Lecture 8 - Java Database

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

Outline

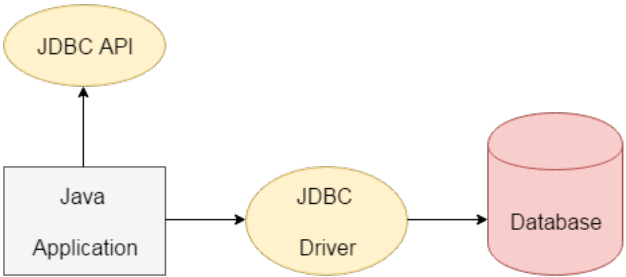
- Basic Structured Query Language (SQL)
- Java Database Connectivity (JDBC)

Java JDBC Introduction

- JDBC API uses JDBC drivers to connect with the database.
- JDBC API provides the access to tabular data stored in any relational database.
- JDBC API enables executing the query like save, update, delete, and fetch data from the database.

Java JDBC Introduction

JDBC is a Java API to connect and executes the query with the database.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

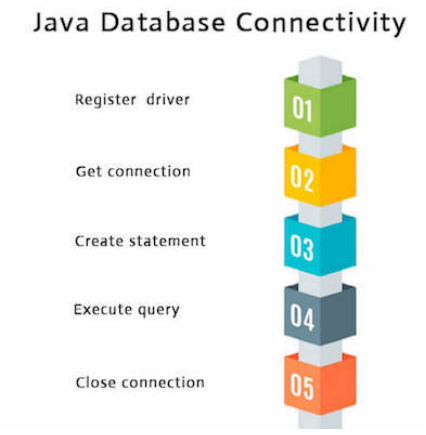
Java JDBC Introduction

- Why Should We Use JDBC?

Before JDBC, Open Database Connectivity (ODBC) API was the database API to connect and execute the query with the database. However, the ODBC driver is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

Java Database Connectivity Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:



1. Register the Driver class.
2. Create connections.
3. Create statements.
4. Execute queries.
5. Close connection.

Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Example to register the OracleDriver class:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Create the connection object

The getConnection() method of DriverManager class is used to establish a connection with the database.

Example to establish a connection with the Oracle database:

```
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create the Statement object

The createStatement() method of the Connection interface is used to create a statement. The object of the statement is responsible to execute queries with the database.

Example to create the statement object:

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

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create the Statement object

The PreparedStatement() method of the Connection interface is used to create a parameterized statement. It is used to execute the **parameterized query** with the database.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use the PreparedStatement interface because query is compiled only once.

Example to create the statement object:

```
PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)")
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Execute the query and obtain the result

In the following example, the executeQuery() method of the Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the table records.

Example to create the statement object:

```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Execute the query

The **Statement interface** provides methods to **execute queries** with the database.

- public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.
- public int executeUpdate(String SQL): is used to execute the specified query, it may be create, drop, insert, update, delete, etc.
- public boolean execute(String sql): is used to execute queries that may return multiple results.
- public int[] executeBatch(): is used to execute batch of commands.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Execute the query

The **PreparedStatement** interface is a subinterface of Statement. It is used to execute the **parameterized query**.

- public void setInt(int paramIndex, int value): sets the integer value to the given parameter index..
- public void setString(int paramIndex, String value): sets the String value to the given parameter index.
- public void setFloat(int paramIndex, float value): sets the float value to the given parameter index.
- public void setDouble(int paramIndex, double value): sets the double value to the given parameter index.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Execute the query

The **PreparedStatement** interface is a subinterface of Statement. It is used to execute the **parameterized query**.

- public int executeUpdate(): executes the query. It is used for create, drop, insert, update, delete etc.
- public ResultSet executeQuery(): executes the select query. It returns an instance of ResultSet.

More about Execute the query

The **PreparedStatement** interface is a subinterface of Statement. It is used to execute the **parameterized query**.

Example of PreparedStatement interface that updates the record:

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);

int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Execute the query

The **PreparedStatement** interface is a subinterface of Statement. It is used to execute the **parameterized query**.

Example of PreparedStatement interface that deletes the record :

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);

int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Execute the query

The **PreparedStatement** interface is a subinterface of Statement. It is used to execute the **parameterized query**.

Example of PreparedStatement interface that retrieves the records of a table :

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Obtain the result

The object of **ResultSet** maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

- By default, ResultSet object **can be moved forward only** and **it is not updatable**.
- But we can make this object to move forward and backward direction by:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_UPDATABLE);
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Obtain the result

The object of **ResultSet** maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

- public boolean next(): moves the cursor to the one row next from the current position.
- public boolean previous(): moves the cursor to the one row previous from the current position.
- public boolean first(): moves the cursor to the first row in result set object.
- public boolean last(): moves the cursor to the last row in result set object.

More about Obtain the result

The object of **ResultSet** maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

- public boolean absolute(int row): moves the cursor to the specified row number in the ResultSet object.
- public boolean relative(int row): moves the cursor to the relative row number in the ResultSet object, it may be positive or negative.
- public int getInt(int columnIndex): returns the data of the specified column index of the current row as int.
- public int getInt(String columnName): returns the data of the specified column name of the current row as int.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

More about Obtain the result

The object of **ResultSet** maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

- public String getString(int columnIndex): returns the data of the specified column index of the current row as String.
- public String getString(String columnName): returns the data of the specified column name of the current row as String.

More about Obtain the result

Example of obtaining the results:

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");

//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```

Close the connection object

By closing the connection object statement and ResultSet will be closed automatically.

The close() method of the Connection interface is used to close the connection.

Example to close connection:

```
public void close()throws SQLException
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java Database Connectivity with MySQL

To connect the Java application with the MySQL database, we need the following steps.

1. **Driver class**: The driver class for the MySQL database is com.mysql.jdbc.Driver.
2. **Connection URL**: jdbc:mysql://localhost:3306/sonoo where 1) jdbc is the API, 2) mysql is the database, 3) localhost is the MySQL server name, (we may use other IP address), 5) 3306 is the port number and 6) sonoo is the database name.
3. **Username**: The default username for the mysql database is root.
4. **Password**: It is the password given by the user at the time of installing the mysql database. In this example, we are going to use the root as the password.

Java Database Connectivity with MySQL

Let's first create a table in the MySQL database, but before creating a table, we need to create a database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with MySQL database: (in the next page)

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java Database Connectivity with MySQL

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen