




JDBC-Java Database Connectivity

JDBC

- JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database.
- It is a part of JavaSE (java.sql pkg)
- JDBC API uses JDBC drivers to connect with the database.
- There are four types of JDBC drivers:
 1. JDBC-ODBC Bridge Driver
 2. Native Driver
 3. Network Protocol Driver
 4. Thin Driver/ Pure Java Driver




JDBC interfaces

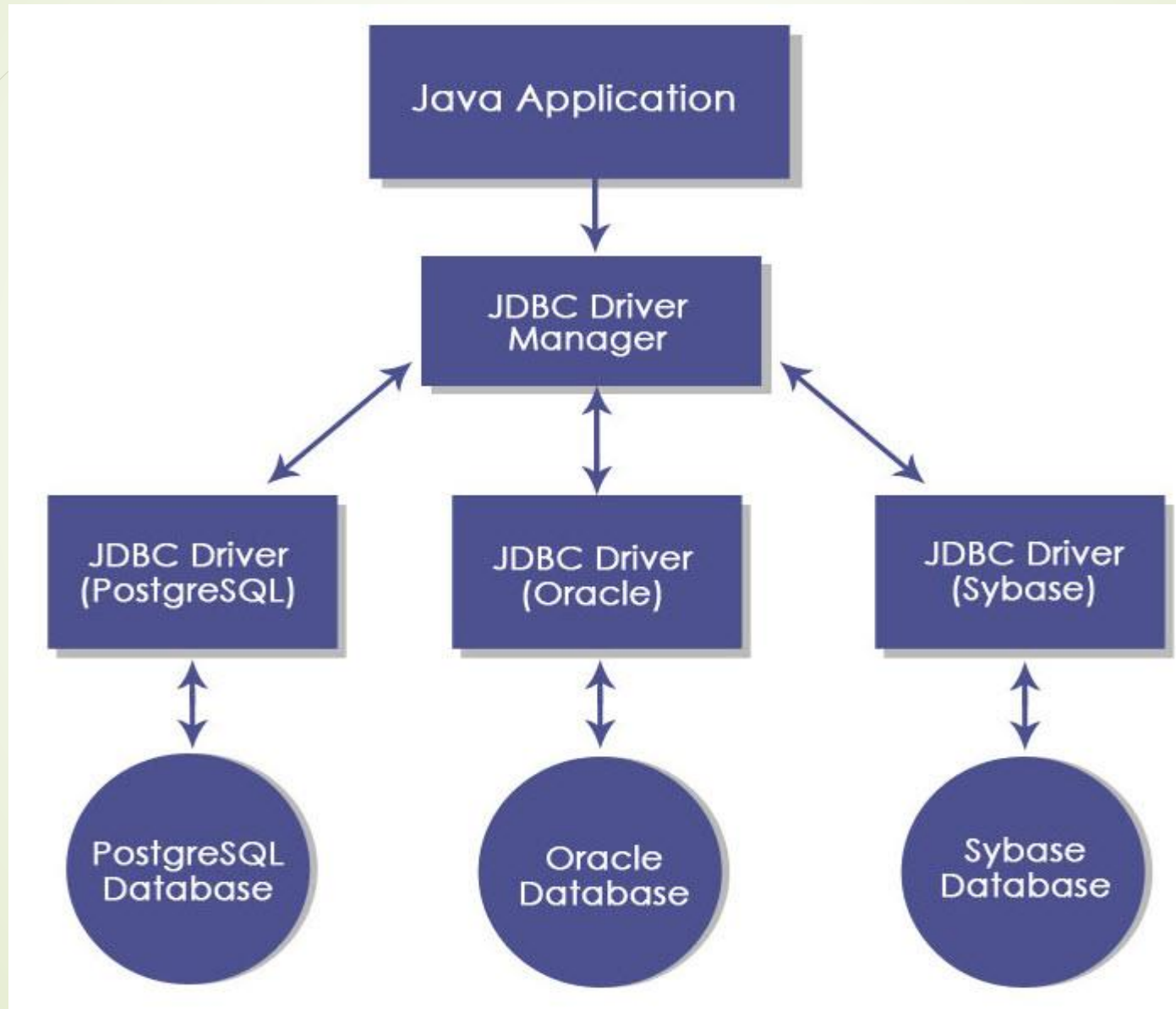
- ➡ Driver interface
 - ➡ Connection interface
 - ➡ Statement interface
 - ➡ PreparedStatement interface
 - ➡ CallableStatement interface
 - ➡ ResultSet interface
 - ➡ ResultSetMetaData interface
 - ➡ DatabaseMetaData interface
 - ➡ RowSet interface
- 



Why JDBC ?

