**Java JDBC**

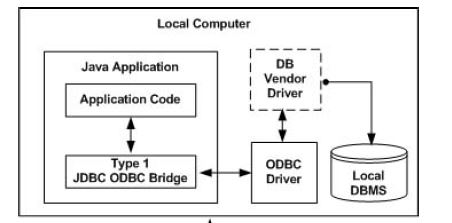
Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.

**Whatis JDBC Driver?**

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with the database server.

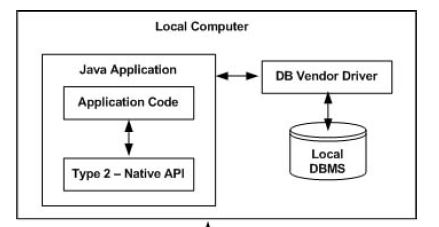
**Type 1: JDBC-ODBC Bridge Driver**

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.



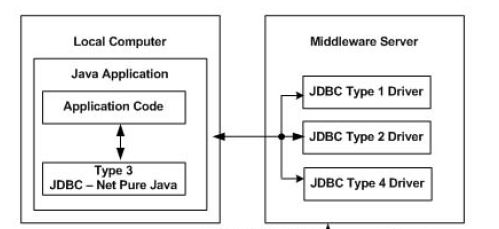
**Type 2: JDBC-Native API**

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls,which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.



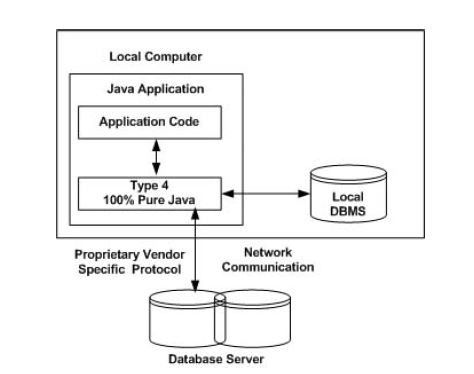
**Type 3: JDBC-Network protocol driver**

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.



**Type 4: Thin Driver**

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself. This kind of driver is extremely flexible, we don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically. Drivers depends on the Database.



**Steps to connect to the database in java:(mysql)**

**1) 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.

**Syntax of forName() method:**

public static void forName(String className)throws ClassNotFoundException

**Example to register the mysql Driver class**

Class.forName("com.mysql.jdbc.Driver")

**2) Create the connection object**

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

Syntax of getConnection() method

**1)** public static Connection getConnection(String url)throws SQLException

**2)** public static Connection getConnection(String url,String name,String password)

throws SQLException

Example to establish connection with the my database

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","root")

**3) Create the Statement object**

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

**Syntax of createStatement() method**

public Statement createStatement()throws SQLException

Example to create the statement object:

Statement stmt=con.createStatement();

**4) Execute the query**

The executeQuery() method of 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 records of a table.

Syntax of executeQuery() method

public ResultSet executeQuery(String sql)throws SQLException

**5) Close the connection object**

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

public void close()throws SQLException

**Example to close connection**

con.close();

=====================================================================================

**Example to connect to the mysql database in java**

**Driver class:** The driver class for the mysql database is com.mysql.jdbc.Driver.

**Connection URL:** The connection URL for the mysql database is jdbc:mysql://localhost:3306/test where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and test is the database name. We may use any database, in such case, you need to replace the test with your database name.

**Username:** The default username for the mysql database is root.

**Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

=====================================================================================

**Example to Connect Java Application with mysql database:**

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/test","root","root");

//here test 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);

}

}

}

=====================================================================================

**Connection interface**

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData

**Commonly used methods of Connection interface:**

**1) public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

**2) public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

**3) public void setAutoCommit(boolean status):** is used to set the commit status.By default it is true.

**4) public void commit():** saves the changes made since the previous commit/rollback permanent.

**5) public void rollback():** Drops all changes made since the previous commit/rollback.

**6) public void close():** closes the connection and Releases a JDBC resources immediately.

=====================================================================================

**Statement interface**

The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

**1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.

**2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.

**3) public boolean execute(String sql):** is used to execute queries that may return multiple results.

**4) public int[] executeBatch():** is used to execute batch of commands.

====================================================================

**Example of Statement interface**

import java.sql.\*;

class FetchRecord{

public static void main(String args[])throws Exception{

Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","root");

Statement stmt=con.createStatement();

//stmt.executeUpdate("insert into emp1 values(56,'Alok',28000)");

//int result=stmt.executeUpdate("update emp set name='James',salary=10000 where age=56");

int result=stmt.executeUpdate("delete from emp where age=56");

System.out.println(result+" records affected");

con.close();

}}

=====================================================================================

**ResultSet interface**

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

**Commonly used methods of ResultSet interface**

**1) public boolean next():** is used to move the cursor to the one row next from the current position.

**2) public boolean previous():** is used to move the cursor to the one row previous from the current position.

**3) public boolean first():** is used to move the cursor to the first row in result set object.

**4) public boolean last():** is used to move the cursor to the last row in result set object.

**5) public boolean absolute(int row):** is used to move the cursor to the specified row number in the ResultSet object.

**6) public boolean relative(int row):** is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.

**7) public int getInt(int columnIndex):** is used to return the data of specified column index of the current row as int.

**8) public int getInt(String columnName):** is used to return the data of specified column name of the current row as int.

**9) public String getString(int columnIndex):** is used to return the data of specified column index of the current row as String.

**10) public String getString(String columnName):** is used to return the data of specified column name of the current row as String.

=====================================================================================

**Example of Scrollable ResultSet**

import java.sql.\*;

class FetchRecord{

public static void main(String args[])throws Exception{

Class.forName("com.mysql.jdbc.Driver")

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","root");

Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);

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

**//getting the record of 3rd row**

rs.absolute(3);

System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();

}}

=====================================================================================

**PreparedStatement interface**

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

**String sql="insert into emp values(?,?,?)";**

**Methods of PreparedStatement interface**

**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.

**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.

=====================================================================================

**Example of PreparedStatement interface that inserts the record**

import java.sql.\*;

class InsertPrepared{

public static void main(String args[]){

try{

Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","root");

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

stmt.setInt(1,109); //1 specifies the first parameter in the query

stmt.setString(2,"Cage");

int i=stmt.executeUpdate();

System.out.println(i+" records inserted");

con.close();

}catch(Exception e){ System.out.println(e);}

}

}

==================================================================================

**PreparedStatement interface that updates the record**

PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");

stmt.setString(1,"Ada"); //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");

==================================================================================

**PreparedStatement interface that deletes the record**

PreparedStatement stmt=con.prepareStatement("select \* from emp");

ResultSet rs=stmt.executeQuery();

while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

=====================================================================================

**Batch Processing in JDBC**

Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

**Methods of Statement interface**

**void addBatch(String query):** It adds query into batch.

**int[] executeBatch():** It executes the batch of queries.

**Example of batch processing in jdbc**

import java.sql.\*;

class FetchRecords{

public static void main(String args[])throws Exception{

Class.forName("com.mysql.jdbc.Driver")

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","root")

con.setAutoCommit(false);

Statement stmt=con.createStatement();

stmt.addBatch("insert into emp values(256,'Nicolas',160000)");

stmt.addBatch("insert into emp values(191,'Tom',98000)");

stmt.executeBatch();//executing the batch

con.commit();

con.close();

}}