

Подключение СУБД H2 к программе на java.

H2 — открытая кроссплатформенная СУБД, полностью написанная на языке Java.

Основные особенности H2:

- Очень быстрая, с открытым исходным кодом, JDBC API
- Встроенный и серверный режимы; базы данных в памяти
- Консольное приложение на основе браузера
- Небольшой размер: размер файла jar около 2,5 МБ.

1. Скачаем драйвер JDBC для H2:

Заходим на официальный сайт проекта:

<https://www.h2database.com/html/main.html>

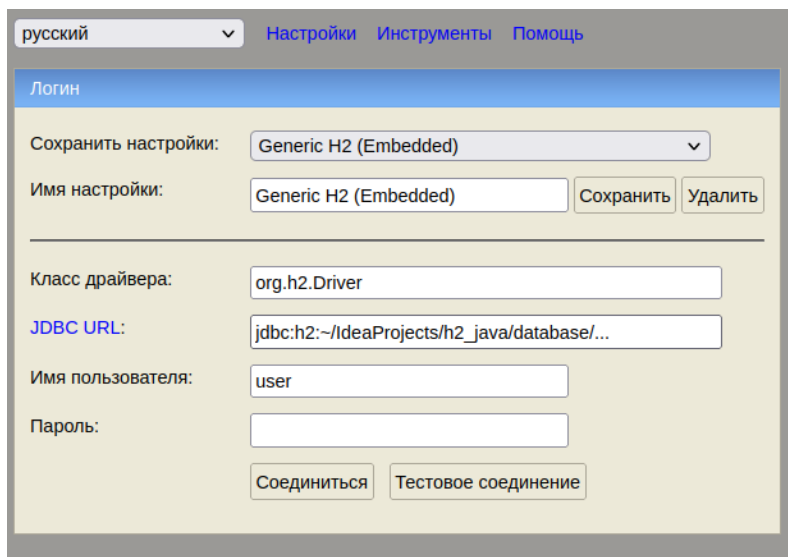
Находим форму «Download», скачиваем драйвер.

2. Создадим простой проект на Java и добавим драйвер jdbc:

Для этого, после создания проекта, открываем в верхнем меню вкладку «File» → «Project Structure» → «Libraries» и в верхнем меню второй вертикальной вкладки нажмём на «+», далее выберем «java» и в открывшемся окне нам необходимо прописать путь до драйвера H2.

3. запустим браузерное приложение h2-console:

Мне достаточно прописать h2-console в консоль и откроется новая вкладка в браузере:



Класс драйвера должен соответствовать org.h2.Driver, а JDBC URL я прописал путь до папки в моём проекте, в которой хранит файлы базы данных. Не нужно ничего создавать в папке, нужно указать саму папку, а

уже H2 все создаст за вас.

Имя пользователя : user, без пароля.

Пройдем тестовое соединение, что бы убедиться, что база данных настроенная успешно для работы:

Тест прошел успешно

3. Создадим класс «Database» через который будем работать с базой данных:

```
package com.company;

import java.sql.*;

public class Database {
    private Connection connection = null;
    Database(String url, String username, String password) {
        try {
            connection = DriverManager.getConnection(url, username, password);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public void createTable(String nameTable, String filds) {
        try {
            Statement statement = connection.createStatement();
            statement.execute("drop table if exists " + nameTable);
            statement.execute("create table " + nameTable + "( " + filds + " );");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public void insertInto(String tableName, String data) {
        try {
            Statement statement = connection.createStatement();
            statement.executeUpdate("insert into " + tableName + " values ( " + data
+ " );");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public int delete(String nameTable, String fild, String valueFild) {
        try {
            Statement statement = connection.createStatement();
            return statement.executeUpdate("delete from " + nameTable + " where " + fild + "
= " + valueFild);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
    }
    public void selectAll(String nameTable, String filds, int count) {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("select " + filds + " from " +
nameTable);
            while (resultSet.next()) {
                for(int i = 1; i <= count; i++) {
```

```

        System.out.print(resultSet.getString(i) + "\t");
    }
    System.out.println();
}
} catch (SQLException e) {
    e.printStackTrace();
}
}
public String select(String nameTable, String fild, String data, int count) {
    try {
        Statement statement = connection.createStatement();
        ResultSet resultSet =
            statement.executeQuery("select * from " + nameTable + " where " + fild +
" = " + data);
        String executeFilds = "";
        if(resultSet.next()) {
            for(int i = 1; i <= count; i++) {
                executeFilds += resultSet.getString(i) + "\t";
            }
            return executeFilds;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return "not data";
}
public int update(String nameTable, String fild, String currentData, String newData) {
    try {
        Statement statement = connection.createStatement();
        return statement.executeUpdate("update " + nameTable + " set " + fild + " = " +
newData + " where " + fild + " = " + currentData);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return 0;
}
public void insertClient(int id, String name, int age) {
    try {
        PreparedStatement preparedStatement =
            connection.prepareStatement("insert into client values(?, ?, ?);");
        preparedStatement.setInt(1, id);
        preparedStatement.setString(2, name);
        preparedStatement.setInt(3, age);
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
}

