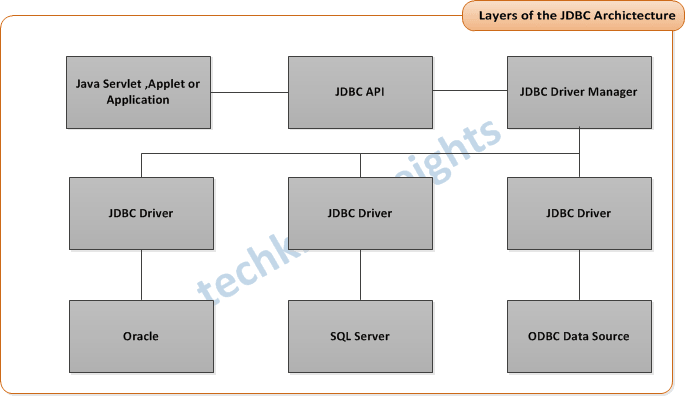
**JDBC Introduction**

* The interface for accessing relational databases from Java is JDBC. Via JDBC you maintain the databases connection, issue database queries and updates and receive the results.
* JDBC is Java application programming interface that allows the Java programmers to access database management system from Java code It was developed byJavaSoft, a subsidiary ofSun Microsystems.
* It is a java API which enables the java programs to execute SQL statements.
* In short JDBC helps the programmers to write java applications that manage these three programming activities :
  1. It helps us to connect to a data source, like a database.
  2. It helps us in sending queries and updating statements to the database.
  3. Retrieving andprocessing the results received from the database in terms of answering to your query.

**JDBC Driver Types:**

The JDBC API uses a Driver Manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

**Layers of the JDBC Architecture**



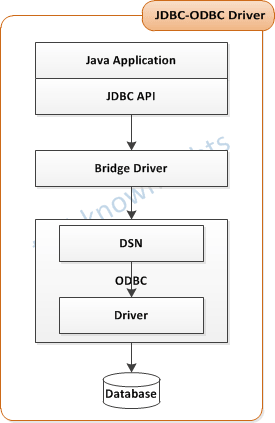
**Four types of JDBC Drivers :**

* Type -1 JDBC-ODBC Bridge driver.
* Type -2 Native-API Partly-Java driver.
* Type -3 JDBC-Net Pure-Java driver.
* Type -4 Native Protocol Pure-Java driver.

**Type-1 JDBC-ODBC Bridge driver.**

 The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API.

 The JDBC-ODBC Bridge driver is recommended only for experimental use or when no other alternative is available.



**Advantage :**

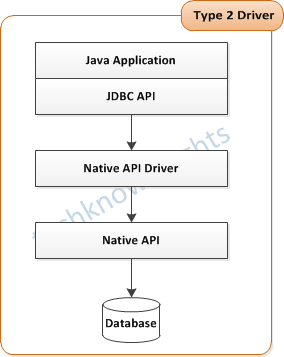
The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

**Disadvantage :**

* 1. Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.
* 2. A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.
* 3. The client system requires the ODBC Installation to use the driver.
* 4. Not good for the Web.

**Type-2 Native-API Partly-Java driver.**

The distinctive characteristic of type 2 jdbc drivers are that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native api.



**Advantages:**

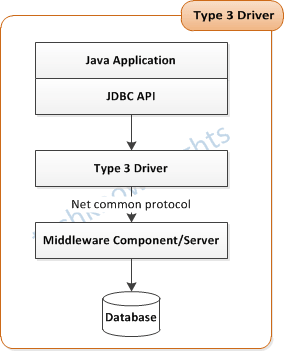
The distinctive characteristic of type 2 jdbc drivers are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type-1 and also it uses Native api which is Database specific.

**Disadvantages:**

* 1. Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
* 2. Like Type 1 drivers, it's not written in Java Language which forms a portability issue.
* 3. If we change the Database we have to change the native api as it is specific to a database
* 4. Mostly obsolete now
* 5. Usually not thread safe.

### Type-3 JDBC-Net Pure-Java driver

* Type 3 database requests are passed through the network to the middle-tier server.
* The middle-tier then translates the request to the database.



##### Advantage :

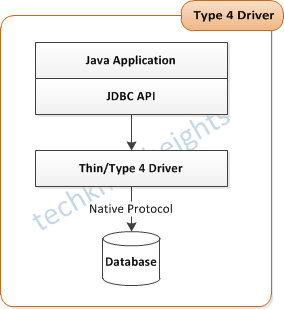
1. This driver is server-based, so there is no need for any vendor database library to be present on client machines.
2. This driver is fully written in Java and hence Portable. It is suitable for the web.
3. There are many opportunities to optimize portability, performance, and scalability.
4. The net protocol can be designed to make the client JDBC driver very small and fast to load.
5. 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.
6. This driver is very flexible allows access to multiple databases using one driver.
7. They are the most efficient amongst all driver types.

##### Disadvantage :

1. It requires another server application to install and maintain.
2. Traversing the recordset may take longer, since the data comes through the backend server.

#### Type-4 Native Protocol Pure-Java driver

The Type 4 uses java networking libraries to communicate directly with the database server.



