Лабораторная работа №10

Работа с базами данных (JDBC)

ODBC и JDBC

ODBC (*Open Database Connectivity - открытый механизм взаимодействия с базами данных*) — это программный интерфейс доступа к базам данных, разработанный фирмой Microsoft в сотрудничестве с Simba Technologies, призван унифицировать программное взаимодействие с СУБД, сделать его независимым от поставщика СУБД и программно-аппаратной платформы.

В начале 1990 г. существовало несколько поставщиков баз данных, каждый из которых имел собственный интерфейс. Если приложению было необходимо общаться с несколькими источниками данных, для взаимодействия с каждой из баз данных было необходимо написать свой код. Для решения возникшей проблемы создали стандартный интерфейс для получения и отправки источникам данных различных типов. Этот интерфейс был назван ODBC. С помощью ODBC прикладные программисты могли разрабатывать приложения для использования одного интерфейса доступа к данным, не беспокоясь о тонкостях взаимодействия с несколькими источниками. Это достигается благодаря тому, что поставщики различных баз данных создают драйверы, реализующие конкретное наполнение стандартных функций из ODBC с учётом особенностей их продукта.

JDBC (*Java Database Connectivity — соединение с базами данных на **Java***) — позволяет **Java-**приложениям получить доступ к данным различных СУБД. ODBC был взят в качестве основы JDBC из-за его популярности среди независимых поставщиков программного обеспечения и пользователей.

Уже в версии JDK1.1 появился пакет классов java.sql, обеспечивающий большинство функций, известных к тому времени разработчикам ODBC-приложений, например java.sql.CallableStatement, который обеспечивает выполнение на Java хранимых процедур; java.sql.DatabaseMetaData, который исследует базу данных на предмет ее реляционной полноты и целостности с получением самых разнообразных данных о типах и содержимом таблиц, колонок, индексов, ключей и т.д.; java.sql.ResultSetMetaData, с помощью которого можно выводить в удобном виде всю необходимую информацию из таблиц базы данных или печатать сами метаданные в виде названий таблиц и колонок.

JDBC позволяет выполняет основные три функции:

1. Установить соединение с базой данных
2. Посылать запросы и изменять состояние базы данных
3. Обрабатывать результаты запросов

Установка соединения. Классы DriverManager и Connection

Поскольку каждая СУБД является отдельным программным продуктом, для подключения к ней **Java** использует специальный драйвер, который пишется разработчиками данного СУБД. На официальном сайте, как правило, доступно скачивание соответствующих драйверов под каждую из версий СУБД.

Скачанный драйвер нужно добавить в ваш проект. В **Eclipse** это делается слудующим образом:

Нажмите правой кнопкой мыши на имя проекта в обозревателе проектов и выберите пункт *Build Path -> Configure Build Path*. Далее нажмите на вкладке *Library* и *Add External JARs*, выберите скачанный драйвер, нажмите *OK*.

Прежде чем использовать драйвер, его необходимо сперва зарегистрировать:

```
Class.forName("DriverName");
```

Имя драйвера можно найти на сайте разработчиков. Например для **Oracle** имя драйвера будет *oracle.jdbc.driver.OracleDriver*, для **MySQL -** *com.mysql.jdbc.Driver,* для **MS Access -** *sun.jdbc.odbc.JdbcOdbcDriver.*

Класс **DriverManager** используется для установления соединения с базой данных. Для этого необходимо указать ему специальный *URL адрес*, а также *логин* и *пароль* пользователя, зарегистрированного в СУБД. *URL* — это специальная строка, имеющая следующий формат:

jdbc:<subprotocol>:<subname>

где <*subprotocol*> — имя драйвера или имя механизма подключения,
<*subname*> — это строка, в которой указывается хост, порт, имя базы данных.

Например, для **MySQL** *URL* может быть таким: *jdbc:mysql://localhost:3306/MyDataBaseName*

В этом случае часть URL *//localhost:3306/MyDataBaseName* представляет собой и описывает имя хоста, порт и соответствующий идентификатор для доступа к соответствующей базе данных.

