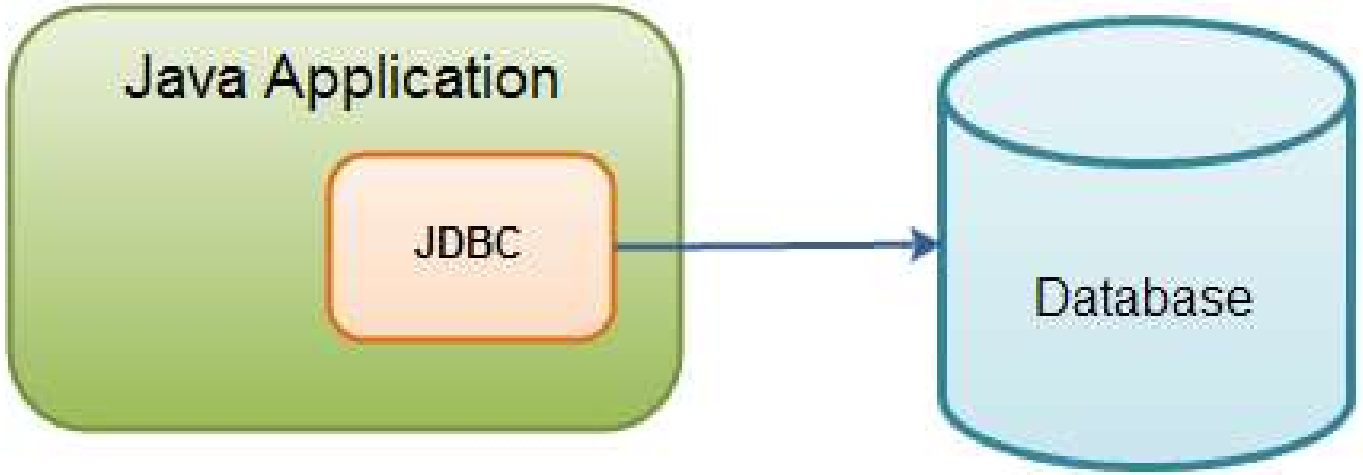
JDBC

Ajay Khatri
Senior Assistant Professor
Information Technology Department
Acropolis Institute of Technology & Research
ajay.acropolis@gmail.com

JDBC

- JDBC - Java Database Connectivity
- It is a standard Java API for database-independent connectivity between the Java program and a wide range of databases.



