков

- В полученных отношениях устранены аномалии вставки и удаления, характерные для отношения "Абитуриенты-Факультеты-Предметы".
- Заметим, что полученные отношения остались полностью ключевыми, и в них по-прежнему нет функциональных зависимостей.
- Отношения с нетривиальными многозначными зависимостями возникают, как правило, в результате естественного соединения двух отношений по общему полю, которое *не является ключевым ни в одном из отношений*. Фактически это приводит к попытке хранить в одном отношении информацию о двух *независимых* сущностях. В качестве еще одного примера можно привести ситуацию, когда сотрудник может иметь много работ и много детей. Хранение информации о работах и детях в одном отношении приводит к возникновению нетривиальной многозначной зависимости Работник →→ Работа|Дети.

Нормальные формы более высоких порядков

- **5НФ (Пятая Нормальная Форма)**
- Функциональные и многозначные зависимости позволяют произвести декомпозицию исходного отношения без потерь на *две* проекции. Можно, однако, привести примеры отношений, которые нельзя декомпозировать без потерь ни на какие две проекции.
- **Пример 3**. Рассмотрим следующее отношение R:

X	Y	Z
1	1	2
1	2	1
2	1	1
1	1	1

Нормальные формы более высоких порядков

- Всевозможные проекции отношения , включающие по два атрибута, имеют вид:

X	Y
1	1
1	2
2	1

Проекция R_1=R[X,Y]

X	Z
1	2
1	1
2	1

Проекция R_2=R[X,Z]

Y	Z
1	2
2	1
1	1

Проекция R_3=R[Y,Z]

Как легко заметить, отношение R не восстанавливается ни по одному из *попарных* соединений R_1 JOIN R_2, R_1 JOIN R_3 или R_2 JOIN R_3. Действительно, соединение R_1 JOIN R_2 имеет вид:

X	Y	Z
1	1	2
1	1	1
1	2	2
1	2	1
2	1	1

Нормальные формы более высоких порядков

- Серым цветом выделен лишний кортеж, отсутствующий в отношении R. Аналогично (в силу соображений симметрии) и другие попарные соединения не восстанавливают отношения R.
- Однако отношение R восстанавливается соединением *всех трех* проекций:

  R_1 JOIN R_2 JOIN R_3=R.
- Это говорит о том, что между атрибутами этого отношения также имеется некоторая зависимость, но эта зависимость не является ни функциональной, ни многозначной зависимостью.

Нормальные формы более высоких порядков

- *Определение 5*. Пусть R является отношением, а A, B, …, Z- произвольными (возможно пересекающимися) *подмножествами* множества атрибутов отношения R. Тогда отношение R удовлетворяет ***зависимости соединения***

  *(A, B, …, Z)

  тогда и только тогда, когда оно равносильно соединению всех своих проекций с подмножествами атрибутов A, B, …, Z, т.е.
- R=R[A] JOIN R[B] JOIN … JOIN R[Z].

- Можно *предположить*, что отношение R в примере удовлетворяет следующей зависимости соединения:
- *(XY, XZ, YZ).

Нормальные формы более высоких порядков

- Можно доказать, что *многозначная зависимость* является частным случаем *зависимости соединения*, т.е., если в отношении имеется многозначная зависимость, то имеется и зависимость соединения. Обратное, конечно, неверно.

- *Определение 6*. Зависимость соединения *(A, B, …, Z) называется ***нетривиальной зависимостью соединения***, если выполняется два условия:

  - Одно из множеств атрибутов A, B, …, Z *не содержит потенциального ключа* отношения R.

  - Ни одно из множеств атрибутов не совпадает со всем множеством атрибутов отношения R.

Нормальные формы более высоких порядков

- Для удобства работы сформулируем это определение так же и в отрицательной форме:

- *Определение 7*. Зависимость соединения *(A, B, …, Z) называется ***тривиальной зависимостью соединения***, если выполняется одно из условий:

  - Либо *все множества атрибутов* A, B, …, Z *содержат потенциальный ключ* отношения R.

  - Либо одно из множеств атрибутов *совпадает со всем множеством атрибутов* отношения R.

Нормальные формы более высоких порядков

- *Определение 8*. Отношение R находится в ***пятой нормальной форме*** (***5НФ***) тогда и только тогда, когда *любая имеющаяся зависимость соединения является тривиальной*.

- Определения 5НФ может стать более понятным, если сформулировать его в отрицательной форме:

- *Определение 9*. Отношение R *не находится* в ***5НФ***, если в отношении *найдется нетривиальная зависимость соединения*.

- Возвращаясь к примеру, становится понятно, что не зная ничего о том, какие потенциальные ключи имеются в отношении и как взаимосвязаны атрибуты, нельзя делать выводы о том, находится ли данное отношение в 5НФ (как, впрочем, и в других нормальных формах). По данному конкретному примеру можно только предположить, что отношение в примере не находится в 5НФ.

Нормальные формы более высоких порядков

- **Продолжение алгоритма нормализации (приведение к 5НФ)**
- *Шаг 4 (Приведение к НФБК)*. Если имеются отношения, содержащие несколько потенциальных ключей, то необходимо проверить, имеются ли функциональные зависимости, детерминанты которых не являются потенциальными ключами. Если такие функциональные зависимости имеются, то необходимо провести дальнейшую декомпозицию отношений. Те атрибуты, которые зависят от детерминантов, не являющихся потенциальными ключами выносятся в отдельное отношение вместе с детерминантами.
- *Шаг 5 (Приведение к 4НФ)*. Если в отношениях обнаружены нетривиальные многозначные зависимости, то необходимо провести декомпозицию для исключения таких зависимостей.
- *Шаг 5 (Приведение к 5НФ)*. Если в отношениях обнаружены нетривиальные зависимости соединения, то необходимо провести декомпозицию для исключения и таких зависимостей.

Реляционная алгебра

Обзор реляционной алгебры

- Третья часть реляционной модели, манипуляционная часть, утверждает, что доступ к реляционным данным осуществляется при помощи реляционной алгебры или эквивалентного ему реляционного исчисления.
- В реализациях конкретных реляционных СУБД сейчас не используется в чистом виде ни реляционная алгебра, ни реляционное исчисление. Фактическим стандартом доступа к реляционным данным стал язык SQL (Structured Query Language). Язык SQL представляет собой смесь операторов реляционной алгебры и выражений реляционного исчисления, использующий синтаксис, близкий к фразам английского языка и расширенный дополнительными возможностями, отсутствующими в реляционной алгебре и реляционном исчислении. Вообще, язык доступа к данным называется ***реляционно-полным***, если он по выразительной силе не уступает реляционной алгебре (или, что то же самое, реляционному исчислению), т.е. любой оператор реляционной алгебры может быть выражен средствами этого языка. Именно таким и является язык SQL.

Реляционная алгебра

- **Замкнутость реляционной алгебры**
- Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата. Таким образом, реляционный оператор выглядит как функция с отношениями в качестве аргументов:

  $R = f(R_1, R_2, \ldots, R_n)$
- Реляционная алгебра является замкнутой, т.к. в качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу:

  $R = f(f_1(R_{11}, R_{12}, \ldots), f_2(R_{21}, R_{22}, \ldots), \ldots)$
- Таким образом, в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры.

Реляционная алгебра

- Каждое отношение обязано иметь уникальное имя в пределах базы данных. Имя отношения, полученного в результате выполнения реляционной операции, определяется в левой части равенства. Однако можно не требовать наличия имен от отношений, полученных в результате реляционных выражений, если эти отношения подставляются в качестве аргументов в другие реляционные выражения. Такие отношения будем называть ***неименованными отношениями***. Неименованные отношения реально не существуют в базе данных, а только вычисляются в момент вычисления значения реляционного оператора.

Реляционная алгебра

- Традиционно, вслед за Коддом, определяют восемь реляционных операторов, объединенных в две группы.
- Теоретико-множественные операторы:
  - Объединение
  - Пересечение
  - Вычитание
  - Декартово произведение
- Специальные реляционные операторы:
  - Выборка
  - Проекция
  - Соединение
  - Деление
- Не все они являются независимыми, т.е. некоторые из этих операторов могут быть выражены через другие реляционные операторы.

Реляционная алгебра

- **Отношения, совместимые по типу**
- Некоторые реляционные операторы (например, объединение) требуют, чтобы отношения имели одинаковые заголовки. Действительно, отношения состоят из заголовка и тела. Операция объединения двух отношений есть просто объединение двух множеств кортежей, взятых из тел соответствующих отношений. Но будет ли результат отношением? Во-первых, если исходные отношения имеют разное количество атрибутов, то, очевидно, что множество, являющееся объединением таких разнотипных кортежей нельзя представить в виде отношения. Во-вторых, пусть даже отношения имеют одинаковое количество атрибутов, но атрибуты имеют различные наименования. Как тогда определить заголовок отношения, полученного в результате объединения множеств кортежей? В-третьих, пусть отношения имеют одинаковое количество атрибутов, атрибуты имеют одинаковые наименования, но определенны на различных доменах. Тогда снова объединение кортежей не будет образовывать отношение.

Реляционная алгебра

- *Определение 1*. Будем называть отношения ***совместимыми по типу***, если они имеют идентичные заголовки, а именно:

  - Отношения имеют *одно и то же множество имен атрибутов*, т.е. для любого атрибута в одном отношении найдется атрибут с таким же наименованием в другом отношении;

  - Атрибуты с одинаковыми именами *определены на одних и тех же доменах*.
- Некоторые отношения не являются совместимыми по типу, но становятся таковыми после некоторого переименования атрибутов. Для того чтобы такие отношения можно было использовать в реляционных операторах, вводится вспомогательный ***оператор переименования атрибутов***.

Реляционная алгебра

- **Оператор переименования атрибутов**
- Оператор переименования атрибутов имеет следующий синтаксис:

  R RENAME $Atr_1, Atr_2, \ldots$ AS $NewAtr_1, NewAtr_2, \ldots$
- где

  R - отношение,

  Atr_1, Atr_2 - исходные имена атрибутов,

  $NewAtr_1, NewAtr_2$ - новые имена атрибутов.
- В результате применения оператора переименования атрибутов получаем новое отношение, с измененными именами атрибутов.
- **Пример 1**.
- Следующий оператор возвращает неименованное отношение, в котором атрибут *City_Num* переименован в *CityId*:
- *City* RENAME *City_Num* AS *CityId*

Теоретико-множественные операторы

- **Объединение** /UNION/
- *Определение 2*. ***Объединением*** двух совместимых по типу отношений *A* и *B* называется отношение с тем же заголовком, что и у отношений *A* и *B*, и телом, состоящим из кортежей, принадлежащих или *A*, или *B*, или обоим отношениям.
- Синтаксис операции объединения:

  A **UNION** *B*
- Замечание. Объединение, как и любое отношение, не может содержать одинаковых кортежей. Поэтому, если некоторый кортеж входит и в отношение *A*, и отношение *B*, то в объединение он входит один раз.

Теоретико-множественные операторы

- **<u>Пример 2</u>**. Пусть даны два отношения *A* и *B* с информацией о сотрудниках:

Табельный номер	**Фамилия**	**Зарплата**
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

Отношение A

Табельный номер	**Фамилия**	**Зарплата**
1	Иванов	1000
2	Пушников	2500
4	Сидоров	3000

Отношение B

Объединение отношений *A* и *B* будет иметь вид:

Табельный номер	**Фамилия**	**Зарплата**
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000
2	Пушников	2500
4	Сидоров	3000

Отношение A UNION B

Теоретико-множественные операторы

- Замечание. Как видно из приведенного примера, потенциальные ключи, которые были в отношениях *A* и *B не наследуются* объединением этих отношений. Поэтому, в объединении отношений *A* и *B* атрибут "Табельный номер" может содержать дубликаты значений. Если бы это было не так, и ключи наследовались бы, то это противоречило бы понятию объединения как "объединение множеств". Конечно, объединение отношений *A* и *B* имеет, как и любое отношение, потенциальный ключ, например, состоящий из всех атрибутов.

Теоретико-множественные операторы

- **Пересечение** /INTERSECT/
- *Определение 3*. ***Пересечением*** двух совместимых по типу отношений *A* и *B* называется отношение с тем же заголовком, что и у отношений *A* и *B*, и телом, состоящим из кортежей, принадлежащих одновременно обоим отношениям *A* и *B*.
- Синтаксис операции пересечения:

  A **INTERSECT** *B*

- **Пример 3**. Для тех же отношений *A* и *B*, что и в предыдущем примере пересечение имеет вид:

Табельный номер	Фамилия	Зарплата
1	Иванов	1000

Отношение A INTERSECT B

Теоретико-множественные операторы

- Замечание. Казалось бы, что в отличие от операции объединения, потенциальные ключи могли бы наследоваться пересечением отношений. Однако это не так. Вообще, *никакие реляционные операторы не передают результатирующему отношению никаких данных о потенциальных ключах*. В качестве причины этого можно было бы привести тривиальное соображение, что так получается более просто и симметрично - все операторы устроены одинаково. На самом деле причина более глубока, и заключается в том, что потенциальный ключ - семантическое понятие, отражающее различимость объектов предметной области. Наличие потенциальных ключей *не выводится* из структуры отношения, а явно задается для каждого отношения, исходя из его смысла. Реляционные же операторы являются *формальными* операциями над отношениями и выполняются одинаково, независимо от смысла данных, содержащихся в отношениях. Поэтому, реляционные операторы ничего не могут "знать" о смысле данных. Трактовка результата реляционных операций - дело пользователя.

Теоретико-множественные операторы

- **Вычитание** /MINUS/, /SET DIFFERENCE/
- *Определение 4*. ***Вычитанием*** двух совместимых по типу отношений *A* и *B* называется отношение с тем же заголовком, что и у отношений *A* и *B*, и телом, состоящим из кортежей, принадлежащих отношению *A* и не принадлежащих отношению *B*.
- Синтаксис операции вычитания:

  A **MINUS** *B*

- **Пример 4**. Для тех же отношений *A* и *B*, что и в предыдущем примере вычитание имеет вид:

Табельный номер	**Фамилия**	**Зарплата**
2	Петров	2000
3	Сидоров	3000

Отношение A MINUS B

Теоретико-множественные операторы

- **Декартово произведение** /TIMES/, /CARTESIAN PRODUCT/
- *Определение 5*. ***Декартовым произведением*** двух отношений $A(A_1,A_2,\ldots,A_n)$ и $B(B_1,B_2,\ldots,B_m)$ называется отношение, заголовок которого является ***сцеплением заголовков*** отношений *A* и *B*:

  $(A_1,A_2,\ldots,A_n,B_1,B_2,\ldots,B_m)$,

  а тело состоит из кортежей, являющихся ***сцеплением кортежей*** отношений *A* и *B*:

  $(a_1,a_2,\ldots,a_n,b_1,b_2,\ldots,b_m)$,

  таких, что $(a_1,a_2,\ldots,a_n)\in A$, $(b_1,b_2,\ldots,b_m)\in B$.
- Синтаксис операции декартового произведения:

  A **TIMES** *B*

Теоретико-множественные операторы

- Замечание. Мощность произведения *A* **TIMES** *B* равна произведению мощностей отношений *A* и *B*, т.к. каждый кортеж отношения *A* соединяется с каждым кортежем отношения *B*.
- Замечание. Если в отношениях *A* и *B* имеются атрибуты с одинаковыми наименованиями, то перед выполнением операции декартового произведения такие атрибуты необходимо переименовать.
- Замечание. Перемножать можно любые два отношения, совместимость по типу при этом не требуется.

Теоретико-множественные операторы

- **<u>Пример 5</u>**. Пусть даны два отношения *A* и *B* с информацией о поставщиках и деталях:

Номер поставщика	**Наименование поставщика**
1	Иванов
2	Петров
3	Сидоров

Отношение A (Поставщики)

Номер детали	**Наименование детали**
1	Болт
2	Гайка
3	Винт

Отношение B (Детали)

Декартово произведение отношений *A* и *B* будет иметь вид:

Номер поставщика	**Наименование поставщика**	**Номер детали**	**Наименование детали**
1	Иванов	1	Болт
1	Иванов	2	Гайка
1	Иванов	3	Винт
2	Петров	1	Болт
2	Петров	2	Гайка
2	Петров	3	Винт
3	Сидоров	1	Болт
3	Сидоров	2	Гайка
3	Сидоров	3	Винт

Отношение A TIMES B

Теоретико-множественные операторы

- Замечание. Сама по себе операция декартового произведения не очень важна, т.к. она не дает никакой новой информации, по сравнению с исходными отношениями. Для реальных запросов эта операция почти никогда не используется. Однако операция декартового произведения важна для выполнения специальных реляционных операций, о которых речь пойдет ниже.

Специальные реляционные операторы

- **Выборка (ограничение, селекция)** /WHERE/, /SELECT/
- *Определение 6*. ***Выборкой (ограничением, селекцией)*** на отношении *A* с условием *c* называется отношение с тем же заголовком, что и у отношения *A*, и телом, состоящем из кортежей, значения атрибутов которых при подстановке в условие *c* дают значение ИСТИНА. *c* представляет собой логическое выражение, в которое могут входить атрибуты отношения *A* и (или) скалярные выражения.
- В простейшем случае условие *c* имеет вид $X\theta Y$, где θ - один из операторов сравнения (=,≠,>,≥,<,≤ и т.д.), а *X* и *Y* - атрибуты отношения *A* или скалярные значения. Такие выборки называются θ -***выборки*** (***тэта-выборки***) или θ -***ограничения***, θ -***селекции***.
- Синтаксис операции выборки:

  A **WHERE** *c*,

  или

  A **WHERE** $X\theta Y$

Специальные реляционные операторы

- **Пример 6**. Пусть дано отношение *A* с информацией о сотрудниках:

Табельный номер	**Фамилия**	**Зарплата**
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

Отношение A

Результат выборки *A* **WHERE** Зарплата < 3000 будет иметь вид:

Табельный номер	**Фамилия**	**Зарплата**
1	Иванов	1000
2	Петров	2000

Отношение A WHERE Зарплата<3000

Смысл операции выборки очевиден - выбрать кортежи отношения, удовлетворяющие некоторому условию. Таким образом, операция выборки дает "*горизонтальный срез*" отношения по некоторому условию.

Специальные реляционные операторы

- **Проекция** /PROJECT/
- *Определение 7*. ***Проекцией*** отношения *A* по атрибутам *X,Y,…,Z*, где каждый из атрибутов принадлежит отношению *A*, называется отношение с заголовком *(X,Y,…,Z)* и телом, содержащим множество кортежей вида *(x,y,…,z)*, таких, для которых в отношении *A* найдутся кортежи со значением атрибута *X* равным *x*, значением атрибута *Y* равным *y*, …, значением атрибута *Z* равным *z*.
- Синтаксис операции проекции:

  A[*X,Y,…,Z*]
- Замечание. Операция проекции дает "*вертикальный срез*" отношения, в котором удалены все возникшие при таком срезе дубликаты кортежей.

Специальные реляционные операторы

- **Пример 7**. Пусть дано отношение *A* с информацией о поставщиках, включающих наименование и месторасположение:

Номер поставщика	**Наименование поставщика**	**Город поставщика**
1	Иванов	Уфа
2	Петров	Москва
3	Сидоров	Москва
4	Сидоров	Челябинск

Отношение A (Поставщики)

Проекция *A*[Город поставщика] будет иметь вид:

Город поставщика
Уфа
Москва
Челябинск

Отношение A[Город поставщика]

Специальные реляционные операторы

- **Соединение**
- Операция соединения отношений, наряду с операциями выборки и проекции, является одной из наиболее важных реляционных операций.
- Обычно рассматривается несколько разновидностей операции соединения:

  - Общая операция соединения;

  - θ -соединение (тэта-соединение);

  - Экви-соединение;

  - Естественное соединение.
- Наиболее важным из этих частных случаев является операция естественного соединения. Все разновидности соединения являются частными случаями общей операции соединения.

Специальные реляционные операторы

- **Общая операция соединения**
- *Определение 8*. ***Соединением*** отношений *A* и *B* по условию *c* называется отношение

  (*A* **TIMES** *B*) **WHERE** *c*
- *c* представляет собой логическое выражение, в которое могут входить атрибуты отношений *A* и *B* и (или) скалярные выражения.
- Таким образом, операция соединения есть результат последовательного применения операций декартового произведения и выборки. Если в отношениях *A* и *B* имеются атрибуты с одинаковыми наименованиями, то перед выполнением соединения такие атрибуты необходимо переименовать.

Специальные реляционные операторы

- **Тэта-соединение**
- *Определение 9*. Пусть отношение *A* содержит атрибут *X*, отношение *B* содержит атрибут *Y*, а θ - один из операторов сравнения (=,≠,>, ≥,<,≤ и т.д.). Тогда θ-***соединением*** отношения *A* по атрибуту *X* с отношением *B* по атрибуту *Y* называют отношение

  (*A* **TIMES** *B*) **WHERE** *X*θ*Y*

- Это частный случай операции общего соединения.
- Иногда, для операции θ-соединения применяют следующий, более короткий синтаксис:

  A **[***X*θ*Y***]** *B*

Специальные реляционные операторы

- **Пример 8**. Рассмотрим некоторую компанию, в которой хранятся данные о поставщиках и поставляемых деталях. Пусть поставщикам и деталям присвоен некий статус. Пусть бизнес компании организован таким образом, что поставщики имеют право поставлять только те детали, статус которых не выше статуса поставщика (смысл этого может быть в том, что хороший поставщик с высоким статусом может поставлять больше разновидностей деталей, а плохой поставщик с низким статусом может поставлять только ограниченный список деталей, важность которых (статус детали) не очень высока).

Специальные реляционные операторы

Номер поставщика	**Наименование поставщика**	**X (Статус поставщика)**
1	Иванов	4
2	Петров	1
3	Сидоров	2

Отношение A (Поставщики)

Номер детали	**Наименование детали**	**Y (Статус детали)**
1	Болт	3
2	Гайка	2
3	Винт	1

Отношение B (Детали)

- Ответ на вопрос “Какие поставщики имеют право поставлять какие детали?" дает θ-соединение $A\ [X \geq Y]\ B$:

Номер поставщика	**Наименование поставщика**	**X (Статус поставщика)**	**Номер детали**	**Наименование детали**	**Y (Статус детали)**
1	Иванов	4	1	Болт	3
1	Иванов	4	2	Гайка	2
1	Иванов	4	3	Винт	1
2	Петров	1	3	Винт	1
3	Сидоров	2	2	Гайка	2
3	Сидоров	2	3	Винт	1

Отношение "Какие поставщики поставляют какие детали"

Специальные реляционные операторы

- **Экви-соединение**
- Наиболее важным частным случаем θ-соединения является случай, когда θ есть просто равенство.
- Синтаксис ***экви-соединения***:

  A **[***X*=*Y***]** *B*

Специальные реляционные операторы

- **Пример 9**. Пусть имеются отношения *P*, *D* и *PD*, хранящие информацию о поставщиках, деталях и поставках соответственно (для удобства введем краткие наименования атрибутов):

Номер поставщика PNUM	**Наименование поставщика PNAME**
1	Иванов
2	Петров
3	Сидоров

Отношение P (Поставщики)

Номер детали DNUM	**Наименование детали DNAME**
1	Болт
2	Гайка
3	Винт

Отношение D (Детали)

Номер поставщика PNUM	***Номер детали DNUM***	**Поставляемое количество VOLUME**
1	*1*	100
1	*2*	200
1	*3*	300
2	*1*	150
2	*2*	250
3	*1*	1000

Отношение PD (Поставки)

Специальные реляционные операторы

- Ответ на вопрос, какие детали поставляются поставщиками, дает экви-соединение *P* [*PNUM=PNUM*] *PD*. На самом деле, т.к. в отношениях имеются одинаковые атрибуты, то требуется сначала переименовать атрибуты, а потом выполнить экви-соединение. Запись становится более громоздкой:
- *(P* RENAME *PNUM* AS *PNUM1)* [*PNUM1=PNUM2*] (*PD* RENAME *PNUM* AS *PNUM2)*
- Обычно, такой сложной формой записи не пользуются. Но как бы то ни было, в результате имеем отношение:

Специальные реляционные операторы

Номер поставщика PNUM1	**Наименование поставщика PNAME**	**Номер поставщика PNUM2**	**Номер детали DNUM**	**Поставляемое количество VOLUME**
1	Иванов	1	1	100
1	Иванов	1	2	200
1	Иванов	1	3	300
2	Петров	2	1	150
2	Петров	2	2	250
3	Сидоров	3	1	1000

Отношение "Какие детали поставляются какими поставщиками?"

Специальные реляционные операторы

- Недостатком экви-соединения является то, что если соединение происходит по атрибутам с одинаковыми наименованиями (а так чаще всего и происходит!), то в результирующем отношении появляется два атрибута с одинаковыми значениями. В нашем примере атрибуты PNUM1 и PNUM2 содержат дублирующие данные. Избавиться от этого недостатка можно, взяв проекцию по всем атрибутам, кроме одного из дублирующих. Именно так действует естественное соединение.

Специальные реляционные операторы

- **Естественное соединение** /JOIN/
- *Определение 10*. Пусть даны отношения $A(A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_p)$ и $B(X_1,X_2,\ldots,X_p,B_1,B_2,\ldots,B_m)$, имеющие одинаковые атрибуты $X_1,X_2,\ldots,X_p$ (т.е. атрибуты с одинаковыми именами и определенные на одинаковых доменах).
- Тогда ***естественным соединением*** отношений A и B называется отношение с заголовком $(A_1,A_2,\ldots,A_n,X_1,X_2,\ldots,X_p,B_1,B_2,\ldots,B_m)$ и телом, содержащим множество кортежей $(a_1,a_2,\ldots,a_n,x_1,x_2,\ldots,x_p,b_1,b_2,\ldots,b_m)$, таких, что $(a_1,a_2,\ldots,a_n,x_1,x_2,\ldots,x_p) \in A$ и $(x_1,x_2,\ldots,x_p,b_1,b_2,\ldots,b_m) \in B$.

- Естественное соединение настолько важно, что для него используют специальный синтаксис:

  A **JOIN** B

Специальные реляционные операторы

- Замечание. В синтаксисе естественного соединения не указываются, по каким атрибутам производится соединение. Естественное соединение производится *по всем* одинаковым атрибутам.
- Замечание. Естественное соединение эквивалентно следующей последовательности реляционных операций:

  1. Переименовать одинаковые атрибуты в отношениях;
  2. Выполнить декартово произведение отношений;
  3. Выполнить выборку по совпадающим значениям атрибутов, имевших одинаковые имена;
  4. Выполнить проекцию, удалив повторяющиеся атрибуты;
  5. Переименовать атрибуты, вернув им первоначальные имена.

