JDBC

By Rahul Barve

Objectives

- Introduction to JDBC
- Why JDBC
- JDBC Drivers
- JDBC Core API
- Executing Simple Queries
- Executing Parameterized Queries
- Transaction Management

Introduction to JDBC

By Rahul Barve

Introduction to JDBC

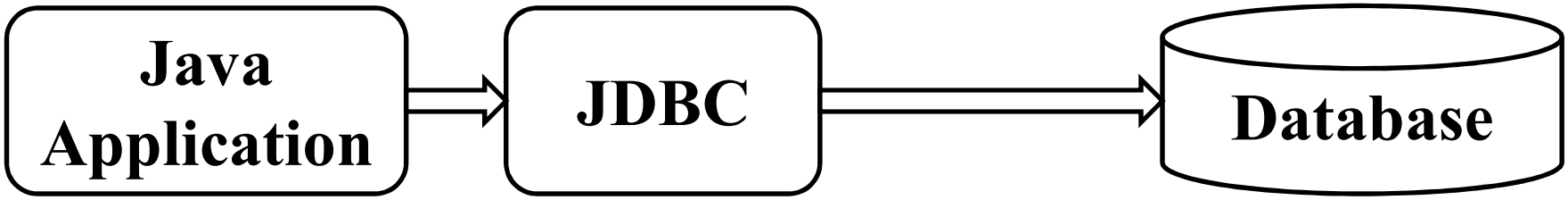
- JDBC stands for Java to Database Connectivity.
- It is an API that allows Java applications to interact with Database.

Why JDBC

Why JDBC

- Applications need to write data into or load data from database.
- JDBC provides a channel to bridge the gap between a Java application and a Database.

Why JDBC



JDBC Driver

By Rahul Barve

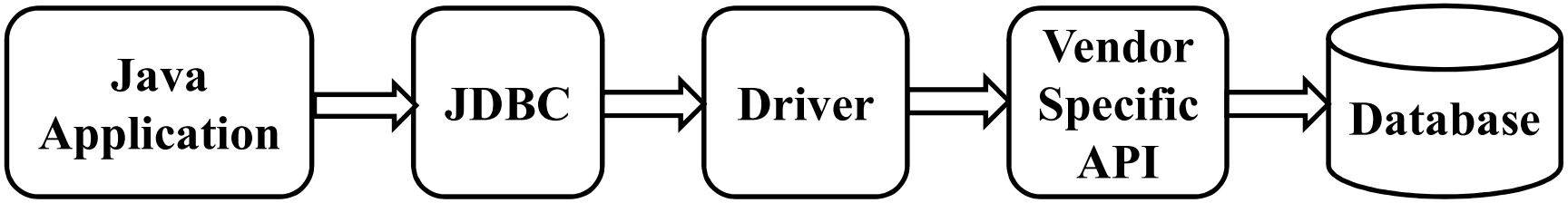
JDBC Driver

- Every DB vendor provides its own API that simplifies access for the client programs to connecting to the database.
- Such an API is known as a Vendor Specific API.

JDBC Driver

- Since both the APIs are written as per the proprietary standards, they are not compatible with each other.
- This mismatch is resolved by a mediator known as a Driver.

JDBC Driver


Java Application
JDBC
Driver
Vendor Specific API
Database

JDBC Driver

- There are 4 types of JDBC drivers:
  - Type 1
  - Type 2
  - Type 3
  - Type 4

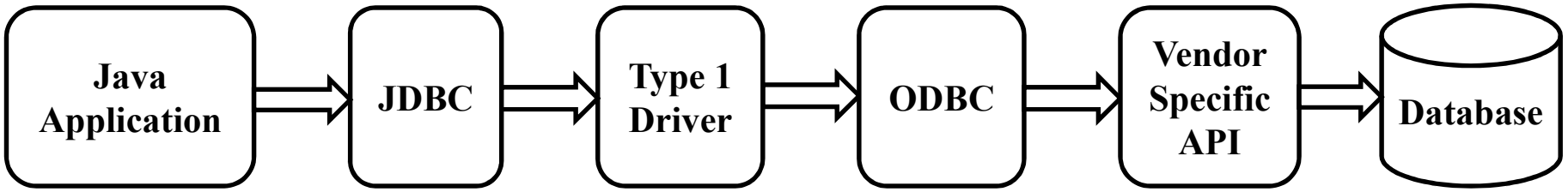
Type 1 Driver

By Rahul Barve

Type 1 Driver

- It is called as a JDBC – ODBC Bridge.
- It uses a 3rd party library known as ODBC which is provided by Microsoft.

Type 1 Driver



Type 1 Driver

- It is platform dependent.
- It is the slowest, takes much time for processing.
- Every client machine must have ODBC configuration setup.

Type 1 Driver

- Suitable for simple desktop applications or even just for testing purpose.
- Not much recommended in case of large scale applications or even in production environment.

