Содержание

1. Данные, сущности, атрибуты, базы данных.

Часто, говоря о базе данных, имеют в виду просто некоторое автоматизированное хранилище данных. Такое представление не вполне корректно.

Действительно, **в узком смысле слова, база данных** - это некоторый набор данных, необходимых для работы (актуальные данные).

Однако данные - это абстракция; никто никогда не видел "просто данные"; они не возникают и не существуют сами по себе. **Данные** суть отражение объектов реального мира.

Пусть, например, требуется хранить сведения о деталях, поступивших на склад. Как объект реального мира - деталь - будет отображена в базе данных?

Для того, чтобы ответить на этот вопрос, необходимо знать, какие **признаки** или стороны детали будут **актуальны, необходимы для работы**. Среди них могут быть название детали, ее вес, размер, цвет, дата изготовления, материал, из которого она сделана и т.д.

В традиционной терминологии объекты реального мира, сведения о которых хранятся в базе данных, называются **сущностями** (entities), а их актуальные признаки - **атрибутами** (attributes).

Каждый признак конкретного объекта есть **значение атрибута**. Так, деталь "двигатель" имеет значение атрибута "вес", равное "50", что отражает тот факт, что данный двигатель весит 50 килограммов.

Таким образом, **в широком смысле слова база данных** - это совокупность описаний объектов реального мира и связей между ними, актуальные для конкретной прикладной области. В дальнейшем мы будем исходить из этого определения, которое будет уточняться по ходу изложения.

База данных (БД) - совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными.

Важнейшая задача компьютерных систем - хранение и обработка данных. Для ее решения были предприняты усилия, которые привели к появлению в конце 60-х - начале 70-х годов специализированного программного обеспечения – СУБД - систем управления базами данных (DataGBase Management Systems - DBMS).

СУБД позволяют структурировать, систематизировать и организовать данные для их компьютерного хранения и обработки. Сегодня невозможно представить себе деятельность любого современного предприятия или организации без использования профессиональных СУБД. Несомненно, они составляют фундамент информационной деятельности во всех сферах - начиная с производства и заканчивая финансами и телекоммуникациями.

Системой управления базой данных называется комплекс программ, обеспечивающих централизованное хранение, накопление, модификацию и выдачу данных, входящих в БД.

2. Типы данных.

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

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

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

- Простые типы данных.
- Структурированные типы данных.
- Ссылочные типы данных.

Простые типы данных.

Простые, или атомарные, типы данных не обладают внутренней структурой. Данные такого типа называют скалярами. К простым типам данных относятся следующие типы:

- Логический.
- Строковый.
- Численный.

Различные языки программирования могут расширять и уточнять этот список, добавляя такие типы как:

- Целый.
- Вещественный.
- Дата.
- Время.
- Денежный.
- Перечислимый.
- Интервальный.
- И т.д.…

Конечно, понятие атомарности довольно относительно. Так, строковый тип данных можно рассматривать как одномерный массив символов, а целый тип данных - как набор битов. Важно лишь то, что при переходе на такой низкий уровень теряется семантика (смысл) данных. Если строку, выражающую, например, фамилию сотрудника, разложить в массив символов, то при этом теряется смысл такой строки как единого целого.

Структурированные типы данных

Структурированные типы данных предназначены для задания сложных структур данных. Структурированные типы данных конструируются из составляющих элементов, называемых компонентами, которые, в свою очередь, могут обладать структурой. В качестве структурированных типов данных можно привести следующие типы данных:

- Массивы
- Записи (Структуры)

С математической точки зрения **массив** представляет собой функцию с конечной областью определения. Например, рассмотрим конечное множество натуральных чисел

$$A = \{1,2,\ldots,n\}$$

называемое множеством индексов. Отображение

$$F : A \to \mathrm{R}$$

из множества A во множество вещественных чисел R задает одномерный вещественный массив. Значение этой функции для некоторого значения индекса i называется элементом массива, соответствующим i. Аналогично можно задавать многомерные массивы.

Запись (или структура) представляет собой кортеж из некоторого декартового произведения множеств. Действительно, запись представляет собой именованный упорядоченный набор элементов r_i, каждый из которых принадлежит типу T_i. Таким образом, запись $r = (r_1, r_2, \ldots, r_n)$ есть элемент множества $T = T_1 \times T_2 \times \ldots \times T_n$. Объявляя новые типы записей на основе уже имеющихся типов, пользователь может конструировать сколь угодно сложные типы данных.

Общим для структурированных типов данных является то, что они имеют внутреннюю структуру, используемую на том же уровне абстракции, что и сами типы данных.

Поясним это следующим образом. При работе с массивами или записями можно манипулировать массивом или записью и как с единым целым (создавать, удалять, копировать

целые массивы или записи), так и поэлементно. Для структурированных типов данных есть специальные функции - конструкторы типов, позволяющие создавать массивы или записи из элементов более простых типов.

Работая же с простыми типами данных, например с числовыми, мы манипулируем ими как неделимыми целыми объектами. Чтобы "увидеть", что числовой тип данных на самом деле сложен (является набором битов), нужно перейти на более низкий уровень абстракции. На уровне программного кода это будет выглядеть как ассемблерные вставки в код на языке высокого уровня или использование специальных побитных операций.

