Семен Игонин

Практический MySQL

пособие для быстрого старта

Пособие для тех, кто стремится начать знакомство с базами данных. Сперва кратко рассматривается теоретическая часть. Затем подробно разбираются вопросы связанные с работой с системой управления базами данных MySQL, от установки и настройки необходимых программ, до создания дампа готовой базы данных. Основная цель пособия: дать возможность читателю создать за один вечер свою первую базу данных. Не смотря на малый объем, материала книги достаточно для создания и управления базой данных простого сайта, интернет магазина или социальной сети.

Книга будет полезна и тем, кто планирует использовать графические оболочки для работы с базами данных. Знания, как формируются SQL запросы, позволят почувствовать уверенность в своих силах и более эффективно использовать готовые приложения для работы с системами управления базами данных.

Рецензенты и редакторы: *вакантная позиция, буду премного благодарен за любую помощь в указании на грамматические, стилистические и логические ошибки. Исправления выделять жёлтым цветом и присылать на is-info64@ya.ru, отзывы и пожелания туда же.*

Ссылка на .odt версию для редактирования: https://disk.yandex.ru/d/ehZb6LB6eewe1w

Третье издание: Саратов 2024

Оглавление

1. Введение в Базы Данных

1. 1 Определение

База данных (БД, **DB – Data Base**) – специальным образом организованная совокупность данных о некоторой предметной области, хранящаяся во внешней памяти компьютера.

Информация в БД хранится в упорядоченном виде. Например, записная книжка, классный журнал, библиотечный каталог. В простейшем случае, базы данных реализуются через обыкновенные текстовые файлы или таблицы. Потребность управлять большими объёмами данных приводит к более сложным способам организации хранения информации. Далее будем понимать под БД специальные файлы, в которой информация хранится структурировано определенным образом. Работать с такими БД напрямую, через текстовые или табличные редакторы затруднительно. Для работы с БД используют специальное программное обеспечение.

Система управления базами данных (СУБД, **DBMS – Data Base Managment System**) – программа, позволяющая выполнять операции с БД и хранимой в ней информацией.

1. 2 Свойства БД.

СУБД выполняет следующие функции:

- Создание, изменение и удаление данных.
- Поиск данных.
- Простые расчёты.
- Обеспечение целостности (корректности, непротиворечивости данных).
- Восстановление после сбоев.

Целостность означает, что БД содержит полную и непротиворечивую информацию и удовлетворяет всем заданным ограничениям. Выделяют физическую и логическую целостность.

1. Физическая — защита от разрушения при отказе оборудования реализуется через:

Использование транзакций — позволяет вернуть состояние БД, которое было до транзакции, в том случае, если транзакция не завершилась успешно, т. е. сбой произошел во время транзакции. **Транзакция** — группа операций, которая представляет собой одно законченное действие и должна быть выполнена целиком или не выполнена совсем.

При переводе **`$100`** со счета **`12345`** на счет **`54321`** через транзакцию требуется выполнить следующие действия:

1. Прочитать сумму на счёте **`12345`**
2. Уменьшить её на **`$100`**
3. Прочитать сумму на счёте **`54321 [<< произошел сбой питания]`**
4. Увеличить ее на **`$100.`**

Сбой произошел на **`3-м`** шаге, деньги были списаны, но не были зачислены. Но т. к. транзакция не завершилась, после включения питания, СУБД вернет БД в исходное состояние до начала транзакции (**`п.1`**). Транзакция реализуема посредством **журналирования**. Перед транзакцией исходные данные сохраняются в журнал, при сбое данные восстанавливаются из журнала. Если сбоя не было, то журнал после завершения транзакции очищается.

Использование RAID (**`Redundant Array of Independent Disks`**) — избыточный массив независимых дисков. Информация дублируется сразу на несколько жестких дисках. При выходе из строя одного из дисков, в работу вступает исправный диск, который содержит ту же самую информацию, что и сломанный диск.

2. Логическая целостность — означает, что данные не противоречат логике. Например, год рождения сотрудника не может быть **`1698`**, заработная плата не может быть отрицательной. Для сохранения логической целостности в БД:

- Поле имеет определенный тип.
- Некоторые поля обязательны для заполнения (например, поле, хранящее пароль).
- Вводят условия для значений полей (зарплата больше нуля, дата увольнения позже даты приема).

Следует отметить, что целостность не гарантирует достоверность. Достоверность — правдивость, целостность — непротиворечивость.

Выделяют различные типы БД в зависимости от модели данных: иерархические, объектные или объектно-ориентированные, объектно-реляционные, реляционные, сетевые, функциональные. Рассмотрим самую распространенную разновидность БД, с которой часто имеют дело на практике.

1.3 Реляционные БД

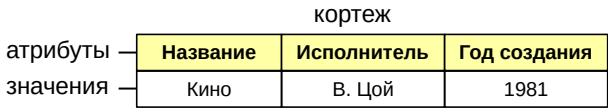
Реляционная модель является математическим описанием БД независимо от способа хранения данных. Разработана Эдгаром Коддом (IBM) в 1970 г. Согласно реляционной модели:

1. **Данные** — свойства некоторых объектов. Объекты делятся на классы — сущности.

2. Данные о некотором объекте представлены кортежем.

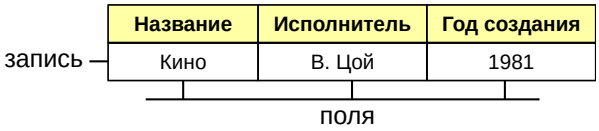
   Кортеж — набор пар «**атрибут** (свойство) — **значение**». Если объектом является музыкальная группа, то данные представляют собой свойства группы (название, исполнитель, год создания), а кортеж несет информацию об одной группе:

```
{
      Название: Кино;
      Лидер: В. Цой;
      Год создания: 1981;
}
```



3. **Отношение** — множество кортежей, описывающих объекты одного класса. Порядок кортежей в отношении отсутствует (не важен). Отношение можно представить таблицей, в которой столбцы являются атрибутами, а строки кортежами. Порядок записей отсутствует.

Определимся с терминологией. Будем рассматривать реляционную БД как набор **таблиц**, состоящих из столбцов и строк. **Запись** – одна строка таблицы с данными. **Поле** – значение, хранимое в одном из столбцов записи (ячейка таблицы).



В таблицах хранится информация об объектах. Для каждого столбца реляционной БД задан тип. В столбце всегда хранятся значения одного типа.

Количество и состав полей определяет разработчик при проектировании БД. Пользователь может только изменять записи. Каждое поле имеет уникальное **имя** и **тип**. Основные типы записей:

- Целое число
- Вещественное число
- Денежная сумма
- Логическое значение
- Текстовые данный
- Время и дата
- Произвольные двоичные данные (звук, виде, фото). Следует отметить, что двоичные данные обычно хранятся отдельными файлами на сервере, а в БД хранится ссылка на них в виде строки.

1.4 Ключи

Ключ — поле или комбинация полей, однозначно определяющая запись. Ключи бывают простыми — состоят из одного поля, и составными — из нескольких полей.

Каждая строка таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с использованием внешних ключей. К этим данным можно получить доступ многими способами, и при этом реорганизовывать таблицы БД не требуется.

1.4.1 Первичный ключ (Primary Key)

Первичный ключ – столбец, в полях которого значения уникальны (не повторяются) для каждой записи в таблице. В таблице не может быть двух записей с одинаковым первичным ключом. Примерами простого первичного ключа могут служить **`ИНН`**, **`СНИЛС`**, **`номер читательского билета`**. **Составной ключ** – состоит из двух и более столбцов (**`Фамилия-Имя`** ученика в БД школы).

Составной ключ

Фамилия	Имя	Класс
Иванов	Иван	9А
Петров	Петр	7В
Иванов	Иван	8Б

Первичный
Суррогатный ключ

ID	Фамилия	Имя	Класс
1	Иванов	Иван	9А
2	Петров	Петр	7В
3	Иванов	Иван	8Б

Приведенные выше примеры, являются плохими примерами первичного ключа. **`Номер читательского`** может быть не уникальным, а вероятность повторения **`Фамилия-Имя`** у разных людей достаточно велика. Правильнее отводить в таблице для первичного ключа отдельное поле — **суррогатный ключ** (искусственный). Суррогатным ключом является уникальный номер, например, числовой **`ID`** (id пользователей в социальных сетях) или **`GUID`** (**`globally unique identifier`**).

1.4.2 Внешний ключ (Foreign Key)

Внешний (вторичный) ключ – устанавливается для столбцов в подчиненной таблице и указывает на один из столбцов в главной таблице (обычно на первичный). Используется для связи записией из двух различных таблиц.