Специальные реляционные операторы

- Замечание. Можно выполнять последовательное естественное соединение нескольких отношений. Нетрудно проверить, что естественное соединение (как, впрочем, и соединение общего вида) обладает свойством ***ассоциативности***, т.е.

  (*A* JOIN *B*) JOIN *C* = *A* JOIN (*B* JOIN *C*)

  поэтому такие соединения можно записывать, опуская скобки:

  A JOIN *B* JOIN *C*

- **Пример 10**. В предыдущем примере ответ на вопрос «Какие детали поставляются поставщиками?", более просто записывается в виде естественного соединения трех отношений *P* JOIN *PD* JOIN *D* (для удобства просмотра порядок атрибутов изменен, это является допустимым по свойствам отношений):

Специальные реляционные операторы

Номер поставщика PNUM	**Наименование поставщика PNAME**	**Номер детали DNUM**	**Наименование детали DNAME**	**Поставляемое количество VOLUME**
1	Иванов	1	Болт	100
1	Иванов	2	Гайка	200
1	Иванов	3	Винт	300
2	Петров	1	Болт	150
2	Петров	2	Гайка	250
3	Сидоров	1	Болт	1000

Отношение P JOIN PD JOIN D

Специальные реляционные операторы

- **Деление** /DEVIDBY/, /DIVISION/
- *Определение 11*. Пусть даны отношения $A(X_1,X_2,\ldots,X_n,Y_1,Y_2,\ldots,Y_m)$ и $B(Y_1,Y_2,\ldots,Y_m)$, причем атрибуты $Y_1,Y_2,\ldots,Y_m$ - общие для двух отношений. ***Делением отношений*** *A* на *B* называется отношение с заголовком $(X_1,X_2,\ldots,X_n)$ и телом, содержащим множество кортежей $(x_1,x_2,\ldots,x_n)$, таких, что для *всех* кортежей $(y_1,y_2,\ldots,y_m)\in B$ в отношении *A* найдется кортеж $(x_1,x_2,\ldots,x_n,y_1,y_2,\ldots,y_m)$.

- Отношение *A* выступает в роли ***делимого***, отношение *B* выступает в роли ***делителя***. Деление отношений аналогично делению чисел с остатком.
- Синтаксис операции деления:
- *A* **DEVIDBY** *B*

Специальные реляционные операторы

- Замечание. Типичные запросы, реализуемые с помощью операции деления, обычно в своей формулировке имеют слово "все" – «Какие поставщики поставляют *все* детали?".

- **Пример 11**. В примере с поставщиками, деталями и поставками ответим на вопрос, «Какие поставщики поставляют *все* детали?".
- В качестве *делимого* возьмем проекцию *X* = *PD* [*PNUM,DNUM*], содержащую номера поставщиков и номера поставляемых ими деталей.

- В качестве *делителя* возьмем проекцию *Y* = *D* [*DNUM*], содержащую список номеров *всех* деталей (не обязательно поставляемых кем-либо).

- Деление *X* DEVIDBY *Y* дает список номеров поставщиков, поставляющих *все* детали.

Специальные реляционные операторы

- Оказалось, что только поставщик с номером 1 поставляет все детали.

Номер поставщика PNUM	**Номер детали DNUM**
1	1
1	2
1	3
2	1
2	2
3	1

Проекция X=PD[PNUM,DNUM]

Номер детали DNUM
1
2
3

Проекция Y=D[DNUM]

Номер поставщика PNUM
1

Отношение X DEVIDEBY Y

Примеры использования реляционных операторов

- **Пример 12**. Получить имена поставщиков, поставляющих деталь номер 2.
- Решение:

  ((*PD* JOIN *P*) WHERE *DNUM*=2) [*PNAME*]

- **Пример 13**. Получить имена поставщиков, поставляющих по крайней мере одну гайку.
- Решение:

  (((*D* WHERE *DNAME=Гайка*) JOIN *PD*) JOIN *P*) [*PNAME*]
- Ответ на этот запрос можно получить и иначе:

  (((*D* JOIN *PD*) JOIN *P*) WHERE *DNAME=Гайка*) [*PNAME*]

Примеры использования реляционных операторов

- **Пример 14**. Получить имена поставщиков, поставляющих все детали.
- Решение:

  ((*PD* [*PNUM,DNUM*] DEVIDBY *D* [*DNUM*]) JOIN *P*) [*PNAME*]

- **Пример 15**. Получить имена поставщиков, не поставляющих деталь номер 2.
- Решение:

  ((*P* [*PNUM*] MINUS ((*P* JOIN *PD*) WHERE *DNUM*=2) [*PNUM*]) JOIN *P*) [*PNAME*]

Примеры использования реляционных операторов

- Ответ на этот запрос можно получить и пошагово:

  T_1 = *P* [*PNUM*] - получить список номеров всех поставщиков;

  T_2 = *P* JOIN *PD* - соединить данные о поставщиках и поставках;

  T_3 = T_2 WHERE *DNUM*=2- в данных о поставщиках и поставках оставить только данные о поставках детали номер 2;

  T_4 = T_3 [*PNUM*] - получить список номеров поставщиков, поставляющих деталь номер 2;

  T_5 = T_1 MINUS T_4- получить список номеров поставщиков, не поставляющих деталь номер 2.

  T_6 = T_5 JOIN *P*- соединить список номеров поставщиков, не поставляющих деталь номер 2 с данными о поставщиках (получатся полные данные о поставщиках, не поставляющих деталь номер 2);

  T_7 = T_6 [*PNAME*] - искомый ответ (имена поставщиков, не поставляющих деталь номер 2).

Зависимые реляционные операторы

- Как было сказано в начале главы, не все операторы реляционной алгебры являются независимыми - некоторые из них выражаются через другие реляционные операторы.

- **Оператор соединения**
- Оператор соединения определяется через операторы декартового произведения и выборки. Для оператора естественного соединения добавляется оператор проекции.

- **Оператор пересечения**
- Оператор пересечения выражается через вычитание следующим образом:

  A INTERSECT *B* = *A* MINUS (*A* MINUS *B*)

Зависимые реляционные операторы

- **Оператор деления**
- Оператор деления выражается через операторы вычитания, декартового произведения и проекции следующим образом:

  A DEVIDEBY *B* = *A*[*X*] MINUS ((*A*[*X*] TIMES *B*) MINUS *A*)[*X*]

- Таким образом, показано, что операторы *соединения*, *пересечения* и *деления* можно выразить через другие реляционные операторы, т.е. эти операторы не являются примитивными.

Примитивные реляционные операторы

- Оставшиеся реляционные операторы (*объединение*, *вычитание*, *декартово произведение*, *выборка*, *проекция*) являются ***примитивными операторами*** - их нельзя выразить друг через друга.

- **Оператор декартового произведения**
- Оператор декартового произведения - это единственный оператор, *увеличивающий количество атрибутов*, поэтому его нельзя выразить через объединение, вычитание, выборку, проекцию.

- **Оператор проекции**
- Оператор проекции - единственный оператор, уменьшающий количество атрибутов, поэтому его нельзя выразить через объединение, вычитание, декартово произведение, выборку.

Примитивные реляционные операторы

- **Оператор выборки**
- Оператор выборки - единственный оператор, позволяющий проводить сравнения по атрибутам отношения, поэтому его нельзя выразить через объединение, вычитание, декартово произведение, проекцию.

- **Операторы объединения и вычитания**
- Доказательство примитивности операторов объединения и вычитания более сложны и мы их здесь не приводим.

Язык SQL

- В предыдущих разделах мы рассмотрели "штатные" средства манипулирования данными, поддерживаемые реляционной моделью - реляционная алгебра и реляционное исчисление. Однако, на практике крайне редко одно из этих средств принимается в качестве полной основы какого-либо языка базы данных. Так и SQL (Structured Query Language - структурированный язык запросов) основывается на некоторой смеси алгебраических и логических конструкций.
- Язык SQL (эта аббревиатура должна произноситься как "сикуель", однако все чаще говорят "эс-ку-эль") в настоящее время является промышленным стандартом, который в большей или меньшей степени поддерживает любая СУБД, претендующая на звание "реляционной".

Язык SQL

- *Из истории SQL:*
- В начале 70-х годов в компании IBM была разработана экспериментальная СУБД System R на основе языка SEQUEL (Structured English Qeury Language - структурированный английский язык запросов), который можно считать непосредственным предшественником SQL. Целью разработки было создание простого ***непроцедурного*** языка, которым мог воспользоваться любой пользователь, даже не имеющий навыков программирования. В 1981 году IBM объявила о своем первом, основанном на SQL программном продукте, SQL/DS. Чуть позже к ней присоединились Oracle и другие производители. Первый стандарт языка SQL был принят Американским национальным институтом стандартизации (ANSI) в 1987 (так называемый SQL level /уровень/ 1) и несколько уточнен в 1989 году (SQL level 2). Дальнейшее развитие языка поставщиками СУБД потребовало принятия в 1992 нового расширенного стандарта (ANSI SQL-92 или просто SQL-2).

Язык SQL

- *Из истории SQL:*
- Попытки совместить средства манипулирования данными реляционной модели и способы описания внешнего мира объектно-ориентированной модели получили развитие в языке SQL-3.
- В 1999 году заговорили об очередном (третьем по счету) стандарте SQL и было опубликовано пять частей стандарта SQL-3 (SQL:99): Однако многие специалисты вновь скептически отнеслись SQL-3, обвинив его в незавершенности. Поэтому в 2003 году к вопросу модернизации SQL вернулись вновь. В обновленный стандарт с необходимыми изменениями вошли все части прежнего SQL:99. В дополнение к перечисленным частям SQL:2003 приобрел еще несколько документов.

Язык SQL

- *Из истории SQL:*
- Совершенствование SQL отчасти подтверждает один из неписаных законов программирования — лучшее враг хорошего. С каждым очередным этапом развития стандарта все меньше и меньше производителей программного обеспечения могут его поддерживать в полном объеме. У Криса Дейта на этот счет есть хорошее высказывание: "… в наши дни ни один программный продукт не поддерживает полностью даже SQL:92; вместо этого такие продукты, как правило, поддерживают то, что можно было бы назвать "надмножеством подмножества" стандарта…".
- Как следствие стандарт не поспевает за производителями, а это неминуемо ведет к появлению различных ветвей языка, что с каждым годом все более и более уменьшает вероятность появления новой, общепринятой редакции SQL, однозначно поддерживаемой всеми разработчиками ПО. Это затрудняет переносимость приложений, разработанных для одних СУБД в другие СУБД.

Язык SQL

- Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например, вместо "отношений" используются "таблицы", вместо "кортежей" - "строки", вместо "атрибутов" - "колонки" или "столбцы".
- Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит он нее. Например, отношение в реляционной модели данных не допускает наличия одинаковых кортежей, а таблицы в терминологии SQL могут иметь одинаковые строки. Имеются и другие отличия.
- Язык SQL является реляционно полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором SQL.

Язык SQL

- Необходимо сказать, что хотя SQL и задумывался как средство работы конечного пользователя, в конце концов он стал настолько сложным, что превратился в инструмент программиста. Вопросы создания приложений обработки данных с использованием SQL будут рассмотрены далее.
- В SQL определены два подмножества языка:

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

  - SQL-DML (Data Manipulation Language) - язык манипулирования данными: добавление, изменение, удаление и извлечение данных, управления транзакциями
- Некоторыми авторами вводятся дополнительные подмножества языка для отдельных задач.

Типы данных SQL

- Символьные типы данных - содержат буквы, цифры и специальные символы.
  - **CHAR** или **CHAR(n)** -символьные строки фиксированной длины. Длина строки определяется параметром **n**. **CHAR** без параметра соответсвует **CHAR(1)**. Для хранения таких данных всегда отводится **n** байт вне зависимости от реальной длины строки.
  - **VARCHAR(n)** - символьная строка переменной длины. Для хранения данных этого типа отводится число байт, соответствующее реальной длине строки.

Типы данных SQL

- Целые типы данных - поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значения столбца реляционной таблицы).
  - **INTEGER** или **INT**- целое, для хранения которого отводится, как правило, 4 байта. *(Замечание: число байт, отводимое для хранения того или иного числового типа данных зависит от используемой СУБД и аппаратной платформы, здесь приводятся наиболее "типичные" значения).* Интервал значений от - 2147483647 до + 2147483648
  - **SMALLINT** - короткое целое (2 байта), интервал значений от - 32767 до +32768

Типы данных SQL

- Вещественные типы данных - описывают числа с дробной частью.
  - **FLOAT** и **SMALLFLOAT** - числа с плавающей точкой (для хранения отводится обычно 8 и 4 байта соответсвенно).
  - **DECIMAL(p)** - тип данных аналогичный **FLOAT** с числом значащих цифр **p**.
  - **DECIMAL(p,n)** - аналогично предыдущему, **p** - общее количество десятичных цифр, **n** - количество цифр после десятичной запятой.
- Денежные типы данных - описывают, естественно, денежные величины. Если в ваша система такого типа данных не поддерживает, то используйте DECIMAL(p,n).
  - **MONEY(p,n)** - все аналогично типу **DECIMAL(p,n)**. Вводится только потому, что некоторые СУБД предусматривают для него специальные методы форматирования.

Типы данных SQL

- Дата и время - используются для хранения даты, времени и их комбинаций. Большинство СУБД умеет определять интервал между двумя датами, а также уменьшать или увеличивать дату на определенное количество времени.
  - **DATE** - тип данных для хранения даты.
  - **TIME** - тип данных для хранения времени.
  - **INTERVAL** - тип данных для хранения временного интервала.
  - **DATETIME** - тип данных для хранения моментов времени (год + месяц + день + часы + минуты + секунды + доли секунд).
- Двоичные типы данных - позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.). Определения этих типов наиболее сильно различаются от системы к системе, часто используются ключевые слова:
  - **BINARY**
  - **BYTE**
  - **BLOB**

Типы данных SQL

- Последовательные типы данных - используются для представления возрастающих числовых последовательностей.
  - **SERIAL** - тип данных на основе **INTEGER**, позволяющий сформировать уникальное значение (например, для первичного ключа). При добавлении записи СУБД автоматически присваивает полю данного типа значение, получаемое из возрастающей последовательности целых чисел.
- В заключение следует сказать, что для всех типов данных имеется общее значение **NULL** - "не определено". Это значение имеет каждый элемент столбца до тех пор, пока в него не будут введены данные. При создании таблицы можно явно указать СУБД могут ли элементы того или иного столбца иметь значения **NULL** (это не допустимо, например, для столбца, являющего первичным ключом).

DDL: Операторы создания схемы базы данных

- При описании команд предполагается, что:

  - текст, набранный строчными буквами (например, **CREATE TABLE**) является обязательным;

  - текст, набранный прописными буквами и заключенный в угловые скобки (например, **<имя_базы_данных>**) обозначает переменную, вводимую пользователем;

  - в квадратные скобки (например, **[NOT NULL]**) заключается необязательная часть команды;

  - взаимоисключающие элементы команды разделяются вертикальной чертой (например, **[UNIQUE | PRIMARY KEY]**).

DDL: Операторы создания схемы базы данных

Операторы базы данных

Команда	Описание
CREATE DATABASE <имя_базы_данных>	Создание базы данных.
DROP DATABASE <имя_базы_данных>	Удаление базы данных.

DDL: Операторы создания схемы базы данных

- **Создание таблицы:**

```
CREATE TABLE <имя_таблицы>
   (<имя_столбца> <тип_столбца>
              [NOT NULL]
                   [UNIQUE | PRIMARY KEY]
                   [REFERENCES <имя_мастер_таблицы> [<имя_столбца>]]
     , ...)
```

- Пользователь обязан указать имя таблицы и список столбцов. Для каждого столбца обязательно указываются его имя и тип, а также опционально могут быть указаны параметры:

  - NOT NULL - в этом случае элементы столбца всегда должны иметь определенное значение (не NULL);

  - один из взаимоисключающих параметров **UNIQUE** - значение каждого элемента столбца должно быть уникальным или **PRIMARY KEY** - столбец является первичным ключом;

DDL: Операторы создания схемы базы данных

- REFERNECES <имя_мастер_таблицы> [<имя_столбца>] - эта конструкция определяет, что данный столбец является внешним ключом и указывает на ключ какой мастер_таблицы он ссылается.

- Контроль за выполнением указанных условий осуществляет СУБД.

- **Удаление таблицы:**

  DROP TABLE <имя_таблицы>

DDL: Операторы создания схемы базы данных

- *Пример: создание базы данных* ***publications:***

```
CREATE DATABASE publications;
CREATE TABLE authors (au_id INT PRIMARY KEY,
                      author VARCHAR(25) NOT NULL);
CREATE TABLE publishers (pub_id INT PRIMARY KEY,
                     publisher VARCHAR(255) NOT NULL, url VARCHAR(255));
CREATE TABLE titles (title_id INT PRIMARY KEY,
                      title VARCHAR(255) NOT NULL,
                      yearpub INT,
                      pub_id INT REFERENCES publishers(pub_id));
CREATE TABLE titleautors (au_id INT REFERENCES authors(au_id),
                      title_id INT REFERENCES titles(title_id));
CREATE TABLE wwwsites (site_id INT PRIMARY KEY,
                      site VARCHAR(255) NOT NULL,
                      url VARCHAR(255));
CREATE TABLE wwwsiteauthors (au_id INT REFERENCES authors(au_id),
                      site_id INT REFERENCES wwwsites(site_id));
```

DDL: Операторы создания схемы базы данных

Модификация таблицы:

Добавить столбцы	**ALTER TABLE <имя_таблицы> ADD** **(<имя_столбца> <тип_столбца>** **[NOT NULL]** **[UNIQUE | PRIMARY KEY]** **[REFERENCES <имя_мастер_таблицы>** **[<имя_столбца>]]** **,...)**
Удалить столбцы	**ALTER TABLE <имя_таблицы> DROP (<имя_столбца>,...)**
Модификация типа столбцов	**ALTER TABLE <имя_таблицы> MODIFY** **(<имя_столбца> <тип_столбца>** **[NOT NULL]** **[UNIQUE | PRIMARY KEY]** **[REFERENCES <имя_мастер_таблицы>** **<имя_столбца>]]** **,...)**

DDL: Операторы создания индексов

- **Создание индекса:**
  CREATE [UNIQUE] INDEX <имя_индекса> ON <имя_таблицы>(<имя_столбца>,...)

- Эта команда создает индекс с заданным именем для таблицы <имя_таблицы> по столбцам, входящим в список, указанный в скобках. Создание индексов значительно ускоряет работу с таблицами. В случае указания необязательного параметра **UNIQUE** СУБД будет проверять каждое значение индекса на уникальность.
- Очень часто встает вопрос, какие поля необходимо индексировать. Обязательно надо строить индексы для первичных ключей, поскольку по их значениям осуществляется доступ к данным при операциях соединения двух и более таблиц. Также в ответе на этот вопрос поможет анализ наиболее частых запросов к базе данных. Например, для БД **publications** можно ожидать, что одним из наиболее частых запросов будет выборка всех публикаций данного автора.

DDL: Операторы создания индексов

- Для минимизации времени этого запроса необходимо построить индекс для таблицы **authors** по именам авторов:

  CREATE INDEX au_names ON authors (author);

- Создание индексов для первичных ключей:

  CREATE INDEX au_index ON authors (au_id);
  CREATE INDEX title_index ON titles (title_id);
  CREATE INDEX pub_index ON publishers (pub_id);
  CREATE INDEX site_index ON wwwsites (site_id);

- Первоначальное определение структуры индексов производится разработчиком на стадии создания прикладной системы. В дальнейшем она уточняется администратором системы по результатам анализа ее работы, учета наиболее часто выполняющихся запросов и т.д.

- **Удаление индекса:**

  DROP INDEX <имя_индекса>

DDL: Операторы управления правами доступа

- По соображениям безопасности не каждому пользователю прикладной системы может быть разрешено получать информацию из какой-либо таблицы, а тем более изменять в ней данные. Для определения прав пользователей относительно объектов базы данных (таблицы, представления, индексы) в **SQL** определена пара команд GRANT и REVOKE. Синтаксис операции передачи прав на таблицу:

GRANT <тип_права_на_таблицу>
ON <имя_таблицы> [<список_столбцов>]
TO <имя_пользователя>

DDL: Операторы управления правами доступа

- Права пользователя на уровне таблицы определяются следующими ключевыми словами (как мы увидим чуть позже эти ключевые слова совпадают с командами выборки и изменения данных):

  SELECT - получение информации из таблицы;

  UPDATE - изменение информации в таблице;

  INSERT - добавление записей в таблицу;

  DELETE - удаление записей из таблицы;

  INDEX - индексирование таблицы;

  ALTER - изменение схемы определения таблицы;

  ALL - все права.

DDL: Операторы управления правами доступа

- В поле <тип_права_на_таблицу> может быть указано либо ключевое слово ALL или любая комбинация других ключевых слов. Например, предоставим все права на таблицу **publishers** пользователю **andy**:

  GRANT ALL ON publishers TO andy;

  Пользователю **peter** предоставим права на извлечение и дбавление записей на эту же таблицу:

  GRANT SELECT INSERT ON publishers TO peter;

  В том случае, когда одинаковые права надо предоставить сразу всем пользователям, вместо выполнения команды GRANT для каждого из них, можно вместо имени пользователя указать ключевое слово PUBLIC:

  GRANT SELECT ON publishers TO PUBLIC;

DDL: Операторы управления правами доступа

- Отмена прав осуществляется командой REVOKE:

  REVOKE <тип_права_на_таблицу>

  ON <имя_таблицы> [<список_столбцов>]

  FROM <имя_пользователя>

  Все ключевые слова данной команды эквивалентны оператору GRANT.

DDL: Операторы управления правами доступа

- Большинство систем поддерживают также команду GRANT для назначения привилегий на базу данных в целом. В этом случае формат команды:

  GRANT <тип_права_на_базу_данных>

  ON <имя_базы данных>

  TO <имя_пользователя>

  К сожалению, способы задания прав на базу данных различны для разных СУБД, и точную их формулировку нужно уточнять в документации.
- Отмена прав на базу данных осуществляется командой:

  REVOKE <тип_права_на_базу_данных>

  FROM <имя_пользователя>

DML: Команды модификации данных

- К этой группе относятся операторы добавления, изменения и удаления записей.
- **Добавить новую запись в таблицу:**

  INSERT INTO <имя_таблицы> [(<имя_столбца>,<имя_столбца>,...)]

  VALUES (<значение>,<значение>,..)

  Список столбцов в данной команде не является обязательным параметром. В этом случае должны быть указаны значения для всех полей таблицы в том порядке, как эти столбцы были перечислены в команде CREATE TABLE, например:

```
INSERT INTO publishers
VALUES (16,"Microsoft Press", "http://www.microsoft.com");
```

- Пример с указанием списка столбцов:

```
INSERT INTO publishers (publisher,pub_id)
    VALUES ("Super Computer Publishing",17);
```

DML: Команды модификации данных

- **Модификация записей:**

  UPDATE <имя_таблицы>

  SET <имя_столбца>=<значение>,...

  [WHERE <условие>]

- Если задано ключевое слово WHERE и условие, то команда UPDATE применяется только к тем записям, для которых оно выполняется. Если условие не задано, UPDATE применяется ко всем записям. Пример:

```
UPDATE publishers
SET url="http://www.superpub.com"
WHERE pub_id=17;
```

DML: Команды модификации данных

В качестве условия используются логические выражения над константами и полями. В условиях допускаются:

- операции сравнения: > , < , >= , <= , = , <> , != . В SQL эти операции могут применяться не только к числовым значениям, но и к строкам ("<" означает раньше, а ">" позже в алфавитном порядке) и датам ("<" раньше и ">" позже в хронологическом порядке);
- оперции проверки поля на значение NULL: IS NULL, IS NOT NULL;
- операции проверки на вхождение в диапазон: BETWEEN и NOT BETWEEN;
- операции проверки на вхождение в список: IN и NOT IN;
- операции проверки на вхождение подстроки: LIKE и NOT LIKE;
- отдельные операции соединяются связями AND, OR, NOT и группируются с помощью скобок.

DML: Команды модификации данных

Подробно все эти ключевые слова будут описаны и проиллюстрированы в параграфе, посвященном оператору SELECT. Здесь мы ограничимся приведением несложного примера:

```
    UPDATE publishers
 SET url="url not defined"
 WHERE url IS NULL;
```

Эта команда находит в таблице **publishers** все неопределенные значения столбца **url** и заменяет их строкой "url not defined".

DML: Команды модификации данных

- **Удаление записей**

  DELETE FROM <имя_таблицы>

  [WHERE <условие>]

  Удаляются все записи, удовлетворяющие указанному условию. Если ключевое слово WHERE и условие отсутствуют, из таблицы удаляются все записи. Пример:

```
DELETE FROM publishers
WHERE publisher = "Super Computer Publishing";
```

  Эта команда удаляет запись об издательстве Super Computer Publishing.

DML: Выборка данных

- Для извлечения записей из таблиц в SQL определен оператор **SELECT**. С помощью этой команды осуществляется не только операция реляционной алгебры "выборка" (горизонтальное подмножество), но и предварительное соединение (join) двух и более таблиц. Это наиболее сложное и мощное средство SQL, полный синтаксис оператора **SELECT** имеет вид:

SELECT [ALL | DISTINCT] <список_выбора>
FROM <имя_таблицы>, ...
[WHERE <условие>]
[GROUP BY <имя_столбца>,...]
[HAVING <условие>]
[ORDER BY <имя_столбца> [ASC | DESC],...]

Порядок предложений в операторе **SELECT** должен строго соблюдаться (например, GROUP BY должно всегда предшествовать ORDER BY), иначе это приведет к появлению ошибок.

DML: Выборка данных

- Мы начнем рассмотрение **SELECT** с наиболее простых его форм. Все примеры, приведенные ниже, касающиеся базы данных publications, можно выполнить самостоятельно.

