Database Programming with JDBC API

Java Database Connectivity

- Overview of JDBC technology
- JDBC Drivers
- Seven basic steps in using JDBC
- Retrieving Data from ResultSet
- Manipulation Queries
- Using prepared and callable statements
- Handling SQL exceptions
- Submitting multiple statements as a transaction

- JDBC stands for Java Database Connectivity.
- Java provides JDBC API to create Java applications that are capable of interacting with a database.
- As we know that Java is an ideal language for persistent data storage.
- By using class inheritance and data encapsulation, a Java application developed to work with one type of data storage can be ported or extended to work with another.

Overview of JDBC technology

- An example for that is an application currently working with a RDBMS that is extended to use RMI to store data in a file.
- A general misunderstanding with Java is that database access is possible only with the JDBC API.
- Even though the JDBC provides lower level classes to manage database connections and transactions, it is not a compulsory component.
- We can easily connect our application with MS – SQL using MSSQLJAVA package instead of the MSSQLJDBC dirver.

- JDK 1.0 was released in January 1996.
- It contained all the classes necessary to implement database access – although developers had to code their own connections.
- The JDK contained core classes such as java.net and java.io were grouped together to provide necessary classes to connect to a database and send – receive data.
- Drawback of these classes for connection is that we have to create a socket and stream data to and from it to implement a simple transaction like inserting or selecting data from database.

- To overcome this drawback in February 1996, Sun released the JDBC as a separate package under the name java.sql.
- With the release of JDK 1.1, the JDBC package is not included as part of the core Java classes.
- The JDBC API is based on the SQL CLI (Call Level Interface).
- This standard explains how to access databases with function class embedded in applications.
- Certain limitations are there when we are using JDBC.

- One of them is to connect to a database the JDBC – ODBC bridge is used with native method calls and DLL (Dynamic Link Library).
- So it does not allow its use in applet accessible through internet.
- Because of potential security issues of an applet, the JDBC limits network connectivity to same host from which the applet was downloaded.
- Because of potential security risk of an applet, database must be installed on same server as the HTTP server.

- The JDBC – ODBC bridge driver converts all JDBC calls into ODBC calls and sends them to the ODBC driver.
- The ODBC driver then forwards the call to the database server.

Type – 1: JDBC – ODBC Bridge

JDBC – ODBC driver

- JDBC ODBC bridge is that it allows access to almost any database.
- Since the database's ODBC drivers are already available.

Advantages

- Since the bridge is not written fully in java, type 1 drivers are not portable.
- Performance is slowest in comparison to all drivers because the JDBC calls are first converted to ODBC calls. ODBC calls are then converted to native database calls. The same steps are performed in reverse order when the result of the query is obtained.
- The client system requires the ODBC installation to use the driver.
- Not good for the web application or for large scale database based application.

Disadvantages

- The JDBC type – 2 driver or the native-API/partly-java driver, communicates directly with the database server.
- Therefore, the vendor database library needs to be loaded on each client computer.
- The type – 2 driver converts JDBC calls into database – specific calls for databases. Example: Oracle will have oracle Native API.

Type – 2 Native – API/Partly-Java driver

Native API/partly – java driver

- They are typically offer better performance than the JDBC – ODBC Bridge.
- Because as the layers of communication are less than Type – 1 driver and also it uses native API which is database specific.

Advantage of Type-2 driver

- Native API must be installed in the client system and hence type – 2 drivers can not used for the internet.
- Like type-1 drivers, it is not written in java language which forms a portability issue.
- If we change the database we have to change the native API as it is specific to a database.
- Mostly outdated now because it is not thread safe.
- It is not suitable for distributed application.

Disadvantages are:

- The JDBC type – 3 driver, or the net – protocol /all java driver, follows a three – tiered approach.
- In the 3-tier approach, JDBC database requests are passed to a middle-tier server.
- The middle-tier server then translate the request and passes to the database – specific native – connectivity interface and forwards the requests to the database server.

Type – 3 Net-protocol/all-java driver

- If the middle – tier server is written in java, it can use the type-1 or type-2 drivers to forward the requests to the database server.

