УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационной безопасности

Кафедра инфокоммуникационных технологий

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ
Часть 2

Контрольная работа
Соединение с базами данных в Java

Минск 2023

Макейчик Е.Г.

Содержание

Контрольная работа
Соединение с базами данных в Java

JDBC (Java DataBase Connectivity)

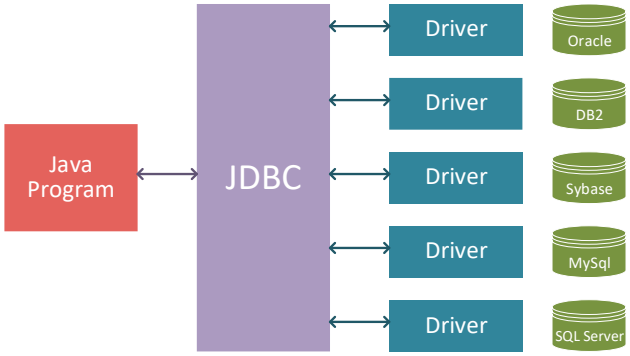
DBC (Java DataBase Connectivity – соединение с базами данных на Java) – платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, входящего в состав Java SE.

Схема подключения к базе данных и драйвера

Драйвера

JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL.

Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, за который драйвер отвечает.



Алгоритм работы с БД

Шаг 1. Загрузка драйвера в память (до Java 8).

```
Class.forName("com.mysql.jdbc.Driver");
```

Шаг 2. Установка соединения с БД.

```
Connection cn = DriverManager.getConnection(
        url: "jdbc:mysql://localhost/my_db", user: "login", password: "password");
```

Шаг 3. Создание объекта для передачи запросов.

Шаг 4. Закрытие всех соединений.

Имена драйверов и url для различных БД

БД	Имя драйвера	Пример URL
Oracle 8i	oracle.jdbc.driver.OracleDriver	dbc:oracle:thin:@localhost:1521:scorpian
Oracle 9i	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@localhost:1521:scorpian
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://localhost:3306/scorpian
Microsoft Access	sun.jdbc.odbc.JdbcOdbcDriver	jdbc:odbc: Driver={Microsoft Access Driver (*.mdb)}
Sybase	com.sybase.jdbc2.jdbc.SybDriver	jdbc:sybase:Tds:scorpian:2638
MS SQL	com.microsoft.jdbc.sqlserver.SQLServerDriver	jdbc:microsoft:sqlserver://localhost:1433
IBM DB2	com.ibm.db2.jdbc.net.DB2Connection	jdbc:db2://localhost:6789/scorpian
H2	org.h2.Driver	jdbc:h2:tcp://localhost/~/test

Работа с БД с помощью JDBC

Утверждения (Statements)

Взаимодействовать с БД мы можем с помощью трех интерфейсов, которые реализуются каждым драйвером:

1. Statement – этот интерфейс используется для доступа к БД для общих целей. Он крайне полезен, когда используются статические SQL-выражения во время работы программы. Этот интерфейс не принимает никаких параметров.

2. PreparedStatement – этот интерфейс может принимать параметры во время работы программы.

3. CallableStatement – этот интерфейс становится полезным в случае, когда нужно получить доступ к различным процедурам БД. Он также может принимать параметры во время работы программы.

Интерфейс Statement

Создание объекта происходит следующим образом.

```
Statement statement = connection.createStatement();
```

После этого можно использовать экземпляр statement для выполнения SQL-запросов. Для этой цели интерфейс Statement имеет три метода, которые реализуются каждой конкретной реализацией JDBC драйвера:

- boolean execute(String SQL) – позволяет выполнить Statement, когда неизвестно заранее, является SQL-строка запросом или обновлением. Метод возвращает true, если команда создала результирующий набор.
- int executeUpdate(String SQL) – используется для выполнения обновлений. Он возвращает количество обновленных строк, для выполнения операторов INSERT, UPDATE или DELETE.
- ResultSet executeQuery(String SQL) – используется для выполнения запросов (SELECT). Он возвращает для обработки результирующий набор.

