**Java и база данных**

**JDBC**

JDBC — это платформенно независимый промышленный стандарт взаимодействия Java-приложений с реляционными базами данных. Впервые был включен в состав JDK 1.1 в 1997 году. JDBC управляет:

* подключением к базе данных;
* выдачей запросов и команд;
* обработкой данных, полученных из базы.

**Как работает JDBC**

Пакет JDBC состоит из двух главных компонентов:

1. API (программного интерфейса), который поддерживает связь между Java-приложением и менеджером JDBC;
2. Драйвера JDBC, который поддерживает связь между менеджером JDBC и драйвером базы данных.

Соединение с базой устанавливается по-особому URL. При этом разработчику не нужно знать специфику конкретной базы — API выступает в качестве посредника между базой и приложением. Это упрощает как процесс создания приложения, так и переход на базу данных другого типа.

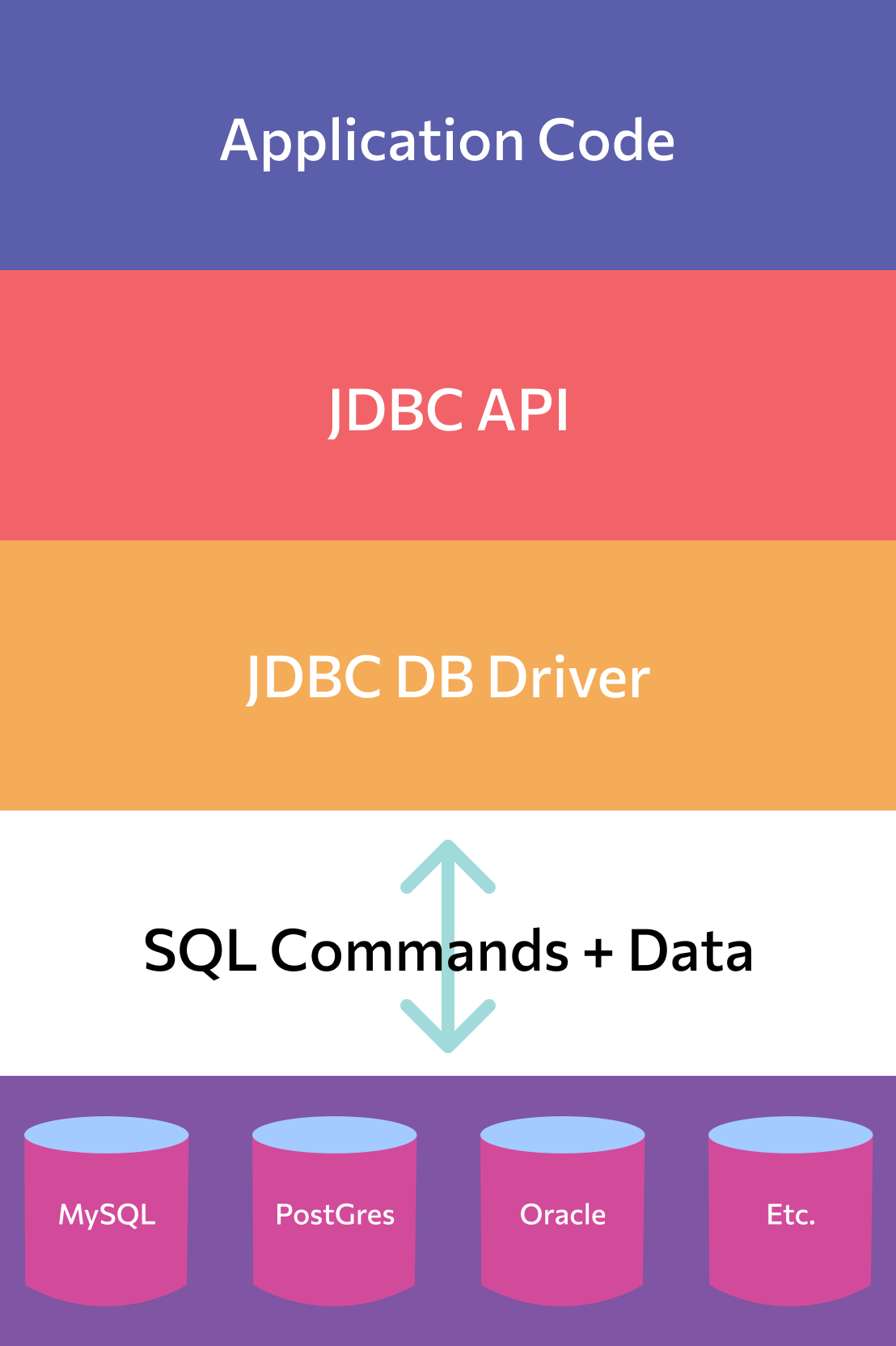
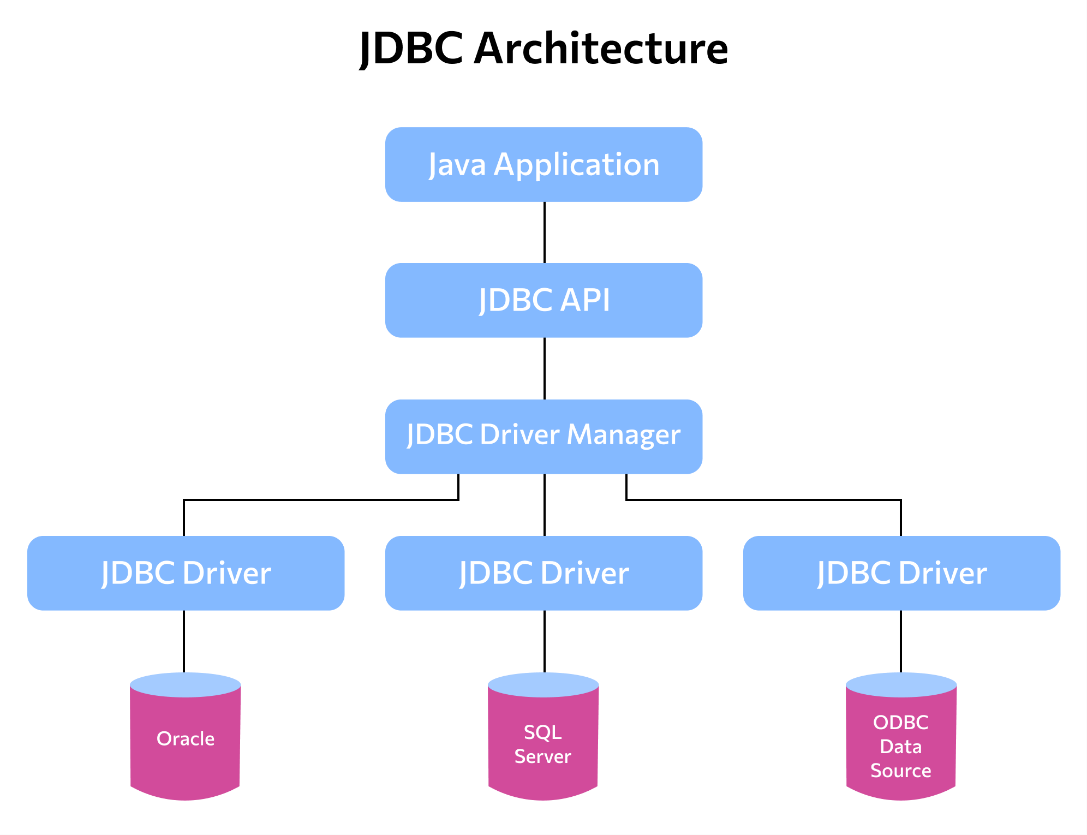


