**Базы данных - Урок 1. Понятие базы данных**

Возможно, вы еще не знаете, что входит в понятие базы данных, но то, что вы ими постоянно пользуетесь абсолютно точно. Каждый раз, когда вы что-то ищете в поисковике, вы используете базу данных. Когда вы вводите свои логин и пароль для входа на какой-нибудь сервис, они сравниваются со значениями, которые хранятся в базе данных этого сервиса.

Несмотря на то, что мы постоянно используем базы данных, для многих остается непонятным, что же это такое на самом деле. И связано это отчасти с тем, что одни и те же термины, относящиеся к базам данных, используются людьми для определения совершенно разных вещей.

Давайте разберемся с терминами и понятиями баз данных:

*База данных* - набор сведений, хранящихся некоторым упорядоченным способом. Можно сравнить базу данных со шкафом, в котором хранятся документы. Иными словами, база данных - это хранилище данных. Сами по себе базы данных не представляли бы интереса, если бы не было систем управления базами данных (СУБД).

*Система управления базами данных* - это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, менять и удалять, обеспечивает безопасность данных и т.д. В общем СУБД - это система, позволяющая создавать базы данных и манипулировать сведениями из них. А осуществляет этот доступ к данным СУБД посредством специального языка - SQL.

*SQL* - язык структурированных запросов, основной задачей которого является предоставление простого способа считывания и записи информации в базу данных.

Итак, простейшая схема работы с базой данных выглядит примерно так:



По характеру использования СУБД делят на однопользовательские (предназначенные для создания и использования БД на персональном компьютере) и многопользовательские (предназначенные для работы с единой БД нескольких компьютеров, объединенных в локальные сети). Вообще деление по характеру использования можно представить следующей схемой:

 Не вдаваясь далее в подробности, отметим, что на сегодняшний день число используемых СУБД исчисляется десятками. Наиболее известные однопользовательские СУБД - Microsoft Visual FoxPro и Access, многопользовательские - MS SQL Server, Oracle и MySQL.

В этих уроках мы будем использовать СУБД MySQL. Во-первых, она бесплатная, а во-вторых, она является стандартом де-факто у российских хост-провайдеров. Но об этом позже, а пока вернемся к основам. В определении базы данных говорится, что это сведения, которые упорядочены некоторым образом. А как собственно они упорядочены? Об этом и пойдет речь в следующем уроке.

# Базы данных - Урок 2. Реляционные базы данных

Реляционные базы данных, как мы уже знаем, состоят из таблиц. Каждая таблица состоит из столбцов (их называют *полями или атрибутами*) и строк (их называют *записями или кортежами*). Таблицы в реляционных базах данных обладают рядом свойств. Основными являются следующие:

· В таблице не может быть двух одинаковых строк. В математике таблицы, обладающие таким свойством, называют *отношениями* - по-английски relation, отсюда и название - реляционные.

· Столбцы располагаются в определенном порядке, который создается при создании таблицы. В таблице может не быть ни одной строки, но обязательно должен быть хотя бы один столбец.

· У каждого столбца есть уникальное имя (в пределах таблицы), и все значения в одном столбце имеют один тип (число, текст, дата...).

· На пересечении каждого столбца и строки может находиться только атомарное значение (одно значение, не состоящее из группы значений). Таблицы, удовлетворяющие этому условию, называют *нормализованными*.

Все будет понятнее на примере. Предположим, мы захотели создать базу данных для форума. У форума есть зарегистрированные пользователи, которые создают темы и оставляют сообщения в этих темах. Эта информация и должна храниться в базе данных.

Теоретически (на бумаге) мы можем все это расположить в одной таблице, например, так:



Но это противоречит свойству атомарности (одно значение в одной ячейке), а в столбцах Темы и Сообщения у нас предполагается неограниченное количество значений. Значит, нашу таблицу надо разбить на три: Пользователи, Темы и Сообщения.



Наша таблица Пользователи удовлетворяет всем условиям. А вот таблицы Темы и Сообщения - нет. Ведь в таблице не может быть двух одинаковых строк, а где гарантия, что один пользователь не оставит два одинаковых сообщения, например:



Кроме того, мы знаем, что каждое сообщение обязательно относится к какой-либо теме. А как это можно узнать из наших таблиц? Никак. Для решения этих проблем, в реляционных базах данных существуют *ключи*.

*Первичный ключ* (сокращенно РК - primary key) - столбец, значения которого во всех строках различны. Первичные ключи могут быть логическими (естественными) и суррогатными (искусственными). Так, для нашей таблицы Пользователи первичным ключом может стать столбец e-mail (ведь теоретически не может быть двух пользователей с одинаковым e-mail). На практике лучше использовать суррогатные ключи, т.к. их применение позволяет абстрагировать ключи от реальных данных. Кроме того, первичные ключи менять нельзя, а что если у пользователя сменится e-mail?

