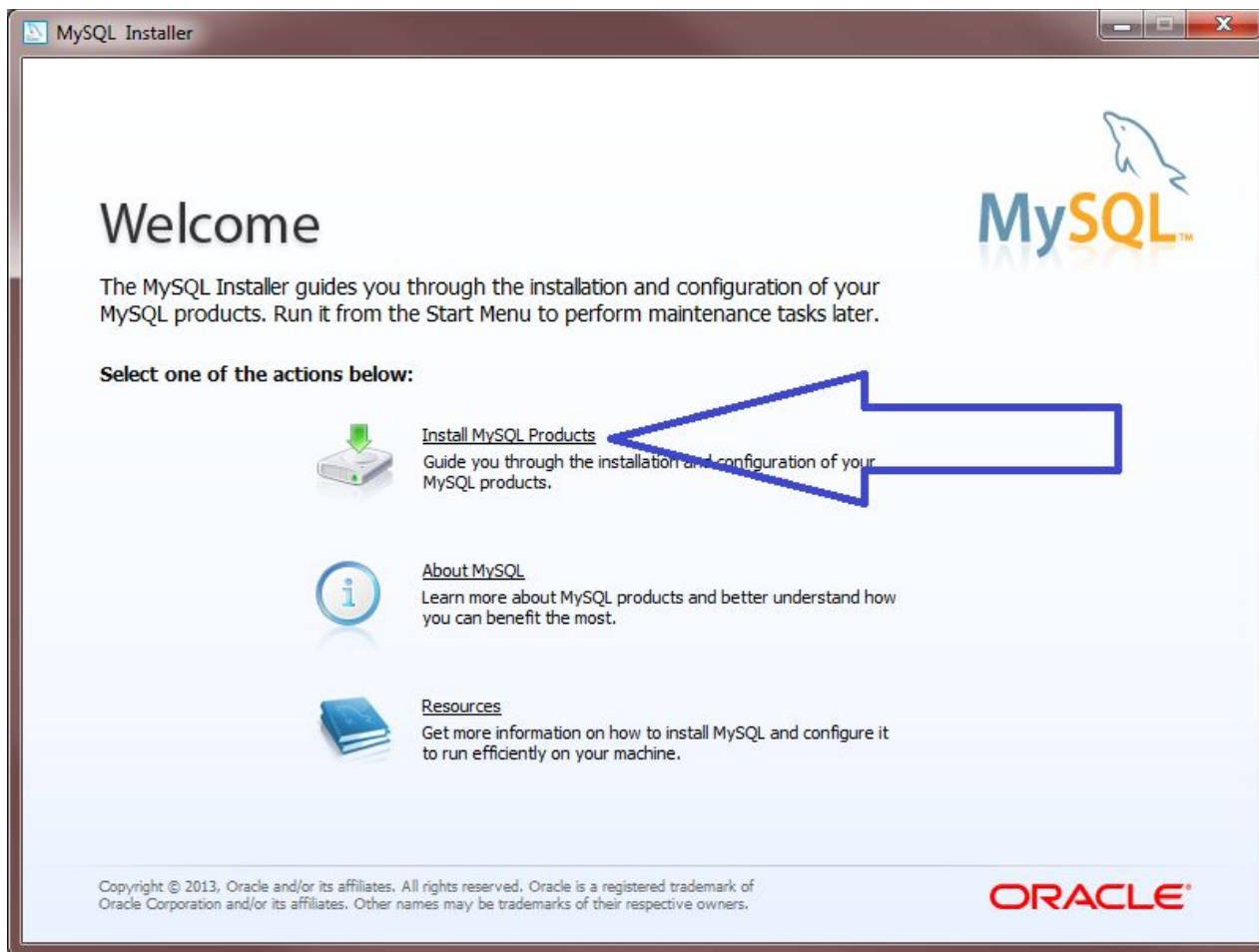


JDBC

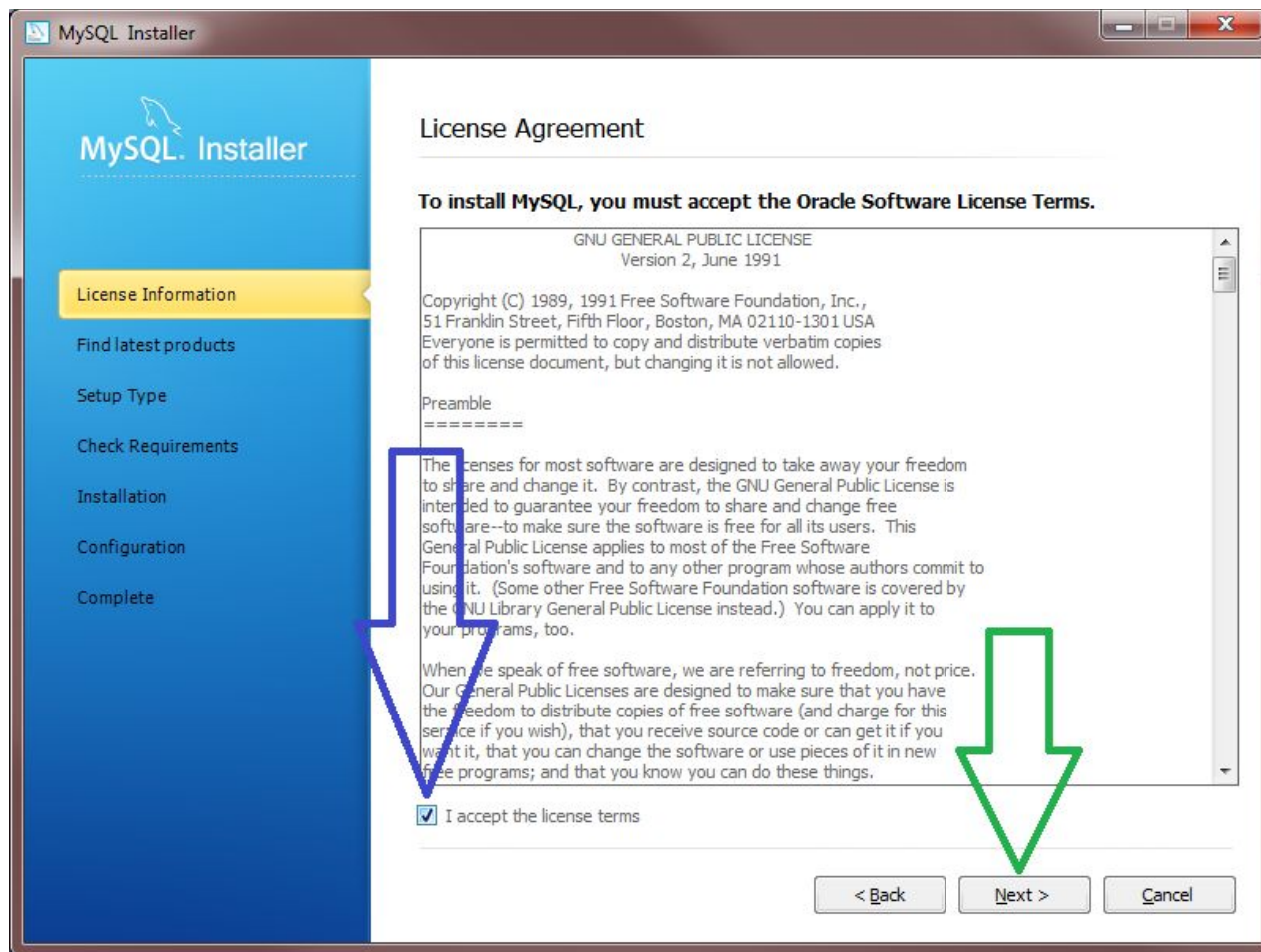
# My SQL – установка

## Шаг-1



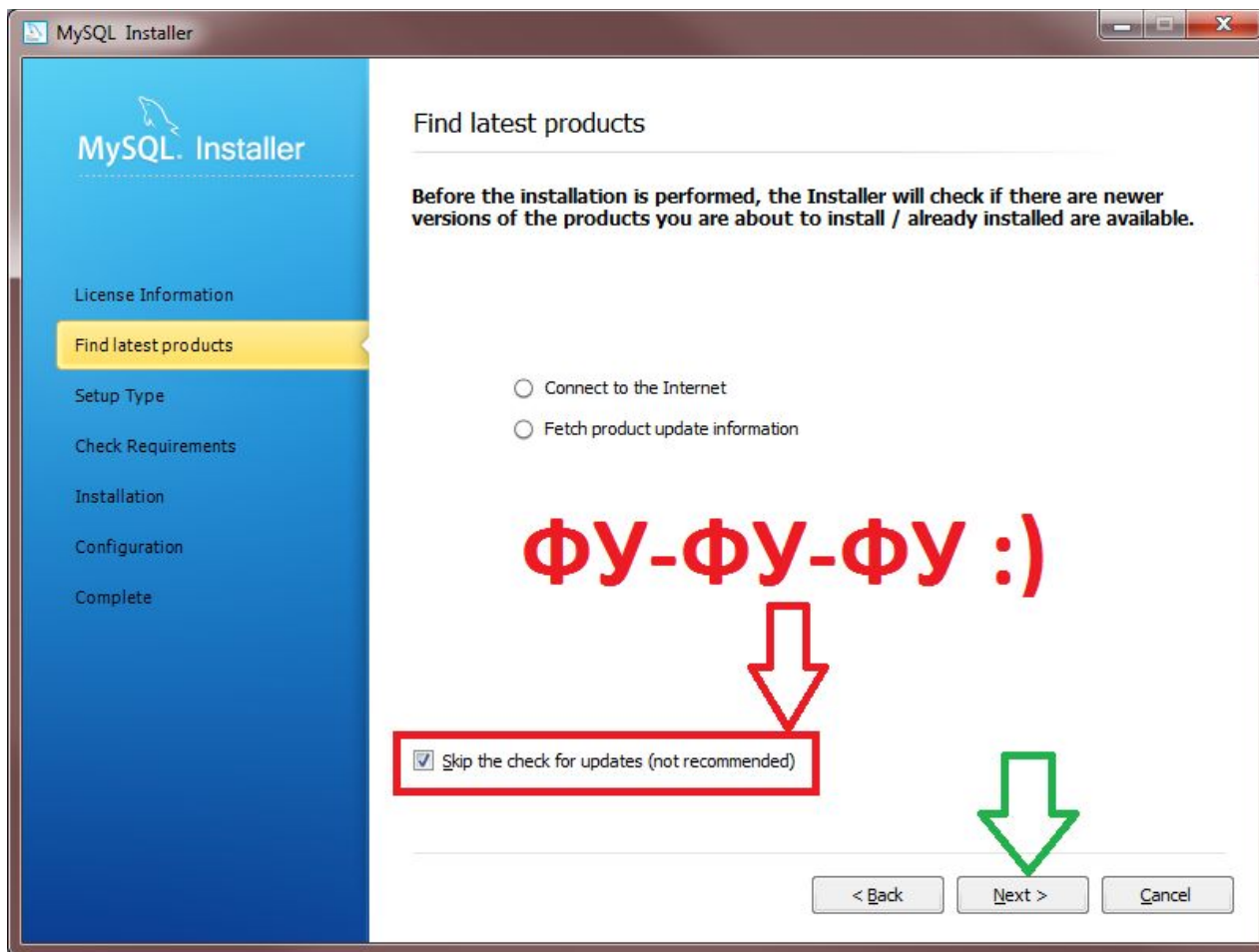
# My SQL – установка

## Шаг-2



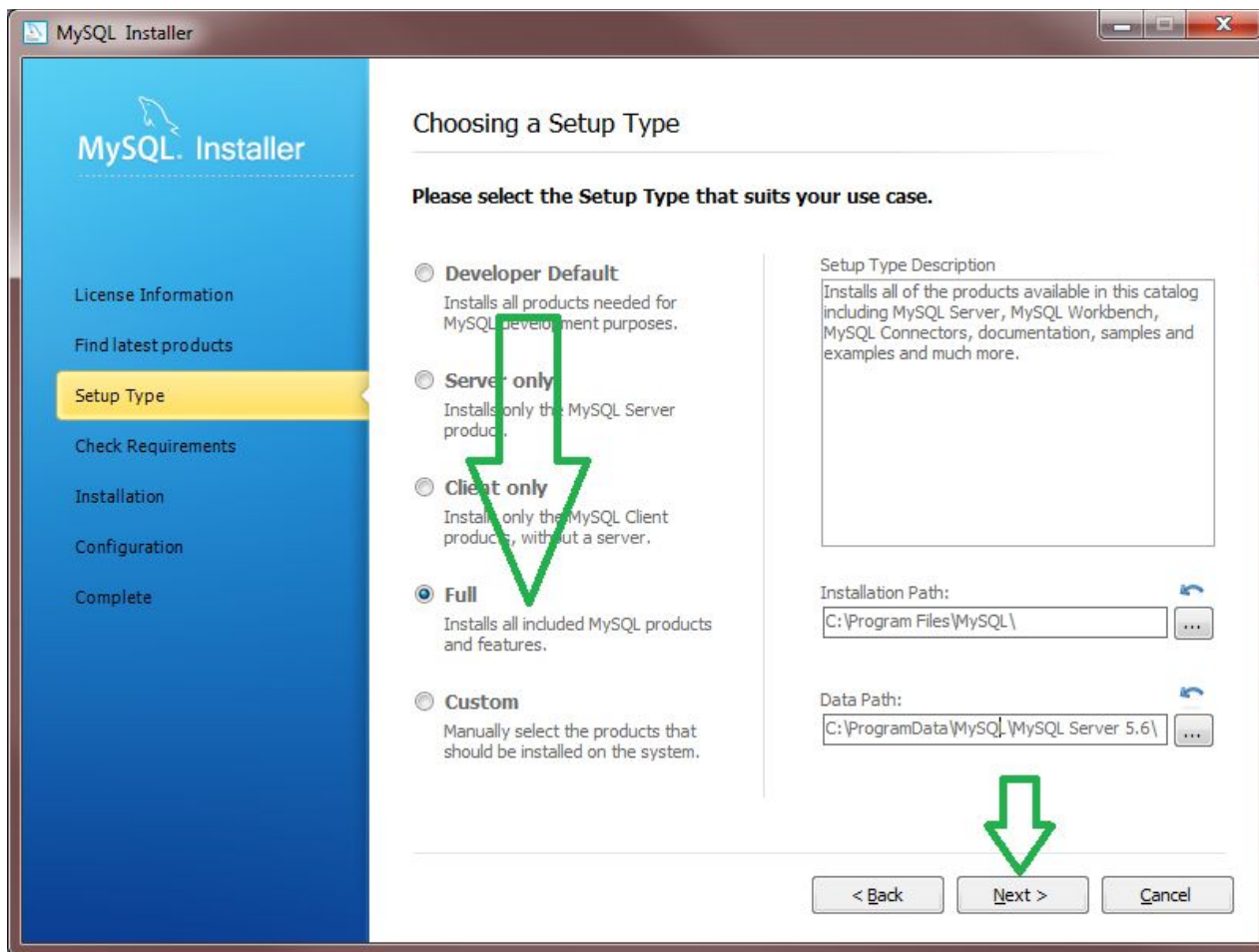
# My SQL – установка

## Шаг-3



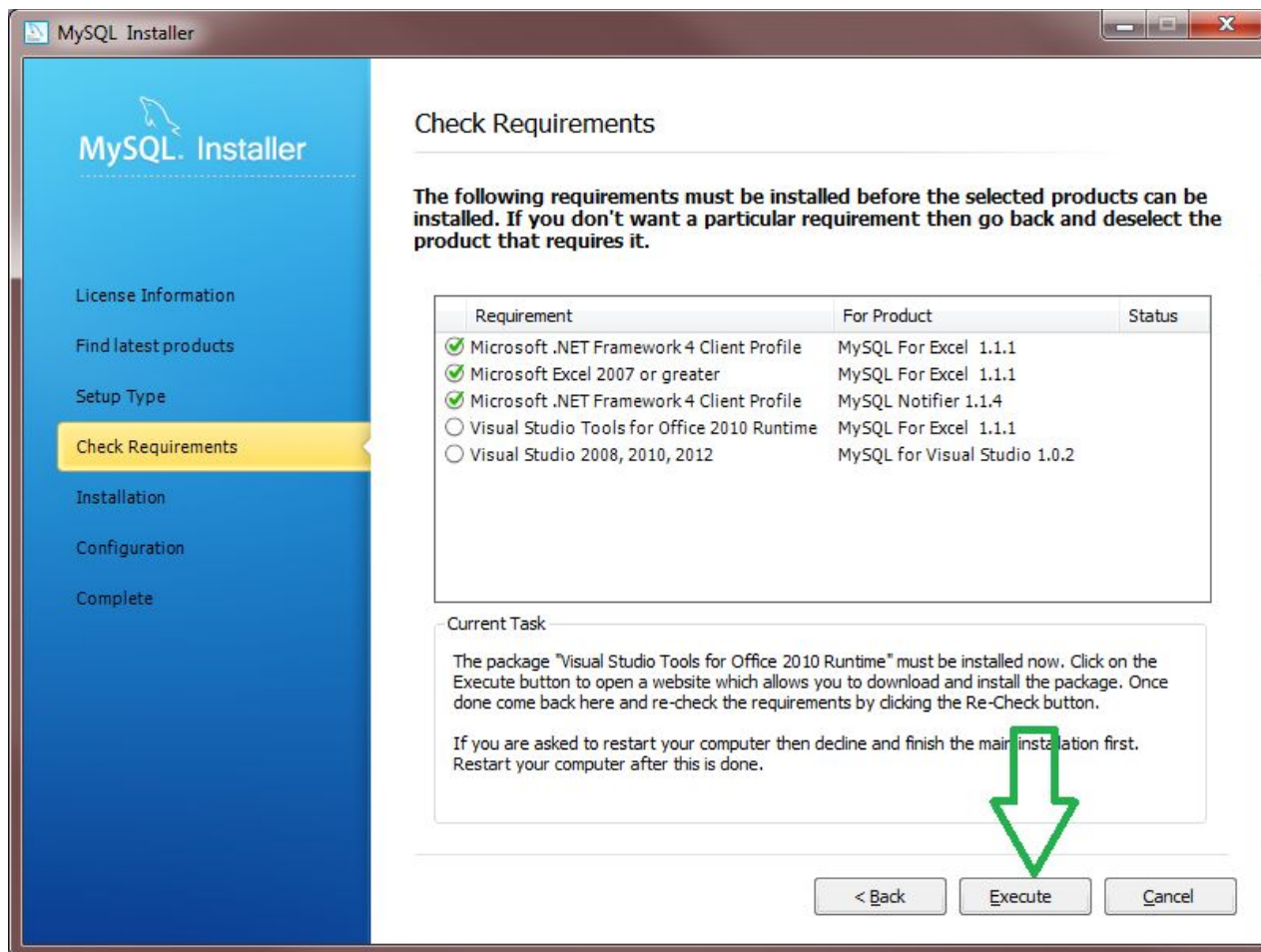
# My SQL – установка

## Шаг-4



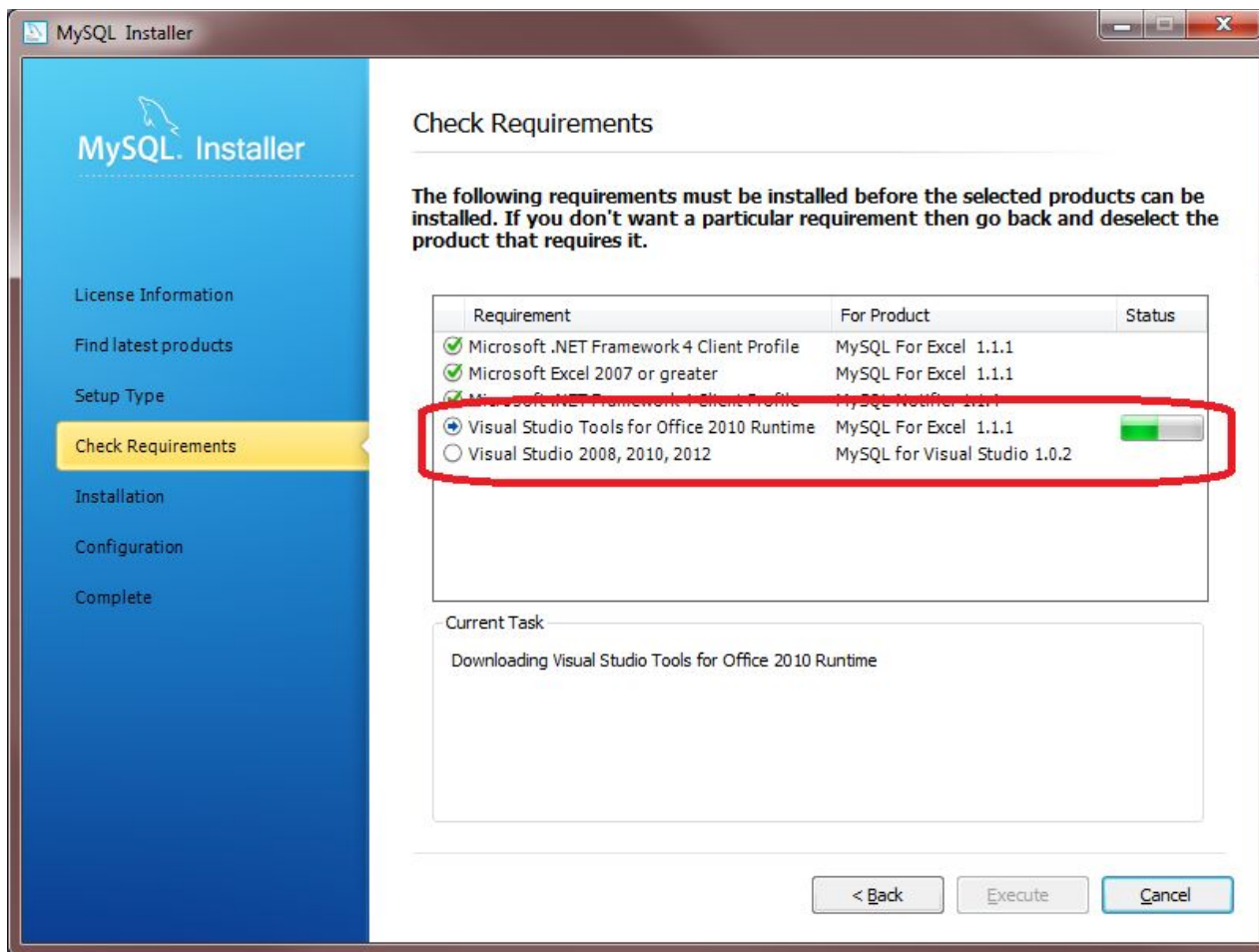
# My SQL – установка

## Шаг-5



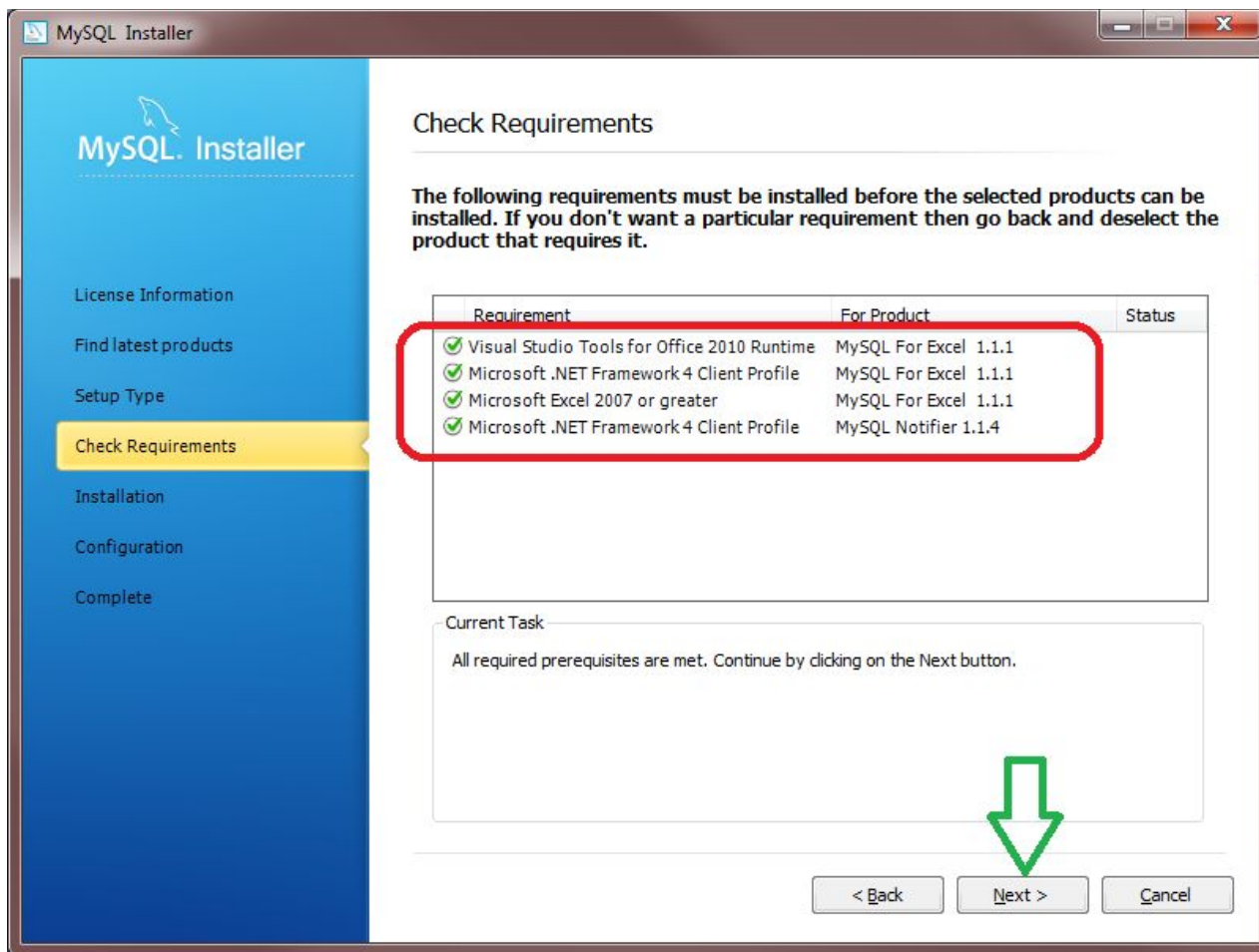