Схема взаимодействия компонентов JDBC, базы данных и Java-приложения

**Этапы подключения к базе данных**

1. Установка базы данных на сервер или выбор облачного сервиса, к которому нужно получить доступ.
2. Подключение библиотеки JDBC.
3. Проверка факта нахождения необходимого драйвера JDBC в classpath.
4. Установление соединения с базой данных с помощью библиотеки JDBC.
5. Использование установленного соединения для выполнения команд SQL.
6. Закрытие соединения после окончания сеанса.

Рассмотрим каждый из этих шагов подробнее.



Архитектура JDBC

**Импорт JDBC в Java-приложение**

Для использования JDBC, как и в случае со всеми остальными приложениями на платформе Java, в системе должен быть установлен JDK. Код для работы с JDBC можно писать как в среде разработки (IDE), так и в обычном текстовом редакторе. Простейшая программа может выглядеть так:

class WhatIsJdbc{

public static void main(String args[]){

System.out.println(«Hello World»);

}

}

Скомпилируйте этот код с помощью команды:

javac WhatIsJdbc.java

Каждый из импортированных модулей предоставляет доступ к классам, которые необходимы для взаимодействия Java-приложения с базой данных:

1. Connection подготавливает подключение к базе.
2. DriverManager обеспечивает подключение. Другая опция — модуль DataSource.
3. SQLException обрабатывает SQL-ошибки, возникающие при взаимодействии приложении и базы данных.
4. ResultSet и Statement моделируют наборы результатов данных и операторы SQL.

