



Пакет Java Database Connectivity

Лекция 9

Цели занятия

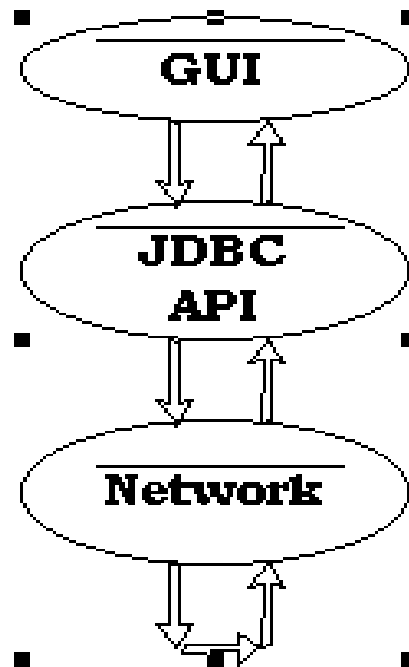
- Дать определение прикладного интерфейса JDBC API
- Рассмотреть драйверы JDBC
- Описать двухзвенную модель "Клиент–Сервер"
- Описать трёхзвенную модель "Клиент–Сервер–БД"
- Использовать пакет JDBC для доступа к базе данных
- Устанавливать соединение с базой данных
- Создавать и выполнять команды SQL
- Описать интерфейс ResultSetMetaData
- Понимать важность обеспечения безопасности базы данных

Прикладной интерфейс JDBC API

- **JDBC API** - Java Database Connectivity Application Programming Interface – это набор спецификаций, которые определяют, каким образом Java-программы могут обмениваться информацией с базой данных.
- Этот интерфейс определяет, как приложение открывает соединение, ведёт обмен информацией с базой данных, выполняет команды SQL и извлекает результаты.
- Многие концепции JDBC API позаимствованы из других источников, в частности, из Microsoft ODBC (Open Database Connectivity).

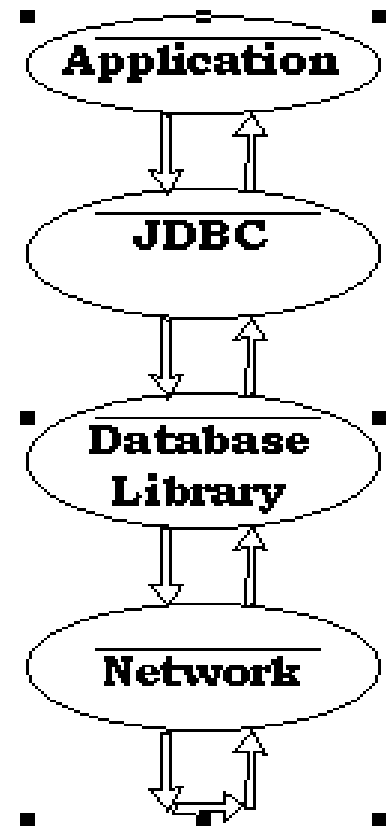
JDBC API

- Функциональная схема прикладного программного интерфейса JDBC API.



Драйверы JDBC

- Даёт гарантию того, что запрос, выполняемый приложением к базе данных, представляется на том языке, который понимает эта база данных.
- Драйвер JDBC принимает запросы от клиента, преобразует их в формат, который понимает база данных, а затем передаёт их в базу данных.
- Затем драйвер JDBC принимает ответ, преобразует его в формат данных языка Java и передаёт результат в клиентское приложение.
- Представляет собой **библиотеку Java**, которая транслирует JDBC-запросы непосредственно в протокол конкретной базы данных.



Программные продукты JDBC

- Программные продукты JDBC содержат следующие три компонента:
 - ❖ Пакет java.sql
 - ❖ Тестовый комплект
 - ❖ Промежуточное ПО, так называемый "мост" JDBC–ODBC

Пакет **java.sql**

- JDBC API, определяет набор интерфейсов и классов для обмена информацией с базой данных, который содержится в пакете **java.sql**
- Пакет **java.sql** состоит из следующих интерфейсов:
 - CallableStatement
 - Connection
 - DatabaseMetaData
 - Driver
 - PreparedStatement
 - ResultSet
 - ResultSetMetaData
 - Statement

Пакет java.sql

CallableStatement	Содержит методы, используемые для выполнения хранимых процедур SQL.
Connection	Осуществляет управление и контроль сеансов подключения к базе данных.
DatabaseMetaData	Предоставляет информацию о базе данных, такую как номер версии, имена таблиц и поддерживаемые функции.
Driver	Создает объекты Connection.
PreparedStatement	Выполняет предварительно откомпилированные команды SQL.
ResultSet	Предоставляет методы для извлечения данных, которые возвращает команда SQL.
ResultSetMetaData	Содержит мета-данные, связанные с последним используемым объектом ResultSet.
Statement	Выполняет команды SQL и извлекает данные из итогового набора ResultSet.