- Этот оператор всегда начинается с ключевого слова SELECT. В конструкции <список_выбора> определяется столбец или столбцы, включаемые в результат. Он может состоять из имен одного или нескольких столбцов, или из одного символа * (звездочка), определяющего все столбцы. Элементы списка разделяются запятыми.
- Пример: получить список всех авторов

```
SELECT author
FROM authors;
```

получить список всех полей таблицы **authors**:

```
SELECT *
FROM authors;
```

DML: Выборка данных

- В том случае, когда нас интересуют не все записи, а только те, которые удовлетворяют некому условию, это условие можно указать после ключевого слова WHERE. Например, найдем все книги, опубликованные после 1996 года:

```
SELECT title
FROM titles
WHERE yearpub > 1996;
```

Допустим теперь, что нам надо найти все публикации за интервал 1995 - 1997 гг. Это условие можно записать в виде:

```
SELECT title
FROM titles
WHERE yearpub>=1995 AND yearpub<=1997;
```

DML: Выборка данных

- Другой вариант этой команды можно получить с использованием логической операции проверки на вхождение в интервал:

```
SELECT title
FROM titles
WHERE yearpub BETWEEN 1995 AND 1997;
```

При использовании конструкции NOT BETWEEN находятся все строки, не входящие в указанный диапазон.

- Еще один вариант этой команды можно построить с помощью логической операции проверки на вхождение в список:

```
SELECT title
FROM titles
WHERE yearpub IN (1995,1996,1997);
```

Здесь мы задали в явном виде список интересующих нас значений. Конструкция NOT IN позволяет найти строки, не удовлетворяющие условиям, перечисленным в списке.

DML: Выборка данных

- Наиболее полно преимущества ключевого слова IN проявляются во вложенных запросах, также называемых подзапросами. Предположим, нам нужно найти все издания, выпущенные компанией "Oracle Press". Наименования издательских компаний содержатся в таблице **publishers**, названия книг в таблице **titles**. Ключевое слово NOT IN позволяет объединить обе таблицы (без получения общего отношения) и извлечь при этом нужную информацию:

```
 SELECT title
FROM titles
WHERE pub_id IN
        (SELECT pub_id
    FROM publishers
    WHERE publisher='Oracle Press');
```

DML: Выборка данных

- При выполнении этой команды СУБД вначале обрабатывает вложенный запрос по таблице **publishers**, а затем его результат передает на вход основного запроса по таблице **titles**.
- Некоторые задачи нельзя решить с использованием только операторов сравнения. Например, мы хоти найти web-site издательства "Wiley", но не знаем его точного наименования. Для решения этой задачи предназначено ключевое слово LIKE, его синтаксис имеет вид:

  WHERE <имя_столбца> LIKE <образец> [ESCAPE <ключевой_символ>]

  Образец заключается в кавычки и должен содержать шаблон подстроки для поиска. Обычно в шаблонах используются два символа:

  - % (знак процента) - заменяет любое количество символов;

  - _ (подчеркивание) - заменяет одиночный символ.

DML: Выборка данных

- Попробуем найти искомый web-site:

```
SELECT publiser, url
FROM publishers
WHERE publisher LIKE '%Wiley%';
```

В соответствии с шаблоном СУБД найдет все строки включающие в себя подстроку "Wiley".

Другой пример: найти все книги, название которых начинается со слова "SQL":

```
    SELECT title
FROM titles
WHERE title LIKE 'SQL%';
```

DML: Выборка данных

- В том случае, когда надо найти значение, которое само содержит один из символов шаблона, используют ключевое слово ESCAPE и <ключевой_символ>. Литерал, следующий в шаблоне после ключевого символа, рассматривается как обычный символ, все последующие символы имеют обычное значение. Например, нам надо найти ссылку на web-страницу, о которой известно, что в ее url содержится подстрока "my_works":

```
SELECT site, url
FROM wwwsites
WHERE url LIKE '%my@_works%' ESCAPE '@';
```

В заключение заметим, что при выполнении оператора SELECT результирующее отношение может иметь несколько записей с одинаковыми значениями всех полей. Чтобы исключить повторяющиеся записи из выборки используется ключевое слово DISTINCT. Ключевое слово ALL указывает, что в результат необходимо включать все строки.

DML: Выборка из нескольких таблиц

- Очень часто возникает ситуация, когда выборку данных надо производить из отношения, которое является результатом слияния (join) двух других отношений. Например, нам нужно получить из базы данных **publications** информацию о всех печатных изданиях в виде следующей таблицы:

название_книги	год_выпуска	издательство

- Для этого СУБД предварительно должна выполнить слияние таблиц **titles** и **publishers**, а только затем произвести выборку из полученного отношения.

DML: Выборка из нескольких таблиц

- Для выполнения операции такого рода в операторе SELECT после ключевого слова FROM указывается список таблиц, по которым производится поиск данных. После ключевого слова WHERE указывается условие, по которому производится слияние. Для того, чтобы выполнить данный запрос, нужно дать команду:

```
SELECT titles.title, titles.yearpub, publishers.publisher
FROM titles, publishers
WHERE titles.pub_id=publishers.pub_id;
```

А вот пример, где одновременно задаются условия и слияния, и выборки (результат предыдущего запроса ограничивается изданиями после 1996 года):

```
SELECT titles.title, titles.yearpub, publishers.publisher
FROM titles, publishers
WHERE titles.pub_id=publishers.pub_id AND titles.yearpub>1996;
```

DML: Выборка из нескольких таблиц

- Следует обратить внимание на то, что когда в разных таблицах присутствуют одноименные поля, то для устранения неоднозначности перед именем поля указывается имя таблицы и знак "." (точка). (Хорошее правило: имя таблицы указывать всегда!)
- Естественно, имеется возможность производить слияние и более чем двух таблиц. Например, чтобы дополнить описанную выше выборку именами авторов книг необходимо составить оператор следующего вида:

```
SELECT authors.author, titles.title, titles.yearpub, publishers.publisher
FROM titles, publishers, titleauthors
WHERE titleauthors.au_id=authors.au_id
  AND  titleauthors.title_id=titles.title_id
  AND  titles.pub_id=publishers.pub_id
  AND  titles.yearpub > 1996;
```

DML: Выборка из нескольких таблиц

- **Использование имен корреляции (алиасов, псевдонимов)**
- Иногда приходится выполнять запросы, в которых таблица соединяется сама с собой, или одна таблица соединяется дважды с другой таблицей. При этом используются ***имена корреляции*** (***алиасы***, ***псевдонимы***), которые позволяют различать соединяемые копии таблиц. Имена корреляции вводятся в разделе FROM и идут через пробел после имени таблицы. Имена корреляции должны использоваться в качестве префикса перед именем столбца и отделяются от имени столбца точкой. Если в запросе указываются одни и те же поля из разных экземпляров одной таблицы, они должны быть переименованы для устранения неоднозначности в именованиях колонок результирующей таблицы. Определение имени корреляции действует только во время выполнения запроса.

DML: Выборка из нескольких таблиц

- **Пример 19**. Отобрать все пары поставщиков таким образом, чтобы первый поставщик в паре имел статус, больший статуса второго поставщика:

```
SELECT
   P1.PNAME AS PNAME1,
   P1.PSTATUS AS PSTATUS1,
   P2.PNAME AS PNAME2,
   P2.PSTATUS AS PSTATUS2
 FROM
   P P1, P P2
 WHERE P1.PSTATUS1 > P2.PSTATUS2;
```

- В результате получим следующую таблицу:

PNAME1	PSTATUS1	PNAME2	PSTATUS2
Иванов	4	Петров	1
Иванов	4	Сидоров	2
Сидоров	2	Петров	1

DML: Выборка из нескольких таблиц

- **Синтаксис соединенных таблиц**
- В разделе FROM оператора SELECT можно использовать соединенные таблицы. Пусть в результате некоторых операций мы получаем таблицы A и B. Такими операциями могут быть, например, оператор SELECT или другая соединенная таблица. Тогда синтаксис соединенной таблицы имеет следующий вид:
- *Соединенная таблица* ::=
  Перекрестное соединение
  | *Естественное соединение*
  | *Соединение посредством предиката*
  | *Соединение посредством имен столбцов*
  | *Соединение объединения*
- *Тип соединения* ::=
  INNER
  | **LEFT** [**OUTER**]
  | **RIGTH** [**OUTER**]
  | **FULL** [**OUTER**]

DML: Выборка из нескольких таблиц

- *Перекрестное соединение* ::=
  Таблица A **CROSS JOIN** *Таблица B*
- *Естественное соединение* ::=
  Таблица A [**NATURAL**] [*Тип соединения*] **JOIN** *Таблица B*
- *Соединение посредством предиката* ::=
  Таблица A [*Тип соединения*] **JOIN** *Таблица B* **ON** Предикат
- *Соединение посредством имен столбцов* ::=
  Таблица A [*Тип соединения*] **JOIN** *Таблица B* **USING** (Имя столбца.,..)
- *Соединение объединения* ::=
  Таблица A **UNION JOIN** *Таблица B*

DML: Выборка из нескольких таблиц

- Опишем используемые термины.
- CROSS JOIN - Перекрестное соединение возвращает просто декартово произведение таблиц. Такое соединение в разделе FROM может быть заменено списком таблиц через запятую.
- NATURAL JOIN - Естественное соединение производится по всем столбцам таблиц A и B, имеющим одинаковые имена. В результирующую таблицу одинаковые столбцы вставляются только один раз.
- JOIN … ON - Соединение посредством предиката соединяет строки таблиц A и B посредством указанного предиката.
- JOIN … USING - Соединение посредством имен столбцов соединяет отношения подобно естественному соединению по тем общим столбцам таблиц A и Б, которые указаны в списке USING.
- OUTER - Ключевое слово OUTER (внешний) не является обязательными, оно не используется ни в каких операциях с данными.

DML: Выборка из нескольких таблиц

- INNER - Тип соединения "внутреннее". Внутренний тип соединения используется по умолчанию, когда тип явно не задан. В таблицах A и B соединяются только те строки, для которых найдено совпадение.
- LEFT (OUTER) - Тип соединения "левое (внешнее)". Левое соединение таблиц A и B включает в себя все строки из левой таблицы A и те строки из правой таблицы B, для которых обнаружено совпадение. Для строк из таблицы A, для которых не найдено соответствия в таблице B, в столбцы, извлекаемые из таблицы B, заносятся значения NULL.
- RIGHT (OUTER) - Тип соединения "правое (внешнее)". Правое соединение таблиц A и B включает в себя все строки из правой таблицы B и те строки из левой таблицы A, для которых обнаружено совпадение. Для строк из таблицы B, для которых не найдено соответствия в таблице A, в столбцы, извлекаемые из таблицы A заносятся значения NULL.

DML: Выборка из нескольких таблиц

- FULL (OUTER) - Тип соединения "полное (внешнее)". Это комбинация левого и правого соединений. В полное соединение включаются все строки из обеих таблиц. Для совпадающих строк поля заполняются реальными значениями, для несовпадающих строк поля заполняются в соответствии с правилами левого и правого соединений.
- UNION JOIN - Соединение объединения является обратным по отношению к внутреннему соединению. Оно включает только те строки из таблиц A и B, для которых не найдено совпадений. В них используются значения NULL для столбцов, полученных из другой таблицы. Если взять полное внешнее соединение и удалить из него строки, полученные в результате внутреннего соединения, то получится соединение объединения.

DML: Выборка из нескольких таблиц

- Использование соединенных таблиц часто облегчает восприятие оператора SELECT, особенно, когда используется естественное соединение. Если не использовать соединенные таблицы, то при выборе данных из нескольких таблиц необходимо явно указывать условия соединения в разделе WHERE. Если при этом пользователь указывает сложные критерии отбора строк, то в разделе WHERE смешиваются семантически различные понятия - как условия связи таблиц, так и условия отбора строк.
- Рассмотренный ранее пример может выглядеть так:

```
SELECT titles.title, titles.yearpub, publishers.publisher
    FROM titles JOIN publishers ON titles.pub_id=publishers.pub_id
WHERE  titles.yearpub>1996;
```

DML: Вычисления внутри SELECT

- SQL позволяет выполнять различные арифметические операции над столбцами результирующего отношения. В конструкции <список_выбора> можно использовать константы, функции и их комбинации с арифметическими операциями и скобками. Например, чтобы узнать сколько лет прошло с 1992 года (год принятия стандарта SQL-92) до публикации той или иной книги можно выполнить команду:

```
SELECT title, yearpub-1992
FROM titles
WHERE yearpub > 1992;
```

В арифметических вражения допускаются операции сложения (+), вычитания (-), деления (/), умножения (*), а также различные функции (COS, SIN, ABS - абсолютное значение и т.д.). Также в запрос можно добавить строковую константу:

```
SELECT 'the title of the book is', title, yearpub-1992
FROM titles
WHERE yearpub > 1992;
```

DML: Вычисления внутри SELECT

- В SQL также определены так называемые агрегатные функции, которые совершают действия над совокупностью одинаковых полей в группе записей. Среди них:

  - AVG(<имя поля>) - среднее по всем значениям данного поля;
  - COUNT(<имя поля>) или **COUNT (*)** - число записей;
  - MAX(<имя поля>) - максимальное из всех значений данного поля;
  - MIN(<имя поля>) - минимальное из всех значений данного поля;
  - SUM(<имя поля>) - сумма всех значений данного поля.

DML: Вычисления внутри SELECT

- Следует учитывать, что каждая агрегирующая функция возвращает единственное значение.
- Примеры: определить дату публикации самой "древней" книги в нашей базе данных

```
SELECT MIN(yearpub)
FROM titles;
```

подсчитать количество книг в нашей базе данных:

```
SELECT COUNT(*)
FROM titles;
```

- Область действия данных функции можно ограничить с помощью логического условия. Например, количество книг, в названии которых есть слово "SQL":

```
SELECT COUNT(*)
FROM titles
WHERE title LIKE '%SQL%';
```

DML: Группировка данных

- Группировка данных в операторе SELECT осуществляется с помощью ключевого слова GROUP BY и ключевого слова HAVING, с помощью которого задаются условия разбиения записей на группы.
- GROUP BY неразрывно связано с агрегирующими функциями, без них оно практически не используется. GROUP BY разделяет таблицу на группы, а агрегирующая функция вычисляет для каждой из них итоговое значение. Определим для примера количество книг каждого издательства в нашей базе данных:

```
SELECT publishers.publisher, count(titles.title)
FROM titles, publishers
WHERE titles.pub_id=publishers.pub_id
GROUP BY publisher;
```

DML: Группировка данных

- Ключевое слово HAVING работает следующим образом: сначала GROUP BY разбивает строки на группы, затем на полученные наборы накладываются условия HAVING. Например, устраним из предыдущего запроса те издательства, которые имеют только одну книгу:

```
    SELECT publishers.publisher, count(titles.title)
FROM titles,publishers
WHERE titles.pub_id=publishers.pub_id
GROUP BY publisher
HAVING COUNT(*)>1;
```

DML: Группировка данных

- Другой вариант использования HAVING - включить в результат только те издательства, название которых оканчивается на подстроку "Press":

```
SELECT publishers.publisher, count(titles.title)
FROM titles,publishers
WHERE titles.pub_id=publishers.pub_id
GROUP BY publisher
HAVING publisher LIKE '%Press';
```

- В чем различие между двумя этими вариантами использования HAVING? Во втором варианте условие отбора записей мы могли поместить в раздел ключевого слова WHERE, в первом же варианте этого сделать не удастся, поскольку WHERE не допускает использования агрегирующих функций.

DML: Сортировка данных

- Для сортировки данных, получаемых при помощи оператора SELECT служит ключевое слово ORDER BY. С его помощью можно сортировать результаты по любому столбцу или выражению, указанному в **<списке_выбора>.** Данные могут быть упорядочены как по возрастанию, так и по убыванию. Пример: сортировать список авторов по алфавиту:

```
SELECT author
FROM authors
ORDER BY author;
```

DML: Сортировка данных

- Более сложный пример: получить список авторов, отсортированный по алфавиту, и список их публикаций, причем для каждого автора список книг сортируется по времени издания в обратном порядке (т. е. сначала более "свежие" книги, затем все более "древние"):

```
SELECT authors.author, titles.title, titles.yearpub, publishers.publisher
FROM authors, titles, publishers, titleauthors
WHERE titleauthors.au_id=authors.au_id
  AND   titleauthors.title_id=titles.title_id
  AND   titles.pub_id=publishers.pub_id
ORDER BY authors.author ASC, titles.yearpub DESC;
```

Ключевое слово DESC задает здесь обратный порядок сортировки по полю **yearpub**, ключевое слов ASC (его можно опускать) - прямой порядок сортировки по полю **author**.

DML: Операция объединения

- В SQL предусмотрена возможность выполнения операции реляционной алгебры "ОБЪЕДИНЕНИЕ" (UNION) над отношениями, являющимися результатами оператора SELECT. Естественно, эти отношения должны быть определены по одной схеме. Пример: получить все Интеренет-ссылки, хранимые в базе данных **publications**. Эти ссылки хранятся в таблицах **publishers** и **wwwsites**. Для того, чтобы получить их в одной таблице, мы должны построить следующие запрос:

```
 SELECT publisher, url
FROM publishers
UNION
SELECT site, url
FROM wwwsites;
```

Порядок выполнения оператора SELECT

- Для того чтобы понять, как получается результат выполнения оператора SELECT, рассмотрим концептуальную схему его выполнения. Эта схема является именно концептуальной, т.к. гарантируется, что результат будет таким, как если бы он выполнялся шаг за шагом в соответствии с этой схемой. На самом деле, реально результат получается более изощренными алгоритмами, которыми "владеет" конкретная СУБД.

- **Стадия 1. Выполнение одиночного оператора SELECT**
- Если в операторе присутствуют ключевые слова UNION, EXCEPT и INTERSECT, то запрос разбивается на несколько независимых запросов, каждый из которых выполняется отдельно:
- *Шаг 1 (FROM)*. Вычисляется прямое декартовое произведение всех таблиц, указанных в обязательном разделе FROM. В результате шага 1 получаем таблицу A.

Порядок выполнения оператора SELECT

- *Шаг 2 (WHERE)*. Если в операторе SELECT присутствует раздел WHERE, то сканируется таблица A, полученная при выполнении шага 1. При этом для каждой строки из таблицы A вычисляется условное выражение, приведенное в разделе WHERE. Только те строки, для которых условное выражение возвращает значение TRUE, включаются в результат. Если раздел WHERE опущен, то сразу переходим к шагу 3. Если в условном выражении участвуют вложенные подзапросы, то они вычисляются в соответствии с данной концептуальной схемой. В результате шага 2 получаем таблицу B.

Порядок выполнения оператора SELECT

- *Шаг 3 (GROUP BY)*. Если в операторе SELECT присутствует раздел GROUP BY, то строки таблицы B, полученной на втором шаге, группируются в соответствии со списком группировки, приведенным в разделе GROUP BY. Если раздел GROUP BY опущен, то сразу переходим к шагу 4. В результате шага 3 получаем таблицу C.
- *Шаг 4 (HAVING)*. Если в операторе SELECT присутствует раздел HAVING, то группы, не удовлетворяющие условному выражению, приведенному в разделе HAVING, исключаются. Если раздел HAVING опущен, то сразу переходим к шагу 5. В результате шага 4 получаем таблицу D.

Порядок выполнения оператора SELECT

- *Шаг 5 (SELECT)*. Каждая группа, полученная на шаге 4, генерирует одну строку результата следующим образом. Вычисляются все скалярные выражения, указанные в разделе SELECT. По правилам использования раздела GROUP BY, такие скалярные выражения должны быть одинаковыми для всех строк внутри каждой группы. Для каждой группы вычисляются значения агрегатных функций, приведенных в разделе SELECT. Если раздел GROUP BY отсутствовал, но в разделе SELECT есть агрегатные функции, то считается, что имеется всего одна группа. Если нет ни раздела GROUP BY, ни агрегатных функций, то считается, что имеется столько групп, сколько строк отобрано к данному моменту. В результате шага 5 получаем таблицу E, содержащую столько колонок, сколько элементов приведено в разделе SELECT и столько строк, сколько отобрано групп.

Порядок выполнения оператора SELECT

- **Стадия 2. Выполнение операций UNION, EXCEPT, INTERSECT**
- Если в операторе SELECT присутствовали ключевые слова UNION, EXCEPT и INTERSECT, то таблицы, полученные в результате выполнения 1-й стадии, объединяются, вычитаются или пересекаются.
- **Стадия 3. Упорядочение результата**
- Если в операторе SELECT присутствует раздел ORDER BY, то строки полученной на предыдущих шагах таблицы упорядочиваются в соответствии со списком упорядочения, приведенном в разделе ORDER BY.

Как на самом деле выполняется оператор SELECT

- Если внимательно рассмотреть приведенный выше концептуальный алгоритм вычисления результата оператора SELECT, то сразу понятно, что выполнять его непосредственно в таком виде чрезвычайно накладно. Даже на самом первом шаге, когда вычисляется декартово произведение таблиц, приведенных в разделе FROM, может получиться таблица огромных размеров, причем практически большинство строк и колонок из нее будет отброшено на следующих шагах.
- На самом деле в РСУБД имеется ***оптимизатор***, функцией которого является нахождение такого *оптимального алгоритма* выполнения запроса, который гарантирует получение правильного результата.

Как на самом деле выполняется оператор SELECT

- Схематично работу оптимизатора можно представить в виде последовательности нескольких шагов:
- *Шаг 1 (Синтаксический анализ)*. Поступивший запрос подвергается синтаксическому анализу. На этом шаге определяется, правильно ли вообще (с точки зрения синтаксиса SQL) сформулирован запрос. В ходе синтаксического анализа вырабатывается некоторое внутренне представление запроса, используемое на последующих шагах.

Как на самом деле выполняется оператор SELECT

- *Шаг 2 (Преобразование в каноническую форму)*. Запрос во внутреннем представлении подвергается преобразованию в некоторую каноническую форму. При преобразовании к канонической форме используются как синтаксические, так и семантические преобразования. Синтаксические преобразования (например, приведения логических выражений к конъюнктивной или дизъюнктивной нормальной форме, замена выражений "x AND NOT x" на "FALSE", и т.п.) позволяют получить новое внутренне представление запроса, синтаксически *эквивалентное* исходному, но стандартное в некотором смысле. Семантические преобразования используют дополнительные знания, которыми владеет система, например, ограничения целостности. В результате семантических преобразований получается запрос, синтаксически *не эквивалентный* исходному, но дающий *тот же самый результат*.

Как на самом деле выполняется оператор SELECT

- *Шаг 3 (Генерация планов выполнения запроса и выбор оптимального плана)*. На этом шаге оптимизатор генерирует множество возможных планов выполнения запроса. Каждый план строится как комбинация низкоуровневых процедур доступа к данным из таблиц, методам соединения таблиц. Из всех сгенерированных планов выбирается план, обладающий минимальной стоимостью. При этом анализируются данные о наличии индексов у таблиц, статистических данных о распределении значений в таблицах, и т.п. Стоимость плана это, как правило, сумма стоимостей выполнения отдельных низкоуровневых процедур, которые используются для его выполнения. В стоимость выполнения отдельной процедуры могут входить оценки количества обращений к дискам, степень загруженности процессора и другие параметры.

Как на самом деле выполняется оператор SELECT

- *Шаг 4. (Выполнение плана запроса)*. На этом шаге план, выбранный на предыдущем шаге, передается на реальное выполнение.
- Во многом качество конкретной СУБД определяется качеством ее оптимизатора. Хороший оптимизатор может повысить скорость выполнения запроса на несколько порядков. Качество оптимизатора определяется тем, какие методы преобразований он может использовать, какой статистической и иной информацией о таблицах он располагает, какие методы для оценки стоимости выполнения плана он знает.

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

- Для того, чтобы показать, что язык SQL является реляционно полным, нужно показать, что любой реляционный оператор может быть выражен средствами SQL. На самом деле достаточно показать, что средствами SQL можно выразить любой из *примитивных* реляционных операторов.

- **Оператор декартового произведения**

  Реляционная алгебра: *A* **TIMES** *B*

  Оператор SQL:

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
   FROM A, B;
```

  или

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
   FROM A CROSS JOIN B;
```

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

- **Оператор проекции**

  Реляционная алгебра: *A*[*X,Y,…,Z*]

  Оператор SQL:

  ```
  SELECT DISTINCT X, Y, …, Z
     FROM A;
  ```

- **Оператор выборки**

  Реляционная алгебра: *A* **WHERE** *c*

  Оператор SQL:

  ```
  SELECT *
     FROM A
     WHERE c;
  ```

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

- **Оператор объединения**

  Реляционная алгебра: *A* **UNION** *B*

  Оператор SQL:

```
SELECT *
   FROM A
   UNION
   SELECT *
      FROM B;
```

- **Оператор вычитания**

  Реляционная алгебра: *A* **MINUS** *B*

  Оператор SQL:

```
SELECT *
   FROM A
   EXCEPT
   SELECT *
      FROM B
```

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

- Реляционный оператор переименования RENAME выражается при помощи ключевого слова AS в списке отбираемых полей оператора SELECT. Таким образом, язык SQL является реляционно-полным.
- Остальные операторы реляционной алгебры (соединение, пересечение, деление) выражаются через примитивные, следовательно, могут быть выражены операторами SQL. Тем не менее, для практических целей приведем их.
- **Оператор соединения**

  Реляционная алгебра: (*A* **TIMES** *B)* **WHERE** *c*

  Оператор SQL:

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
   FROM A, B
   WHERE c;
```

  или

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
   FROM A CROSS JOIN B
   WHERE c;
```

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

- **Оператор пересечения**

  Реляционная алгебра: *A* **INTERSECT** *B*

  Оператор SQL:

```
SELECT *
   FROM A
   INTERSECT
   SELECT *
      FROM B;
```

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

- **Оператор деления**

  Реляционная алгебра: *A(X,Y)* **DEVIDBY** *B(Y)*

  Оператор SQL:

```
SELECT DISTINCT A.X
  FROM A
  WHERE NOT EXIST
    (SELECT *
       FROM B
       WHERE NOT EXIST
         (SELECT *
            FROM A A1
            WHERE
               A1.X = A.X AND
               A1.Y = B.Y));
```

Использование представлений

- До сих пор мы говорили о таблицах, которые *реально* хранятся в базе данных. Это, так называемые, базовые таблицы (base tables). Существует другой вид таблиц, получивший название "представления" (иногда их называют"представляемые таблицы").

  Определение:
  Представление (**view**) - это таблица, содержимое которой берется из других таблиц посредством запроса. При этом новые копии данных не создаются

- Когда содержимое базовых таблиц меняется, СУБД автоматически перевыполняет запросы, создающие view, что приводит к соответствующим изменениям в представлениях.

Использование представлений

- Представление определяется с помощью команды

  CREATE VIEW <имя_представления> [<имя_столбца>,...]
  AS <запрос>

- При этом должны соблюдаться следующие ограничения:

  - представление должно базироваться на единственном запросе (UNION не допустимо);

  - выходные данные запроса, формирующего представление, должны быть не упорядочены (ORDER BY не допустимо).