Для **Oracle**: *jdbc:oracle:thin:@localhost:1521/MyDataBaseName*

Для **MS Access**: *jdbc:odbc:db1_SQL*

В последнем случае используется механизм ODBC.

На сайте разработчиков также можно найти *URL* для их продукта.

Зная *URL, логин* и *пароль* пользователя, а также имея зарегистрированный в системе *драйвер*, установить подключение можно так:

```
Connection c = DriverManager.getConnection("URL", "User_Login", "User_Password");
```

Объект класса **Connection** представляет собой соединение с базой данных.

Создание и выполнение запросов. Классы Statement и ResultSet

После того, как соединение с базой данных установлено, мы можем отправлять запросы. Для этого нам понадобиться класс **Statement**, который предназначен для хранения SQL команд и может быть создан следующим образом:

```
Statement st = c.createStatement();
```

После этого мы можем выполнять запросы. Результаты запроса будут храниться в объекте класса **ResultSet.**

```
ResultSet rs = st.executeQuery("select * from Table_Name");
```

Данный запрос выберет все данные из таблицы *Table_Name* и вернет объект **ResultSet**. Также могут быть полезны методы *st.execute(«Запрос»);* Но тогда объект **ResultSet** надо получать отдельным методом *st.getResultSet()*. Также есть метод *st.executeUpdate(«Запрос»);* Вернет число строк, подвергшихся изменению.

Для параметризованного SQL запроса используется класс **PreparedStatement**. Он может быть использован, например, так:

```
PreparedStatement pst = c.prepareStatement("select * from MoCoUser where login = ?");
//? - это параметр
pst.setString(1, "user");//установление значения параметра. Обратите внимание: нумерация параметров начинается не с 0, а с 1!
//Выполнение запроса
ResultSet prs = pst.executeQuery();
```

Для вызова функции или процедуры используется класс **CallableStatement**:

```
CallableStatement cst = c.prepareCall("CALL proc_name(?,?)");//В процедуру также можно передавать параметры
cst.setInt(1, 100);
cst.setString(2, "String");
ResultSet rs = cst.executeQuery();
```

Обработка результатов запроса. Класс ResultSet

Результатом выполнения SQL запроса к БД будет таблица. В **java** результат сохраняется в объекте класса **ResultSet**. Для вывода строк этой таблицы на экран используется ряд методов.

Для перехода по строкам вперед и назад в классе **ResultSet** используются методы *next()* и *previous()*. Для перехода к первой или последней строке *first()* и *last()* соответственно. Обработка результатов в цикле будет выглядеть примерно следующим образом:

```
while(rs.next()){
//обработка результатов
}
```

Где *rs* — это объект класса **ResultSet**. Метод *next()* возвращает *true*, если есть следующая строка, *false* — больше строк нет.

Для получения значений из определенной колонки текущей строки можно получить методами *getInteger(<param>), getString(<param>), getDouble(<param>), getDate(<param>)* и так далее, где *<param>* — это номер колонки, если типа *int* или название колонки, если типа *String*. Например:

```
rs.getString(2); //Вернет строку, находящуюся во втором столбце текущей строки
rs.getDouble("average_score"); //Вернет значение типа double, находящееся в колонке с названием "average_score".
```

Поскольку данные в БД могут иметь значение *null*, имеет смысл перед их извлечением проверить это, чтобы не получить исключение. Данная проверка осуществляется методом *isNull(<param>)*, который вернет *true* или *false*.