Пакет `java.sql`

- Пакет **`java.sql`** определяет следующие исключения:
 - ❖ `DataTruncation`
 - ❖ `SQLException`
 - ❖ `SQLWarning`
- Класс **`SQLException`** является основой всех исключительных ситуаций JDBC.
- Класс **`SQLWarning`** подобен классу `SQLException`, однако, он предполагает не критические ошибки и не приводит к "выбросу" из программы (можно запросить сообщения `SQLWarning` при помощи метода `getWarnings` классов `Connection`, `ResultSet` и `Statement`).
- Класс **`DataTruncation`** - это особый тип `SQLWarning`. Он описывается вместе с экземплярами `SQLWarning` и показывает, была ли потеряна информация в процессе выполнения операций чтения/записи.

Тестовый комплект для драйвера JDBC

- Проверяет функциональность драйверов JDBC.
- Гарантирует, что все классы и методы, определённые в JDBC API, действительно реализованы.
- После того, как драйвер проходит все тесты, конкретный тестовый комплект может быть обозначен, как JDBC COMPLAINT.

Мост JDBC–ODBC

- **JDBC–ODBC** - это драйвер JDBC, который позволяет Java-приложениям использовать драйвер ODBC для обмена информацией с базой данных.
- **Главная задача драйвера** – позволить разработчикам писать JDBC-приложения, не дожидаясь выпуска соответствующих ODBC-драйверов для конкретных баз данных.
- Он является составной частью пакета JDBC.

Программные продукты JDBC

- Для работы с JDBC API необходимо:
 - ❖ Пакет разработчика Java Development ToolKit (JDK)
 - ❖ SQL-совместимая база данных
 - ❖ Драйвер JDBC для этой базы данных

Условия проектирования JDBC

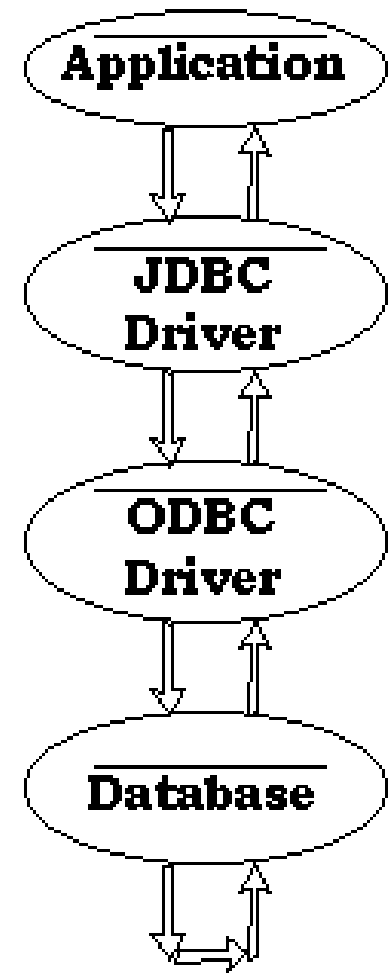
- Драйвер JDBC очень хорошо соответствует архитектуре разнообразных моделей клиент/сервер.
- Четыре типа драйверов JDBC:
 - ❖ Мост JDBC–ODBC
 - ❖ Собственный прикладной интерфейс API Java (Драйвер Native-API-Partly-Java)
 - ❖ JDBC Network (Драйвер JDBC-Net-All-Java)
 - ❖ Собственный протокол (Драйвер Native-Protocol-All-Java)

Мост JDBC–ODBC

- **Мост JDBC–ODBC** предоставляет и поддерживает компания JavaSoft.
- Драйвер ODBC является единственным драйвером, который может использоваться с многочисленными базами данных.
- Интерфейс ODBC остаётся постоянным, вне зависимости от используемой базы данных.
- После того, как JDBC передаёт запрос в драйвер ODBC, именно этот драйвер ODBC берёт на себя обязанности по обеспечению обмена информацией с базой данных.
- Недостатком моста JDBC–ODBC является то, что он добавляет ещё один уровень сложности в программу и может существенно затруднить поиск и устранение ошибок в ПО.

