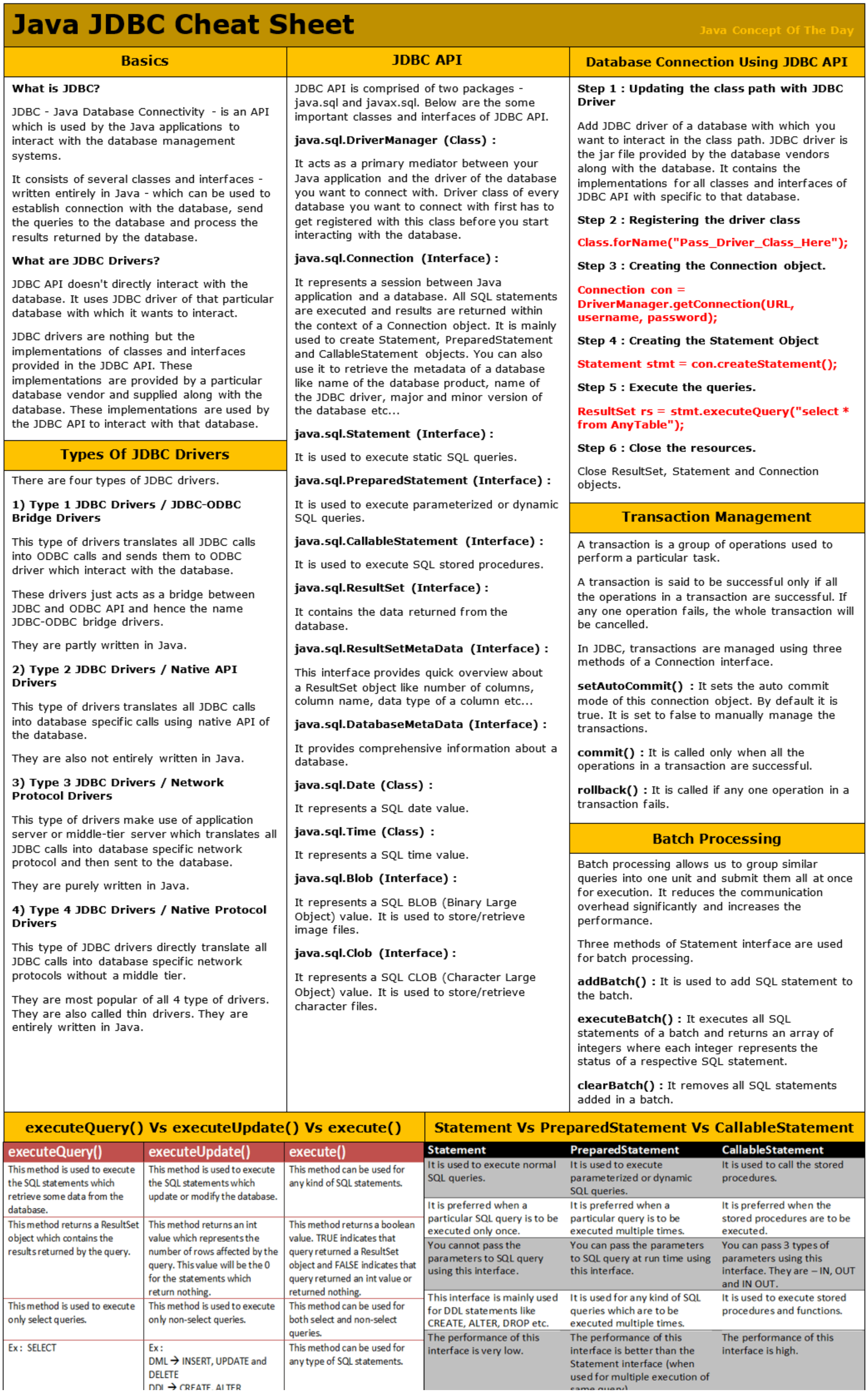

# Java JDBC Cheat Sheet

Java Concept Of The Day

## Basics

**What is JDBC?**

JDBC - Java Database Connectivity - is an API which is used by the Java applications to interact with the database management systems.

It consists of several classes and interfaces - written entirely in Java - which can be used to establish connection with the database, send the queries to the database and process the results returned by the database.

**What are JDBC Drivers?**

JDBC API doesn't directly interact with the database. It uses JDBC driver of that particular database with which it wants to interact.

JDBC drivers are nothing but the implementations of classes and interfaces provided in the JDBC API. These implementations are provided by a particular database vendor and supplied along with the database. These implementations are used by the JDBC API to interact with that database.

## Types Of JDBC Drivers

There are four types of JDBC drivers.

**1) Type 1 JDBC Drivers / JDBC-ODBC Bridge Drivers**

This type of drivers translates all JDBC calls into ODBC calls and sends them to ODBC driver which interact with the database.

These drivers just acts as a bridge between JDBC and ODBC API and hence the name JDBC-ODBC bridge drivers.

They are partly written in Java.

**2) Type 2 JDBC Drivers / Native API Drivers**

This type of drivers translates all JDBC calls into database specific calls using native API of the database.

They are also not entirely written in Java.

**3) Type 3 JDBC Drivers / Network Protocol Drivers**

This type of drivers make use of application server or middle-tier server which translates all JDBC calls into database specific network protocol and then sent to the database.