###### Advantage

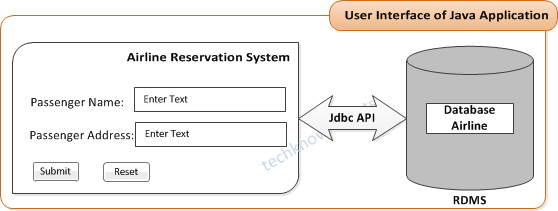
1. The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
2. Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good.
3. You don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

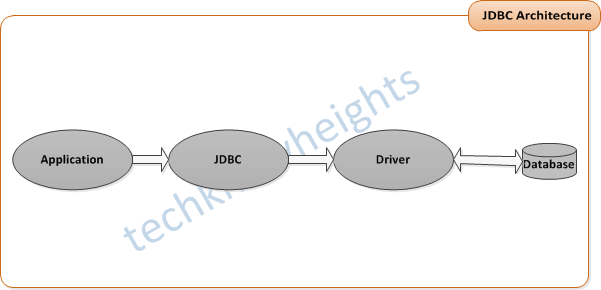
##### Disadvantage

1. With type 4 drivers, the user needs a different driver for each database.

**JDBC Architecture:**

* Sun Microsystems has included JDBC API as a part of J2SDK to develop Java applications that can communicate with databases.
* The following figure shows the Airline Reservation System developed in Java interacting with the Airlines database using the JDBC API :





### Types of Statements

* In java the statements specify the sequence of actions to be performed when a method or constructor is invoked. They can alter the value of variables, generate output, process input, or respond to user mouse or keyboard actions.
* Different types of statements are described in the following sections.
  + Statement: Execute simple sql queries without parameters. Statement createStatement() Creates an SQL Statement object.
  + PreparedStatement: Execute precompiled sql queries with or without parameters. PreparedStatement prepareStatement(String sql) returns a new PreparedStatement object. PreparedStatement objects are precompiled SQL statements.
  + Callable Statement: Execute a call to a database stored procedure. CallableStatement prepareCall(String sql) returns a new CallableStatement object. CallableStatement objects are SQL stored procedure call statements.

#### Using Statement

##### Various Database Operations:

Various database operations that you can perform using a Java application are:

* Querying a table
* Inserting rows in a table
* Updating rows in a table
* Deleting rows from a table

##### Select

###### Querying a Table

* The SELECTstatement is executed using the executeQuery() method and returns the output in the form of a ResultSet object.
* The code snippet to retrieve data from the authors table is:

String str = "SELECT \* FROM authors";

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(str);

##### Insert

###### Inserting Rows in a Table

* The executeUpdate() method enables you to add rows in a table.
* The code snippet to insert a row in the authors table is:

String str = "INSERT INTO authors(au\_id, au\_lname, au\_fname, address, city, state, contract)

VALUES ('998-72-3568','Ringer','Albert','801 826-0752 67 Seventh Av.','Salt Lake City','UT','1')";

Statement stmt = con.createStatement();

int count = stmt.executeUpdate(str);

##### Update

###### Updating Rows in a Table

* The code snippet to modify a row in the authors table is:

String str = "UPDATE authors SET address='10932 Second Av.a

WHERE au\_id='998-72-3568'";

Statement stmt = con.createStatement();

int count = stmt.executeUpdate(str);

##### Delete

###### Deleting Rows from a Table

* The code snippet to delete a row from the authors table is:

String str = "DELETE FROM authors WHERE au\_id='998-72-3568'";

Statement stmt = con.createStatement();

int count = stmt.executeUpdate(str);

#### Using Prepared Statement

* Java JDBC Prepared statements are pre-compiled SQL statements. Precompiled SQL is useful if the same SQL is to be executed repeatedly, for example, in a loop. Prepared statements in java only save you time if you expect to execute the same SQL over again.
* Prepared statement work same as the statement,but there are some differences. when you submitting the query   
  1st time following things happened
  1. compiling the query
  2. executing the query
  3. sending results back to the program

2nd time onwards

* 1. executing the query directly
  2. sending results back to the program

###### Quering Records

* The prepareStatement() method of the Connection object is used to submit parameterized query to a database.
* The SQL statement can contain **<?>** symbol as placeholders that can be replaced by input parameters at runtime. For example,

stat=con.prepareStatement("SELECT \* FROM authors WHERE au\_id = ?");

 The value of each parameter is set by calling an appropriate setXXX() method, where XXX is the data type of the parameter. For example,

stat.setString(1,"1001");

ResultSet result=stat.executeQuery();

###### Inserting Rows

* The code snippet to create a PreparedStatement object that inserts a row into authors table by passing authors data at runtime is:

String str = "INSERT INTO authors (au\_id, au\_fname, au\_lname) VALUES (?, ?, ?)";

PreparedStatement ps = con.prepareStatement(str);

ps.setString(1, "1001");

ps.setString(2, "Techknow");

ps.setString(3, "White");

int rt=ps.executeUpdate();

###### Updating Rows

* The code snippet to modify the state to CA where city is Oakland in the authors table using the PreparedStatement object is:

String str = "UPDATE authors SET state= ? WHERE city= ? ";

PreparedStatement ps = con.prepareStatement(str);

ps.setString(1, "CA");

ps.setString(2, "Oakland");

int rt=ps.executeUpdate();

###### Deleting Rows

* The code snippet to delete a row from the authors table where author€™s first name is Techknow using the PreparedStatement object is:

String str = "DELETE FROM authors WHERE au\_fname= ? ";

PreparedStatement ps = con.prepareStatement(str);

ps.setString(1, "Techknow");

int rt=ps.executeUpdate();

#### Using Callable Statement

* The CallableStatement extends the PreparedStatement interface.
* The CallableStatement provides an interface for calling database stored procedures.
* The simplest form of this syntax would be:{call procedure-name}
* which represents a call to a stored procedure with no parameters.
* A call to a stored procedure accepting two input parameters:

{call procedure-name (?, ?)}

CallableStatement callproc = connection.prepareCall("{call updateLast (?, ?)}");

callproc.setInt (1, 5);

// 1 specifies the first parameter

callproc.setString (2, "J");

// 2 specifies the second parameter

* To now execute the stored procedure, we use the following statement:callproc.executeUpdate();

**Transaction Management**

* When a connection is created using JDBC, by default it is in auto-commit mode.
* This means that each SQL statement is treated as a transaction and will be automatically committed immediately after it is executed.
* Sometimes, you want a group of statements to execute together or fail together.
* Transactions are used to group a set of statements so that they all execute successfully, or all fail.
* The way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode.
* The following line of code will do this:

conn.setAutoCommit(false);

* Once auto-commit mode is disabled, no SQL statement will be committed until the commit method is called explicitly.
* All statements starting from the previous call to the commit will be committed together.
* If there is a problem, the entire set of statements can be rolled back, without committing.
* By Default JDBC Transactions automatically committed immediately after it is executed and it is treated as a transaction. But imagine a situation where you want to execute a batch of statements, either they should commit at on go or they should get failed together.
* For this we need to disable the auto- commit mode by using the method: con.setAutoCommit(false). After setting the auto- commit as false, no SQL statement will be committed until we call the con.commit() method.
* If there arises any problem while committing then the set of statements will be rollback, without committing. con.rollback();

**Batch Upadate**

* One of the more advanced features of JDBC 2.0 is the ability to submit multiple update statements to the database for processing as a single unit.
* This batch updating can be significantly more efficient compared to JDBC 1.0, where each update statement has to be executed separately.
* In batch update more than one records can be added in the database simultaneously by using the some java methods like: addBatch and executeUpdate.

**Scrollable ResultSet**

* It is a interface. A ResultSet object maintains a cursor pointing to its current row of data
* ResultSet provides access to a table of data generated by executing a Statement. The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The next() method is used to successively step through the rows of the tabular results
* next() method is used to move the cursor to next position.
* A ResultSet object maintains a cursor that enables you to move through the rows stored in a ResultSet object.

**Types of ResultSet:**

There is three types of ResultSet object-

1. TYPE\_ FORWARD ONLY: The Result Set is not scrollable that means the cursor can move only forward direction before first row to last row.
2. TYPE\_SCROLL INSENSITIVE: This ResultSet is scrollable, its cursor can move both forward or backword direction .
3. TYPE\_SCROLL SENSITIVE: This ResultSet is scrollable, its cursor can move both forward and backword direction to the current position and it can move to an absolute position.

There are the different types of method in ResutSetScrollableData, some mostly used described here.

* First();- This method move the cursor to the first row of the data table.
* Last(); This method move the cursor to last row of the data table.
* Previous(); This method move the cursor current row to back row.
* Next(); This method move the cursor next row.
* Absolute(); This method move the cursor at the given row.
* Relative(); This method Moves the cursor a relative number of rows, either positive or negative
* Scrollable ResultSet : Result Set Scrollabale set allows the cursor to be moved to any row in the Result Set Back and forward

**ResultSetMetaData**

* The simple meaning of metadata is data about data.
* There are two metadata available in the JDBC API -ResultSetMetaData and DatabaseMetaData.
* Result Set Meta Data is a interface that can be used to get information about the type and properties of the columns in a ResultSet object.
* ResultSetMetaData is used to make descriptive information about ResultSet object, like; number of columns, name of columns and datatype of columns.
* It does not provide any information regarding database and how many rows are available in the ResultSet object.
* First we have to create object of ResultSetMetadata by calling getMetaData() method from ResultSet object.

Syntax :

ResultSetMetaData rsmd=res.getMeataData();

// res is a valid object of ResultSet object

* The following are common methods in ResultSetMetadata interface -

int getColumnCount()     String getColumnName()   
int getColumnType()      String getTableName()

* The following is the source code to demostrates ResultSetMetaData interface. Here user will input table name and display the result with column heading-

**DataBaseMetaData**

* JDBC DatabaseMetaData is an interface of java.sql.\*; package.
* DatabaseMetaData is generally implemented by the Database application vendors to know the capability of Database Management System in combination with the driver based JDBC technology.
* This interface is the tool for the user who need to discover how to deal with underlying database.
* Some meta data methods returns the string in form of ResultSet object.
* Then we retrieve data from this object with getInt(), and getString() methods.
* Some metadata objects takes argument also.
* DatabaseMetaData provides comprehensive information about the database.
* This interface is implemented by the driver vendors to allow the user to obtain information about the tables of a relational database as a part of JDBC application.
* User can use this interface to deal with various underlying DBMSs.
* Some of the methods are:-  
  getDatabaseMajorVersion() getDatabaseMinorVersion()   
  getDatabaseProductName() getDatabaseProductVersion()   
  getMaxRowSize() getPrimaryKeys()   
  getURL() getUserName() etc.