Использование представлений

- Создадим представление, хранящее информацию об авторах, их книгах и издателях этих книг:

```
CREATE VIEW books AS
  SELECT authors.author, titles.title, titles.yearpub, publishers.publisher
    FROM authors, titles, publishers, titleauthors
    WHERE titleauthors.au_id=authors.au_id
  AND titleauthors.title_id=titles.title_id
  AND titles.pub_id=publishers.pub_id;
```

- Теперь любой пользователь, чьих прав на доступ к данному представлению достаточно, может осуществлять выборку данных из **books** (Права пользователей на доступ в представлениям назначаются также с помощью команд GRANT / REVOKE).
- Например:

```
    SELECT titles
  FROM books
  WHERE author LIKE '%Date‘;
```

Использование представлений

- Другой пример:

```
SELECT author, count(title)
    FROM books
    GROUP BY author
```

- Из приведенного выше примера достаточно ясен смысл использования представлений. Если запросы типа "выбрать все книги данного автора с указанием издательств" выполняются достаточно часто, то создание представляемой таблицы **books** значительно сократит накладные расходы на выполнение соединения четырех базовых таблиц **authors**, **titles**, **publishers** и **titleauthors**. Кроме того, в представлении может быть представлена информация, явно не хранимая ни в одной из базовых таблиц. Например, один из столбцов представления может быть вычисляемым:

Использование представлений

```
CREATE VIEW amount (publisher, books_count) AS
  SELECT publishers.publisher, count(titles.title)
  FROM titles,publishers
  WHERE titles.pub_id=publishers.pub_id
  GROUP BY publisher;
```

- Здесь использована еще одна, ранее не описанная, возможность SQL - присвоение новых имен столбцам представления. В приведенном примере число изданий, осуществленных каждым издателем, будет хранится в столбце с именем **books_count**. Заметим, что если мы хотим присвоить новые имена столбцам представления, нужно указывать имена для *всех* столбцов. Тип данных столбца представления и его нулевой статус всегда зависят от того, как он был определен в базовой таблице (таблицах).

Использование представлений

- Запрос на выборку данных к представлению выглядит абсолютно аналогично запросу к любой другой таблице. Однако на изменение данных в представлении накладываются ограничения. Кратко о них можно сказать следующее:

  - Если представление основано на одной таблице, изменения данных в нем допускаются. При этом изменяются данные в связанной с ним таблице.

  - Если представление основано более чем на одной таблице, то изменения данных в нем не допускаются, т.к. в большинстве случаев СУБД не может правильно восстановить схему базовых таблиц из схемы представления.

- Удаление представления производится с помощью оператора:

  DROP VIEW <имя_представления>

Хранимые процедуры

- Практический опыт создания приложений обработки данных показывает, что ряд операций над данными, реализующих общую для всех пользователей логику и не связанных с пользовательским интерфейсом, целесообразно вынести на сервер. Однако, для написания процедур, реализующих эти операции стандартных возможностей SQL не достаточно, поскольку здесь необходимы операторы обработки ветвлений, циклов и т.д. Поэтому многие поставщики СУБД предлагают собственные **процедурные** расширения SQL (PL/SQL компании Oracle и т.д.). Эти расширения содержат логические операторы (IF ... THEN ... ELSE), операторы перехода по условию (SWITCH ... CASE ...), операторы циклов (FOR, WHILE, UNTIL) и операторы передачи управления в процедуры (CALL, RETURN). С помощью этих средств создаются функциональные модули, которые хранятся на сервере вместе с базой данных. Обычно такие модули называют *хранимые процедуры* (Stored Procedures) . Они могут быть вызваны с передачей параметров любым пользователем, имеющим на то соответствующие права.

Хранимые процедуры

- В некоторых системах хранимые процедуры могут быть реализованы и в виде внешних по отношению к СУБД модулей на языках общего назначения, таких как *C* или *Pascal*. Пример для СУБД PostgreSQL:

  CREATE FUNCTION <имя_функции>
  ([<тип_параметра1>,...<тип_параметра2>])
  RETURNS <возвращаемые_типы>
  AS [<SQL_оператор> | <имя_объектного_модуля>]
  LANGUAGE 'SQL' | 'C' | 'internal'

- Вызов созданной функции осуществляется из оператора SELECT (также, как вызываются функции агрегирования).

Триггеры

- Для каждой таблицы может быть назначена хранимая процедура без параметров, которая вызывается при выполнении оператора модификации этой таблицы (INSERT, UPDATE, DELETE). Такие хранимые процедуры получили название триггеров. Триггеры выполняются автоматически, независимо от того, что именно является причиной модификации данных - действия человека оператора или прикладной программы.

Триггеры

- "Усредненный" синтаксис оператора создания триггера:

  CREATE TRIGGER <имя_триггера>
  ON <имя_таблицы>
  FOR { INSERT | UPDATE | DELETE }
  [, INSERT | UPDATE | DELETE] ...
  AS <SQL_оператор>

- Ключевое слово ON задает имя таблицы, для которой определяется триггер, ключевое слово FOR указывает какая команда (команды) модификации данных активирует триггер. Операторы SQL после ключевого слова AS описывают действия, которые выполняет триггер и условия выполнения этих действий. Здесь может быть перечислено любое число операторов SQL, вызовов хранимых процедур и т.д. Использование триггеров очень удобно для выполнения операций контроля ограничений целостности.

Транзакции, блокировки и многопользовательский доступ к данным

- Любая база данных годна к использованию только тогда, когда ее состояние соответствует состоянию предметной области. Такие состояния называют целостными. Очевидно, что при изменении данных БД должна переходить от одного целостного состояния к другому. Однако, в процессе обновления данных возможны ситуации, когда состояние целостности нарушается. Например:

  В банковской системе производится перевод денежных средств с одного счета на другой. На языке SQL эта операция описывается последовательностью двух команд UPDATE:

  UPDATE accounts SET summa=summa-1000 WHERE account="PC_1"

  UPDATE accounts SET summa=summa+1000 WHERE account="PC_2"

- Как видим, после выполнения первой команды и до завершения второй команды база данных не находится в целостном состоянии (искомая сумма списана с первого счета, но не зачислена на второй). Если в этот момент в системе произойдет сбой (например, выключение электропитания), то целостное состояние БД будет безвозвратно утеряно.

Транзакции, блокировки и многопользовательский доступ к данным

- Целостность БД может нарушаться и во время обработки одной команды SQL. Пусть выполняется операция увеличения зарплаты всех сотрудников фирмы на 20%:

  UPDATE employers SET salary=salary*1.2

- При этом СУБД последовательно обрабатывает все записи, подлежащие обновлению, т.е. существует временной интервал, когда часть записей содержит новые значения, а часть - старые.
- Во избежание таких ситуаций в СУБД вводится понятие **транзакции** - атомарного действия над БД, переводящего ее из одного целостного состояния в другое целостное состояние.
- Другими словами, *транзакция - это последовательность операций, которые должны быть или все выполнены или все не выполнены (все или ничего).*

Транзакции, блокировки и многопользовательский доступ к данным

- Методом контроля за транзакциями является ведение *журнала*, в котором фиксируются все изменения, совершаемые транзакцией в БД. Если во время обработки транзакции происходит сбой, транзакция откатывается - из журнала восстанавливается состояние БД на момент начала транзакции.
- В СУБД различных поставщиков начало транзакции может задаваться явно (например, командой **BEGIN TRANSACTION**), либо предполагаться неявным (так определено в стандарте SQL), т. е. очередная транзакция открывается автоматически сразу же после удачного или неудачного завершения предыдущей. Для завершения транзакции обычно используют команды SQL:

  - COMMIT - успешно завершить транзакцию;

  - ROLLBACK - откатить транзакцию, т.е. вернуть БД в состояние, в котором она находилась на момент начала транзакции.

Транзакции, блокировки и многопользовательский доступ к данным

- Стандарт SQL определяет, что транзакция начинается с первого SQL-оператора, инициируемого пользователем или содержащегося в прикладной программе. Все последующие SQL-операторы составляют тело транзакции. Транзакция завершается одним из возможных способов:

  - оператор **COMMIT** означает успешное завершение транзакции, все изменения, внесенные в базу данных делаются постоянными;

  - оператор **ROLLBACK** прерывает транзакцию и отменяет все внесенные ею изменения;

  - успешное завершение программы, инициировавшей транзакцию, означает успешное завершение транзакции (как использование **COMMIT**);

  - ошибочное завершение программы прерывает транзакцию (как **ROLLBACK**).

Транзакции, блокировки и многопользовательский доступ к данным

- Пример явно заданной транзакции:

```
BEGIN TRANSACTION;            /* Начать транзакцию */
DELETE ...;                   /* Изменения */
UPDATE ...;                   /* данных    */
if (обнаружена_ошибка) ROLLBACK;
else COMMIT;                  /* Завершить транзакцию */
```

- Пример неявно заданной транзакции:

```
COMMIT;                   /* Окончание предыдущей транзакции */
DELETE ...;               /* Изменения */
UPDATE ...;               /* данных    */
 if (обнаружена_ошибка) ROLLBACK;
else COMMIT;              /* Завершить транзакцию */
```

Транзакции, блокировки и многопользовательский доступ к данным

- К сожалению, описанный механизм транзакций гарантирует обеспечение целостного состояния базы данных только в том случае, когда все транзакции выполняются последовательно, т.е. в каждую единицу времени активна только одна транзакция. Если работу с данными ведут одновременно несколько пользователей, вряд ли их устроит такой способ организации обработки запросов, т.к. это приведет к увеличению времени реакции системы. В то же время, если одновременно выполняются две транзакции, могут возникнуть следующие ошибочные ситуации:
- **Грязное чтение** (Dirty Read) - транзакция T1 модифицировала некий элемент данных. После этого другая транзакция T2 прочитала содержимое этого элемента данных до завершения транзакции T1. Если T1 завершается операцией ROLLBACK, то получается, что транзакция T2 прочитала не существующие данные.

Транзакции, блокировки и многопользовательский доступ к данным

- **Неповторяемое (размытое) чтение** (Non-repeatable or Fuzzy Read) - транзакция T1 прочитала содержимое элемента данных. После этого другая транзакция T2 модифицировала или удалила этот элемент. Если T1 прочитает содержимое этого элемента заново, то она получит другое значение или обнаружит, что элемент данных больше не существует.
- **Фантом (фиктивные элементы)** (Phantom) - транзакция T1 прочитала содержимое нескольких элементов данных, удовлетворяющих некому условию. После этого T2 создала элемент данных, удовлетворяющий этому условию и зафиксировалась. Если T1 повторит чтение с тем же условием, она получит другой набор данных.

Транзакции, блокировки и многопользовательский доступ к данным

- Как уже было сказано, ни одна из этих ситуаций не может возникнуть при последовательном выполнении транзакций. Отсюда возникло понятие **сериализуемости** (способности к упорядочению) параллельной обработки транзакций. Т.е. чередующееся (параллельное) выполнение заданного множества транзакций будет верным, если при его выполнении будет получен такой же результат, как и при *последовательном* выполнении тех же транзакций.
- Все описанные выше ситуации возникли только потому, что чередующееся выполнение транзакций T1 и T2 не было упорядочено, т.е. не было эквивалентно выполнению сначала транзакции T1, а затем T2, либо, наоборот, сначала транзакции T2, а затем T1.

Транзакции, блокировки и многопользовательский доступ к данным

- Принудительное упорядочение транзакций обеспечивается с помощью механизма **блокировок**. Суть этого механизма в следующем: если для выполнения некоторой транзакции необходимо, чтобы некоторый объект базы данных (кортеж, набор кортежей, отношение, набор отношений,…) не изменялся непредсказуемо и без ведома этой транзакции, такой объект блокируется. Основными видами блокировок являются:

  - блокировка со взаимным доступом, называемая также *S-блокировкой* (от Shared locks) и *блокировкой по чтению;*

  - монопольная блокировка (без взаимного доступа), называемая также *X-блокировкой* от (eXclusive locks) или *блокировкой по записи*. Этот режим используется при операциях изменения, добавления и удаления объектов.

Транзакции, блокировки и многопользовательский доступ к данным

- При этом:

  - если транзакция налагает на объект X-блокировку, то любой запрос другой транзакции с блокировкой этого объекта будет отвергнут;

  - если транзакция налагает на объект S-блокировку, то:

  – запрос со стороны другой транзакции с X-блокировокй на этот объект будет отвергнут;

  – запрос со стороны другой транзакции с S-блокировокй этого объекта будет принят.

- Транзакция, запросившая доступ к объекту, уже захваченному другой транзакцией в несовместимом режиме, останавливается до тех пор, пока захват этого объекта не будет снят.

Транзакции, блокировки и многопользовательский доступ к данным

- Доказано, что ***сериализуемость*** транзакций (или, иначе, их ***изоляция***) обеспечивается при использовании двухфазного протокола блокировок (2LP - Two-Phase Locks), согласно которому все блокировки, произведенные транзакцией, снимаются только при ее завершении. Т.е. выполнение транзакции разбивается на две фазы: (1) - накопление блокировок, (2) - освобождение блокировок в результате фиксации или отката.
- К сожалению, применение механизма блокировки приводит к замедлению обработки транзакций, поскольку система вынуждена ожидать пока освободятся данные, захваченные конкурирующей транзакцией. Решить эту проблему можно за счет уменьшения фрагментов данных, захватываемых транзакцией

Транзакции, блокировки и многопользовательский доступ к данным

- В зависимости от захватываемых объектов различают несколько *уровней блокировки*:

  - блокируется вся база данных - очевидно, этот вариант неприемлем, поскольку сводит многопользовательский режим работы к однопользовательскому;

  - блокируются отдельные таблицы;

  - блокируются страницы (страница - фрагмент таблицы размером обычно 2-4 Кб, единица выделения памяти для обработки данных системой);

  - блокируются записи;

  - блокируются отдельные поля.

- Современные СУБД, как правило, могут осуществлять блокировку на уровне записей или страниц.

Транзакции, блокировки и многопользовательский доступ к данным

- Язык SQL также предоставляет способ косвенного управления скоростью выполнения транзакций с помощью указания *уровня изоляции* транзакции. Под уровнем изоляции транзакции понимается возможность возникновения одной из описанных выше ошибочных ситуаций. В стандарте SQL определены 4 уровня изоляции:

Уровень изоляции	**Грязное чтение**	**Размытое чтение**	**Фантом**
Незафиксированное чтение (READ UNCOMMITTED)	возможно	возможно	возможно
Зафиксированное чтение (READ COMMITED)	невозможно	возможно	возможно
Повторяемое чтение (REPEATABLE READ)	невозможно	невозможно	возможно
Сериализуемость (SERIALIZABLE)	невозможно	невозможно	невозможно

Транзакции, блокировки и многопользовательский доступ к данным

- Для определения характеристик транзакции используется оператор

SET TRANSACTION <режим_доступа>, <уровень_изоляции>

- Список уровней изоляции приведен в таблице. Режим доступа по умолчанию используется **READ WRITE** (чтение запись), если задан уровень изоляции READ UNCOMMITED, то режим доступа должен быть **READ ONLY** (только чтение).

Транзакции, блокировки и многопользовательский доступ к данным

- Одним из наиболее серьезных недостатков метода сериализации транзакций на основе механизма блокировок является возможность возникновения тупиков (dead locks) между транзакциями. Пусть, например, транзакция T1 наложила монопольную блокировку на объект O1 и претендует на доступ к объекту O2, который уже монопольно заблокирован транзакцией T2, ожидающей доступа к объекту O1. В этом случае ни одна из транзакций продолжаться не может, следовательно, блокировки объектов O1 и O2 никогда не будут сняты. Естественного выхода из такой ситуации не существует, поэтому тупиковые ситуации обнаруживаются и устраняются искусственно. При этом СУБД откатывает одну из транзакций, попавших в тупик ("жертвует" ею), что дает возможность продолжить выполнение другой транзакции.

Этапы проектирования данных

- Напомним еще раз определение понятия "предметная область":
- **Предметная область** - часть реального мира, подлежащая изучению с целью организации управления и, в конечном счете, автоматизации. Предметная область представляется множеством *фрагментов*, например, предприятие - цехами, дирекцией, бухгалтерией и т.д. Каждый фрагмент предметной области характеризуется множеством *объектов* и *процессов*, использующих объекты, а также множеством *пользователей*, характеризуемых различными взглядами на предметную область.
- В теории проектирования информационных систем предметную область (или, если угодно, весь реальный мир в целом) принято рассматривать в виде трех представлений:

  - представление предметной области в том виде, как она реально существует;

  - как ее воспринимает человек (имеется в виду проектировщик базы данных);

  - как она может быть описана с помощью символов.

Этапы проектирования данных

- Т.е. говорят, что мы имеем дело с реальностью, описанием (представлением) реальности и с данными, которые отражают это представление.
- Данные, используемые для описания предметной области, представляются в виде трехуровневой схемы (так называемая модель ANSI/SPARC):

Этапы проектирования данных

- *Внешнее представление* (внешняя схема) данных является совокупностью требований к данным со стороны некоторой конкретной функции, выполняемой пользователем.
- *Концептуальная схема* является полной совокупностью всех требований к данным, полученной из пользовательских представлений о реальном мире.
- *Внутренняя схема* - это сама база данных.

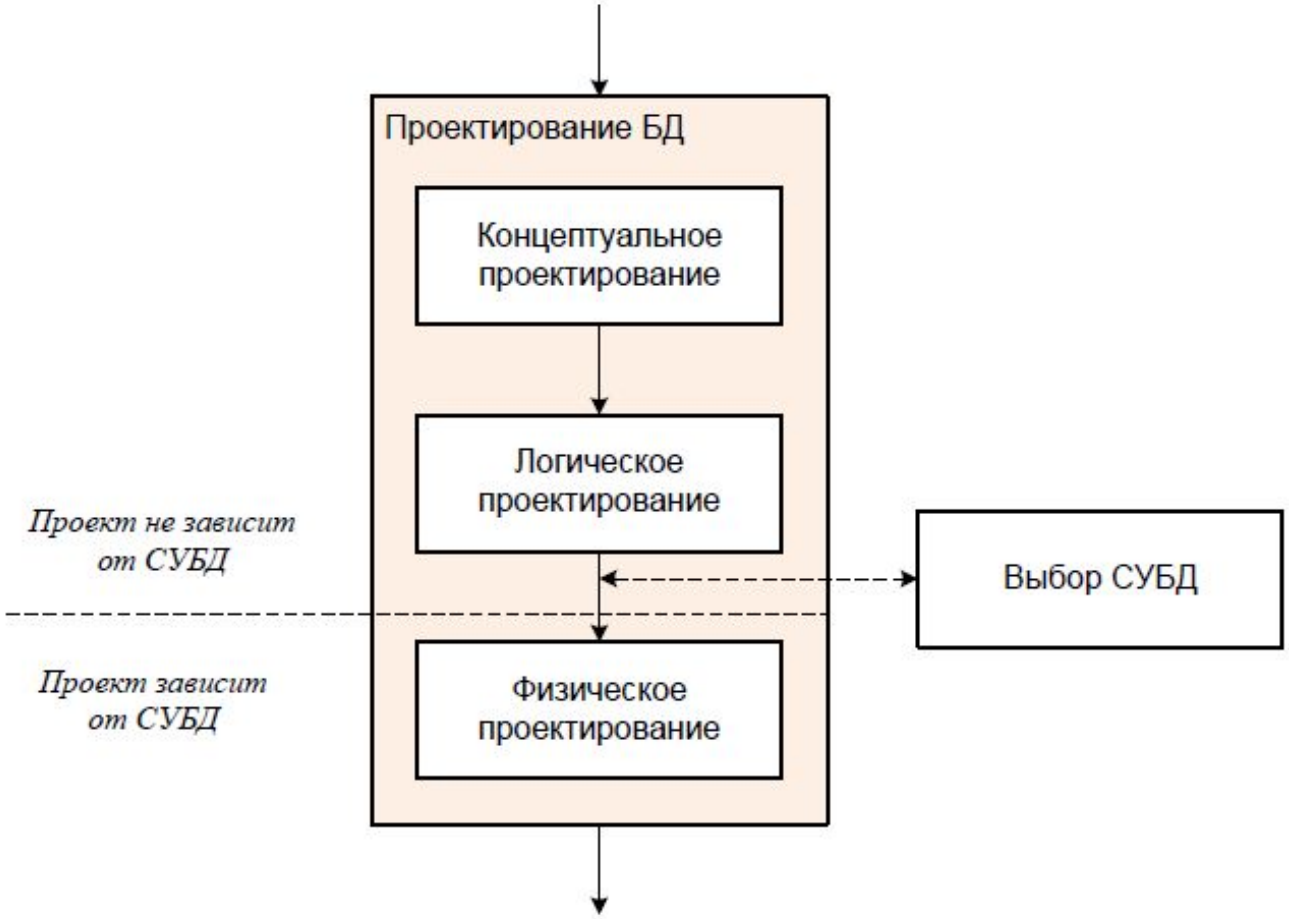
- Отсюда вытекают основные этапы, на которые разбивается процесс проектирования базы данных информационной системы:

  1. Концептуальное проектирование;

  2. Логическое проектирование;

  3. Физическое проектирование.

Этапы проектирования данных



Этапы проектирования БД

- **Концептуальное проектирование**
- Во время концептуального проектирования окончательно формируется замысел будущей базы данных, но без учета любых физических аспектов ее реализации.
- На этой ступени проектирования разработчика пока не интересует ни конкретная СУБД, на которой позднее развернется БД, ни используемый для создания приложений язык программирования, ни особенности аппаратной платформы. Пока его основной интерес направлен на создание общей модели, отражающей представления будущих пользователей БД об автоматизируемом участке компании (складе, бухгалтерии, отделе кадров, производственных цехах и т. п.). Все необходимая для этого информация уже должна быть собрана на предыдущем этапе жизненного цикла БД.

Этапы проектирования БД

- **Логическое проектирование**
- Фаза логического проектирования предназначена для преобразования обобщенной концептуальной модели в завершенную логическую.
- Разработчик уточняет все требования, выявленные на концептуальной стадии проектирования, и стремится несколько упростить решение (не снижая его функциональные возможности). Для этого ER-модель проверяют с помощью правил нормализации. В результате мы получаем неизбыточные реляционные таблицы, свободные от присущих ненормализованным данным аномалиям вставки, редактирования и удаления.
- Помимо нормализации, на логическом этапе осуществляют следующие действия:

  - уточняют ограничения на данные;

  - определяют домены данных;

  - вводят бизнес-правила и корпоративные ограничения целостности.

Этапы проектирования БД

- На этапах *концептуального* и *логического* проектирования разработчики обычно используют одну из двух стратегий проектирования БД: *восходящее проектирование (bottom-up design) или нисходящее проектирование (top-down design).*
- *Восходящий подход* обычно применяется для сравнительно небольших проектов. Суть метода заключается в том, что проектировщик совместно с заказчиком БД строят полный список атрибутов (полей таблиц) подлежащих хранению. Позднее атрибуты группируют в типы сущностей, которые попадают в модель. Здесь можно заметить, что процесс нормализации основан на восходящем подходе.
- Обратная восходящей, *нисходящая стратегия* лучше подходит для средних и больших проектов. Здесь проектирование начинается с выявления основных типов сущностей и только затем сущности "обрастают" атрибутами. Классический пример нисходящего метода — модель "сущность-связь" (ER-модель).

Этапы проектирования БД

- **Физическое проектирование**
- К следующей (физической) фазе проектирования БД переходят после выбора целевой СУБД, именно она определяет особенности будущего программного продукта. С этого момента все остальные фазы проектирования и этапы жизненного цикла БД приобретают зависимость от СУБД.
- Физическое проектирование — это уточнение решения с учетом имеющихся в наличии разработчика технологий, возможности реализации и требуемой производительности. Только на заключительной фазе проектирования БД на смену так нелюбимой программистами бумажной деятельности приходит реальная работа на компьютере. Во время физического проектирования задачей проектировщика становится перенос логической модели на платформу целевой СУБД.

Этапы проектирования БД

- С этой целью разработчик делает следующее:

  - создает таблицы и связи между ними;

  - назначает вторичные индексы таблиц;

  - реализует бизнес-логику БД (в первую очередь, с помощью триггеров и хранимых процедур);

  - определяет функциональные характеристики транзакций;

  - разрабатывает представления;

  - внедряет механизмы защиты (как минимум предусматривает авторизацию пользователей и назначает правила доступа к данным).
- Полученная БД тщательным образом документируется. Особенно важно определить пользовательские типы данных, описать таблицы и связи между ними, задать порядок поддержки бизнес-логики, установить назначение порядок вызова хранимых процедур и триггеров.

Инструментальные средства проектирования информационных систем

- Во многих случаях эффективную информационную систему не удается построить вручную. Это объясняется следующими причинами:

  - не обеспечивается достаточно глубокий анализ требований к данным;

  - большая длительность процесса структурирования;

  - трудность учета и согласования изменений, сделанных в системе несколькими разработчиками;

  - ограничения сроков на разработку системы;

  - и т.д.
- При разработке крупных информационных систем происходит концентрация сложности на начальных этапах (анализ требований и проектирование спецификаций системы), в то время как сложность и трудоемкость последующих этапов остается относительно невысокой.

Инструментальные средства проектирования информационных систем

- Для преодоления сложностей начальных этапов разработки предназначен структурный анализ - метод исследования, которое начинается с общего обзора системы и затем детализуется, приобретая иерархическую структуру со все большим числом уровней. На каждом уровне рассматривается ограниченное число элементов (обычно от 3 до 6-8), каждый из которых в свою очередь может быть декомпозирован на составляющие детали на следующем уровне. При этом соблюдаются строгие формальные правила записи информации (обычно используются диаграммы различных типов).
- Такая технология получила название **CASE** (Computer Aided Software Engeneering - создание программного обеспечения с помощью компьютера).