## **Добавление JDBC-драйвера в classpath**

JDBC-драйвер — это класс, обеспечивающий взаимодействие интерфейса JDBC API с базой данных определенного типа. Драйвер для SQLite представляет собой .jar-файл — его нужно добавить в classpath, как показано ниже:

java.exe -classpath /path-to-driver/sqlite-jdbc-3.23.1.jar:. WhatIsJdbc

**Установление соединения с базой данных**

Теперь в classpath есть доступ к драйверу. Вставьте приведенный ниже код в файл с вашей первой программой:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.ResultSet;

import java.sql.Statement;

class WhatIsJdbc{

public static void main(String[] args) {

Connection conn = null;

try {

String url = «jdbc:sqlite:path-to-db/chinook/chinook.db»;

conn = DriverManager.getConnection(url);

System.out.println(«Соединение установлено»);

} catch (SQLException e) {

throw new Error(«Ошибка при подключении к базе данных», e);

} finally {

try {

if (conn != null) {

conn.close();

}

} catch (SQLException ex) {

System.out.println(ex.getMessage());

}}}}

Теперь можно компилировать и запускать код. В случае успешного подключения появится сообщение «Соединение установлено».

**Создание запроса к базе данных**

В приведенном ниже примере показано, как создать запрос к базе SQLite, используя Connection и Statement:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.ResultSet;

import java.sql.Statement;

class WhatIsJdbc{

public static void main(String[] args) {

Connection conn = null;

try {

String url = «jdbc:sqlite:path-to-db-file/chinook/chinook.db»;

conn = DriverManager.getConnection(url);

Statement stmt = null;

String query = «select \* from albums»;

try {

stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query);

while (rs.next()) {

String name = rs.getString(«title»);

System.out.println(name);

}

} catch (SQLException e ) {

throw new Error(«Problem», e);

} finally {

if (stmt != null) { stmt.close(); }

}

} catch (SQLException e) {

throw new Error(«Problem», e);

} finally {

try {

if (conn != null) {

conn.close();

}

} catch (SQLException ex) {

System.out.println(ex.getMessage());

}}}

Результатом выполнения этого кода будет вывод списка музыкальных альбомов из учебной базы в консоль.

Возможностей JDBC API достаточно для реализации простых приложений. Для более масштабных решений чаще используют JPA API, который позволяет сохранять Java-объекты в базе данных.

**Добавление, изменение и удаление данных**

Для добавления, редактирования и удаления данных мы можем ипользовать рассмотренный ранее метод **executeUpdate**. С помощью результата метода мы можем проконтроллировать, сколько строк было добавлено, изменено или удалено.

**Добавление**

Так, возьмем созданную таблицу Products:

CREATE TABLE Products (

Id INT PRIMARY KEY AUTO\_INCREMENT,

ProductName VARCHAR(20),

Price INT

)

И добавим в эту таблицу несколько объектов:

try{

String url = "jdbc:mysql://localhost/store?serverTimezone=Europe/Moscow&useSSL=false";

String username = "root";

String password = "password";

Class.forName("com.mysql.cj.jdbc.Driver").getDeclaredConstructor().newInstance();

try (Connection conn = DriverManager.getConnection(url, username, password)){

Statement statement = 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);

}

}

catch(Exception ex){

System.out.println("Connection failed...");

System.out.println(ex);

}

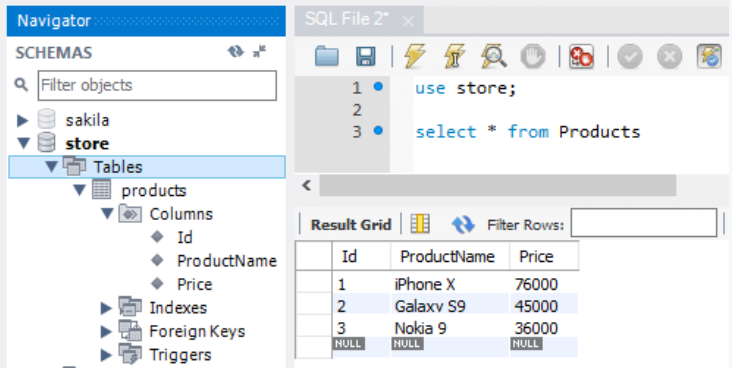
Для добавления данных в БД применяется команда INSERT. В данном случае в таблицу Products добавляется три объекта. И после выполнения программы на консоли мы увидим число добавленных объектов:

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:

try{

String url = "jdbc:mysql://localhost/store?serverTimezone=Europe/Moscow&useSSL=false";

String username = "root";

String password = "password";

Class.forName("com.mysql.cj.jdbc.Driver").getDeclaredConstructor().newInstance();

try (Connection conn = DriverManager.getConnection(url, username, password)){

Statement statement = conn.createStatement();

int rows = statement.executeUpdate("UPDATE Products SET Price = Price - 5000");

System.out.printf("Updated %d rows", rows);

}

}

catch(Exception ex){

System.out.println("Connection failed...");

System.out.println(ex);

}

**Удаление**

Удалим один объект из таблицы с помощью команды DELETE:

try{

String url = "jdbc:mysql://localhost/store?serverTimezone=Europe/Moscow&useSSL=false";

String username = "root";

String password = "password";

Class.forName("com.mysql.cj.jdbc.Driver").getDeclaredConstructor().newInstance();

try (Connection conn = DriverManager.getConnection(url, username, password)){

Statement statement = 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...");

System.out.println(ex);

}