Первичный ключ

id	название	исполнитель	год создания
1	Кино	В. Цой	1981
2	Алиса	К. Кинчев	1983
3	Кукрыниксы	А. Горшенёв	1997

Группы

Первичный ключ Внешний ключ

id	название	группа
1	Не беда	3
2	Хочу перемен	1
3	Стук	1

Песни

Выше приведена схема БД из двух таблиц: **`Группы`** и **`Песни`**. Поле **`id`** в обеих таблицах назначено первичным ключом и однозначно определяет запись в таблице. Поле **`группа`** в таблице **`Песни`** назначено внешним ключом и ссылается на первичный ключ **`id`** в таблице **`Группы`**. По внешнему ключу можно найти для каждой песни информацию о группе. Удобно, что, при изменении информации о группах, таблица **`Песни`** остается не тронутой. Суррогатный ключ **`id`**, назначается для каждой записи при ее добавлении в таблицу и не изменяется, вплоть до удаления записи.

Внешний ключ необходим, чтобы поддерживать логическую целостность данных. Например, если попытаться добавить запись в таблицу песни со значением группа = 4, то СУБД выдаст ошибку, т. к. в таблице Группы отсутствует запись с id = 4.

При удалении Кино из таблицы Группы СУБД также выдаст ошибку, т. к. после этого останутся песни без групп. Другими словами, в таблице Песни значение из поля группа = 1 будет ссылаться на несуществующую запись с id = 1 из таблицы Группы. Имеется возможность настроить каскадное удаление, при котором при удалении группы по внешним ключам будут удаляться все связанные песни (этой группы).

Реляционная база данных – это БД, основанная на реляционной модели, т. е. представляет собой набор связанных отношений. Связь отношений осуществляется посредством внешних ключей.

1.5 Нормализация

Структуру БД следует делать такой, чтобы иметь удобный и быстрый доступ к информации. Под структурой понимается: какие столбцы в каких таблицах существуют и как организована связь между ними.

Нормализация – процесс улучшения структуры БД, удаления из БД избыточности, приведения к нормальной форме. **Избыточность** — когда храним данных больше, чем необходимо, дублирование. Нормализация позволяет избежать нарушения целостности данных и различных аномалий. Нормализация отражает **«здравый смысл»** при проектировании структуры БД. Всего нормальных форм насчитывается по различным источникам от **8** до **10**. Познакомимся с первыми тремя, что, на данный момент, достаточно для практических целей.

1NF

Переменная отношения находится в первой нормальной форме <=> в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

Простыми словами:

- В каждой ячейке таблицы хранится только одно значение.
- Не должно быть повторяющихся строк.

Первая нормальная форма характеризуется **атомарностью** значений (неделимость значения в каждом поле). Столбец **модели** в левой таблице хранит несколько значений, что не удовлетворяет **1NF**. Таблица справа удовлетворяет **1NF**.

фирма	модели
BMW	M5, X5M, M1
Nissan	GTR

фирма	модель
BMW	M5
BMW	X5M
BMW	M1
Nissan	GTR

2NF

Переменная отношения находится в первой нормальной форме <=> удовлетворяет 1NF и каждый неключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа.

Простыми словами:

- Таблица находится в **1NF**.
- Имеется первичный ключ.
- Все неключевые поля должны быть зависимы от первичного ключа.

В таблице необходимо хранить данные, непосредственно связанные с ней и не имеющие отношения к другой сущности. Данная форма решает вопрос нахождения дублирующихся данных. Различные сущности необходимо выносить в отдельные таблицы.
В других источниках, вторую форму относят к БД, в которых имеются составные ключи. Неключевые поля не могут зависеть от части составного ключа. Они должны зависеть от составного ключа как единого целого (сразу от всех столбцов ключа одновременно):

модель	фирма	цена	скидка
M5	BMW	5.5	5%
X5M	BMW	6.0	5%
M1	BMW	2.5	5%
GTR	Nissan	5.0	10%

В представленной таблице первичным ключом является сочетание полей **модель-фирма**. Цена зависит от первичного ключа, это удовлетворяет **2NF**. Скидка же зависит только от фирмы (от части ключа), поле **скидка** в таблице не удовлетворяет **2NF**. Следует разбить большую таблицу:

модель	фирма	цена
M5	BMW	5.5
X5M	BMW	6.0
M1	BMW	2.5
GTR	Nissan	5.0

фирма	скидка
BMW	5%
Nissan	10%

3NF

Переменная отношения находится в третьей нормальной форме <=> удовлетворяет 2NF и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

Простыми словами:

- Таблица находится в **2NF**.
- Все неключевые поля зависят только от первичного ключа, но не от других неключевых поля. Иными словами, отсутствуют неключевые поля, зависящие от других неключевых полей.

Функциональная зависимость — зависимость одного атрибута от другого, но не наоборот.

Зависимость **f(x)** является функциональной, если для любого **x** существует одно и только одно значение **f(x)**. Формально данное определение можно записать следующим образом:

f(x) функциональная зависимость <=> ∀ x ∃ одно значение f(x)

Пусть **x** — человек, а **f** — рост человека:

x	f
Женя	156
Никита	147
Сергей	156
Никита	151
Толя	163

Каждому человеку соответствует только одно значение роста, значит **f(x)** — функциональная зависимость. Построить график **f(x)** не составляет труда.

Но росту нельзя сопоставить строго одного человека, например, для роста **156** подходит два человека, значит **x(f)** не является функциональной зависимостью. График **x(f)** построить нельзя. Если на оси абсцисс отметить точку **156**, то не ясно какое значение **f** следует выбрать по оси ординат (**Женя** или **Сергей**). Однако, если бы в таблице не было повторяющихся значений роста, то формально можно составить зависимость человека от роста, хотя это и не логично. Если бы в столбце **x** было два одинаковых имени, то и зависимость **f(x)** не являлась функциональной.

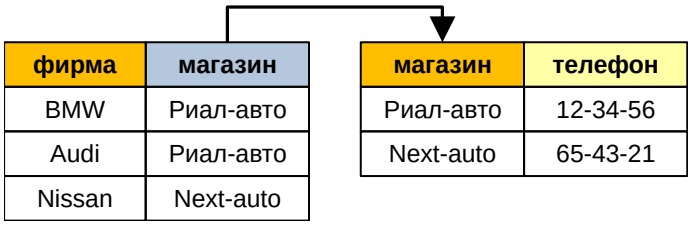
В представленном ниже отношении первичным ключом является поле **фирма**. В отношении существуют две **функциональные зависимости**:

- **магазин** зависит от **фирмы**, в нашем случае, автомобили каждой фирмы продаются только в одном магазине.
- **телефон** зависит от **магазина**, но не от фирмы.

фирма	магазин	телефон
BMW	Риал-авто	12-34-56
Audi	Риал-авто	12-34-56
Nissan	Next-auto	65-43-21

Транзитивная функциональная зависимость — функциональная зависимость, при условии, что ни один из атрибутов не является ключом (например, **индекс-город**).

В представленном отношении транзитивной зависимостью является **телефон-магазин**, т. к. оба поля (атрибута) не ключевые. Чтобы БД удовлетворяла **3NF**, как и в предыдущих случаях, следует разбить большую таблицу на несколько отношений:

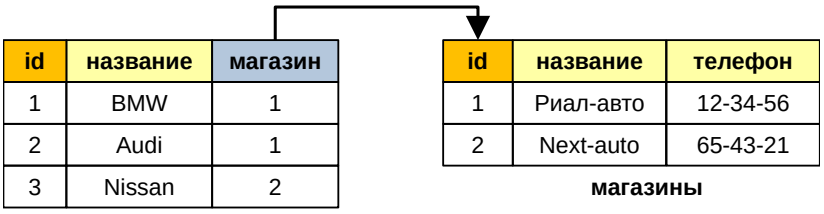


фирма	магазин
BMW	Риал-авто
Audi	Риал-авто
Nissan	Next-auto

магазин	телефон
Риал-авто	12-34-56
Next-auto	65-43-21

Информация о магазинах была вынесена в отдельную таблицу. В таблице с фирмами поле **`магазин`** является внешним ключом. Он связывает каждую запись из таблицы с фирмами с соответствующей записью из таблицы с магазинами.

Использование суррогатного ключа не применялось, чтобы не усложнять примеры. Более удачная структура БД выглядит следующим образом:



id	название	магазин
1	BMW	1
2	Audi	1
3	Nissan	2

фирмы

id	название	телефон
1	Риал-авто	12-34-56
2	Next-auto	65-43-21

магазины

Общее правило при проектировании БД

Выделять в объеме имеющихся данных отдельные сущности и выносить их в разные таблицы. Каждая строка таблицы имеет свой суррогатный первичный ключ id. Связь между таблицами осуществляется через эти id с использованием внешних ключей.