**JDBC provides four interfaces that deal with database metadata**

1. java.sql.DatabaseMetaData: about the database as a whole: table names, table indexes, database product name and version, and actions the database supports.
2. java.sql.ResultSetMetaData: about the types and properties of the columns in a ResultSet object.
3. java.sql.ParameterMetaData: about the types and properties of the parameters in a PreparedStatement object.
4. javax.sql.RowSetMetaData: about the columns in a RowSet object.

**RowSet**

* JDBC RowSet is an interface ofjavax.sql.rowset interface.
* This interface is wrapper around a ResultSet object that makes possible to use resultSet as java beans object.
* It can be one bean that makes available for composing of an application. Because a it continually maintain a connection JDBC connection to the database.
* Another advantage of JDBC RowSet is that it is used to makes ResultSet object scrollable and updateable.
* By default all the RowSet object are scrollable and updateable.
* An Example of Row Set Event listener is given below, To run this example at first create a database name student and create a table also named student

CREATE TABLE student (

RollNo int(9) PRIMARY KEY NOT NULL,

Name tinytext NOT NULL,

Course varchar(25) NOT NULL,

Address text

);

* Then insert the value into it as

insert into student VALUES(1, 'tkhts', 'BCA', 'Chennai') ;

insert into student VALUES(2, 'Techknow', 'MCA', 'Mumbai') ;

**SQL Exception**

* Handling SQL Exceptions
  + The java.sql package provides the SQLException class, which is derived from the java.lang.Exception class.
  + You can catch the SQLException in a Java application using the try and catch exception handling block.
  + The SQLException class contains various methods that provide error information, these methods are:
    - int getErrorCode(): Returns the error code associated with the error occurred.
    - String getSQLState(): Returns X/Open error code.
    - SQLException getNextException(): Returns the next exception in the chain of exceptions.

**Result Sets**

ResultSet provides access to a table of data generated by executing a Statement. The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The next() method is used to successively step through the rows of the tabular results.  
A ResultSet object maintains a cursor that enables you to move through the rows stored in a ResultSet object.

**Types of Result Sets**

The sensitivity of the ResultSet object is determined by one of three different ResultSet types:

* TYPE\_FORWARD\_ONLY:the result set is not scrollable i.e. the cursor moves only forward, from before the first row to after the last row.
* TYPE\_SCROLL\_INSENSITIVE:the result set is scrollable; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position.
* TYPE\_SCROLL\_SENSITIVE:the result set is scrollable; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position. Before you can take advantage of these features, however, you need to create a scrollable ResultSet object

The following line of code illustrates one way to create a scrollable ResultSet object:

Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,

ResultSet.CONCUR\_READ\_ONLY);

ResultSet srs = stmt.executeQuery("");

The first argument is one of three constants added to the ResultSet API to indicate the type of a ResultSet object: TYPE\_FORWARD\_ONLY, TYPE\_SCROLL\_INSENSITIVE, and TYPE\_SCROLL\_SENSITIVE.  
  
The second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable: CONCUR\_READ\_ONLY and CONCUR\_UPDATABLE.  
  
If you do not specify any constants for the type and updatability of a ResultSet object, you will automatically get one that is TYPE\_FORWARD\_ONLY and CONCUR\_READ\_ONLY.

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. import java.sql.Connection;
2. import java.sql.DriverManager;
3. import java.sql.PreparedStatement;
4. import java.sql.ResultSet;
5. import java.sql.SQLException;
6. import java.util.Scanner;
8. public class jdbcDemo
9. {
10. public static Connection getConnection() throws
11. ClassNotFoundException, SQLException
12. {
13. Connection con;
14. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
15. con=DriverManager.getConnection("jdbc:odbc:mydsn");
16. return con;
17. }
18. public static void selectData() throws ClassNotFoundException,
19. SQLException
20. {
21. Connection con=getConnection();
22. PreparedStatement ps=con.prepareStatement
23. ("select name,age from user");
24. ResultSet rs=ps.executeQuery();
25. while(rs.next())
26. {
27. System.out.println("name = "+rs.getString("name")+
28. " age = "+rs.getString("age"));
29. }
30. if(ps!=null)
31. {
32. ps.close();
33. }
34. if(con!=null){
35. con.close();
36. }
38. }
39. public static void main(String[] args) throws
40. ClassNotFoundException, SQLException
41. {
42. selectData();
43. }
44. }