Ссылочные типы данных

Ссылочный тип данных (указатели) предназначен для обеспечения возможности указания на другие данные. Указатели характерны для языков процедурного типа, в которых есть понятие области памяти для хранения данных. Ссылочный тип данных предназначен для обработки сложных изменяющихся структур, например деревьев, графов, рекурсивных структур.

3. Архитектуры баз данных

Централизованная база данных.

При использовании этой технологии база данных располагается на одном компьютере, который может даже не иметь поддержки сети и работать автономно. В этом случае работа с базой данных возможна только локально. Если же компьютер работает в сети, то доступ к информации может осуществляться удаленно с других компьютеров сети. Централизованные базы данных с распределенным доступом являются наиболее используемыми в настоящее время. Для этой технологии возможны два способа обработки данных:

Файл-сервер.

Эта архитектура централизованных баз данных с сетевым доступом предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы централизованной базы данных. В соответствии с запросами пользователей файлы с файл-сервера передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных. После завершения работы пользователи копируют файлы с обработанными данными обратно на сервер, откуда их смогут взять и обработать другие пользователи. На основе скопированных файлов централизованной базы данных пользователи могут создавать на своих рабочих станциях локальные базы данных.

Недостатки такой организации работы очевидны. При одновременном обращении множества пользователей к одним и тем же данным производительность работы резко падает, т. к. необходимо дождаться, пока пользователь, работающий с данными, завершит свою работу. В противном случае возможно затирание исправлений, сделанных одними пользователями, изменениями других пользователей.

Клиент-сервер.

В основе этой концепции лежит идея о том, что помимо хранения файлов базы данных, центральный сервер должен выполнять и основную часть обработки данных. Пользователи обращаются к центральному серверу с помощью специального языка структурированных запросов (SQL, Structured Query Language), на котором описывается список задач, выполняемых сервером. Запросы пользователей принимаются сервером и порождают на нем процессы обработки данных. В ответ пользователь получает уже обработанный набор данных.

Между клиентом и сервером передается не весь набор данных, как это происходит в технологии файл-сервер, а только данные, которые действительно необходимы пользователю. Запрос пользователя длиной всего в несколько строк способен породить процесс обработки данных, затрагивающий множество таблиц и миллионы строк. В ответ клиент может получить лишь несколько чисел.

Технология клиент-сервер позволяет избежать передачи по сети огромных объемов информации, переложив всю обработку данных на центральный сервер. Кроме того, рассматриваемый подход позволяет избежать конфликтов изменений одних и тех же данных множеством пользователей, которые характерны для технологии файл-сервер. Технология

клиент-сервер реализует согласованное изменение данных множеством пользователей, обеспечивая автоматическое соблюдение целостности данных.

Трехуровневая архитектура предполагает наличие еще одного звена – сервера приложений (application server).

Эти и некоторые другие преимущества сделали технологию клиент-сервер весьма популярной.

К недостаткам технологии клиент-сервер можно отнести высокие требования к производительности центрального сервера. Чем больше пользователей обращаются к данным, и чем больше объем обрабатываемых данных, тем более мощным должен быть центральный сервер.

Распределенная база данных.

Базы данных этого типа располагаются на нескольких компьютерах. Информация на этих компьютерах может пересекаться и даже дублироваться. Для управления подобными базами данных предназначена система управления распределенными базами данных (СУРБД). Работа с распределенной базой данных может быть прозрачна для пользователей. Система скрывает от клиентов обращения к данным, расположенным на других компьютерах. Для пользователя все выглядит так, как будто вся информация находится на одном сервере.

Распределенные базы данных пока не получили большого распространения.

Data Warehouses и Data Mining.

В различных подразделениях крупных компаний могут использоваться различные базы данных. Строгую централизацию БД проводить не всегда приемлемо.

Специализированные разработки слишком дорого переделывать; при укрупнении компаний в распоряжение компании поступают сторонние базы данных. Исходя из этих и других причин унаследованные базы данных не могут (и не должны) непосредственно заменяться единой центральной базой данных компании. Более разумно поверх существующих решений построить новую информационную структуру, способную представить информацию всех подразделений в удобном виде.