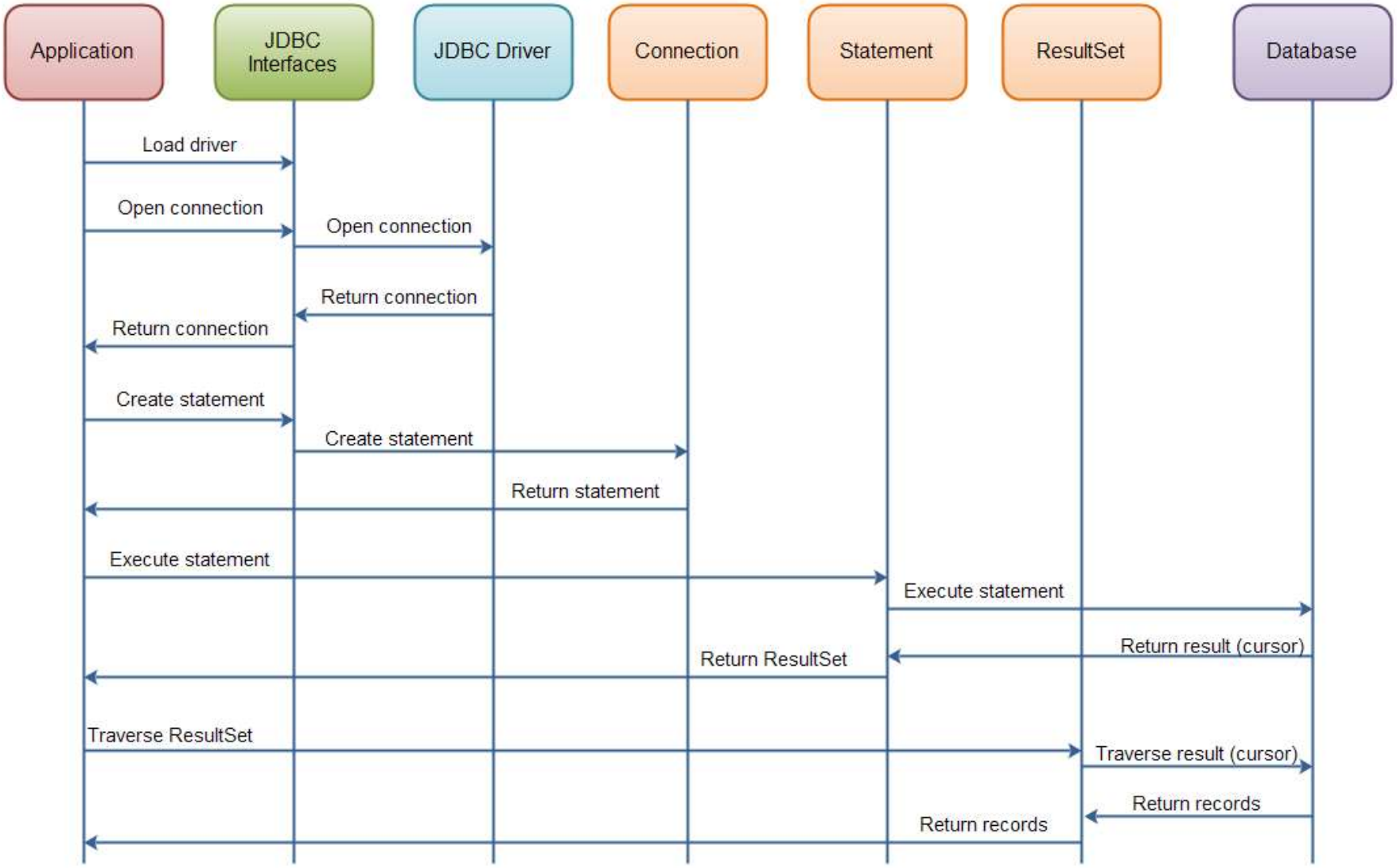
JDBC standardizes

- How to connect to a database.
- How to execute queries against it.
- How to navigate the result of such a query.
- How to execute updates in the database.
- JDBC does not standardize the SQL sent to the database. This may still vary from database to database.

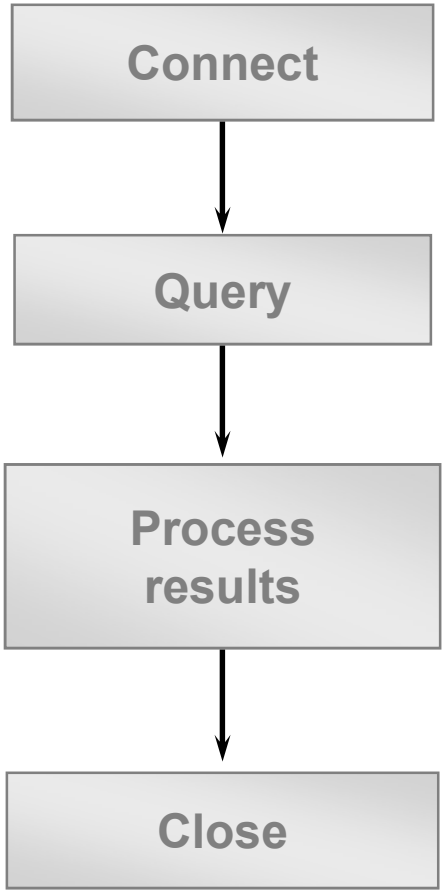
JDBC API : Core Parts

- JDBC Drivers
  - A JDBC driver is a collection of Java classes that enables you to connect to a certain database.
- Connections
  - Once a JDBC driver is loaded and initialized, you need to connect to the database.
- Statements
  - A Statement is what you use to execute queries and updates against the database.
- Result Sets
  - When you perform a query against the database you get back a ResultSet.