```

Подключаемся к базе данных передавая в метод getConnection путь до базы данных (jdbc:h2:~/IdeaProjects/h2_java/database/...), имя пользователя(user) и пароль («»).

```

private Connection connection = null;
Database(String url, String username, String password) {
    try {
        connection = DriverManager.getConnection(url, username, password);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

Метод `createTable` принимает как аргумент строковые объекты, имя таблицы и поля.

```
public void createTable(String nameTable, String fields) {  
    try {  
        Statement statement = connection.createStatement();  
        statement.execute("drop table if exists " + nameTable);  
        statement.execute("create table " + nameTable + "( " + fields + " );");  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Создаём объект `Statement`, вызываем метод `createStatement` у объекта `Connection`.

Объект `Statement` располагает тремя методами:

- `executeUpdate`: выполняет такие команды, как `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE`, `DROP TABLE`. В качестве результата возвращает количество строк, затронутых операциями (например, количество добавленных, измененных или удаленных строк), или 0, если ни одна строка не затронута операцией или если команда не изменяет содержимое таблицы (например, команда создания новой таблицы)
- `executeQuery`: выполняет команду `SELECT`. Возвращает объект `ResultSet`, который содержит результаты запроса.
- `execute()`: выполняет любые команды и возвращает значение `boolean`: `true` - если команда возвращает набор строк (`SELECT`), иначе возвращается `false`.

Вызываем метод `execute` с sql кодом «`drop table if exists` + имя таблицы» и «`create table` + имя таблицы + (+ поля таблицы +);»

Это значит, что если таблица с таким именем существует существует, мы её удаляем и создаём новую.

Метод `insertInto` принимает имя таблицы и данные.

Вызывается `executeUpdate`, он выполняет команды: `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE`, `DROP TABLE`. Возвращает количество затронутых строк.

Добавляем в таблицу данные переданные в метод.

```
public void insertInto(String tableName, String data) {  
    try {  
        Statement statement = connection.createStatement();  
        statement.executeUpdate("insert into " + tableName + " values ( " + data  
+ " );");  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Метод delete принимает имя таблицы, поле и значение поля.

Используется метод executeUpdate, удаляем данные из таблицы где поле fild имеет значение valueFild.

```
public int delete(String nameTable, String fild, String valueFild) {
    try {
        Statement statement = connection.createStatement();
        return statement.executeUpdate("delete from " + nameTable + " where " + fild + "
= " + valueFild);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return 0;
}
```

SelectAll принимает имя таблицы, поля и их количество.

Создается объект класса ResultSet, который сохранит в себя выборку из таблицы (возвращает метод executeQuery).

ResultSet имеет метод next(), если он возвращает истину, значит объект не пуст. Данные вывожу на экран.

```
public void selectAll(String nameTable, String filds, int count) {
    try {
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery("select " + filds + " from " +
nameTable);
        while (resultSet.next()) {
            for(int i = 1; i <= count; i++) {
                System.out.print(resultSet.getString(i) + "\t");
            }
            System.out.println();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Метод Select принимает имя таблицы, поле, данные и количество полей в таблице. Sql запрос требует вернуть записи, где введенное поле соответствует введенным данным. Даже если таких записей больше чем одно метод вернет первую запись.

```
public String select(String nameTable, String fild, String data, int count) {
    try {
        Statement statement = connection.createStatement();
        ResultSet resultSet =
            statement.executeQuery("select * from " + nameTable + " where " + fild +
" = " + data);
        String executeFilds = "";
        if(resultSet.next()) {
            for(int i = 1; i <= count; i++) {
                executeFilds += resultSet.getString(i) + "\t";
            }
            return executeFilds;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return "not data";
}
```