**Output**

name = abc age = 18

name = xyz age = 20

**Type-1 CRUD**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. import java.sql.Connection;
2. import java.sql.DriverManager;
3. import java.sql.PreparedStatement;
4. import java.sql.ResultSet;
5. import java.sql.SQLException;
6. import java.util.Scanner;
8. public class jdbcDemo
9. {
10. public static Connection getConnection() throws
11. ClassNotFoundException, SQLException
12. {
13. Connection con;
14. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
15. con=DriverManager.getConnection("jdbc:odbc:mydsn");
16. return con;
17. }
18. public static void insertData() throws
19. ClassNotFoundException, SQLException
20. {
21. System.out.println("enter the name you want to insert = ");
22. String name=new Scanner(System.in).nextLine();
23. System.out.println("enter the age you want to insert = ");
24. String age=new Scanner(System.in).nextLine();
25. Connection con=getConnection();
26. PreparedStatement ps=con.prepareStatement("insert
27. into user(name,age) values(?,?)");
28. ps.setString(1, name);
29. ps.setString(2,age);
30. int count=ps.executeUpdate();
31. if(count>0)
32. {
33. System.out.println(count+" record added");
34. }
35. if(ps!=null)
36. {
37. ps.close();
38. }
39. if(con!=null){
40. con.close();
41. }
42. }
43. public static void selectData() throws
44. ClassNotFoundException, SQLException
45. {
46. Connection con=getConnection();
47. PreparedStatement ps=con.prepareStatement
48. ("select name,age from user");
49. ResultSet rs=ps.executeQuery();
50. while(rs.next())
51. {
52. System.out.println("name = "+rs.getString("name")+
53. " age = "+rs.getString("age"));
54. }
55. if(rs!=null)
56. {
57. rs.close();
58. }
59. if(ps!=null)
60. {
61. ps.close();
62. }
63. if(con!=null){
64. con.close();
65. }
67. }
68. public static void updateData() throws
69. ClassNotFoundException, SQLException
70. {
71. System.out.print("enter old name = ");
72. String oldName=new Scanner(System.in).nextLine();
73. System.out.print("enter new name = ");
74. String newName=new Scanner(System.in).nextLine();
75. Connection con=getConnection();
76. PreparedStatement ps=con.prepareStatement
77. ("update user set name=? where name=?");
78. ps.setString(1, newName);
79. ps.setString(2,oldName);
80. int count=ps.executeUpdate();
81. if(count>0)
82. {
83. System.out.println(count+" record updated");
84. }
85. if(ps!=null)
86. {
87. ps.close();
88. }
89. if(con!=null){
90. con.close();
91. }
93. }
94. public static void deleteData() throws
95. ClassNotFoundException, SQLException
96. {
97. System.out.print("enter name to delete = ");
98. String name=new Scanner(System.in).nextLine();
99. Connection con=getConnection();
100. PreparedStatement ps=con.prepareStatement
101. ("delete from user where name = ");
102. ps.setString(1, name);
103. int count=ps.executeUpdate();
104. if(count>0)
105. {
106. System.out.println(count+" record deleted");
107. }
108. if(ps!=null)
109. {
110. ps.close();
111. }
112. if(con!=null){
113. con.close();
114. }
115. }
116. public static void main(String[] args) throws
117. ClassNotFoundException, SQLException
118. {
119. while(true)
120. {
121. System.out.print("1- insert data\n2- read data\n3-
122. update data\n4- delete data\n5- exit\nEnter your choice =");
123. int choice=new Scanner(System.in).nextInt();
124. if(choice==1)
125. {
126. insertData();
127. }
128. if(choice==2)
129. {
130. selectData();
131. }
132. if(choice==3)
133. {
134. updateData();
135. }
136. if(choice==4)
137. {
138. deleteData();
139. }
140. if(choice==5)
141. {
142. System.exit(0);
143. }
144. else
145. {
146. System.out.println("wrong choice");
147. }
148. }
149. }
150. }

**Output**

Enter your choice =1

enter the name you want to insert =

abc

enter the age you want to insert =

21

1 record added

----

Enter your choice =2

name = abc age = 21

-----

Enter your choice =3

enter old name = abc

enter new name = xyz

1 record updated

------

Enter your choice =4

enter name to delete = xyz

1 record deleted

------

Enter your choice =5

**Type-4 Simple Example**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. import java.sql.Connection;
2. import java.sql.DriverManager;
3. import java.sql.PreparedStatement;
4. import java.sql.ResultSet;
5. import java.sql.SQLException;
6. import java.util.Scanner;
8. public class jdbcDemo
9. {
10. public static Connection getConnection() throws
11. ClassNotFoundException, SQLException
12. {
13. Connection con;
14. Class.forName("com.mysql.jdbc.Driver");
15. con=DriverManager.getConnection
16. ("jdbc:mysql://localhost:3306/myDb","root","root");
17. return con;
18. }
19. public static void selectData() throws
20. ClassNotFoundException, SQLException
21. {
22. Connection con=getConnection();
23. PreparedStatement ps=con.prepareStatement
24. ("select name,age from user");
25. ResultSet rs=ps.executeQuery();
26. while(rs.next())
27. {
28. System.out.println("name = "+rs.getString("name")+
29. " age = "+rs.getString("age"));
30. }
31. if(ps!=null)
32. {
33. ps.close();
34. }
35. if(con!=null){
36. con.close();
37. }
38. if(rs!=null)
39. {
40. rs.close();
41. }
42. }
43. public static void main(String[] args) throws
44. ClassNotFoundException, SQLException
45. {
46. selectData();
47. }
48. }