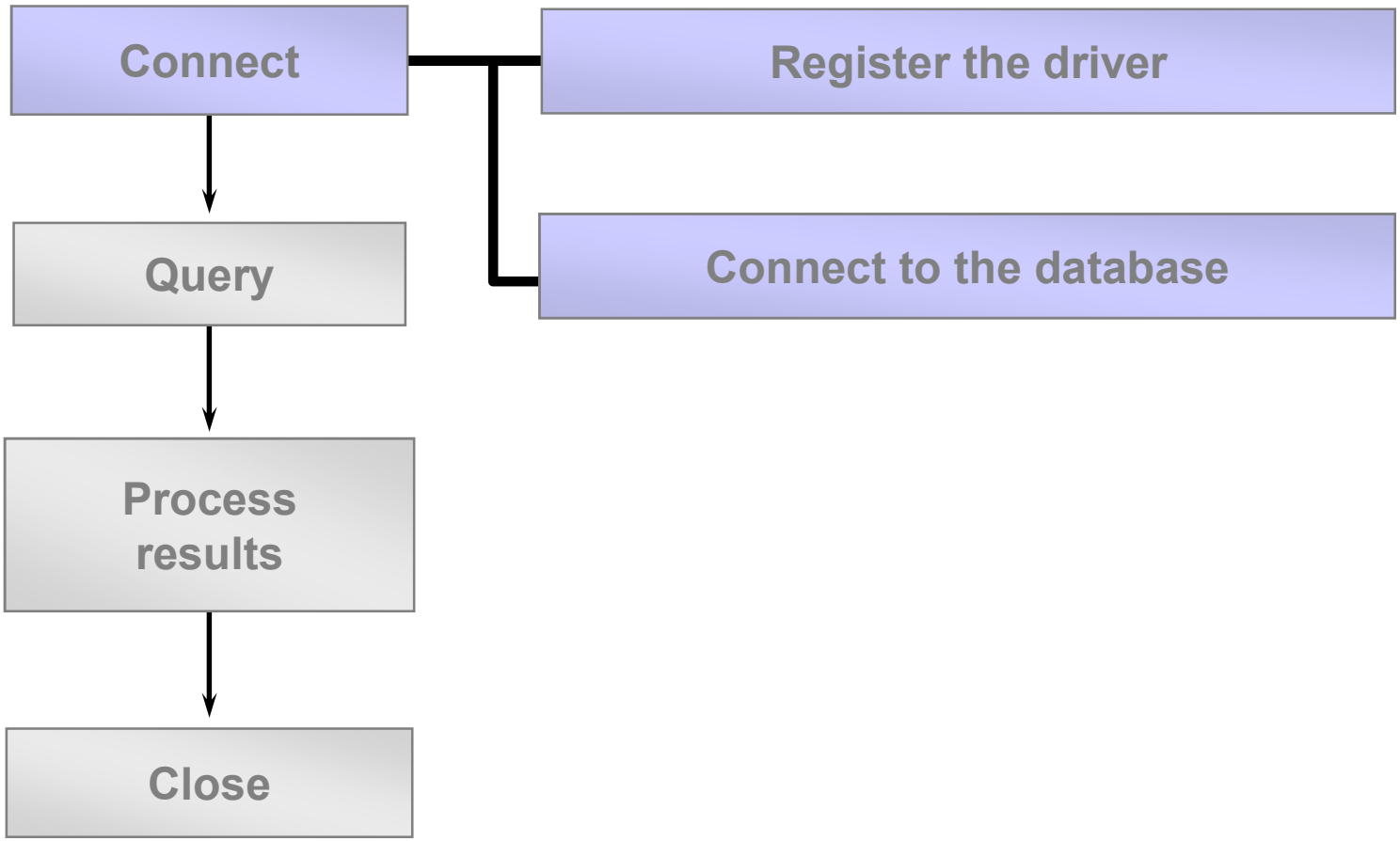
JDBC Component Interaction Diagram



