HEXAVARSITY
Enhancing Excellence

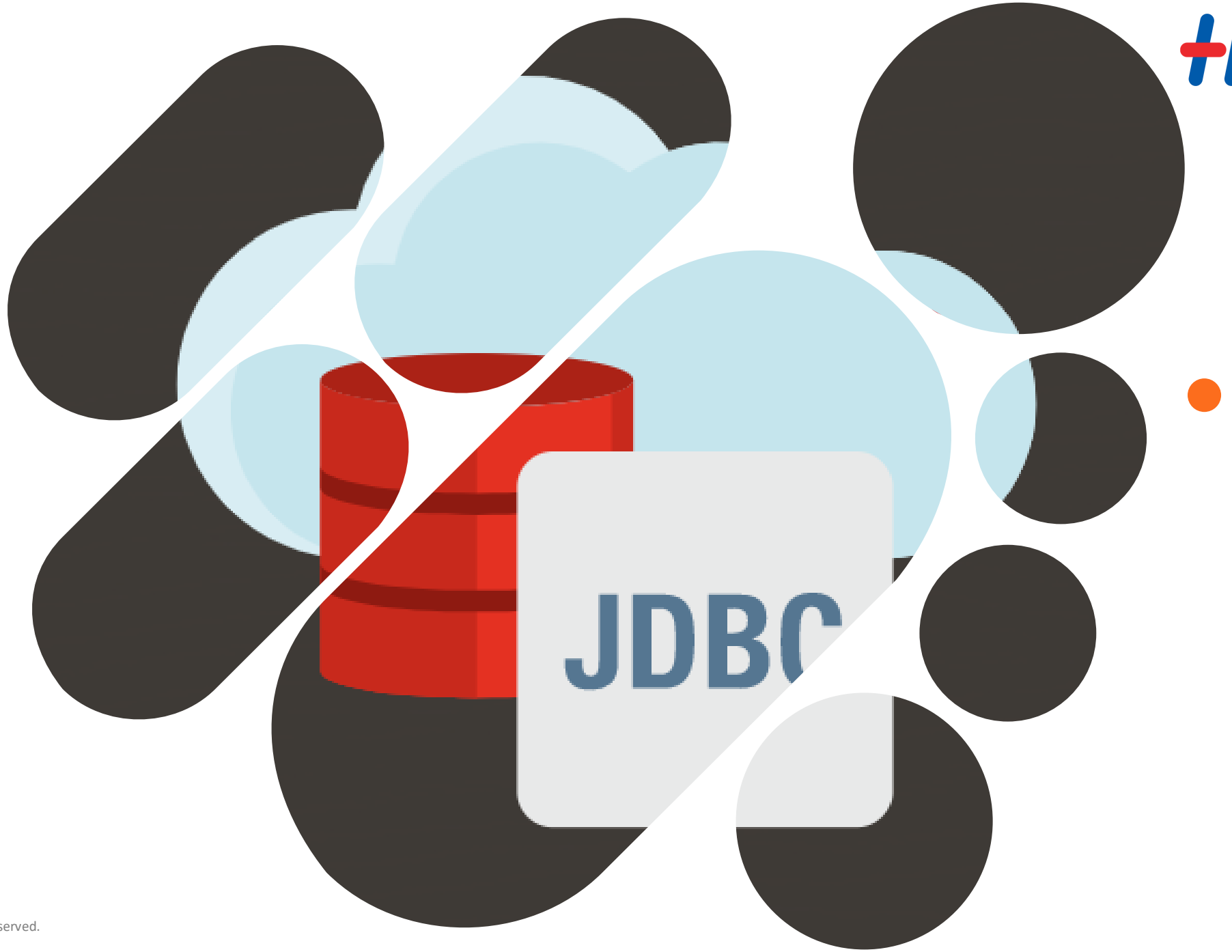
Java Programming Basic level

Java Data Base Connectivity

Session Plan

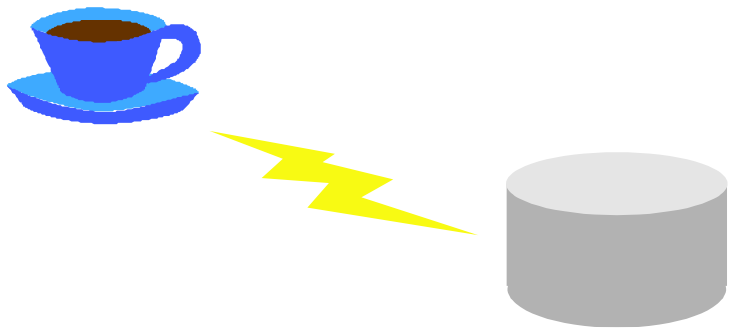
- JDBC
- JDBC Architecture and types of JDBC Drivers
- Accessing Database through JDBC- steps
- Statement and its type