Суррогатный ключ представляет собой дополнительное поле в базе данных. Как правило, это порядковый номер записи (хотя вы можете задавать их на свое усмотрение, контролируя, чтобы они были уникальны). Давайте внесем поля первичных ключей в наши таблицы:





Теперь каждая запись в наших таблицах уникальна. Нам осталось установить соответствие между темами и сообщениями в них. Делается это также при помощи первичных ключей. В таблицу сообщения мы добавим еще одно поле:



Теперь понятно, что сообщение с id=2 принадлежит теме "О рыбалке" (id темы = 4), созданной Васей, а остальные сообщения принадлежать теме "О рыбалке" (id темы = 1), созданной Кириллом. Такое поле называется *внешний ключ*(сокращенно FK - foreign key). Каждое значение этого поля соответствует какому-либо первичному ключу из таблицы "Темы". Так устанавливается однозначное соответствие между сообщениями и темами, к которым они относятся.

Последний нюанс. Предположим, у нас добавился новый пользователь, и зовут его тоже Вася:



Как мы узнаем, какой именно Вася оставил сообщения? Для этого поля автор в таблицах "Темы" и "Сообщения" мы сделаем также внешними ключами:




Наша база данных готова. Схематично ее можно представить так:



В нашей маленькой базе данных всего три таблички, а если бы их было 10 или 100? Понятно, что сразу невозможно представить все таблицы, поля и связи, которые нам могут понадобиться. Именно поэтому проектирование базы данных начинается с ее концептуальной модели, которую мы и рассмотрим в следующем уроке.

# Базы данных - Урок 3. Концептуальная модель базы данных

Концептуальная модель - это отражение предметной области, для которой разрабатывается база данных. Не вдаваясь в теорию, отметим, что это некая диаграмма с принятыми обозначениями элементов. Так, все объекты, обозначающие вещи, обозначаются в виде прямоугольника. Атрибуты, характеризующие объект - в виде овала, а связи между объектами - ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим - двойная стрелка, а со стороны, где она равна единице - одинарная).

Давайте в качестве примера рассмотрим интернет-магазин. У магазина есть товары, которые поставляются поставщиками и покупаются покупатели. Это можно представить тремя объектами и двумя связями:



Но как поставщик поставляет товары? Он делает поставку, которая подтверждается документом. Аналогично и покупатель делает покупку, которая также может подтверждаться документом. Таким образом, поставка и покупка могут рассматриваться, как самостоятельные объекты:



Теперь у нас пять объектов и четыре связи. Две связи "один ко многим" (один поставщик может осуществить несколько поставок, но каждая поставка осуществляется только одним поставщиком, аналогично и для связи Покупатель - Покупка) и две связи "многие ко многим" (каждая поставка может содержать несколько товаров, а один и тот же товар может содержаться в нескольких поставках, аналогично и для связи Покупка - Товар).

Но связи "многие ко многим" недопустимы в реляционной модели, поэтому каждую такую связь надо заменить на две связи "один ко многим". Делается это добавлением промежуточного объекта:



Таким образом, у нас появилось еще два объекта - журнал покупок и журнал поставок, со связями "один ко многим" (один журнал поставок может включать несколько поставок, но каждая поставка может входить только в один журнал, аналогично и для остальных).

Каждый объект нашего магазина имеет свои атрибуты:



Вот собственно мы и создали концептуальную модель базы данных магазин, вернее ее части, ведь в магазине еще есть сотрудники, склады, доставка товаров и т.д.

Вообще, если предметная область обширная, то ее полезно разбить на несколько локальных предметных областей (наша концептуальная модель отражает именно локальную предметную область). Объем локальной области выбирается таким образом, чтобы в нее входило не более 6-7 объектов. После создания моделей каждой выделенной предметной области производится объединение локальных концептуальных моделей в одну общую, как правило, довольно сложную схему.

Для наших учебных целей, мы ограничимся созданной моделью. Теперь нашу концептуальную модель надо преобразовать в реляционную модель данных, т.е. в уже известные нам таблицы, поля, ключи и т.д. Этим мы и займемся на следующем уроке.

# Базы данных - Урок 4. Преобразование модели в реляционную

Преобразование концептуальной модели в реляционную состоит в следующем:

· Построить набор предварительных таблиц и указать первичные ключи.

· Провести процесс нормализации.

Первый пункт мы рассматривали в третьем уроке, со вторым мы пока не знакомы, но ознакомимся на практике.

Итак, нам надо построить набор таблиц. Сделать это несложно, т.к. таблицы - это наши объекты, а поля таблиц - атрибуты объектов. Набор предварительных таблиц, исходя из нашей концептуальной модели, выглядит так:

 Таким образом, у нас определены таблицы, поля, первичные ключи (РК) и связи (FK). Обратите внимание, в таблицах Журнал поставок и Журнал покупок первичные ключи - составные, т.е. состоят из двух полей. Теоретически бывают таблицы, в которых все поля являются одним составным ключом.