2. Настройка окружения

Прежде чем приступать к работе с БД, необходимо установить специализированную программу (СУБД) и произвести некоторые настройки в операционной системе. В книге рассматривается **DBMS MySql**.

2.1 Скачивание и установка MySQL

MySQL – это СУБД (**SQL** от англ. **Structured Query Language** – язык структурированных запросов). Скачиваем и устанавливаем **MySQL**:

https://dev.mysql.com/downloads/installer/

В данном установочном пакете содержится много утилит. При установке переключаемся на **Custom Install** и выбираем из списка только **MySQL Server**.

Мастер установки по окончании работы попросит произвести конфигурацию **MySQL Server**. Не отказывайтесь от нее. Нажимайте **Next**, пока конфигурация не завершится. На одном из шагов мастер попросит придумать и указать пароль администратора (**root password**), обязательно запомните его. Пароль необходим, чтобы управлять БД, находящейся на вашем компьютере. Для работы с БД из всего установившегося разнообразия нам понадобиться утилита **MySQL Command Line Client**.

База данных, расположенная на компьютере пользователя называется **локальной**. База данных, расположенная на другом компьютере сети (например, **Интернет**) — **удаленной** (от слова далеко). При желании, можно создать удаленную БД. Существует множество хостингов, позволяющих регистрировать БД, после чего имеется возможность подключаться к БД с любого компьютера, на котором установлен **MySql Client** и имеется выход в интернет.

2.2 Регистрация БД на хостинге

Регистрацию БД можно осуществить по ссылке:

https://www.db4free.net/signup.php

Обязательно выпишите на листочек данные для входа:

```
db name: ****
login: ****
password: ****
server: db4free.net
port:    3306
```

2.3 Добавление в PATH

После конфигурации БД на локальном компьютере (или регистрации БД на хостинге) требуется настроить операционную систему — добавить значение в переменную окружения **PATH**. На компьютере следует найти путь, по которому располагается файл **mysql.exe**. Выглядеть путь может следующим образом:

```
C:\ProgramFiles\MySQL\MySQLServer 8.0\bin
```

Открываем командную строку, для этого необходимо:

- нажать сочетание клавиш **Window + R**
- ввести **cmd**
- нажать **Enter**

Выполните команду представленную ниже, не забудьте указать актуальный для вашей машины путь до директории, в которой хранится файл **mysql.exe**:

```
setx /MPATH "%PATH%; C:\ProgramFiles\MySQL\MySQLServer 8.0\bin\"
```

Команда **setx** позволяет добавить директорию к переменной среды **PATH**. В **PATH** содержатся пути к директориям, в которых **Windows** автоматически пытается найти исполняемые файлы и библиотечные модули. Добавив путь к файлу **mysql.exe** в переменную среды **PATH**, мы избавимся от необходимости для запуска **mysql.exe** каждый раз переходить в каталог **MySQL SERVER 8.0\bin** с использованием команды **cd**.

3.Работа с MySQL

Рассмотрим подключение и взаимодействие с БД через СУБД **MySql**. Работу с СУБД будем осуществлять через командную строку операционной системы.

3.1 Подключение к БД через MySQL Command Line Client

Прежде чем вводить команды для работы с БД необходимо подключиться к БД. Открываем **mysql** и передаем параметры для подключения к удаленной БД, для этого выполните:

```
mysql -u<user_name> -p -h<host_name>
```

-u задает имя пользователя (**login**), если подключаетесь к БД на локальном компьютере укажите в качестве пользователя **-uroot**.

-p означает, что для подключения необходимо ввести пароль,

-h задает имя сервера (**host**) (**server**). Для подключения к БД на локальном компьютере используйте **-hlocalhost**, для подключения к удаленной БД укажите **-hdb4free.net**.

Вводим пароль. Если подключение прошло успешно, откроется приложение **mysql** в режиме командной строки.

Примечание: Треугольные скобки **<...>** в записи команд обозначают место, в которое требуется подставить собственное значение. Сами треугольные скобки писать не нужно. Расширение **.exe** в названии программы **mysql.exe** можно не указывать.

3.2 Команды для «навигации по БД»

После подключения к БД можно вводить **SQL** команды. Система на сервере или локальном компьютере может содержать множество БД. Чтобы отобразить список имен баз данных, после успешного подключения к **mysql**, выполните:

```
SHOW DATABASES;
```

Прежде чем совершать запросы к БД, ее необходимо выбрать командой:

```
USE <database_name>;
```

Создание новой БД осуществляется командой:

```
CREATE DATABASE <database_name>;
```

Не забудьте выбрать БД после создания **USE <database_name>;**

Следующие команды имеют смысл, если в выбранной БД уже имеются таблицы. Команда для отображения всех таблиц выбранной БД выглядит следующим образом:

```
SHOW TABLES;
```

Структуру таблицы с именем **<tb_name>** можно увидеть, выполнив запрос:

```
DESCRIBE <tb_name>;
```

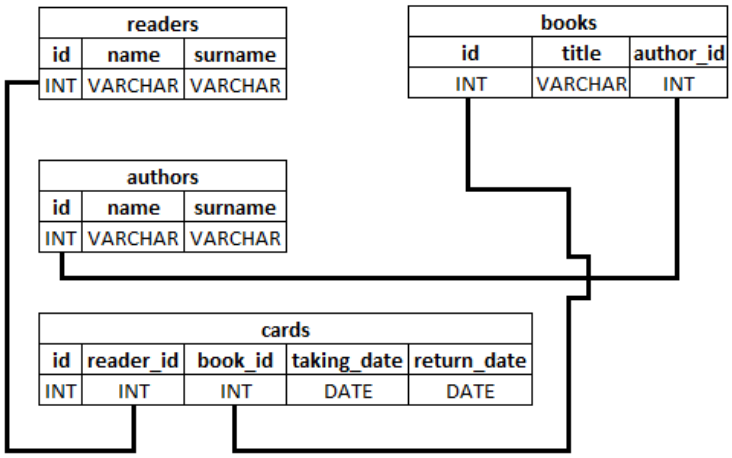
Полностью отобразить содержимое таблицы **<tb_name>** позволяет запрос:

```
SELECT * FROM <tb_name>;
```

Для выхода из СУБД **mysql** необходимо выполнить команду **exit**.

3.3 Основные команды MySQL

Изучение запросов будем производить на примере создания и заполнения БД **«Library»**, которую можно представить следующей схемой:



первая строка – название таблицы, вторая строка – названия столбцов, третья строка – типы данных в столбцах.

3.3.1 Создание таблицы

Для создания таблицы используется команда **CREATE TABLE**. Переносы строк и отступы значения не имеют, главное в конце запроса поставить точку с запятой.

```
CREATE TABLE readers(
          id INT AUTO_INCREMENT,
          name VARCHAR(255),
          surname VARCHAR(255),
    PRIMARY KEY(id)
);
```

- После ключевых слов **CREATE TABLE** указывается **имя**, которое вы придумали для таблицы. Имя следует выбирать осмысленным. Имя должно начинаться с буквы и не иметь пробелов, регистр не важен. В примере создается таблица с именем **readers**.

- В круглых скобках после имени перечисляются через запятую поля (столбцы), которые будут присутствовать в таблице. Для каждого поля следует указать два слова — название и тип значения. Например, **name VARCHAR(255)** означает поле с именем **name**, которое будет хранить строку длиной не более **255** символов. Наиболее распространенные типы:

    - **VARCHAR(X)** — строка длиной **X** символов.
    - **INT** — целое число
    - **DATETIME** — дата и время в формате **YYYY-MM-DD HH:MM:SS**

- **DATE, TIME, YEAR** — дата, время, год
- **TEXT** — большой текст длиной до 65 КБ
- **TINYINT(1)** — логический тип **(0 / 1)**, в **mysql** можно использовать псевдоним **BOOL**.
- **DOUBLE** — вещественное число.

- Атрибут **AUTO_INCREMENT**, означает, что, при добавлении к таблице новой строки, значение в поле столбца (**id**) заносится автоматически. Каждый раз при добавлении строки, это значение будет увеличиваться на единицу (**инкремент**). Значения в столбце **id** будут уникальными (будут отсутствовать одинаковые **id**), что позволит однозначно идентифицировать строку.

- Оператор **PRIMARY KEY(id)** указывает, что значения в столбце **id** будут являться первичными ключами.

Нельзя создать таблицы с одинаковыми именами. Написание запросов для создания оставшихся трёх таблиц на самостоятельное изучение.