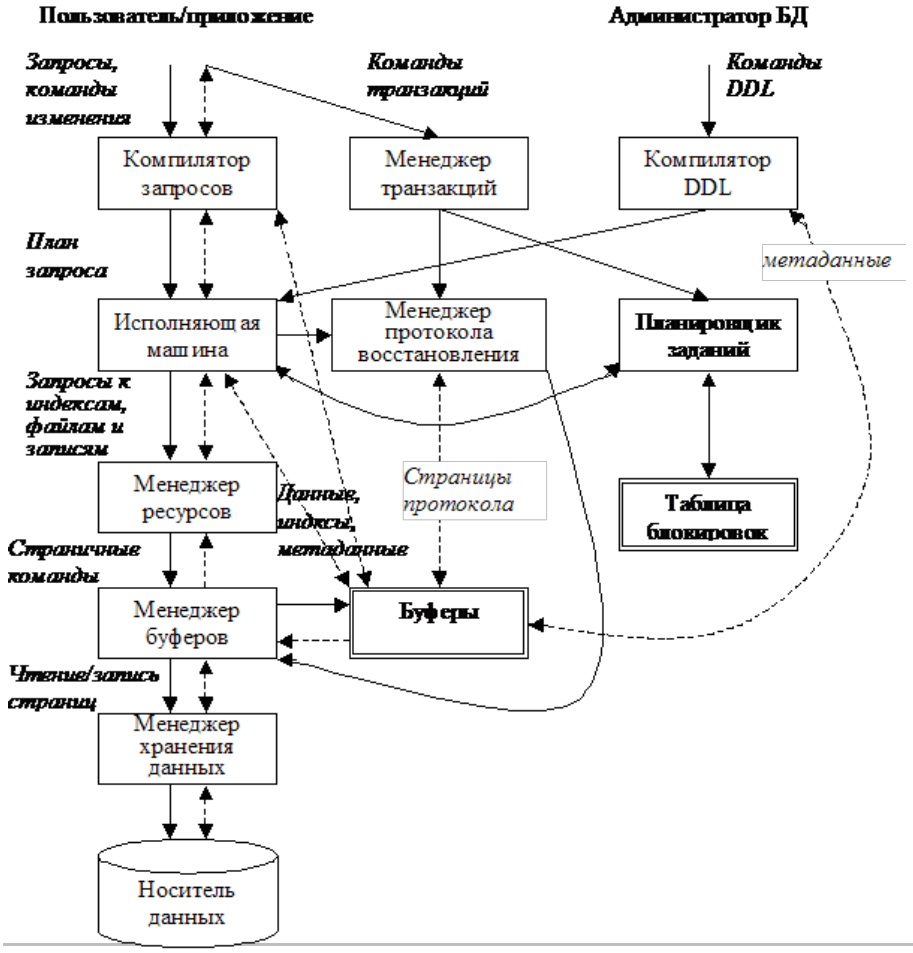
Одно из популярных решений такой задачи связано с использованием технологии хранилищ данных (**data warehouses**), которые предполагают копирование информации из унаследованных БД с соответствующей трансляцией и последующим сохранением в центральной БД. После внесения изменений в унаследованной базе данных необходимые исправления вносятся и в содержимое хранилища, хотя не обязательно автоматически и немедленно. Репликация данных обычно производится ночью, когда нагрузка на базы данных наиболее низка.

Унаследованные базы данных продолжают выполнять свои обычные функции, а новые, такие как публикация данных на Web, или построение сводных отчетов возлагаются на хранилище данных. Хранилища данных открывают перспективы применения технологии «разработки» данных (**data mining**) – поиска аномалий, необычных образцов информации и использования их для оптимизации бизнес процессов.

Требования к СУБД.

Сегодня СУБД обязаны обеспечивать реализацию следующих требований.

1. Позволять пользователям создавать новые базы данных и определять их логические схемы (schemata) (логические структуры данных) с помощью некоторого специализированного языка, называемого ***языком определения данных*** (Data Definition Language - DDL).
2. Предлагать пользователям возможности задания ***запросов*** (queries) и модификации данных средствами соответствующего ***языка запросов*** (query language), или ***языка управления данными*** (Data Manipulation Language - DML).
3. Поддерживать возможность сохранения больших объемов информации – до многих гигабайтов и более на протяжении длительного времени, предотвращая возможность несанкционированного доступа к данным и гарантируя эффективность операций их просмотра и изменения.
4. Управлять единовременным доступом к данным со стороны многих пользователей, исключая возможность влияния действия одних пользователей на результаты, получаемые другим, и запрещая совместное обращение к данным, чреватое порчей.



Обзор структуры СУБД.

Рис. Компоненты системы управления базами данных.

Непрерывные линии – команды, пунктирные – потоки данных.

Прямоугольники – компоненты системы.

Прямоугольники с двойной рамкой – структуры данных образованные в памяти.

Два источника команд:

- **Рядовые пользователи и приложения**, запрашивающие или изменяющие данные.
- **Администратор базы данных** (database administrator, DBA) – лицо или группа лиц, ответственных за поддержку и развитие структуры, или **схемы** базы данных.

a) Команды определения данных
Язык DDL (Data Definition Language – язык определения метаданных) -> Компилятор DDL -> Исполняющая машина изменяет метаданные.

b) Обработка запросов
Язык управления данными (Data Manipulation language - DML), язык запросов (query language). SQL – самый распространенный.

Получение ответа на запрос. Компиляция -> план запроса (query plan) -> запросы на выборку данных *менеджеру ресурсов,* который осведомлен о размещении файлов данных, таблиц, индексов. Запросы на получение данных преобразуются в адреса страниц -> менеджер буферов. Единица обмена с диском – «дисковый блок» или страница. –> Менеджер хранения. Инструкции ОС или непосредственно контролера.

Обработка транзакций. Запросы группируются в транзакции – процессы, которые должны выполняться *атомарно* (atomically) и *изолированно* (in isolation) друг от друга. Часто – один запрос – отдельная транзакция. Устойчивость (durability). Процессор транзакций = планировщик заданий (scheduler) + менеджер протоколирования и восстановления (logging and recovery manager).

c) Менеджер буферов и хранения данных

Кластеры 4К или 16К –> страничные блоки в оперативной памяти содержащие:

- Данные
- Метаданные
- Статистику
- Индексы

d) Обработка транзакций

- команды транзакций определяются логикой работы приложения
- ведётся протоколирование
- существует управление параллельными заданиями, используются признаки блокировок ресурсов, которые запрещают обращаться к заблокированным ресурсам.
- осуществляется разрешение проблемы взаимоблокировок (deadlock resolution)

ACID: свойства транзакций:

A – atomicity (*атомарность*) - каждая транзакция выполняется целиком, либо не выполняется вовсе;

I – isolation (*изолированность*) - конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно;

D – durability (*долговечность*) - трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок);

C – consistency (*согласованность*) - гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных.

e) Процессор запросов (query processor)

Компилятор запросов

- Синтаксический анализ (query parser)
- Препроцессор (query preprocessor) – проверка на существование объектов
- Оптимизатор (query optimizer), используется статистика и метаданные.