Net Protocol/ all – java driver

- Type – 3 drivers are used to connect a client application or applet to a database over TCP/IP connection.
- The presence of any vendor database library on client computers is not required because the type – 3 driver is server based.

- It requires database – specific coding to be done in the middle tier and it requires additional server for that.
- Additionally, traversing the recordset may take longer because the data comes through backend server.

Disadvantage of type-3 driver

- This driver is server based, so there is no need for any vendor database library to be present on client machines.
- This driver is fully written in java and hence portable, so it is suitable for the web applications.
- There are many opportunities to optimize portability, performance, and scalability.
- The net protocol can be designed to make the client JDBC driver very small and fast to load.

Advantages of type-3 driver

- The type – 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.
- This driver is very flexible and allows access to multiple databases using one driver.
- They are the most efficient amongst all driver types.

- Type – 4 drivers, or native protocol/ all java drivers, are completely implemented in java to achieve platform independence.
- Type 4 drivers convert JDBC calls into the vendor specific DBMS protocol.
- Therefore, client applications can communicate directly with the database server when type 4 drivers are used.

Type – 4 Native – protocol/All Java Driver

Net Protocol/all – java driver

- The performance of type 4 JDBC driver is better than other drivers because type 4 JDBC drivers do not have to translate database requests to ODBC calls or a native connectivity interface or pass the request on to another server.

- Disadvantage is that a different driver is needed for each database.

Disadvantage of type – 4 driver

- The major benefit of using type – 4 JDBC driver is that they are completely written java to achieve platform independence. And using this benefit we can reduce deployment administration issues. So, it is most suitable for the web application.
- Number of translation layers is very less.
- You need not install special software on the client or server. Further, these drivers can be downloaded automatically.

Advantages of type – 4 driver

JDBC API for JAVA Connectivity

- Connection interface defines connection to the different databases.
- An instance of the connection interface obtained from the getConnection() of DriverManager class.
- It is also able to get the information about table structure of database, its supported SQL grammar, its stored procedures, the capabilities of this connection and so on.
- This information is obtained with the getMetaData().

Connection Interface:

- createStatement():
  - It is used to create a statement object for sending SQL statement to the database. For example:
  - Statement st;
  - st=con.createStatement();   OR
  - st=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

- It is overloaded method with following signatures:
- Public statement createStatement() throws SQLException
- Public statement createStatement(int resultsetType, int concurremcy) throws SQLException
- If the same SQL statement is executed many times, it is difficult to use Statement Interface instead of it we can use PreparedStatement interface and its prepareStatement().

- prepareStatement():
  - It is used to create a PreparedStatement object for sending SQL statement to the database.
  - SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object.
  - This object can then be used to efficiently execute this statement multiple times. For Example:
  - PreparedStatement pst;
  - pst=con.prepareStatement(sql);

prepareStatement():

- This method is having following syntax:
- Public PreparedStatement prepareStatement(String sql)throws SQLException.

- It is used to create object of Callable Statement for calling stored procedure of database.
- The Callable Statement object provides methods for setting up it's IN and OUT parameters and methods for executing the call to a stored procedure. For example:

prepareCall():

- CallableStatement cst=con.prepareCall("{ call proc(?,?)}");
- cst.setString(1,id);
- cst.setString(2,nm);
- cst.executeUpdate();
- This method is having following signature:
- public CallableStatement prepareCall(String sql)throws SQLException

- It is used to release the connection. They do not wait for it to release automatically.
- It can be written as follow:
- con.close();
- This method is having following signatures:
- Public void close() throws SQLException

close():

commit():

- It is used to save all the states of the database which are changed by transaction.
- It can be written as follow:
- con.commit();
- This method is having following syntax:
- Public void commit() throws SQLException

- It is used to get the meta data related to current connection with the database.
- It provides the information related to dataset like table driver name, driver type its version and so on.
- It uses DatabaseMetaData interface to get the information database drivers.
- For example:

getMetaData():

- Connection con;
- con=DriverManager.getConnection(url,””,” ”);
- DatabaseMetaData ds=con.getMetaData();
- This method is having following syntax:
- Public DatabaseMetaData getMetaData() throws SQLException.