3.3.2 Внешние ключи и ссылочная целостность

Связь между таблицами осуществляется с использованием внешних ключей. Задать внешний ключ можно с помощью команды изменения **ALTER**, но проще это сделать при создании таблицы. Для этого после **PRIMARY KEY** указывают выражение:

```
FOREIGN KEY (столбец_внешнего_ключа)
    REFERENCES внешняя_таблица (столбец_первичного_ключа)
```

Таблица **books** на схеме связана с таблицей **authors** через ключевые поля **books.author_id – authors.id**. Поле **author_id** является внешним (**FOREIGN**) ключом и указывает (**REFERENCES**) на первичный (**PRIMARY**) ключ **id** таблицы **authors**. Команда создания таблицы **books** с внешним ключом выглядит следующим образом:

```
CREATE TABLE books(..., PRIMARY KEY(id),
    FOREIGN KEY(author_id) REFERENCES authors(id)
    ON DELETE CASCADE
);
```

Выражение **ON DELETE CASCADE** вынуждает СУБД производить каскадное удаление. Например, если произвести удаление автора с **id = 7** из таблицы **authors**, то СУБД автоматически удалит из таблицы **books** записи, для которых поле **author_id = 7**. Таким образом, обеспечивается ссылочная целостной. Можно использовать выражение **ON DELETE SET NULL**, тогда при удалении автора в соответствующие поля **author_id** таблицы **books** будет занесено значение **NULL** (отсутствие автора). Если действие при удалении не указать, то при попытке удаления автора СУБД будет выводить ошибку до тех пор, пока не будут удалены все книги данного автора.

Существует ещё одно выражение: **ON UPDATE CASCADE**, которое автоматически изменяет значение

books.author_id, при изменении **authors.id**. Однако, на практике первичные ключи не изменяются и **ON UPDATE CASCADE** используют редко.

3.3.3 Изменение структуры таблицы после создания

Команда **ALTER** позволяет произвести изменения в структуре таблицы уже после ее создания. Рассмотрим некоторые варианты ее применения.

Добавление нового столбца:

```
ALTER TABLE <table_name> ADD <column_name> <TYPE>;
```

Удаление столбца:

```
ALTER TABLE <table_name> DROP COLUMN <column_name>;
```

Изменение типа столбца:

```
ALTER TABLE <table_name> MODIFY COLUMN <column_name> <NEW_TYPE>;
```

Добавление внешнего ключа к таблице **<table_name>**:

```
ALTER TABLE <table_name>
    ADD FOREIGN KEY(<column>) REFERENCES <table>(<column>);
```

3.3.4 Добавление и удаление записей

Запись – простыми словами, это одна строка таблицы. Добавление записи осуществляется командой **INSERT INTO ... VALUES ...:**

```
INSERT INTO readers(name, surname) VALUES('Ivan', 'Pupkin');
```

Обратите внимание на порядок следования столбцов и добавляемых значений. Между ними имеется прямое соответствие. Столбец **name** стоит на первой позиции, значит, в столбец **name** будет записано первое значение из скобок **VALUES** (т.е. **'Ivan'**). Значения записываются в одинарных кавычках. В команде **INSERT INTO readers(name, surname)** отсутствует столбец **id**, поскольку значения в его поля заносятся автоматически (**AUTO_INCREMENT**).

Пример команды удаления записей:

```
DELETE FROM readers WHERE id = '7';
```

Удалит из таблицы **readers** все записи с **id=7**. Если **id** — первичный ключ, то будет удалена одна запись.

3.4 Изменение значений в имеющихся записях

Синтаксис запроса, изменяющего значения в столбцах **<column_1>** и **<column_2>** таблицы **<table_name>**, удовлетворяющих условий **<condition>**, выглядит следующим образом:

```
UPDATE <table_name> SET
          <column_1> = 'value_1', <column_2> = 'value_2'
WHERE <condition>;
```

Рассмотрим на примере. Задача изменить фамилию читателя с **id = 2** (**'Pupkin'**) на **'Surokin'**:

```
UPDATE readers SET surname = 'Surokin' WHERE id = 2;
```

Обычно в условии **WHERE** указывается конкретный **id**. Если бы мы выбирали записи для изменения по фамилии **WHERE surname='Pupkin'**, то изменились бы все записи, для которых фамилия **Pupkin**. Скорее требуется изменить фамилию определенного человека, а не всех его однофамильцев. С запросами **UPDATE** и **DELETE** нужно быть предельно аккуратным, чтобы не испортить данные.

3.5 Выборка строк из таблицы

Самым интересным и важным действием при работе с БД является извлечение **необходимых** данных из БД, которое называется **выборкой**. Рассмотрим, как происходит выполнение запросов, на примере БД библиотека, имеющей следующие таблицы с данными:

```
mysql> SELECT * FROM readers;
+----+-------+---------+
| id | name  | surname |
+----+-------+---------+
|  1 | Petr  | Petrov  |
|  2 | Ivan  | Pupkin  |
|  3 | Yurii | Sidorov |
+----+-------+---------+
```

```
mysql> SELECT * FROM authors;
+----+----------+----------+
| id | name     | surname  |
+----+----------+----------+
|  1 | Alexandr | One      |
|  2 | Leo      | Greed    |
|  3 | Ya       | Yakahoma |
+----+----------+----------+
```

```
mysql> SELECT * FROM books;
+----+-----------+-------------+
| id | author_id | title       |
+----+-----------+-------------+
|  1 |         2 | Informatics |
|  2 |         3 | Geology     |
|  3 |         1 | Biology     |
|  4 |         1 | Philosophy  |
+----+-----------+-------------+
```

```
mysql> SELECT * FROM cards;
+----+-----------+---------+-------------+-------------+
| id | reader_id | book_id | taking_date | return_date |
+----+-----------+---------+-------------+-------------+
|  1 |         3 |       1 | 2015-10-05  | NULL        |
|  2 |         1 |       1 | 2016-12-15  | NULL        |
|  3 |         2 |       2 | 2017-12-15  | 2018-01-05  |
|  4 |         1 |       3 | 2014-07-10  | 2015-05-25  |
|  5 |         2 |       3 | 2016-07-10  | 2016-07-20  |
|  6 |         3 |       1 | 2012-08-12  | NULL        |
+----+-----------+---------+-------------+-------------+
```

3.5.1 Простой SELECT по одной таблице

Выбор всех записей (строк) из таблицы осуществляется запросом:

```
SELECT * FROM <table_name>;
```

Символ ***** указывает, что требуется выбрать значения из всех столбцов. Вместо ***** можно перечислить названия столбцов, которые требуется выбрать:

```
SELECT <column_1>, <column_2> FROM <table_name>;
```

Ключевое слово **WHERE** позволяет выбрать из таблицы записи, удовлетворяющие некоторому условию **condition**:

```
SELECT <column_1>, <column_2> FROM <table_name> WHERE <condition>;
```

В качестве условия обычно выступают, равенства, неравенства или сложные логические выражения. Ниже представлены некоторые примеры **<condition>**:

```
WHERE surname = 'Ivanov' равно
WHERE color <> 'RED' не равно
WHERE x > '10' AND x < '20' и
WHERE x < '10' OR x >= '20' или
```

Запрос, который выбирает идентификаторы и заголовки всех книг автора с **id=1**, выглядит следующим образом:

```
SELECT id, title FROM books WHERE author_id = '1';
```

```
+----+------------+
| id | title      |
+----+------------+
|  3 | Biology    |
|  4 | Philosophy |
+----+------------+
```

3.5.2 Сортировка ORDER BY

Следует учитывать, что информация в результате запроса представлена в неотсортированном виде. Ключевое слове **ORDER BY** позволяет отсортировать строки, извлеченные запросом в порядке возрастания **ASC** или убывания **DESC** по значениям из некоторого столбца.

Запишем предыдущий запрос, результаты которого будет отсортированы по названию книги в обратном порядке:

```
SELECT id, title FROM books WHERE author_id = '1' ORDER BY title DESC;
```

```
+----+------------+
| id | title      |
+----+------------+
|  4 | Philosophy |
|  3 | Biology    |
+----+------------+
```

3.5.3 Вложенный SELECT

Попробуем выбрать все книги автора **Alexandr One**. Таблица **books** не содержит фамилий, поэтому сперва придется узнать id интересующего автора:

```
SELECT id FROM authors WHERE name = 'Alexandr' AND surname = 'One';
```

```
+----+
| id |
+----+
|  1 |
+----+
```

Затем выбрать необходимые книги:

```
SELECT title FROM books WHERE author_id = 1;
```

```
+------------+
| title      |
+------------+
| Biology    |
| Philosophy |
+------------+
```

Имеется возможность первый запрос вложить внутрь второго запроса. Получится один запрос с вложенным **SELECT**:

```
SELECT title FROM books WHERE author_id = (SELECT id FROM authors
    WHERE name = 'Alexandr' AND surname = 'One');
```

Круглые скобки здесь обязательны.