# My SQL – установка

## Шаг-6



# My SQL – установка

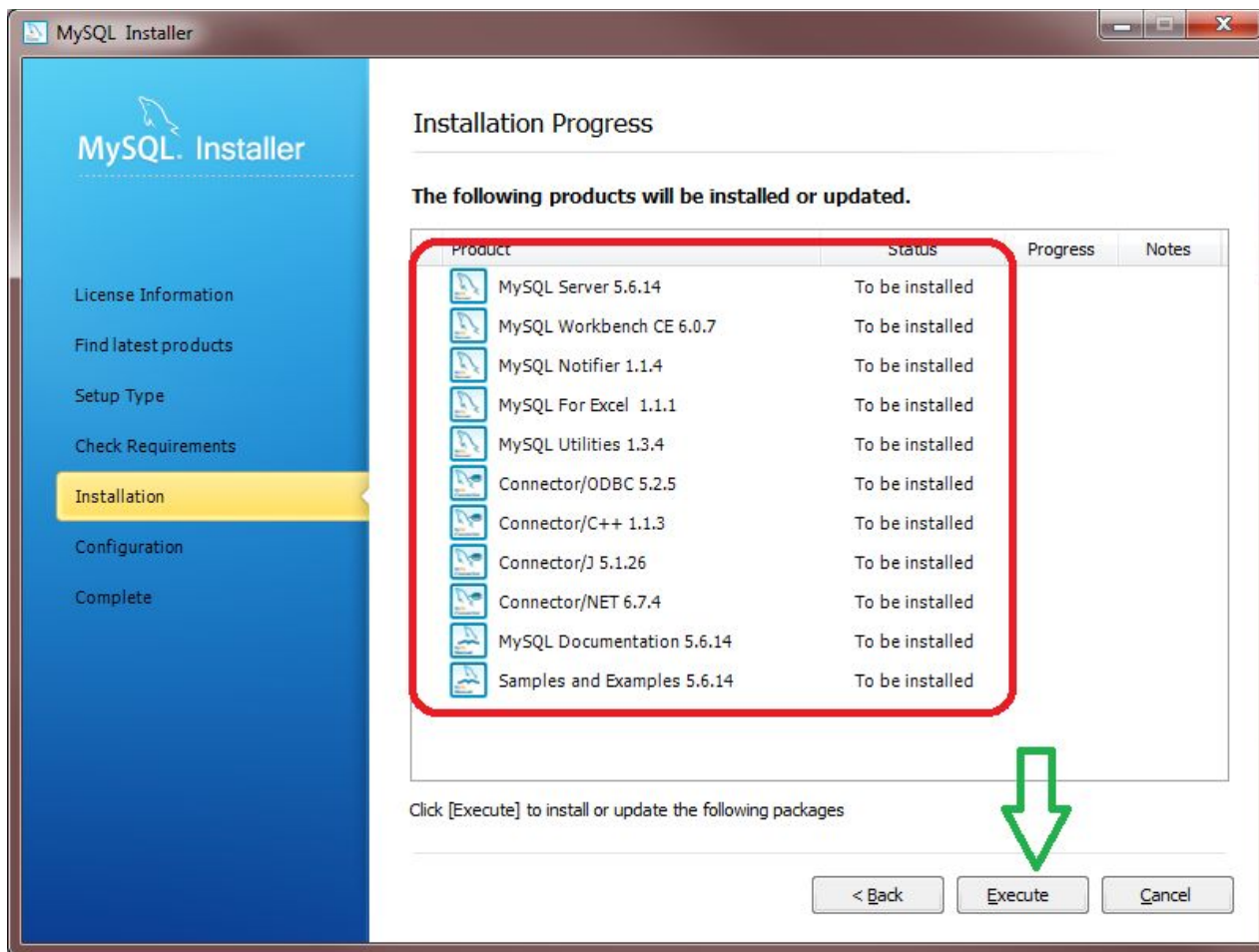
## Шаг-7





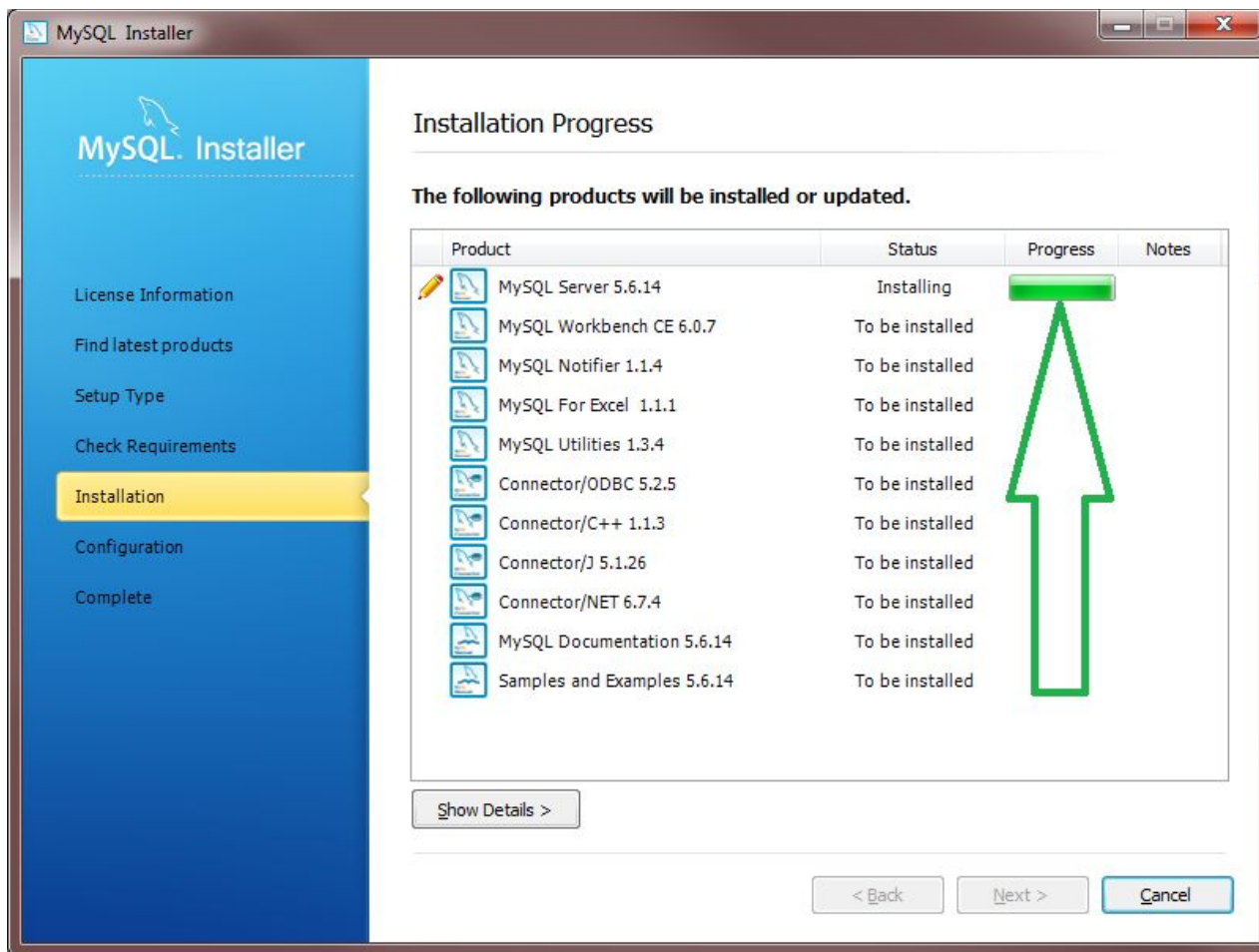
# My SQL – установка

## Шаг-8



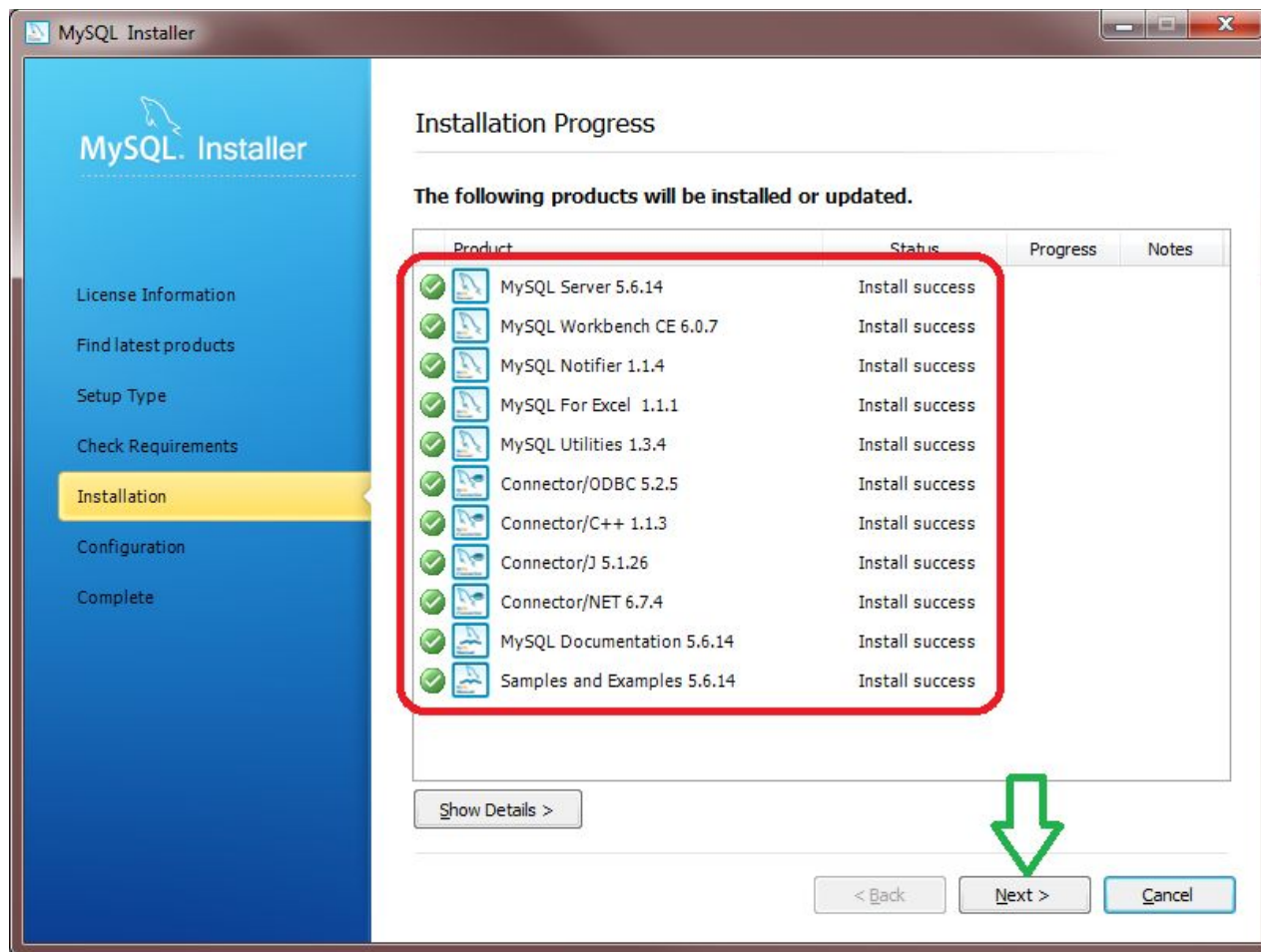
# My SQL – установка

## Шаг-9



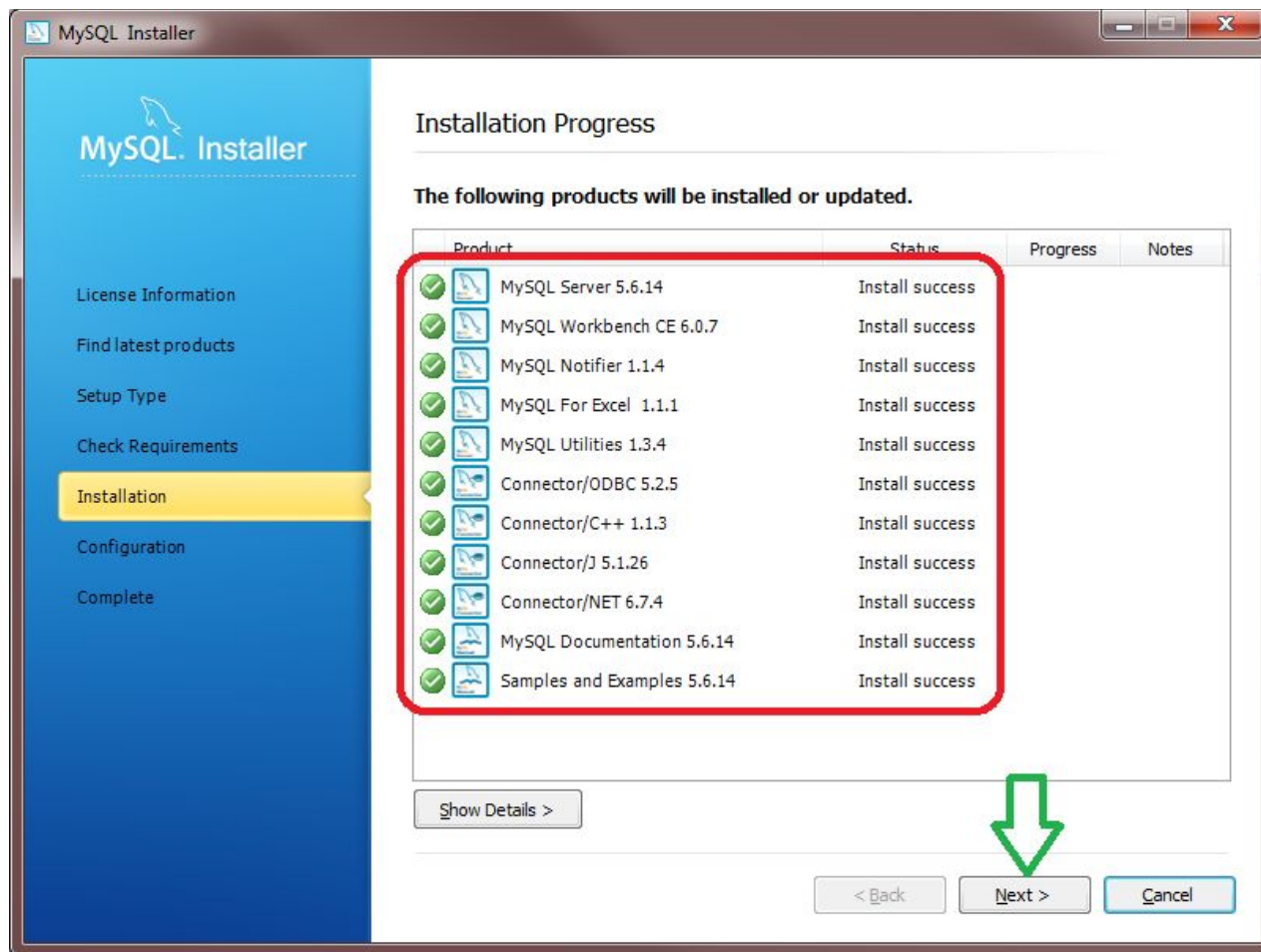
# My SQL – установка

## Шаг-10



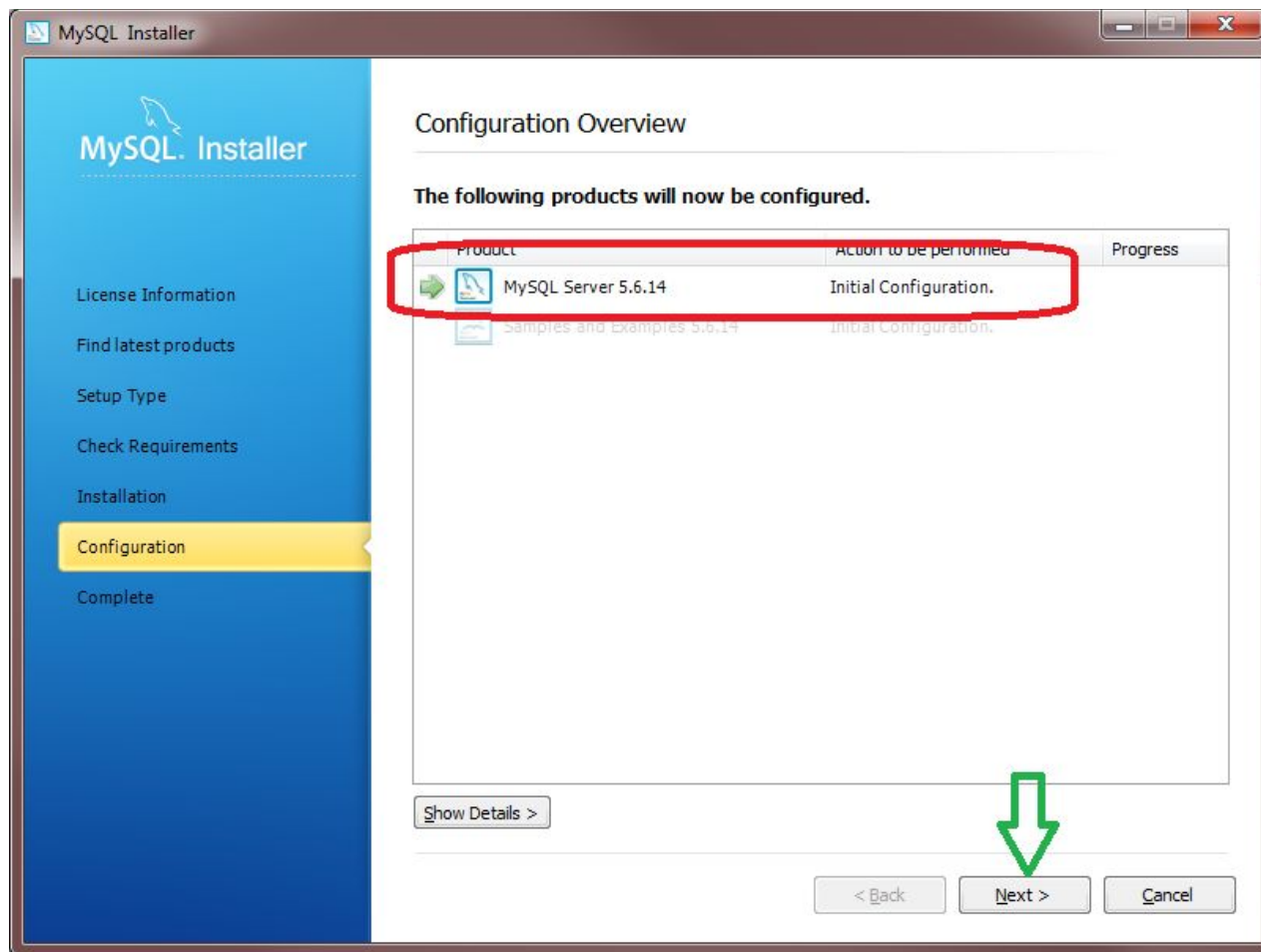
# My SQL – установка

## Шаг-10



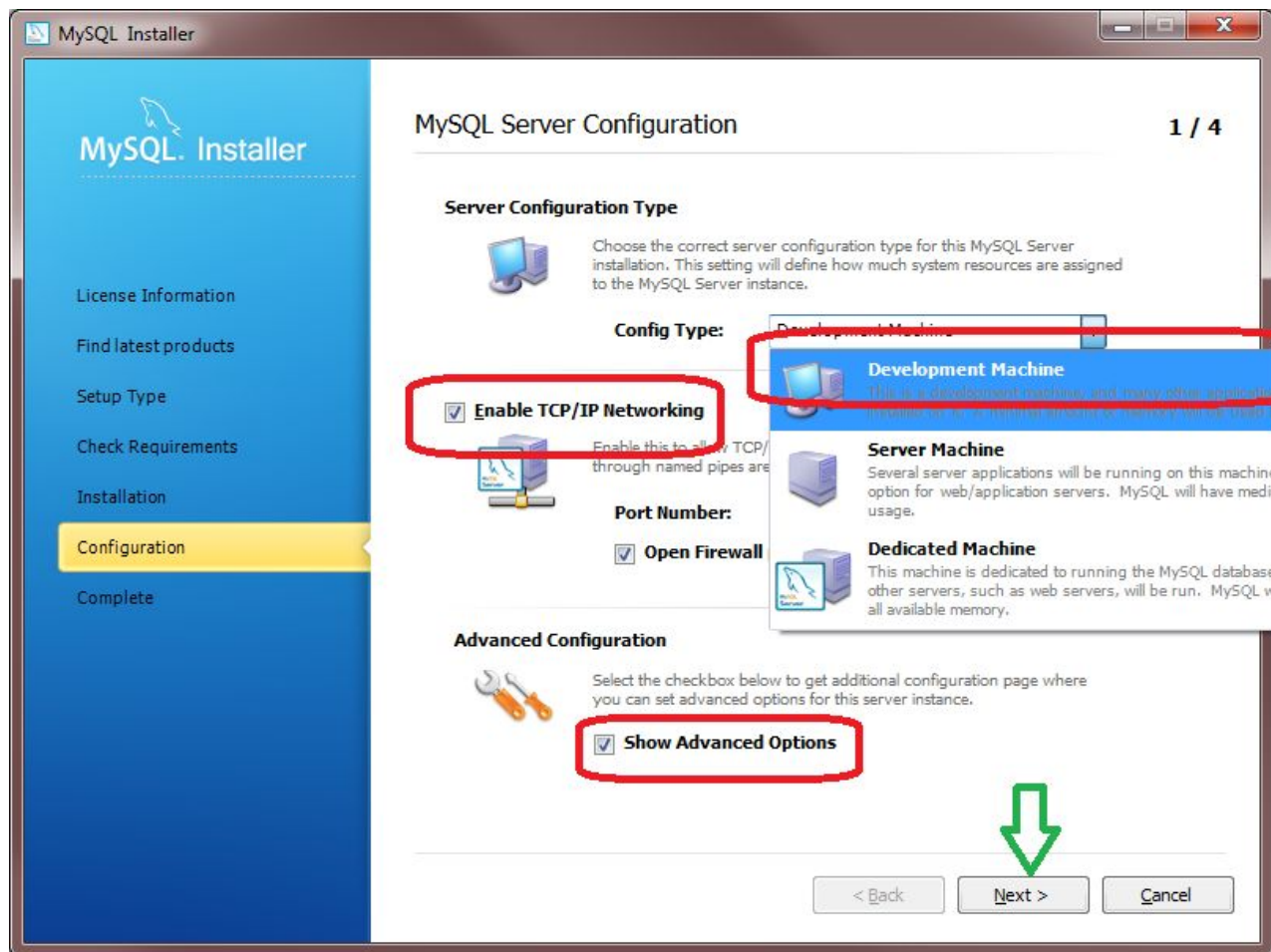
# My SQL – установка

## Шаг-11



# My SQL – установка

## Шаг-12



# My SQL – установка

## Шаг-13

MySQL Installer