Исполняющая машина

4. Модели организации баз данных

1. Иерархический подход к организации БД. Иерархические БД имеют форму деревьев с дугами-связями и узлами-элементами данных. Иерархическая структура предполагала неравноправие между данными – одни жестко подчинены другим. Подобные структуры, безусловно, четко удовлетворяют требованиям многих, но далеко не всех реальных задач.

2. Сетевая модель данных. В сетевых БД наряду с вертикальными реализованы и горизонтальные связи. Однако унаследованы многие недостатки иерархической и главный из них, необходимость четко определять на физическом уровне связи данных и столь же четко следовать этой структуре связей при запросах к базе.

3. Реляционная модель. Реляционная модель появилась вследствие стремления сделать базу данных как можно более гибкой. Данная модель предоставила простой и эффективный механизм поддержания связей данных.

а) Во-первых, все данные в реляционной модели представляются в виде таблиц и только таблиц. Реляционная модель – единственная из всех обеспечивает единообразие представления данных. И сущности, и связи этих самых сущностей представляются в модели совершенно одинаково – таблицами. Правда, такой подход усложняет понимание смысла хранящейся в базе данных информации, и, как следствие, манипулирование этой информацией.

б) Избежать трудностей манипулирования позволяет **второй элемент** модели – реляционно-полный язык (отметим, что язык является неотъемлемой частью любой модели данных, без

него модель не существует). Полнота языка в приложении к реляционной модели означает, что он должен выполнять любую операцию реляционной алгебры или реляционного исчисления (полнота последних доказана математически Э.Ф. Коддом). Более того, язык должен описывать любой запрос в виде операций с таблицами, а не с их строками. Одним из таких языков является SQL.

в) Третий элемент реляционной модели требует от реляционной модели поддержания некоторых ограничений целостности. Одно из таких ограничений утверждает, что каждая строка в таблице должна иметь некий уникальный идентификатор, называемый первичным ключом. Второе ограничение накладывается на целостность ссылок между таблицами. Оно утверждает, что атрибуты таблицы, ссылающиеся на первичные ключи других таблиц, должны иметь одно из значений этих первичных ключей.

4. Объектно-ориентированная модель. Новые области использования вычислительной техники, такие как научные исследования, автоматизированное проектирование и автоматизация учреждений, потребовали от баз данных способности хранить и **обрабатывать новые объекты** – текст, аудио, видео, документы. Основные трудности объектно-ориентированного моделирования данных проистекают из того, что такого развитого математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных, не существует. В большой степени поэтому до сих пор **нет базовой объектно-ориентированной модели**. Несмотря на преимущества и успехи объектно-ориентированных систем – реализация сложных типов данных, связь с языками программирования и т.п. – **на ближайшее время превосходство реляционных СУБД гарантировано**.

4.1. Иерархические структуры данных.

Описание.

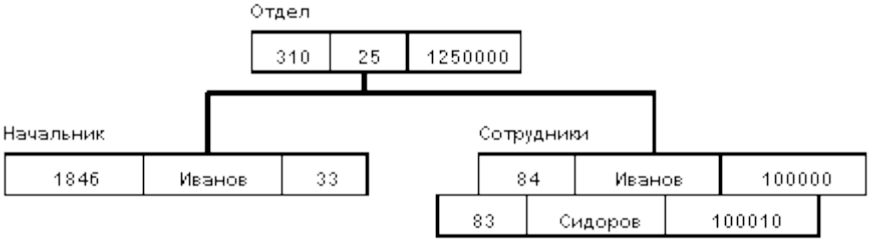
Иерархическая БД состоит из упорядоченного набора **деревьев**; более точно, из упорядоченного набора нескольких экземпляров одного типа дерева.

Тип дерева состоит из одного "корневого" типа записи и упорядоченного набора из нуля или более типов поддеревьев (каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор **типов записи**.

Пример типа дерева (схемы иерархической БД):

Здесь Отдел является **предком** для Начальник и Сотрудники, а Начальник и Сотрудники - **потомки** Отдел. Между типами записи поддерживаются **связи**.

База данных с такой схемой могла бы выглядеть следующим образом (показан один экземпляр дерева):



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

Для БД определен полный порядок обхода - сверху-вниз, слева-направо.

В IMS (IBM Information Management System) использовалась нестандартная терминология: "сегмент" вместо "запись", а под "записью БД" понималось все дерево сегментов.

Манипулирование данными.

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

- Найти указанное дерево БД (например, отдел 310);
- Перейти от одного дерева к другому;
- Перейти от одной записи к другой внутри дерева (например, от отдела - к первому сотруднику);
- Перейти от одной записи к другой в порядке обхода иерархии;
- Вставить новую запись в указанную позицию;
- Удалить текущую запись.

Ограничения целостности.

Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя. Заметим, что аналогичное поддержание целостности по ссылкам между записями, не входящими в одну иерархию, не поддерживается (примером такой "внешней" ссылки может быть содержимое поля Каф_Номер в экземпляре типа записи Куратор).