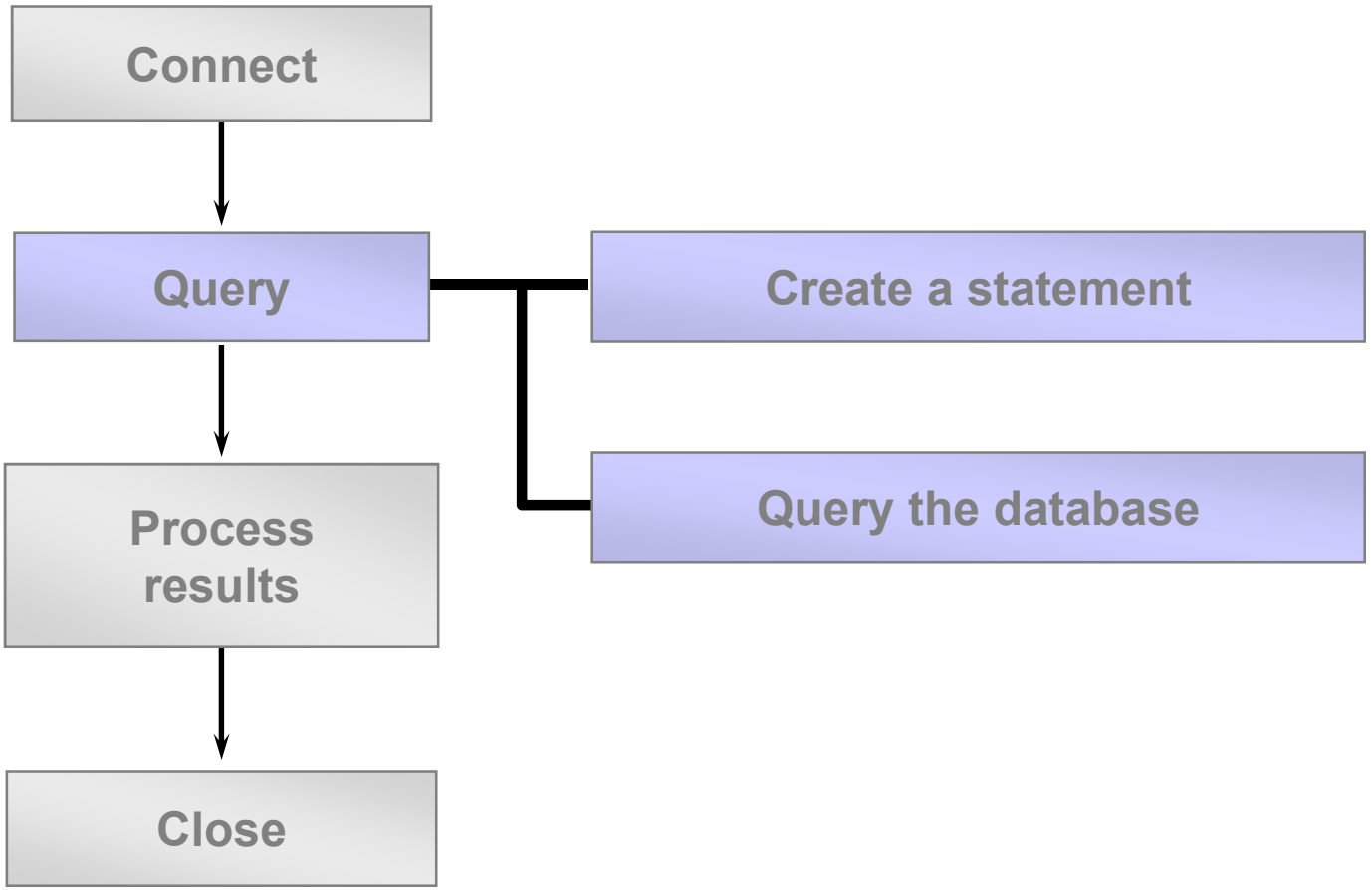
Overview of Querying a Database With JDBC