MySQL Server Configuration 2 / 4

**Root Account Password**

Enter the password for the root account. Please remember to store the password in a secure place.

**MySQL Root Password:** ....

**Repeat Password:** ....

Password Strength: **Weak**

**MySQL User Accounts**

Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL Username	Host	User Role
----------------	------	-----------

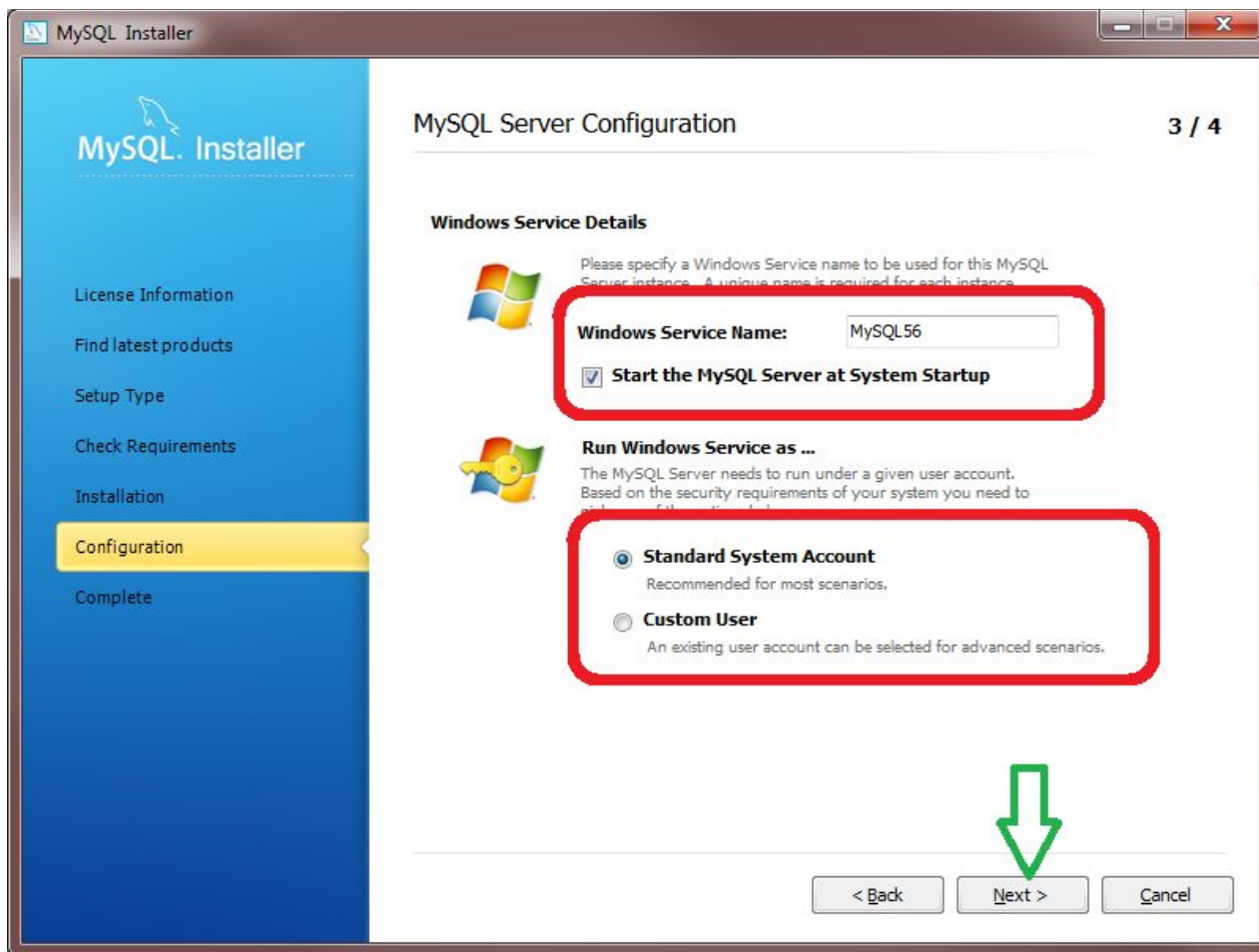
[Add User](#)  
[Edit User](#)  
[Delete User](#)

