JDBC

https://www.javatpoint.com/java-jdbc

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. By the help of JDBC API, we can save, update, delete and fetch data from the database.

There are four types of JDBC drivers:

JDBC Driver is a software component that enables java application to interact with the database.

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is
Java language.

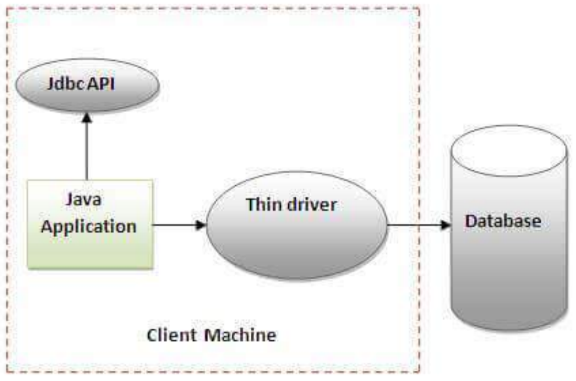


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.

- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

The current version of JDBC is **4.3.** It is the stable release since 21st September, 2017

The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

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

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.

   ```
   Class.forName("oracle.jdbc.driver.OracleDriver")
   Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
   Or
   DriverManager.registerDriver(new SQLServerDriver());
   ```

Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.

2. Create connection

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

   public static Connection getConnection(String url)throws SQLException
   public static Connection getConnection(String url,String name,String password) throws SQLException

```
Connection con =
DriverManager.getConnection("jdbc:sqlserver://localhost:1433","sa","password_123");
```

   String connUrl="jdbc:sqlserver://localhost:1433;"+"databaseName=Company;user=sa;password=password_123;";

   Connection con = DriverManager.getConnection(connUrl);

3. Create statement

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

Statement stmt=con.createStatement();

4. Execute queries

   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.

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

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

By default, connection commits the changes after executing queries.

5. Close connection

   con.close();

   By closing connection object, Statement and ResultSet will be closed automatically.

Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

It avoids explicit connection closing step.

The try-with-resources Statement

The `try`-with-resources statement is a `try` statement that declares one or more resources. A *resource* is an object that must be closed after the program is finished with it. The `try`-with-resources statement ensures that each resource is closed at the end of the statement.

The following example reads the first line from a file. It uses an instance of `BufferedReader` to read data from the file. `BufferedReader` is a resource that must be closed after the program is finished with it:

```
try (BufferedReader br = new BufferedReader(new FileReader(path)) ) {
    return br.readLine();
}
```

Prior to Java SE 7, you can use a `finally` block to ensure that a resource is closed regardless of whether the `try` statement completes normally or abruptly.

```
BufferedReader br = new BufferedReader(new FileReader(path));
    try {
        return br.readLine();
    } finally {
        if (br != null) br.close();
    }
```

***Note**: A `try`-with-resources statement can have `catch` and `finally` blocks just like an ordinary `try` statement. In a `try`-with-resources statement, any `catch` or `finally` block is run after the resources declared have been closed.*

```
try(  Connection conn =
DriverManager.getConnection("jdbc:sqlserver://localhost:1433","sa","password_123"))
{     }
```

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

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.

ResultSet interface

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

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.

PreparedStatement interface

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

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

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

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

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	**Description**
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.

INSERT:

```
PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

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

UPDATE:

```
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")
```

DELETE:

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

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

Retrieve:

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

Java ResultSetMetaData Interface

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Commonly used methods of ResultSetMetaData interface

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.

```
PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();
ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));
```

```
System.out.println("count: "+rsmd.getColumnCount());
System.out.println("column name: "+rsmd.getColumnName(1));
System.out.println("data type: "+rsmd.getColumnTypeName(1));
System.out.println("length: "+rsmd.getColumnDisplaySize(1));
```

Similarly we have DatabaseMetaData.

Java CallableStatement Interface

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

What is the difference between stored procedures and functions.

The differences between stored procedures and functions are given below:

Stored Procedure	**Function**
is used to perform business logic.	is used to perform calculation.(salary calculation, age calculation)
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

Procedure:

```
create or replace procedure "INSERTR"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user420 values(id,name);
end;
```

```
CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
stmt.setInt(1,1011);
stmt.setString(2,"Amit");
stmt.execute();
```

Function:

```
create or replace function sum4
(n1 in number,n2 in number)
return number
is
temp number(8);
begin
temp :=n1+n2;
return temp;
end;
```