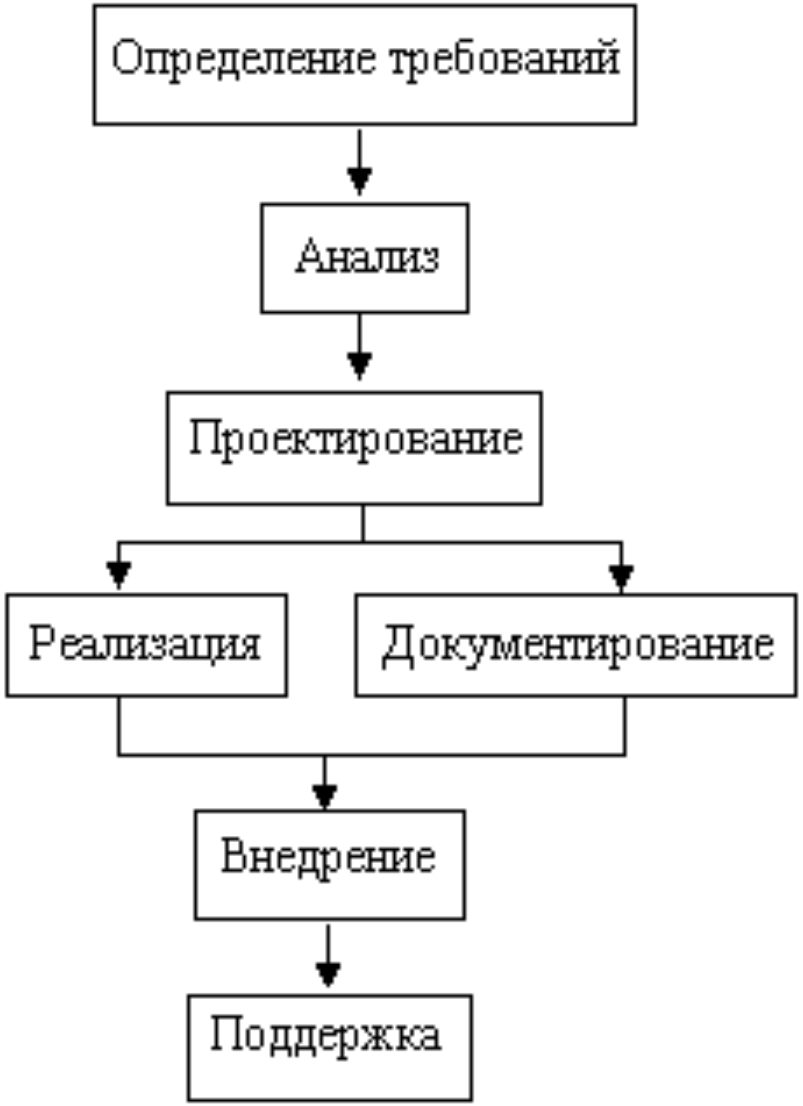
Инструментальные средства проектирования информационных систем

- Основные черты CASE - технологии:

  - использование методологии структурного проектирования "сверху-вниз“;

  - разработка прикладной системы представляется в виде последовательных четко определенных этапов:

Инструментальные средства проектирования информационных систем



Инструментальные средства проектирования информационных систем

- поддержка всех этапов жизненного цикла информационной системы, начиная с самых общих описаний предметной области до получения и сопровождения готового программного продукта;
- поддержка репозитария, хранящего спецификации проекта информационной системы на всех этапах ее разработки;
- возможность одновременной работы с репозитарием многих разработчиков;
- автоматизация различных стандартных действий по проектированию и реализации приложения.

Инструментальные средства проектирования информационных систем

- Как правило, CASE-системы поддерживают следующие этапы процесса разработки:

  - *Моделирование и анализ деятельности пользователей в рамках предметной области* (Методология функционального моделирования). Здесь осуществляется функциональная декомпозиция, определение иерархий (вложенности) функций, построение диаграмм потоков данных. Перечень информационных объектов, которыми манипулируют функции, передается на следующий этап проектирования.

  - *Концептуальное моделирование* - создание модели "сущность-связь" на основе перечня объектов, полученного на предыдущем этапе. Здесь уточняются характеристики каждого объекта (атрибуты), устанавливаются связи между объектами.

  - *Реляционное моделирование* - преобразование модели "сущность-связь" в соответствии с требованиями реляционной модели (реляционная модель допускает только бинарные связи, не разрешает существование атрибутов у связей, не поддерживает связи типа *n : m*).

Инструментальные средства проектирования информационных систем

- *Генерация схемы базы данных*. Результатом выполнения данного этапа является набор SQL-операторов, описывающих создание схемы базы данных (CREATE TABLE, CREATE INDEX,...), с учетом особенностей целевой СУБД.

- *Генерация прототипов программных модулей по иерархии функций и потокам данных*. Для каждого модуля автоматически подготавливается описание используемых им фрагментов данных (таблицы, атрибуты, индексы), а также создаются заготовки экранных форм или отчетов.

Инструментальные средства проектирования информационных систем

- Изучение CASE-систем выходит за рамки данного курса. Перечислим наиболее известные CASE-системы существующие на рынке:

  - Power Designer компании Sybase;

  - Silverrun компании Silverrun Technologies Ltd;

  - BPWin и ERWin компании LogicWorks;

  - Designer/2000 компании Oracle.

- Приведенные CASE-системы поддерживают следующие этапы процесса разработки:

  - Функциональное моделирование;

  - Концептуальное моделирование;

  - Реляционное моделирование;

  - Генерация схемы базы данных;

  - Генерация прототипов программных модулей по иерархии функций и потокам данных.

Методологии функционального моделирования

- Существуют различные методологии функционального моделирования, например:

  - Диаграммы потоков данных (DFD - Data Flow Diagramm);

  - Методология SADT (Structured Analisys and Design Technique);

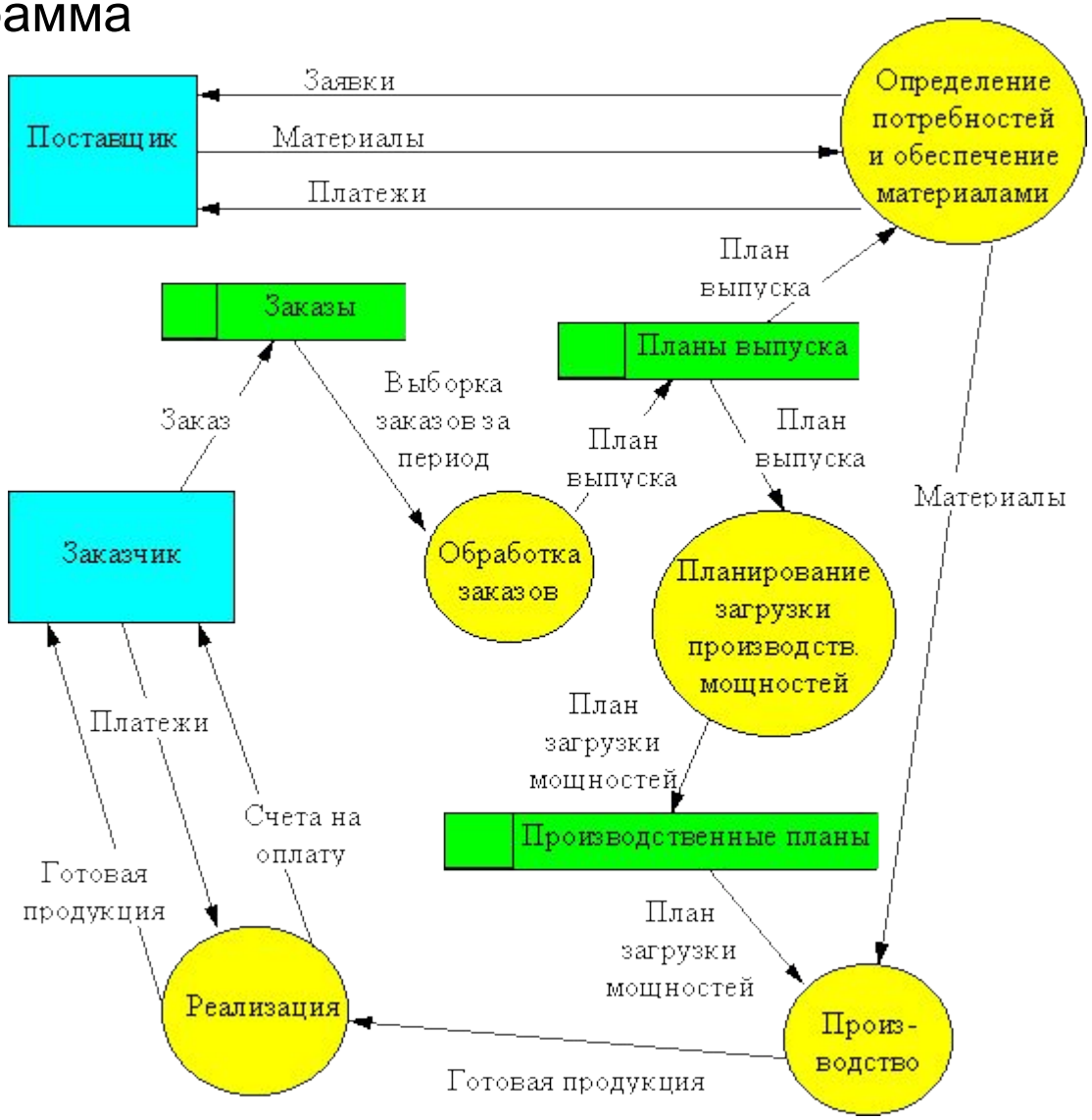
  - другие методологии.

- Методология функционального моделирования, позволяют выделить первичные информационные объекты, из которых затем строятся *концептуальная и реляционная модели данных*.
- Рассмотрение этих методов выходит за рамки данного курса.

- Приведем пример DFD-диаграммы для предприятия, строящего свою деятельность по принципу "изготовление на заказ".

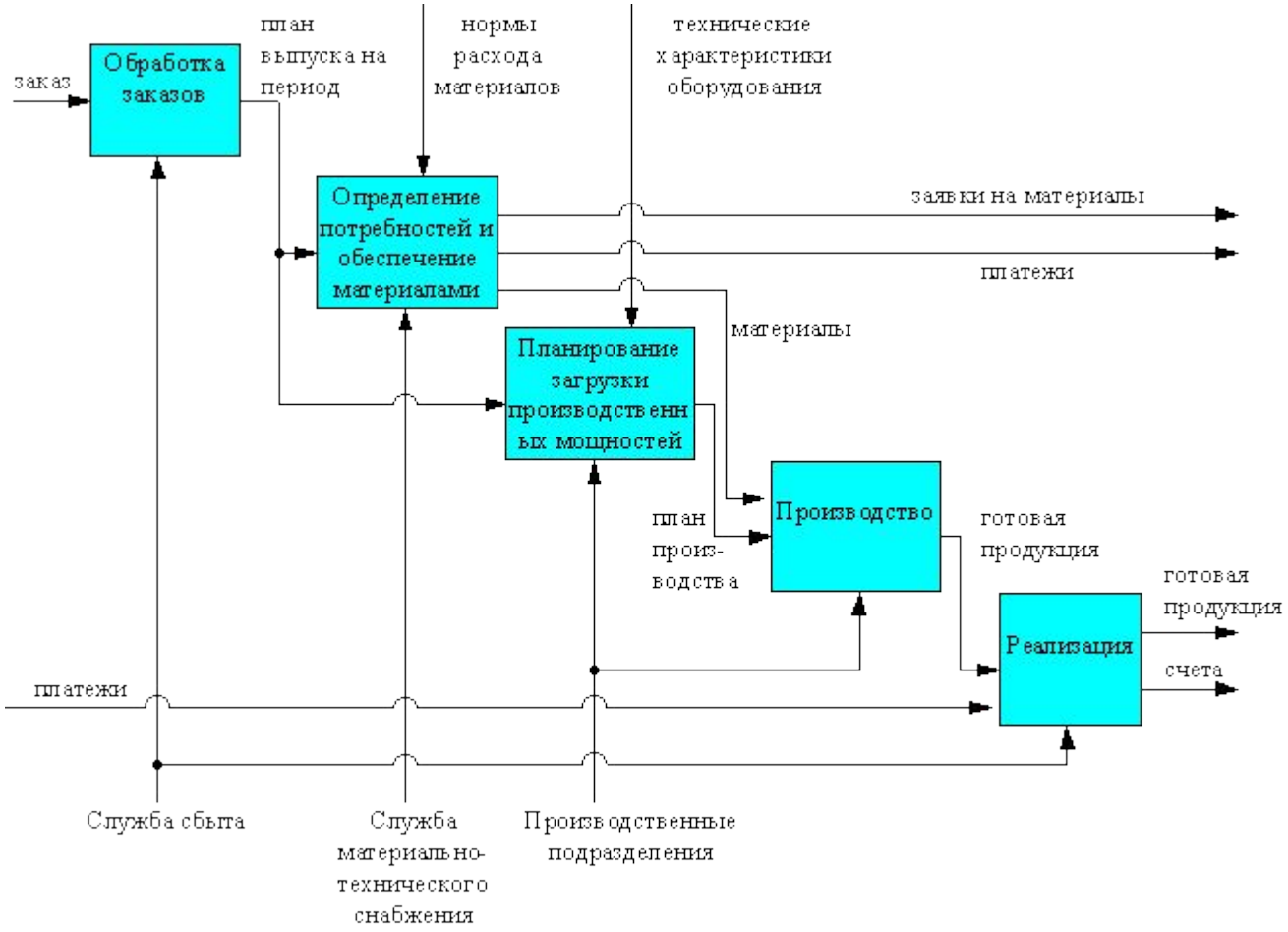
Методологии функционального моделирования

- DFD-диаграмма



Методологии функционального моделирования

- IDEF0-модель (Методология SADT)



Концептуальное моделирование

- Методология функционального моделирования, позволяют выделить первичные информационные объекты, из которых затем строятся *концептуальная и реляционная модели данных*. Однако, в случае достаточно простой предметной области выделение информационных объектов можно произвести и без функционального анализа.
- Один из способов такого проектирования структуры реляционной базы данных описан в этом разделе. Далее будет рассмотрен другой способ проектирования реляционной структуры, основанный на декомпозиции универсального отношения.

Концептуальное моделирование

Пример построения модели "сущность-связь"

- Здесь мы рассмотрим пример, связанный с проектированием базы данных **publucations**, которая использовалась для практических занятий при изучении языка SQL.
- БД **publications** должна хранить сведения о печатных изданиях, а также ссылки на интересные ресурсы в Internet. И те и другие источники информации будут касаться одной темы, а именно "баз данных". Попробуем выделить интересующие нас сущности и определить связи между ними.
- Прежде всего займемся понятием "печатное издание". Что это такое? Мы знаем, что объект "печатное издание" воплощается в виде книги, которую можно полностью описать с помощью следующих характеристик: название, автор, год издания и издатель (издательство). Можно ли на основании этого ввести сущность "книга", а названные характеристики определить в качестве ее атрибутов?

Концептуальное моделирование

- Прежде чем сделать это рассмотрим более внимательно отношения между книгой и ее характеристиками:

  - Один автор может написать несколько книг, и, в то же время, одна книга может быть написана несколькими авторами. Следовательно, "книга" и "автор" в данном случае выступают как различные сущности, объединяемые связью *N : M*. Для того, чтобы определить класс принадлежности сущностей в связи, отметим, что книг без авторов не бывает, как и авторов без книг. Значит, каждая сущность должна иметь обязательный класс принадлежности (кардинальность связи*(1,N) : (1,N)*).

  - Точно так же один издатель может издавать сразу несколько книг, но каждая конкретная книга издается только в одном месте. Следовательно, мы должны ввести сущность "издатель", ассоциируемую с "книгой" связью типа *1 : N*. Т.к. каждая книга кем-то издана, класс принадлежности сущности "издатель" в данной связи будет *(1,1)*, но в то же время мы допускаем хранение сведений об издательствах, чьих книг в нашей базе данных пока нет. Соответственно, класс принадлежности сущности "книга" в этой связи *(0,N)*.

Концептуальное моделирование

- По поводу характеристики книги "название" можно сказать следующее: как правило авторы, пишущие на одну тему, стараются придумывать для своих произведений оригинальные названия. Поэтому, можно уверенно предположить, что каждое название обязательно связано только с одной книгой (и каждая книга имеет только одно название). Следовательно, "название" нужно оставить в списке атрибутов "книги".

- Те же рассуждения можно повторить и для характеристики "год издания". Ее мы тоже оставим в списке атрибутов "книги".

- Таким образом, мы определили, что у сущности "книга" имеется два атрибута "название" и "год издания". Как уже говорилось, название, скорее всего, будет однозначно определять данную книгу, чего не скажешь о годе издания. Поэтому объявим ключом сущности атрибут "название" (или "имя_книги").

Концептуальное моделирование

- Что касается всех возможных авторов, то нас интересует только одна их характеристика - имя. Поэтому, сущность "автор" имеет только один атрибут "имя_автора", который и является ключом.
- С сущностью "издатель" дел обстоит несколько сложнее. Практически все крупные издательства имеют сейчас собственные web-страницы, которые могут содержать информацию полезную для пользователей проектируемой базы данных. Поэтому, нужно рассмотреть две характеристики этого объекта: "имя_издателя" и "URL" (uniform resource locator - универсальный указатель ресурсов, с помощью которого в Internet определяется путь к web - странице). Ясно, что каждый издатель имеет уникальное имя и уникальный url, но прежде чем внести их в список атрибутов, вспомним, что наша база данных должна также содержать ссылки и на другие Internet-ресурсы. Возможно, при дальнейшем анализе возникнет необходимость во введении отдельной сущности "URL". Поэтому "имя_издателя" внесем в список атрибутов сущности "издатель", а "URL" будем считать атрибутом отдельной сущности "web - страница", ассоциируемой с "издателем" связью *(1,1):(1,1)*.

Концептуальное моделирование

- Теперь настала пора заняться объектом "ресурс Internet". Его мы можем описать с помощью понятий "имя ресурса", "url", "автор". Внимательно рассмотрев связи этих понятий с описываемым объектом, можно прийти к заключению, что "имя_ресурса" и "url" однозначно с ним связаны, т.е. являются атрибутами. В то же время, "автор" является отдельной сущностью (один ресурс может иметь много авторов, и один автор может быть создателем многих web - страниц). Т.к. мы уже ранее ввели сущность "автор" просто определим характеристики ее связи с сущностью "Internet-ресурс". Из сказанного выше следует, что эти сущности объединяются связью *n : m*, в то же время, автор какой-либо книги может не иметь собственной web - страницы, а авторы некоторых Internet ресурсов не указывают своих имен (т.е. можно формально сказать, что эти ресурсы не имеют авторов). Следовательно, класс принадлежности обеих сущностей будет необязательным.
- Прежде чем объявить нашу модель готовой, проверим еще раз определение каждой сущности. Внимательный анализ покажет, что построенная модель имеет несколько ошибок:

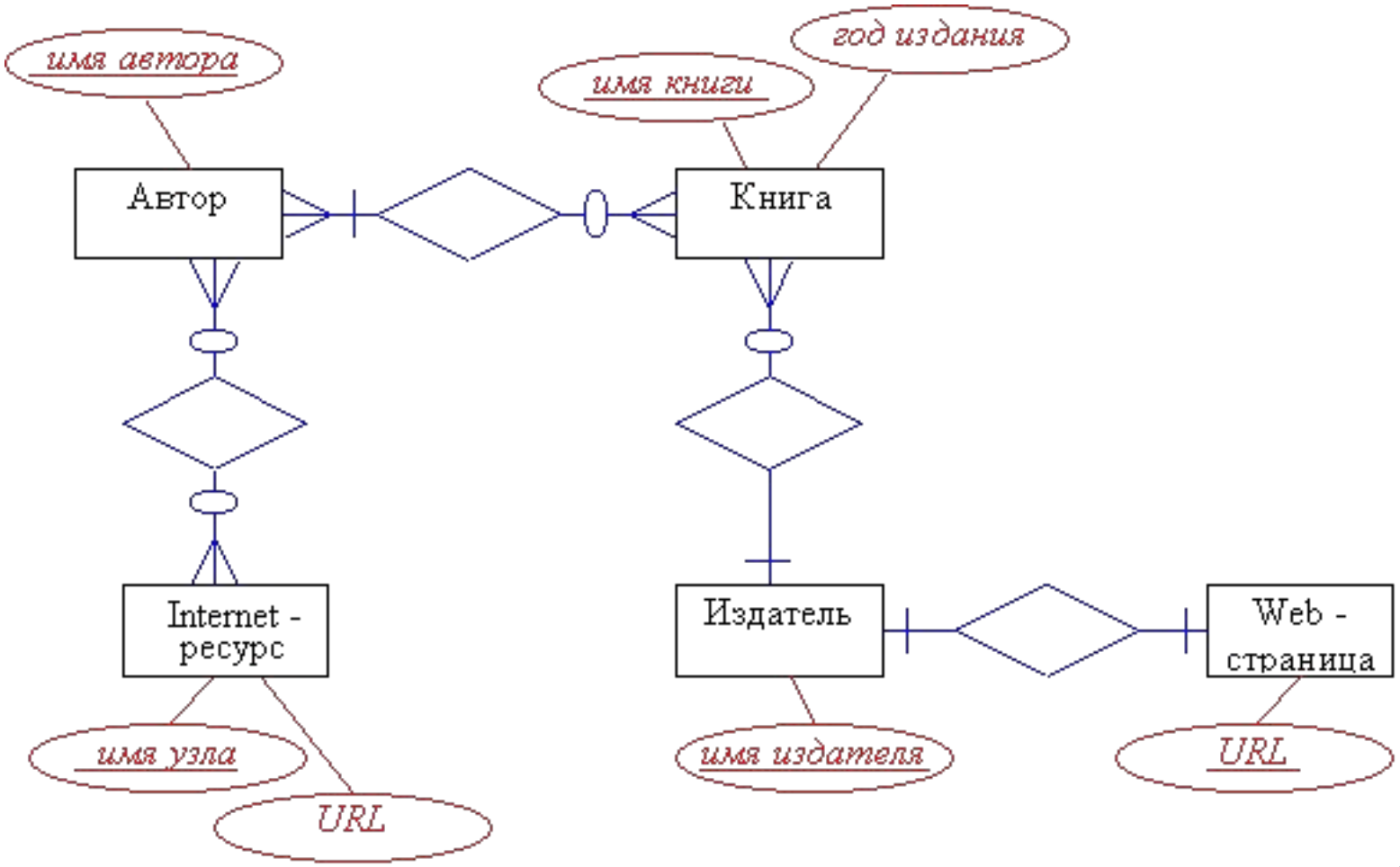
Концептуальное моделирование

- Сущность "автор" имеет обязательный класс принадлежности в связи с сущностью "книга". Это означает, что мы не сможем добавить в базу данных сведения о человеке, который создал собственный web - сайт, но не написал ни одной книги. Для того, что бы устранить это ограничение изменим класс принадлежности сущности "книга" в рассматриваемой связи "автор" - "книга" на необязательный.

- При анализе объекта "издатель" мы предположили, что сущность "web-страница" может быть объединена с сущностью "Internet-ресурс". Однако, мы видим, что эти сущности имеют разный набор атрибутов, следовательно выполнить такое объединение нельзя. Вспомним, что в противном случае, предполагалось единственный атрибут сущности "web - страница" присоединить к атрибутам сущности "издатель". Тем не менее, не будем этого делать, в следующем разделе мы увидим, что с помощью правил порождения реляционных отношений из модели "сущность-связь" в том и в другом случае мы получим одинаковый результат.

Концептуальное моделирование

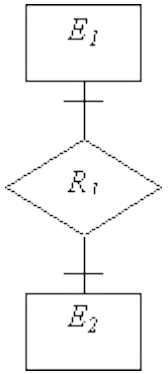
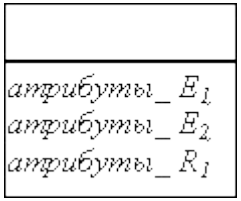
- Готовая модель "сущность-связь" (***концептуальная модель***) представлена на следующем рисунке:



Правила порождения реляционных отношений из модели "сущность-связь"

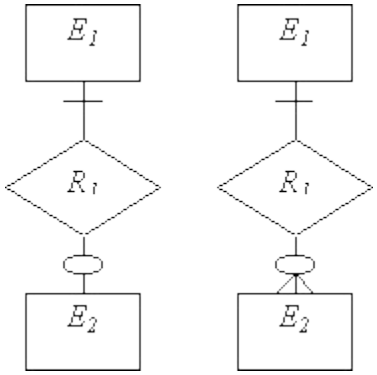
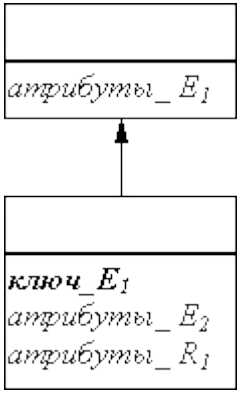
- Для перехода от концептуальной модели к логической необходимо определить правила порождения реляционных отношений. Эти правила определяются для разных типов связи.

- **Бинарные связи**

Тип связи	**Пример связи**	**Правило построения отношений**	**Отношения**
(1,1):(1,1)		Требуется только одно отношение. Первичным ключом данного отношения может быть ключ любой из сущностей.	

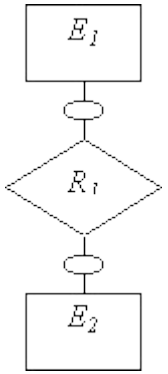
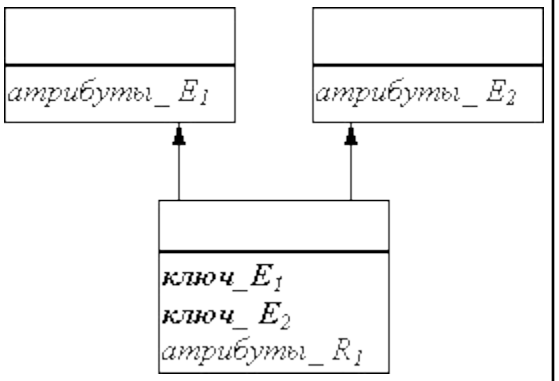
Правила порождения реляционных отношений из модели "сущность-связь"

- **Бинарные связи**

Тип связи	Пример связи	Правило построения отношений	Отношения
(1,1):(0,1) (1,1):(0,n)		Для каждой сущности создается свое отношение, при этом ключи сущностей служат ключами соответствующих отношений. Кроме того, ключ сущности с обязательным классом принадлежности добавляется в качестве внешнего ключа в отношение, созданное для сущности с необязательным классом принадлежности.	