[< Back](#) [Next >](#) [Cancel](#)



# My SQL – установка

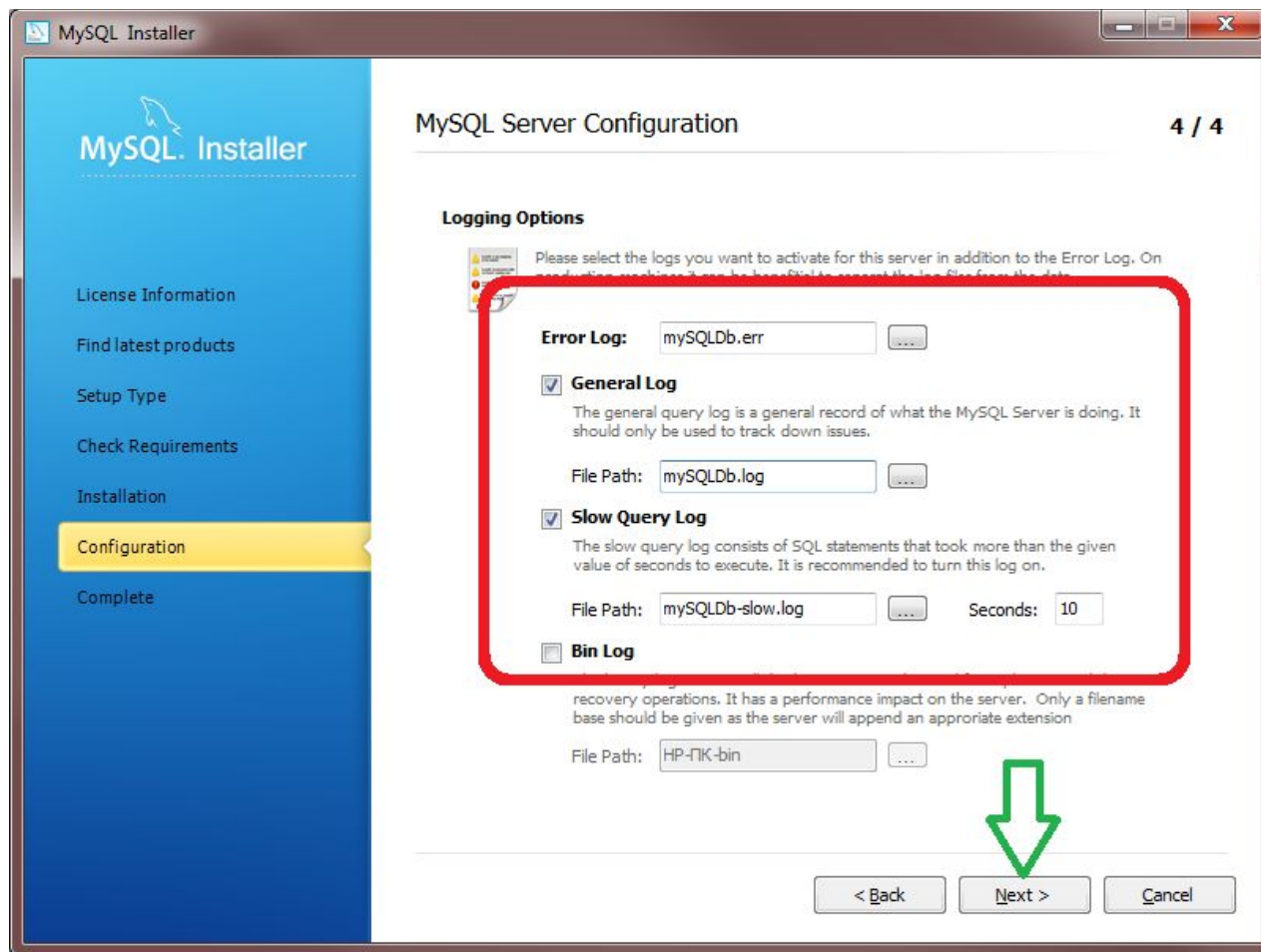
## Шаг-14





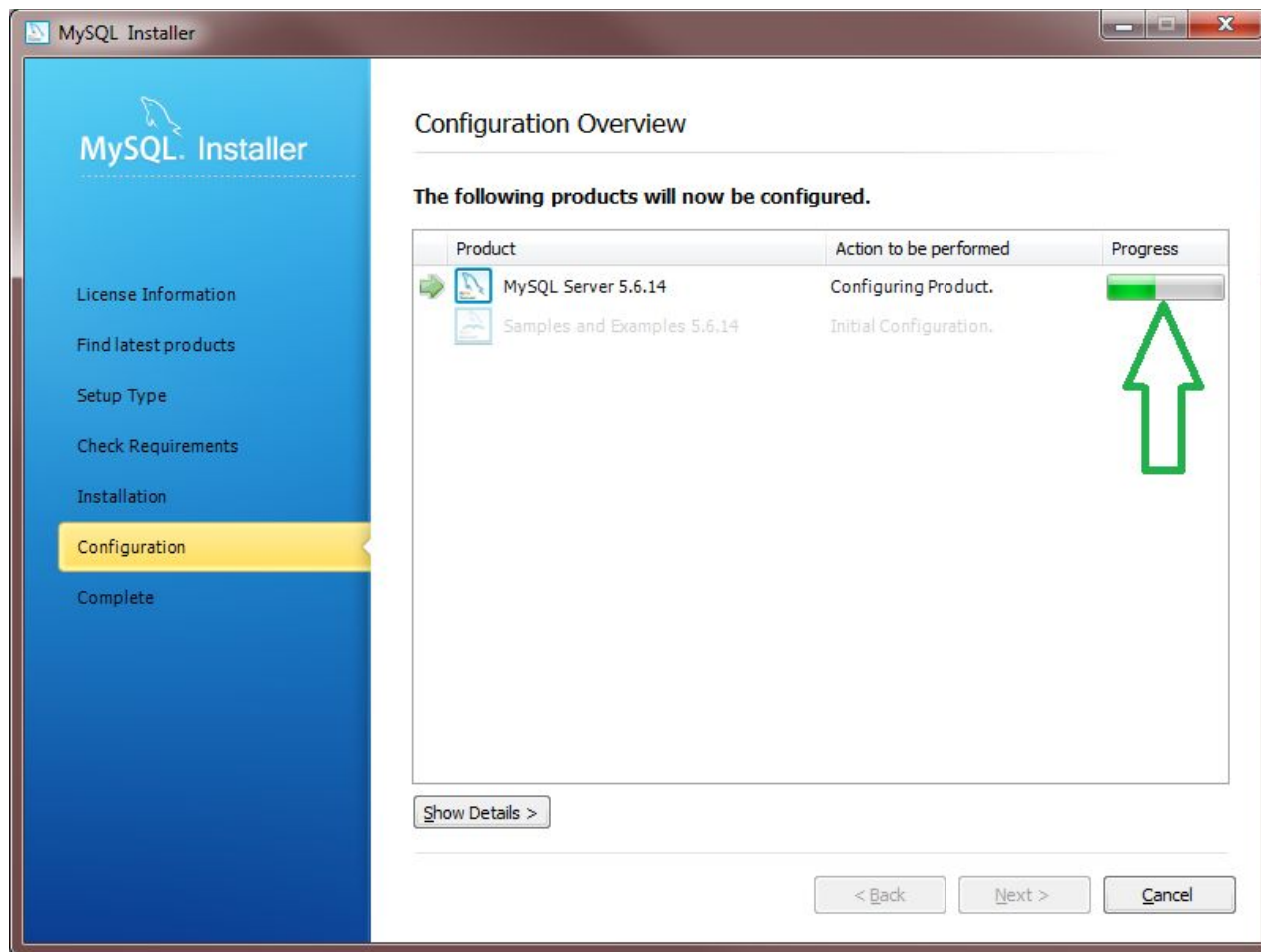
# My SQL – установка

## Шаг-15



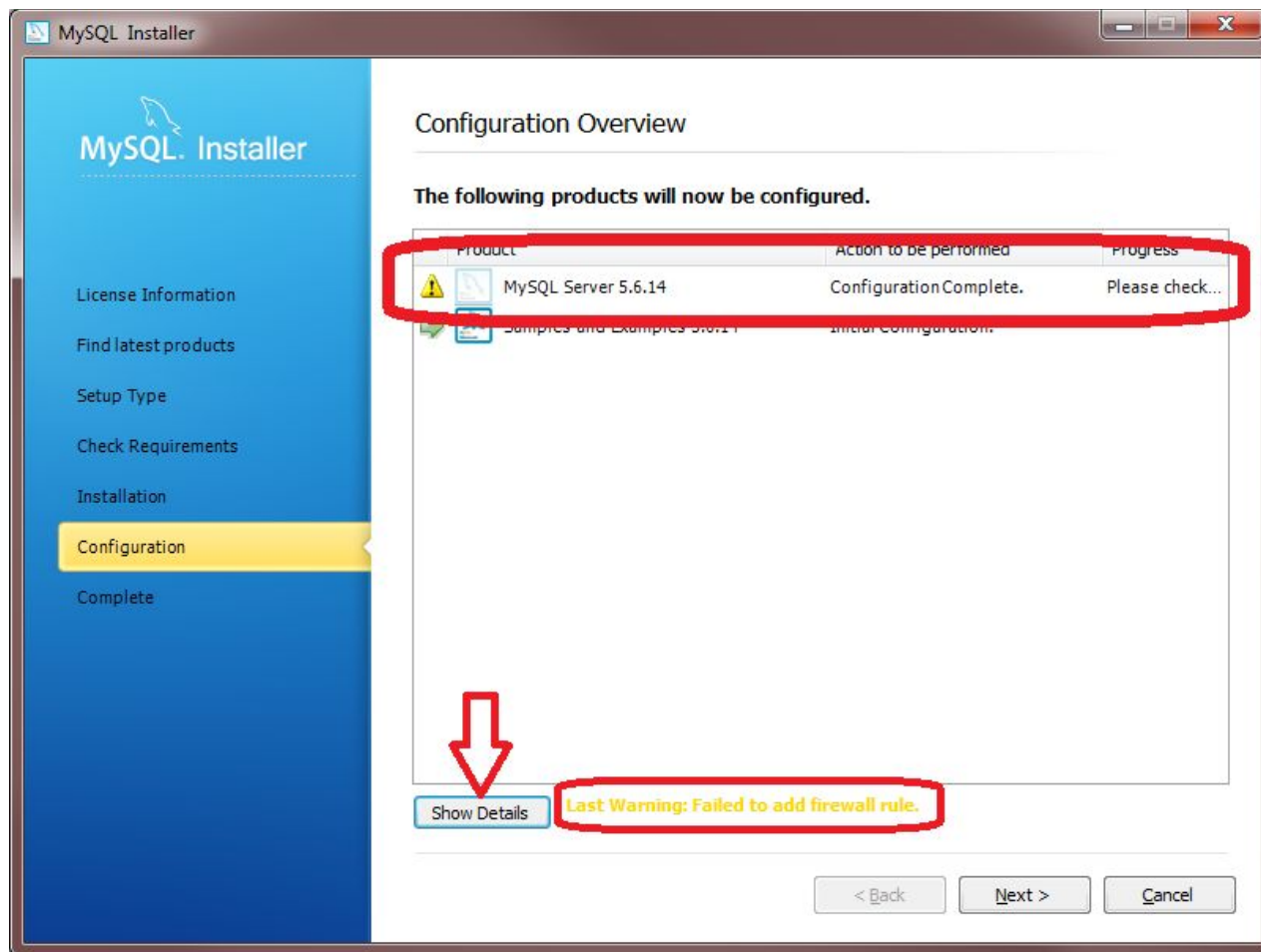
# My SQL – установка

## Шаг-16



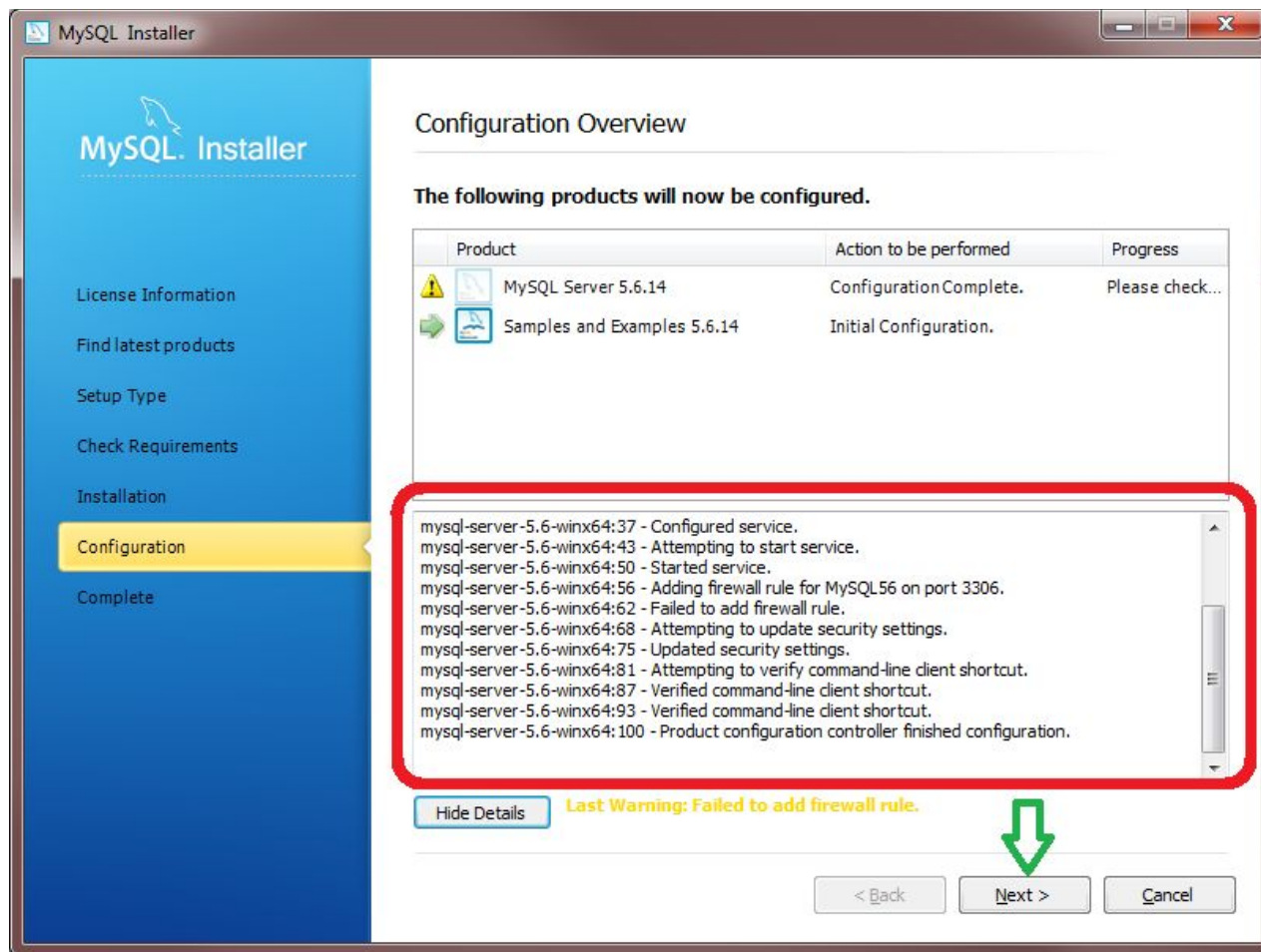
# My SQL – установка

## Шаг-17



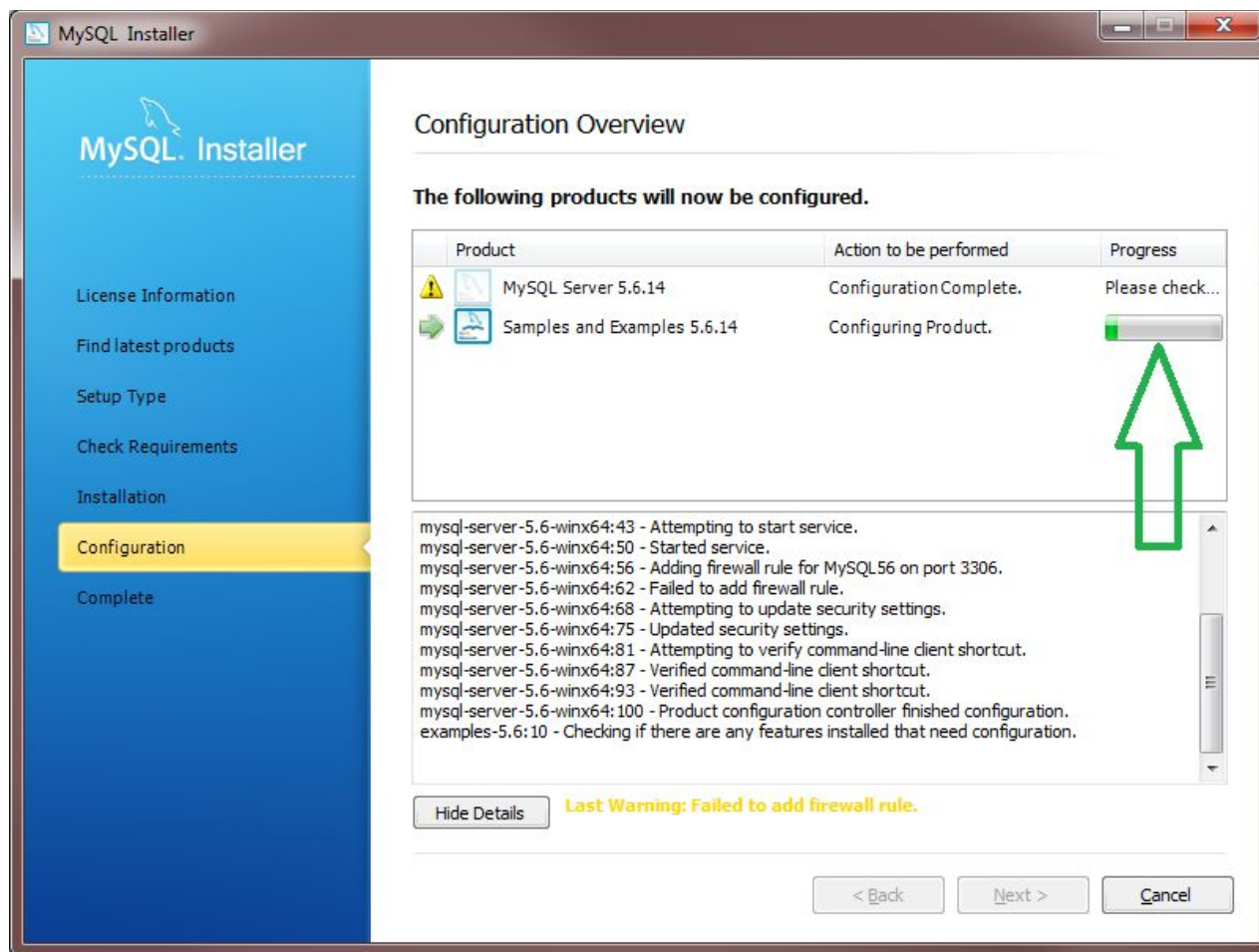
# My SQL – установка

## Шаг-18



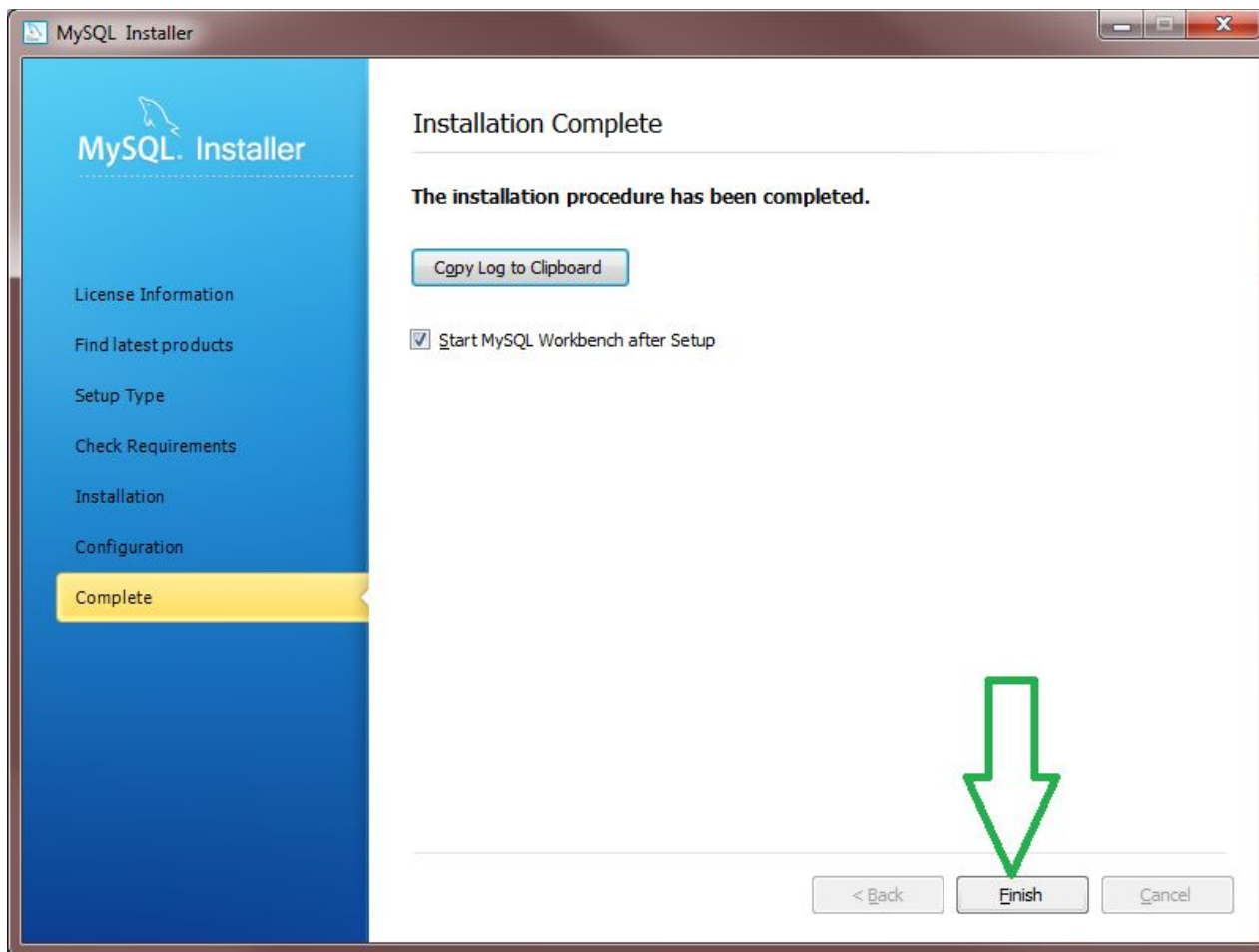
# My SQL – установка

## Шаг-19



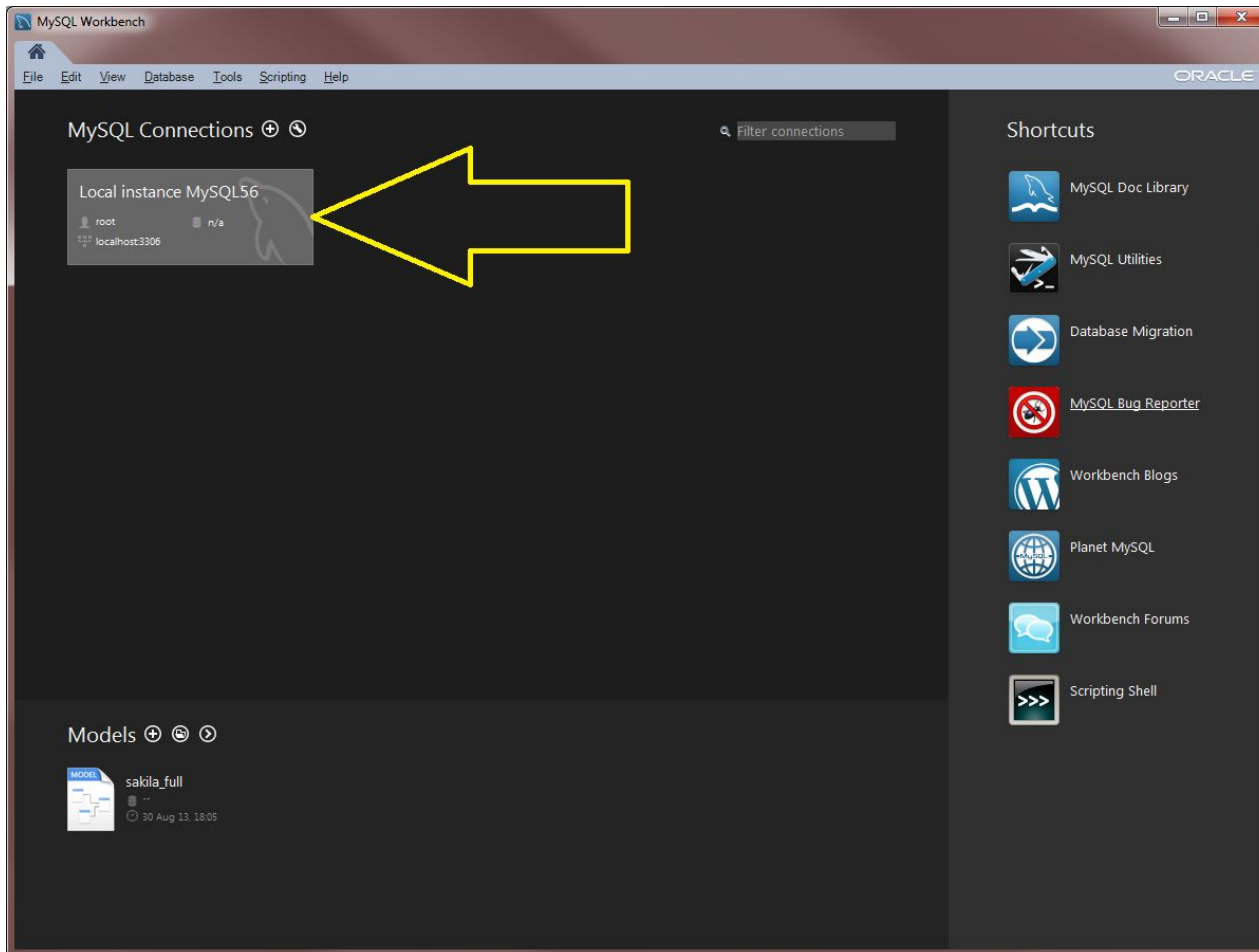
# My SQL – установка

## Шаг-20



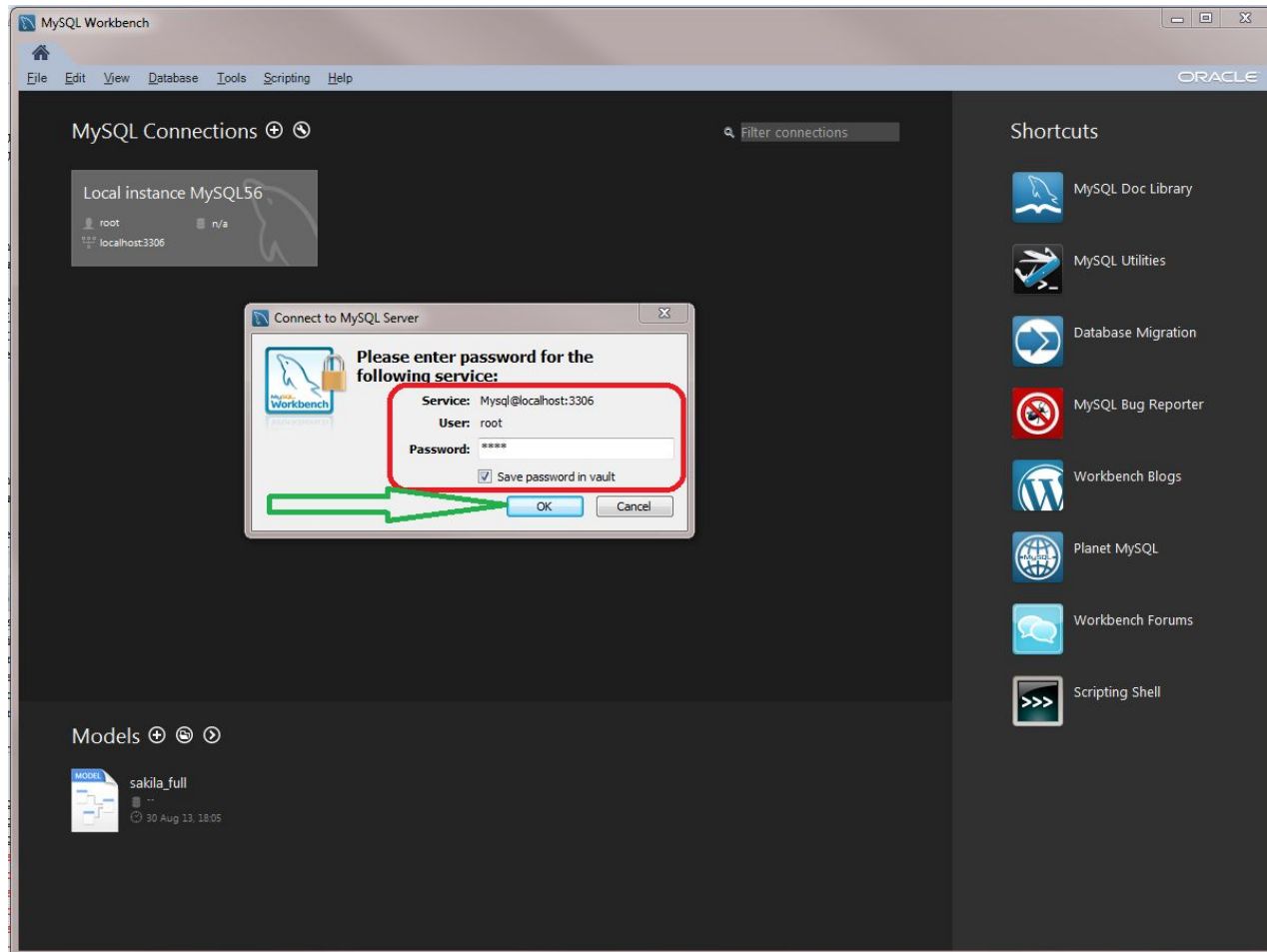