В иерархических системах поддерживалась некоторая форма представлений БД на основе ограничения иерархии. Пример представления приведенной выше БД будет иерархия на рис.

4.2. Сетевые структуры данных.

Описание.

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

- Каждый экземпляр типа P является предком только в одном экземпляре L;
- Каждый экземпляр C является потомком не более, чем в одном экземпляре L.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

- Тип записи потомка в одном типе связи L1 может быть типом записи предка в другом типе связи L2 (как в иерархии).
- Данный тип записи P может быть типом записи предка в любом числе типов связи.
- Данный тип записи P может быть типом записи потомка в любом числе типов связи.
- Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если L1 и L2 - два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.
- Типы записи X и Y могут быть предком и потомком в одной связи и потомком и предком - в другой.
- Предок и потомок могут быть одного типа записи.

Простой пример сетевой схемы БД →

Сетевые базы нашли отражение в стандарте CODASYL (Committee on Data Systems and Languages).

Язык запросов CODASYL для перехода от одного элемента данных к другому предусматривал просмотр всех вершин графа предоставляющего элементы и связи между ними. Совершенно очевидно, что при размере графа в несколько тысяч элементов, система становится абсолютно непригодной из за малого быстродействия.

Манипулирование данными.

Примерный набор операций может быть следующим:

- Найти конкретную запись в наборе однотипных записей (инженера Сидорова);
- Перейти от предка к первому потомку по некоторой связи (к первому сотруднику отдела 310);
- Перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
- Перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
- Создать новую запись;
- Уничтожить запись;
- Модифицировать запись;
- Включить в связь;
- Исключить из связи;
- Переставить в другую связь и т.д.

Ограничения целостности.

В принципе их поддержание не требуется, но иногда требуют целостности по ссылкам (как в иерархической модели).

4.3. Реляционная модель данных.

4.3.1. Описание.

Реляционная модель предложена сотрудником компании IBM Е.Ф. Коддом в 1970 г. **(E.F. Codd. A Relational model for large shared data banks)**.

В настоящее время эта модель является фактическим стандартом, на который ориентируются практически все современные коммерческие СУБД. Структуры таблиц могут быть очень сложными, но это не снижает скорости выполнения запросов. В отличие от ранних баз данных, пользователю не требуется знать особенности хранения информации на носителе. Запросы к такой базе данных выражаются средствами языка высокого уровня, позволяющего повысить эффективность программиста.

4.3.2. Структура данных – таблица-отношение.

В реляционной модели достигается гораздо более высокий уровень абстракции данных, чем в иерархической или сетевой. В упомянутой статье 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 должно зависеть от способа их физической организации.

Один из самых естественных способов представления данных для пользователя непрограммиста – это двумерная таблица.

Таблицы могут быть построены таким образом, что не будет утеряна информация об отношениях между элементами данных. Таблицы – это прямоугольные массивы, которые можно описать математически. Таблица должна обладать следующими свойствами:

1. каждый элемент таблицы представляет собой один элемент данных, повторяющиеся группы отсутствуют;
2. все столбцы в таблице однородные, то есть элементы столбца имеют одинаковую природу;
3. столбцам однозначно присвоены имена;
4. в таблице нет двух одинаковых строк;
5. в операциях с такой таблицей ее строки и столбцы могут просматриваться в любом порядке и в любой последовательности безотносительно к их информационному содержанию и смыслу.

Эти свойства обеспечиваются за счет использования математической теории отношений (само название "реляционная" происходит от английского relation - "отношение").

Таблица такого вида, называется **отношением**. БД, построенная с помощью отношений называются **реляционной БД**.

Кодд разработал специальный ЯМД для такой БД. В этом языке есть возможность извлекать подмножества столбцов таблицы для одних пользователей, создавая таблицы меньшей размерности, а также объединять таблицы для других пользователей, создавая таблицы большей размерности. Язык Кодда обладает гибкостью, которой лишено большинство древовидных и сетевых структур. Существует много способов отображения БД Кодда на физическом носителе.

Операции реляционной алгебры будут рассмотрены в одной из следующих тем, здесь приведены только два понятия.

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

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)

Отношение: Отношением R, определенным на множествах $D_1, D_2, ..., D_n$ называется подмножество декартова произведения $D_1 * D_2 * ... * D_n$. При этом:

- множества $D_1, D_2, ..., D_n$ называются доменами отношения
- элементы декартова произведения $d_1 * d_2 * ... * d_n$ называются кортежами
- число n определяет степень отношения (n=1 - унарное, n=2 - бинарное, n=3 – тернарное, ..., 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)

Реляционное отношение.

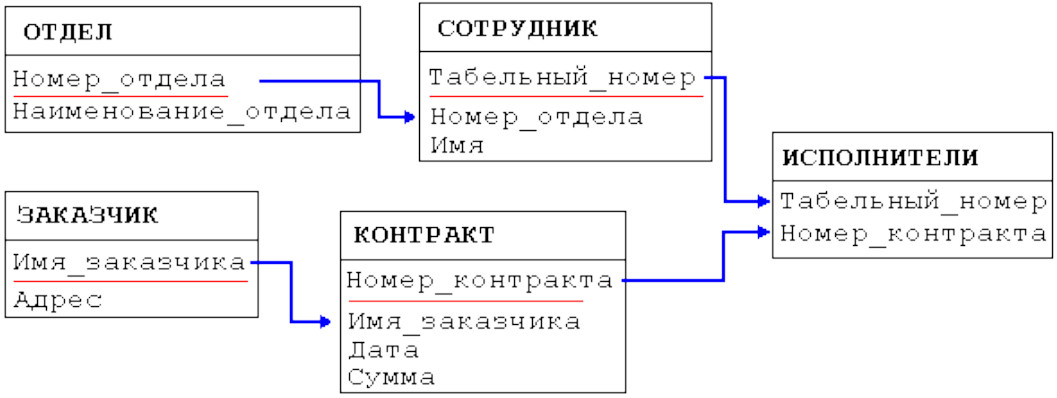
Рис. На рис. представлена таблица (отношение степени 5), содержащая некоторые сведения о работниках гипотетического предприятия.