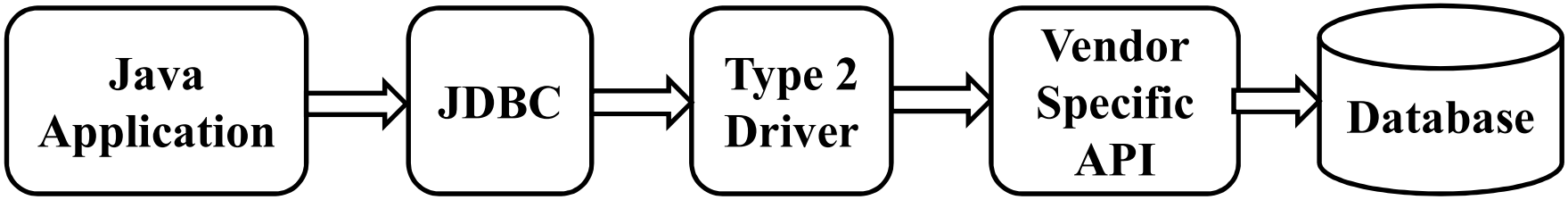
Type 2 Driver

By Rahul Barve

Type 2 Driver

- Native API, partly Java driver.
- It uses a combination of Java as well as Database proprietary standards for implementation.

Type 2 Driver



Type 2 Driver

- It does not use any 3rd party library and hence it is platform independent and faster in processing as compared to Type 1.

Type 2 Driver

- Since it uses a DB specific native API, the corresponding API must be installed on every client machine.

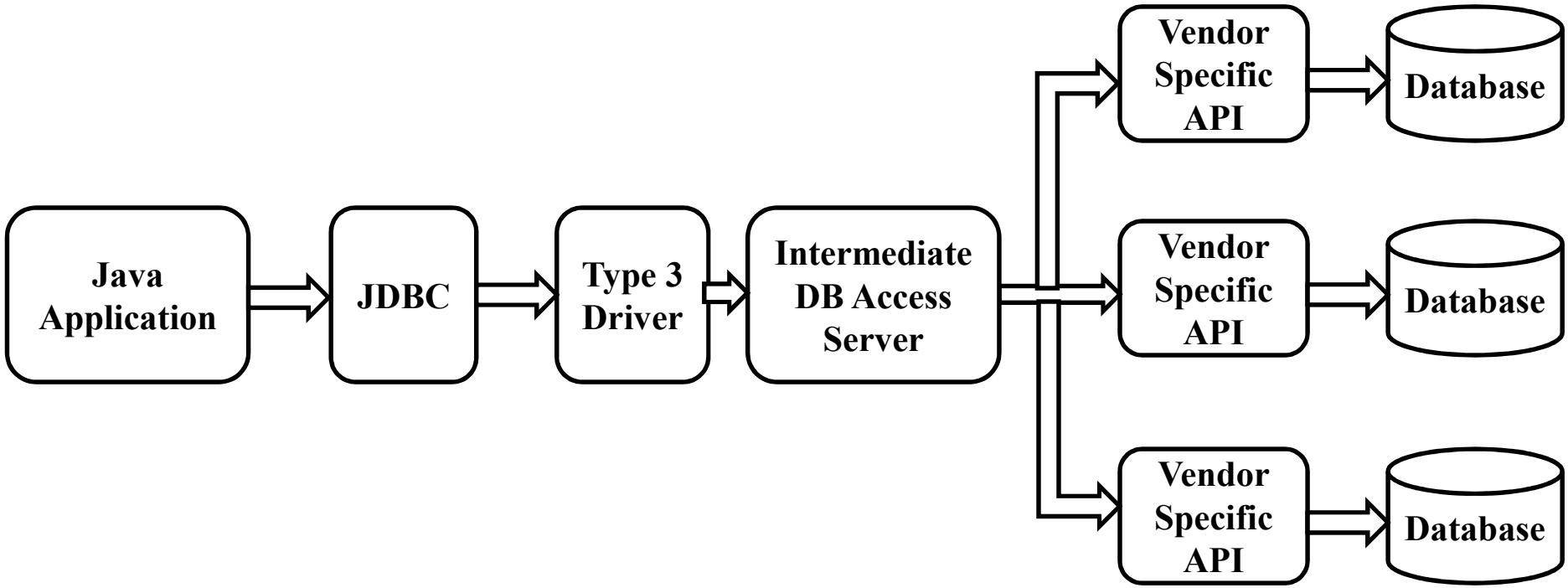
Type 3 Driver

By Rahul Barve

Type 3 Driver

- Net Protocol, Intermediate DB Access Server
- It is used especially when an application needs to connect to multiple databases.

