МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

**Лабораторная работа № 5**

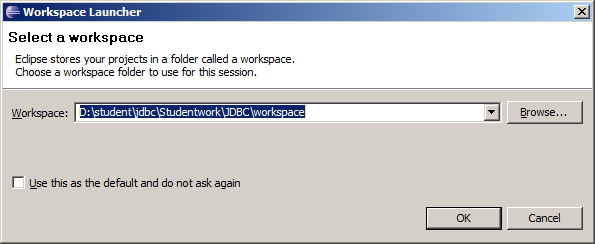
Выполнил студент группы № M3307  
Бойцов Виталий

САНКТ-ПЕТЕРБУРГ  
2019

# Упражнение 1. Настройка окружения JDBC

Цель упражнения: Научиться настраивать окружение JDBC.

Описание упражнения: В этом упражнении вы создадите базу Derby и импортируете в eclipse две библиотеки – драйвера для работы с базами Oracle и Derby

1. Необходимые файлы для работы с БД Derby находятся в директории jStudentwork\JDBC\Derby в архиве LabSetup\_JDBC.zip (далее – архив1). Папку Derby требуется разархивировать в любую папку (желательно без русских букв и пробелов в имени) локального диска.
2. Запустите Derby-сервер командой dbStart.cmd
3. Создайте базу данных командой dbCreate.cmd. База будет создана в директории JavaTunesDB.
4. Для запуска утилиты командной строки ij для работы с базой данных, выполните команду dbSQL.cmd. Для вывода команд, доступных в этой утилите, выполните команду help;
   1. Для просмотра списка схем в базе, выполните команду SHOW SCHEMAS; Среди прочих схем должна быть схема GUEST.
   2. Оцените структуру таблицы ITEM схемы GUEST, выполнив команду DESCRIBE GUEST.ITEM;
   3. Для проверки таблицы выполните SQL-предложение: select \* from GUEST.ITEM;
5. Запустите eclipse, указав в качестве Workspace папку workspace, разархивированную из архива1 :  
   
6. Создайте в eclipse новый проект (меню File – New – Project) с именем JDBCLabs. Постарайтесь не создать проект с другим именем, иначе придется копировать в него файлы с классами из архива1.
7. Добавить в этот проект библиотеки для работы с БД Oracle и Derby. Эти библиотеки лежат в файлах derbyclient.jar и ojdbc6.jar, которые находятся в архиве1.
   1. Создать в директории workspace\JDBCLabs (это директория, в которой хранится наш проект JDBCLabs) директорию lib и скопируйте в нее файлы derbyclient.jar и ojdbc6.jar из архива1.
   2. Для появления этой папки в eclipse Package Explorer или Project Explorer нажмите клавишу F5
   3. В окне свойств проекта (Project\Properties) перейдите в свойстве Java Build Path на вкладку Libraries и, нажав кнопку Add Jars… , импортируйте файлы derbyclient.jar и ojdbc6.jar из директории lib.

|  |
| --- |
|  |

Запустить на выполнение класс DBCapabilities и проверить вывод в консоли.   
Вывод должен начинаться с таких строк:  
Connected to: jdbc:derby://localhost:1527/JavaTunesDB  
Connected as: guest  
Driver name: Apache Derby Network Client JDBC Driver  
Driver version: 10.3.1.4 - (561794)  
Database name: Apache Derby  
Database version: 10.3.1.4 - (561794)  
Если возникли ошибки, проверьте работоспособность базы и подключение библиотек.

# Упражнение 2. Подключение к базе

Цель упражнения: Научиться подключаться к различным базам данных при помощи JDBC.

Описание упражнения: В этом упражнении вы создадите подключение к базам данных Derby и Oracle.

1. Создать новый класс JDBCTest в проекте JDBCLabs
2. В методе main этого класса подключитесь к базе данных Derby и выведите мета-информацию о соединении.
   1. Создайте переменную conn типа Connection и присвойте ей значение null
   2. Загрузите класс org.apache.derby.jdbc.ClientDriver при помощи статического метода forName класса Class
   3. Присвойте переменной conn значение, возвращенное статическим методом getConnection класса DriverManager В качестве параметра передайте этому методу следующее значение: jdbc:derby://localhost:1527/JavaTunesDB
   4. Создайте переменную dbmd - экземпляр класса DatabaseMetaData и присвойте ей значение, возвращенное методом getMetaData() переменной conn.
   5. Покажите в консоли некоторые метаданные, при помощи соответствующих методов переменной dbmd, например getDriverName().
   6. Проверьте имя пользователя, под которым вы подключились к базе данных Derby. Для этого можно использовать метод getUserName()
   7. Закройте подключение, выполнив метод close() у переменной conn.
   8. Сделайте так, чтобы подключение было выполнено под пользователем GUEST.
3. Создайте новый класс JDBCOracle и подключитесь из метода main к БД Oracle. Данные для подключения (файл tnsnames.ora, имя пользователя и пароль возьмите у преподавателя).  
   Имя класса для загрузки - oracle.jdbc.driver.OracleDriver, значение параметра для метода getConnection класса DriverManager - jdbc:oracle:thin:@//ИМЯ\_СЕРВЕРА:ПОРТ/ИМЯ\_СЕРВИСА

# Упражнение 3. Работа с DAO

Цель упражнения: Научиться создавать объекты для получения информации из базы данных и сохранять результат в объектах Java.

