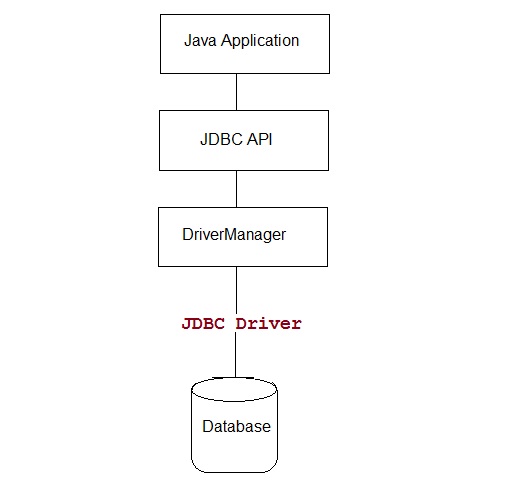
JDBC

The Java JDBC API enables Java applications to connect to relational databases via a standard API,

so your Java applications become independent (almost) of the database the application uses.

|  |
| --- |
| Java application using JDBC to connect to a database. |
| **Java application using JDBC to connect to a database.** |

The JDBC API consists of the following core parts:

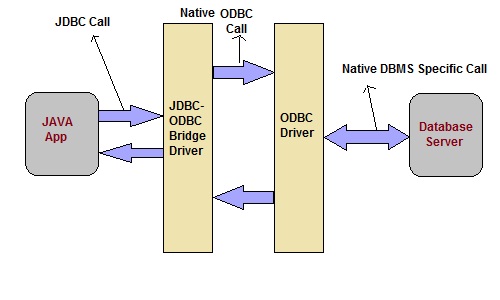
* JDBC Drivers
* Connections
* Statements
* Result Sets
* DBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.
* 

JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.

* **Type-1 Driver** or **JDBC-ODBC bridge**
* **Type-2 Driver** or **Native API Partly Java Driver**
* **Type-3 Driver** or **Network Protocol Driver**
* **Type-4 Driver** or **Thin Driver**

**JDBC-ODBC bridge**

**Type-1 Driver** act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.



**Advantage**

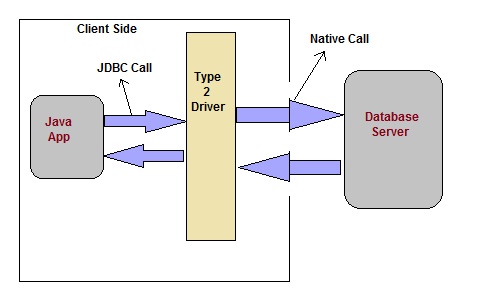
* Easy to use
* Allow easy connectivity to all database supported by the ODBC Driver.

**Disadvantage**

* Slow execution time
* Dependent on ODBC Driver.
* Uses Java Native Interface(JNI) to make ODBC call.

**Native API Driver**

This type of driver make use of Java Native Interface(JNI) call on database specific native client API. These native client API are usually written in C and C++.



**Advantage**

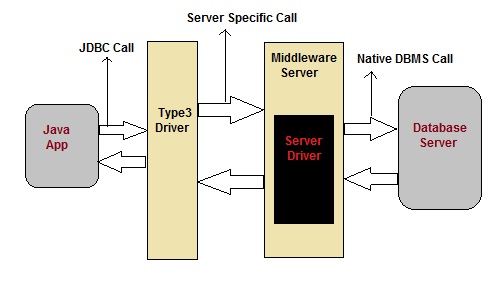
* faster as compared to **Type-1 Driver**
* Contains additional features.

**Disadvantage**

* Requires native library
* Increased cost of Application

**Network Protocol Driver**

This driver translate the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translate JDBC calls into database specific calls.