# MySQL Workbench

## Стартовая страница.



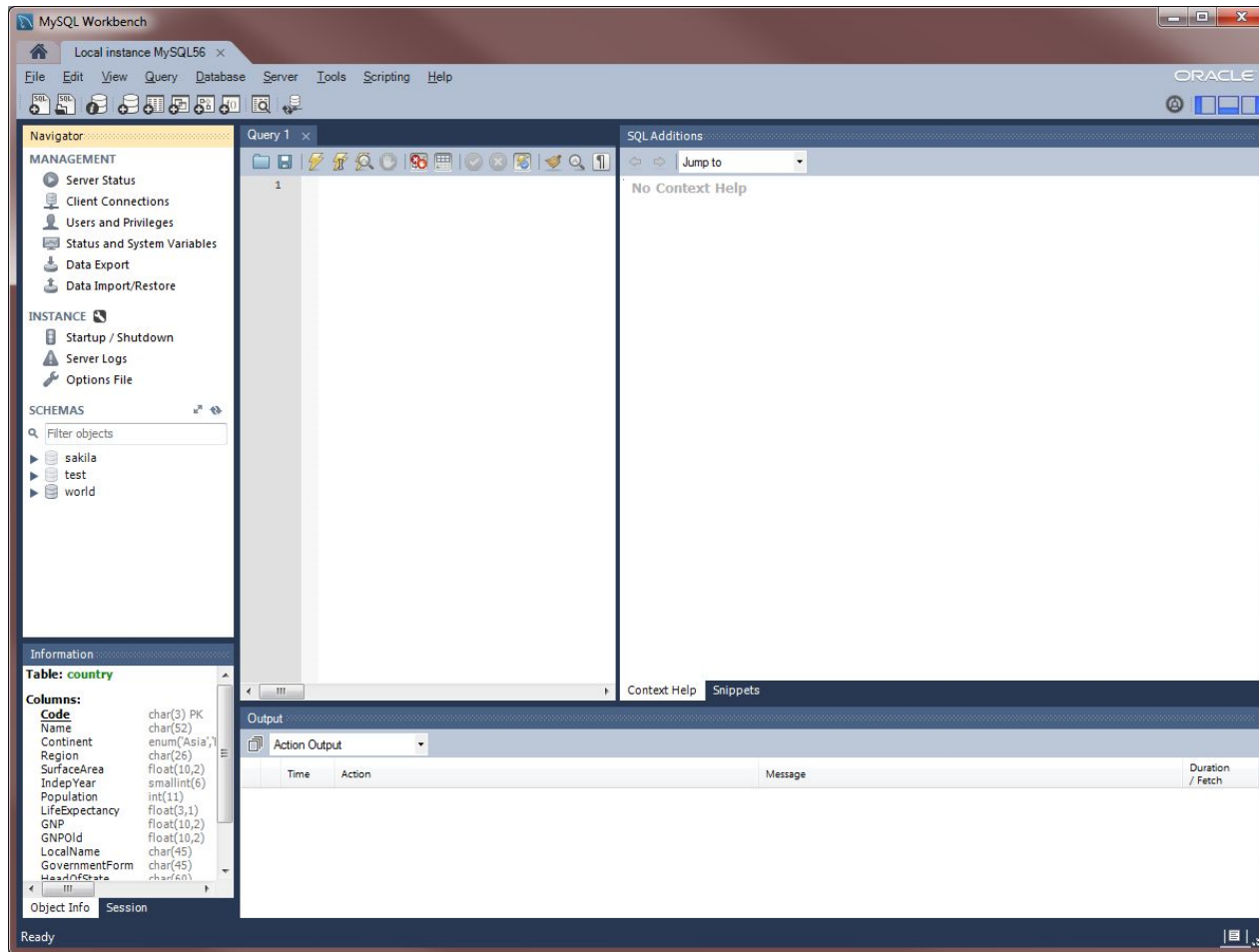
# MySQL Workbench

## Авторизация.





# MySQL Workbench Рабочее окно.



# Итак, что нужно установить?

- Основные программы  
(<https://dev.mysql.com/downloads/mysql>)
  - MySQL Server
  - MySQL Workbench
  - MySQL Connector/J
- Дополнительные программы:
  - DbVisualizer (<https://www.dbvis.com>)

# Предпосылки проблемы

- Любое приложение так или иначе будет обслуживать данные (информацию), которые необходимы пользователям приложения.
- Приложение не работающее с данными – бесполезно, а следовательно – не востребовано!