Рассмотрим случай, когда вложенный **SELECT** возвращает несколько значений. Например, запрос выводящий фамилии всех читателей, бравших информатику:

```
SELECT surname FROM readers WHERE id = (SELECT reader_id FROM cards WHERE book_id
= '1');
```

Выполнение запроса завершится ошибкой:

```
ERROR 1242 (21000): Subquery returns more than 1 row
```

Следует заменить **id =** ... на **id IN …**:

```
SELECT surname FROM readers
    WHERE id IN (SELECT reader_id FROM cards WHERE book_id =
          (SELECT id FROM books WHERE title = 'Informatics')
);
```

```
+---------+
| surname |
+---------+
| Petrov  |
| Sidorov |
+---------+
```

3.5.4 Простой JOIN по двум таблицам. Псевдонимы AS

Если необходимо выбрать записи сразу из разных таблиц, прибегают к их соединению между собой. Например, требуется произвести выбор названий всех книг с указанием имени и фамилии автора. Информация об авторах и книгах хранится в разных таблицах. Когда в запросе задействовано несколько таблиц, название столбца указывается после имени таблицы через точку, например, **books.id**. В запросе **SELECT** следует объединить (**INNER JOIN**) две таблицы на (**ON**) основании внешнего ключа:

```
SELECT * FROM books INNER JOIN authors ON books.author_id = author.id;
```

```
mysql> SELECT * FROM authors;
+----+----------+----------+
| id | name     | surname  |
+----+----------+----------+
|  1 | Alexandr | One      |
|  2 | Leo      | Greed    |
|  3 | Ya       | Yakahoma |
+----+----------+----------+
```

```
                books                                  authors
+----+-----------+-------------+----+----------+----------+
| id | author_id | title       | id | name     | surname  |
+----+-----------+-------------+----+----------+----------+
|  3 |         1 | Biology     |  1 | Alexandr | One      |
|  4 |         1 | Philosophy  |  1 | Alexandr | One      |
|  1 |         2 | Informatics |  2 | Leo      | Greed    |
|  2 |         3 | Geology     |  3 | Ya       | Yakahoma |
+----+-----------+-------------+----+----------+----------+
```

В итоговую таблицу для каждой книги размещается информация о её авторе, значения для полей **author** берутся из равенства **author.id = books.author_id**. На схеме цветом выделены значения, по которым происходит присоединение записей из таблицы **authors**. Ключевое слово **INNER** указывает, что требуется выбрать только те книги, для которых существуют авторы. Если автор не найден в таблице **authors**, книга не будет включена в выборку.

Другими словами, выражение:

```
INNER JOIN authors ON author.id = books.author_id
```

означает пересечение множеств **books** и **athors**. Записи из двух таблиц «склеиваются» по критерию **books.author_id = author.id**. Остальные записи в выборку не попадут, например, книги для которых в таблице **authors** не были найдены авторы. Если у книги несколько авторов, то она попадет в выборку дважды.

Запрос, который выбирает только интересующие нас поля:

```
SELECT b.title AS book_title, a.name AS name, a.surname AS surname
    FROM books AS b
    INNER JOIN authors AS a ON b.author_id = a.id;
```

Ключевое слово **AS** позволяет задать более удобное имя – **псевдоним** - для соответствующей таблицы или столбца. Обратите внимание, что теперь в результирующей таблице отображаются псевдонимы вместо названия столбцов:

```
+-------------+----------+----------+
| book_title  | name     | surname  |
+-------------+----------+----------+
| Biology     | Alexandr | One      |
| Philosophy  | Alexandr | One      |
| Informatics | Leo      | Greed    |
| Geology     | Ya       | Yakahoma |
+-------------+----------+----------+
```

3.5.5 Сложный JOIN по трем таблицам с вложенным SELECT

Стоит задача выбрать все книги, которые когда-либо брал **Ivan Pupkin**. Посмотрим ещё раз на имеющуюся БД:

```
mysql> SELECT * FROM readers;
+----+-------+---------+
| id | name  | surname |
+----+-------+---------+
|  1 | Petr  | Petrov  |
|  2 | Ivan  | Pupkin  |
|  3 | Yurii | Sidorov |
+----+-------+---------+
```

```
mysql> SELECT * FROM authors;
+----+----------+----------+
| id | name     | surname  |
+----+----------+----------+
|  1 | Alexandr | One      |
|  2 | Leo      | Greed    |
|  3 | Ya       | Yakahoma |
+----+----------+----------+
```

```
mysql> SELECT * FROM books;
+----+-----------+-------------+
| id | author_id | title       |
+----+-----------+-------------+
|  1 |         2 | Informatics |
|  2 |         3 | Geology     |
|  3 |         1 | Biology     |
|  4 |         1 | Philosophy  |
+----+-----------+-------------+
```

```
mysql> SELECT * FROM cards;
+----+-----------+---------+-------------+-------------+
| id | reader_id | book_id | taking_date | return_date |
+----+-----------+---------+-------------+-------------+
|  1 |         3 |       1 | 2015-10-05  | NULL        |
|  2 |         1 |       1 | 2016-12-15  | NULL        |
|  3 |         2 |       2 | 2017-12-15  | 2018-01-05  |
|  4 |         1 |       3 | 2014-07-10  | 2015-05-25  |
|  5 |         2 |       3 | 2016-07-10  | 2016-07-20  |
|  6 |         3 |       1 | 2012-08-12  | NULL        |
+----+-----------+---------+-------------+-------------+
```

Чтобы выбрать все книги, которые читал **Ivan Pupkin** следует выполнить следующие действия:

#1. Узнать **id** читателя **Ivan Pupkin**

#2. Найти в таблице **cards** идентификаторы **book_id** всех книг для читателя с **id** из п.1

#3. Присоединение к таблице **cards** таблицы **books**, с целью по **book_id** определить название книги.

#4. Присоединение к сводной таблице **cards-books** таблицы **authors**, с целью по **author_id** определить имя автора.

Оператор **JOIN** объединяет несколько таблиц в одну временную таблицу. В неё будут включены все строки из таблиц **cards**, **authors**, **books**, которые удовлетворяют условиям, указанным после ключевого слова **ON**. Сперва объединяется **cards** с **books**, затем к ним добавляется **authors**.

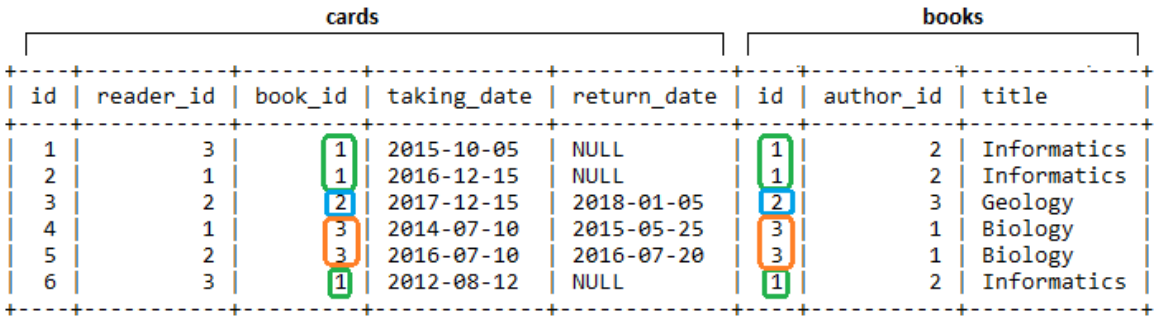
Начнем разбирать устройство сложного запроса с третьего пункта.