They are purely written in Java.

**4) Type 4 JDBC Drivers / Native Protocol Drivers**

This type of JDBC drivers directly translate all JDBC calls into database specific network protocols without a middle tier.

They are most popular of all 4 type of drivers. They are also called thin drivers. They are entirely written in Java.

## JDBC API

JDBC API is comprised of two packages - java.sql and javax.sql. Below are the some important classes and interfaces of JDBC API.

**java.sql.DriverManager (Class) :**

It acts as a primary mediator between your Java application and the driver of the database you want to connect with. Driver class of every database you want to connect with first has to get registered with this class before you start interacting with the database.

**java.sql.Connection (Interface) :**

It represents a session between Java application and a database. All SQL statements are executed and results are returned within the context of a Connection object. It is mainly used to create Statement, PreparedStatement and CallableStatement objects. You can also use it to retrieve the metadata of a database like name of the database product, name of the JDBC driver, major and minor version of the database etc...

**java.sql.Statement (Interface) :**

It is used to execute static SQL queries.

**java.sql.PreparedStatement (Interface) :**

It is used to execute parameterized or dynamic SQL queries.

**java.sql.CallableStatement (Interface) :**

It is used to execute SQL stored procedures.

**java.sql.ResultSet (Interface) :**

It contains the data returned from the database.

**java.sql.ResultSetMetaData (Interface) :**

This interface provides quick overview about a ResultSet object like number of columns, column name, data type of a column etc...

**java.sql.DatabaseMetaData (Interface) :**

It provides comprehensive information about a database.

**java.sql.Date (Class) :**

It represents a SQL date value.

**java.sql.Time (Class) :**

It represents a SQL time value.

**java.sql.Blob (Interface) :**

It represents a SQL BLOB (Binary Large Object) value. It is used to store/retrieve image files.

**java.sql.Clob (Interface) :**

It represents a SQL CLOB (Character Large Object) value. It is used to store/retrieve character files.

## Database Connection Using JDBC API

**Step 1 : Updating the class path with JDBC Driver**

Add JDBC driver of a database with which you want to interact in the class path. JDBC driver is the jar file provided by the database vendors along with the database. It contains the implementations for all classes and interfaces of JDBC API with specific to that database.

**Step 2 : Registering the driver class**

```
Class.forName("Pass_Driver_Class_Here");
```

**Step 3 : Creating the Connection object.**

```
Connection con =
DriverManager.getConnection(URL,
username, password);
```

**Step 4 : Creating the Statement Object**

```
Statement stmt = con.createStatement();
```

**Step 5 : Execute the queries.**

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

**Step 6 : Close the resources.**

Close ResultSet, Statement and Connection objects.

## Transaction Management

A transaction is a group of operations used to perform a particular task.

A transaction is said to be successful only if all the operations in a transaction are successful. If any one operation fails, the whole transaction will be cancelled.

In JDBC, transactions are managed using three methods of a Connection interface.

**setAutoCommit() :** It sets the auto commit mode of this connection object. By default it is true. It is set to false to manually manage the transactions.

**commit() :** It is called only when all the operations in a transaction are successful.

**rollback() :** It is called if any one operation in a transaction fails.

## Batch Processing

Batch processing allows us to group similar queries into one unit and submit them all at once for execution. It reduces the communication overhead significantly and increases the performance.

Three methods of Statement interface are used for batch processing.

**addBatch() :** It is used to add SQL statement to the batch.

**executeBatch() :** It executes all SQL statements of a batch and returns an array of integers where each integer represents the status of a respective SQL statement.

**clearBatch() :** It removes all SQL statements added in a batch.

## executeQuery() Vs executeUpdate() Vs execute()

| executeQuery() | executeUpdate() | execute() |
|---|---|---|
| This method is used to execute the SQL statements which retrieve some data from the database. | This method is used to execute the SQL statements which update or modify the database. | This method can be used for any kind of SQL statements. |
| This method returns a ResultSet object which contains the results returned by the query. | This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing. | This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing. |
| This method is used to execute only select queries. | This method is used to execute only non-select queries. | This method can be used for both select and non-select queries. |
| Ex : SELECT | Ex :<br>DML → INSERT, UPDATE and DELETE<br>DDL → CREATE, ALTER | This method can be used for any type of SQL statements. |

## Statement Vs PreparedStatement Vs CallableStatement

| Statement | PreparedStatement | CallableStatement |
|---|---|---|
| It is used to execute normal SQL queries. | It is used to execute parameterized or dynamic SQL queries. | It is used to call the stored procedures. |
| It is preferred when a particular SQL query is to be executed only once. | It is preferred when a particular query is to be executed multiple times. | It is preferred when the stored procedures are to be executed. |
| You cannot pass the parameters to SQL query using this interface. | You can pass the parameters to SQL query at run time using this interface. | You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT. |
| This interface is mainly used for DDL statements like CREATE, ALTER, DROP etc. | It is used for any kind of SQL queries which are to be executed multiple times. | It is used to execute stored procedures and functions. |
| The performance of this interface is very low. | The performance of this interface is better than the Statement interface (when used for multiple execution of same query) | The performance of this interface is high. |

same query).