Описание упражнения: В этом упражнении вы попробуете создать метод для возврата данных из таблицы базы данных и сохранения их в объекте Java

1. Проверьте наличие в проекте JDBCLabs класса ItemDAO. У этого класса должны быть методы searchById, create, close, а также конструктор ItemDAO.
2. Измените метод searchById, чтобы он выполнил SQL-запрос к таблице ITEM из базы данных Derby и вернул строку со значением в колонке ID равной параметру этого метода, а затем сохранил полученные данные в созданном экземпляре класса MusicItem этого же пакета.
   1. Создайте String-переменную sql и присвойте ей строку SELECT для получения всех колонок таблицы ITEM из схемы GUEST для выборки строки таблицы, в которой колонка ITEM\_ID равна параметру этого метода
   2. В созданном ранее блоке try , инициируйте переменную stmt используя метод createStatement() переменной-экземпляра класса Connection (переменную создавать не требуется – она уже есть, надо ее найти).
   3. В этом же блоке try объявите переменную типа ResultSet с именем rs и инициируйтее ее возвращенным значением из метода executeQuery переменной stmt. В качестве параметра этого метода используйте переменную sql.
   4. В случае, если строк с указанным ITEM\_ID в таблице нет, метод должен вернуть null. Если же строка есть, требуется создать новый экземпляр класса MusicItem (переменная result) и заполнить его свойства данными из таблицы, а затем вернуть ссылку на него из метода. Для получения строки таблицы воспользуйтесь методом next(). При заполнении свойств переменной result пользуйтесь методами экземпляра класса ResultSet - getString(), getLong(“колонка”),getDate(”колонка”), getBigDecimal(”колонка”) в зависимости от типа данных свойства экземпляра класса.
   5. В блоке finally закройте экземпляр класса Statement (переменную stmt), выполнив метод close().
3. Создайте класс ItemDAOMain, в методе main которого, выполните проверку работоспособности метода searchById класса ItemDAO.
   1. Создайте экземпляр класса MusicItem
   2. Создайте экземпляр класса Connection для подключения к базе данных Derby,
   3. Создайте экземпляр класса ITemDAO и передайте в качестве параметра конструктора переменную, созданную на предыдущем шаге.
   4. Выполните дважды метод searchByID для этой переменной, передав в качестве параметра значения 1 и 100 соответственно.
   5. Сравните полученные результаты.
4. Если есть силы, время и желание, можно скопировать класс ItemDAO и сохранив его под другим именем (например, ItemDAOOracle) выполнить запрос select 99 id, ‘myTitle’ title, sysdate reldate from dual where 13=XXX к базе данных Oracle и сохранить полученный результат в экземпляре класса MusicItem. После этого создайте класс ItemDAOOracleMain и протестируйте работу класса ItemDAOOracle. Вместо XXX укажите значение заранее созданной переменной.

# Упражнение 4. Работа с DAO – запросы с параметром

Цель упражнения: Научиться реализовывать запросы с параметром при помощи методов PreparedStatement.

Описание упражнения: В этом упражнении вы создадите метод для реализации запроса с параметром.