**Output**

name = xyz age = 20

**Type-4 CRUD**

**jdbcDemo.java**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. import java.sql.Connection;
2. import java.sql.DriverManager;
3. import java.sql.PreparedStatement;
4. import java.sql.ResultSet;
5. import java.sql.SQLException;
6. import java.util.Scanner;
8. public class jdbcDemo
9. {
10. public static Connection getConnection() throws
11. ClassNotFoundException, SQLException
12. {
13. Connection con;
14. Class.forName("com.mysql.jdbc.Driver");
15. con=DriverManager.getConnection
16. ("jdbc:mysql://localhost:3306/trial","root","root");
17. return con;
18. }
19. public static void insertData() throws
20. ClassNotFoundException, SQLException
21. {
22. System.out.println("enter the name you want to insert = ");
23. String name=new Scanner(System.in).nextLine();
24. System.out.println("enter the age you want to insert = ");
25. String age=new Scanner(System.in).nextLine();
26. Connection con=getConnection();
27. PreparedStatement ps=con.prepareStatement("insert into user
28. (name,age) values(?,?)");
29. ps.setString(1, name);
30. ps.setString(2,age);
31. int count=ps.executeUpdate();
32. if(count>0)
33. {
34. System.out.println(count+" record added");
35. }
36. if(ps!=null)
37. {
38. ps.close();
39. }
40. if(con!=null){
41. con.close();
42. }
43. }
44. public static void selectData() throws
45. ClassNotFoundException, SQLException
46. {
47. Connection con=getConnection();
48. PreparedStatement ps=con.prepareStatement
49. ("select name,age from user");
50. ResultSet rs=ps.executeQuery();
51. while(rs.next())
52. {
53. System.out.println("name = "+rs.getString("name")+
54. " age = "+rs.getString("age"));
55. }
56. if(ps!=null)
57. {
58. ps.close();
59. }
60. if(con!=null){
61. con.close();
62. }
63. if(rs!=null)
64. {
65. rs.close();
66. }
67. }
68. public static void updateData() throws
69. ClassNotFoundException, SQLException
70. {
71. System.out.print("enter old name = ");
72. String oldName=new Scanner(System.in).nextLine();
73. System.out.print("enter new name = ");
74. String newName=new Scanner(System.in).nextLine();
75. Connection con=getConnection();
76. PreparedStatement ps=con.prepareStatement
77. ("update user set name=? where name=?");
78. ps.setString(1, newName);
79. ps.setString(2,oldName);
80. int count=ps.executeUpdate();
81. if(count>0)
82. {
83. System.out.println(count+" record updated");
84. }
85. if(ps!=null)
86. {
87. ps.close();
88. }
89. if(con!=null){
90. con.close();
91. }
93. }
94. public static void deleteData() throws
95. ClassNotFoundException, SQLException
96. {
97. System.out.print("enter name to delete = ");
98. String name=new Scanner(System.in).nextLine();
99. Connection con=getConnection();
100. PreparedStatement ps=con.prepareStatement
101. ("delete from user where name = ?");
102. ps.setString(1, name);
103. int count=ps.executeUpdate();
104. if(count>0)
105. {
106. System.out.println(count+" record deleted");
107. }
108. if(ps!=null)
109. {
110. ps.close();
111. }
112. if(con!=null){
113. con.close();
114. }
115. }
116. public static void main(String[] args) throws
117. ClassNotFoundException, SQLException
118. {
119. while(true)
120. {
121. System.out.print("1- insert data\n2- read data\n3- update data\n4-
122. delete data\n5- exit\nEnter your choice =");
123. int choice=new Scanner(System.in).nextInt();
124. if(choice==1)
125. {
126. insertData();
127. }
128. if(choice==2)
129. {
130. selectData();
131. }
132. if(choice==3)
133. {
134. updateData();
135. }
136. if(choice==4)
137. {
138. deleteData();
139. }
140. if(choice==5)
141. {
142. System.exit(0);
143. }
144. else
145. {
146. System.out.println("wrong choice");
147. }
148. }
149. }
150. }

**Output**

Enter your choice =1

enter the name you want to insert =

ABC

enter the age you want to insert =

30

1 record added

-------

Enter your choice =2

name = ABC age = 30

------

Enter your choice =3

enter old name = ABC

enter new name = XYZ

1 record updated

------

Enter your choice =4

enter name to delete = XYZ

1 record deleted

---------

Enter your choice =5

**Transaction Management Example**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. package com.tkhts;
3. import java.sql.Connection;
4. import java.sql.DriverManager;
5. import java.sql.PreparedStatement;
6. import java.sql.ResultSet;
7. import java.sql.SQLException;
9. public class TransManagement
10. {
11. public static void main(String[] args) throws
12. ClassNotFoundException, SQLException
13. {
14. Class.forName("com.mysql.jdbc.Driver");
15. Connection con = null;
16. PreparedStatement pstmt = null;
17. PreparedStatement pstmtInsert = null;
18. ResultSet rs = null;
19. boolean r = false;
20. con = DriverManager.getConnection
21. ("jdbc:mysql://localhost:3306/mydb", "root","root");
22. con.setAutoCommit(false);
23. if(con == null)
24. {
25. System.out.println("Error in MySQL Connection ....");
26. System.exit(0);
27. }
28. pstmtInsert = con.prepareStatement
29. ("insert into emp(empno,name)values(?,?)");
30. pstmtInsert.setInt(1, 1004);
31. pstmtInsert.setString(2, "ABCD32323");
32. pstmtInsert.executeUpdate();
33. if(r)
34. {
35. con.commit();
36. System.out.println("Record Added");
37. }
38. else
39. {
40. con.rollback();
41. System.out.println("Record RollBack");
42. }
43. pstmtInsert.close();
44. /\*pstmt = con.prepareStatement("select empno ,
45. name  from emp ");
46. rs = pstmt.executeQuery();
47. while(rs.next()){
48. System.out.println("Empno "+rs.getInt("empno")+
49. " Name "+rs.getString("name"));
50. }\*/
51. /\*rs.close();\*/
52. /\*pstmt.close();\*/
53. con.close();
54. }
55. }