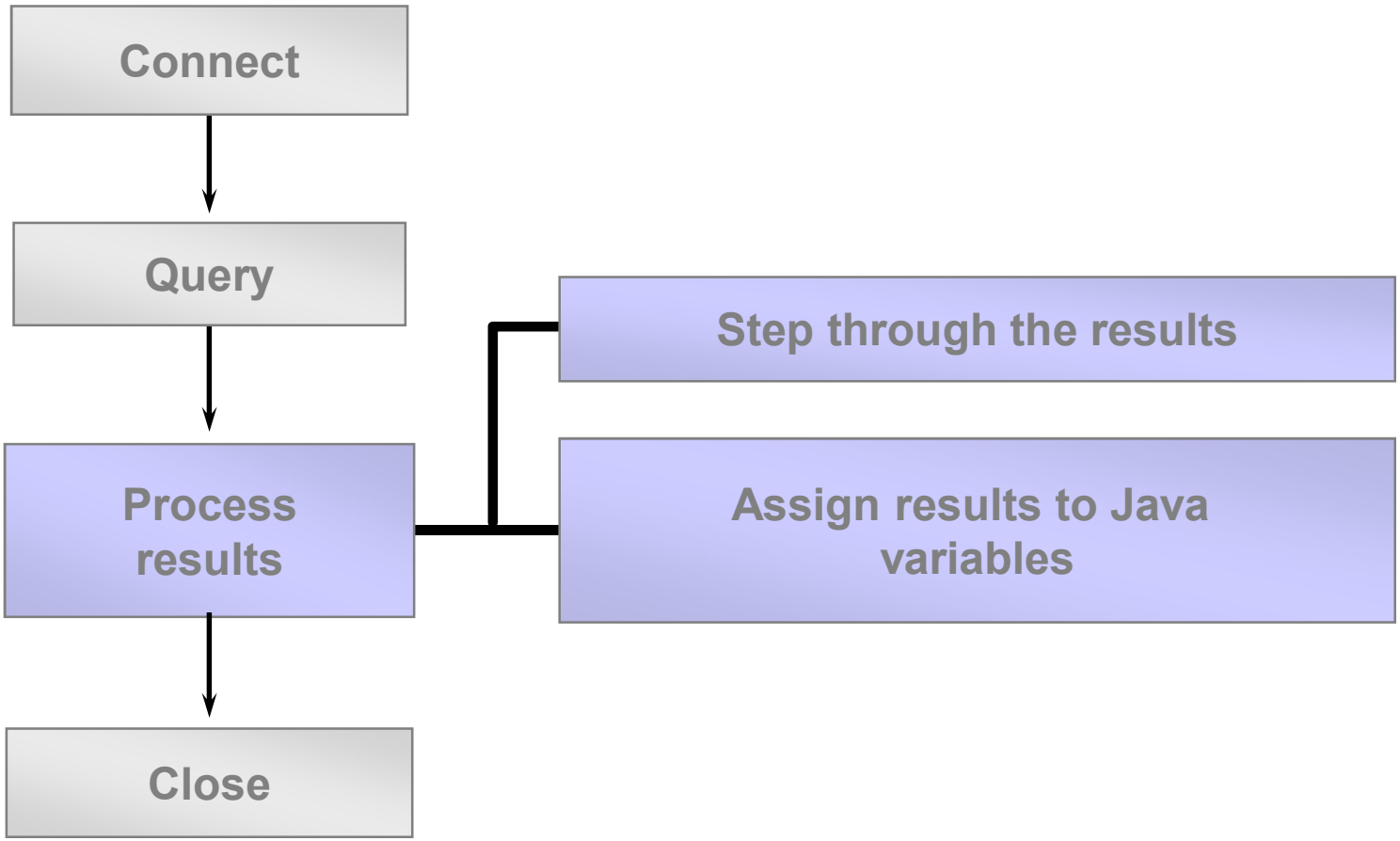
Stage 1: Connect



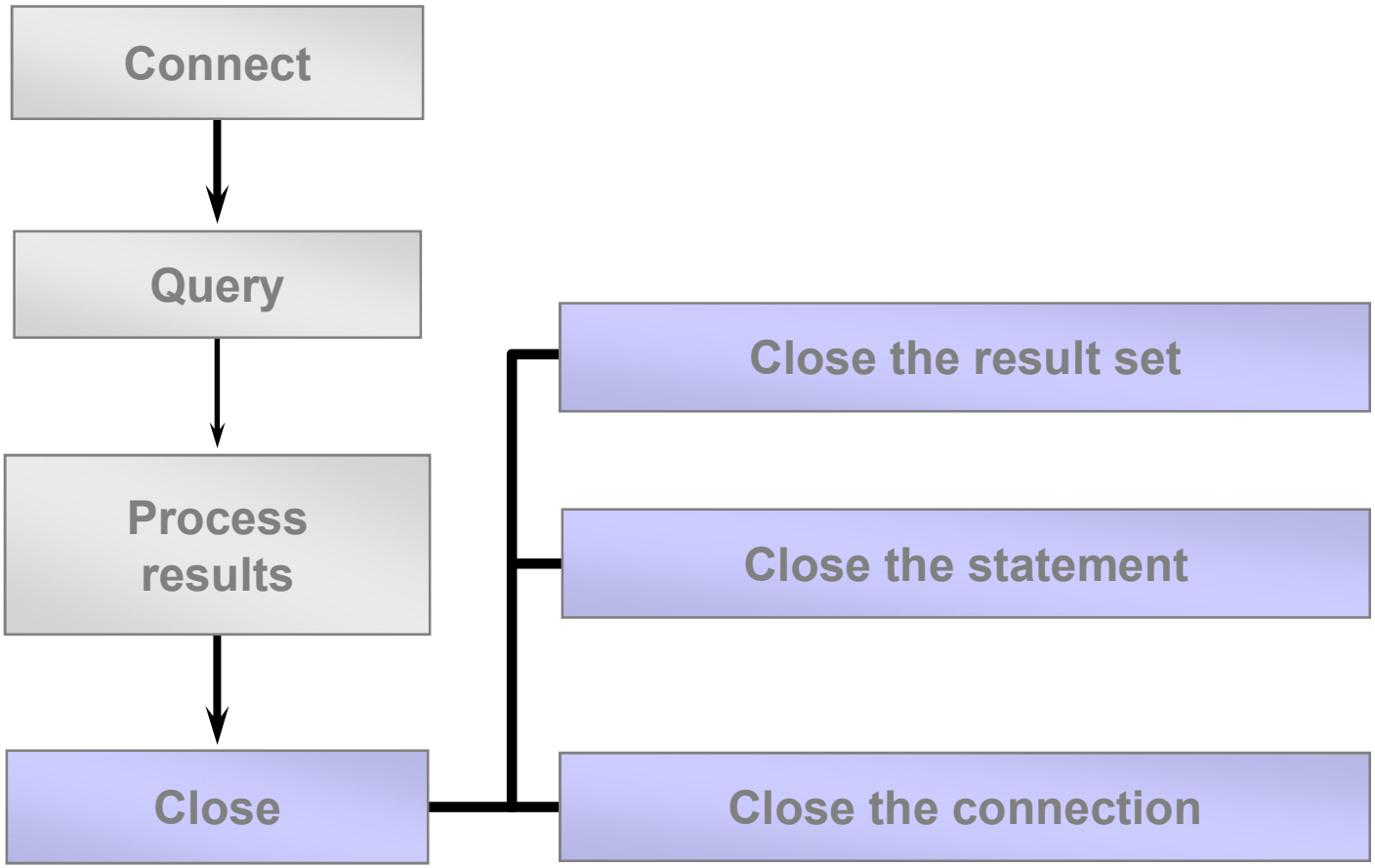
Stage 2: Query



Stage 3: Process Results



Stage 4: Close



Creating JDBC Application

- **Import the packages:** *import java.sql.**
- **Register the JDBC driver:** Requires that you initialize a driver so you can open a communication channel with the database.
- **Open a connection:** *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database.
- **Execute a query:** Requires using an object of type Statement for building and submitting an SQL statement to the database.
- **Extract data from result set:** Requires that you use the appropriate *ResultSet.getString(), ResultSet.getXYZ()* method to retrieve the data from the result set.
- **Clean up the environment:** Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

JDBC Driver

- A JDBC driver is a set of Java classes that implement the JDBC interfaces, targeting a specific database.
- The JDBC interfaces comes with standard Java, but the implementation of these interfaces is specific to the database you need to connect to. Such an implementation is called a JDBC drive