Type 3 Driver



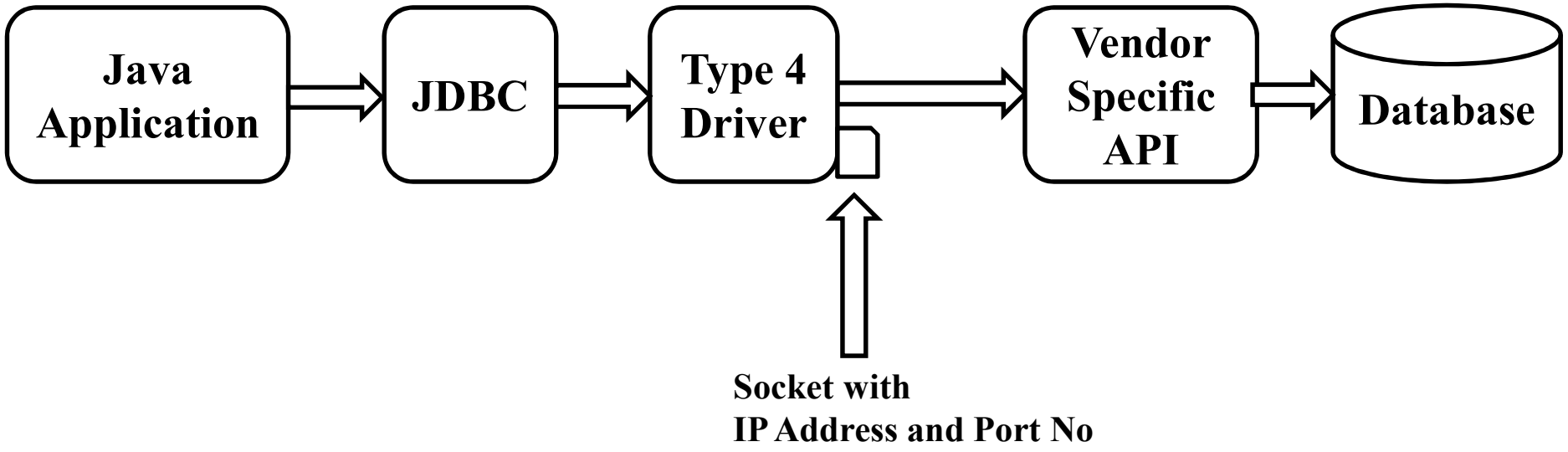
Type 4 Driver

By Rahul Barve

Type 4 Driver

- Database specific, Pure Java Driver.
- Every DB vendor provides its own driver implementation.
- Directly connects to a DB server using TCP/IP socket connections.

Type 4 Driver



Type 4 Driver

- It is the fastest as compared to Type 1 and Type 2 drivers.
- Platform independent.

Type 4 Driver

- No configuration is required on the client machine.
- Hence, highly recommended for large scale applications as well as production environment.

JDBC Core API

By Rahul Barve

JDBC Core API

- To implement any JDBC program, Java provides an API known as a JDBC API.
- It belongs to a package `java.sql`.

JDBC Core API

- `DriverManager`
- `Driver`
- `Connection`
- `Statement`
- `PreparedStatement`
- `CallableStatement`
- `ResultSet`

Steps in JDBC Application

By Rahul Barve

Steps in JDBC Application

- Load the Driver.
- Establish `Connection`.
- Obtain the `Statement`.
- Execute SQL query.
- (For `SELECT` query) Obtain the `ResultSet` and perform navigation.

Load the Driver

By Rahul Barve

Load the Driver

- A driver can be loaded either by using `Class.forName()` or by creating an object of the driver implementation class.

Establish Connection

By Rahul Barve

Establish `Connection`

- A Connection to the database can be established either by using a `DriverManager` class or a `Driver` interface.

Establish Connection

- E.g.

```
Connection conn =
     DriverManager.getConnection(….);
```

OR

```
Driver dr =
    new <<DriverImplClassNAME>>();
Connection conn = dr.connect(…);
```

Obtain the `Statement`

Obtain the `Statement`

- Once a connection is established, depending upon the type of the operation, a statement needs to be obtained.

Obtain the `Statement`

- `Statement` is used to execute simple queries.

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

Execute SQL Query

By Rahul Barve

Execute SQL Query

- `Statement` interface provides relevant methods to execute SQL queries.
- To execute `SELECT` query, `executeQuery()` method is used that returns `ResultSet`.

Execute SQL Query

- ```
  String sqlQuery = "select ….";
  ResultSet rs =
          stmt.executeQuery(sqlQuery);
  ```
  ```

Perform Navigation

By Rahul Barve

Perform Navigation

- `ResultSet` maintains data fetched from database in a tabular format.
- Every column has a column index and a row has a record position.

Perform Navigation

Before First	1	2	3
1			
2			
3			
After Last			

Perform Navigation

- By default, the cursor position of `ResultSet` points to `BeforeFirst`.
- To move in the forward direction, `next()` method is used.