Ниже приведен пример работы программы с **JDBC**:

```
package ru.javaxblog.jdbc;

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

public class JDBCExample {
     public static void main(String []args){
          String user = "User_Login";//Логин пользователя
          String password = "User_Password";//Пароль пользователя
          String url = "jdbc:oracle:thin:@localhost:1521/MyDBName";//URL адрес
          String driver = "oracle.jdbc.driver.OracleDriver";//Имя драйвера
          try {
               Class.forName(driver);//Регистрируем драйвер
          } catch (ClassNotFoundException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }
          Connection c = null;//Соединение с БД
          try{
            c = DriverManager.getConnection(url, user, password);//Установка соединения с БД
```

```
            Statement st = c.createStatement();//Готовим запрос
            ResultSet rs = st.executeQuery("select * from Table_Name");//Выполняем запрос к БД, результат в переменной rs
            while(rs.next()){
      System.out.println(rs.getString("Login"));//Последовательно для каждой строки выводим значение из колонки ColumnName
                }
           } catch(Exception e){
                e.printStackTrace();
           }
           finally{
                //Закрыть соединение с БД
                try {
                     if(c != null)
                     c.close();
                } catch (SQLException e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
                }
           }
      }
}
```

Основы SQL

SQL (Structured Query Language**)** – это структурированный язык запросов к реляционным базам данных (БД). SQL является декларативным языком, основанным на операциях реляционной алгебры.

Существуют два стандарта SQL, определённые американским национальным институтом стандартов (ANSI): SQL-89 (SQL-1) и SQL-92 (SQL-2). В настоящее время разрабатывается новый стандарт – SQL-3.

Большинство коммерческих систем управления базами данных (СУБД) поддерживают стандарт SQL-92, который принят ISO (International Standards Organization) в качестве международного стандарта. Многие версии имеют свои отличия, которые касаются, в основном, синтаксиса.

Синтаксис команд и примеры, рассмотренные в данном пособии, соответствуют синтаксису СУБД MS Access.

Извлечение данных из таблиц (отношений)

Извлечение данных из отношений выполняется с помощью команды **SELECT**. Эта команда не изменяет данные в БД.

Результатом выполнения команды *SELECT* является временное отношение, которое помещается в курсор (специальную область памяти СУБД) и обычно сразу выводится на экран. Синтаксис этой команды:

```
SELECT * | { [ ALL | DISTINCT ] <список выбора>.,..}
FROM {<имя таблицы> [<алиас>] }.,..
[ WHERE <условие>]
[ GROUP BY {<имя поля> | <целое>}.,.. [ HAVING <условие>] ]
[ ORDER BY {<имя поля> | <целое> [ ASC | DESC ] }.,..]
[ UNION [ALL] SELECT …];
```

Расшифровка элементов описания приведена в табл..

Таблица. Элементы команды SELECT

Элемент	Описание
<список выбора>	Список элементов, разделённых запятыми. Элемент списка выбора – выражение и необязательный алиас. Выражение может включать имена полей, знаки операций, вызовы функций и константы.
<имя таблицы>	Имя или синоним имени таблицы или представления.
<алиас>	Временный синоним имени таблицы, определённый только внутри запроса.
<условие>	Условие, которое может быть истинным или ложным для каждого поля или комбинации полей из таблицы (таблиц), определённых предложением FROM.
<имя поля>	Имя поля (столбца) таблицы.
<целое>	Число без десятичной точки. Номер поля в <списке полей>.

DISTINCT – предикат удаления из результирующего отношения повторяющихся кортежей.

ALL – предикат, обратный к *DISTINCT* (используется по умолчанию).

Рассмотрим основные предложения команды *SELECT*:

SELECT – после этого ключевого слова указывается **список выбора** – список выражений, которые будут образовывать результирующее отношение. Выражению можно сопоставить временный синоним (алиас), который будет названием поля результирующего отношения, например:

```
sal*0.87+bonus as salary
```

Если надо вывести все поля из тех отношений, к которым обращается данный запрос, можно указать символ * (если в отношениях нет полей с одинаковыми именами). В этом случае сначала будут выведены поля таблицы, стоящей первой в предложении *FROM*, затем – второй и т.д. Поля, относящиеся к одной таблице, будут выводиться в том порядке, в каком они были записаны при создании таблицы.

FROM – в этом предложении указывается имя таблицы (имена таблиц), в которой будет производиться поиск.