#3. Присоединение к таблице cards таблицы books

Выполним выборку всех записей из таблицы books и присоединим к ним cards:

```
SELECT * FROM cards INNER JOIN books ON cards.book_id = books.id;
```

Столбцы из таблицы **cards** добавляются к таблице **books**. Объединяются строки, у которых значение **id** из таблицы **books** и значение **book_id** из таблицы **cards** совпадают (**cards.book_id = books.id**). Обратите внимание, что книга с **id = 4** не добавляется (т. к. значение **4** остуствует в столбце **cards.book_id**), а некоторые книги присутствуют по несколько раз (поскольку их брали разные читатели):

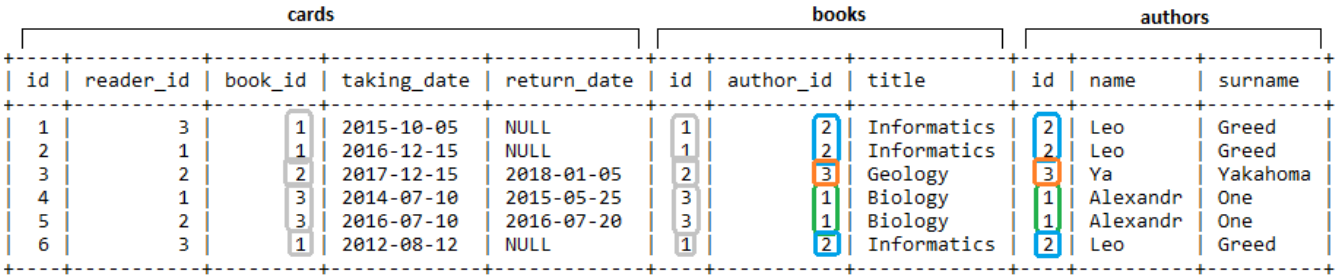


id	reader_id	book_id	taking_date	return_date	id	author_id	title
1	3	1	2015-10-05	NULL	1	2	Informatics
2	1	1	2016-12-15	NULL	1	2	Informatics
3	2	2	2017-12-15	2018-01-05	2	3	Geology
4	1	3	2014-07-10	2015-05-25	3	1	Biology
5	2	3	2016-07-10	2016-07-20	3	1	Biology
6	3	1	2012-08-12	NULL	1	2	Informatics

#4. Присоединение к сводной таблице cards-books таблицу authors

К таблице, получившейся после первого **`JOIN`**, добавим столбцы, состоящие из записей таблицы **`authors`**, у которых **`id`** совпадает с **`books.author_id`**:

```
SELECT * FROM cards
    INNER JOIN books ON cards.book_id = books.id
    INNER JOIN authors ON books.author_id = authors.id;
```



id	reader_id	book_id	taking_date	return_date	id	author_id	title	id	name	surname
1	3	1	2015-10-05	NULL	1	2	Informatics	2	Leo	Greed
2	1	1	2016-12-15	NULL	1	2	Informatics	2	Leo	Greed
3	2	2	2017-12-15	2018-01-05	2	3	Geology	3	Ya	Yakahoma
4	1	3	2014-07-10	2015-05-25	3	1	Biology	1	Alexandr	One
5	2	3	2016-07-10	2016-07-20	3	1	Biology	1	Alexandr	One
6	3	1	2012-08-12	NULL	1	2	Informatics	2	Leo	Greed

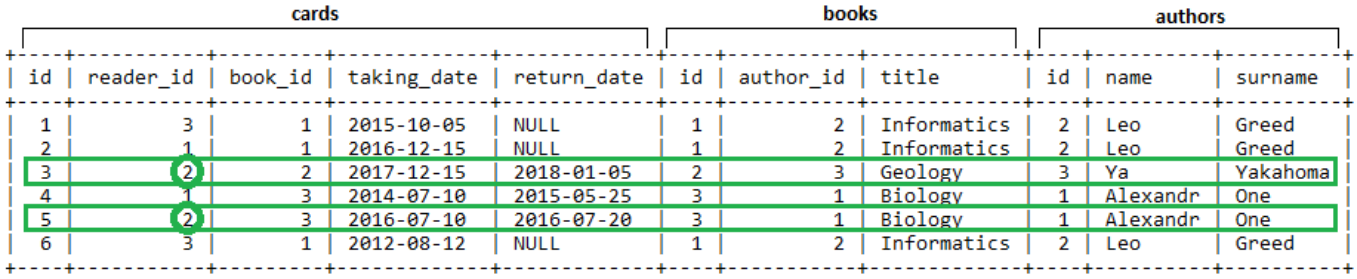
#1. Узнать id читателя Ivan Pupkin

```
SELECT id FROM readers WHERE name = 'Ivan' AND surname = 'Pupkin';
```

#2. Найти в таблице cards идентификаторы book_id всех книг для читателя с id из п.1

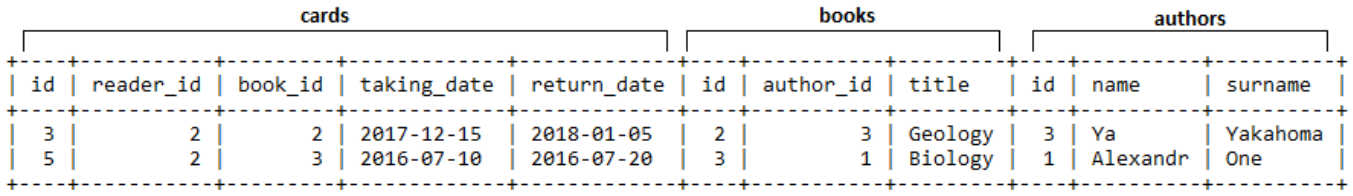
```
SELECT * FROM cards
    INNER JOIN books ON cards.book_id = books.id
    INNER JOIN authors ON books.author_id = authors.id
WHERE cards.reader_id IN(
    SELECT id FROM readers WHERE name = 'Ivan' AND surname = 'Pupkin'
);
```

Вложенный **`SELECT`** определяет, что **`id`** читателя **`"Ivan Pupkin"`** равен **`2`**. Из объединенной таблицы **`cards-books-authors`** выбираются (**`WHERE cards.reader_id IN`**) строки у которых **`reader_id = 2`**:



id	reader_id	book_id	taking_date	return_date	id	author_id	title	id	name	surname
1	3	1	2015-10-05	NULL	1	2	Informatics	2	Leo	Greed
2	1	1	2016-12-15	NULL	1	2	Informatics	2	Leo	Greed
3	2	2	2017-12-15	2018-01-05	2	3	Geology	3	Ya	Yakahoma
4	1	3	2014-07-10	2015-05-25	3	1	Biology	1	Alexandr	One
5	2	3	2016-07-10	2016-07-20	3	1	Biology	1	Alexandr	One
6	3	1	2012-08-12	NULL	1	2	Informatics	2	Leo	Greed

Результат выполнения запроса будет следующим:



cards

id	reader_id	book_id	taking_date	return_date
3	2	2	2017-12-15	2018-01-05
5	2	3	2016-07-10	2016-07-20

books

id	author_id	title
2	3	Geology
3	1	Biology

authors

id	name	surname
3	Ya	Yakahoma
1	Alexandr	One

Окончательный запрос с использованием псевдонимов будет выглядеть следующим образом:

```
SELECT b.id AS book_id, b.title AS book_title,
    a.name AS author_name, a.surname AS author_surname
FROM cards AS c
    INNER JOIN books AS b ON c.book_id = b.id
    INNER JOIN authors AS a ON b.author_id = a.id
WHERE c.reader_id IN
        (SELECT id FROM readers WHERE name='Ivan' AND surname='Pupkin');
```

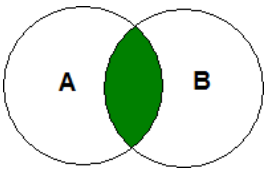
book_id	book_title	author_name	author_surname
2	Geology	Ya	Yakahoma
3	Biology	Alexandr	One

3.5.6 Разновидности JOIN

- **INNER JOIN**. В примере выше, использовался **INNER JOIN**. Данный оператор объединяет строки, у которых значения в некоторых столбцах совпадают. Остальные строки отсекаются:

```
SELECT FROM A INNER JOIN B ON <condition>
```

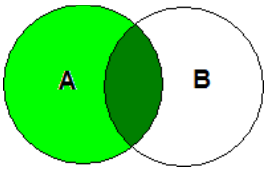
На диаграмме Венна **INNER JOIN** соответствует пересечению множеств. Зеленым цветом показаны строки, которые будут выбраны в результате запроса.



- **LEFT JOIN**. Например:

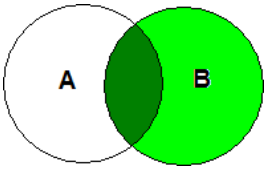
```
SELECT FROM A LEFT JOIN B ON <condition>
```

выбираются все строки из таблицы **A**, к ней присоединяются строки из таблицы **B**, удовлетворяющие некоторому условию (на рисунке показаны темным цветом). Для строк, не соответствующих условию **<condition>**, вместо значений из таблицы **B** заносится значение **NULL** (на рисунке им соответствует светло-зеленая область).