# Предпосылки проблемы

- Под «обслуживанием» можно понимать:
  - Сохранение данных в персистентном (постоянном) хранилище.
  - Извлечение данных из хранилища.
  - Обновление данных в хранилище.
  - Организация многопользовательского (многопоточного) доступа к данным.
  - Защита данных от несанкционированного доступа.
  - «Слежение» за связностью и корректностью данных.

# Предпосылки проблемы

- Следовательно, для любого приложения программист создаст:
  - Алгоритмы для бизнес процессов приложения.
  - Модель, хранящую все данные приложения.
  - Алгоритмы для обслуживания модели данных приложения

# Проблема

## С точки зрения человека.

- Алгоритмы для обслуживания данных всегда будут **сложнее** алгоритмов бизнес логики.
- Они всегда потребуют **больше времени** и «человеческого ресурса».
- В силу сложности они всегда будут содержать большое количество **ошибок (багов)**.
- Большое количество времени придется потратить на вопросы **производительности, защиты** и организации **многопоточного доступа** к данным.
- Разработчики не смогут сосредоточиться на исследовании бизнес процессов системы, так как фактически, им придется развивать **ДВА приложения**.
- И самое главное – с каждым новым приложением

**Этот «АД» будет повторяться снова и снова!**

# Проблема

## С точки зрения машины.

- Хранилище может предоставить данные, но оно не всегда предоставляет **описание хранимых данных** (мета данные):
  - Типы (имена) хранимых объектов.
  - Все свойства каждого хранимого объекта.
  - Типы данных для этих свойств.
  - Связь объектов друг с другом и т. д.
- Обмен данными между системами **затруднен**, так как:
  - Они работают с разными хранилищами.
  - Каждое хранилище имеет свою структуру:
    - Текстовые файлы
    - Бинарные файлы
    - XML и т. д.
  - Каждое хранилище работает на основании «своих» алгоритмов.
- Следовательно, для организации «межсистемного» взаимодействия придется писать **«Систему по стыковке систем»**

# Возможное решение проблемы

- С каждым новым приложением программист работает над алгоритмами по обслуживанию данных.
- Со временем он замечает, что:
  - Функциональная часть алгоритмов практически одинакова.
  - Изменяется только модель данных, с которыми работает новое приложение.



# Возможное решение проблемы

- Таким образом, программист понимает, что хорошо бы иметь в своем арсенале:
  - **Теорию**, которая структурирует все фундаментальные вопросы по описанию данных и организации доступа к ним.
  - **Программное обеспечение** (набор универсальных алгоритмов), которое позволит:
    - Описать (создать) модель данных.
    - Сохранять и модифицировать данные в модели.
    - Извлекать данные для нужд приложения и пользователей.
  - **Стандарт**, который позволит описывать, извлекать и модифицировать данные независимо от того, каким программным обеспечением они обслуживаются.

# И наконец – **решение**

- **База Данных:**

- Теория, фундаментальные вопросы по описанию данных и доступу к ним.
- Модель данных, описанная для конкретного приложения.

- **Система Управления Базами Данных:**

- Универсальное программное обеспечение для создания баз данных и управления ими.

- **SQL**

- Универсальный язык для описания, модификации и доступа к данным.

# Термины, сокращения, переводы

## DB

DB

Data Base

БД

База Данных

## BDMS

DBMS

Data Base Management System

СУБД

Система Управления Базами  
Данных

## SQL

SQL

Structured Query Language

Структурированный язык запросов

# Таблица в БД

- **Таблица** – Основной элемент базы данных.
- **Одна таблица** хранит данные **всех** сущностей (объектов) **одного** типа.
- Таблица должна описать **все** свойства сущности, которые интересны приложению (пользователю).
- Физически данные в СУБД могут храниться как угодно, но пользователи БД «видят» их как двумерную таблицу, где:
  - Каждый столбец хранит **одно** свойство **всех** сущностей таблицы.
  - Каждая строка представляет **все** свойства **одной** сущности в таблице.
- Строку в таблице часто называют **записью в БД**.

# Таблица в БД

ACCOUNTS				
ID	First Name	Last Name	Login	Password
1	Ivan	Ivanov	Ivanov@gmail.com	IvanovSecret
2	Petr	Petrov	Petrov@gmail.com	PetrovSecret
3	Sidor	Sidorov	Sidorov@gmail.com	SidorovSecret

Строка - **ВСЕ** свойства  
**ОДНОЙ** сущности

Столбец - **ОДНО** свойство **ВСЕХ** сущностей.

# Столбец в таблице

- **Один** столбец описывает **одно** свойство сущности.
- Каждый столбец должен иметь **ИМЯ**.
- Имя столбца в таблице должно быть **уникально**.
- Каждый столбец должен иметь **тип данных**.
- В столбец можно сохранить только те данные, которые соответствуют типу столбца.
- Любой столбец может обладать дополнительными свойствами, такими как:
  - Ненулевое значение (NOT NULL)
  - Значение по умолчанию (DEFAULT)
  - Максимальная длина (для строк) и т. д.

# Типы данных в MySQL

- Numeric Types (Числа)
- Date and Time Types (Дата и время)
- String Types (Строки)
- BLOB – Binary Large Object
- TEXT

**Обязательно прочитать эту главу:**

<https://dev.mysql.com/doc/refman/8.0/en/data-types.html>

\*Внимание, главу про типы надо читать под той версией MySQL, с которой вы собираетесь работать.

# Служебные столбцы

- Любая таблица может иметь столбцы, которые **«не интересны»** пользователю, но необходимы для функционирования БД.
- Такие столбцы можно назвать служебными
- Наиболее распространенные примеры:
  - Первичные ключи (Primary Keys)
  - Внешние ключи (Foreign Keys)



# Первичный ключ

## Primary Key

- Одна из главных задач СУБД – обеспечить уникальность записей в таблице.
- За уникальность данных отвечает механизм **первичного ключа**.
- Чаще всего первичный ключ – это:
  - Отдельный столбец в таблице.
  - Целочисленного типа данных.
  - С алгоритмом генерации значений – AUTO\_INCREMENT.
- Однако первичным ключом может быть любой столбец таблицы, который хранит **уникальные значения**.
- Также первичным ключом может быть совокупность столбцов.
- В этом случае говорят, что таблица использует **составной первичный ключ** или **Composite Primary Key**.
- Использование составных ключей **усложнит** программирование приложения.

# Виды первичных ключей

- **AUTO\_INCREMENT**
  - наиболее популярный
- **Natural Primary Key**
  - натуральный первичный ключ.
  - любой столбец, с уникальными значениями.
  - Например, код валюты или номер паспорта
- **Composite Primary Key**
  - составной первичный ключ.
  - Самый популярный пример: совокупность ключей в промежуточной таблице many-to-many

# Внешний ключ

## Foreign Key

- **Внешний ключ** – это специальный столбец, который хранит значения первичных ключей из таблицы, по отношению к которой текущая таблица **является подчиненной**.
- Внешний ключ **не обязательно** должен присутствовать в таблице.
- Значения внешнего ключа могут быть **не уникальными**.
- Значением внешнего ключа может быть **NULL**.
- Тип данных внешнего ключа **должен совпадать** с типом данных первичного ключа главной таблицы.

# Отношения таблиц

- Рассматривать внешний ключ тяжело без понятия **«отношения между таблицами»**
- Давайте по рассуждаем:
  - Любое приложение работает с объектами различных типов
  - например:
    - Преподаватели
    - Студенты
    - Учебные курсы
    - Учебные группы и т. д.
  - Каждому объекту соответствует своя таблица.

# Отношения таблиц

- Но приложению «не интересно», знать данные только из одной таблицы
- Например, «не интересно» знать, какие преподаватели есть в учебном центре.
- Приложение должно ответить на следующие вопросы:
  - Какие преподаватели есть в учебном центре.
  - Какие учебные курсы создал каждый преподаватель.
  - Какие группы сформированы по каждому курсу.
  - Сколько студентов занимается в каждой группе
  - Какие лекции, задания и тесты относятся к каждому курсу
  - И т. д.

# Таким образом

- Кроме самих таблиц, в БД большое значение имеют

## Отношения между таблицами

- Все таблицы находятся в отношении друг с другом.
- Каждая таблица БД может быть:
  - **главной таблицей** — от нее зависят данные в других таблицах.
  - **подчиненной таблицей** — данные этой таблицы зависят от главной таблицы.
- Таблица является подчиненной, если

**В ней есть внешний ключ!**

# Как создать внешний ключ?

```
CREATE TABLE `courses` (  
  `id` INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,  
  `course_name` VARCHAR(555) NOT NULL,  
  `fk_category_id` INTEGER default NULL,  
  
  CONSTRAINT `fk_course_to_category` FOREIGN KEY (`fk_category_id`) REFERENCES `categories` (`id`)  
    ON DELETE RESTRICT  
    ON UPDATE RESTRICT  
);
```

# Какие бывают отношения?

- **One-To-Many** – Один ко многим
- **Many-To-One** – Многие к одному
- **Many-To-Many** – Многие ко многим
- **One-To-One** – Один к одному



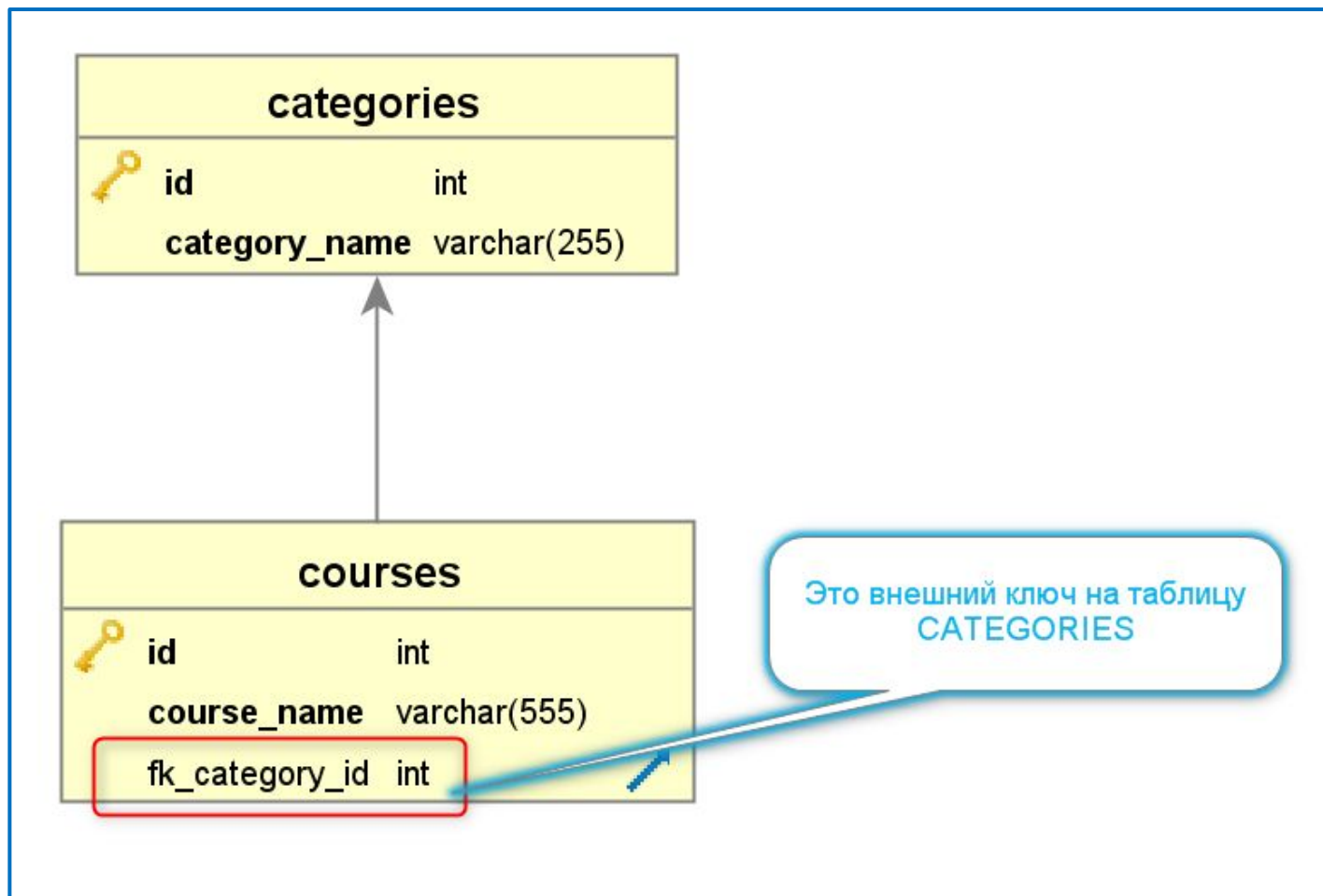
# One-To-Many

- Самое популярное отношение
- Означает, что

**Одна сущность главной таблицы может владеть множеством сущностей из подчиненной таблицы**

- Чтобы создать отношение One-To-Many  
**Нужно добавить Foreign Key в подчиненную таблицу**

# One-To-Many - пример



# Many-To-One

- Это такая же связь, как и One-To-Many
- Реализуется она точно так же:  
**Добавлением внешнего ключа**
- Many-To-One отличается не реализацией, а  
**Способом рассмотрения!**
- **One-To-Many**
  - когда мы смотрим от главной таблице к подчиненной
- **Many-To-One**
  - когда мы смотрим от подчиненной таблицы к главной

# Many-To-Many

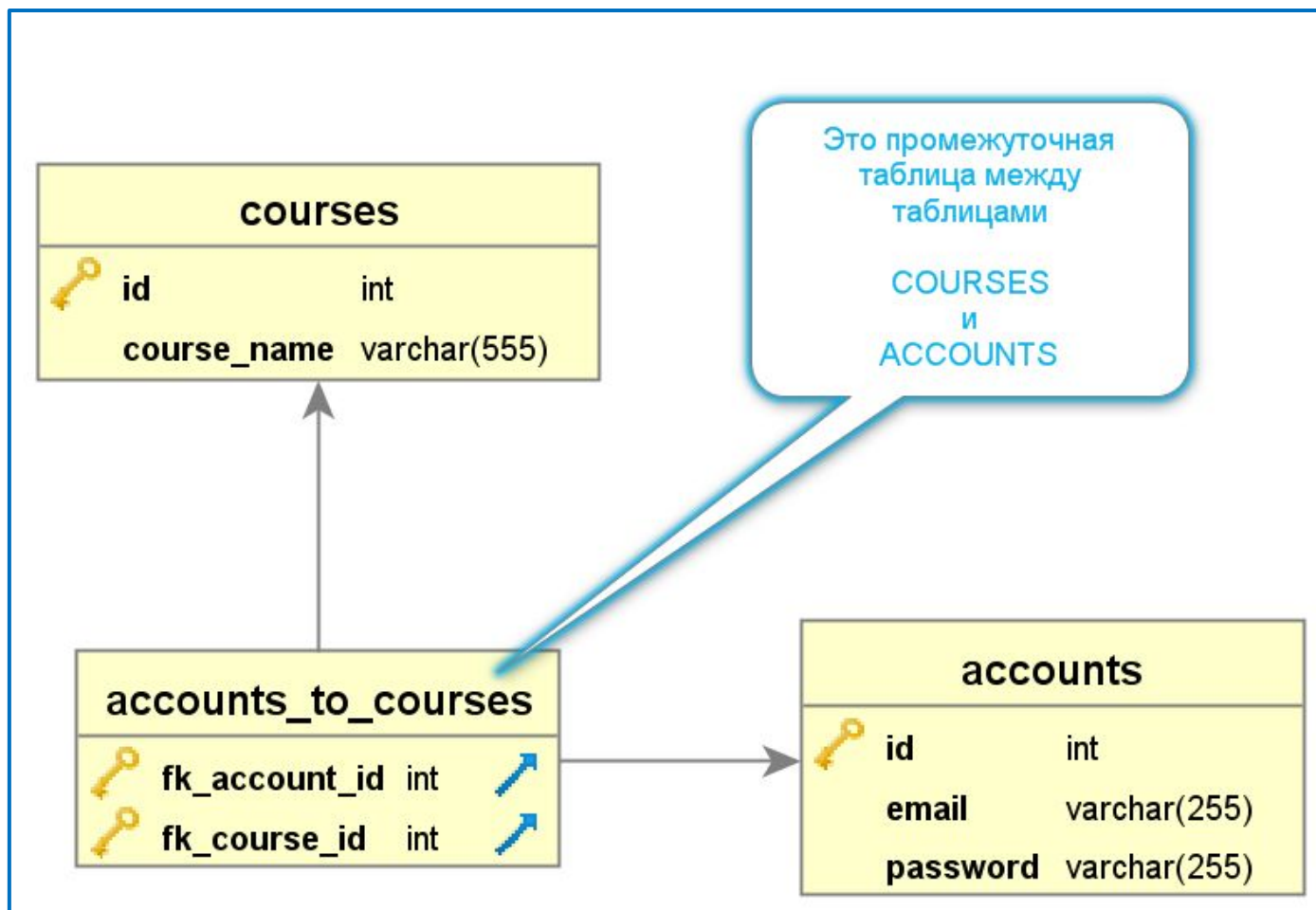
- Означает, что:

**Одна сущность главной таблицы может владеть множеством сущностей из подчиненной таблицы, но и каждая сущность подчиненной таблицы может владеть множеством сущностей из главной таблицы**

- Получается, что в Many-To-Many нет четко подчиненных таблиц
- Чтобы создать отношение Many-To-Many