WHERE – содержит условия выбора отдельных записей.

GROUP BY – группирует записи по значению одного или нескольких полей. Каждой группе в результирующем отношении соответствует одна запись.

HAVING – позволяет указать условия выбора для групп записей. Может использоваться только после *group by*.

ORDER BY – упорядочивает результирующие записи по значению одного или нескольких полей: *ASC* – по возрастанию, *DESC* – по убыванию.

Порядок выполнения операции *SELECT* такой:

1. Выбор из указанной таблицы тех записей, которые удовлетворяют условию отбора (*where*).
2. Группировка полученных записей (*group by*).
3. Выбор тех групп, которые удовлетворяют условию отбора (*having*).
4. Сортировка записей в указанном порядке (*order by*).
5. Извлечение из записей полей, заданных в списке выбора, и формирование результирующего отношения.

Если во фразе *FROM* указаны две и более таблицы, то эта последовательность действий выполняется для декартова произведения указанных таблиц.

Задание.
Создана БД Access **db1_SQL.mdb**, состоящая из трех отношений (таблиц).

Отношение "Сотрудники" (Emp)

TabNo	*DepNo*	Name	Post	Salary	Born	Tel
988	*1*	Рюмин В.П.	начальник отдела	4850.0	01.02.60	5-26-12
909	*1*	Серова Т.В.	вед. программист	4850.0	20.10.71	5-91-19
100	*2*	Волков Л.Д.	программист	4650.0	16.10.72	null
034	*3*	Петрова К.В.	секретарь	3200.4	24.04.58	null
110	*2*	Буров Г.О.	бухгалтер	4588.5	22.05.65	5-46-32
023	*2*	Малова Л.А.	гл. бухгалтер	4924.0	24.11.54	4-24-55
002	*3*	Сухов К.А.	начальник отдела	4850.0	18.06.48	5-12-69

Отношение "Отделы" (Depart)

DepNo	Name
2	Бухгалтерия
3	Отдел кадров
4	Отдел технического контроля
1	Плановый отдел

Отношение "Дети"(Children)

TabNo	**Name**	Born	Gender
988	Вадим	03.05.85	м
110	Ольга	18.07.91	ж
023	Илья	19.02.77	м
023	Анна	26.12.79	ж
909	Инна	25.01.99	ж

Пример: Выполнить запрос по двум таблицам: вывести список сотрудников с детьми (e, c – алиасы):

```
select e.name, c.name, c.born from emp e, children c where e.tabno = c.tabno order by e.name, c.born;
```

Name	Child	Born
Буров Г.О.	Ольга	18.07.91
Малова Л.А.	Илья	19.02.77
Малова Л.А.	Анна	26.12.79
Рюмин В.П.	Вадим	03.05.85
Серова Т.В.	Инна	25.01.99

Ниже приведена инструкция подключения к БД Access **db1_SQL.mdb** из программы на Java.

Прописать созданную БД в ОС:

Пуск - Панель управления – Администрирование – Источник данных (ODBC) - Системный DSN (System DSN) – Добавить (Add) - Выбрать из списка "Micrososft Access Driver(*.mdb)" - Нажать "Готово" – В появившемся диалоговом окне в поле Имя источника данных ("Data Source Name") ввести имя БД, например db1 - Нажать кнопку "SELECT" и выбрать на дереве папок местоположение БД и ее имя (в этом случае – файл базы данных db1.mdb) – нажать кнопку ОК и в окне "System DSN" появится "db1 Microsoft Access Driver(*.mdb)"

Создать программу на Java для доступа к БД (см.пример выше).

Вставить в программу операторы SELECT языка SQL для вывода записей:

1. Вывести список сотрудников по отделам с указанием должности
2. Вывести список сотрудников второго и третьего отдела, имеющих оклады выше 4600 рублей
3. Вывести список программистов и ведущих программистов
4. Вывести список сотрудников старше 40 лет из 1-го и 3-го отделов
5. Вывести список сотрудников, не имеющих телефонов