Мост JDBC–ODBC

- Приложение отправляет запрос через драйвер JDBC.
- Драйвер JDBC транслирует этот запрос в вызов ODBC.
- Драйвер ODBC еще раз преобразует запрос и передает его в интерфейс базы данных.
- В полном цикле работы моста JDBC-ODBC выполняются двунаправленные преобразования (трансляции) для каждого запроса и для каждого возвращаемого результата.

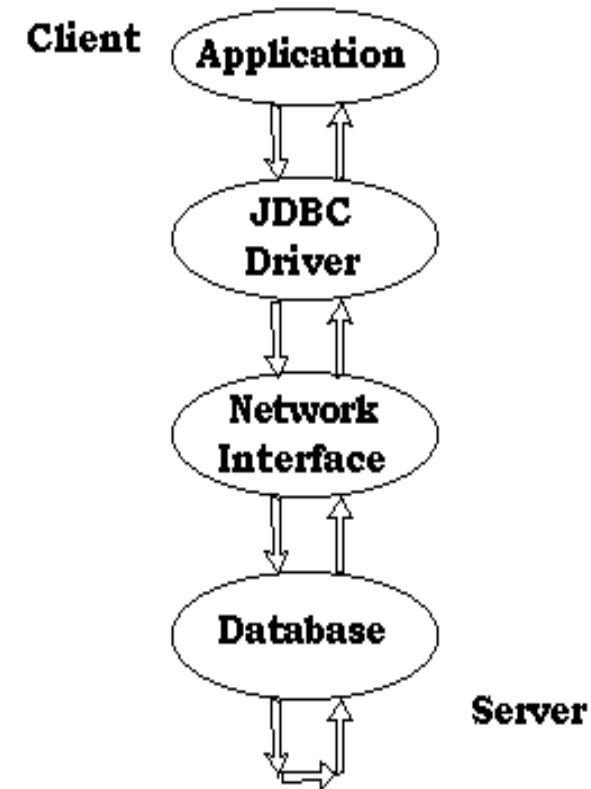


Мост JDBC—ODBC

- ODBC (Open DataBase Connectivity) наиболее популярен при разработке приложений для доступа к БД. Этот интерфейс "понимают" почти все СУБД на всех платформах. Возникает вопрос: а почему бы не использовать ODBC в языке Java?
 - ❖ ODBC использует интерфейс, наиболее родной для языка C и не совсем подходящий для Java. Вызовы C-кода из языка Java страдают плохой защищенностью, гибкостью и переносимостью приложений.
 - ❖ Непосредственная трансляция интерфейса ODBC API в Java API нежелательна по той причине, что в языке Java нет адресной арифметики, а ODBC активно ее использует в виде указателей.
 - ❖ ODBC сложнее для изучения. Он перемешивает в себе одновременно и простые, и сложные возможности и требует сложных приемов даже для выполнения простейших запросов. В то же время в JDBC все простое делается просто, но и сложные возможности остаются доступными.
 - ❖ JDBC отвечает концепции "истинной Java". Когда используется ODBC, необходимо на каждую клиентскую машину ставить odbc-драйвер. Если JDBC-драйвер написан целиком на языке Java, то приложение, использующее его, лишено такого недостатка и, кроме того, обладает переносимостью и защищенностью, присущей языку Java.
- JDBC - это "родной" для Java интерфейс к базовым абстракциям и концепциям языка SQL.

Двухзвенная модель Клиент–Сервер

- По умолчанию архитектурой любой среды Клиент–Сервер является двухзвенная система.
- Клиент является первым звеном, а сервер – вторым.
- В двухзвенной среде JDBC приложение базы данных является клиентом, а сама СУБД является сервером.
- Клиент обменивается информацией непосредственно с сервером.