**Нужно добавить промежуточную таблицу с двумя Foreign Keys на каждую из главных таблиц!**

# Many-To-Many - пример



# One-To-One

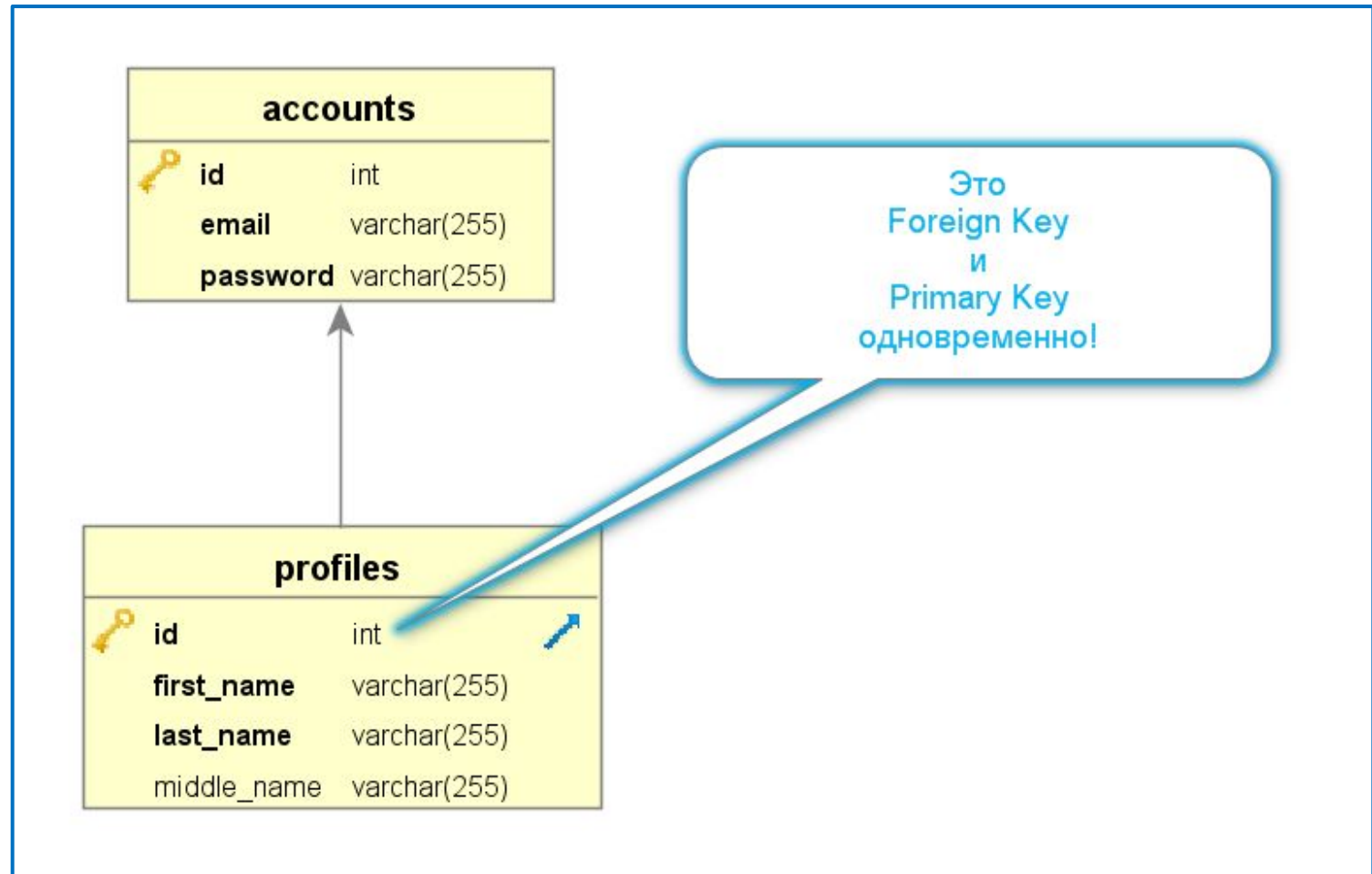
- Довольно редкая связь
- Означает, что:

**Одна сущность из главной таблицей может владеть только одной сущностью из подчиненной таблицы**

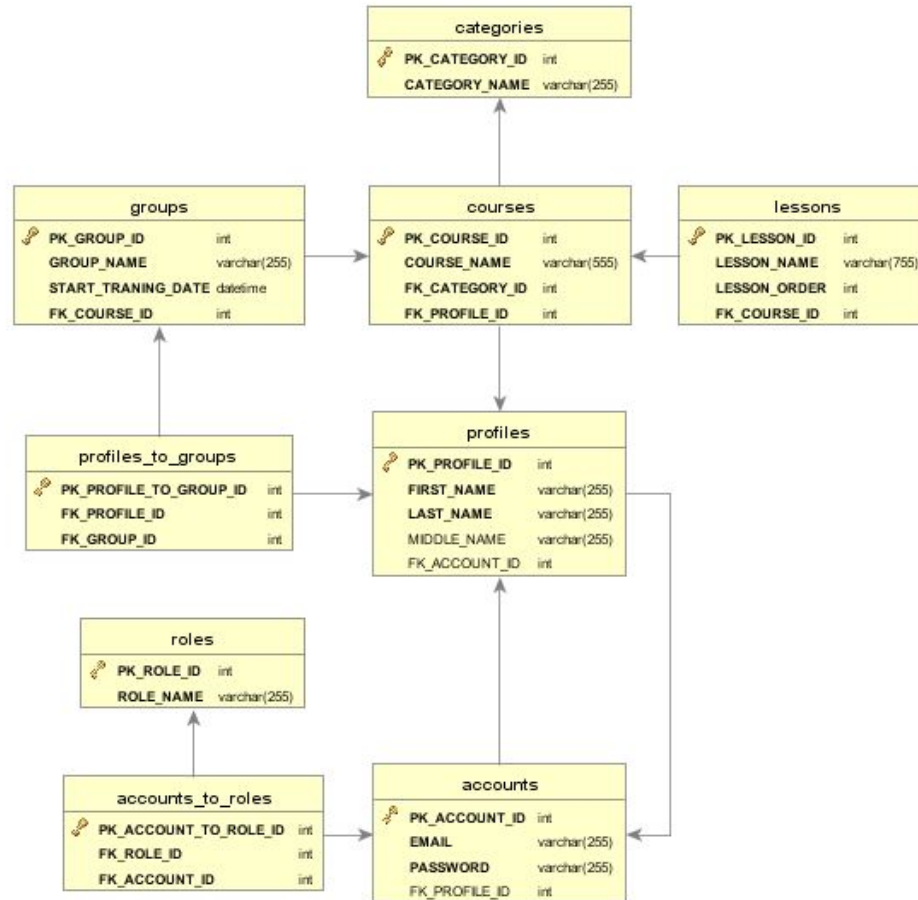
- Чтобы создать отношение One-To-One:

**Нужно добавить Foreign Key в подчиненную таблицу, но этот Foreign Key должен быть одновременно и Primary Key в подчиненной таблице!**

# One-To-One - пример



# BD SCHEMA

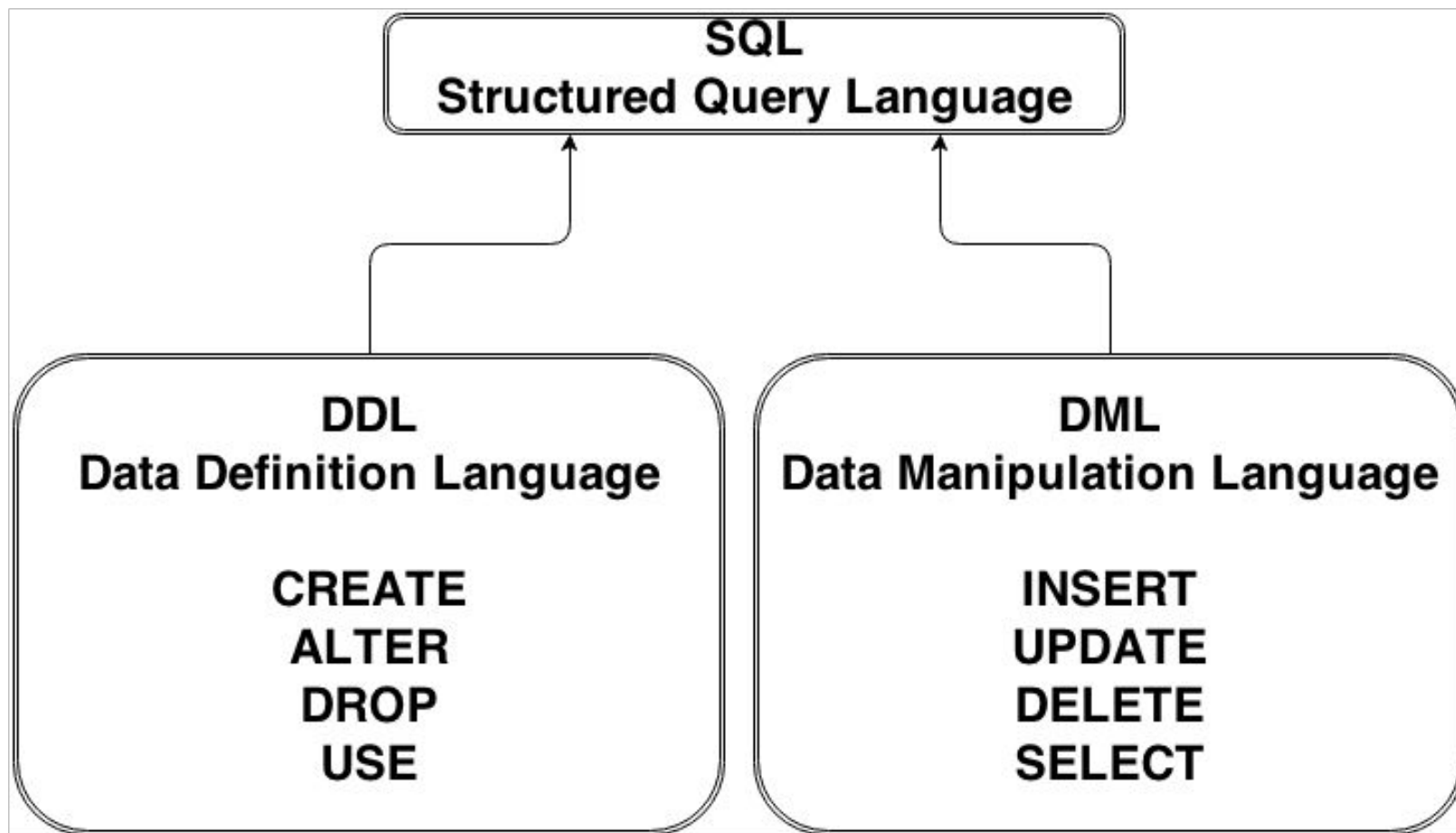




**Что насупился весь  
И сидишь как буржуй?  
Раз возникла проблема  
Ты сопли не  
жуй!!!  
Не станет рабочий  
над траблой стонать!  
Бери **SQL** – и айда  
выгребать!!!**

М. Лазаревич  
Из раннего...

# SQL – Условное деление



# Команда **CREATE**

- При помощи команды **CREATE** можно создать следующие элементы Базы Данных:

- Базу данных (Схему БД)

- Таблицу в БД

- Индекс в таблице

# CREATE для Базы Данных

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] `db_name`  
[CHARACTER SET charset_name]  
[COLLATE collation_name]
```

## Пример:

```
CREATE SCHEMA IF NOT EXISTS `TRAINING_DB`  
CHARACTER SET `utf8`;
```

# CREATE для Базы Данных.

## Пояснения.

- **Character set** – кодировка, в которой СУБД будет представлять символы, хранящиеся в таблицах БД.
- **Collation** – набор правил (алгоритмы), по которым СУБД будет сравнивать символы определенной кодировки – например – при сортировке записей.
- Каждая кодировка имеет некий Collation **по умолчанию**.
- **Желательно** использовать именно Collation по умолчанию.
- Следовательно, при создании БД, кодировку **лучше указать**, а Collation – **нет**.
- Информацию по кодировкам и Collation **лучше всего** брать из **официальной документации** к СУБД, и стоит учитывать **версию** СУБД!

# CREATE для Базы Данных.

## Пояснения.

- Каждая СУБД имеет некоторую кодировку **по умолчанию**.
- Она указывается при **установке и настройке** сервера СУБД.
- Значит при описании своей БД кодировку можно не указывать?
- Нет – лучше **указать**!
- Ведь СУБД по умолчанию может работать с кодировкой, которая Вашему приложению **не подходит**.
- Лучшая кодировка для Java программиста – **UTF-8**.
- В этом случае (Для СУБД MySQL):
  - CHARACTER SET: **utf8**
  - COLLATE: **utf8\_general\_ci**
- А что если СУБД не поддерживает UTF-8? Ответ – далее!

# СУБД не поддерживает UTF-8. Логическая задача.

**Условие:** Java-программист работает под СУБД, которая не поддерживает кодировку UTF-8.

**Задание:** Основываясь на исходных данных, а так же на времени, за которое программист произнесет фразу «Ну... Эта...

Типа... Таво.... Ну....» (а эта фраза – 90 процентов его лексикона, плюс маты), определите сколько раз (в среднем в день) этому Java-программисту прилетало табуретом промеж глаз в детстве?

**Альтернативное задание:** Определите, где в помещении прячутся Java-террористы, которые пытаются Java-программиста?

**Альтернативное задание-2:** Определите, сколько часов тому назад

Java-программисту вырезали аппендицит, и когда его «отпустит» наркоз?

# CREATE для таблицы

```
CREATE TABLE [IF NOT EXISTS] `db_name`.`tbl_name`  
(  
    create_definition  
);
```

- Здесь **create\_definition** – это блок SQL команд, которые:
  - Опишут все столбцы таблицы.
  - Укажут все Primary Keys для таблицы.
  - Опишут все Foreign Keys для таблицы.
  - И т. д. (индексы, уникальные поля, ...)
- **create\_definition** в рамках курса подробно не рассматривается.



# CREATE для таблицы.

## Пример.

```
CREATE TABLE `TRAINING_DB`.`STUDENTS` (  
  `PK_STUDENT_ID` INTEGER NOT NULL AUTO_INCREMENT,  
  `FIRST_NAME` VARCHAR(255) NOT NULL,  
  `LAST_NAME` VARCHAR(255) NOT NULL,  
  `MIDDLE_NAME` VARCHAR(255) NOT NULL,  
  `FK_GROUP_ID` INTEGER NOT NULL,  
  
  PRIMARY KEY (`PK_STUDENT_ID`),  
  KEY `FK_STUDENT_TO_GROUP` (`FK_GROUP_ID`),  
  
  CONSTRAINT `FK_STUDENT_TO_GROUP` FOREIGN KEY (`FK_GROUP_ID`)  
    REFERENCES `GROUPS` (`PK_GROUP_ID`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE  
);
```

# CREATE для индекса

- **CREATE INDEX** index\_name **ON** tbl\_name

# Синтаксис команды **DROP**

- **DROP** {**DATABASE** | **SCHEMA**} [**IF EXISTS**] db\_name
- **DROP TABLE** [**IF EXISTS**] tbl\_name-1, tbl\_name-1, ...
- **DROP INDEX** index\_name **ON** tbl\_name

# SQL - DDL

```
1 DROP SCHEMA IF EXISTS `TRAINING_DB`;
2
3 CREATE SCHEMA IF NOT EXISTS `TRAINING_DB`;
4
5 USE `TRAINING_DB`;
6
7 CREATE TABLE `COURSES` (
8   `PK_COURSE_ID` INTEGER NOT NULL PRIMARY KEY,
9   `PK_COURSE_NAME` VARCHAR(255) NOT NULL
10 );
11
12 CREATE TABLE `GROUPS` (
13   `PK_GROUP_ID` INTEGER NOT NULL,
14   `GROUP_NAME` VARCHAR(255) NOT NULL,
15   `START_TRAINING_DATE` DATETIME NOT NULL,
16   `FK_COURSE_ID` INTEGER NOT NULL,
17
18   PRIMARY KEY (`PK_GROUP_ID`),
19
20   KEY `FK_GROUP_TO_COURSE` (`FK_COURSE_ID`),
21
22   CONSTRAINT `FK_GROUP_TO_COURSE` FOREIGN KEY (`FK_COURSE_ID`) REFERENCES `COURSES` (`PK_COURSE_ID`)
23     ON DELETE RESTRICT
24     ON UPDATE RESTRICT
25 );
26
27 CREATE TABLE `STUDENTS` (
28   `PK_STUDENT_ID` INTEGER NOT NULL,
29   `FIRST_NAME` VARCHAR(255) NOT NULL,
30   `LAST_NAME` VARCHAR(255) NOT NULL,
31   `MIDDLE_NAME` VARCHAR(255) NOT NULL,
32   `FK_GROUP_ID` INTEGER NOT NULL,
33
34   PRIMARY KEY (`PK_STUDENT_ID`),
35
36   KEY `FK_STUDENT_TO_GROUP` (`FK_GROUP_ID`),
37
38   CONSTRAINT `FK_STUDENT_TO_GROUP` FOREIGN KEY (`FK_GROUP_ID`) REFERENCES `GROUPS` (`PK_GROUP_ID`)
39     ON DELETE CASCADE
40     ON UPDATE CASCADE
41 );
```

# SQL - DML

- INSERT
- UPDATE
- DELETE
- SELECT