1. Connect to the database
 2. Execute queries and update statements to the database
 3. Retrieve the result received from the database.
- 

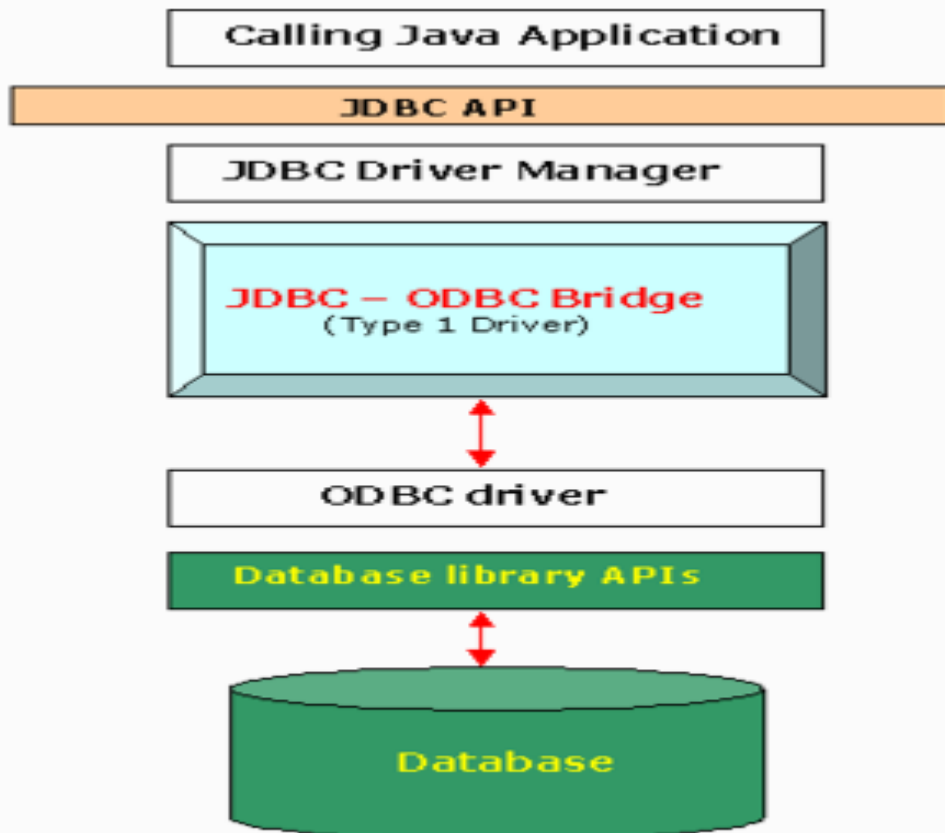
JDBC flow



JDBC Driver Types

1. JDBC-ODBC bridge driver

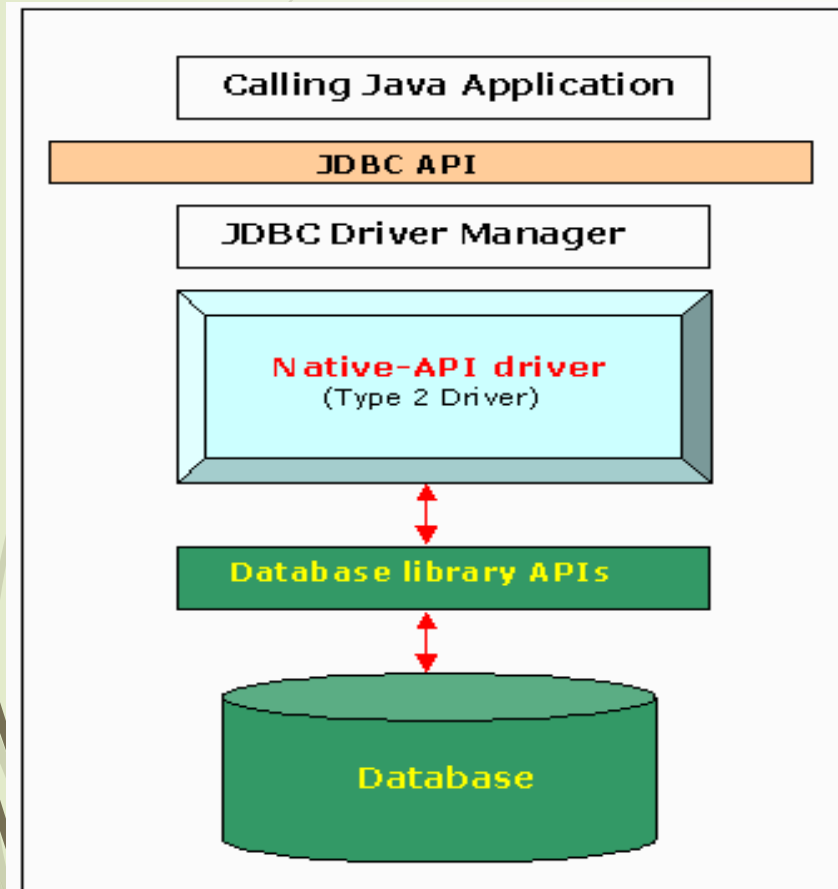
ODBC- Open Database Connectivity.



1. A type 1 JDBC driver consists of a Java part that **translates the JDBC interface calls to ODBC calls**. An ODBC bridge then calls the ODBC driver of the given database i.e. the driver converts JDBC method calls into ODBC function calls.
2. The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the underlying operating system the JVM is running upon
3. ODBC must be installed on the computer having the driver and the database must support an ODBC driver

