C:\"Program Files"\MySQL\"MySQL Server 8.0"\bin\mysql.exe -u root –p

/c/’Program Files’/MySQL/’MySQL Server 8.0’/bin

<https://trofimovdigital.ru/blog/mysql-command-line>

<https://javarush.ru/groups/posts/2172-jdbc-ili-s-chego-vsje-nachinaetsja>

1. Insert password
2. SHOW DATABASE
3. USE – we come in the database
4. SHOW TABLES
5. etc.
6. CREATE DATABASE DB\_NAME- We create a new database
7. QUIT- exit into windows cmd
8. C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -prootRoot DB\_NAME < C:\Users\User\Desktop\programming\JavaPrograms\Skillbox\10 MySQL в JAVA\zadaniya\skillbox\_sql\_dump.sql – we add in the DB\_NAME new data
9. SELECT \* FROM TABLE\_NAME- return the needed table

Получить информацию об установленной версии MySQL

mysql -V

Для подключения к mysql в консоли наберите команду

mysql -h you\_sql\_server -u user\_name -p

* h — хост c MySQL. Если подключаемся с локальной машины, параметр можно опустить
* u — имя пользователя MySQL (root или другой пользователь MySQL)
* p — пароль, который будет предложено ввести после нажатия enter

Показать все таблицы выбранной базы данных в текущей БД.

SHOW TABLES;

Показать все таблицы базы данных db\_name.

SHOW TABLES FROM db\_name;

Показать список столбцов в таблице table\_name в текущей БД

SHOW COLUMNS FROM table\_name;

Показать список столбцов в таблице table\_name из БД db\_name

SHOW COLUMNS FROM table\_name FROM db\_name;

Вывести структуру нужной таблицы

DESCRIBE table\_name;

Показать структуру таблицы, будет выведен sql-запрос на её создание через «CREATE TABLE».

SHOW CREATE TABLE table\_name;

Вывести значения системных переменных.

SHOW VARIABLES;

Показать список выполняющихся в настоящий момент запросов.

SHOW PROCESSLIST;

Общая статистика MySQL.

SHOW STATUS;

Статистика по всем таблицам в базе db\_name.

SHOW TABLE STATUS FROM db\_name;

Что бы выполнять запросы к MySQL из консоли, не всегда требуется предварительно подключаться к mysql. Параметр -e позволяет исполнить команду, вывести результат на экран, после чего отключиться от сервера MySQL. Например, можно вывести список таблиц базы данных.

mysql -uroot -e 'SHOW TABLES' db\_name && echo done

Создание базы данных из консоли сервера

$ mysqladmin -u root -p create db\_name

Удаление базы данных из консоли сервера

mysqladmin -u root -p drop db\_name

Создание базы данных db\_name из консоли MySQL

CREATE DATABASE db\_name COLLATE utf8\_general\_ci;

Удаление базы данных db\_name из консоли MySQL

DELETE DATABASE db\_name;

Замена в поле одной подстроки на другую

UPDATE table SET field=replace(field,'original string','new string');

**Работа с пользователями**

Вывести список пользователей

SELECT User,Host FROM mysql.user;

Показать список прав пользователя user

SHOW GRANTS FOR user\_name FROM db\_name;

Создать нового пользователя

CREATE USER 'user'@'host' IDENTIFIED BY'password';

host — здесь имя хоста, доменное имя или ip адрес, с которого пользователь сможет подключаться к серверу, например user@localhost.

Чтобы создаваемый пользователь смог подключаться к серверу MySQL с любого IP адреса или хоста (за исключением localhost), можно использовать символ процента, вот так

CREATE USER 'user'@'%' IDENTIFIED BY 'password';

Чтобы разрешрешить пользователю подключаться вообще со всех хостов, придётся создать для него две учётные записи.

CREATE USER 'user'@'%' IDENTIFIED BY 'password';

CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';

Добавить указанные привилегии для таблиц БД db\_name пользователю user@localhost

GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER,INDEX

ON db\_name.\* TO 'user'@'localhost';

Чтобы изменить права пользователю, иногда удобно сначала сбросить все права

REVOKE ALL PRIVILEGES ON \*.\* FROM 'user'@'localhost';

А затем установить нужные с помощью GRANT, как было показано выше.