4.3.3. Основные компоненты реляционного отношения.

a. Строки таблицы соответствуют **кортежам**. Каждая строка фактически представляет собой описание одного объекта реального мира (в данном случае работника), характеристики которого содержатся в столбцах.
b. 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**.
c. Каждый атрибут определен на **домене**, поэтому домен можно рассматривать как множество допустимых значений данного атрибута.
d. Несколько атрибутов одного отношения и даже атрибуты разных отношений могут быть определены на одном и том же домене. В примере, показанном на рис. 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.
e. Именованное множество пар "имя атрибута - имя домена" называется **схемой отношения**.
f. Мощность этого множества - называют **степенью или "арностью" отношения.**
g. Набор именованных схем отношений представляет из себя **схему базы данных**.
h. 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.

i. Если кортежи идентифицируются только сцеплением значений нескольких атрибутов, то говорят, что отношение имеет **составной ключ**.
j. Ключ должен обладать двумя **свойствами**:
   А) **Однозначная идентификация кортежа**: кортеж должен однозначно определяться значением ключа.
   Б) **Отсутствие избыточности**: никакой атрибут нельзя удалить из ключа, не нарушая при этом свойства однозначной идентификации.
k. Отношение может содержать несколько ключей. Всегда один из ключей объявляется **первичным ключом**, его значения не могут обновляться. Для основного ключа атрибуты следует выбирать так, чтобы для атрибутов был заранее известен диапазон значений, а количество атрибутов было бы как можно меньше.
l. Все остальные ключи отношения называются **возможными ключами**.
m. 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. Атрибуты, представляющие собой копии ключей других отношений, называются **внешними ключами**.

Например. Рассмотренная ранее база данных, содержащая сведения о подразделениях предприятия и работающих в них сотрудниках, применительно к реляционной модели будет иметь вид показанный на рис. ниже.



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, выполняют обратную операцию:

- определить "Номер_отдела" из таблицы СОТРУДНИК
- по полученному значению найти запись в таблице ОТДЕЛ.

4.3.4. Свойства отношений.

a. **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.
b. **Отсутствие упорядоченности кортежей.**
c. **Отсутствие упорядоченности атрибутов.** Для ссылки на значение атрибута всегда используется имя атрибута.
d. **Атомарность значений атрибутов**, т.е. среди значений домена не могут содержаться множества значений (отношения).

В настоящее время реляционные системы управления базами данных составляют основу рынка информационных технологий. Дальнейшие исследования ведутся в направлении сочетания той или иной степени реляционной модели.

5. Представление данных с помощью модели "сущность-связь".

5.1. 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. 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" была предложена в 1976 г. **Питером Пин-Шэн Ченом**, есть русский перевод его статьи 'Модель "сущность-связь" - шаг к единому представлению данных'.

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.

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.

5.2. Элементы модели.

1. Сущность (entity) - это объект, который может быть идентифицирован неким способом, отличающим его от других объектов. Примеры: конкретный человек, предприятие, проект, событие и т.д.

2. Набор сущностей (entity set) - множество сущностей одного типа (обладающих одинаковыми свойствами). Примеры: все люди, предприятия, проекты, праздники и т.д. Наборы сущностей не обязательно должны быть непересекающимися. Например, сущность, принадлежащая к набору МУЖЧИНЫ, также принадлежит набору ЛЮДИ; или сущность СТОЛИЦЫ принадлежит набору ГОРОДА.