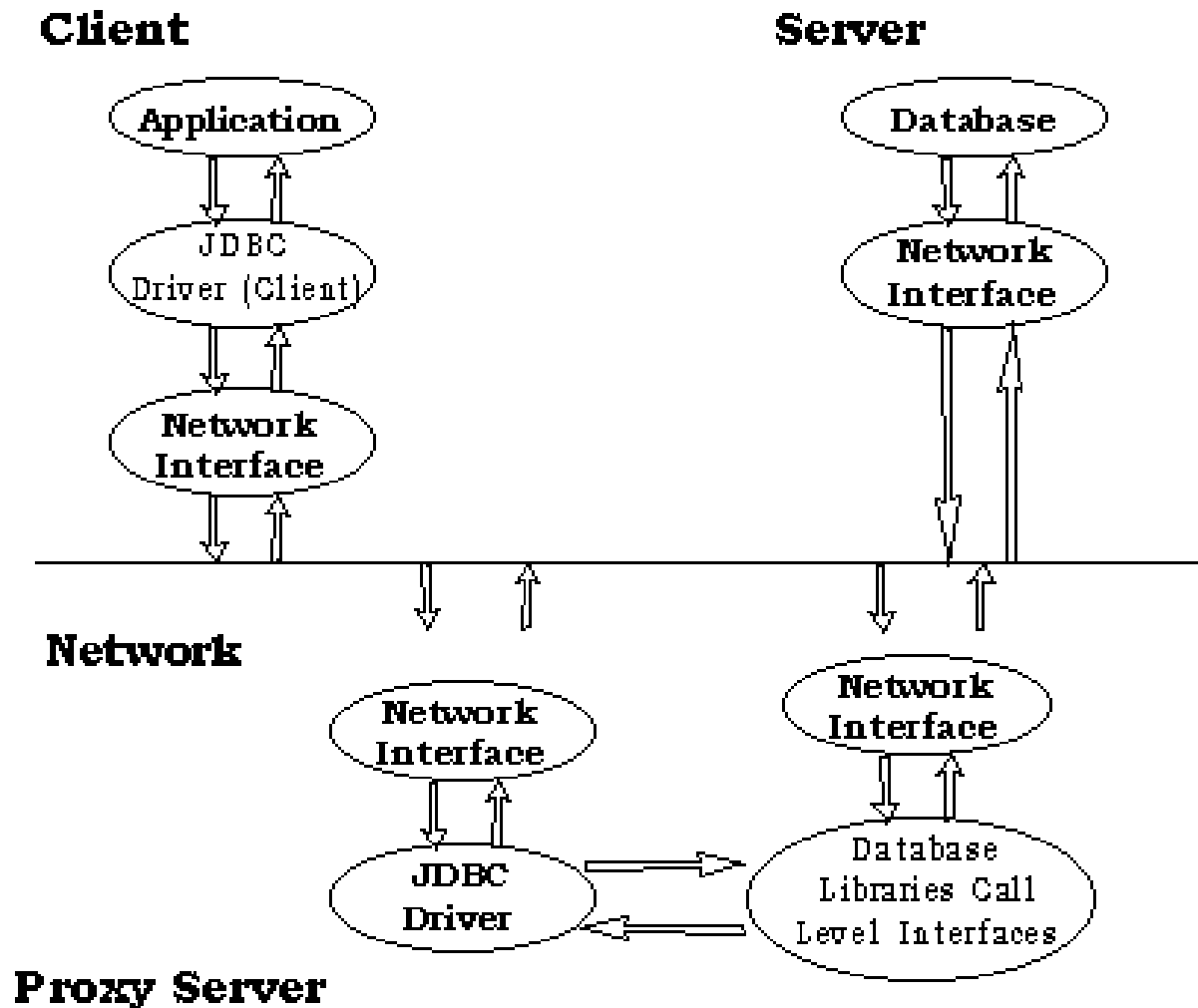
Двухзвенная модель Клиент–Сервер

- Преимущества использования двухзвенной системы базы данных:
 - ❖ Это система, наименее сложная в реализации.
 - ❖ Эта архитектура поддерживает и обслуживает постоянное соединение между клиентом и базой данных.
 - ❖ Эта система обычно работает быстрее, чем реализация трёхзвенной модели.
- Недостатки использования этой системы:
 - ❖ Большинство драйверов требуют, чтобы штатные (native) библиотеки были загружены на машине клиента.
 - ❖ Апплеты могут установить открытое (незащищённое) соединение с сервером, с которого они загружаются.

Трёхзвенная модель Клиент–Сервер

- В этой системе третий сервер предназначен для обработки запросов от клиента, а после обработки он передаёт запросы серверу базы данных.
- Этот третий сервер работает, как **прокси** (посредник) для всех клиентских запросов.
- Эта модель обладает преимуществом, заключающимся в отделении сервера базы данных от web-сервера.
- В такой среде драйвер преобразует запрос в соответствии с сетевым протоколом, а затем отправляет запрос через прокси-сервер.