- It is used to do auto commit with all the SQL statements.
- If we set the auto commit with the connection all the SQL statements executed and committed with together as individual transaction.
- It has one parameter of boolean type with it.
- If we pass false it disables the auto commit.

setAutoCommit():

- For example:
- con.setAutoCommit(true);
- This method is having following syntax:
- Public void setAutoCommit(boolean b)throws SQLException

- It is used to get the current committed state.
- For example:
- Boolean b=con.getAutoCommit();
- This method is having following syntax:
- Public boolean getAutoCommit()throws SQLException.

getAutoCommit():

- It is used to cancel the effect of current running transaction.
- For example:
- con.rollback();
- This method is having following syntax:
- Public void rollback()throws SQLException

Rollback():

- It is used to check whether the connection has been closed or not.
- For example:
- Boolean b=con.isClosed();
- This method is having following syntax:
- Public boolean isClosed()throws SQLException.

isClosed():

- The DriverManager class belongs to java.sql package.
- It consists of the basic methods to manage the JDBC Drivers.
- Each and Every JDBC Driver must register with the DriverManager class.
- There are many JDBC drivers used for different JDBC server.

The DriverManager Class:

- For example, the JDBC driver for microsoft access is different than JDBC Driver for oracle.
- In short, DriverManager class controls the interface between the application and JDBC drivers.
- In DriverManager class important method is getConnection() method which is used to establish the connection with the different database servers.
- It has three overloaded syntaxes which are as follows:

- Public static Connection getConnection(String url)throws SQLException.
- Public static Connection getConnection(String url,Properties info)throws SQLException.
- Public static Connection getConnection(String url,String unm,String pwd)throws SQLException.

- This interface object is used to execute SQL statements and obtain the result produced after executing the statement.
- It is having following different types of methods to execute the different types of SQL statements.

Statement Interface:

executeQuery():

- This method is used to fetch the records from the database by using select query.
- It can only return the single ResultSet object at a time.
- It passes a parameter with it which is SQL statement which selects the records from the database. For example,
- Statement st=con.createStatement();
- ResultSet rs=st.executeQuery("select *from table");

- This method is having following syntax:
- Public ResultSet executeQuery(String str)throws SQLException

- This method is used to perform the insert, update and delete operations on the records of the database.
- It returns the integer value which indicates number of records updated in the database.
- It passes an SQL statement as a parameter which updates the records of the database. For example:

executeUpdate():

- Statement st=con.createStatement();
- Int n=st.executeUpdate("insert into table values(val1,val2…)");
- This method is having following syntax:
- Public int executeUpdate(String str)throws SQLException

- It is used to execute the SQL statements which return the multiple results.
- It returns the boolean indicating value if it returns true then it defines that the next result as ResultSet object and if it returns false then it defines number of records updated or no more result is there.
- It passes any SQL statement as parameter. For example:

execute():

- Statement st=con.createStatement();
- String str="create table tbl(col1 datatype(size),col2 datatype(size))";
- St.execute(str);
- This method has following syntax:
- Public boolean execute(String str)throws SQLException

- PreparedStatement interface is derived from the Statement interface.
- Disadvantages of Statement are that it executes a simple SQL statement without parameters and SQL statement is not precompiled.
- But PreparedStatement object uses a template to create a SQL request.
- and use to send a precompiled query with one or more parameters.

PreparedStatement Interface:

- So, sometimes it is more convenient to use a PreparedStatement object for sending SQL statements to the database.
- The main advantage of PreparedStatement is to execute a SQL statement many times, PreparedStatement reduces the execution time in comparison to Statement object.