Пример создания таблицы представлен в коде ниже.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

import static jdbc.ConnectionData.PASSWORD;
import static jdbc.ConnectionData.URL;
import static jdbc.ConnectionData.USER;

public class CreatingTable {
    private static final String CREATE_TABLE_QUERY =
            "CREATE TABLE users "
                    + "(id INT(5) NOT NULL AUTO_INCREMENT,"
                    + " username VARCHAR(50), "
                    + "PRIMARY KEY(id));";

    public static void main(String[] args) {
        try (Connection connection =
                     DriverManager.getConnection(URL, USER, PASSWORD);
             Statement statement = connection.createStatement()) {
            statement.executeUpdate( sql: CREATE_TABLE_QUERY);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```java
public class ConnectionData {
    public static final String DRIVER = "com.mysql.jdbc.Driver";
    public static final String DB = "catalog";
    public static final String URL = "jdbc:mysql://localhost:3306/" + DB;
    public static final String USER = "root";
    public static final String PASSWORD = "admin";
}
```

Интерфейс ResultSet

Этот интерфейс представляет результирующий набор базы данных. Он обеспечивает приложению построчный доступ к результатам запросов в базе данных.

Во время обработки запроса ResultSet поддерживает указатель на текущую обрабатываемую строку. Приложение последовательно перемещается по результатам, пока они не будут все обработаны или не будет закрыт ResultSet.

Доступ к данным ResultSet обеспечивает посредством набора get-методов, которые организуют доступ к колонкам текущей строки. Метод ResultSet.next() используется для перемещения к следующей строке ResultSet, делая ее текущей.

Основные методы интерфейса ResultSet:

- public boolean absolute(int row) throws SQLException – метод перемещает курсор на заданное число строк от начала, если число положительно, и от конца – если отрицательно.
- public void afterLast() throws SQLException – этот метод перемещает курсор в конец результирующего набора за последнюю строку.
- public void beforeFirst() throws SQLException – этот метод перемещает курсор в начало результирующего набора перед первой строкой.
- public void deleteRow() throws SQLException – удаляет текущую строку из результирующего набора и базы данных.
- public ResultSetMetaData getMetaData() throws SQLException – предоставляет объект метаданных для данного ResultSet. Класс ResultSetMetaData содержит информацию о результирующей таблице, такую как количество столбцов, их заголовок и т.д.
- public int getRow() throws SQLException – возвращает номер текущей строки.
- public Statement getStatement() throws SQLException – возвращает экземпляр Statement, который произвел данный результирующий набор.
- public boolean next() throws SQLException, public boolean previous() throws SQLException – эти методы позволяют переместиться в результирующем наборе на одну строку вперед или назад. Во вновь созданном результирующем наборе курсор устанавливается перед первой строкой, поэтому первое обращение к методу next() влечет позиционирование на первую строку. Эти методы возвращают true, если остается строка для дальнейшего перемещения. Если строк для обработки больше нет, возвращается false.
- public void close() throws SQLException – осуществляет немедленное закрытие ResultSet вручную. Обычно этого не требуется, так как закрытие Statement, связанного с ResultSet, автоматически закрывает ResultSet.

Результирующий набор данных ResultSet можно не закрывать. Это делается автоматически родительским объектом Statement, когда он закрывается, начинает выполняться повторно или используется для извлечения следующего результата в последовательности нескольких результатов.

В коде ниже приведен пример использования интерфейса ResultSet.

```java
public class RetrieveData {
    private static final String SELECT_QUERY = "SELECT * FROM users;";

    public static void main(String[] args) {
        try (Connection connection =
                     DriverManager.getConnection(URL, USER, PASSWORD);
             Statement statement = connection.createStatement()) {

            ResultSet resultSet = statement.executeQuery( sql: SELECT_QUERY);
            System.out.printf("%-20s%s%n", "id", "username");
            System.out.println("------------------------------");
            while (resultSet.next()) {
                int id = resultSet.getInt( columnLabel: "id");
                String name = resultSet.getString( columnLabel: "username");
                System.out.printf("%-20d%s%n", id, name);
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Пакетное выполнение запросов

Для выполнения набора из нескольких запросов на обновление данных в интерфейс Statement были добавлены следующие методы.

```
void addBatch(String)
int[] executeBatch()
```

Пакетное выполнение запросов уменьшает трафик между клиентом и СУБД и может привести к существенному повышению производительности.

