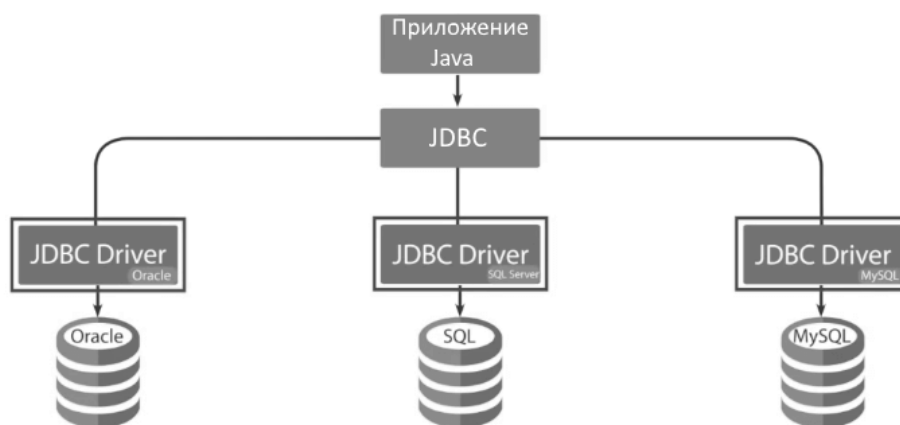


Раздел 3 (Базы данных в Java)

Для хранения данных мы можем использовать различные базы данных - Oracle, MS SQL Server, MySQL, Postgres и т.д. Все эти системы управления базами данных имеют свои особенности. Главное, что их объединяет это взаимодействие с хранилищем данных посредством команд SQL. И чтобы определить единый механизм взаимодействия с этими СУБД в Java еще начиная с 1996 был введен специальный прикладной интерфейс API, который называется **JDBC**.

То есть если мы хотим в приложении на языке Java взаимодействовать с базой данных, то необходимо использовать функциональные возможности JDBC. Данный API входит в состав Java (на текущий момент это версия JDBC 4.3), в частности, для работы с JDBC в программе Java достаточно подключить пакет **java.sql**. Для работы в Java EE есть аналогичный пакет **javax.sql**, который расширяет возможности JDBC.

Однако не все базы данных могут поддерживаться через JDBC. Для работы с определенной СУБД также необходим специальный драйвер. Каждый разработчик определенной СУБД обычно предоставляет свой драйвер для работы с JDBC. То есть если мы хотим работать с MySQL, то нам потребуется специальный драйвер для работы именно MySQL. Как правило, большинство драйверов доступны в свободном доступе на сайтах соответствующих СУБД. Обычно они представляют JAR-файлы. И преимущество JDBC как раз и состоит в том, что мы абстрагируемся от строения конкретной базы данных, а используем унифицированный интерфейс, который един для всех.



Для взаимодействия с базой данных через JDBC используются запросы SQL. В то же время возможности SQL для работы с каждой конкретной СУБД могут отличаться. Например, в MS SQL Server это T-SQL, в Oracle - это PL/SQL. Но в целом эти разновидности языка SQL не сильно отличаются.

Особенности запуска программы

На процесс компиляции необходимость работы с БД никак не сказывается, но влияет на процесс запуска программы. При запуске программы в командной строке необходимо указать путь к JAR-файлу драйвера после параметра - **classpath**.

```
java -classpath путь_к_файлу_драйвера:путь_к_классу_программы  
главный_класс_программы
```

Например, в папке C:\Java располагаются файл программы - Program.java, скомпилированный класс Program и файл драйвер, допустим, MySQL - mysql-connector-java-8.0.11.jar. Для выполнения класса Program мы можем использовать следующую команду:

```
java -classpath c:\Java\mysql-connector-java-8.0.11.jar;c:\Java  
Program
```

Если C:\Java является текущим каталогом, то мы можем сократить команду:

```
java -classpath mysql-connector-java-8.0.11.jar;. Program
```

В принципе мы можем и не использовать параметр -classpath, и запустить программу на выполнение обычным способом с помощью команды "java Program". Но в этом случае путь к драйверу должен быть добавлен в переменную Path.