**Output**

Record RollBack

**Batch Update Example**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. package com.tkhts;
3. import com.mysql.jdbc.PreparedStatement;
4. import java.sql.Connection;
5. import java.sql.DriverManager;
6. import java.sql.SQLException;
7. public class BatchUpadate
8. {
9. public BatchUpadate()
10. {}
11. private Connection getConnection() throws
12. ClassNotFoundException, SQLException
13. {
14. Connection con=null;
15. Class.forName("com.mysql.jdbc.Driver");//Driver name of mysql
16. con=DriverManager.getConnection
17. ("jdbc:mysql://localhost:3306/mydb","root","root");
18. //connection path ,user and password.
19. if(con!=null)
20. {
21. System.out.println("connection");//connection check
22. }
23. else
24. {
25. System.out.println("no connection..");
26. }
27. return con;
28. }
29. boolean insertTable() throws ClassNotFoundException, SQLException
30. {
31. boolean isTrue=false;
32. PreparedStatement psmt=null;
33. Connection con=null;
34. String insertTableSQL = "INSERT INTO user10
35. (userid, pwd, createdby, created\_date) VALUES(?,?,?,?)";
36. // query for insert table
37. con = getConnection();
38. psmt = (PreparedStatement)con.prepareStatement(insertTableSQL);
39. con.setAutoCommit(false);
40. psmt.setString(1,"techknow");
41. psmt.setString(2, "heights");
42. psmt.setString(3, "system");
43. psmt.setTimestamp(4, getCurrentTimeStamp());
44. psmt.addBatch();//add batch 1 here
45. psmt.setString(1,"techknow");
46. psmt.setString(2, "tkhts");
47. psmt.setString(3, "system");
48. psmt.setTimestamp(4, getCurrentTimeStamp());
49. psmt.addBatch();// add batch 2 here
51. psmt.setString(1,"techknow");
52. psmt.setString(2, "systemdb");
53. psmt.setString(3, "system");
54. psmt.setTimestamp(4, getCurrentTimeStamp());
55. psmt.addBatch();//add batch 3 here
56. psmt.setString(1,"techknow");
57. psmt.setString(2, "tkh-heights");
58. psmt.setString(3, "system");
59. psmt.setTimestamp(4, getCurrentTimeStamp());
60. psmt.addBatch();//add batch 4 here
61. psmt.setString(1,"techknow");
62. psmt.setString(2, "tkh");
63. psmt.setString(3, "system");
64. psmt.setTimestamp(4, getCurrentTimeStamp());
65. psmt.addBatch(); // add batch 5 here
66. psmt.executeBatch();
68. //excute the all batch
69. con.commit();// commit the database
70. System.out.println("Record is inserted into DBUSER table!");
71. return isTrue;// return the result
72. }
73. private static java.sql.Timestamp getCurrentTimeStamp()
74. {
75. java.util.Date today = new java.util.Date();
76. return new java.sql.Timestamp(today.getTime());
77. //method to change the java.util.date into java.sql.lang
79. }
80. public static void main(String[] args) throws ClassNotFoundException,
81. SQLException
82. {
83. BatchUpadate batchUpadate = new BatchUpadate();
84. // create the BatchUpdate class object
85. batchUpadate.insertTable();
86. }
87. }

**Output**

connection

Record is inserted into DBUSER table!

**ResultSet MetaData Example**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. package com.tkhts;
3. import java.sql.\*;
4. import java.io.\*;
5. public class ResultSetMetaDataDemo
6. {
7. Connection con;
8. public ResultSetMetaDataDemo()
9. {
10. try
11. {
12. Class.forName("com.mysql.jdbc.Driver");
13. con=DriverManager.getConnection
14. ("jdbc:mysql://localhost:3306/mydb", "root","root");
15. }
16. catch(Exception e)
17. {
18. System.out.println("Error in Connection"+e);
19. }
20. }
21. public void displyRecords(String tableName)
22. {
23. String columnHeading="";
24. try
25. {
26. Statement stmt=con.createStatement();
27. ResultSet res=stmt.executeQuery("select\* from "+tableName.trim());
28. if(res.next())
29. {
30. ResultSetMetaData rsmd=res.getMetaData();
31. int colomntype=rsmd.getColumnDisplaySize(3);
32. System.out.println(colomntype);
33. int columnCount=rsmd.getColumnCount();
34. for(int i=1;i<=columnCount;i++)
35. {
36. columnHeading=columnHeading+"\t"+rsmd.getColumnName(i);
37. }
38. System.out.println(columnHeading);
39. while(res.next())
40. {
41. for(int i=1;i<=columnCount;i++)
42. {
43. System.out.println("\t"+res.getString(i));
44. }
45. System.out.println("\n");
46. }
47. }
48. else
49. {
50. System.out.println("There is no record in the table");
51. }
52. }
53. catch(Exception e)
54. {
55. e.printStackTrace();
56. }
57. }
58. public static void main(String[] args)
59. {
60. ResultSetMetaDataDemo obj=new ResultSetMetaDataDemo();
61. BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
62. String choice="";
63. try
64. {
65. do
66. {
67. System.out.println
68. ("Enter table name to display records
69. using ResultSetMetaData interface");
70. String name=br.readLine();
71. obj.displyRecords(name);
72. System.out.println("Do you want to continue(yes)");
73. choice=br.readLine();
74. }
75. while(choice.trim().equals("yes"));
76. }
77. catch(Exception e)
78. {
79. e.printStackTrace();
80. }
81. }
82. }