JDBC Driver Types

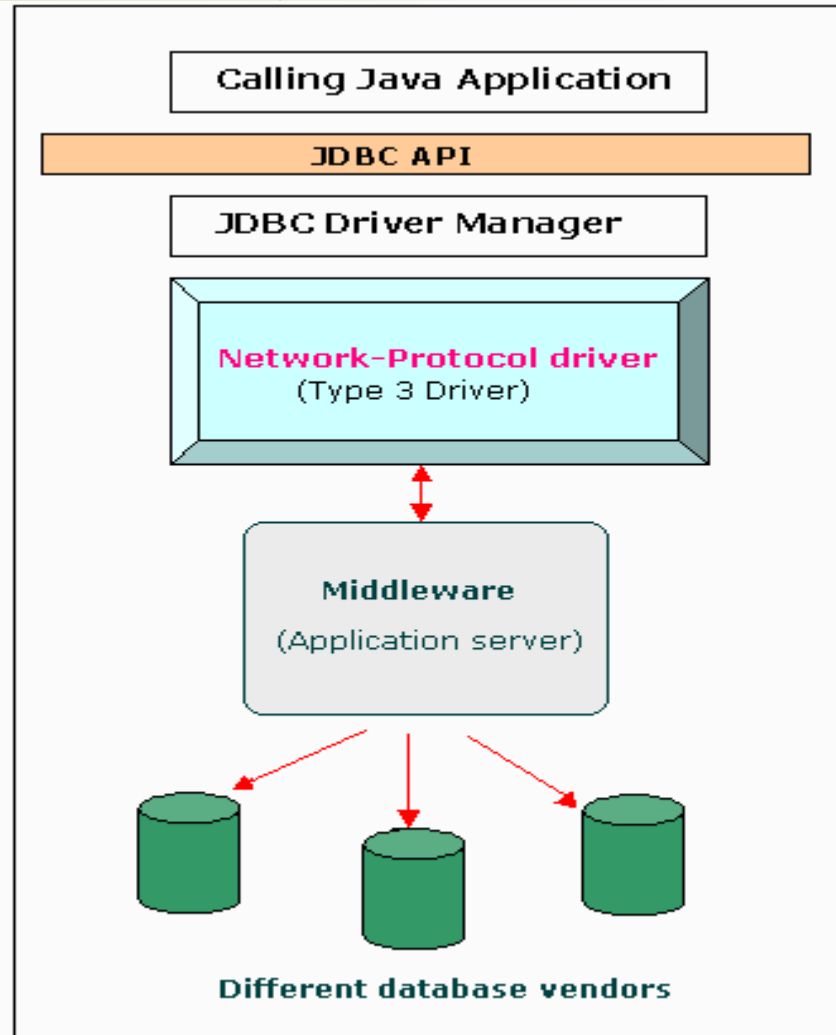
2. Native driver



1. A type 2 JDBC driver is like a type 1 driver, except the ODBC part is replaced with a native code part instead.
2. The native code part is targeted at a specific database product i.e. uses the client-side libraries of the database product. The driver converts JDBC method calls into native calls of the database native API.
3. This architecture eliminated the need for the ODBC driver and instead directly called the native client libraries shipped by the database vendors. This was quickly adopted by the DB vendors as it was quick and inexpensive to implement since they could reuse the existing C/ C++ based native libraries.