Трёхзвенная модель Клиент–Сервер

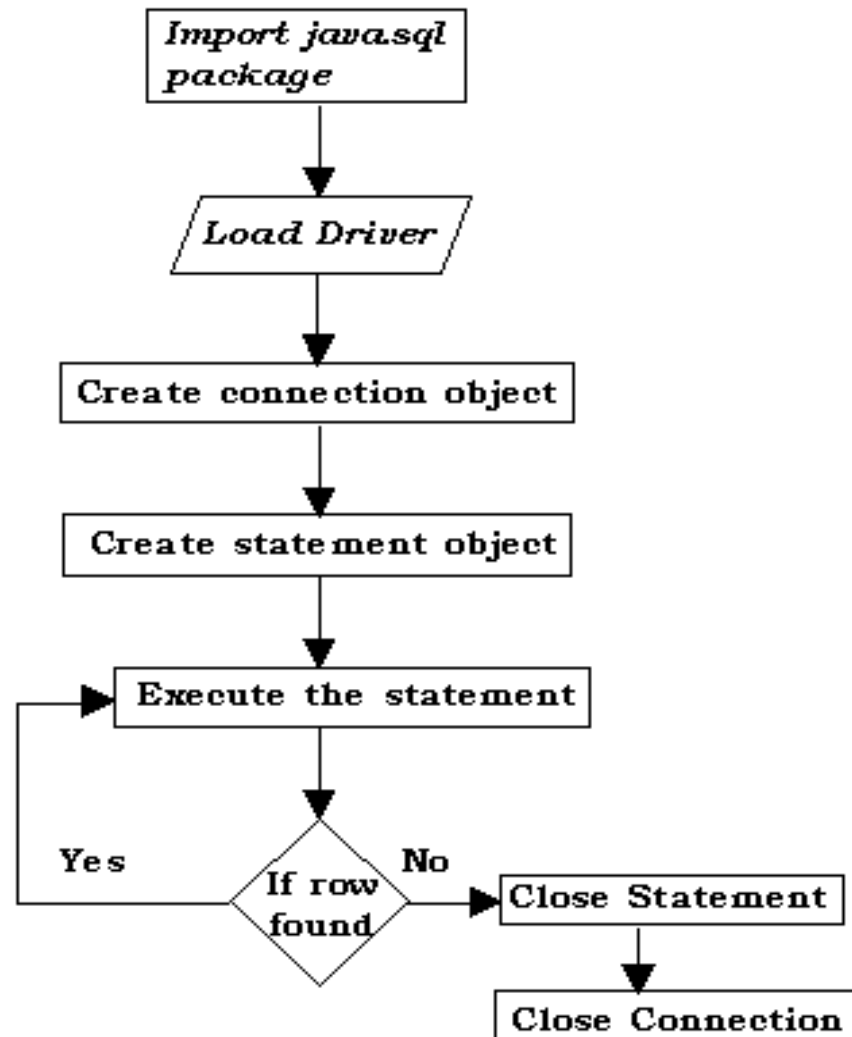


Использование JDBC для доступа к БД

- Существуют семь этапов (или шагов) при использовании JDBC для доступа к базе данных:
 - Импортирование пакета `java.sql`
 - Загрузка и регистрация драйвера
 - Установление соединения с сервером базы данных
 - Создание команды (инструкции)
 - Выполнение команды
 - Получение результатов
 - Завершение команды и закрытие соединения

Использование JDBC для доступа к БД

- На рисунке ниже схематически показаны эти шаги:



Установка и настройка соединения

- Класс **java.sql.DriverManager** предоставляет методы для загрузки драйверов. В этот класс включены следующие методы:
 - ❖ `getDrivers()`
 - ❖ `getConnection()`
 - ❖ `registerDriver()`
 - ❖ `deregisterDriver()`
 - ❖ `getLoginTimeout()`
 - ❖ `setLoginTimeout()`
 - ❖ `getLogStream()`
 - ❖ `setLogStream()`

Отслеживание доступных драйверов

- Класс **DriverManager** является уровнем управления в JDBC и находится между пользователем и драйверами.
- Метод `DriverManager.getConnection` устанавливает соединение с БД.