```
CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");
stmt.setInt(2,10);
stmt.setInt(3,43);
stmt.registerOutParameter(1,Types.INTEGER);
stmt.execute();

System.out.println(stmt.getInt(1));
```

`void registerOutParameter(int parameterIndex, int sqlType)`

Registers the OUT parameter in ordinal position `parameterIndex` to the JDBC type `sqlType`.

Note- **registerOutParameter** method of CallableStatement interface, that registers the output parameter with its corresponding type. It provides information to the CallableStatement about the type of result being displayed.

The **Types** class defines many constants such as INTEGER, VARCHAR, FLOAT, DOUBLE, BLOB, CLOB etc.

Transaction Management in JDBC

Transaction represents **a single unit of work**.

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

Atomicity means either all successful or none.

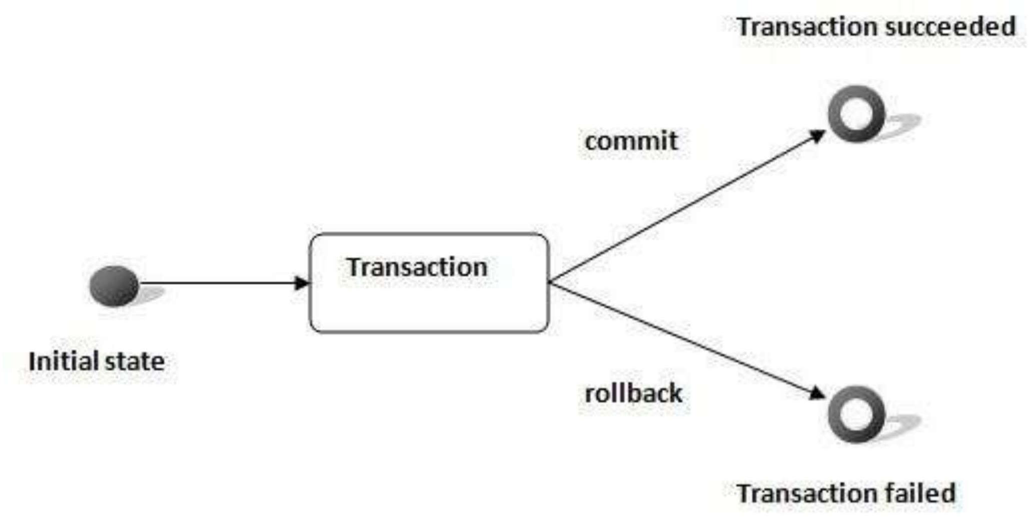
Consistency ensures bringing the database from one consistent state to another consistent state.

Isolation ensures that transaction is isolated from other transaction.

Durability means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Advantage of Transaction Management

fast performance It makes the performance fast because database is hit at the time of commit.



In JDBC, **Connection interface** provides methods to manage transaction.

Method	Description
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

```
con.setAutoCommit(false);

Statement stmt=con.createStatement();
stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");