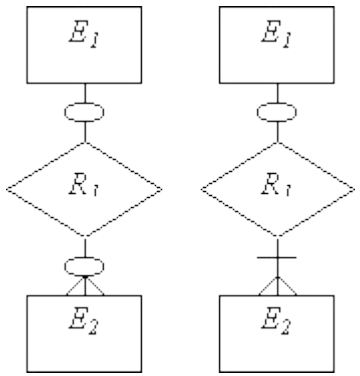
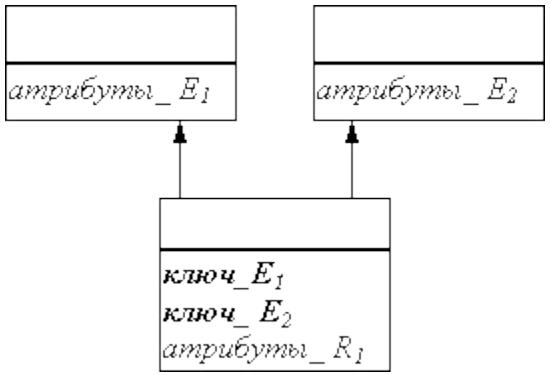
Правила порождения реляционных отношений из модели "сущность-связь"

- **Бинарные связи**

Тип связи	Пример связи	Правило построения отношений	Отношения
(0,1):(0,1)		Необходимо использовать три отношения: по одному для каждой сущности (ключи сущностей служат первичными ключами отношений) и одно отношение для связи. Отношение, выделенное для связи, имеет два атрибута - внешних ключа - по одному от каждой сущности.	

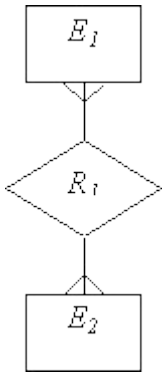
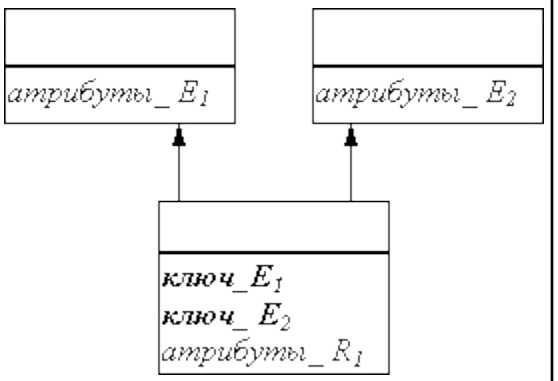
Правила порождения реляционных отношений из модели "сущность-связь"

- **Бинарные связи**

Тип связи	Пример связи	Правило построения отношений	Отношения
(0,1):(0,n) (0,1):(1,n)		Формируются три отношения: по одному для каждой сущности, причем ключ каждой сущности служит первичным ключом соответствующего отношения, и одно отношение для связи. Отношение, выделенное для связи, имеет два атрибута - внешних ключа - по одному от каждой сущности.	

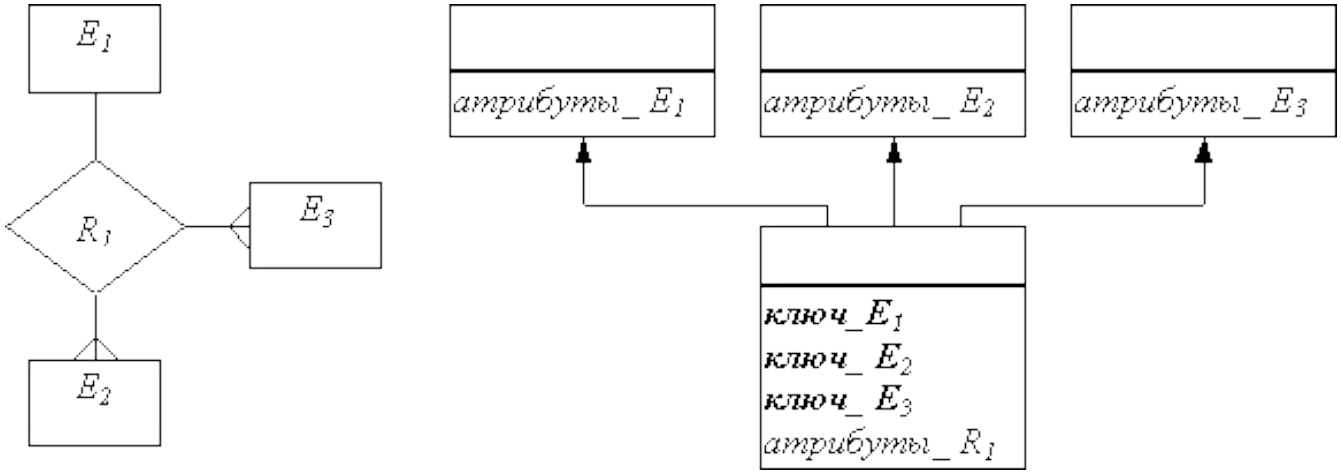
Правила порождения реляционных отношений из модели "сущность-связь"

- **Бинарные связи**

Тип связи	Пример связи	Правило построения отношений	Отношения
n : m		В этом случае всегда используются три отношения: по одному для каждой сущности, причем ключ каждой сущности служит первичным ключом соответствующего отношения, и одно отношение для связи. Последнее отношение должно иметь среди своих атрибутов внешние ключи, по одному от каждой сущности.	

Правила порождения реляционных отношений из модели "сущность-связь"

- **N - арные связи**
- Общее правило: для представления n-сторонней связи всегда требуется n+1 отношение. Например, в случае трехсторонней связи необходимо использовать четыре отношения, по одному для каждой сущности (причем ключ сущности служит первичным ключом соответствующего отношения), и одно для связи. Отношение, порождаемой для связи, будет иметь среди своих атрибутов ключи от каждой сущности.



Правила порождения реляционных отношений из модели "сущность-связь"

- **Иерархические связи**
- К сожалению, надо признать, что реляционная модель мало подходит для отображения отношений наследования между сущностями (иерархических связей). Напомним, что в таких связях дочерние сущности наследуют все атрибуты родительской, и каждая из них обладает своим уникальным набором дополнительных атрибутов. Ранее приведен пример такой связи между родительской сущностью ЗАКАЗЧИК и дочерними - ЗАРУБЕЖНОЕ_ПРЕДПРИЯТИЕ и ОТЕЧЕСТВЕННОЕ_ПРЕДПРИЯТИЕ.
- В этом случае возможны два варианта построения реляционных отношений. Согласно первому для иерархической структуры создается одно отношение, которое содержит атрибуты связи и всех сущностей. Для приведенного ранее примера мы должны создать отношение

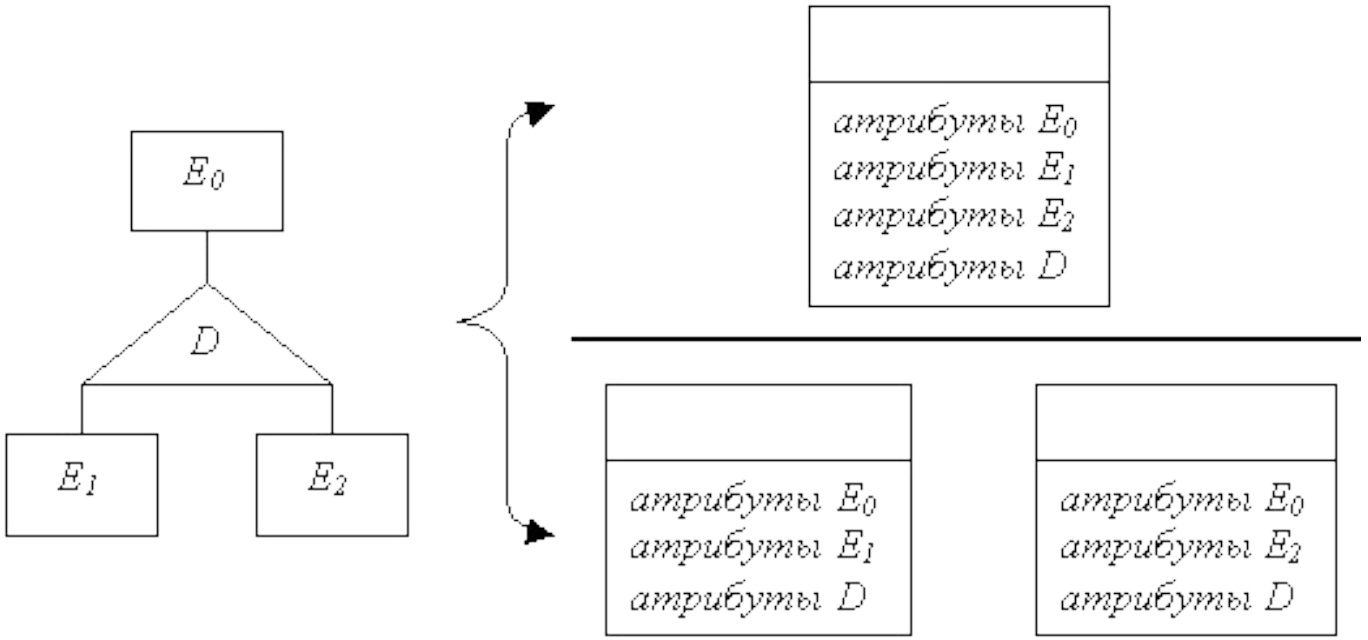
  ЗАКАЗЧИК(НАЦ_ПРИНАДЛЕЖНОСТЬ, ВАЛЮТА, ЯЗЫК, ФОРМА_СОБСТВЕННОСТИ).

Правила порождения реляционных отношений из модели "сущность-связь"

- Недостаток такого способа - для каждого кортежа часть атрибутов всегда будет неопределенна. Т.е. для отечественного предприятия всегда будут иметь значения NULL атрибуты ВАЛЮТА и ЯЗЫК, а для зарубежного атрибут ФОРМА_СОБСТВЕННОСТИ. Более того, этот факт является требованием целостности сущности, следовательно, для СУБД должны быть явно указаны несколько списков атрибутов (по числу дочерних сущностей), причем определенные значения могут быть присвоены только членам одного из них. Реляционная модель не поддерживает такого ограничения, на практике его реализуют с помощью триггеров.
- По второму способу генерируется по одному отношению для каждой дочерней сущности. Каждое из этих отношений включает атрибуты родительской сущности и связи кроме атрибутов - дискриминантов т.е. ЗАРУБЕЖНОЕ_ПРЕДПРИЯТИЕ(ВАЛЮТА, ЯЗЫК) и ОТЕЧЕСТВЕННОЕ_ПРЕДПРИЯТИЕ(ФОРМА_СОБСТВЕННОСТИ). Недостатком данного способа является невозможность получить в одном запросе список всех заказчиков.

Правила порождения реляционных отношений из модели "сущность-связь"

- Оба описанных способа представлены на рисунке:



Логическая модель данных

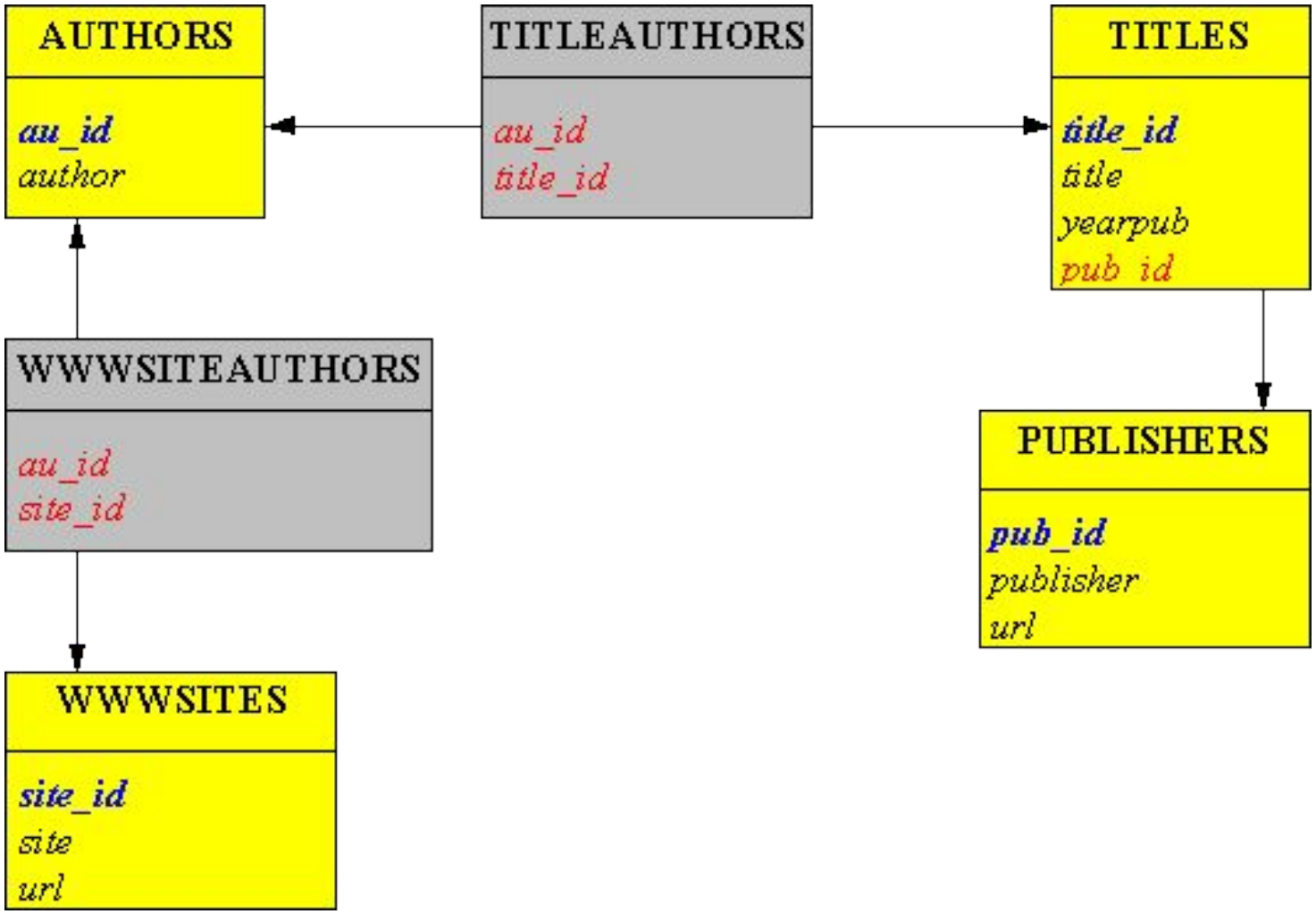
- **Следует отметить, что построенные таким образом реляционные отношения, не являются окончательной схемой базы данных. Их необходимо проверить на избыточные функциональные зависимости и привести к NFBK или нормальной форме более высокого порядка.**

- Применив все эти правила к модели "сущность-связь" базы данных **publications**, построенной в предыдущем параграфе, получим следующую реляционную структуру: (см. рисунок).
- Эта реляционная структура называется ***логической моделью***.

- Синим цветом на диаграмме выделены первичные ключи, красным - внешние. Отношения, созданные для представления связей, обозначены серыми прямоугольниками, для сущностей - желтыми прямоугольниками.

Логическая модель данных

- Логическая модель рассмотренного примера

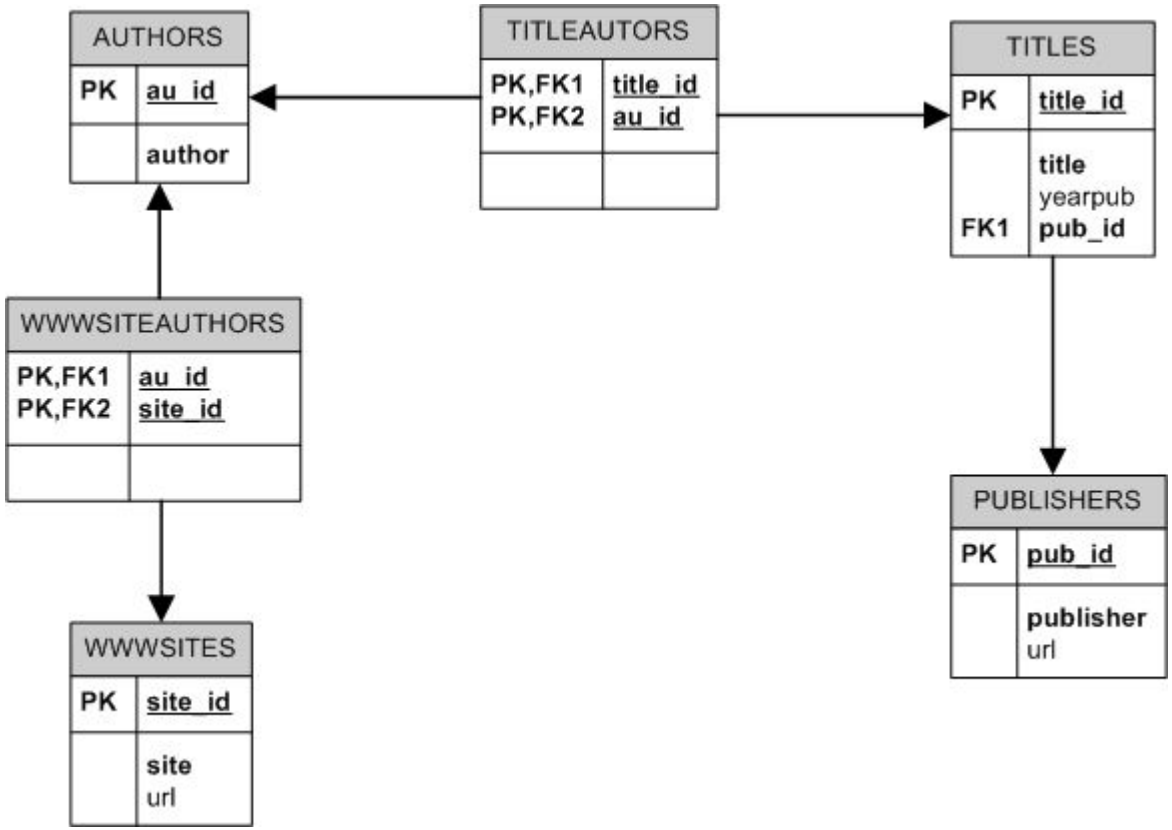


Логическая модель данных

- Для отображения логической модели так же используются различные нотации и кроме того, в логическую модель могут включать разный набор элементов, характеризующих реляционные данные. Далее будут приведены ещё некоторые варианты отображения логической модели из рассмотренного примера.

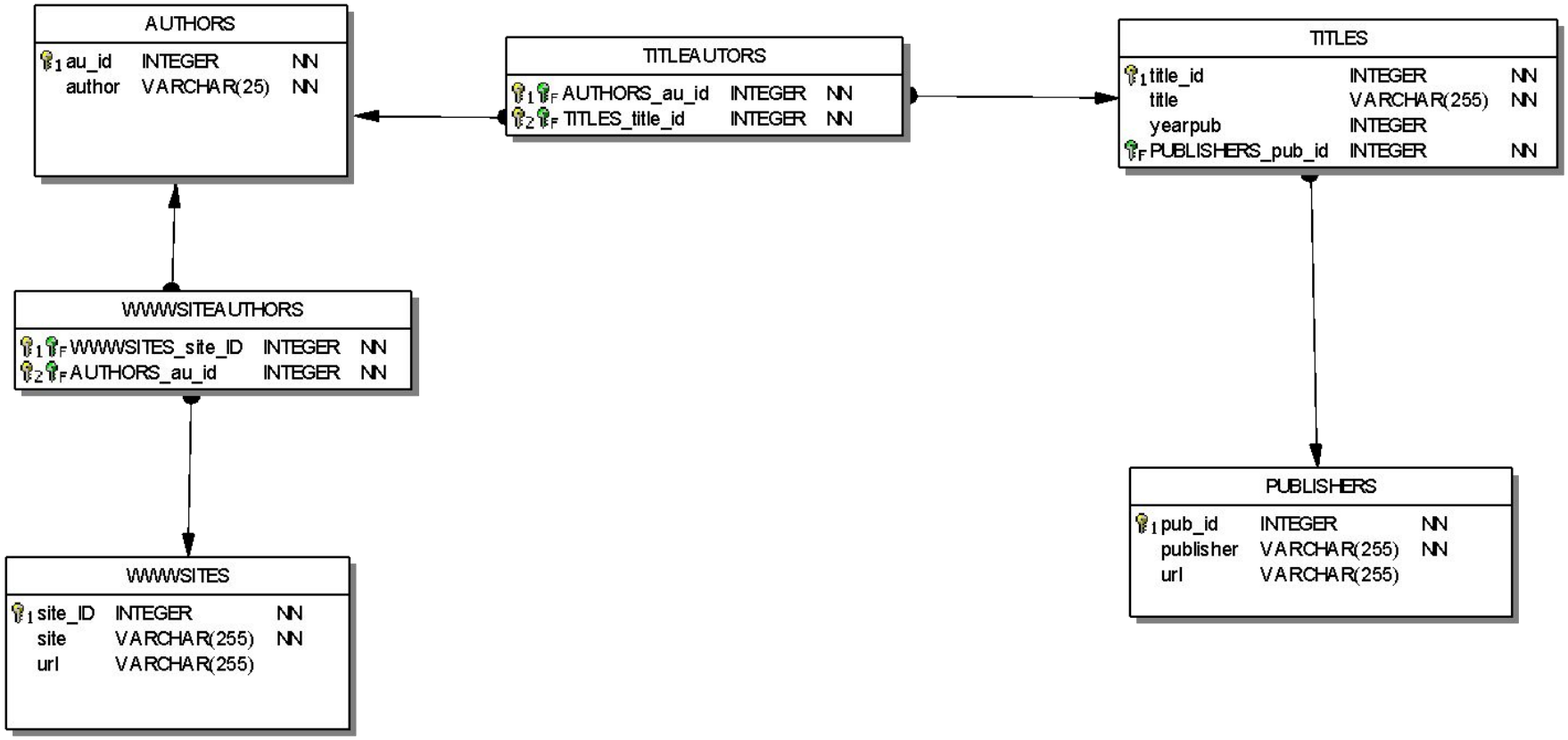
Логическая модель данных

- Логическая модель рассмотренного примера (построена в Visio)



Физическая модель данных

- Для построения физической модели необходимо определить используемую СУБД. Для нашего примера выберем СУБД FireBird.
- Физическую модель отобразим с помощью утилиты IBExpert. (NN сокращение от Not Null).



Физическая модель данных

- Пример скрипта создания БД на языке SQL сгенерированного автоматически из физической модели приведенной ранее с помощью утилиты IBExpert для СУБД FireBird.

```
/******************************************************************************
    /
/***           Generated by IBExpert 2010.03.23 18.12.2013 17:52:19
    ***/
/******************************************************************************
    /

/******************************************************************************
    /
/***                                 Domains
    ***/
/******************************************************************************
    /

/******************************************************************************
    /
/***                               Exceptions
    ***/
/******************************************************************************
    */

/******************************************************************************
    */
/***                               Procedures
    ***/
/******************************************************************************
```

Физическая модель данных

```
/*****************************************************************************
   */
/***                              Tables and Views
   ***/
/*****************************************************************************
   */

CREATE TABLE AUTHORS (
    AU_ID INTEGER NOT NULL,
    AUTHOR VARCHAR(25) NOT NULL);

CREATE TABLE PUBLISHERS (
    PUB_ID INTEGER NOT NULL,
    PUBLISHER VARCHAR(255) NOT NULL,
    URL VARCHAR(255));

CREATE TABLE TITLEAUTORS (
    AUTHORS_AU_ID INTEGER NOT NULL,
    TITLES_TITLE_ID INTEGER NOT NULL);

CREATE TABLE TITLES (
    TITLE_ID INTEGER NOT NULL,
    TITLE VARCHAR(255) NOT NULL,
    YEARPUB INTEGER,
    PUBLISHERS_PUB_ID INTEGER NOT NULL);

CREATE TABLE WWWSITEAUTHORS (
    WWWSITES_SITE_ID INTEGER NOT NULL,
    AUTHORS_AU_ID INTEGER NOT NULL);
```

Физическая модель данных

```
CREATE TABLE WWWSITES (
    SITE_ID INTEGER NOT NULL,
    SITE VARCHAR(255) NOT NULL,
    URL VARCHAR(255));


/*==============================================================
   */
/***                          Primary keys
   ***/
/*==============================================================
   */

ALTER TABLE AUTHORS ADD CONSTRAINT PK_AUTHORS PRIMARY KEY (AU_ID);
ALTER TABLE PUBLISHERS ADD CONSTRAINT PK_PUBLISHERS PRIMARY KEY (PUB_ID);
ALTER TABLE TITLEAUTORS ADD CONSTRAINT PK_TITLEAUTORS PRIMARY KEY
   (AUTHORS_AU_ID, TITLES_TITLE_ID);
ALTER TABLE TITLES ADD CONSTRAINT PK_TITLES PRIMARY KEY (TITLE_ID);
ALTER TABLE WWWSITEAUTHORS ADD CONSTRAINT PK_WWWSITEAUTHORS PRIMARY KEY
   (WWWSITES_SITE_ID, AUTHORS_AU_ID);
ALTER TABLE WWWSITES ADD CONSTRAINT PK_WWWSITES PRIMARY KEY (SITE_ID);

/*==============================================================
   */
/***                        Unique constraints
   ***/
/*==============================================================
   */
```

Физическая модель данных

```
/*==============================================================*/
/*                          Foreign keys                        */
/*==============================================================*/


ALTER TABLE TITLEAUTORS ADD CONSTRAINT FK_TITLEAUTORS_1 FOREIGN KEY
   (AUTHORS_AU_ID) REFERENCES AUTHORS (AU_ID);
ALTER TABLE TITLEAUTORS ADD CONSTRAINT FK_TITLEAUTORS_2 FOREIGN KEY
   (TITLES_TITLE_ID) REFERENCES TITLES (TITLE_ID);
ALTER TABLE TITLES ADD CONSTRAINT FK_TITLES_1 FOREIGN KEY (PUBLISHERS_PUB_ID)
   REFERENCES PUBLISHERS (PUB_ID);
ALTER TABLE WWWSITEAUTHORS ADD CONSTRAINT FK_WWWSITEAUTHORS_1 FOREIGN KEY
   (WWWSITES_SITE_ID) REFERENCES WWWSITES (SITE_ID);
ALTER TABLE WWWSITEAUTHORS ADD CONSTRAINT FK_WWWSITEAUTHORS_2 FOREIGN KEY
   (AUTHORS_AU_ID) REFERENCES AUTHORS (AU_ID);

/*==============================================================*/
/*                        Check constraints                     */
/*==============================================================*/


/*==============================================================*/
/*                           Indices                            */
/*==============================================================*/
```

Физическая модель данных

```
/*****************************************************************************
    */
/***                                    Triggers
    ***/
/*****************************************************************************
    */

SET TERM ^ ;

SET TERM ; ^

/*****************************************************************************
    */
/***                                   Procedures
    ***/
/*****************************************************************************
    */

SET TERM ^ ;

SET TERM ; ^
```