В коде ниже приведен пример пакетного выполнения запросов.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import static jdbc.ConnectionData.PASSWORD;
import static jdbc.ConnectionData.URL;
import static jdbc.ConnectionData.USER;

public class InsertBatchData {
    public static void main(String[] args) {
        try (Connection connection =
                     DriverManager.getConnection(URL, USER, PASSWORD);
             Statement statement = connection.createStatement()) {

            statement.addBatch( sql: "INSERT INTO users (username) VALUES ('Sidorov')");
            statement.addBatch( sql: "INSERT INTO users (username) VALUES ('Petrov')");
            statement.addBatch( sql: "INSERT INTO users (username) VALUES ('Kozlov')");

            statement.executeBatch();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Интерфейс PreparedStatement

Особенностью SQL-выражений в PreparedStatement является то, что они могут иметь параметры. Параметризованное выражение содержит знаки вопроса в своем тексте.

```
SELECT name FROM persons WHERE age=?
```

Перед выполнением запроса значение каждого вопросительного знака явно устанавливается методами set-методами.

```
ps.setInt(1, 30);
```

Использование PreparedStatement приводит к более быстрому выполнению запросов при их многократном вызове с различными параметрами.

В коде ниже приведены примеры использования интерфейса PreparedStatement.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import static jdbc.ConnectionData.PASSWORD;
import static jdbc.ConnectionData.URL;
import static jdbc.ConnectionData.USER;

public class RetrieveDataPreparedStatement {
    private static final String SELECT_QUERY =
            "SELECT * FROM users WHERE id>? AND username LIKE ?";

    public static void main(String[] args) {
        try (Connection connection =
                     DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement preparedStatement =
                     connection.prepareStatement( sql: SELECT_QUERY)) {

            preparedStatement.setInt( parameterIndex: 1,  x: 2);
            preparedStatement.setString( parameterIndex: 2,  x: "P%");
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                System.out.printf("%d%23s%n", resultSet.getInt( columnLabel: "id"),
                        resultSet.getString( columnLabel: "username"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import static jdbc.ConnectionData.PASSWORD;
import static jdbc.ConnectionData.URL;
import static jdbc.ConnectionData.USER;

public class InsertDataPreparedStatement {
    private static final String INSERT_QUERY =
            "INSERT INTO users (username) VALUES (?)";

    public static void main(String[] args) {
        try (Connection connection =
                     DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement preparedStatement =
                     connection.prepareStatement( sql: INSERT_QUERY)) {

            preparedStatement.setString( parameterIndex: 1,  x: "Misha");
            preparedStatement.addBatch();

            preparedStatement.setString( parameterIndex: 1,  x: "Grisha");
            preparedStatement.addBatch();
            preparedStatement.executeBatch();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Использование properties файлов

Содержимое database.properties файла приведено ниже.

```
db.driver = com.mysql.jdbc.Driver
db.name = catalog
db.url = jdbc:mysql://localhost:3306/
db.user = root
db.password = admin
```

В коде ниже приведен пример использования ResourceBundle для чтения данных для аутентификации.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.ResourceBundle;

public class ConnectorDB {
    public static Connection getConnection() throws SQLException {
        ResourceBundle resource = ResourceBundle.getBundle( baseName: "database");
        String url = resource.getString( key: "db.url");
        String user = resource.getString( key: "db.user");
        String pass = resource.getString( key: "db.password");
        String dbName = resource.getString( key: "db.name");

        return DriverManager.getConnection( url: url + dbName, user, pass);
    }
}
```

Data Access Object (DAO)

В программном обеспечении Data Access Object (DAO) – это объект, который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. DAO может использоваться для разных видов доступа к БД (JDBC, JPA).

При проектировании информационной системы выявляются некоторые слои, которые отвечают за взаимодействие различных модулей системы. Соединение с базой данных является одной из важнейшей составляющей приложения. Всегда выделяется часть кода, модуль, отвечающий за передачу запросов в БД и обработку полученных от нее ответов. В общем случае, определение Data Access Object описывает его как прослойку между БД и системой. DAO абстрагирует сущности системы и делает их отображение на БД, определяет общие методы использования соединения, его получение, закрытие и (или) возвращение в Connection Pool.

В коде ниже приведены примеры использования абстрактного класса DAO.