JDBC Driver Types

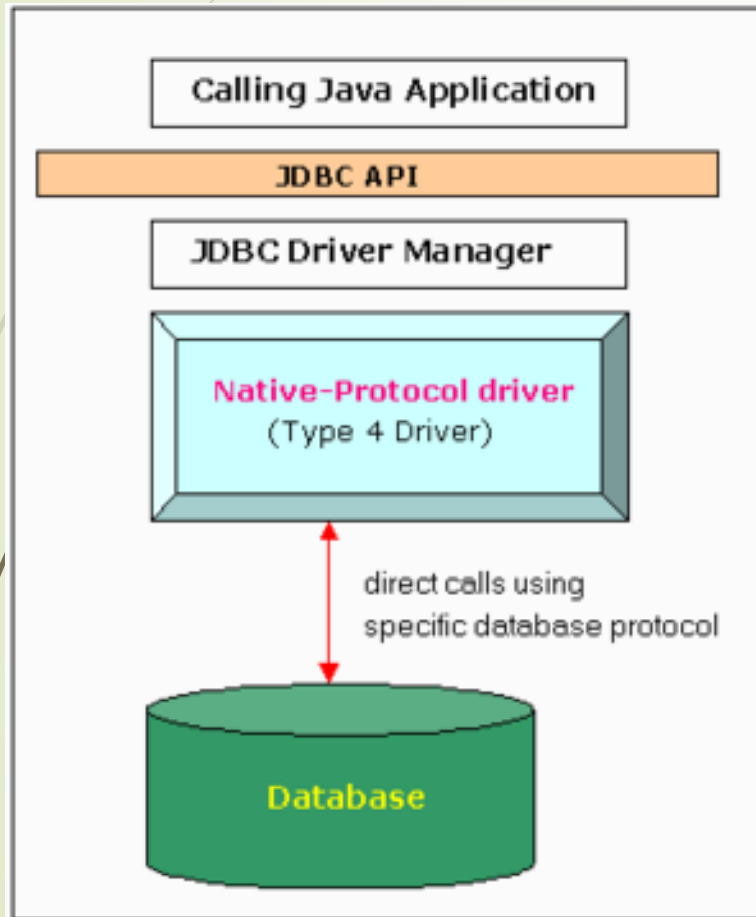
3. Network Protocol Driver



1. A type 3 JDBC driver is an all Java driver that sends the JDBC interface calls to an intermediate server. The intermediate server then connects to the database on behalf of the JDBC driver. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.
2. Type 3 drivers sought to be a 100% Java solution but never really gained much traction