Переходим ко второму пункту, а именно к нормализации отношений (таблиц). *Нормализация* - это пошаговый, обратимый процесс замены исходной схемы другой схемой, в которой таблицы имеют более простую и логичную структуру. Для чего это нужно?

Во-первых, для устранения избыточности данных. Например, в нашем примере для форума (из третьего урока), мы оставили бы вот такую таблицу:



В поле Темы часто повторяются одни и те же названия. Помимо того, что для их хранения потребуются дополнительные ресурсы памяти, при дублировании информации очень несложно допустить ошибку при вводе значений атрибута, в результате чего БД перейдет в несогласованное состояние.

Кроме того, при работе с такими таблицами могут возникнуть так называемые аномалии обновления. Например, если мы удалим из этой таблицы четвертое сообщение, то вместе с ним пропадет и информация о теме. Такая ситуация представляет собой аномалию удаления. Если мы решим поменять название темы, то нам придется просмотреть все строки и в каждой заменить старую тему на новую. Это так называемая аномалия модификации. Существуют и другие виды аномалий.

Далеко не всегда эти недостатки можно учесть сразу. Для их устранения и применяется процесс нормализации. Он включает ряд правил, используемых для проверки всех таблиц базы данных. Различают:

· 1НФ - первая нормальная форма

· 2НФ - вторая нормальная форма

· 3НФ - третья нормальная форма

· НФБК - нормальная форма Бойса-Кодда

· 4НФ - четвертая нормальная форма

· 5НФ - пятая нормальная форма

Каждая нормальная форма налагает определенные ограничения на данные. Каждая нормальная форма более высокого уровня предполагает, что анализируемая таблица уже находится в нормальной форме на уровень ниже рассматриваемой. В ходе нормализации схема базы данных становится все более строгой, а ее таблицы все менее подвержены различного рода аномалиям.

Для реляционных баз данных необходимо, чтобы ее таблицы находились в 1НФ. Нормальные формы более высоких уровней могут использоваться разработчиками по своему усмотрению. Однако грамотный специалист стремится к тому, чтобы довести уровень нормализации базы данных хотя бы до 3НФ, тем самым исключив избыточность данных и аномалии обновления. Надо сказать, что НФБК, 4НФ и 5НФ используются крайне редко. Поэтому и мы рассмотрим только первые три.

*Первая нормальная форма*

Таблица находится в первой нормальной форме, если все ее поля имеют простые (атомарные) значения. Само понятие атомарности определить достаточно трудно. Значение, атомарное в одном случае, может быть неатомарным в другом. Общий принцип здесь такой: значение не атомарно, если оно используется по частям. Понятнее будет на примере.

В нашей таблице Поставщики есть поле Адрес. Если наш магазин работает только с поставщиками из одного города, то значения поля Адрес можно считать атомарными, а саму таблицу - приведенной к 1НФ.

Но что если наши поставщики находятся в разных городах? Тогда, посылая машину за товарами в определенный город, мы должны быть уверенны, что она заберет товары у всех поставщиков, находящихся в этом городе. Т.е. нам могут понадобиться сведения о поставщикам, находящихся в определенном городе. В этом случае, значения в поле Адрес уже не являются атомарными (т.к. мы используем часть адреса), и для приведения таблицы к 1НФ нам надо выделить еще одно поле - Город:



Таким образом надо проанализировать все таблицы нашей базы данных. Так, в таблице Покупатель есть поле ФИО. Если мы собираемся, например, поздравлять наших покупателей с именинами (которые, как известно, завися от имени), то это поле пришлось бы разбить на три: Фамилию, Имя и Отчество. Наш магазин этого делать не собирается, поэтому поле ФИО можно считать атомарным, а таблицу - приведенной к 1НФ.

Для запросов нашего магазина все остальные таблицы приведены к 1НФ.

*Вторая нормальная форма*

Эта форма применяется к таблицам с составными ключами. Таблица, у которой первичный ключ включает только одно поле, всегда находится во 2НФ.

Таблица находится во второй нормальной форме, если она находится в первой нормальной форме, а каждое неключевое поле функционально полно зависит от составного ключа.

В нашей базе данных две таблицы имеют составной ключ - Журнал покупок и Журнал поставок. Значение поля Количество зависит, как от Поставки (Покупки), так и от Товара. Значит, наши таблицы находятся во 2НФ.