**Output**

Enter table name to display records using ResultSetMetaData interface

user8

20

userid pwd createdby created\_date

techknow

tkhts

system

2013-11-29

techknow

systemdb

system

2013-11-29

techknow

tkh-heights

system

2013-11-29

techknow

tkh

system

2013-11-29

Do you want to continue(yes)

no

**Example of DataBaseMetaData**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. package com.tkhts;
3. import java.sql.Connection;
4. import java.sql.DatabaseMetaData;
5. import java.sql.DriverManager;
6. public class DataBaseMetaData
7. {
8. static
9. {
10. try
11. {
12. Class.forName("com.mysql.jdbc.Driver").newInstance();
13. }
14. catch(Exception e)
15. {
16. e.printStackTrace();
17. }
18. }
19. private static Connection getConnection()
20. {
21. Connection connection=null;
22. try
23. {
24. connection=DriverManager.getConnection
25. ("jdbc:mysql://localhost:3306/mydb","root","root");
26. }
27. catch(Exception e)
28. {
29. e.printStackTrace();
30. }
31. return connection;
32. }
33. public static void main(String args[])
34. {
35. Connection con=getConnection();
36. try
37. {
38. DatabaseMetaData dbmd = con.getMetaData();
39. System.out.println
40. ("dbmd:driver version = "+dbmd.getDriverVersion());
41. System.out.println
42. ("dbmd:driver name = "+dbmd.getDriverName());
43. System.out.println
44. ("db name = "+dbmd.getDatabaseProductName());
45. System.out.println
46. ("db version = "+dbmd.getDatabaseProductVersion());
47. }
48. catch(Exception e)
49. {
50. e.printStackTrace();
51. }
52. }
53. }

**Output**

dbmd:driver version = mysql-connector-java-5.0.8 ( Revision: )

dbmd:driver name = MySQL-AB JDBC Driver

db name = MySQL

db version = 5.5.28

**Example of Rowset**

[view plaincopy to clipboardprint?](http://www.tkhts.com/core-java/jdbc/examples.jsp)

1. import java.sql.Connection;
2. import java.sql.DriverManager;
3. import java.sql.ResultSet;
4. import java.sql.Statement;
5. import javax.sql.RowSetEvent;
6. import javax.sql.RowSetListener;
7. import javax.sql.rowset.JdbcRowSet;
8. import com.sun.rowset.JdbcRowSetImpl;
10. public class JDBCRowSetExample
11. {
12. public static void main(String[] args) throws Exception
13. {
14. Connection connection = getMySqlConnection();
15. System.out.println("Connection Done");
16. Statement statement = connection.createStatement();
17. JdbcRowSet jdbcRowSet;
18. jdbcRowSet = new JdbcRowSetImpl(connection);
19. jdbcRowSet.setType(ResultSet.TYPE\_SCROLL\_INSENSITIVE);
20. String queryString = "SELECT \* FROM student";
21. jdbcRowSet.setCommand(queryString);
22. jdbcRowSet.execute();
23. jdbcRowSet.addRowSetListener(new ExampleListener());
24. while (jdbcRowSet.next())
25. {
26. // Generating cursor Moved event
27. System.out.println("Roll No- " + jdbcRowSet.getString(1));
28. System.out.println("name- " + jdbcRowSet.getString(2));
29. }
30. connection.close();
31. }
32. // My Sql connection method
33. public static Connection getMySqlConnection() throws Exception
34. {
35. String driver = "com.mysql.jdbc.Driver";
36. String url = "jdbc:mysql://localhost:3306/student";
37. String username = "root";
38. String password = "root";
39. Class.forName(driver);
40. Connection connection =
41. DriverManager.getConnection(url, username, password);
42. return connection;
43. }
44. }
46. class ExampleListener implements RowSetListener
47. {
48. @Override
49. public void cursorMoved(RowSetEvent event)
50. {
51. // TODO Auto-generated method stub
52. System.out.println("Cursor Moved Listener");
53. System.out.println(event.toString());
54. }
55. @Override
56. public void rowChanged(RowSetEvent event)
57. {
58. // TODO Auto-generated method stub
59. System.out.println("Cursor Changed Listener");
60. System.out.println(event.toString());
61. }
62. @Override
63. public void rowSetChanged(RowSetEvent event)
64. {
65. // TODO Auto-generated method stub
66. System.out.println("RowSet changed Listener");
67. System.out.println(event.toString());
68. }
69. }

**Output**

Techknow Heights

Tkhts