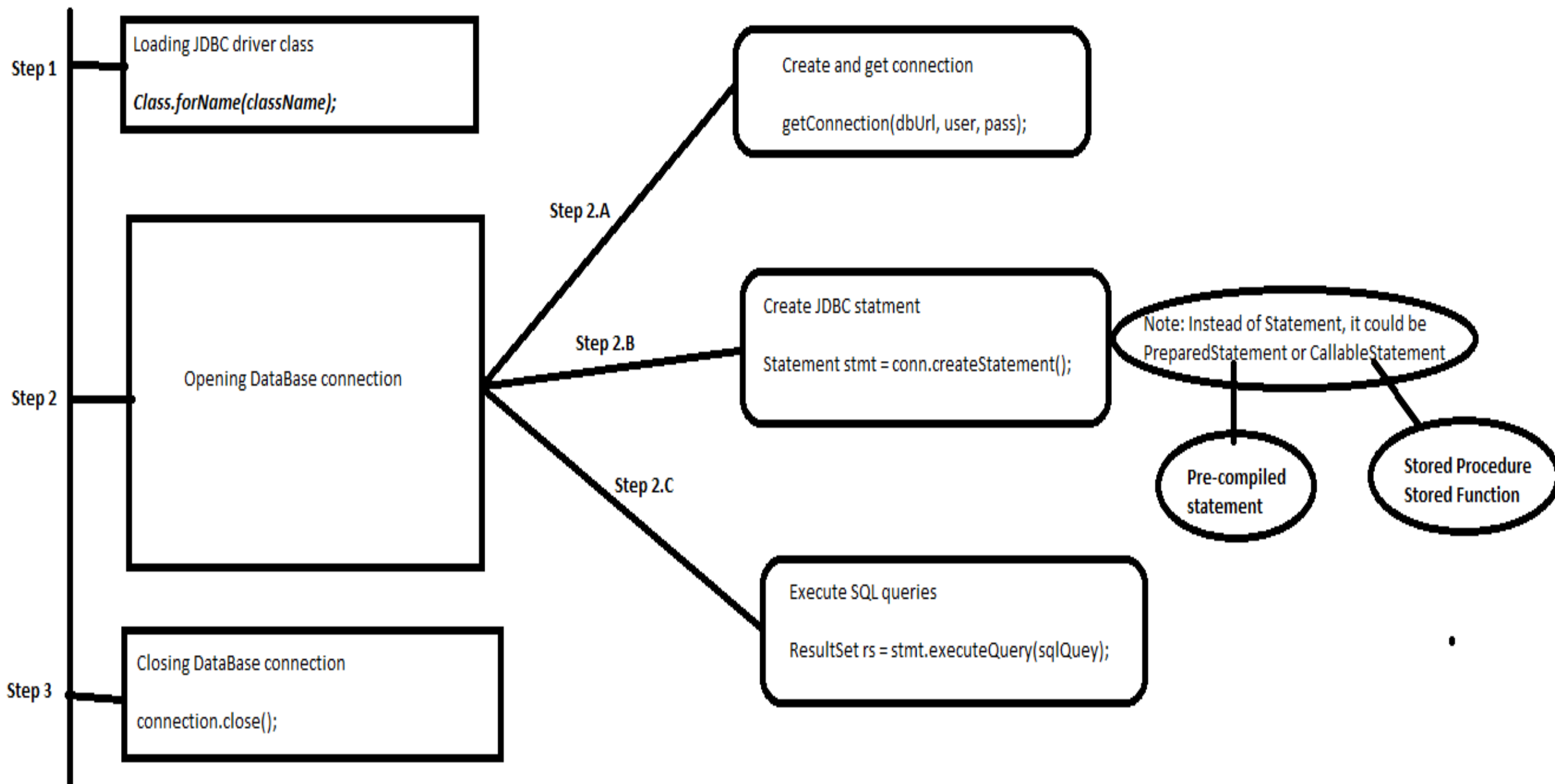
JDBC Driver Types

4. Pure Java Driver



1. The JDBC type 4 driver, also known as the **Direct to Database Pure Java Driver**, is a database driver implementation that converts JDBC calls directly into a vendor-specific database protocol. It is implemented for a specific database product. Today, most JDBC drivers are type 4 drivers.
2. Written completely in Java, type 4 drivers are thus platform independent. They install inside the Java Virtual Machine of the client

JDBC Steps



JDBC URL

Database URL for mysql

jdbc:mysql://localhost:3306/brndb

colon

colon

jdbc protocol

jdbc driver

Server Name or IP

Server port

Database Name

BenchResources.Net

JDBC Statements

Interface	Recommended Use
Statement	Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Result Set

- The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The *java.sql.ResultSet* interface represents the result set of a database query.
- A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.
- The methods of the ResultSet interface can be broken down into three categories
 - **Navigational methods** – Used to move the cursor around.
 - **Get methods** – Used to view the data in the columns of the current row being pointed by the cursor.
 - **Update methods** – Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

JDBC Transactions

- JDBC Connection is in **auto-commit** mode, which it is by default, then every SQL statement is committed to the database upon its completion.
- **auto-commit** can be turned off and manage your own transactions –
 - To increase performance.
 - To maintain the integrity of business processes.
 - To use distributed transactions.
- Transactions enable you to control if, and when, changes are applied to the database. It treats a single SQL statement or a group of SQL statements as one logical unit, and if any statement fails, the whole transaction fails.
- Connection object's `setAutoCommit()` method can be used to set `setAutoCommit` true or false