Но предположим, что на этапе концептуального моделирования нашей базы данных, мы не выделили объекты Поставка и Покупка. Тогда наши таблицы могли бы выглядеть так:



Посмотрим теперь на таблицу Журнал поставок: поле Количество зависит от Наименования товара и от Даты поставки, но не зависит от того, кто поставил товар (поле Поставщика). Т.е. таблица не находится во 2НФ. Если бы на этапе концептуального моделирования нашей базы данных, мы не выделили объекты Поставка и Покупка, нам бы пришлось это делать сейчас. Но мы их выделили, поэтому все наши таблицы находятся во 2НФ.

*Третья нормальная форма*

Таблица находится в третьей нормальной форме, если она находится во второй нормальной форме, и каждое неключевое поле нетранзитивно зависит от первичного ключа.

Транзитивная зависимость наблюдается в том случае, если одно из двух неключевых полей зависит от первичного ключа, а другое зависит от первого неключевого поля. На примере будет понятнее.

Посмотрим на нашу таблицу Товар. В ней есть поле Цена, но цены, как известно, имеют свойство меняться. Если мы будем их менять прямо здесь, то будет пропадать вся информация о предыдущих ценах. Чтобы не терять эту информацию, надо добавить поле Дата (когда изменилась цена). Тогда наша таблица будет выглядеть так:



Даже не прибегая к 3НФ видно, что такая таблица будет содержать избыточную информацию. Но посмотрим на ее поля: поля Наименование и Дата зависят от id товара, а поле Цена зависит также и от Даты. Т.е. таблица не находится в 3НФ. Для устранения транзитивной зависимости необходимо провести "расщепление" объекта на два:



Все остальные таблицы нашей базы данных находятся в 3НФ. Кстати, в таблице Товар можно было и не вводить поле id товара, а сделать первичным ключом поле Наименование, но как уже говорилось в третьем уроке суррогатные ключи все-таки предпочтительнее.

Подведем итог. Схема нашей базы данных после нормализации несколько изменилась и выглядит теперь так:



Таким образом, мы преобразовали нашу концептуальную модель в реляционную. Дальше необходимо эту модель реализовать в конкретной СУБД. Для этого нам понадобится сама СУБД и знание языка SQL. Этим вопросам будут посвящены Уроки SQL.

А пока подведем итог уроков "Основы БД". Проектирование БД процесс, как правило, трудоемкий и небыстрый. Ведь нужно очень хорошо изучить предметную область, чтобы учесть все нюансы, пожелания и требования пользователей. Затем всю собранную информацию изобразить в виде объектов, атрибутов и связей. Причем сделать это надо наиболее рационально.

Вообще, среди разработчиков наблюдаются различные взгляды на процесс проектирования БД. Одни игнорируют всякую теорию и руководствуются только своим опытом и здравым смыслом. Другие считают этот процесс искусством, отводя главную роль интуиции. Но в любом случае, знания не бывают лишними. И если вы к интуиции и здравому смыслу добавите теорию, то результат будет гораздо лучше.

Да, база данных - это всего лишь хранилище данных, но от того насколько грамотно вы организуете это хранилище, будет зависеть работа вашего приложения, использующего данные. Помните об этом и не пренебрегайте теорией.

В заключение хотелось бы напомнить, зачем вообще вам нужно уметь проектировать базы данных. Предположим, вы решили организовать у себя на сайте регистрацию пользователей с тем, чтобы обеспечить им доступ к закрытым материалам сайта.

Для реализации этого вопроса вам потребуется создать БД, которая будет хранить информацию о пользователях, их логинах и паролях. А также сделать html-формы регистрации и входа в закрытый раздел.

Когда пользователь регистрируется, эти данные программными средствами (например, с помощью языка PHP) заносятся в созданную вами БД. Когда пользователь вводит логин и пароль в форме входа в закрытый раздел, к базе данных отправляется запрос (на языке SQL), есть ли пользователь с такими данными. И если ответ положительный, то пользователю посылается запрашиваемая страница (разумеется тоже с помощью программы на PHP).

Таким образом, чтобы реализовывать такие приложения вам необходимо уметь создавать БД, строить запросы на языке SQL к БД и знать какой-нибудь язык программирования, применимый для разработки динамических web-страниц (например, PHP).

В принципе, вы можете сначала изучить язык программирования, а потом изучать БД и SQL. Но на этом сайте обучение построено в следующем порядке: БД - SQL - PHP. Сделано это потому, что без баз данных ничего интересного на PHP сделать не удастся. Поэтому до встречи в Уроках SQL.

Перед тем, как переходить к уроках SQL, вам потребуется установить сервер MySQL. В принципе вы можете установить только его, но для полноценной дальнейшей работы вам понадобится интерфейс phpMyAdmin, а для работы с ним - модуль PHP и локальный сервер. Поэтому настоятельно рекомендую установить сразу и сервер Apache, и модуль PHP, и MySQL. К тому же вам все-равно придется их устанавливать, когда вы начнете изучать PHP. Так что идите в раздел Предварительные установки и устанавливайте все по порядку, чтобы больше к этому не возвращаться.