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;
```

В дальнейшем для определения сущности и ее атрибутов будем использовать обозначение:

СОТРУДНИК (ТАБЕЛЬНЫЙ_НОМЕР, ИМЯ, ВОЗРАСТ).

Например отделы, на которые подразделяется предприятие, и в которых работают сотрудники, можно описать как ОТДЕЛ(НОМЕР_ОТДЕЛА, НАИМЕНОВАНИЕ).

3. Множество значений (область определения) атрибута называется **доменом**. Например, для атрибута ВОЗРАСТ его 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.

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).

4. 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;

Отметим, что в методике проектирования данных есть **правило**, согласно которому сущности обозначаются с помощью имен существительных, а связи - глагольными формами. Данное правило, однако, не является обязательным.

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 связь "является" с сущностью "СОТРУДНИК".

5. 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**.

6. Набор связей (relationship set) - это отношение между n (причем n обычно 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

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.

7. Степенью (кардинальностью) связи называют число сущностей, которое может быть ассоциировано с другой сущностью. Рассмотрение степеней особенно полезно для бинарных связей. Могут существовать следующие степени бинарных связей: 1:1, 1:n, n:1, n:m.

5.3. Степени (кардинальность) связи и зависимость отношений.

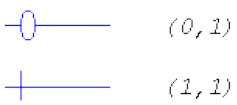
5.3.1. Один к одному (обозначается 1:1).

В такой связи сущности с одной ролью всегда соответствует не более одной сущности с другой ролью. В рассмотренном 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 рисунке, где прямоугольники обозначают сущности, а ромб - связь. **Степень (кардинальность) связи** для каждой сущности равна 1, поэтому они соединяются одной линией на конце.

Далее для графического отображения элементов ER модели использована **смешанная нотация** Чена и Мартина.

Второй важной характеристикой связи является **обязательность** принадлежности входящих в нее сущностей. Так как в каждом отделе обязательно должен быть руководитель, то каждой сущности "ОТДЕЛ" непременно должна соответствовать сущность "СОТРУДНИК". Но, не каждый сотрудник является руководителем отдела, следовательно 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 (интервал 0,1). Теперь данную связь мы можем описать как (1,1:0,1). В дальнейшем обязательность бинарных связей степени 1 будем обозначать как на рисунке справа (O и I) →

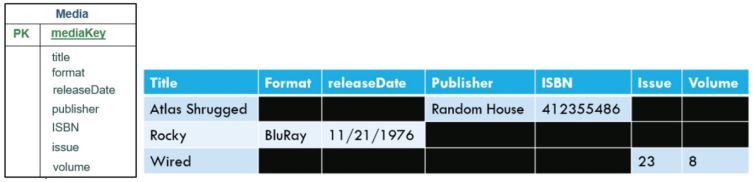


Расширенные обозначения вариантов связи 1:1 - это 0,1:0,1 или 1,1:0,1 или 0,1:1,1 или 1,1:1,1.

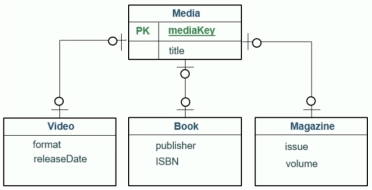
Некоторые особенности связи 1:1.

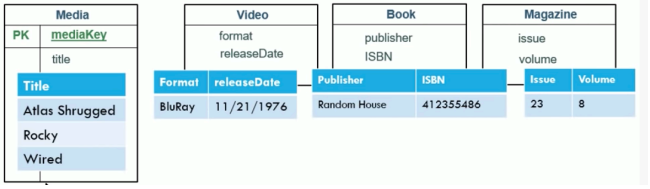
Это редко используемый тип.

Возможно, чтобы избавиться от необходимости хранить null.

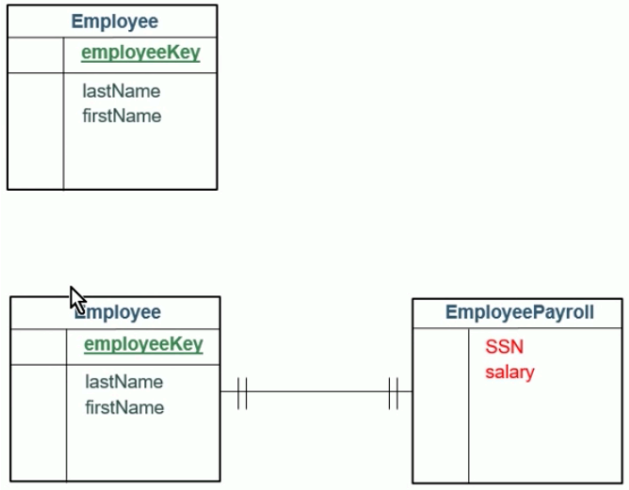


Title	Format	releaseDate	Publisher	ISBN	Issue	Volume
Atlas Shrugged			Random House	412355486		
Rocky	BluRay	11/21/1976				
Wired					23	8



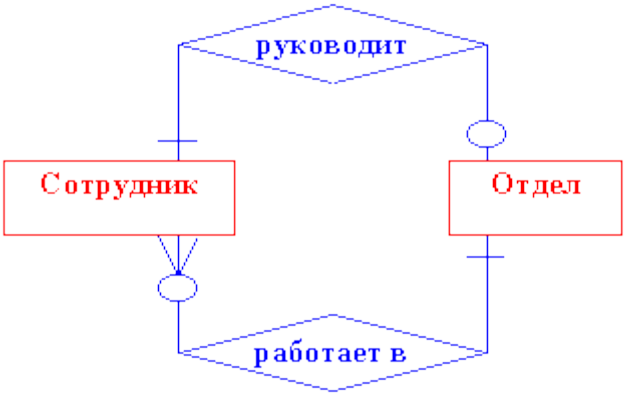


Возможно, для безопасности, чтобы связать разные части одной и той же записи (разделенные на отдельные таблицы).



5.3.2. Один ко многим (1:n).

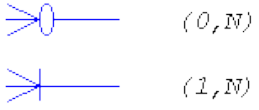
В данном случае сущности с одной ролью может соответствовать любое число сущностей с другой ролью. Такова **связь ОТДЕЛ-СОТРУДНИК**. В каждом отделе может работать произвольное число сотрудников, но сотрудник может работать только в одном отделе, связь 1:n. Графически степень связи n отображается "тройной" линией на конце, как на рис.



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 (1,1:0,n).

Рисунок дополнительно иллюстрирует тот факт, что между двумя сущностями может быть определено несколько наборов связей - вторая связь отделом «руководит» сотрудник. Здесь также необходимо учитывать обязательность принадлежности сущностей (0,1:1,1).

Кардинальность бинарных связей степени n будем обозначать как на рисунке слева (куриная лапка – Crow's Foot) →



5.3.3. Многие к одному (n:1).

Эта связь аналогична обратному 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 заказчиком), каждая сущность имеет обязательный класс принадлежности (1,n:1,1).

5.3.4. 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 (m), а каждая группа должна включать не менее одного сотрудника (n), то связь между сущностями СОТРУДНИК и РАБОЧАЯ_ГРУППА имеет степень n:m (1,n:0,m).

Связи n:m могут быть использованы в логическом дизайне, но они не могут быть поддержаны ни в одной СУБД.

5.3.5. Зависимая (слабая) сущность.

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 выполняет КОНТРАКТ**. Рабочая группа создается только после того, как

будет подписан контракт, и прекращает своё 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 (Arrow Head Notation):