- Creating Object PreparedStatement interface:
- String sql="insert into table(rno,nm)values(?,?)";
- PreparedStatement pstmt=con.prepareStatement(sql);

- 1. setInt(int paramIndex, int x)
- 2. setChar(int paramIndex,char x)
- 3. setString(int paramIndex,String x)
- 4. setFloat(int paramIndex,float x)
- 5. setDouble(int paramIndex,double x)
- 6. setLong(int paramIndex,long x)
- 7. setShort(int paramIndex,short x)
- 8. setDate(int paramIndex,Date x)
- 9. setByte(int paramIndex,byte x)
- 10.setBlob(int paramIndex,Blob x)
- Here, paramIndex - the first parameter is 1, the second is 2, and so on...
- x - the object containing the input parameter value.

setXXX():

- This method is used to fetch the records from
- the database by using select query.
- It can only return the single ResultSet object at a time.
- It passes a parameter with it which is SQL statement which selects the records from the database.

executeQuery():

- For example,
- ResultSet rs;
- String str="select *from table where id=?";
- PreparedStatement pstmt=con.prepareStatement(str);
- pstmt.setInt(1,123);
- rs=pstmt.executeQuery();
- This method is having following syntax:
- public ResultSet executeQuery(String sql)throws SQLException

- A CallableStatement object provides a way to call stored procedures in a standard way for all DBMSs.
- A stored procedure is stored in a database; the call to the stored procedure is what a CallableStatement object contains.
- This call is written in an escape syntax that may take one of two forms; one form with a result parameter, and the other without one.
- A result parameter, a kind of OUT parameter, is the return value for stored procedure.

Callable Statement:

- Both forms may have a variable number of parameters used for IN parameters, OUT parameters, or both INOUT parameters. A question mark serves as a placeholder for a parameter.

- Syntax for stored procedure with IN parameters:
- {call procedure_name [(?,?,...)]}

- Syntax for stored procedure with IN and OUT parameters:
- {? = call procedure_name [(?,?,...)]}

- Syntax for stored procedure with no parameters:
- {call procedure_name}

- It is used to access the data of the table from the database.
- ResultSet object stores the data of the table by executing the query.
- It also maintains the cursor position for navigation of the data.
- Cursor can move on first, next, previous and last position from the current row position.

ResultSet

- It also fetch the data from the table using getXXX methods depends on column type.
- It can fetch the data either passing column name or index number with getXXX methods.
- Index number always starts with one.
- Following methods and static fields are used to perform the above operations.

- CONCUR_READ_ONLY:
  - It defines that ResultSet object can not be modified or updated.
- TYPE_FORWARD_ONLY:
  - It defines that cursor from the current row can move forward only.
- TYPE_SCROLL_INSENSITIVE:
  - It defines that cursor can scroll but can not be modified or updated.
- TYPE_SCROLL_SENSITIVE:
  - It defines that cursor can scroll and also can be modified or updated.

Fields of ResultSet interface

- getXXX methods.
- getString()
- getInt()
- getBoolean()
- getDouble()
- getFloat()
- getDate()
- getLong()
- getShort()
- getByte()
- getBlob()

Methods of ResultSet interface:

- First():
  - It is used to move the cursor position on the first record of the table.
- Previous():
  - It is used to move the cursor position on the previous records of the table from current position.
- Next():
  - It is used to move the cursor position on the next records of the table from current position.

Navigation methods:

- Last():
  - It is used to move the cursor position on the last record of the table.
- afterLast():
  - It is used to move the cursor on after the last row.
- beforeFirst():
  - It is used to move the cursor on before the first row.
- Relative(int row):
  - It is used to move the cursor on relative no. of rows.

- Absolute(int row):
  - It is used to move the cursor on absolute row.

Other APIs (Meta Data)

- Meaning of MetaData is data about data.
- There are two types of MetaData interface are available:
    - ResultSetMetaData
    - DatabaseMetaData

- ResultSetMetaData:
  - This interface provides methods for obtaining information about the types and properties of the columns in a ResultSet object.
  - ResultSetMetaData is used to store following information of ResultSet object and methods of ResultSetMetaData class.

- What is the number of columns in the ResultSet?
- What is a column’s name?
- What is a column’s SQL type?
- What is the column’s normal maximum width in chars?
- What is suggested column title for use in printouts and displays?
- What is a column’s number of decimal digits?
- Can you put a NULL in this column?

- getColumnClassName(int column):
  - Returns the fully – qualified name of the java class whose instances are manufactured if the method ResultSet.getObject is called to retrieve a value from the column.
- getColumnCount():
  - Returns the number of columns in this ResultSet object.
- getColumnDisplaySize(int column):
  - Indicates the designated column's normal maximum width in characters.