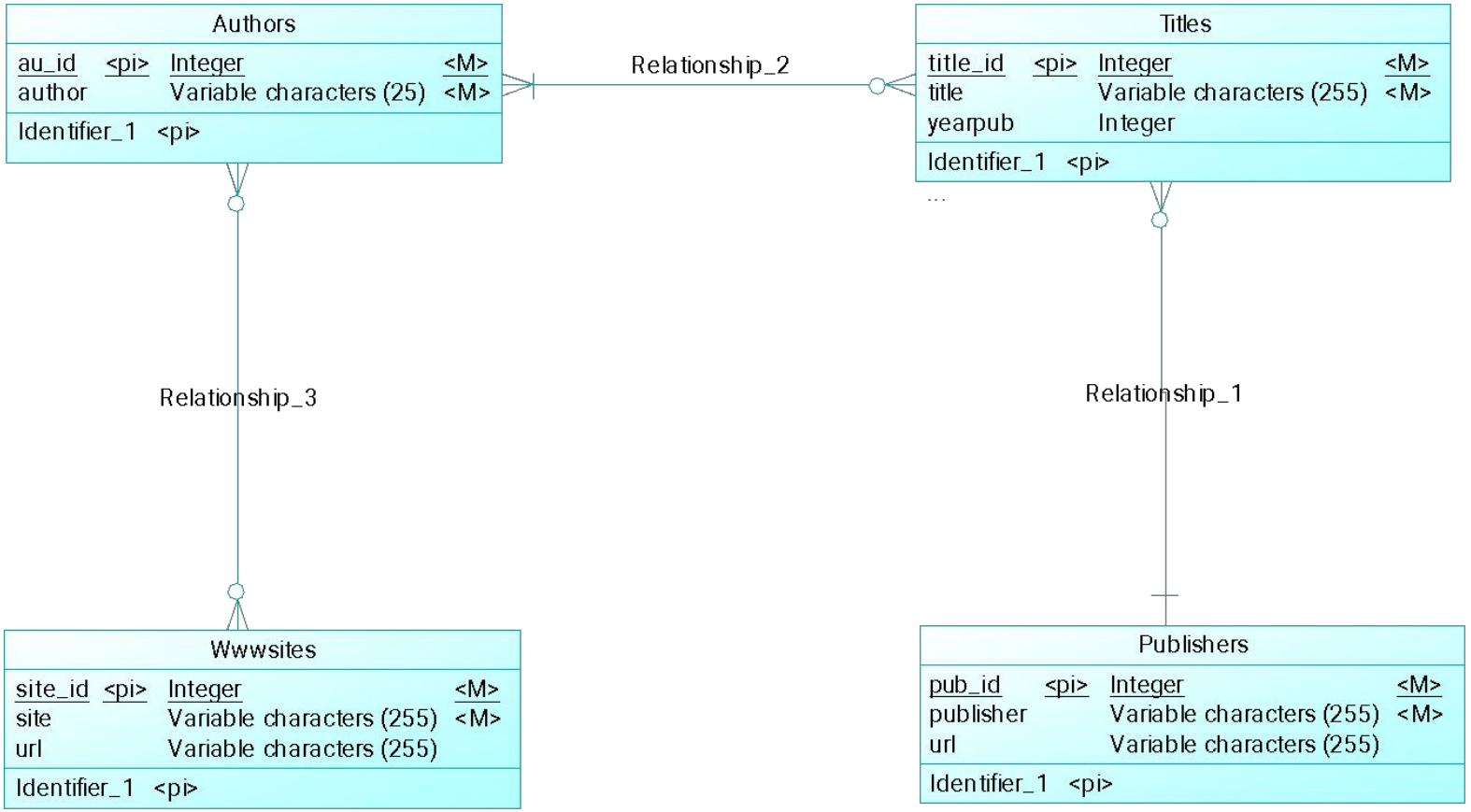
Пример построения моделей БД с использованием CASE системы PowerDesigner

- Рассмотрим процесс построения моделей БД на примере CASE системы PowerDesigner.
- Следует отметить, что первоначально была построена только концептуальная модель БД, а затем из неё автоматически получена логическая модель.
- После выбора целевой СУБД (Interbase или Firebird) автоматически была построена физическая модель, а из физической модели автоматически получен скрипт для создания БД на языке SQL для СУБД Interbase.

- Для построения концептуальная модели используется нотация близкая к нотации Мартина. <M> означает Mandatory – обязательный, т.е. Not Null.

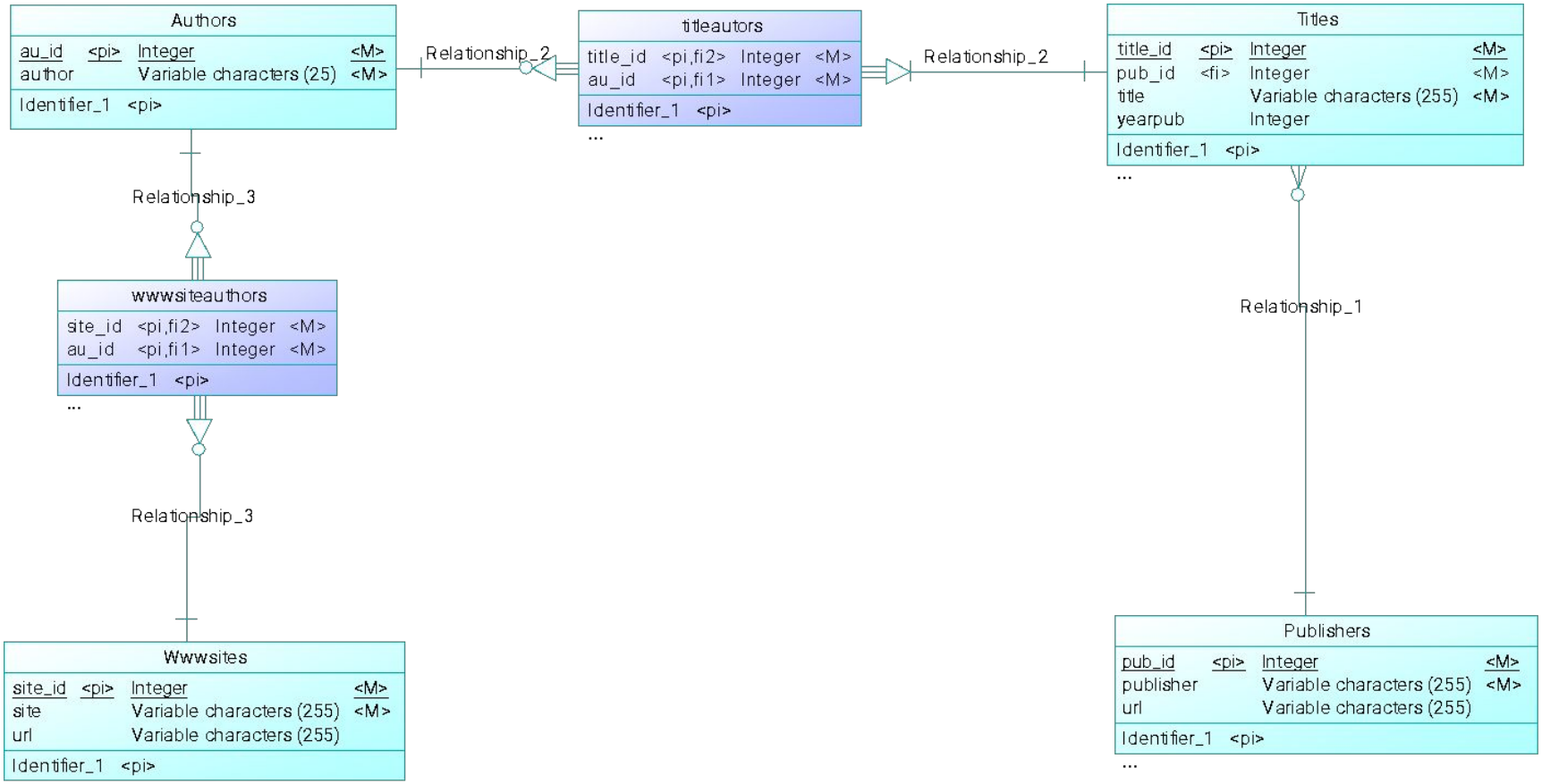
Пример построения моделей БД с использованием CASE системы PowerDesigner

- Концептуальная модель БД



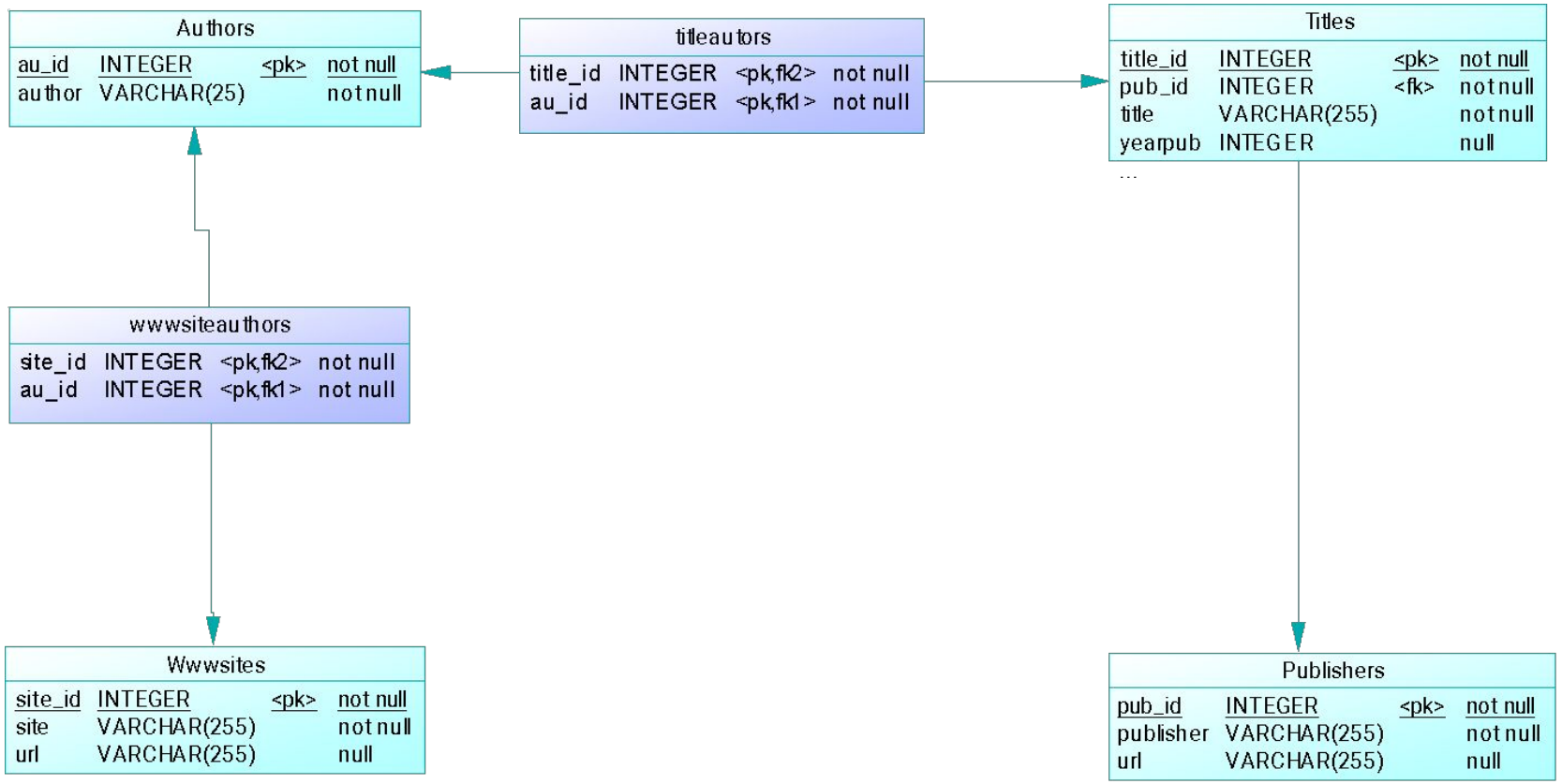
Пример построения моделей БД с использованием CASE системы PowerDesigner

- Логическая модель БД



Пример построения моделей БД с использованием CASE системы PowerDesigner

- Физическая модель БД



Пример построения моделей БД с использованием CASE системы PowerDesigner

- Скрипт создания БД на языке SQL

```
/*==============================================================*/
/* DBMS name:      InterBase 6.x                                */
/* Created on:     17.12.2013 20:46:59                          */
/*==============================================================*/


drop table Authors;

drop table Publishers;

drop table Titles;

drop table Wwwsites;

drop table titleautors;

drop table wwwsiteauthors;
```

Пример построения моделей БД с использованием CASE системы PowerDesigner

```
/*==============================================================*/
/* Table: Authors                                               */
/*==============================================================*/
create table Authors (
au_id                INTEGER                        not null,
author               VARCHAR(25)                    not null,
constraint PK_AUTHORS primary key (au_id)
);

/*==============================================================*/
/* Table: Publishers                                            */
/*==============================================================*/
create table Publishers (
pub_id               INTEGER                        not null,
publisher            VARCHAR(255)                   not null,
url                  VARCHAR(255),
constraint PK_PUBLISHERS primary key (pub_id)
);
```

Пример построения моделей БД с использованием CASE системы PowerDesigner

```
/*==============================================================*/
/* Table: Titles                                                */
/*==============================================================*/
create table Titles (
title_id              INTEGER                        not null,
pub_id                INTEGER                        not null,
title                 VARCHAR(255)                   not null,
yearpub               INTEGER,
constraint PK_TITLES primary key (title_id)
);

/*==============================================================*/
/* Table: Wwwsites                                              */
/*==============================================================*/
create table Wwwsites (
site_id               INTEGER                        not null,
site                  VARCHAR(255)                   not null,
url                   VARCHAR(255),
constraint PK_WWWSITES primary key (site_id)
);
```

Пример построения моделей БД с использованием CASE системы PowerDesigner

```
/*==============================================================*/
/* Table: titleautors                                           */
/*==============================================================*/
create table titleautors (
title_id             INTEGER                        not null,
au_id                INTEGER                        not null,
constraint PK_TITLEAUTORS primary key (title_id, au_id)
);

/*==============================================================*/
/* Table: wwwsiteauthors                                        */
/*==============================================================*/
create table wwwsiteauthors (
site_id              INTEGER                        not null,
au_id                INTEGER                        not null,
constraint PK_WWWSITEAUTHORS primary key (site_id, au_id)
);

alter table Titles
   add constraint FK_TITLES_RELATIONS_PUBLISHE foreign key (pub_id)
      references Publishers (pub_id);
```

Пример построения моделей БД с использованием CASE системы PowerDesigner

```
alter table titleautors
   add constraint FK_TITLEAUT_RELATIONS_TITLES foreign key (title_id)
      references Titles (title_id);

alter table titleautors
   add constraint FK_TITLEAUT_RELATIONS_AUTHORS foreign key (au_id)
      references Authors (au_id);

alter table wwwsiteauthors
   add constraint FK_WWWSITEA_RELATIONS_WWWSITES foreign key (site_id)
      references Wwwsites (site_id);

alter table wwwsiteauthors
   add constraint FK_WWWSITEA_RELATIONS_AUTHORS foreign key (au_id)
      references Authors (au_id);
```

Проектирование реляционной базы данных на основе декомпозиции универсального отношения

- Как мы видели из предыдущего материала, проектирование реляционной базы данных фактически сводится к устранению избыточных функциональных зависимостей (а при необходимости избыточных многозначных зависимостей и зависимостей по соединению) из предварительного набора отношений, полученного каким-либо способом (например, из диаграммы сущность связь). В том случае, когда проектируемая база данных сравнительно невелика (общее число атрибутов не превышает 20-30), предварительный набор отношений можно представить в виде одного отношения, называемого *универсальным*. В него включаются все представляющие интерес атрибуты.

Проектирование реляционной базы данных на основе декомпозиции универсального отношения

- В качестве примера построим универсальное отношение для базы данных **publications**:

  PUBLICATIONS(AUTHOR, TITLE, YEARPUB, PUBLISHER, PUBL_URL, SITE, SITE_URL)

-

  здесь:

  AUTHOR - имя автора;

  TITLE - название книги;

  YEARPUB - год издания книги;

  PUBLISHER - наименование издательства;

  PUBL_URL - ссылка на веб-сервер издательства;

  SITE - наименование Internet-ресурса;

  SITE_URL - указатель на Internet-ресурс.

Проектирование реляционной базы данных на основе декомпозиции универсального отношения

- Функциональные зависимости, имеющиеся в полученном отношении, представлены на следующей схеме:

  (1) TITLE --> YEARPUB

  | (2) -----> PUBLISHER --> PUB_URL

  (3) SITE ---> SITE_URL

- Для устранения избыточной функциональной зависимости (3) декомпозируем исходное отношение на два:
  PUBLICATIONS(AUTHOR, TITLE, YEARPUB, PUBLISHER, PUBL_URL, SITE)
  WWWSITES(SITE,SITE_URL)
- Приняв во внимание, что атрибут SITE требует типа данных "строка" и следовательно его использование в качестве первичного ключа не очень удобно, введем в отношении WWWSITES первичный ключ SITE_ID, основанный на целом типе данных. (Такая подстановка, хотя и ведет к избыточности с точки зрения теории, на практике позволяет ускорить обработку данных. Поэтому, в дальнейшем примем за правило заменять подобным образом строковые первичные ключи, не оговаривая это в каждом отдельном случае).

Проектирование реляционной базы данных на основе декомпозиции универсального отношения

- Теперь наши отношения примут вид:

  PUBLICATIONS(AUTHOR, TITLE, YEARPUB, PUBLISHER, PUBL_URL, SITE_ID)

  WWWSITES(SITE_ID,SITE,SITE_URL)
- Устраним функциональную зависимость (2):

  PUBLICATIONS(AUTHOR, TITLE, YEARPUB, PUB_ID, SITE_ID)
  PUBLISHERS(PUB_ID,PUBLISHER,PUBL_URL)
  WWWSITES(SITE_ID,SITE,SITE_URL)
- Теперь мы имеем следующие избыточные функциональные зависимости в отношении PUBLICATIONS:

  TITLE --> YEARPUB

  | -----> PUB_ID
- Для их устранения необходимо вынести атрибуты TITLE, YEARPUB и PUB_ID в отдельное отношение:

Проектирование реляционной базы данных на основе декомпозиции универсального отношения

- PUBLICATIONS(AUTHOR, TITLE_ID, SITE_ID)
  TITLES(TITLE_ID,TITLE,YEARPUB,PUB_ID)
  PUBLISHERS(PUB_ID,PUBLISHER,PUBL_URL)
  WWWSITES(SITE_ID,SITE,SITE_URL)
- Теперь наша база данных находится в третьей нормальной форме, однако мы видим, что полученный набор отношений не совпадает с набором, полученным из модели "сущность-связь". Для того, чтобы разобраться в причинах этого противоречия, рассмотрим отношение PUBLICATIONS вместе с его данными. Добавим автора, который имеет две книги и две web-страницы:

AUTHOR	TITLE_ID	SITE_ID
J.Doe	1	1
J.Doe	2	1
J.Doe	1	2
J.Doe	2	2

Проектирование реляционной базы данных на основе декомпозиции универсального отношения

- Из этой таблицы становится ясно, что в рассматриваемом отношении существует многозначная зависимость AUTHOR ->> TITLE_ID | SITE_ID. Для ее устранения приведем отношение к четвертой нормальной форме, для чего разобьем его на три.

  PUBLICATIONS(AUTHOR,TITLE_ID,SITE_ID) ->
  AUTHORS(AU_ID,AUTHOR)
  TITLEAUTHORS(TITLE_ID,AU_ID)
  WWWSITEAUTHORS(AU_ID,SITE_ID)
- Окончательно получим:

  AUTHORS(AU_ID,AUTHOR) TITLEAUTHORS(TITLE_ID,AU_ID)
  WWWSITEAUTHORS(AU_ID,SITE_ID)
  TITLES(TITLE_ID,TITLE,YEARPUB,PUB_ID)
  PUBLISHERS(PUB_ID,PUBLISHER,PUBL_URL)
  WWWSITES(SITE_ID,SITE,SITE_URL)
- Теперь схема базы данных соответствует *структуре, полученной другими способами*. Анализ показывает, что избыточные функциональные зависимости в ней отсутствуют.

Вопросы практического программирования

- В этой главе рассматриваются некоторые способы создания приложений, работающих с базой данных при помощи языка SQL. Как правило, любой поставщик СУБД предоставляет вместе со своей системой внешнюю утилиту, которая позволяет вводить операторы SQL в режиме командной строки и выдает на консоль результаты их выполнения. Недостатки такого режима работы очевидны: необходимо знать SQL, необходимо помнить схему БД, отсутствует возможность удобного просмотра результатов выполнения запросов. Поэтому, подобные утилиты стали инструментами администраторов баз данных, а для создания пользовательских приложений используются универсальные и специализированные языки программирования. Приложения, написанные таким образом, позволяют пользователю сосредоточиться на решении собственных задач, а не на структурах данных.

Вопросы практического программирования

- Почти все способы организации взаимодействия пользователя с базой данных, рассматриваемые ниже, основаны на модели "клиент-сервер". Т.е. предполагается, что каждое приложение обработки данных разбито, как минимум, на две части:

  - клиента, который отвечает за организацию пользовательского интерфейса;

  - сервер, который собственно хранит данные, обрабатывает запросы и посылает их результаты клиенту для отображения.
- При этом предполагается, что каждая часть приложения функционирует на отдельном компьютере, т.е. к выделенному серверу БД с помощью локальной сети подключены персональные компьютеры пользователей (клиенты). Это наиболее популярная сегодня схема организации вычислительной среды. Более подробно архитектура "клиент-сервер" и различные способы ее реализации будут обсуждаться далее.

Вопросы практического программирования

- Язык SQL позволяет только манипулировать данными, но в нем отсутствуют средства создания экранного интерфейса, что необходимо для пользовательских приложений.
- Для создания этого интерфейса служат универсальные языки третьего поколения (C, C++, C#, Pascal) или проблемно-ориентированные языки четвертого поколения (xBase, Informix 4GI, Progress, Jam,...). Эти языки содержат необходимые операторы ввода / вывода на экран, а также операторы структурного программирования (цикла, ветвления и т.д.). Также эти языки допускают определение структур, соответствующих записям таблиц обрабатываемой базы данных.
- В исходный текст программы включаются операторы языка SQL, которые во время исполнения передаются серверу БД, который собственно и производит манипулирование данными.

Вопросы практического программирования

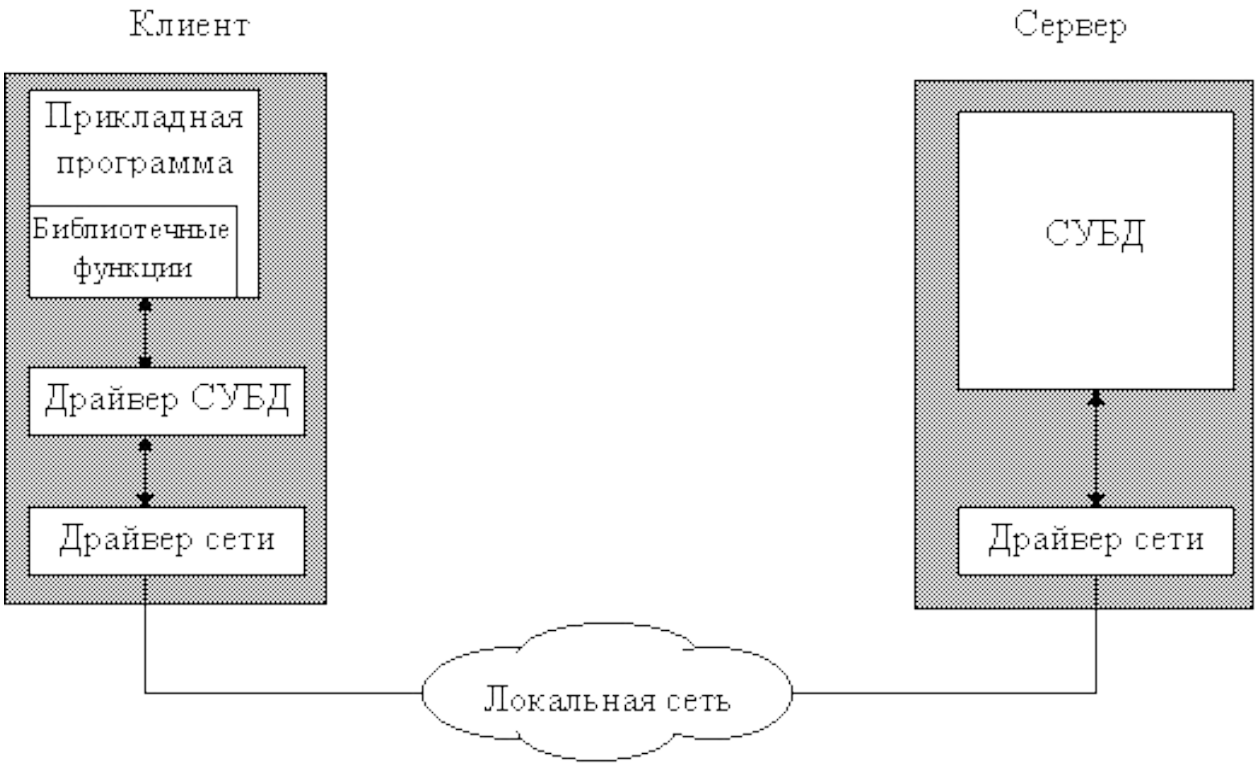
- Отношения, полученные в результате выполнения сервером SQL-запросов, возвращаются прикладной программе, которая заполняет строками этих отношений заранее определенные структуры. Дальнейшая работа клиентской программы (отображение, корректировка записей) ведется с этими структурами.
- Рассмотрим различные способы организации доступа прикладной программы к серверу базы данных.

Использование специализированных библиотек и встраиваемого SQL

- Каждая СУБД помимо интерактивной SQL-утилиты обязательно имеет библиотеку доступа и набор драйверов для различных операционных систем.