JDBC Driver : Type 1

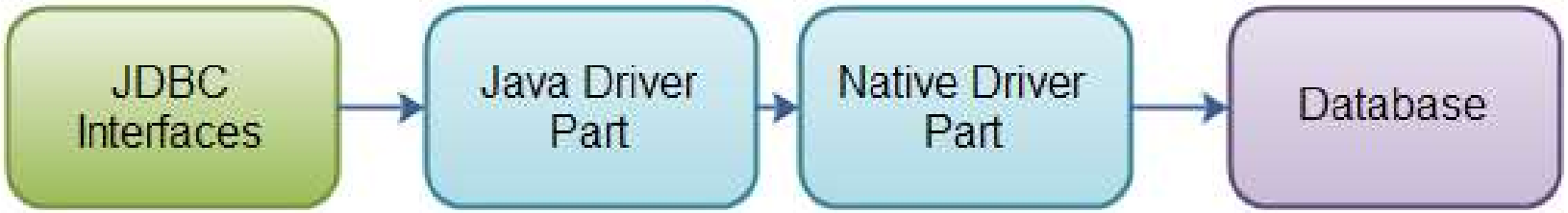
JDBC-ODBC bridge driver

- 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.
- Mostly intended to be used in the beginning, when there were no type 4 drivers (all Java drivers).
- Requires installation/configuration on client machines
- Not good for Web

JDBC Driver : Type 2

Java + Native code driver

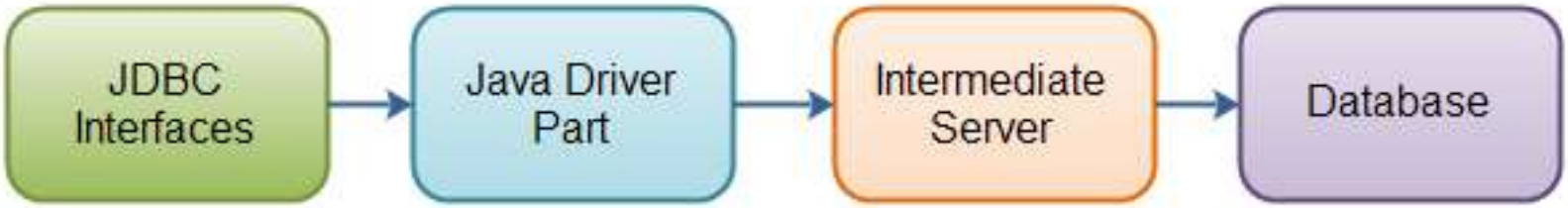
- A type 2 JDBC driver is like a type 1 driver, except the ODBC part is replaced with a native code part instead. The native code part is targeted at a specific database product.
- Requires installation/configuration on client machines
- Mostly obsolete now