`Class.forName("acme.db.Driver");`

- Только после этого драйвер появляется в списке зарегистрированных драйверов в классе **DriverManager** и становится возможным открывать соединения.

```
public static void main(String[] args) {  
    try {  
        Class.forName("com.mysql.jdbc.Driver");  
        System.out.println("Driver loading success!");  
    } catch (ClassNotFoundException e) {  
        e.printStackTrace();  
    }  
}
```


Соединение

- Объект **Connection** представляет собой соединение с БД. Сессия соединения включает в себя выполняемые SQL-запросы и возвращаемые через соединение результаты.
- Стандартный способ получения соединения - это вызов метода **DriverManager.getConnection**. Этому методу передается строка, содержащая "URL".
- Класс **DriverManager**, представляющий собой уровень управления в JDBC, пытается найти драйвер, который может соединиться к БД с помощью данного URL.
- Следующий код демонстрирует открытие соединения с БД, находящейся по URL "jdbc:odbc:wombat", с именем пользователя "boy" и паролем "java":

```
String url = "jdbc:odbc:wombat";  
Connection con = DriverManager.getConnection(url, "boy",  
"java");
```

- **URL** (Uniform Resource Locator) представляет собой информацию для адресации ресурса в Интернет. Другими словами, это адрес ресурса.
- Первая часть URL задает протокол, используемый для доступа к информации, и всегда заканчивается знаком ":",
".".
- Среди протоколов наиболее популярны
 - ❖ "ftp" ("file transfer protocol" - протокол передачи файлов),
 - ❖ "http" ("hypertext transfer protocol" - протокол передачи гипертекста).

ftp://javasoft.com/docs/JDK-1_apidocs.zip

<http://java.sun.com/products/JDK/CurrentRelease>

JDBC-URL

Стандартный синтаксис JDBC URL имеет **три части, разделенных двоеточием:**

`jdbc:<subprotocol>:<subname>`

- ❖ **jdbc** - протокола. Протокол, используемый в JDBC URL - всегда jdbc.
- ❖ **<subprotocol>** (подпротокол) - это имя драйвера или имя механизма соединения с БД.
- ❖ **<subname>** (подимя) - это идентификатор БД.

```
String url = "jdbc:mysql://localhost/bookstore";  
String name = "root";  
String password = "";  
con = DriverManager.getConnection(url, name, password);
```

SQL-запросы

- После того как установлено соединение, оно используется для выполнения SQL-запросов к БД.
- В JDBC есть классы для отправки SQL-запросов в БД и методы в интерфейсе **Connection** создают экземпляры этих классов.
 - ❖ **Statement** - создается методом **createStatement**. Объект Statement используется при простых SQL-запросах.

```
private Connection con;  
con = DriverManager.getConnection(url, name, password);  
Statement st = con.createStatement();
```

Создание и выполнение команд SQL

- Команда SQL занимает центральное положение в любом JDBC.
- Запросы (**Queries**) являются одной из самых важных форм команд SQL.
- В JDBC все запросы возвращаются в виде объектов **ResultSet**.
- Наиболее эффективным способом выполнения запроса является использование метода **Statement.executeQuery()**.

Инструкция SQL для создания таблицы

1. Создайте объект класса **Statement** при помощи объекта подключения:

```
Statement stmt = con.createStatement();
```

2. Необходимо выполнить следующую команду SQL, чтобы создать таблицу *MyData*:

```
create table MyData (  
    programmer varchar (32),  
    day char (3),  
    cups integer,  
    variety varchar (20));
```

3. Для этого напишите следующий Java-код:

```
stmt.execute( "create table MyData (" +  
    "programmer varchar (32)," +  
    "day char (3)," +  
    "cups integer);" +  
    );
```

Объекты **ResultSet** и **ResultSetMetaData**

- Результат запроса возвращается в виде таблицы данных, состоящей из строк и столбцов.
- Интерфейс **ResultSet** используется для доступа к этим данным.
- Результаты запроса возвращаются, как объекты **ResultSet**, которые, в свою очередь, предоставляют построчный доступ к табличным данным (то есть, за одно обращение считывается только одна строка).
- Интерфейс **ResultSetMetaData** предоставляет константы и методы, используемые для получения информации по объекту **ResultSet**.