Добавить все привилегии для таблиц БД db\_name пользователю user@localhost

GRANT ALL PRIVILEGES ON db\_name.\* TO 'user'@'localhost';

Удаление привилегий пользователя user@localhost для БД db\_name:

REVOKEALLON db\_name.\* FROM 'user'@'localhost';

Сделать из пользователя суперпользователя и дать полный доступ ко всем БД на сервере

GRANT ALL ON \*.\* TO 'user'@'localhost';

Удалить пользователя user@localhost

DROP USER user@localhost;

Перезагрузить привилегии

FLUSH PRIVILEGES;

Изменить пароль пользователя в консоли MySQL

SET PASSWORD FOR 'user'@'localhost' = PASSWORD('new\_password');

FLUSH PRIVILEGES;

или так

UPDATE mysql.user SET Password=PASSWORD('new\_password') WHERE User='user\_name';

FLUSH PRIVILEGES;

service mysqld restart;

Установить пароль для пользователя user в консоли сервера.

$ mysqladmin -uuser\_name password'password'

**Исправление и оптимизация баз данных в MySQL**

Чтобы исправить ошибки в поврежденной таблице, в консоли MySQL выполните:

> REPAIR TABLE table\_name;

Если же в БД много поврежденных таблиц можно воспользоваться командой mysqlcheck.

Проверить db\_name на ошибки.

$ mysqlcheck -p db\_name

Восстановление и оптимизация всех БД

$ mysqlcheck -Aor -p

Аргументы:

* p – использовать пароль
* -A, —all-databases – проверять все базы данных
* -o, —optimize – оптимизировать
* -r, —repair – восстанавливать повреждённые
* —auto-repair – автоматическое восстановление

**Бэкап MySQL из командной строки**

Экспорт базы MySQL

$ mysqldump -u username -p db\_name > dump.sql

Дамп нескольких баз

$ mysqldump -u username -p -B db\_name1 db\_name2 > dump.sql

Дамп всех баз на сервере

$ mysqldump -u username -p -A > dump.sql

Дамп только структуры базы, без данных

$ mysqldump -u username -p --no-data db\_name > database.sql

Дамп структуры одной таблицы mysql, без данных:

$ mysqldump -u username -p -h host db\_name table\_name --no-data > /path/dump.sql

Развернуть базу данных MySQL из дампа

$ mysql -u username -p db\_name < dump\_to\_restore.sql

**Клонирование таблиц**

Скопировать структуру и ключи таблицы, без копирования данных.

CREATE TABLE NEW\_TableName LIKE OLD\_TableName

Клонировать таблицы базы данных со всеми данными.

CREATE TABLE NEW\_TableNameSELECT \*FROM OLD\_TableName

или

CREATE TABLE NEW\_TableName

LIKE OLD\_TableName

INSERT INTO NEW\_TableName

SELECT \*

FROM OLD\_TableName;

## Connection

Connectivity - возможности устанавливать подключение. А подключение — это Connection

существует два способа подключения к БД:

* Через DriverManager
* Через DataSource

## Statements

* Statement: SQL выражение, которое не содержит параметров
* PreparedStatement : Подготовленное SQL выражение, содержащее входные параметры
* CallableStatement : SQL выражение с возможностью получить возвращаемое значение из хранимых процедур (SQL Stored Procedures).

Итак, имея подключение, мы можем в рамках этого подключения выполнить какой-нибудь запрос. Поэтому, логично, что экземпляр SQL выражения изначально мы получаем из Connection.

# Методы Statement : executeQuery, executeUpdate, execute

Класс **Statement** содержит три различных метода выполнения SQL-выражений : executeQuery, executeUpdate и execute, которые вызываются в зависимости от текста SQL-запроса.

### executeQuery

Метод **executeQuery** используется в запросах, результатом которых является один единственный набор значений, таких как запросов типа SELECT.

### executeUpdate

Метод **executeUpdate** следует использовать, как для выполнения операторов управления данными типа INSERT, UPDATE или DELETE (DML - Data Manipulation Language), так и для операторов определения структуры базы данных CREATE TABLE, DROP TABLE (DDL - Data Definition Language).

Результатом выполнения операторов INSERT, UPDATE, или DELETE является изменения одной или более строк таблицы.

Результатом выполнения метода **executeUpdate** является целочисленное значение, определяющее, сколько строк было модифицировано. Для выражений **DML**, которые не оперируют со строками, возвращаемое методом executeUpdate значение всегда равно нулю.

### execute

Метод **execute** используется, когда операторы SQL возвращают более одного набора данных, более одного счетчика обновлений или и то, и другое. Такая возможность редко используется программистами.

Объекты Statement сами по себе не "помнят" SQL-выражение. Оно передается в качестве аргумента методов Statement.executeXXX.

## ResultSet

ResultSet предоставляет методы для получения и манипуляции результатами выполненных запросов. То есть если метод execute вернул нам true, значит мы можем получить и ResultSet.