1. Измените в классе ItemDAO метод searchByKeyword, чтобы он выполнил SQL-запрос к таблице ITEM из базы данных Derby и вернул все строки, в которых значения в колонке TITLE содержат (LIKE) значение параметра метода. Результат сохранить в коллекции ArrayList с типами элементов MusicItem.
   1. Создайте String-переменную sql и присвойте ей строку SELECT для получения всех колонок таблицы ITEM из схемы GUEST для выборки всех строк таблицы, в которой колонка TITLE содержит значение, передаваемое в качестве параметра в запрос. Для ограничения выборки используйте оператор LIKE вида TITLE LIKE ?
   2. Подготовьте SQL-предложение к работе, создав переменную pstmt- экземпляр класса PrepareStatement и инициализируйте ее, выполнив метод PrepareStatement у экземпляра класса Connection, – переменной уровня экземпляра класса ItemDAO.
   3. Подготовьте значение параметра запроса, выполнив метод setString у переменной pstmt для первого параметра в SQL-предложении. В качестве значения параметра используйте переменную wildcarded.
   4. Выполните запрос, создав экземпляр класса ResultSet – переменную rs и инициировать ее, применив для переменной pstmt метод ExecuteQuery без параметров
   5. Для сохранения результатов выборки в коллекции ArrayList используйте цикл while, метод next для переменной rs. Создание экземпляра класса MusicItem на каждом шаге цикла выполните так же, как и на предыдущей практике. Для добавления в коллекцию очередного экземпляра класса MusicItem не обойтись без метода add.
2. В классе ItemDaoMain дополните методе main вызвав 2 раза метод searchByKeyword параметрами of и Gay. Проверьте сколько элементов в коллекции возвращено этими методами. Сравните с реальными значениями в таблице ITEM.
3. Если все еще есть силы, время и желание, можно в скопированом классе ItemDAOOracle выполнить запрос select 99 id, ‘myTitle’ title, sysdate reldate from dual where 13=? union all select 98 id, ‘myTitle’ title, sysdate reldate from dual where 13=? к базе данных Oracle и сохранить полученный результат в коллекции ArrayList. После этого создайте класс ItemDAOOracleMain и протестируйте работу класса ItemDAOOracle со значениями параметров 13 и 10.

# Упражнение5. Работа с DAO – DML-операции

Цель упражнения: Научиться реализовывать DML-операции запросы с параметром при помощи метода ExecuteUpdate

Описание упражнения: В этом упражнении вы создадите метод для реализации вставки данных в таблицу.

1. В классе ItemDAO изменить метод create, добавив в него код для вставки данных из экземпляра класса MusicItem в таблицу ITEM базы данных Derby.
   1. Создайте String-переменную sql и поместите в нее SQL-предложение INSERT VALUES для вставки данных в таблицу ITEM. Список колонок в таблице можно посмотреть при помощи утилиты DerbySQL, выполнив команду describe guest.item; Значения для вставки должны передаваться в виде параметров. В колонку ITEM\_ID добавлять значение не требуется – оно заполняется автоматически.
   2. Создайте переменную stmt – экземпляр класса PreparedStatement и инициируйте ее, используя метод prepareStatement() переменной-экземпляра класса Connection. В качестве параметра для метода prepareStatement() используйте переменную sql.
   3. Заполните значения параметров, выполнив для переменной stmt методы setString, setDate, setBigDecimal и setInt. В качестве значений параметров воспользуйтесь get-методами параметра класса. В колонку RELEASEDATE вставьте значение переменной releaseDate, а в колонку VERSION – значение 1.
   4. Выполните запрос при помощи метода executeUpdate (); Выведите на экран сколько строк было вставлено.
   5. Сохраните изменения, выполнив метод commit у переменной-экземпляра класса Connection.
2. Измените метод main у класса ItemDaoMain для проверки работоспособности метода create.