JDBC Driver : Type 3

All Java + Middleware translation driver

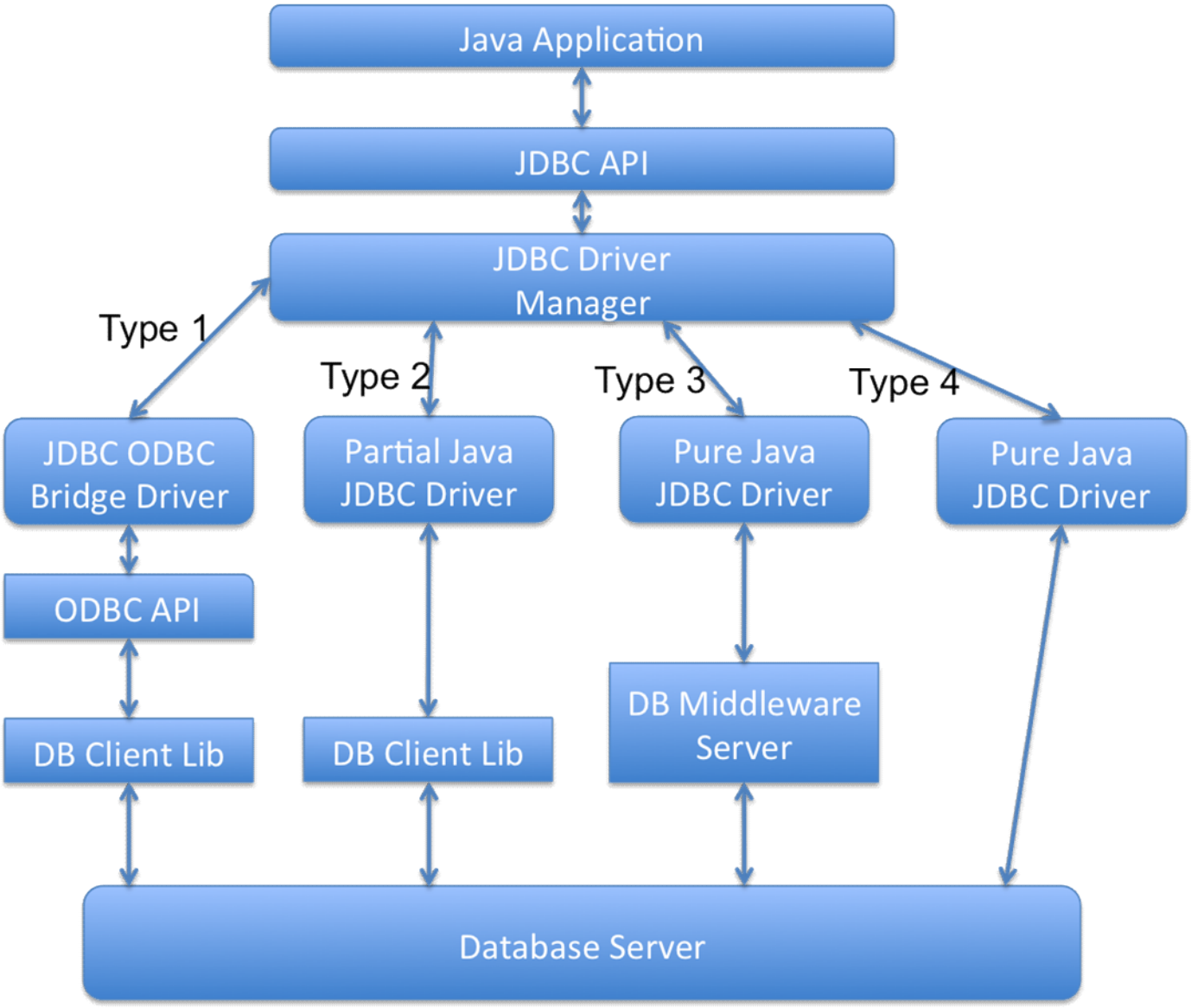
- A type 3 JDBC driver is an all Java driver that sends the JDBC interface calls to an intermediate server. The intermediate server then connects to the database on behalf of the JDBC driver.
- Very flexible & allows access to multiple databases using one driver



JDBC Driver : Type 4

- Class-IV: 100% Pure Java
  - A type 4 JDBC driver is an all Java driver which connects directly to the database. It is implemented for a specific database product.
  - Today, most JDBC drivers are type 4 drivers.
  - This kind of driver is extremely flexible, you don't need to install special software on the client or server.

JDBC Driver Types



Which Driver should be Used?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver **type is 4.**
- If your Java application is accessing multiple types of databases at the same time, **type 3** is the preferred driver.
- **Type 2** drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The **type 1** driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

JDBC : Connect with MySQL DB

- **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
- **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/mydb**
  - where jdbc is the API,
  - mysql is the database,
  - localhost is the server name on which mysql is running, we may also use IP address,
  - 3306 is the port number
  - mydb is the database name.
- **Username:** username ("root ")
- **Password:** dbpassword ("123")

JDBC : Create DB, Table

- create database sonoo;
- use sonoo;
- create table emp(id **int**(10),name varchar(40),age **int**(3));

JDBC : Example Code

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/test", "root", "");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
           System.out.println(rs.getInt(1) + "  " + rs.getString(2));
        }
con.close();
```

PreparedStatement interface

- Easy to insert parameters into the SQL statement.
- Easy to reuse the PreparedStatement with new parameters.
- May increase performance of executed statements.
- Enables easier batch updates.

PreparedStatement interface

```
PreparedStatement ps;
ps =   con.prepareStatement("insert into emp values(?,?)");
ps.setInt(1, 101);
ps.setString(2, “Ajay");
ps.executeUpdate();
ps.setInt(1, 102);
ps.setString(2, “Vikas");
ps.executeUpdate();
```

Thanks