con.commit();
con.close();
```

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.

Advantage of Batch Processing

Fast Performance

Methods of Statement interface

The required methods for batch processing are given below:

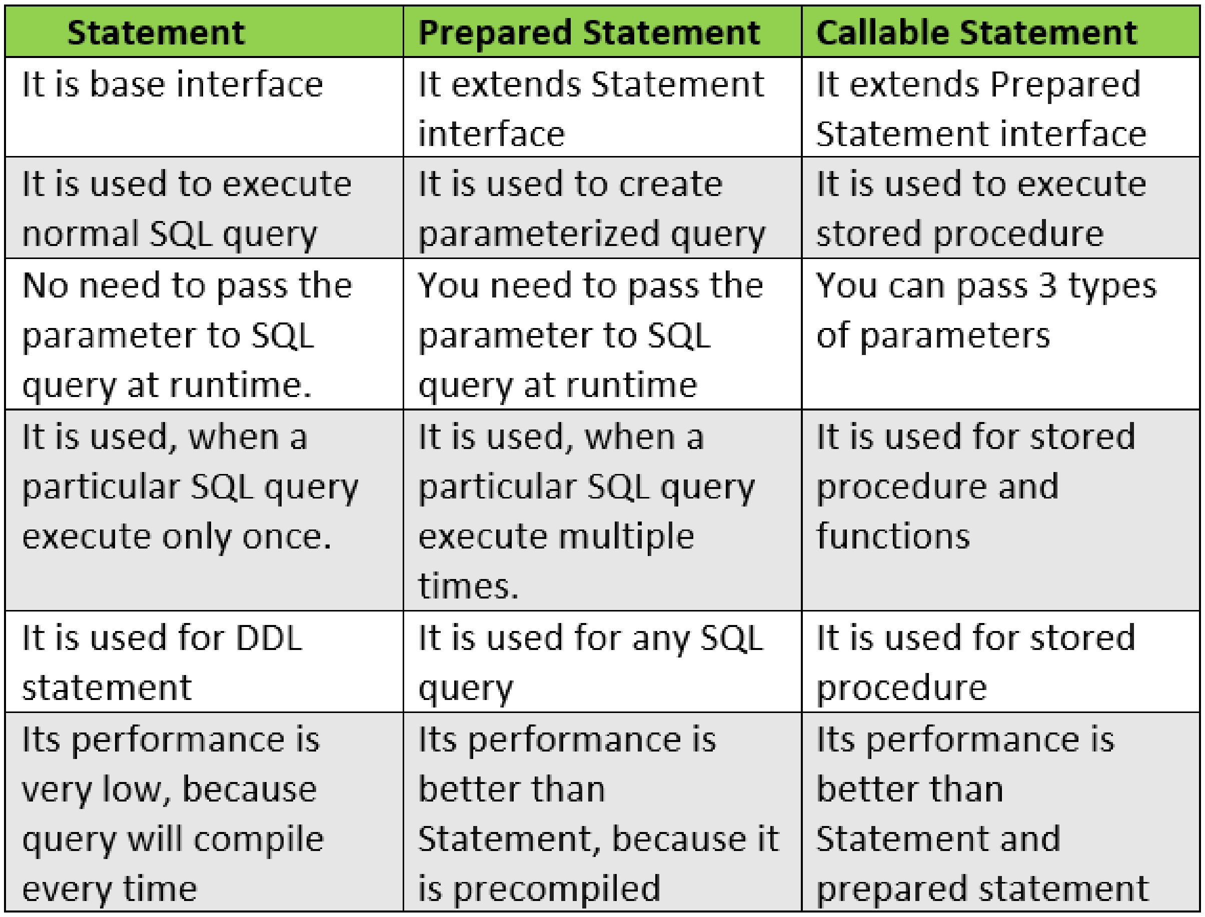
Method	Description
void addBatch(String query)	It adds query into batch.
int[] executeBatch()	It executes the batch of queries.

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

Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,'abhi',40000)");
stmt.addBatch("insert into user420 values(191,'umesh',50000)");

stmt.executeBatch();//executing the batch

con.commit();
con.close();
```



Statement	Prepared Statement	Callable Statement
It is base interface	It extends Statement interface	It extends Prepared Statement interface
It is used to execute normal SQL query	It is used to create parameterized query	It is used to execute stored procedure
No need to pass the parameter to SQL query at runtime.	You need to pass the parameter to SQL query at runtime	You can pass 3 types of parameters
It is used, when a particular SQL query execute only once.	It is used, when a particular SQL query execute multiple times.	It is used for stored procedure and functions
It is used for DDL statement	It is used for any SQL query	It is used for stored procedure
Its performance is very low, because query will compile every time	Its performance is better than Statement, because it is precompiled	Its performance is better than Statement and prepared statement

Some of the benefits of PreparedStatement over Statement are:

1. PreparedStatement helps us in preventing SQL injection attacks because it automatically escapes the special characters.
2. PreparedStatement allows us to execute dynamic queries with parameter inputs.
3. PreparedStatement provides different types of setter methods to set the input parameters for the query.
4. PreparedStatement is faster than Statement. It becomes more visible when we reuse the PreparedStatement or use it's batch processing methods for executing multiple queries.
5. PreparedStatement helps us in writing object Oriented code with setter methods whereas with Statement we have to use String Concatenation to create the query. If there are multiple parameters to set, writing Query using String concatenation looks very ugly and error prone.

The Prepared Statement is a slightly more powerful version of a Statement, and should always be at least as quick and easy to handle as a Statement.
The Prepared Statement may be parametrized

Most relational databases handles a JDBC / SQL query in four steps:

1. Parse the incoming SQL query
2. Compile the SQL query
3. Plan/optimize the data acquisition path
4. Execute the optimized query / acquire and return data

A Statement will always proceed through the four steps above for each SQL query sent to the database. A Prepared Statement pre-executes steps (1) - (3) in the execution process above. Thus, when creating a Prepared Statement some pre-optimization is performed immediately. The effect is to lessen the load on the database engine at execution time.

Now my question is that - "Is any other advantage of using Prepared Statement?"