**Advantage**

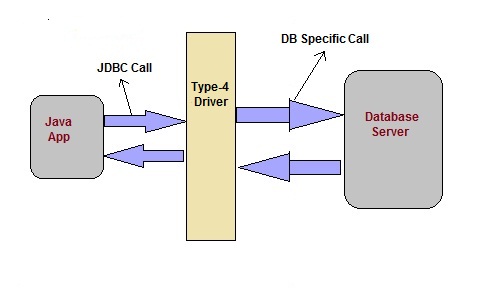
* Does not require any native library to be installed.
* Database Independency.
* Provide facility to switch over from one database to another database.

**Disadvantage**

* Slow due to increase number of network call.

**Thin Driver**

This is Driver called Pure Java Driver because. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.



**Advantage**

* Does not require any native library.
* Does not require any Middleware server.
* Better Performance than other driver.

**Disadvantage**

* Slow due to increase number of network call.

#### Important classes and interface of java.sql package

|  |  |
| --- | --- |
| **classes/interface** | **Description** |
| java.sql.BLOB | Provide support for BLOB(Binary Large Object) SQL type. |
| java.sql.Connection | creates a connection with specific database |
| java.sql.CallableStatement | Execute stored procedures |
| java.sql.CLOB | Provide support for CLOB(Character Large Object) SQL type. |
| java.sql.Date | Provide support for Date SQL type. |
| java.sql.Driver | create an instance of a driver with the DriverManager. |
| java.sql.DriverManager | This class manages database drivers. |
| java.sql.PreparedStatement | Used to create and execute parameterized query. |
| java.sql.ResultSet | It is an interface that provide methods to access the result row-by-row. |
| java.sql.Savepoint | Specify savepoint in transaction. |
| java.sql.SQLException | Encapsulate all JDBC related exception. |
| java.sql.Statement | This interface is used to execute SQL statements. |

#### javax.sql package

This package is also known as JDBC extension API. It provides classes and interface to access server-side data.

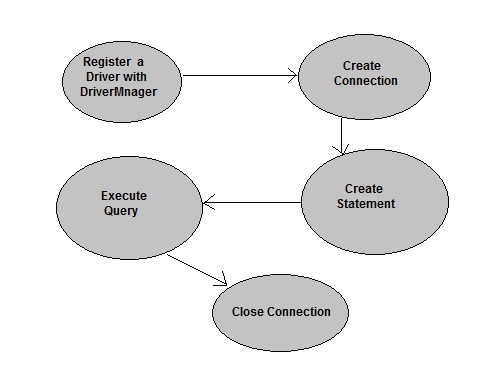
#### Important classes and interface of javax.sql package

|  |  |
| --- | --- |
| **classes/interface** | **Description** |
| javax.sql.ConnectionEvent | Provide information about occurence of event. |
| javax.sql.ConnectionEventListener | Used to register event generated by **PooledConnection**object. |
| javax.sql.DataSource | Represent the **DataSource** interface used in an application. |
| javax.sql.PooledConnection | provide object to manage connection pools. |

### Steps to connect a Java Application to Database

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

1. Register the Driver
2. Create a Connection
3. Create SQL Statement
4. Execute SQL Statement
5. Closing the connection



#### Register the Driver

Class.forName() is used to load the driver class explicitly.

**Example to register with JDBC-ODBC Driver**

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

#### Create a Connection

getConnection() method of **DriverManager** class is used to create a connection.

**Syntax**

getConnection(String url)

getConnection(String url, String username, String password)

getConnection(String url, Properties info)

**Example establish connection with Oracle Driver**

Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","username","password");

#### Create SQL Statement

createStatement() method is invoked on current **Connection** object to create a SQL Statement.

**Syntax**

public Statement createStatement() throws SQLException

**Example to create a SQL statement**

Statement s=con.createStatement();

#### Execute SQL Statement

executeQuery() method of **Statement** interface is used to execute SQL statements.

**Syntax**

public ResultSet executeQuery(String query) throws SQLException

**Example to execute a SQL statement**

ResultSet rs=s.executeQuery("select \* from user");

while(rs.next())

{

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

}

#### Closing the connection

After executing SQL statement you need to close the connection and release the session. The close() method of **Connection** interface is used to close the connection.

**Syntax**

public void close() throws SQLException

**Example of closing a connection**

con.close();