Вначале создадим на сервере MySQL пустую базу данных, которую назовем **store** и с которой мы будем работать в приложении на Java. Для создания базы данных применяется выражение SQL:

1	CREATE DATABASE store;
---	------------------------

Его можно выполнить либо из консольного клиента MySQL Command Line Client, либо из графического клиента MySQL Workbench, которые устанавливаются вместе с сервером MySQL. Подробнее про создание базы данных можно прочитать в статье [Создание и удаление базы данных](#).

Для подключения к базе данных необходимо создать объект **java.sql.Connection**. Для его создания применяется метод:

1	Connection DriverManager.getConnection(url, username, password)
---	---

Метод DriverManager.getConnection в качестве параметров принимает адрес источника данных, логин и пароль. В качестве логина и пароля передаются логин и пароль от сервера MySQL. Адрес локальной базы данных MySQL указывается в следующем формате: jdbc:mysql://localhost/название_базы_данных

Пример создания подключения к созданной выше локальной базе данных store:

1	Connection connection = DriverManager.getConnection("jdbc:m
---	--

	<code>ysql://localhost/store", "root", "password");</code>
--	--

После завершения работы с подключением его следует закрыть с помощью метода **close()**:

	<pre> 1 Connection conn = 2 DriverManager.getConnection("jdbc:m 3 ysql://localhost/store", "root", "password"); // работа с базой данных connection.close(); </pre>
--	---

Либо мы можем использовать конструкцию **try**:

	<pre> 1 try (Connection conn = 2 DriverManager.getConnection("jdbc:m 3 ysql://localhost/store", "root", "password")){ // работа с базой данных } </pre>
--	---

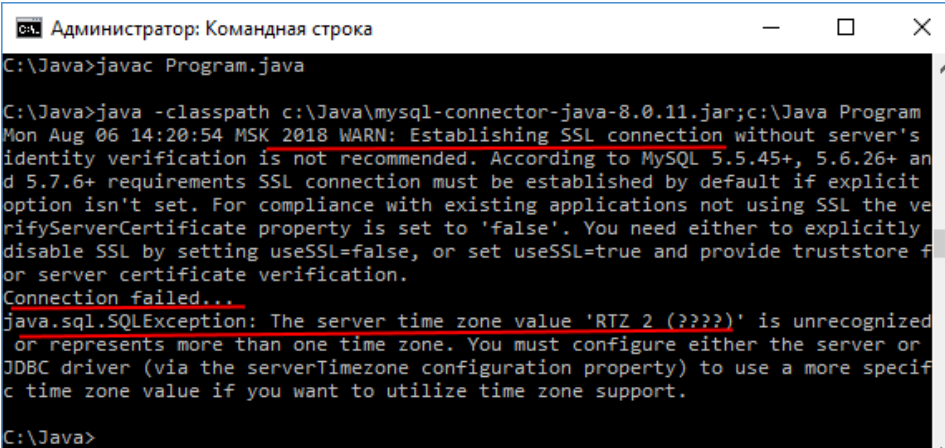
Возможные проблемы с часовыми поясами и SSL

При подключении к базе данных MySQL мы можем столкнуться с рядом проблем. Например, определим следующий код подключения:

	<pre> 1 import java.sql.Connection; 2 import java.sql.DriverManager; 3 4 public class Program{ 5 6 public static void main(String[] 7 args) { 8 try{ 9 String url = 10 "jdbc:mysql://localhost/store"; 11 String username = 12 "root"; 13 String password = 14 "password"; 15 16 Class.forName("com.mysql.cj.jdbc.Dr 17 iver").getDeclaredConstructor().new 18 Instance(); 19 try (Connection conn = 20 DriverManager.getConnection(url, 21 username, password)){ 22 </pre>
--	---

23	<pre> System.out.println("Connection to Store DB succesfull!"); } } catch(Exception ex){ System.out.println("Connection failed..."); System.out.println(ex); } } } </pre>
----	---