Извлечение информации из базы данных

- Для получения информации из базы данных используется SQL-команда **select** с помощью Java-метода **Statement.executeQuery()**, который в качестве результата возвращает строки данных в объекте **ResultSet**.
- Результат выводится построчно при помощи методов **ResultSet.next()** и **ResultSet.getXXX()**

```
String name = result.getString("programmer");  
int cups = result.getInt("cups");
```


Объекты **ResultSet** и **ResultSetMetaData**

- Результат запроса возвращается в виде таблицы данных, состоящей из строк и столбцов.
- Интерфейс **ResultSet** используется для доступа к этим данным.
- Результаты запроса возвращаются, как объекты **ResultSet**, которые, в свою очередь, предоставляют построчный доступ к табличным данным (то есть, за одно обращение считывается только одна строка).
- Интерфейс **ResultSetMetaData** предоставляет константы и методы, используемые для получения информации по объекту **ResultSet**.

```
package javaapplication2;
```

```
import java.sql.*;
```

```
public class BookStore {  
    private Connection con;  
    public BookStore() {  
        String url = "jdbc:mysql://localhost/bookstore";  
        String name = "root";  
        String password = "";  
        try {  
            con = DriverManager.getConnection(url, name, password);  
            System.out.println("Connected.");  
            Statement st = con.createStatement();  
            String query = "select * from books";  
            ResultSet rs = st.executeQuery(query);  
            printResults(rs);  
            System.out.println("Disconnected.");  
            con.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

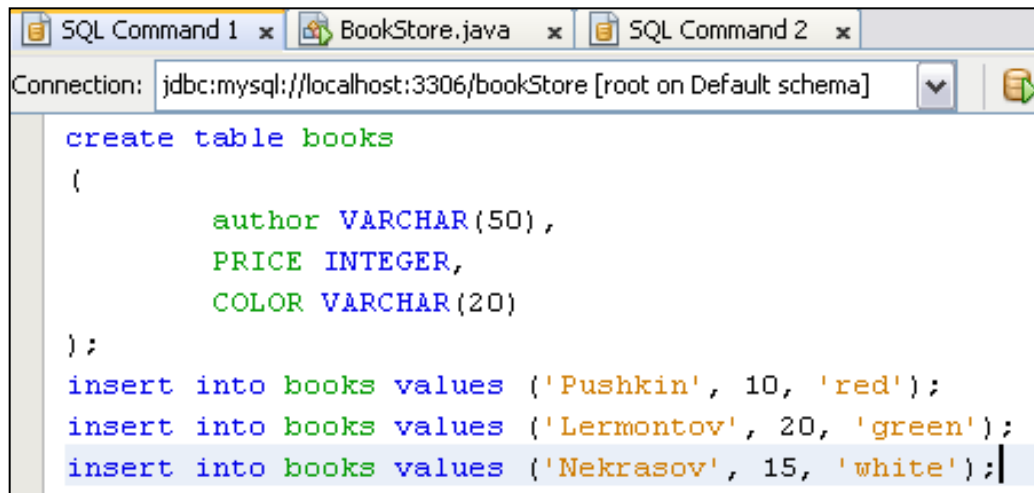
```
private void printResults(ResultSet rs) throws SQLException {  
    String author, color;  
    int price;  
    while (rs.next()) {  
        author = rs.getString("author");  
        color = rs.getString("color");  
        price = rs.getInt("price");  
        System.out.println("*****");  
        System.out.println("Author: " + author);  
        System.out.println("Price: " + price);  
        System.out.println("color: " + color);  
    }  
}
```

```
public static void main(String[] args) {  
    try {  
        Class.forName("com.mysql.jdbc.Driver");  
        System.out.println("Driver loading  
success!");  
    } catch (ClassNotFoundException e) {  
        e.printStackTrace();  
    }  
    BookStore bookStore = new BookStore();  
}
```