Вопросы практического программирования

- Схема взаимодействия клиентского приложения с сервером базы данных в этом случае выглядит так:



Вопросы практического программирования

- Библиотека доступа - это, как правило, объектный файл, исходный код которого создан на универсальном языке типа *C*. Эта библиотека содержит набор функций, позволяющих пользовательскому приложению соединятся с базой данных, передавать запросы серверу и получать ответные данные. Типичный набор функций такой библиотеки (имена функций зависят от используемой библиотеки):
- **DB_connect(char *имя_базы_данных, char *имя_пользователя, char *пароль)** - устанавливает соединение с базой данной, возвращает указатель на структуру **db**, описывающую характеристики этого соединения;
- **DB_exec(db, char *запрос)** - выполнить запрос к базе данных, определяемой структурой **db**. Применяется для любых запросов кроме SELECT. Возвращает код выполнения запроса (0 - удачно, либо код ошибки);

Вопросы практического программирования

- **DB_select(db, char *запрос)** - выполнить запрос на извлечение данных (SELECT). Возвращает структуру **result**, содержащую результаты выполнения запроса (реляционное отношение).
- **DB_fetch(result)** - извлечь следующую запись из структуры **result**.
- **DB_close(db)** - закрыть соединение с базой данных.
- Разумеется это минимальный набор функций для работы с базой данных. Обычно в библиотеке присутствуют также функции, позволяющие определить характеристики структуры **result** (число, порядок и имена столбцов, число строк, номер текущей строки), передвигаться по этой структуре не только вперед, но и назад (**DB_next**, **DB_prev**) и т.д. Пример программы, использующей библиотеку связи с базой данных:

Вопросы практического программирования

```
#include <dblib.h>     /* Файл, содержащий описание функций библиотеки */
........
/* Организация интерфейса с пользователем, запрос его имени и пароля */
/* Присвоение значений переменным:  dbname - имя базы данных         */
/*                                  username - имя пользователя      */
/*                                  password - пароль                */
.........
db=DB_connect(dbname,username,password); /* Установление соединения */
if (db == NULL) {
   error_message();  /* Выдача сообщения об ошибке на монитор пользователя */
   exit(1);          /* Завершение работы */
}
..........
/* Ожидание запроса пользователя. Формирование строки s_query - запроса */
/* на выборку данных                                                    */
..........
result=DB_select(db,s_query);     /* Пересылка запроса на сервер */
if (result==NULL) {
   error_message();  /* Ошибка выполнения запроса. Выдача сообщения */
   exit(2);          /* Завершение работы */
}
..........
```

Вопросы практического программирования

```
..........
/* Вывод результатов запроса на монитор пользователя. Ожидание следующего  */
/* запроса. Подготовка строки u_query="UPDATE ... SET ...", содержащей     */
/* запрос на изменение данных.                                              */
..........
res=DB_exec(db,u_query);      /* Пересылка запроса на сервер */
if (res != 0 ) {
error_message();  /* Ошибка выполнения запроса. Выдача сообщения */
   exit(2);          /* Завершение работы */    }
..........
..........
DB_close(db);        /*Завершение работы */
```

Вопросы практического программирования

- Данная программа, обеспечивающая взаимодействие пользователя с СУБД, компилируется совместно с библиотекой доступа. Библиотечные вызовы преобразуются драйвером базы данных в сетевые вызовы и передаются сетевым программным обеспечением на сервер.
- На сервере происходит обратный процесс преобразования: сетевые пакеты -> функции библиотеки -> SQL-запросы, запросы обрабатываются, их результаты передаются клиенту.
- Как видим, такой способ создания приложений чрезвычайно гибок, позволяет реализовать практически любое приложение, но в то же время имеет явные недостатки:

  - разработка клиентской программы возможна только для той операционной системы и на том языке программирования, который поддерживается библиотекой;

  - необходим драйвер базы данных, который определяет допустимые типы сетевых интерфейсов;

Вопросы практического программирования

- большой объем кодирования;

- не стандартизованные библиотечные функции.

- В результате получаем приложение, которое привязано как к сетевой среде, так и к программно-аппаратной платформе и используемой базе данных.
- Некоторой модификацией данного способа является использование "встроенного" языка SQL. В этом случае в текст программы на языке третьего поколения включаются не вызовы библиотек, а непосредственно предложения SQL, которые предваряются ключевым выражением "EXEC SQL". Перед компиляцией в машинный код такая программа обрабатывается препроцессором, который транслирует смесь операторов "собственного" языка СУБД и операторов SQL в "чистый" исходный код. Затем коды SQL замещаются вызовами соответствующих процедур из библиотек исполняемых модулей, служащих для поддержки конкретного варианта СУБД.

Вопросы практического программирования

- Такой подход позволил несколько снизить степень привязанности к СУБД, например, при переключении прикладной программы на работу с другим сервером базы данных достаточно было заново обработать ее исходный текст новым препроцессором и перекомпилировать.

CLI - интерфейс уровня вызовов

- Большим достижением явилось появление (1994 г.) в стандарте SQL интерфейса уровня вызова - CLI (Call Level Interface), в котором стандартизован общий набор рабочих процедур, обеспечивающий совместимость со всеми основными серверами баз данных. Ключевой элемент CLI - специальная библиотека для компьютера-клиента, в которой хранятся вызовы процедур и большинство часто используемых сетевых компонентов для организации связи с сервером. Это ПО поставляется разработчиком средств SQL, не является универсальным и поддерживает разнообразные транспортные протоколы.

Вопросы практического программирования

- Использование программных вызовов позволяет свести к минимуму операции на компьютере-клиенте. В общем случае клиент формирует оператор языка SQL в виде строки и пересылает ее на сервер посредством процедуры исполнения (execute). Когда же сервер в качестве ответа возвращает несколько строк данных, клиент считывает результат с помощью серии вызовов процедуры выборки данных. Далее информация из столбцов полученной таблицы может быть связана с соответствующими переменными приложения. Вызов специальной процедуры позволяет клиенту определить считанное число строк, столбцов и типы данных в каждом столбце.
- Интерфейс CLI построен таким образом, что перед передачей запроса серверу клиент не должен заботиться о типе оператора SQL, будь то выборка, обновление, удаление или вставка.

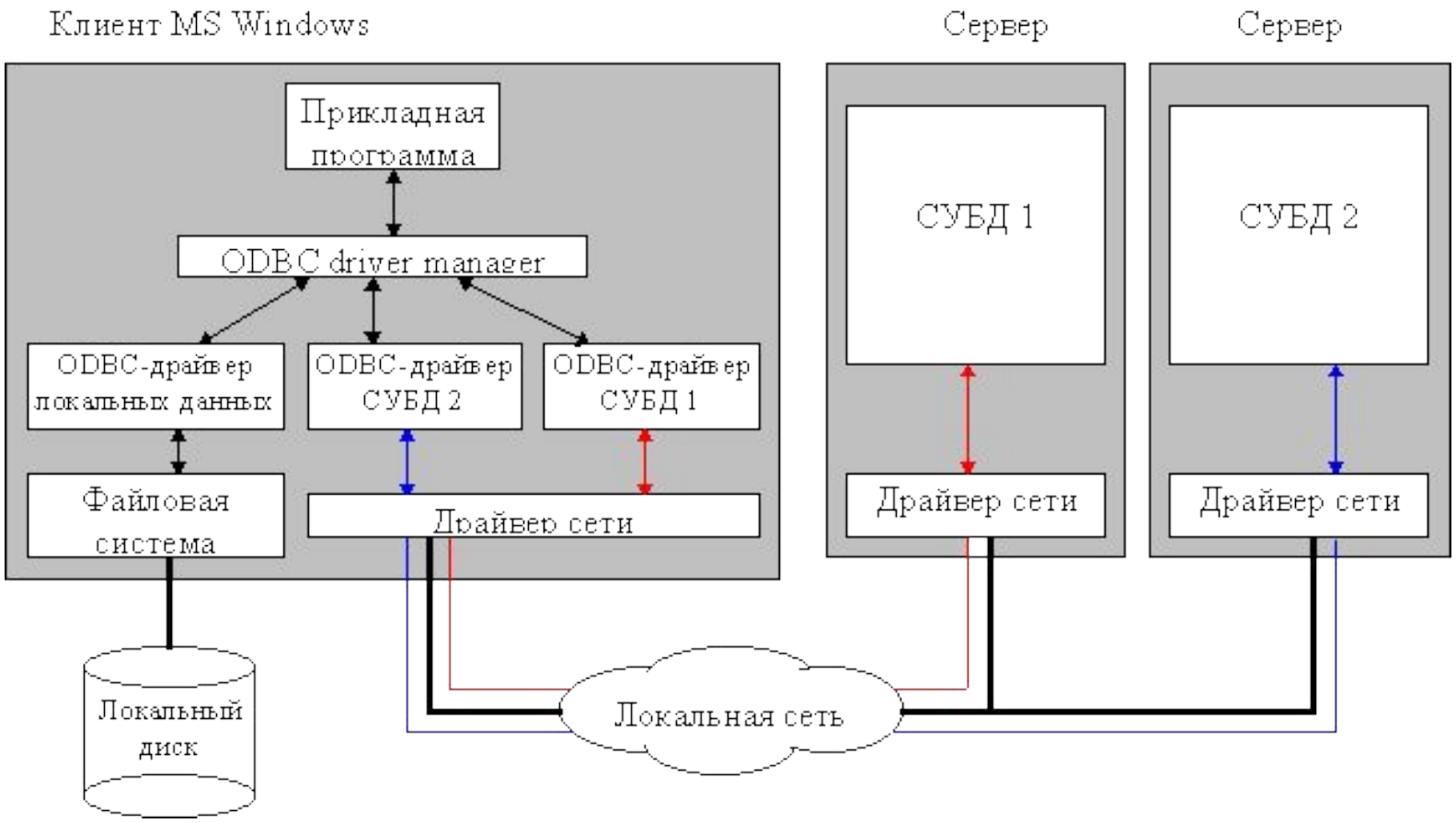
Вопросы практического программирования

ODBC - открытый интерфейс к базам данных на платформе MS Windows

- Очень важный шаг к созданию переносимых приложений обработки данных сделала фирма Microsoft, опубликовавшая в 1992 году спецификацию ODBC (Open Database Connetcivity - открытого интерфейса к базам данных), предназначенную для унификации доступа к данным с персональных компьютеров работающих под управлением операционной системы Windows. (Заметим, что ODBC опирается на спецификации CLI).

Вопросы практического программирования

- Структурная схема доступа к данным с использованием ODBC:



Вопросы практического программирования

- ODBC представляет собой программный слой, унифицирующий интерфейс приложений с базами данных. За реализацию особенностей доступа к каждой отдельной СУБД отвечает специальный ODBC-драйвер. Пользовательское приложение этих особенностей не видит, т.к. взаимодействует с универсальным программным слоем более высокого уровня. Таким образом, приложение становится в значительной степени независимым от СУБД. Однако, этот способ также не лишен недостатков:

  - приложения становятся привязанными к платформе MS Windows;

  - увеличивается время обработки запросов (как следствие введения дополнительного программного слоя);

  - необходимо предварительная инсталляция ODBC-драйвера и настройка ODBC (указание драйвера, сетевого пути к серверу, базы данных и т.д.) на каждом рабочем месте. Параметры этой настройки являются статическими, т.е. приложение их самостоятельно изменить не может.

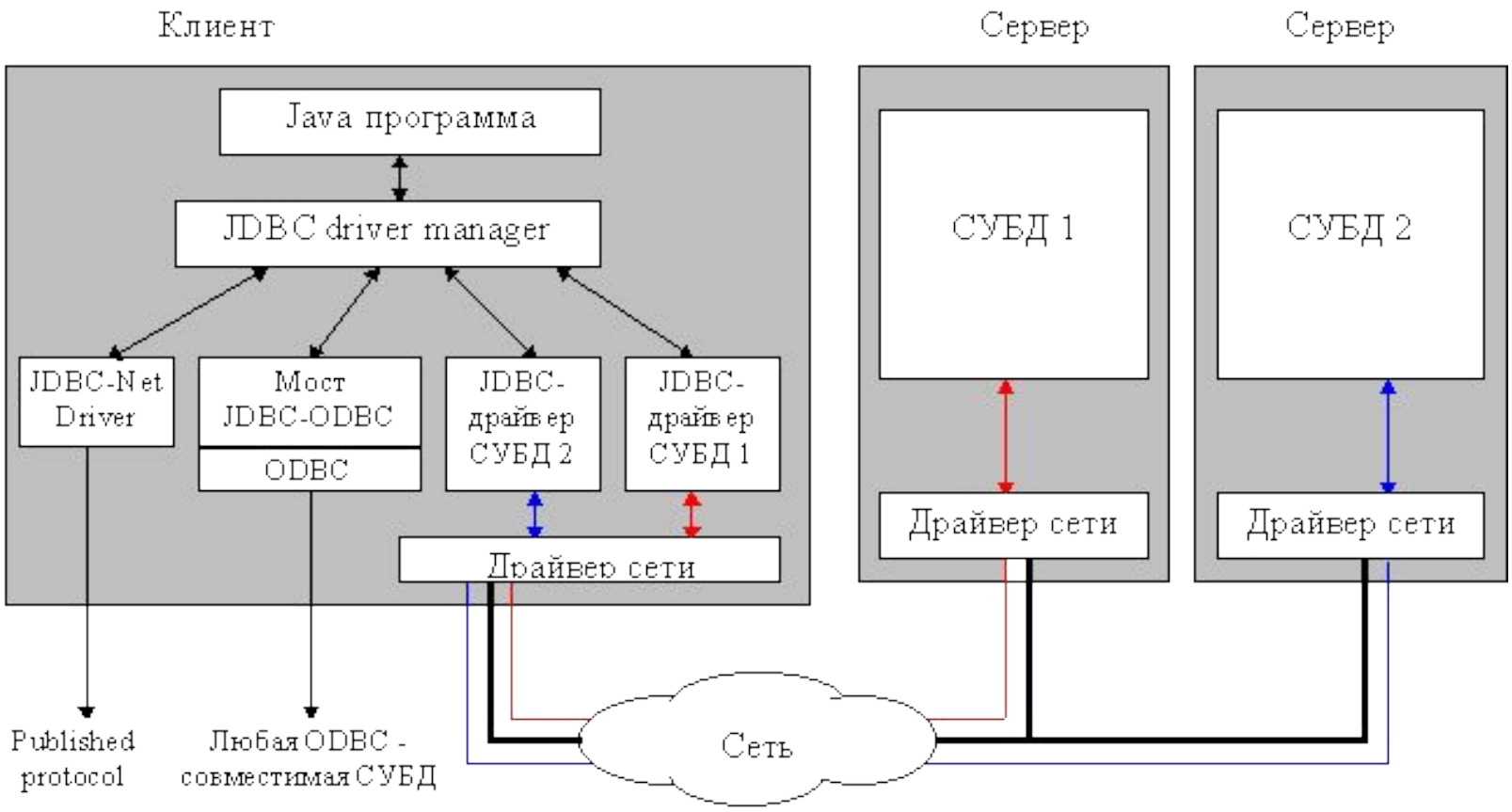
Вопросы практического программирования

JDBC - мобильный интерфейс к базам данных на платформе Java.

- JDBC (Java DataBase Connectivity) - это интерфейс прикладного программирования (API) для выполнения SQL-запросов к базам данных из программ, написанных на языке Java. Напомним, что язык Java, созданный компанией Sun, является платформо - независимым и позволяет создавать как собственно приложения (standalone application), так и программы (апплеты), встраиваемые в web-страницы.

Вопросы практического программирования

- Структурная схема доступа к данным с использованием JDBC:



Вопросы практического программирования

- JDBC во многом подобен ODBC (см. рисунок), также построен на основе спецификации CLI, однако имеет ряд замечательных отличий. Во-первых, приложение загружает JDBC-драйвер динамически, следовательно администрирование клиентов упрощается, более того, появляется возможность переключаться на работу с другой СУБД без перенастройки клиентского рабочего места. Во-вторых, JDBC, как и Java в целом, не привязан к конкретной аппаратной платформе, следовательно проблемы с переносимостью приложений практически снимаются. В-третьих, использование Java-приложений и связанной с ними идеологии "тонких клиентов" обещает снизить требования к оборудованию клиентских рабочих мест.

Архитектура "клиент-сервер"

- **Основные понятия.**
- Как правило, компьютеры и программы, входящие в состав информационной системы, не являются равноправными. Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам. Компьютер (или программу), управляющий ресурсом, называют ***сервером*** этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер...). Клиент и сервер какого-либо ресурса могут находится как в рамках одной вычислительной системы, так и на различных компьютерах, связанных сетью.

Архитектура "клиент-сервер"

- Основной принцип технологии "клиент-сервер" заключается в разделении функций приложения на три группы:

  - ввод и отображение данных (взаимодействие с пользователем);

  - прикладные функции, характерные для данной предметной области;

  - функции управления ресурсами (файловой системой, базой данных и т.д.).

- Поэтому, в любом приложении выделяются следующие компоненты:

  - компонент представления данных;

  - прикладной компонент;

  - компонент управления ресурсом.

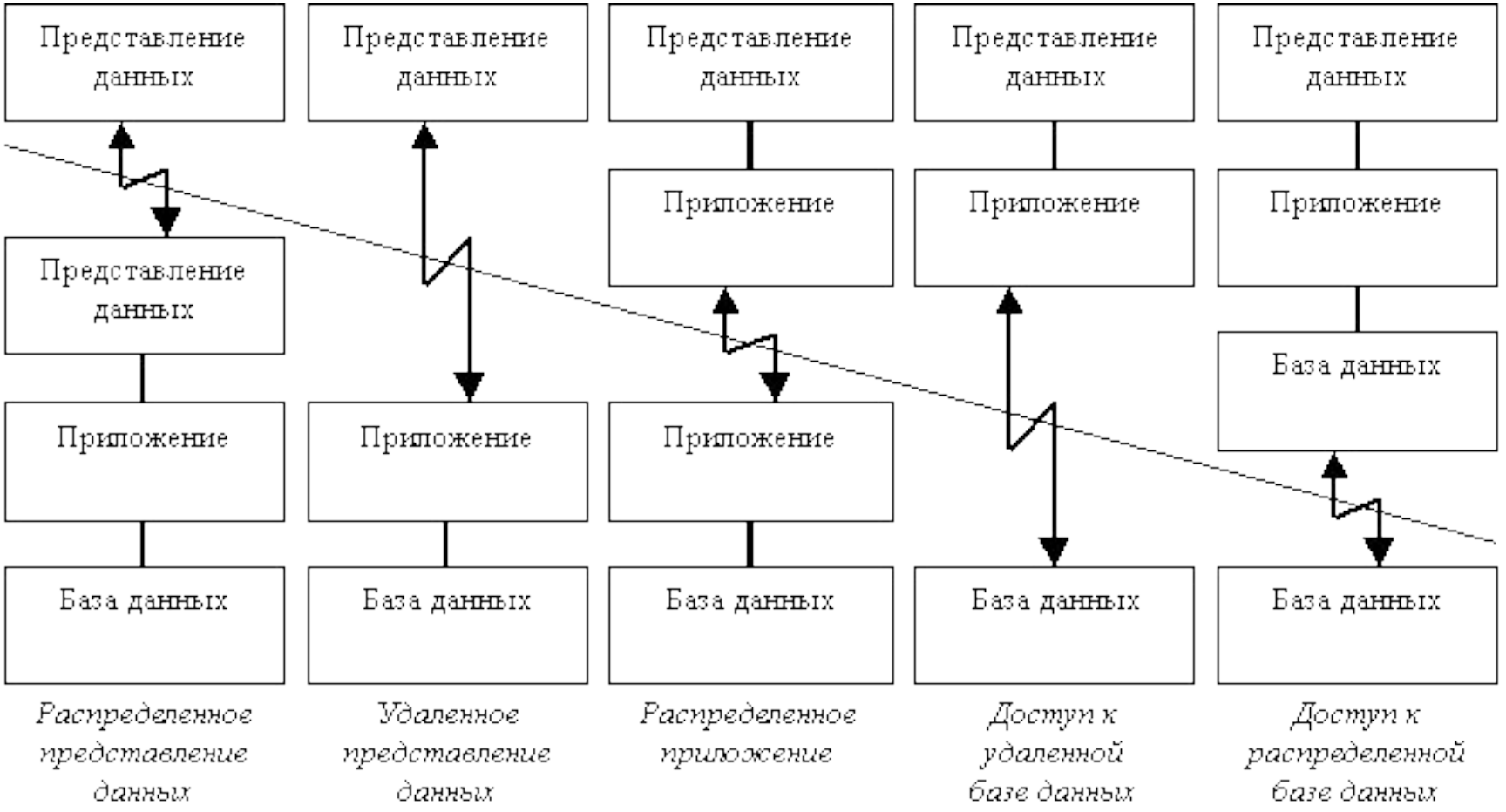
- Связь между компонентами осуществляется по определенным правилам, которые называют "протокол взаимодействия".

Архитектура "клиент-сервер"

- **Модели взаимодействия клиент-сервер.**
- Компанией Gartner Group, специализирующейся в области исследования информационных технологий, предложена следующая классификация двухзвенных моделей взаимодействия клиент-сервер (двухзвенными эти модели называются потому, что три компонента приложения различным образом распределяются между двумя узлами):

Архитектура "клиент-сервер"

- **Модели взаимодействия клиент-сервер**



Архитектура "клиент-сервер"

- Исторически первой появилась модель *распределенного представления данных*, которая реализовывалась на универсальной ЭВМ с подключенными к ней неинтеллектуальными терминалами. Управление данными и взаимодействие с пользователем при этом объединялись в одной программе, на терминал передавалась только "картинка", сформированная на центральном компьютере.
- Затем, с появлением персональных компьютеров (ПК) и локальных сетей, были реализованы модели *доступа к удаленной базе данных*. Некоторое время базовой для сетей ПК была архитектура файлового сервера. При этом один из компьютеров является файловым сервером, на клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программа). Протокол обмена при этом представляет набор низкоуровневых вызовов операций файловой системы. Такая архитектура, реализуемая, как правило, с помощью персональных СУБД, имеет очевидные недостатки - высокий сетевой трафик и отсутствие унифицированного доступа к ресурсам.

Архитектура "клиент-сервер"

- С появлением первых специализированных серверов баз данных появилась возможность другой реализации модели доступа к удаленной базе данных. В этом случае ядро СУБД функционирует на сервере, протокол обмена обеспечивается с помощью языка SQL. Такой подход по сравнению с файловым сервером ведет к уменьшению загрузки сети и унификации интерфейса "клиент-сервер". Однако, сетевой трафик остается достаточно высоким, кроме того, по прежнему невозможно удовлетворительное администрирование приложений, поскольку в одной программе совмещаются различные функции.
- Позже была разработана концепция активного сервера, который использовал механизм хранимых процедур. Это позволило часть прикладного компонента перенести на сервер (*модель распределенного приложения*). Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на том же компьютере, что и SQL-сервер.

Архитектура "клиент-сервер"

- Преимущества такого подхода: возможно централизованное администрирование прикладных функций, значительно снижается сетевой трафик (т.к. передаются не SQL-запросы, а вызовы хранимых процедур). Недостаток - ограниченность средств разработки хранимых процедур по сравнению с языками общего назначения (C и Pascal).
- На практике сейчас обычно используются смешанный подход:

  - простейшие прикладные функции выполняются хранимыми процедурами на сервере;

  - более сложные реализуются на клиенте непосредственно в прикладной программе.
- Сейчас ряд поставщиков коммерческих СУБД объявило о планах реализации механизмов выполнения хранимых процедур с использованием языка Java. Это соответствует концепции "тонкого клиента", функцией которого остается только отображение данных (*модель удаленного представления данных*).

Архитектура "клиент-сервер"

- В последнее время также наблюдается тенденция ко все большему использованию модели распределенного приложения. Характерной чертой таких приложений является логическое разделение приложения на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная архитектура клиент-сервер становится трехзвенной, а к некоторых случаях, она может включать и больше звеньев.

Структура сервера базы данных

- В заключение рассмотрим физическую организацию сервера базы данных. Как правило, он включает следующие компоненты:

  - подсистема взаимодействия с клиентским приложением
  Данный модуль отвечает за поддержание связи с клиентом. Как правило, механизм его работы выглядит следующим образом. Подсистема взаимодействия "прослушивает" сеть в ожидании клиентских запросов на установление соединения. Когда такой запрос обнаруживается, порождается новый процесс, который будет обеспечивать связь с данным клиентом. Клиенту сообщается идентификатор данного процесса, в дальнейшем клиент передает свои запросы и получает данные взаимодействуя с этим интерфейсным процессом. После того, как клиент закрывает соединение, обслуживавший его процесс прекращается. Характеристики интерфейсных процессов зависят от операционной системы, под которой исполняется сервер базы данных.

Структура сервера базы данных

- подсистема синтаксического разбора запросов

Данный модуль отвечает за компиляцию поступающих от клиентов через интерфейсные процессы запросов во внутренний код, который будет исполняться сервером. При ошибках компиляции соответствующие сообщения передаются клиенту. Наиболее современные СУБД позволяют сохранять откомпилированный код запросов некоторое время. Это позволяет избежать стадии компиляции при повторном обращении клиента к запросу.

Структура сервера базы данных

- подсистема планирования выполнения запросов

Данный модуль должен составить такой план выполнения запроса, чтобы он был обработан наиболее быстро. Для этого анализируются условия выборок и соединений, устанавливается порядок их выполнения. Пусть, например, надо извлечь одного сотрудника из списка работников, в качестве критерия поиска задаются его имя и фамилия. Возможны два плана выполнения запроса: (1) вначале делается выборка всех сотрудников с данным именем, из нее извлекаются записи, содержащие данную фамилию; (2) - наоборот, вначале делается выборка по фамилии, затем по имени. Поскольку множество имен, как правило, меньше множества фамилий, во втором случае запрос будет обработан быстрее, т.к. на втором этапе здесь мы получим меньшую выборку. Планировщики запросов ведущих СУБД отслеживают информацию о распределении значений в таблицах. План выполнения запроса включается в его откомпилированный код.

Структура сервера базы данных

- подсистема выполнения транзакций

Здесь выполняется оптимизированный код запроса, обновляются индексы, выполняются в случае необходимости триггеры и хранимые процедуры. Как правило, несколько запросов могут исполняться параллельно, при этом обеспечивается необходимый уровень их изоляции. Также ведется журнал транзакций, обеспечивается их завершение и корректный откат.

- подсистема управления памятью

Этот компонент отвечает за считывание данных с диска в оперативную память, синхронизацию обновлений с данными на диске и т.д. Он может использовать файловые функции операционной системы, но часто СУБД имеет свои собственные низкоуровневые средства доступа к дискам.