- getColumnLabel(int column):
  - Gets the designated column’s suggested title for use in printouts and displays.
- getColumnName(int column):
  - Get the designated column’s name.
- getColumnType(int column):
  - Retrieves the designated column’s SQL type.
- getTableName(int column):
  - Get the designated column’s table name.

- This interface describes the database as a whole.
- Many methods of the DatabaseMetaData interface return lists of information in the form of ResultSet objects.
- This interface provides many methods that represent comprehensive information of the database.
- Database metadata is used to store following information of database metadata object.

DatabaeMetaData Interface

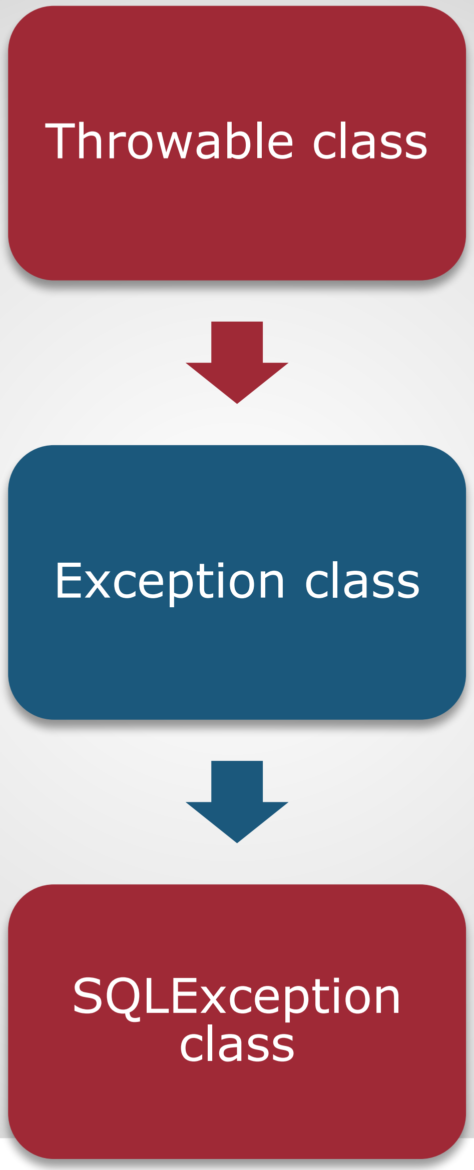
- What tables are available?
- What is our user name as known to the database?
- Is the database in read – only mode?
- If table correlation names are supported, are they restricted to be different from the names of the tables? And so on…

- allProceduresAreCallable():
  - Retrieves whether the current user can call all the procedures returned by the method getProcedures.
- allTablesAreSelectable():
  - Retrieves whether the current user can use all the tables returned by the method getTable in a select statement.
- getConnection():
  - Retrieves the connection that produced this metadata object.
- getDatabaseProductName():
  - Retrieves the name of this database product.

- getDatabaseProductVersion():
  - Retrieves the version number of this database product.
- getDriverName():
  - Retrieves the name of the JDBC driver.
- getDriverVersion():
  - Retrieves the version number of this JDBC driver as a String.

- The SQLException class extends the general java.lang.Exception class.
- It is an exception that provides information on a database access error or other errors.
- Below figure shows hierarchy of SQLException class.

SQL Exception


Throwable class
Exception class
SQLException class

- A string describing the error. This is used as the Java Exception message, available via the method getMessage().
- The SQLState string describing the error according to the XOPEN SQL state conventions. The different values of this string are defined in the XOPEN SQL specification.

SQLException provides following several kinds of information:

- An integer error code that is specific to each vendor. Normally this will be the actual error code returned by the underlying database.
- A chain to a next exception, which can be used to provide additional error information. This is frequently useful when you have exceptions that you want to let the user to know about, but you do not want to stop processing.

Return Type	Method	Description
Int	getErrorCode()	It is used to retrieve the vendor specific exception code for this SQLException object.
SQLException	getNextException()	It is used to retrieve the exception chained to this SQLException object.
String	getSQLState()	Retrieve the SQLState for this SQLException object.
Void	setNextException(SQLException se)	Adds a SQLException object to the end of the chain.