```
create table books
```

```
(  
    author VARCHAR(50),  
    PRICE INTEGER,  
    COLOR VARCHAR(20)
```

```
);  
insert into books values ('Pushkin', 10, 'red');  
insert into books values ('Lermontov', 20, 'green');  
insert into books values ('Nekrasov', 15, 'white');
```

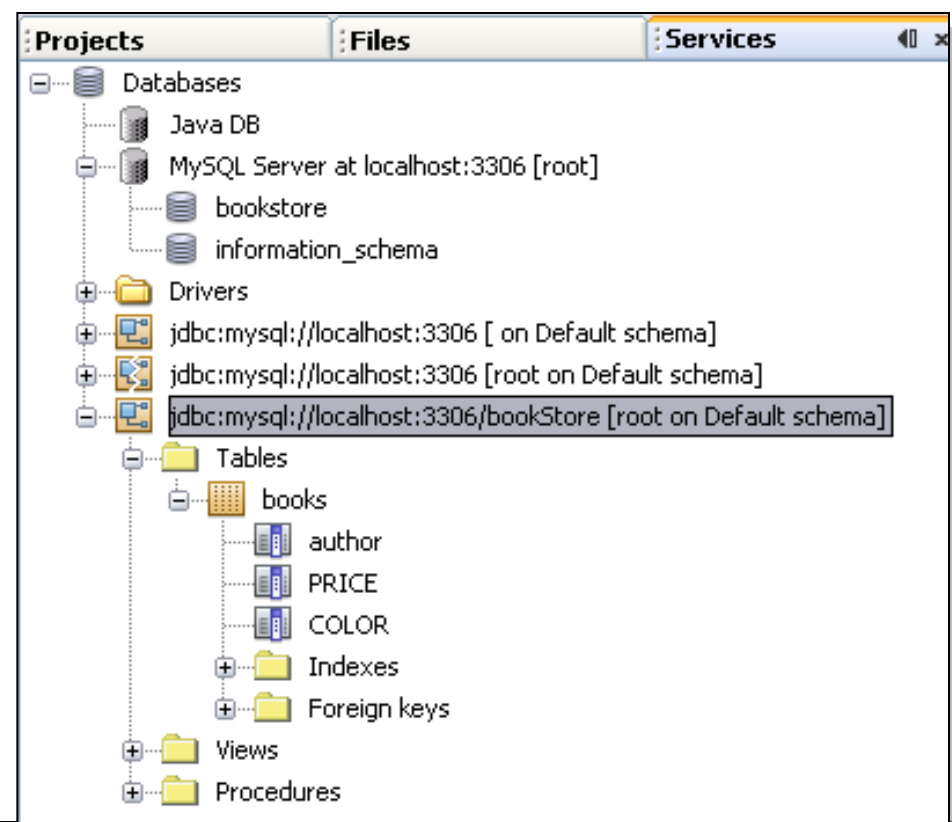
A screenshot of a Java IDE window with three tabs: 'SQL Command 1', 'BookStore.java', and 'SQL Command 2'. The 'SQL Command 1' tab is active, showing a text area with SQL code. The connection string is 'jdbc:mysql://localhost:3306/bookStore [root on Default schema]'. The code in the text area is: 'create table books', '(', 'author VARCHAR(50),', 'PRICE INTEGER,', 'COLOR VARCHAR(20)', ');', 'insert into books values ('Pushkin', 10, 'red');', 'insert into books values ('Lermontov', 20, 'green');', and 'insert into books values ('Nekrasov', 15, 'white');'.

```
SQL Command 1 x BookStore.java x SQL Command 2 x  
Connection: jdbc:mysql://localhost:3306/bookStore [root on Default schema]  
create table books  
(  
    author VARCHAR(50),  
    PRICE INTEGER,  
    COLOR VARCHAR(20)  
);  
insert into books values ('Pushkin', 10, 'red');  
insert into books values ('Lermontov', 20, 'green');  
insert into books values ('Nekrasov', 15, 'white');
```

SQL Command 1 x BookStore.java x SQL Command 2 x

Connection: jdbc:mysql://localhost:3306/bookStore [root on Default schema]

```
create table books
(
    author VARCHAR(50),
    PRICE INTEGER,
    COLOR VARCHAR(20)
);
insert into books values ('Pushkin', 10, 'red');
insert into books values ('Lermontov', 20, 'green');
insert into books values ('Nekrasov', 15, 'white');
```



Результат

```
Driver loading success!  
Connected.  
*****  
Author: Pushkin  
Price: 10  
color: red  
*****  
Author: Lermontov  
Price: 20  
color: green  
*****  
Author: Nekrasov|  
Price: 15  
color: white  
Disconnected.  
BUILD SUCCESSFUL (total time: 2 seconds)
```

Где и что скачать

- Для того, чтобы начать работу с БД нам нужно иметь при себе драйвер.
- Работа с сервером баз данных MySQL . Сам сервер можно взять отсюда: <http://dev.mysql.com/downloads>.
- Скачать к нему драйвер можно здесь:
<http://dev.mysql.com/downloads>.
 - ❖ В разделе MySQL Connector/J выбираем последнюю версию драйвера. В скачанном архиве есть документация, и, что самое главное, jar-архив `mysql-connector-java-5.1.5-bin.jar`.
 - ❖ Он должен быть указан в `classpath`, лежать в каталоге с проектом или добавлен в проект с помощью возможностей IDE(в Еклипсе, например, добавлять библиотеки можно следующим образом: *Import->General->Archive File*).
- Книга «**Руководство JDBC**»
<http://dmivic.chat.ru/JDBC/introTOC.doc.html>
- www.javatalks.ru