```
import java.util.List;

public abstract class AbstractDAO<K extends Number, T> {
    public abstract List<T> findAll();
    public abstract T findEntityById(K id);
    public abstract boolean delete(K id);
    public abstract boolean delete(T entity);
    public abstract boolean create(T entity);
    public abstract T update(T entity);
}
```

```
public class MainDAO {
    public static void main(String[] args) {
        UserDAO userDAO = new UserDAO();
        System.out.println(userDAO.findAll());
        System.out.println(userDAO.findEntityById( id: 1));
    }
}
```

```java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class UserDAO extends AbstractDAO<Integer, User> {
    public static final String SQL_SELECT_ALL_USERS = "SELECT * FROM users";
    public static final String SQL_SELECT_USER_ID =
            "SELECT * FROM users WHERE id=?";
    public List<User> findAll() {
        List<User> users = new ArrayList<>();
        try (Connection connection = ConnectorDB.getConnection();
             Statement statement = connection.createStatement()) {
            ResultSet rs = statement.executeQuery(SQL_SELECT_ALL_USERS);
            while (rs.next()) {
                int id = rs.getInt( columnIndex: 1);
                String name = rs.getString( columnIndex: 2);
                users.add(new User( id: id, name));
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return users;
    }
    public User findEntityById(Integer id) {
        User user = null;
        try (Connection connection = ConnectorDB.getConnection();
             PreparedStatement statement =
                     connection.prepareStatement(SQL_SELECT_USER_ID)) {
            statement.setInt( parameterIndex: 1,  x: id);
            ResultSet rs = statement.executeQuery();
            if (rs.next()) {
                String name = rs.getString( columnIndex: 2);
                user = new User( id: id, name);
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return user;
    }
    public boolean delete(Integer id) {
        throw new UnsupportedOperationException();
    }
    public boolean delete(User entity) {
        throw new UnsupportedOperationException();
    }
    public boolean create(User entity) {
        throw new UnsupportedOperationException();
    }
    public User update(User entity) {
        throw new UnsupportedOperationException();
    }
}
```

Задания к контрольной работе

Вариант 1

Создать БД с нужными таблицами. Создать класс PhoneDAO. Реализовать метод Phone findEntityById(int id) в классе PhoneDAO, который позволяет найти объект в БД по его id. Реализовать метод List<Phone> findAll(), который возвращает все телефоны из БД. Реализовать метод boolean delete(int id), который удаляет телефон из БД по его id. Реализовать метод boolean delete(Phone phone), который удаляет телефон из БД по его id. Реализовать

метод boolean create(Phone phone), который добавляет новый телефон в БД. Реализовать метод Phone update(Phone phone), который обновляет существующий телефон в БД.

Вариант 2

Создать БД с нужными таблицами. Создать класс HospitalDAO. Реализовать метод Hospital findEntityById(int id) в классе HospitalDAO, который позволяет найти объект в БД по его id. Реализовать метод List<Hospital> findAll(), который возвращает всех врачей из БД. Реализовать метод boolean delete(int id), который удаляет врача из БД по его id. Реализовать метод boolean delete(Hospital doctor), который удаляет врача из БД по его id. Реализовать метод boolean create(Hospital doctor), который добавляет нового врача в БД. Реализовать метод Hospital update(Hospital doctor), который обновляет существующих врачей в БД.

Вариант 3

Создать БД с нужными таблицами. Создать класс RentCarDAO. Реализовать метод RentCar findEntityById(int id) в классе RentCarDAO, который позволяет найти объект в БД по его id. Реализовать метод List<RentCar> findAll(), который возвращает все автомобили из БД. Реализовать метод boolean delete(int id), который удаляет автомобиль из БД по его id. Реализовать метод boolean delete(RentCar auto), который удаляет автомобиль из БД по его id. Реализовать метод boolean create(RentCar auto), который добавляет новый автомобиль в БД. Реализовать метод RentCar update(RentCar auto), который обновляет существующие автомобили в БД.

Вариант 4

Создать БД с нужными таблицами. Создать класс StoreDAO. Реализовать метод Store findEntityById(int id) в классе StoreDAO, который позволяет найти объект в БД по его id. Реализовать метод List<Store> findAll(), который возвращает все товары из БД. Реализовать метод boolean delete(int id), который удаляет товар из БД по его id. Реализовать метод boolean delete(Store product), который удаляет товар из БД по его id. Реализовать метод boolean create(Store product), который добавляет новый товар в БД. Реализовать метод Store update(Store product), который обновляет существующие товары в БД.

Вариант 5

Создать БД с нужными таблицами. Создать класс TravelDAO. Реализовать метод Travel findEntityById(int id) в классе TravelDAO, который позволяет найти объект в БД по его id. Реализовать метод List<Travel> findAll(), который возвращает всех клиентов из БД. Реализовать метод boolean delete(int id), который удаляет клиента из БД по его id. Реализовать метод boolean delete(Travel tourist), который удаляет клиента из БД по его id. Реализовать метод boolean create(Travel tourist), который добавляет нового клиента в БД. Реализовать метод Travel update(Travel tourist), который обновляет существующих клиентов в БД.

Вариант 6

Создать БД с нужными таблицами. Создать класс RealtyDAO. Реализовать метод Realty findEntityById(int id) в классе RealtyDAO, который позволяет найти объект в БД по его id. Реализовать метод List<Realty> findAll(), который возвращает все квартиры из БД. Реализовать метод boolean delete(int id), который удаляет квартиру из БД по ее id. Реализовать метод boolean delete(Realty apartment), который удаляет квартиру из БД по ее id. Реализовать метод boolean create(Realty apartment), который добавляет новую квартиру в БД.

Реализовать метод Realty update(Realty apartment), который обновляет существующие квартиры в БД.

Вариант 7

Создать БД с нужными таблицами. Создать класс BookStoreDAO. Реализовать метод BookStore findEntityById(int id) в классе BookStoreDAO, который позволяет найти объект в БД по его id. Реализовать метод List<BookStore> findAll(), который возвращает все книги из БД. Реализовать метод boolean delete(int id), который удаляет книгу из БД по ее id. Реализовать метод boolean delete(BookStore book), который удаляет книгу из БД по ее id. Реализовать метод boolean create(BookStore book), который добавляет новую книгу в БД. Реализовать метод BookStore update(BookStore book), который обновляет существующие книги в БД.

Вариант 8

Создать БД с нужными таблицами. Создать класс UniversityDAO. Реализовать метод University findEntityById(int id) в классе UniversityDAO, который позволяет найти объект в БД по его id. Реализовать метод List<University> findAll(), который возвращает всех преподавателей из БД. Реализовать метод boolean delete(int id), который удаляет преподавателя из БД по его id. Реализовать метод boolean delete(University teacher), который удаляет преподавателя из БД по его id. Реализовать метод boolean create(University teacher), который добавляет нового преподавателя в БД. Реализовать метод University update(University teacher), который обновляет существующих преподавателей в БД.

Вариант 9

Создать БД с нужными таблицами. Создать класс GroupDAO. Реализовать метод Group findEntityById(int id) в классе GroupDAO, который позволяет найти объект в БД по его id. Реализовать метод List<Group> findAll(), который возвращает всех студентов из БД. Реализовать метод boolean delete(int id), который удаляет студента из БД по его id. Реализовать метод boolean delete(Group student), который удаляет студента из БД по его id. Реализовать метод boolean create(Group student), который добавляет нового студента в БД. Реализовать метод Group update(Group student), который обновляет существующих студентов в БД.

Вариант 10

Создать БД с нужными таблицами. Создать класс RestaurantDAO. Реализовать метод Restaurant findEntityById(int id) в классе RestaurantDAO, который позволяет найти объект в БД по его id. Реализовать метод List<Restaurant> findAll(), который возвращает все блюда из БД. Реализовать метод boolean delete(int id), который удаляет блюдо из БД по его id. Реализовать метод boolean delete(Restaurant dish), который удаляет блюдо из БД по его id. Реализовать метод boolean create(Restaurant dish), который добавляет новое блюдо в БД. Реализовать метод Restaurant update(Restaurant dish), который обновляет существующие блюда в БД.