Заметим, что **обязательность и степень связи для сильной сущности всегда будет (1,1),** поэтому его можно не рисовать. 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.

6. Проектирование СУБД.

Фактически проект базы данных - это фундамент будущего программного комплекса, который будет использоваться достаточно долго и многими пользователями.

Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели.

Проект реляционной базы данных - это набор взаимосвязанных отношений, для которых определены все атрибуты, заданы первичные ключи отношений и заданы еще некоторые дополнительные свойства отношений, которые относятся к принципам поддержки целостности.

1. Системный анализ предметной области (неформальный текст)

2. Концептуальное проектирование (ER-модель)

3. Логическое проектирование (реляционная модель, нормализация, огранич.)

4. Определение требований и выбор СУБД (Hardware, OS, mySQL)

5. Физическое проектирование в конкретной СУБД (SQL, View, Transact)

Этапы проектирования СУБД.

1. Системный анализ предметной области - проводится изучение и подробное словесное описание понятий и объектов предметной области и реальных связей между ними. Создаются неформальные текстовые описания пользовательских представлений о функциях (задачах) СУБД.

Системный анализ должен заканчиваться:

- подробным интегрированным описанием информации об объектах предметной области, которая требуется для решения конкретных задач и которая должна храниться в БД;
- формулировкой конкретных задач, которые будут решаться с использованием этой БД с кратким описанием алгоритмов их решения;
- описанием выходных документов, которые должны генерироваться в системе;
- описанием входных документов, которые служат основанием для заполнения базы данными.

2. Концептуальное проектирование БД - частично формализованное описание объектов предметной области в терминах некоторой семантической модели, например, в терминах ER-модели.

Результатом являются общая концептуальная модель БД не зависимая от вида модели представления данных. ER-модель представляется в одной из используемых **нотаций - Чена, IDEF1X, Баркера, Мартина** или др.

3. Логическое проектирование БД – процесс преобразования концептуальной модели предметной области, представленной, например, в виде ER-диаграмм, в логическую схему БД.

- По формализованным правилам преобразовать ER-диаграмму в реляционную модель.
- Выявить и исправить нереализуемые и необычные конструкции данных.
- Определить все первичные ключи (ПК) и альтернативные ключи.
- Провести **проверочную нормализацию отношений по методике Кодда**
- Определить типы данных для атрибутов отношений.
- Описать все ограничения целостности БД.

Результатом являются схемы БД концептуального и внешнего уровней архитектуры, не зависимые от особенностей используемой СУБД.

4. Определение требований к операционной обстановке и выбор СУБД

На этом этапе производится оценка требований к вычислительным ресурсам, необходимым для функционирования системы, выбор типа и конфигурации ЭВМ, типа и версии операционной системы (ОС), выбор СУБД. Выбор зависит от таких показателей, как:

- примерный объём и динамика роста объёма данных в БД;
- характер запросов к данным (извлечение и обновление отдельных записей, обработка групп записей, обработка отдельных отношений или соединение отношений);
- интенсивность запросов к данным и требования ко времени отклика по типам запросов;
- режим работы (интерактивный, пакетный, реального времени) → системные требования к объёму оперативной и дисковой памяти, а также к функциональным возможностям ОС.

5. Физическое проектирование БД, то есть выбор эффективного размещения БД на внешних носителях для обеспечения наиболее эффективной работы приложения.

- Логическая модель реализуется в виде физической схемы БД для конкретной СУБД (например mySQL).
- Для дальнейших настроек, разработок и проверок создаётся скрипт наполнения БД тестовым набором данных.
- Создаются основные объекты: запросы; представления; хранимые процедуры; триггеры; транзакции.
- Определяются параметры распределения памяти для объектов БД, строятся индексы, определяется целесообразность использования хеширования и кластеризации.
- Выявляются транзакции, которые будут выполняться в проектируемой базе данных. Анализируется *пропускная способность транзакций* – количество транзакций, которые могут быть обработаны за заданный интервал времени, и *время ответа* – промежуток времени, необходимый для выполнения одной транзакции.
- На основании указанных показателей принимаются решения об оптимизации производительности базы данных путем определения индексов в таблицах, ускоряющих выборку данных из базы, или снижения требований к уровню нормализации таблиц.
- Организуется мониторинг функционирования и разрабатывается стратегия защиты БД.

Результаты работы документируются в форме **схемы хранения на языках DDL и примеров запросов к базе на языке DML** конкретной СУБД. После этого этапа СУБД готова к эксплуатации. Для созданной БД пишутся приложения.