Update принимает имя таблицы, поле, текущие данные и новые данные и заменяет их в таблице для предложенного поля через метод `executeUpdate`.

```
public int update(String nameTable, String field, String currentData, String newData) {
    try {
        Statement statement = connection.createStatement();
        return statement.executeUpdate("update " + nameTable + " set " + field + " = " +
newData + " where " + field + " = " + currentData);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return 0;
}
```

Метод `insertClient`, специально написанный под работу с таблицей `Client`, уже осведомлён о её полях и требует ввести три типа данных `int(id)`, `String(name)` и `int(age)`. После чего создаётся класс `PreparedStatement`. Кроме собственно выполнения запроса этот класс позволяет подготовить запрос, отформатировать его должным образом.

Для создания объекта `PreparedStatement` применяется метод `prepareStatement()` класса `Connection`. В этот метод передается выражение sql «insert into client values (?, ?, ?)». Это выражение может содержать знаки вопроса ? - знаки подстановки, вместо которых будут вставляться реальные значения.

Чтобы связать отдельные знаки подстановки с конкретными значениями у класса `PreparedStatement` определен ряд методов для различных типов данных. Все методы, которые поставляют значения вместо знаков подстановки, в качестве первого параметра принимают порядковый номер знака подстановки (нумерация начинается с 1), а в качестве второго параметра - собственно значение, которое вставляется вместо знака подстановки.

Некоторые из них:

- `setBigDecimal`
- `setBoolean`
- `setDate`
- `setDouble`
- `setFloat`
- `setLong`
- `setNull`
- `setTime`

Таким образом метод более безопасен, и быст, так как он один раз создаёт запрос, а после подставляет в него значения.

```
public void insertClient(int id, String name, int age) {
    try {
        PreparedStatement preparedStatement =
            connection.prepareStatement("insert into client values(?, ?, ?);");
        preparedStatement.setInt(1, id);
        preparedStatement.setString(2, name);
        preparedStatement.setInt(3, age);
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
```

```
        e.printStackTrace();  
    }  
}
```

4. Создадим класс «Main»:

```
package com.company;  
  
public class Main {  
    public static void main(String[] args) {  
        Database database = new Database("jdbc:h2:mem:~/IdeaProjects/h2_java/database/...",  
        "user","");  
        database.createTable("Client", "id integer, name varchar(20), age integer");  
  
        database.insertInto("Client", "1, 'Ion Paminteanu', 27");  
  
        System.out.println("1) " + database.select("Client", "id", "1", 3));  
  
        database.delete("Client", "age", "27");  
  
        database.insertClient(2, "Iohan Cretu", 32);  
  
        System.out.println("2) " + database.select("Client", "id", "2", 3));  
  
        database.update("client","id", "2", "1");  
  
        System.out.println("3) " + database.select("Client", "id", "1", 3));  
  
        database.insertClient(2, "Arina Manole", 23);  
        System.out.println("Все данные :");  
        database.selectAll("Client", "*", 3);  
    }  
}
```

Создадим таблицу:

```
database.createTable("Client", "id integer, name varchar(20), age integer");
```

Добавим в неё данные:

```
database.insertInto("Client", "1, 'Ion Paminteanu', 27");
```

Выведем на экран:

```
System.out.println("1) " + database.select("Client", "id", "1", 3));
```

Удалим данные:

```
database.delete("Client", "age", "27");
```

Добавим новые данные:

```
database.insertClient(2, "Iohan Cretu", 32);
```

Выведем на экран:

```
System.out.println("2) " + database.select("Client", "id", "2", 3));
```

Обновим данные:

```
database.update("client","id", "2", "1");
```

выведем результат на экран:

```
System.out.println("3) " + database.select("Client", "id", "1", 3));
```

Добавим ещё одну запись и выведем все данные:

```
database.insertClient(2, "Arina Manole", 23);  
System.out.println("Все данные :");  
database.selectAll("Client", "*", 3);
```

Результат:

```
1) 1    Ion Paminteanu  27  
2) 2    Iohan Cretu  32  
3) 1    Iohan Cretu  32  
Все данные :  
1    Iohan Cretu  32  
2    Arina Manole  23
```