|  |
| --- |
| package lab5; |
|  |  | | |
|  | import java.sql.\*; | | |
|  |  | | |
|  | public class DBConnectivity { | | |
|  | private static final String url = "jdbc:mysql://localhost:3306/autentification?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC"; | | |
|  | private static final String user = "root"; | | |
|  | private static final String password = "root"; | | |
|  |  | | |
|  | private static Connection con; | | |
|  |  | | |
|  | public static void main(String args[]) { | | |
|  |  | | |
|  | RequestDAO request; | | |
|  | try { | | |
|  | con = DriverManager.getConnection(url, user, password); | | |
|  | request = new RequestDAO(con); | | |
|  | System.out.println(request.searchById(2)); | | |
|  | System.out.println(request.searchById(90)); | | |
|  |  | | |
|  | request.searchByKeyword("Test").forEach(System.out::println); | | |
|  |  | | |
|  | RequestDTO requestDTO = createDTO(90,3, "Accepted", "Test for JDBC"); | | |
|  | request.create(requestDTO); | | |
|  |  | | |
|  | request.searchByKeyword("Test").forEach(System.out::println); | | |
|  |  | | |
|  | } catch (SQLException sqlEx) { | | |
|  | sqlEx.printStackTrace(); | | |
|  | } finally { | | |
|  | try { | | |
|  | con.close(); | | |
|  | } catch (SQLException se) { | | |
|  | } | | |
|  | } | | |
|  | } | | |
|  |  | | |
|  | private static RequestDTO createDTO(int id, int user\_id, String status, String description) { | | |
|  | RequestDTO dto = new RequestDTO(); | | |
|  | dto.setDescription(description); | | |
|  | dto.setStatus(status); | | |
|  | dto.setUserID(user\_id); | | |
|  | dto.setId(id); | | |
|  | return dto; | | |
|  | } | | |
|  |  | | |
|  | } | | |
| package lab5; | |
|  | |  | |
|  | | import java.sql.\*; | |
|  | | import java.sql.Date; | |
|  | | import java.util.\*; | |
|  | |  | |
|  | | public class RequestDAO { | |
|  | |  | |
|  | | private Connection conn; | |
|  | |  | |
|  | | public RequestDAO(Connection con) { | |
|  | | this.conn = con; | |
|  | | } | |
|  | |  | |
|  | | public RequestDTO searchById(int id) throws SQLException { | |
|  | | String sql = "select \* from requests where request\_id=" + id; | |
|  | | Statement stmt = conn.createStatement(); | |
|  | | ResultSet result = stmt.executeQuery(sql); | |
|  | | if (!result.isBeforeFirst()) return null; | |
|  | | RequestDTO res = new RequestDTO(); | |
|  | | while (result.next()) { | |
|  | | res = toDTO(result); | |
|  | | } | |
|  | | return res; | |
|  | | } | |
|  | |  | |
|  | | private RequestDTO toDTO(ResultSet result) throws SQLException { | |
|  | | RequestDTO dto = new RequestDTO(); | |
|  | | dto.setId(result.getInt("request\_id")); | |
|  | | dto.setDescription(result.getString("description")); | |
|  | | dto.setUserID(result.getInt("user\_id")); | |
|  | | dto.setStatus(result.getString("status")); | |
|  | | return dto; | |
|  | | } | |
|  | |  | |
|  | | public List<RequestDTO> searchByKeyword(String keyword) throws SQLException { | |
|  | | String sql = "select \* from requests where description like ?"; | |
|  | | PreparedStatement stmt = conn.prepareStatement(sql); | |
|  | | stmt.setString(1, keyword); | |
|  | | ResultSet result = stmt.executeQuery(); | |
|  | | List<RequestDTO> list = new ArrayList<>(); | |
|  | | while (result.next()) { | |
|  | | RequestDTO res = toDTO(result); | |
|  | | list.add(res); | |
|  | | } | |
|  | | return list; | |
|  | | } | |
|  | |  | |
|  | |  | |
|  | | public void create(RequestDTO dto) throws SQLException { | |
|  | | java.util.Date today = new java.util.Date(); | |
|  | | String sql = "insert into requests " + | |
|  | | "(request\_id, begin\_date, description, end\_date, status, type\_of\_request\_id, user\_id) " + | |
|  | | "values (?,?,?,?,?,?,?)"; | |
|  | | PreparedStatement stmt = conn.prepareStatement(sql); | |
|  | | stmt.setInt(1, dto.getId()); | |
|  | | stmt.setDate(2, new Date(today.getTime())); | |
|  | | stmt.setString(3, dto.getDescription()); | |
|  | | stmt.setDate(4, new Date(today.getTime())); | |
|  | | stmt.setString(5, dto.getStatus()); | |
|  | | stmt.setInt(6, 3); | |
|  | | stmt.setInt(7, dto.getUserID()); | |
|  | | stmt.executeUpdate(); | |
|  | | // conn.commit(); | |
|  | | } | |
|  | |  | |
|  | | public void close() throws SQLException { | |
|  | | conn.close(); | |
|  | | } | |
|  | | } | |
| package lab5; | | |
|  | | |  |
|  | | | import java.util.Date; |
|  | | |  |
|  | | | public class RequestDTO { |
|  | | | private int id; |
|  | | | private String description; |
|  | | | private String status; |
|  | | | private int userID; |
|  | | |  |
|  | | | public RequestDTO() { |
|  | | | } |
|  | | |  |
|  | | | public int getId() { |
|  | | | return id; |
|  | | | } |
|  | | |  |
|  | | | public void setId(int id) { |
|  | | | this.id = id; |
|  | | | } |
|  | | |  |
|  | | | public String getDescription() { |
|  | | | return description; |
|  | | | } |
|  | | |  |
|  | | | public void setDescription(String description) { |
|  | | | this.description = description; |
|  | | | } |
|  | | |  |
|  | | | public String getStatus() { |
|  | | | return status; |
|  | | | } |
|  | | |  |
|  | | | public void setStatus(String status) { |
|  | | | this.status = status; |
|  | | | } |
|  | | |  |
|  | | | public int getUserID() { |
|  | | | return userID; |
|  | | | } |
|  | | |  |
|  | | | public void setUserID(int userID) { |
|  | | | this.userID = userID; |
|  | | | } |
|  | | |  |
|  | | | @Override |
|  | | | public String toString() { |
|  | | | return "Request: " + |
|  | | | "id=" + id + |
|  | | | ", description='" + description + '\'' + |
|  | | | ", status='" + status + '\'' + |
|  | | | ", userID=" + userID; |
|  | | | } |
|  | | | } |