```
1 INSERT INTO `TRAINING_DB`.`COURSES` (`PK_COURSE_NAME`) VALUES ('Java - step-1 - Standard Edidtion');
2 INSERT INTO `TRAINING_DB`.`COURSES` (`PK_COURSE_NAME`) VALUES ('Java - step-1 - Enterprice Edidtion');
3 INSERT INTO `TRAINING_DB`.`COURSES` (`PK_COURSE_NAME`) VALUES ('C++');
4 INSERT INTO `TRAINING_DB`.`COURSES` (`PK_COURSE_NAME`) VALUES ('Testing');
```

# Синтаксис команды **INSERT**

```
INSERT INTO tbl_name (col-1, col-2, ...)  
VALUES (val-1, val-1, ...);
```

## Примеры:

- **INSERT INTO** ACCOUNTS (`login`, `password`) **VALUES** ('Admin', '123');
- **INSERT INTO** ACCOUNTS (`ID`, `login`, `password`) **VALUES** (null, 'Admin', '123');

**null** используется для Primary Key, если при описании таблицы использовался параметр **AUTO\_INCREMENT**

# Синтаксис команды **UPDATE**

**UPDATE** tbl\_name

**SET** col-1 = expr-1, col-2 = expr-2, ...

[**WHERE** where\_condition];

## Примеры:

- **UPDATE** ACCOUNTS **SET** `password`='123'
- **UPDATE** ACCOUNTS **SET** `password`='123'  
**WHERE** `login` = 'Admin'

# Синтаксис команды **DELETE**

**DELETE FROM** tbl\_name [**WHERE** where\_condition];

## Примеры:

- **DELETE FROM** ACCOUNTS;
- **DELETE FROM** ACCOUNTS **WHERE** `login`='Admin';



# Синтаксис команды **SELECT**

**SELECT** [**DISTINCT**] *select\_expr*, ...

**FROM** *table\_references*

[**WHERE** *where\_condition*]

[**ORDER BY** *col\_name* [**ASC** | **DESC**]]

[**LIMIT** *row\_count*]

# Пример **SELECT**

Выбрать все записи из ACCOUNTS

- **SELECT** `login`, `password` **FROM** ACCOUNTS
- **SELECT** \* **FROM** ACCOUNTS

Пример **SELECT**  
Выбрать всех админов

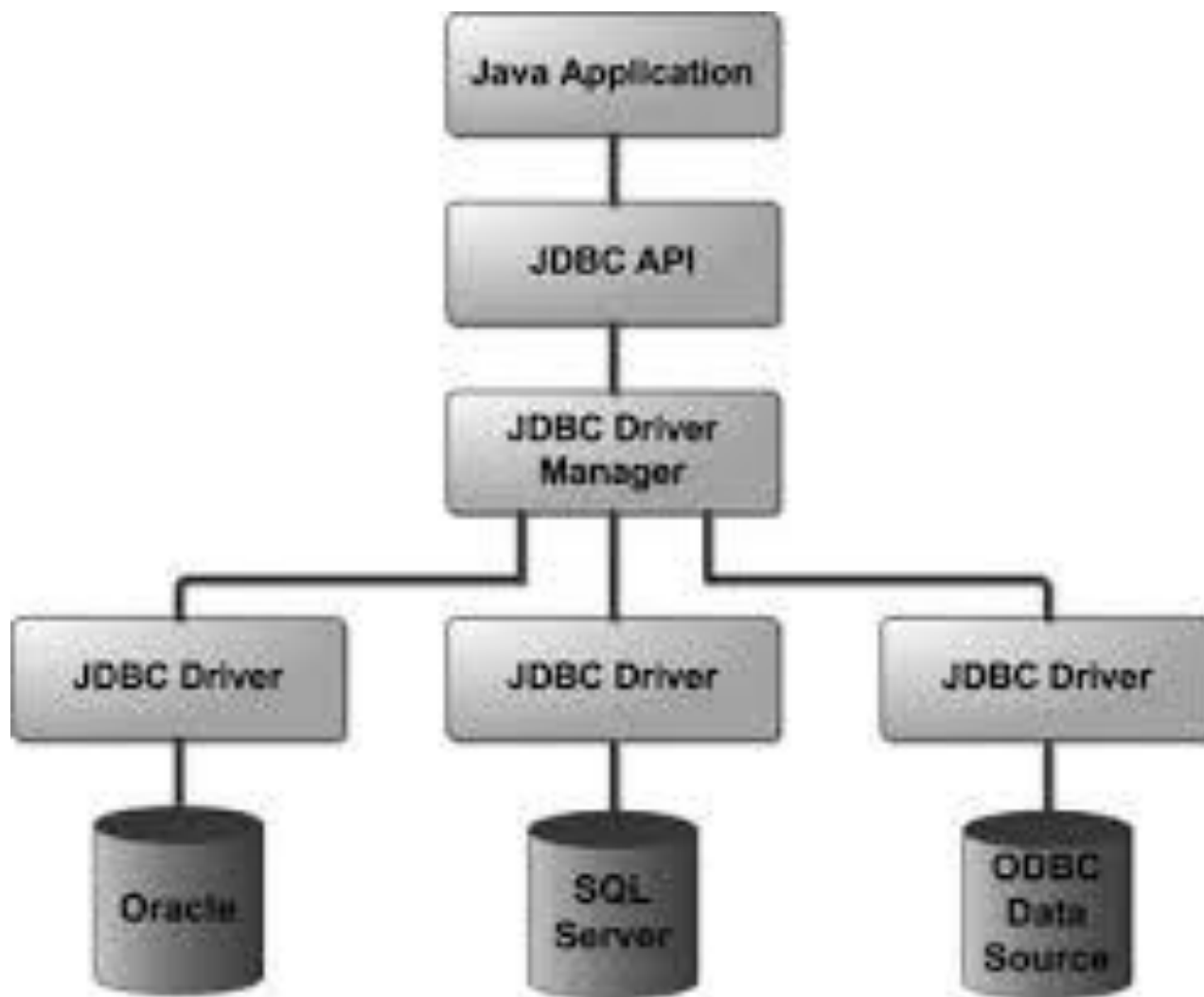
```
SELECT * FROM ACCOUNTS  
WHERE `login` = 'Admin'
```

# Пример **SELECT**

Выбрать всех **умных** админов 😊

```
SELECT * FROM ACCOUNTS  
WHERE `login` = 'Admin' AND  
`password` <> 'qwert';
```

# Архитектура JDBC



# Архитектура JDBC

1. JDBC API – поставляется компанией SUN.
2. JDBC API служит для соединения с **любой** базой данных под управлением **любой** СУБД.
3. Но в мире десятки и сотни СУБД, более того, никто не знает, в какой день появится новая СУБД, которая будет лучше остальных!
4. Несомненно, каждая СУБД использует **свои** алгоритмы по хранению и модификации данных.
5. Вопрос – как бедняги из SUN смогли написать **столько кода**, чтобы JDBC работал с **любой** СУБД?

# Архитектура JDBC

1. Ответ – они не написали **ни строки кода для работы с определенной СУБД.**
2. JDBC API – это только лишь **набор интерфейсов!!!**
3. Эти интерфейсы говорят – **что** Java разработчик **может сделать** с базой данных.
4. Но они не говорят – **как это сделать.**
5. Для того, чтобы работать с **определенной** СУБД необходима **реализация JDBC API для этой СУБД.**
6. Реализация для JDBC разрабатывается и поставляется **производителем СУБД.**
7. Если какая-либо СУБД предоставляет реализацию JDBC она считается Java-совместимой.
8. Вывод - JDBC API служит для соединения с **любой Java-совместимой** базой данных, и отвечают за эту совместимость сами производители!
9. Как такую ситуацию называют люди? – «если не можешь победить движение — возглавь его»! 😊

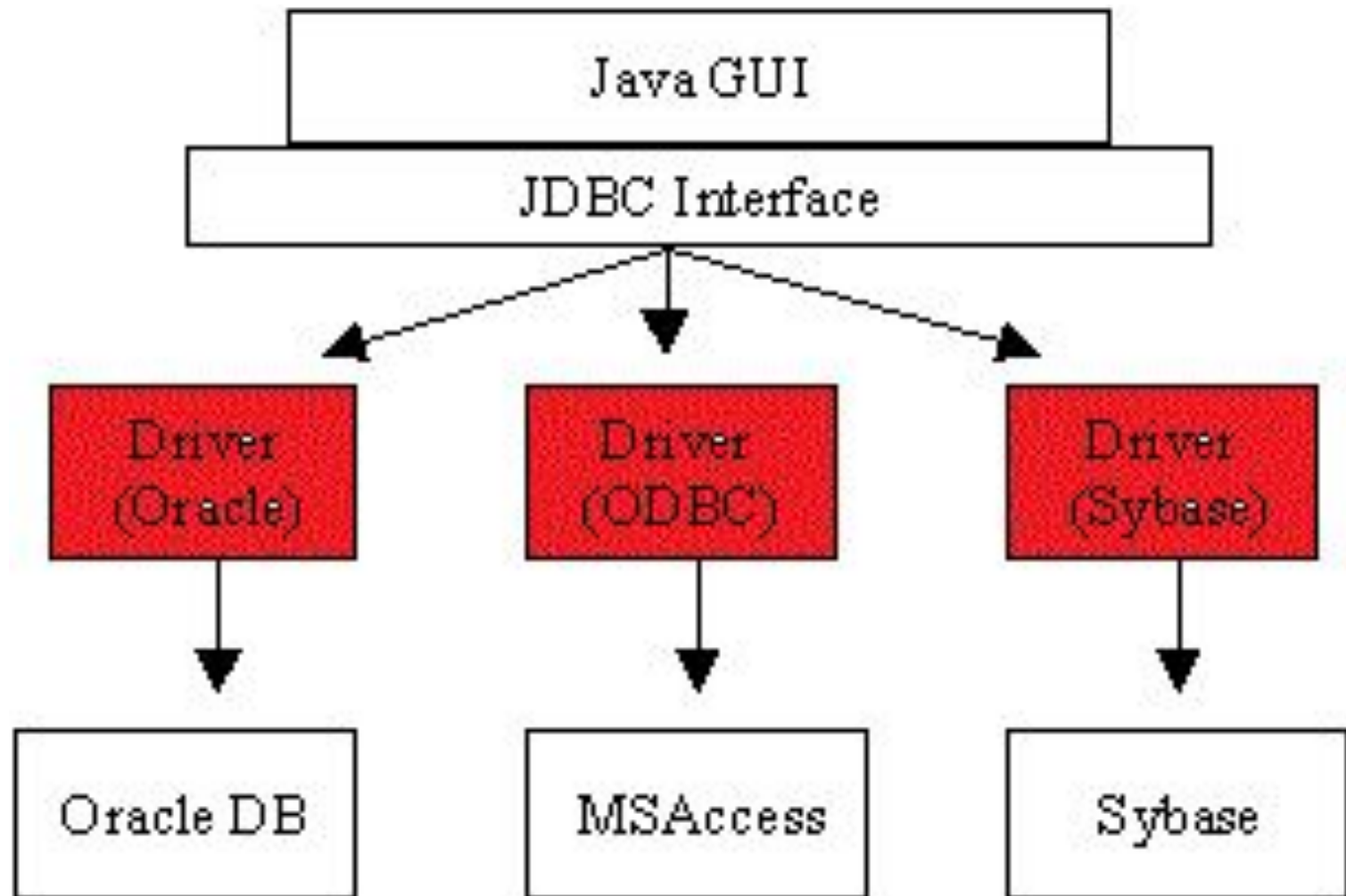
# Работа с БД на Java

## Последовательность шагов

1. Подключить JDBC драйвер для СУБД.
2. Соединиться с Базой Данных.
3. Сформировать и выполнить запрос.
4. Обработать результаты запроса.
- 5. Заккрыть ранее открытые ресурсы.**



# Шаг 1 – JDBC Драйвер для СУБД



# Шаг 1 – JDBC Драйвер для СУБД

1. Как уже говорили JDBC – это всего лишь **интерфейсы**.
2. Для того, чтобы работать с базой – нужна **реализация**.
3. Такая реализация называется **драйвером**.
4. Другое название – **Connector**.
5. Драйвер обычно поставляется в виде **JAR-файла**.

# Шаг 1 – JDBC Драйвер для СУБД

1. Подключение драйвера проходит в два этапа:
  - Загрузка JAR-файла драйвера в CLASS\_PATH.
  - Регистрация драйвера в исходном коде.

# Шаг 1.1 – Загрузка JAR-файла в CLASS\_PATH.



## Шаг 1.2 – загрузка драйвера в коде

### 1. Когда драйвер грузится плохо!

```
try {  
  
    java.sql.Driver dbDriver = new com.mysql.jdbc.Driver();  
  
    DriverManager.registerDriver(dbDriver);  
  
} catch (SQLException e) {  
  
    e.printStackTrace();  
  
}
```

Шаг 1. 2 – загрузка драйвера в  
коде

## 2. Когда драйвер грузится хорошо 😊

```
try {  
  
    Class.forName("com.mysql.jdbc.Driver");  
  
} catch (ClassNotFoundException e) {  
    e.printStackTrace();  
}
```

# Шаг 2 – Соединение с БД

```
//Connection connection = DriverManager.getConnection(String url);  
//Connection connection = DriverManager.getConnection(String url, String user, String password);  
//Connection connection = DriverManager.getConnection(String url, Properties info)  
  
try {  
    Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/TRAINING_DB", "root", "root");  
} catch (SQLException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}
```

# Шаг 3 Выполнение SQL запросов

## Объект Statement

```
Statement statement = connection.createStatement();  
PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM COURSES");  
CallableStatement callableStatement = connection.prepareCall("SELECT * FROM COURSES");
```



# Шаг 4 – обработка результатов. Объект ResultSet.

```
ResultSet resultSet = callableStatement.executeQuery();

ResultSetMetaData metaData = resultSet.getMetaData();

while(resultSet.next()) {
    Integer courseId = resultSet.getInt("PK_COURSE_ID");
    String courseName = resultSet.getString("COURSE_NAME");

    System.out.println(courseId + " - " + courseName);
}
```

# Шаг 5 – закрытие ресурсов.

```
    } finally {  
        if (resultSet != null) {  
            try {  
                resultSet.close();  
                resultSet = null;  
            } catch (SQLException e) {  
                e.printStackTrace();  
            }  
        }  
  
        if (statement != null) {  
            try {  
                statement.close();  
                statement = null;  
            } catch (SQLException e) {  
            }  
        }  
  
        if (connection != null) {  
            try {  
                connection.close();  
                connection = null;  
            } catch (SQLException e) {  
            }  
        }  
    }  
}
```

# All in One

```
Connection connection = null;
PreparedStatement statement = null;
ResultSet resultSet = null;

try {
    Class.forName("com.mysql.jdbc.Driver");

    connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/TRAINING_DB", "root", "root");
    statement = connection.prepareStatement("SELECT * FROM COURSES");
    resultSet = statement.executeQuery();

    while(resultSet.next()) {
        Integer courseId = resultSet.getInt("PK_COURSE_ID");
        String courseName = resultSet.getString("COURSE_NAME");

        System.out.println(courseId + " - " + courseName);
    }
} catch (Exception e) {
    throw new ServletException(e);
} finally {
    // Closing Resources...
}
```

# Connection Pool

- Connection к БД – это «ресурс», который нужно получить.
- Мы говорили, что после обработки результатов из БД ресурсы нужно обязательно закрыть.
- Любое JavaEE приложение – это высоконагруженное приложение – множество пользователей отправляют множество запросов к БД.
- Но! Получение соединения – очень долгий процесс – приложение будет «тормозить».
- Что если, после получения соединения использовать его, но не закрывать, а **оставить на потом**?
- Тогда следующий пользователь, которому нужен доступ к базе не будет открывать соединение, а воспользуется уже ранее открытым!
- Чуть позже мы вернемся к этому вопросу, а пока что у меня к Вам предложение-задание – **реализуйте свой ConnectionPool**.

# Tomcat Connection Pool

## Step 1 – server.xml

```
<GlobalNamingResources>
  <!-- Editable user database that can also be used by
       UserDatabaseRealm to authenticate users
  -->
  <Resource auth="Container" description="User database that can be updated and saved" factory="org.apache.tomcat.jdbc.pool.DataSource" type="javax.sql.DataSource"
    maxActive="100" maxIdle="30" maxWait="10000"
    username="root" password="root" driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/training_db"/>
</GlobalNamingResources>
```

# Tomcat Connection Pool

## Step 2 – context.xml

```
<!-- <Resource name="jdbc/TrainingDB" auth="Container" type="javax.sql.DataSource"
      maxActive="100" maxIdle="30" maxWait="10000"
      username="root" password="root" driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/training_db"/> -->
<ResourceLink name="jdbc/TrainingDB" global="jdbc/TrainingDB" type="javax.sql.DataSource"/>
```

# Tomcat Connection Pool

## Step 3 – web.xml

```
<resource-ref>  
  <description>DB Connection</description>  
  <res-ref-name>jdbc/TrainingDB</res-ref-name>  
  <res-type>javax.sql.DataSource</res-type>  
  <res-auth>Container</res-auth>  
</resource-ref>
```



# Tomcat Connection Pool

## Step 4 – Java Code

```
Connection connection = null;
PreparedStatement statement = null;
ResultSet resultSet = null;

try {
    Context initContext = new InitialContext();
    Context envContext = (Context)initContext.lookup("java:/comp/env");
    DataSource ds = (DataSource)envContext.lookup("jdbc/TrainingDB");
    connection = ds.getConnection();

    statement = connection.prepareStatement("SELECT * FROM COURSES");
    resultSet = statement.executeQuery();

    while(resultSet.next()) {
        Integer courseId = resultSet.getInt("PK_COURSE_ID");
        String courseName = resultSet.getString("COURSE_NAME");

        System.out.println(courseId + " - " + courseName);
    }
} catch (Exception e) {
    throw new ServletException(e);
}
```