ут стоит отметить, что next — это метод, который двигает так называемый "курсор". Курсор в ResultSet указывает на некоторую строку. Таким образом, чтобы считать строку, на неё нужно этот самый курсор установить. Когда курсор перемещается, то метод перемещения курсора возвращает true, если курсор валидный (правильный, корректный), то есть указывает на данные. Если возвращает false, значит данных нет, то есть курсор не указывает на данные. Если попытаться получить данные с невалидным курсором, то мы получим ошибку: No data is available Ещё интересно, что через ResultSet можно обновлять или даже вставлять строки:

@Test

**public** **void** shouldInsertInResultSet() **throws** SQLException {

Statement statement = connection.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);

ResultSet resultSet = statement.executeQuery("SELECT \* FROM customers");

resultSet.moveToInsertRow();

resultSet.updateLong("id", 3L);

resultSet.updateString("name", "John");

resultSet.updateInt("age", 18);

resultSet.insertRow();

resultSet.moveToCurrentRow();

}

SKILLBOX

1. \_10\_2 show db skillbox using resultSet (I cant to fix character encoding for Russian language)
2. \_10\_3 show db skillbox without resultSet 90
3. public class \_10\_hw\_1 {  
    public static void showTable( ){  
    String url = "jdbc:mysql://localhost:3306/skillbox?characterEncoding=utf8";  
    String name = "root";  
    String pass = "rootRoot";  
    try{  
    Connection connection = DriverManager.*getConnection*(url, name, pass);  
    Statement statement = connection.createStatement();  
   // ResultSet resultSet = statement.executeQuery("SELECT course\_name FROM purchaselist ");  
    ResultSet resultSet = statement.executeQuery("SELECT course\_name, " +  
    "COUNT(\*) / (MAX(month(subscription\_date)) - " +  
    "MIN(month(subscription\_date)) + 1) " +  
    "AS count\_course\_name FROM purchaselist " +  
    "WHERE YEAR(subscription\_date) = 2018 GROUP " +  
    "BY course\_name");  
    while (resultSet.next()){  
    String courseName = resultSet.getString("course\_name");  
    String avg = resultSet.getString("count\_course\_name");  
    System.*out*.println(courseName + " - " + avg);  
    }  
    resultSet.close();  
    statement.close();  
    connection.close();  
    }  
     
    catch (Exception e){  
    System.*err*.println("\n\t\tException\n");  
    e.printStackTrace();  
    }  
    System.*out*.println("END");  
    }  
   }
4. \_10\_5 display name of curse and number of curse for this course using Hibernate. We use to use apply different classes to create connection and to ask to db. Now, we use hibernate.cfg.xml file, made class “Course”, where we described our table “courses”, connected to a class “Course” , printed values for name of course using getName and getStudentsCount(which we wrote personally in a class “Course”).

public class \_10\_5 {  
  
 public static void showTable(){  
 StandardServiceRegistry registry = new  
 StandardServiceRegistryBuilder()  
 .configure("hibernate.cfg.xml").build();  
 Metadata metadata = new MetadataSources(registry)  
 .getMetadataBuilder().build();  
 SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();  
 Session session = sessionFactory.openSession();  
 Course course = session.get(Course.class, 1);  
  
 System.*out*.println(course.getName() + " - number of " +  
 "students = " + course.getStudentsCount());  
  
 session.close();  
// Course.sessionCourseClose(Course.sessionCourse());  
  
 }  
}

1. Creating, changing and deleting entry in a table

public class \_10\_7 {  
 public static void showTable(){  
 StandardServiceRegistry registry = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();  
 Metadata metadata = new MetadataSources(registry).getMetadataBuilder().build();  
 SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();  
 Session session = sessionFactory.openSession();  
  
 Transaction transaction = session.beginTransaction();  
  
  
  
// Course course = new Course();//creating new element of a table(new entry in table has id = 48)  
// course.setName("New course");//creating new element  
 //course.setType(CourseType.BUSINESS);//creating new element  
 // course.setTeachersId(1);//creating new element  
 //Course course = session.get(Course.class, 48);//changing of real entry in table(we chose id = 48)  
 //course.setName("the Newestklkook course");//changing of real  
 //Course course = session.get(Course.class, 48);//deleting  
 // session.delete(course);//deleting  
  
  
  
 session.save(course);  
  
 transaction.commit();  
 session.close();  
 }  
  
}

1. public class \_10\_hw\_4\_2 {  
    public static void createTable() {  
    String url = "jdbc:mysql://localhost:3306/skillbox?characterEncoding=utf8";  
    String user = "root";  
    String password = "rootRoot";  
    try {  
    System.*out*.println("Creating of a table...");  
    Connection connection = DriverManager.*getConnection*(url, user, password);//this class using for connectivity to db  
    Statement statement = connection.createStatement(); //this class using for SQL requests  
    statement.executeUpdate("CREATE TABLE IF NOT EXISTS" +  
    " linked\_purchase\_list " +  
    "(student\_id INT PRIMARY KEY, " +  
    "course\_id INT)");//we get result of request  
     