Даже если указаны правильно адрес базы данных, логин, пароль, мы все равно можем столкнуться с ошибками:



```

C:\Java>javac Program.java

C:\Java>java -classpath c:\Java\mysql-connector-java-8.0.11.jar;c:\Java Program
Mon Aug 06 14:20:54 MSK 2018 WARN: Establishing SSL connection without server's
identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ an
d 5.7.6+ requirements SSL connection must be established by default if explicit
option isn't set. For compliance with existing applications not using SSL the ve
rifyServerCertificate property is set to 'false'. You need either to explicitly
disable SSL by setting useSSL=false, or set useSSL=true and provide truststore f
or server certificate verification.
Connection failed...
java.sql.SQLException: The server time zone value 'RTZ 2 (???)' is unrecognized
or represents more than one time zone. You must configure either the server or
JDBC driver (via the serverTimezone configuration property) to use a more specif
ic time zone value if you want to utilize time zone support.

C:\Java>

```

Из консольного вывода видно, что проблема заключается с SSL и часовым поясом. Чтобы решить данную проблему, необходимо указать в адресе подключения часовой пояс бд и параметры для использования ssl. В частности, я указываю, что SSL не будет использоваться и что часовым поясом будет московский часовой пояс:

1	<pre>String url = "jdbc:mysql://localhost/store?serve rTimezone=Europe/Moscow&useSSL=fals e";</pre>
---	---

Параметры подключения указываются после вопросительного знака после названия базы данных. Параметр serverTimezone указывает на название часового пояса сервера бд. В данном случае "Europe/Moscow", список всех допустимых названий часовых поясов можно найти на странице <https://gist.github.com/kinjal/9105369>. И параметр useSSL=false указывает, что SSL не будет применяться.

Файлы конфигурации

Мы можем определить все данные для подключения непосредственно в программе. Однако что если какие-то данные были изменены? В этом случае потребуется перекомпиляция приложения. Иногда это не всегда удобно, например, отсутствует доступ к исходникам, или перекомпиляция займет довольно продолжительное время. В этом случае мы можем хранить настройки в файле.

Так, создадим в папке программы новый текстовый файл **database.properties**, в котором определим настройки подключения:

	1 url = 2 jdbc:mysql://localhost/store?server 3 Timezone=Europe/Moscow&useSSL=false username = root password = password
--	---

Загрузим эти настройки в программе:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28	import java.sql.*; import java.nio.file.*; import java.io.*; import java.util.*; public class Program{ public static void main(String[] args) { try{ Class.forName("com.mysql.cj.jdbc.Dr iver").getDeclaredConstructor().new Instance(); try (Connection conn = getConnection()){ System.out.println("Connection to Store DB succesfull!"); } } catch(Exception ex){ System.out.println("Connection failed..."); } } }
---	--

29	
30	System.out.println(ex);
31	}
32	}
33	
34	public static Connection
35	getConnection() throws SQLException, IOException{
	Properties props = new Properties();
	try(InputStream in = Files.newInputStream(Paths.get("dat abase.properties"))){
	props.load(in);
	}
	String url = props.getProperty("url");
	String username = props.getProperty("username");
	String password = props.getProperty("password");
	return DriverManager.getConnection(url, username, password);
	}
	}

Для взаимодействия с базой данных приложение отправляет серверу MySQL команды на языке SQL. Чтобы выполнить команду, вначале необходимо создать объект **Statement**.

Для его создания у объекта Connection вызывается метод **createStatement()**:

1	Statement statement = conn.createStatement();
---	--

Для выполнения команд SQL в классе Statement определено три метода:

- **executeUpdate**: выполняет такие команды, как INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE. В качестве результата возвращает количество строк, затронутых операцией (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.
Рассмотрим метод **executeUpdate()**. В качестве параметра в него передается собственно команда SQL:

	1	int executeUpdate("Команда_SQL")
--	---	----------------------------------

Ранее была создана база данных store, но она пустая, в ней нет таблиц и соответственно данных. Создадим таблицу и добавим в нее начальные данные:

	1	import java.sql.*;
	2	
	3	public class Program{
	4	
	5	public static void main(String[]
	6	args) {
	7	try{
	8	String url =
	9	"jdbc:mysql://localhost/store?serve
	10	rTimezone=Europe/Moscow&useSSL=fals
	11	e";
	12	String username =
	13	"root";
	14	String password =
	15	"password";
	16	
	17	Class.forName("com.mysql.cj.jdbc.Dr
	18	iver").getDeclaredConstructor().new
	19	Instance();
	20	// команда создания
	21	таблицы
	22	String sqlCommand =
	23	"CREATE TABLE products (Id INT
	24	PRIMARY KEY AUTO_INCREMENT,
	25	ProductName VARCHAR(20), Price
	26	INT)";
	27	
	28	try (Connection conn =
	29	DriverManager.getConnection(url,
		username, password)){
		Statement statement
		= conn.createStatement();
		// создание таблицы

	<pre> statement.executeUpdate(sqlCommand) ; System.out.println("Database has been created!"); } } catch(Exception ex){ System.out.println("Connection failed..."); System.out.println(ex); } } } </pre>
--	--

То есть в данном случае мы выполняем команду `CREATE TABLE products (Id INT PRIMARY KEY AUTO_INCREMENT, ProductName VARCHAR(20), Price INT)`, которая создает таблицу Products с тремя столбцами: Id - идентификатор строки, ProductName - строковое название товара и Price - числовая цена товара.

При этом если необходимо выполнить сразу несколько команд, то необязательно создавать новый объект Statement:

1	Statement statement =
2	conn.createStatement();
3	
4	statement.executeUpdate("Команда_SQL1");
5	statement.executeUpdate("Команда_SQL2");
	statement.executeUpdate("Команда_SQL3");

Для добавления, редактирования и удаления данных мы можем использовать рассмотренный в прошлой теме метод **executeUpdate**. С помощью результата метода мы можем проконтролировать, сколько строк было добавлено, изменено или удалено.

Добавление

Так, возьмем созданную в прошлой теме таблицу Products:

1	CREATE TABLE Products (
2	

3	Id INT PRIMARY KEY
4	AUTO_INCREMENT,
5	ProductName VARCHAR(20),
	Price INT
)

И добавим в эту таблицу несколько объектов:

1	import java.sql.*;
2	
3	public class Program{
4	
5	public static void main(String[]
6	args) {
7	try{
8	String url =
9	"jdbc:mysql://localhost/store?serve
10	rTimezone=Europe/Moscow&useSSL=fals
11	e";
12	String username =
13	"root";
14	String password =
15	"password";
16	
17	Class.forName("com.mysql.cj.jdbc.Dr
18	iver").getDeclaredConstructor().new
19	Instance();
20	
21	try (Connection conn =
22	DriverManager.getConnection(url,
23	username, password)){
24	
25	Statement statement
26	= conn.createStatement();
	int rows =
	statement.executeUpdate("INSERT
	Products(ProductName, Price) VALUES
	('iPhone X', 76000)," +
	"('Galaxy S9',
	45000), ('Nokia 9', 36000)");
	System.out.printf("Added %d rows",
	rows);
	}
	}

	<pre> catch(Exception ex){ System.out.println("Connection failed..."); System.out.println(ex); } } }</pre>
--	--

Для добавления данных в БД применяется команда INSERT. В данном случае в таблицу Products добавляется три объекта. И после выполнения программы на консоли мы увидим число добавленных объектов:

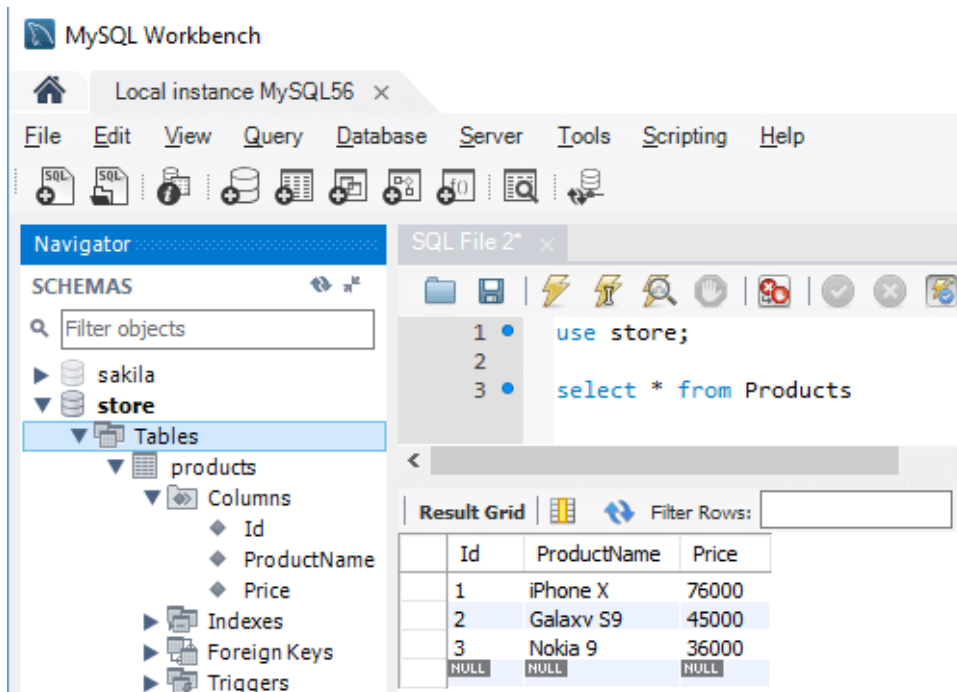
```
C:\Java>javac Program.java
```

```
C:\Java>java -classpath c:\Java\mysql-connector-java-
8.0.11.jar;c:\Java Program
```

```
Added 3 rows
```

```
C:\Java>
```

А добавленные строки мы можем увидеть в таблице в бд MySQL:



Редактирование

Изменим строки в таблице, например, уменьшим цену товара на 5000 единиц. Для изменения применяется команда UPDATE:

1	import java.sql.*;
---	--------------------

```

2
3 public class Program{
4
5     public static void main(String[]
6 args) {
7         try{
8             String url =
9 "jdbc:mysql://localhost/store?serve
10 rTimezone=Europe/Moscow&useSSL=fals
11 e";
12             String username =
13 "root";
14             String password =
15 "password";
16
17 Class.forName("com.mysql.cj.jdbc.Dr
18 iver").getDeclaredConstructor().new
19 Instance();
20
21             try (Connection conn =
22 DriverManager.getConnection(url,
23 username, password)){
24
25                 Statement statement
26 = conn.createStatement();
27
28                 int rows =
29 statement.executeUpdate("UPDATE
30 Products SET Price = Price -
31 5000");
32
33 System.out.printf("Updated %d
34 rows", rows);
35             }
36         }
37         catch(Exception ex){
38
39 System.out.println("Connection
40 failed...");
41
42 System.out.println(ex);
43         }
44     }
45 }

```

	}
--	---

Удаление

Удалим один объект из таблицы с помощью команды DELETE:

1	import java.sql.*;
2	
3	public class Program{
4	
5	public static void main(String[]
6	args) {
7	try{
8	String url =
9	"jdbc:mysql://localhost/store?serve
10	rTimezone=Europe/Moscow&useSSL=false
11	e ";
12	String username =
13	"root";
14	String password =
15	"password";
16	
17	Class.forName("com.mysql.cj.jdbc.Dr
18	iver").getDeclaredConstructor().new
19	Instance();
20	
21	try (Connection conn =
22	DriverManager.getConnection(url,
23	username, password)){
24	
25	Statement statement
26	= conn.createStatement();
	int rows =
	statement.executeUpdate("DELETE
	FROM Products WHERE Id = 3");
	System.out.printf("%d row(s)
	deleted", rows);
	}
	}
	catch(Exception ex){
	System.out.println("Connection
	failed...");

	<pre>System.out.println(ex); } } }</pre>
--	--

Как видно из примеров, не так сложно взаимодействовать с базой данных.
Достаточно передать в метод `executeUpdate` нужную команду SQL.