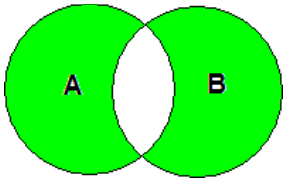
- **RIGTH JOIN**, аналогично:

```
SELECT FROM A RIGHT JOIN B ON <condition>
```



- **OUTER JOIN**, выбирается всё, что не входит в **INNER JOIN**

```
SELECT FROM A OUTER JOIN B ON <condition>
```



- **CROSS JOIN**. Осуществляется связь всех строк со всеми, перекрестное соединение или декартово произведение. Обратите внимание на отсутствие условия:

```
SELECT FROM A CROSS JOIN B
```

3.6 Объединение в группы

Представим, что в библиотеке появился склад. Имеется следующая таблица, отражающая количество книг на складе:

```
mysql> SELECT * FROM storage;
+----+-----------------------------+-------------------------------------------+------+----------------+
| id | author                      | title                                     | cnt  | publisher      |
+----+-----------------------------+-------------------------------------------+------+----------------+
|  1 | Пушкин А.С.                 | Пророк                                    |    4 | Астра          |
|  2 | Достоевский Ф.М.            | Преступление и наказание                  |   10 | Астра          |
|  3 | Достоевский Ф.М.            | Игрок                                     |   12 | Астра          |
|  4 | Пушкин А.С.                 | Дубровский                                |   11 | Астра          |
|  5 | Достоевский Ф.М.            | Идиот                                     |    4 | Креатив        |
|  6 | Тургенев И.С.               | Муму                                      |   12 | Оксфорд        |
|  7 | Пушкин А.С.                 | Капитанская дочка                         |   10 | Астра          |
|  8 | Толстой Л.Н.                | Война и мир                               |   10 | Креатив        |
|  9 | Тургенев И.С.               | Отцы и дети                               |    7 | Оксфорд        |
+----+-----------------------------+-------------------------------------------+------+----------------+
9 rows in set (0,00 sec)
```

Перед нами стоит задача, вывести список авторов, чьи книги имеются на складе.

3.6.1 Группировка GROUP BY

Запрос **`SELECT author FROM storage;`** выдаст все записи из столбца **`author`**, при чём авторы будут повторяться в списке несколько раз:

```
+-----------------------------+
| author                      |
+-----------------------------+
| Пушкин А.С.                 |
| Достоевский Ф.М.            |
| Достоевский Ф.М.            |
| Пушкин А.С.                 |
| Достоевский Ф.М.            |
| Тургенев И.С.               |
| Пушкин А.С.                 |
| Толстой Л.Н.                |
| Тургенев И.С.               |
+-----------------------------+
```

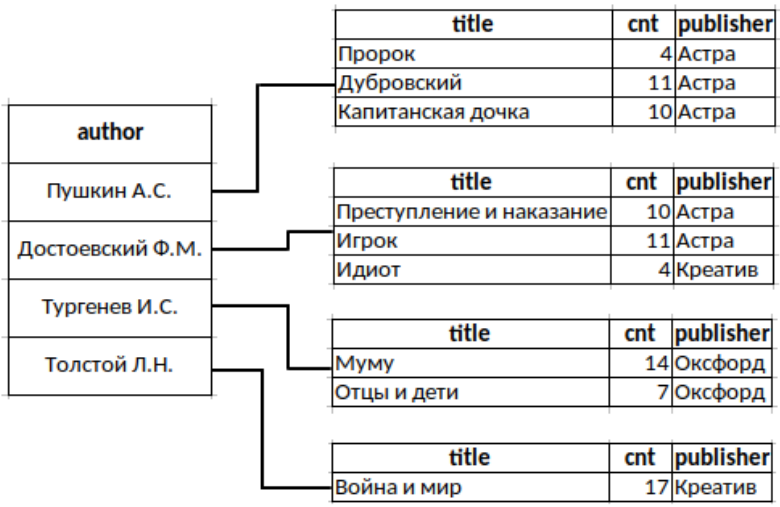
Ключевое слово **`GROUP BY`** позволяет собрать данные в группы, тем самым исключив повторы. В нашем случае, группировать будем по авторам:

```
SELECT author FROM storage GROUP BY author;
```

Результатом будет список без повторов:

```
+-----------------------------+
| author                      |
+-----------------------------+
| Пушкин А.С.                 |
| Достоевский Ф.М.            |
| Тургенев И.С.               |
| Толстой Л.Н.                |
+-----------------------------+
```

Группировку можно представить следующим образом:



3.6.2 Агрегатные функции

После группировки записей мы не можем получить доступ к отдельным полям, таким как **`title`** или **`cnt`**. Зато имеется возможность выбрать некоторую информацию, характерную для всей группы в целом. В основном сюда относят статистическую информацию, например, касающуюся каждого автора. Для сбора информации о группе (авторе) используют агрегатные функции, такие как:

- **`COUNT()`** - подсчет количества
- **`SUM()`** - подсчет суммы
- **`AVG()`** - подсчет среднего арифметического
- **`MIN()`** - минимальное значение
- **`MAX()`** - максимальное значение.

Агрегатная функция выполняет вычисление на наборе значений и возвращает одиночное значение. На вход агрегатной функции, следует передать название столбца, по которому функция будет вычислять значения для каждой группы (для каждого автора).

Например, следующий запрос позволяет вывести для каждого автора, общее количество его книг, имеющихся на складе.

```
SELECT author, SUM(cnt) FROM storage GROUP BY author;
```

```
+------------------------------+----------+
| author                       | SUM(cnt) |
+------------------------------+----------+
| Пушкин А.С.                  |       25 |
| Достоевский Ф.М.             |       26 |
| Тургенев И.С.                |       19 |
| Толстой Л.Н.                 |       10 |
+------------------------------+----------+
```

Обратите внимание, что информация в таблице представлена в неотсортированном виде.

Следующий запрос выдаст результат в порядке возрастания количества книг:

```
SELECT author, SUM(cnt) FROM storage GROUP BY author ORDER BY SUM(cnt) ASC;
```

```
+------------------------------+----------+
| author                       | SUM(cnt) |
+------------------------------+----------+
| Толстой Л.Н.                 |       10 |
| Тургенев И.С.                |       19 |
| Пушкин А.С.                  |       25 |
| Достоевский Ф.М.             |       26 |
+------------------------------+----------+
```

Часто требуется подсчитывать количество элементов в группе. Запрос, выводящий количество различных книг каждого автора, выглядит следующим образом:

```
SELECT author, COUNT(title) FROM storage GROUP BY author;
```

```
+------------------------------+--------------+
| author                       | COUNT(title) |
+------------------------------+--------------+
| Пушкин А.С.                  |            3 |
| Достоевский Ф.М.             |            3 |
| Тургенев И.С.                |            2 |
| Толстой Л.Н.                 |            1 |
+------------------------------+--------------+
```

3.6.3 Фильтрация HAVING

Аналогом **`WHERE`** для групп является оператор **`HAVING`**. Он позволяет произвести фильтрацию значений. Если **`WHERE`** фильтрует значения до группировки, то **`HAVING`** после группировки.

Запрос, который выводит авторов, для которых количество книг больше **`20`**, выглядит следующим образом:

```
SELECT author, SUM(cnt) FROM storage GROUP BY author HAVING SUM(cnt) > 20;
```

```
+------------------------------+----------+
| author                       | SUM(cnt) |
+------------------------------+----------+
| Пушкин А.С.                  |       25 |
| Достоевский Ф.М.             |       26 |
| Тургенев И.С.                |       19 |
+------------------------------+----------+
```

Лев Николаевич Толстой не был выбран, потому что на складе только **`7`** книг.

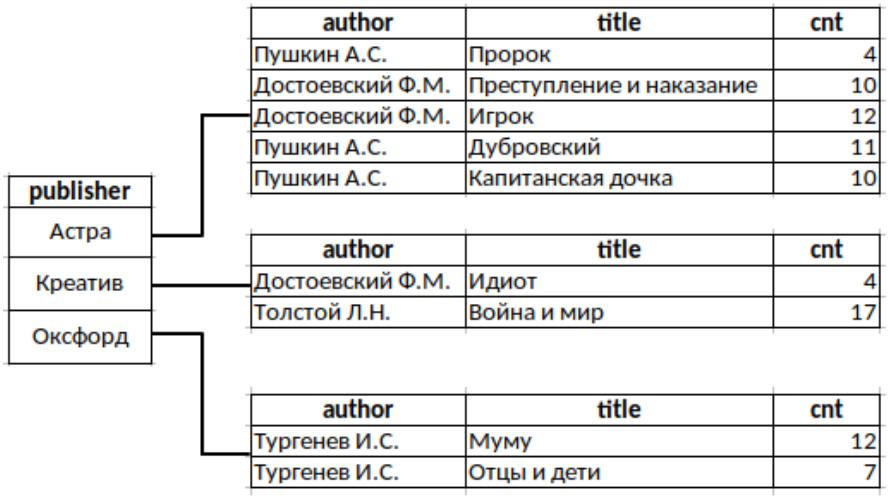
3.6.4 Выбор уникальных записей DISCTINCT

Выведем для каждого издательства количество авторов книг, имеющихся на складе.

```
SELECT publisher, COUNT(author) FROM storage GROUP BY publisher;
```

```
+----------------+---------------+
| publisher      | COUNT(author) |
+----------------+---------------+
| Астра          |             5 |
| Креатив        |             2 |
| Оксфорд        |             2 |
+----------------+---------------+
```

Ощущение, что очуществился подсчет количества книг, а не авторов. Рассмотрим почему это произошло. Группровка **`GROUP BY publisher`** выглядит следующим образом:



`Достоевский` и **`Тургенев`** были учтены при подсчете два раза, а **`Пушкин`** целых три. Ключевое **`DISTINCT`** позволяет агрегатной функции учитывать только уникальные записи. Корректный запрос выглядит следующим образом:

```
+----------------+------------------------+
| publisher      | COUNT(DISTINCT author) |
+----------------+------------------------+
| Астра          |                      2 |
| Креатив        |                      2 |
| Оксфорд        |                      1 |
+----------------+------------------------+
```

Теперь каждый автор подсчитан единожды.