   // resultSet.close();  
    statement.close();  
    connection.close();  
    } catch (Exception e) {  
    System.*err*.println(" exception");  
    e.printStackTrace();  
    }  
    System.*out*.println("Creating od a table is completed successfully");  
    }  
     
    public static void dropTable(){  
    StandardServiceRegistry registry = new StandardServiceRegistryBuilder()  
    .configure("hibernate.cfg.xml").build();  
    Metadata metadata = new MetadataSources(registry)  
    .getMetadataBuilder().build();  
    SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();  
    Session session = sessionFactory.openSession();  
    Transaction transaction = session.beginTransaction();  
    String str = LinkedPurchaseList.class.toString();  
    System.*out*.println(str);  
     
    String sql = "DROP TABLE IF EXISTS linked\_purchase\_list";  
     
    Query query = session.createSQLQuery(sql);  
     
    query.executeUpdate();  
    transaction.commit();  
    session.close();  
     
    }  
     
    public static void changeTableStudent(){  
    StandardServiceRegistry registry = new StandardServiceRegistryBuilder()  
    .configure("hibernate.cfg.xml").build();  
    Metadata metadata = new MetadataSources(registry).getMetadataBuilder().build();  
    SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();  
    Session session = sessionFactory.openSession();  
     
    Transaction transaction = session.beginTransaction();  
    for (int i = 1; i <= 100; i++) {  
    Student student = session.get(Student.class, i);  
    student.setName("| id " + i + " |");  
    session.save(student);  
    }  
     
     
    transaction.commit();  
    session.close();  
    }  
    public static void changeTableCourse(){  
    StandardServiceRegistry registry = new StandardServiceRegistryBuilder()  
    .configure("hibernate.cfg.xml").build();  
    Metadata metadata = new MetadataSources(registry).getMetadataBuilder().build();  
    SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();  
    Session session = sessionFactory.openSession();  
     
    Transaction transaction = session.beginTransaction();  
   // for (int i = 1; i <= 46; i++) {  
   // CourseForTeacher courseForTeacher = session.get(CourseForTeacher.class, i);  
   // courseForTeacher.setName("| Course id " + i + " |");  
   // session.save(courseForTeacher);  
   // }  
    CourseForTeacher courseForTeacher = session.get(CourseForTeacher.class, 54);//deleting  
    session.delete(courseForTeacher);//deleting  
    session.save(courseForTeacher);  
    transaction.commit();  
    session.close();  
    }  
     
     
     
    public static void fillTable(){  
    StandardServiceRegistry registry = new StandardServiceRegistryBuilder()  
    .configure("hibernate.cfg.xml").build();  
    Metadata metadata = new MetadataSources(registry)  
    .getMetadataBuilder().build();  
    SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();  
    Session session = sessionFactory.openSession();  
     
    CriteriaBuilder builder = session.getCriteriaBuilder();//this class uses for difficult queries  
    CriteriaQuery<Student> queryStudent = builder.createQuery(Student.class);  
    Root<Student> root = queryStudent.from(Student.class);//this expression looks like operator FROM in SQL  
    queryStudent.select(root);//SELECT operator in SQL  
    List<Student> studentList = session.createQuery(queryStudent).getResultList();  
     
    Transaction transaction = session.beginTransaction();  
     
    for (Student student: studentList) {  
    System.*out*.print(student.getId() + " ");  
    String hql = "FROM " + PurchaseList.class.getSimpleName() + " WHERE student\_name = " + student.getName();// HQL  
     
    List<PurchaseList> purchaseListList = session.createQuery(hql).getResultList();// get names of courses  
     
    /\*теперь надо эти id запихнуть в linkedCourseList\*/  
    for(PurchaseList purchaseList: purchaseListList) {  
     
    String hql2 = "FROM " + CourseForTeacher.class.getSimpleName()  
    + " WHERE name = " + purchaseList.getCourseName();  
    List<CourseForTeacher> courseForTeacherList= session.createQuery(hql2)  
    .setMaxResults(1).getResultList();  
     
    for(CourseForTeacher courseForTeacher: courseForTeacherList) {  
    Query queryLinked = session.createSQLQuery("INSERT INTO linked\_purchase\_list" +  
    " (student\_id,course\_id) VALUES (" +  
    student.getId() + "," + courseForTeacher.getId() + ");");  
    queryLinked.executeUpdate();  
    System.*out*.print(purchaseList.getCourseName() + " ");  
    }  
    }  
    }  
      
    transaction.commit();  
    session.close();  
    }  
   }