3.7 Ограничение вывода LIMIT

Оператор **LIMIT** позволяет извлечь определенное количество строк. Оператору **LIMIT** можно передать один или два параметра:

- **LIMIT 10** – отобразит первые 10 строк
- **LIMIT 5, 10** – пропустит 5 строк и отобразит следующие 10 строк.

Запросы, выводящие все записи, с 1 по 3 и с 4 по 8 в порядке возрастания названия книг:

```
SELECT * FROM storage ORDER BY title ASC;
```

id	author	title	cnt	publisher
8	Толстой Л.Н.	Война и мир	10	Креатив
4	Пушкин А.С.	Дубровский	11	Астра
3	Достоевский Ф.М.	Игрок	12	Астра
5	Достоевский Ф.М.	Идиот	4	Креатив
7	Пушкин А.С.	Капитанская дочка	10	Астра
6	Тургенев И.С.	Муму	12	Оксфорд
9	Тургенев И.С.	Отцы и дети	7	Оксфорд
2	Достоевский Ф.М.	Преступление и наказание	10	Астра
1	Пушкин А.С.	Пророк	4	Астра

```
SELECT * FROM storage ORDER BY title ASC LIMIT 3;
```

id	author	title	cnt	publisher
8	Толстой Л.Н.	Война и мир	10	Креатив
4	Пушкин А.С.	Дубровский	11	Астра
3	Достоевский Ф.М.	Игрок	12	Астра

```
SELECT * FROM storage ORDER BY title ASC LIMIT 3, 5;
```

id	author	title	cnt	publisher
5	Достоевский Ф.М.	Идиот	4	Креатив
7	Пушкин А.С.	Капитанская дочка	10	Астра
6	Тургенев И.С.	Муму	12	Оксфорд
9	Тургенев И.С.	Отцы и дети	7	Оксфорд
2	Достоевский Ф.М.	Преступление и наказание	10	Астра

Интересный способ вывести книгу, которая присутствует в максимальном количестве, использовать сортировку по убыванию **DESC** и **LIMIT**:

```
SELECT title FROM storage ORDER BY cnt DESC LIMIT 1;
```

title
Игрок

3.8 Совместное использование JOIN и GROUP BY

Вернемся со склада в библиотеку. Используя группировку, выполним несколько запросов, выводящих статистическую информацию.

```
mysql> SELECT * FROM readers;
+----+-------+---------+
| id | name  | surname |
+----+-------+---------+
|  1 | Petr  | Petrov  |
|  2 | Ivan  | Pupkin  |
|  3 | Yurii | Sidorov |
+----+-------+---------+
```

```
mysql> SELECT * FROM authors;
+----+----------+----------+
| id | name     | surname  |
+----+----------+----------+
|  1 | Alexandr | One      |
|  2 | Leo      | Greed    |
|  3 | Ya       | Yakahoma |
+----+----------+----------+
```

```
mysql> SELECT * FROM books;
+----+-----------+-------------+
| id | author_id | title       |
+----+-----------+-------------+
|  1 |         2 | Informatics |
|  2 |         3 | Geology     |
|  3 |         1 | Biology     |
|  4 |         1 | Philosophy  |
+----+-----------+-------------+
```

```
mysql> SELECT * FROM cards;
+----+-----------+---------+-------------+-------------+
| id | reader_id | book_id | taking_date | return_date |
+----+-----------+---------+-------------+-------------+
|  1 |         3 |       1 | 2015-10-05  | NULL        |
|  2 |         1 |       1 | 2016-12-15  | NULL        |
|  3 |         2 |       2 | 2017-12-15  | 2018-01-05  |
|  4 |         1 |       3 | 2014-07-10  | 2015-05-25  |
|  5 |         2 |       3 | 2016-07-10  | 2016-07-20  |
|  6 |         3 |       1 | 2012-08-12  | NULL        |
+----+-----------+---------+-------------+-------------+
```

Составим запрос, который выводит автора и количество книг. Задача осложнена тем, что группировку будем осуществлять по **`books.author_id`**, а фамилии авторов содержатся в таблице **`authors`**. Здесь нам поможет **`INNER JOIN`**:

```
SELECT authors.name, COUNT(books.author_id)
    FROM books INNER JOIN authors ON authors.id = books.author_id
GROUP BY books.author_id;
```

4. Дамп и Восстановление

Перед созданием резервной копии БД (дампа) необходимо выйти из приложения **mysql**, выполнив команду **exit**. Создание дампа осуществляется программой **mysqldump**:

```
mysqldump –u<user_name> -p -h<host_name> <database_name> > C:\dumps\dump.sql
```

Структура и содержимое БД **database_name** сохранится по указанному пути в файл **C:\dumps\dump.sql.** Файл дампа — обыкновенный текстовый файл, содержащий SQL команды, выполнение которых позволяет воссоздать все имеющиеся таблицы и записи.

Восстановить содержимое базы данных из файла дампа **c:\dumps\dump.sql** в уже существующую БД с именем **<db_name>** можно при подключении с использованием команды:

```
mysql –u<user_name> -p –h<host_name> <db_name> < c:\dumps\dump.sql
```

Обратите внимание, что после выполнение команды вход в mysql не осуществляется. Необходимо зайти в mysql в штатном режиме:

```
mysql –u<user_name> -p -h<host_name>
```

Если БД не существует, необходимо создать и выбрать её, а затем загрузить дамп с использованием ключевого слова **source**:

```
mysql –u<user_name> -p -h<host_name>
CREATE DATABASE <db_name>;
USE <db_name>;
source c:\dumps\dump.sql;
```

Эпилог

Были рассмотрены основные понятия, касающиеся темы «Базы данных» и практические приемы работы с **DBMS MySQL**. Полученные знания могут быть применены в различных областях, например, при разработке сайтов. База данных одна из составляющих любого интернет-портала. Помимо БД необходимо изучить **back-end** часть, например, язык программирования **PHP** и **front-end** часть — **HTML/CSS** и **JavaScript**. Можно зарегистрироваться на бесплатном хостинге, который помимо БД **MySQL** предоставляет и **Apache** сервер для запуска **PHP** скриптов, или же установить сервер локально на компьютер (см. **Denwer**). Локальные сервера удобно использовать при изучении и разработке. Готовый код и БД позже можно разместить уже на удаленном хостинге.

PHP — язык программирования, который позволяет делать запросы к БД, извлекать из БД информацию, генерировать **HTML** страницы. Структура **PHP** скрипта примерно следующая:

1. Подключение к БД:

```
mysql_connect("mysql_host", "mysql_user", "mysql_password");
mysql_select_db("db_name");
```

2. Запрос на выборку:

```
$res = mysql_query("SELECT … FROM … WHERE … ");
while($row = mysql_fetch_assoc($res)) {
      $data_from_db[$i++] = $row['column_name'];
}
```

3. Генерация **html** страницы с ипользованием полученной из БД информацией:

```
echo "<html>..".$data_from_db[$i]."..</html>";
```

Передавать данные в **PHP** скрипт можно в адресной строке (**GET-запрос**):

```
http://mysite.com/hello.php?data=some_data&data_2=12345
```

Тогда добавление строки **"some_data"** в БД может выглядеть следующим образом:

```
mysql_connect("...");
mysql_select_db("...");
$var_data = mysql_real_escape_string($_GET["data"]);
mysql_query("INSERT INTO tb_name(col_name) VALUES('$var_data')");
```

Также передавать данные в скрипт можно через тег **<form> (GET или POST запрос).**

Не раскидывайтесь данными понапрасну...