- ResultSet and ResultSetMetadData
- Transaction Control through JDBC

JDBC



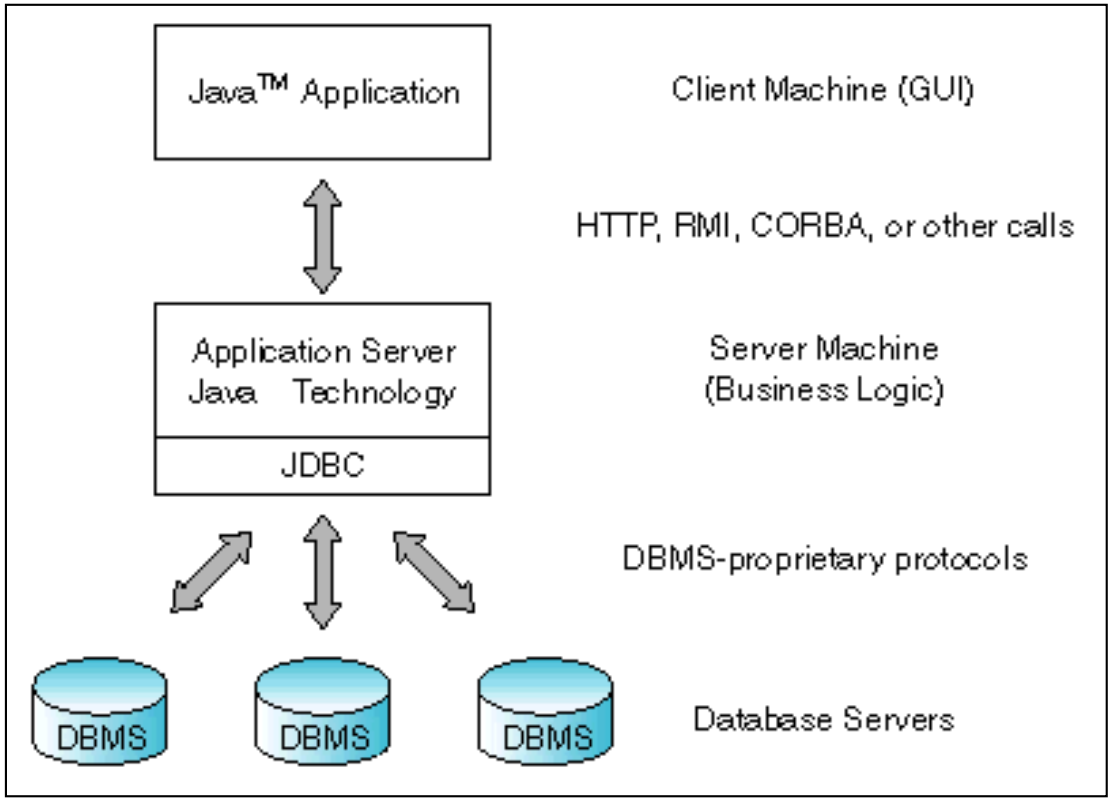
Overview - Java Database Connectivity

- JDBC is a standard interface for connecting to relational databases from Java by embedding SQL inside Java code
- JDBC is a Java API for executing SQL statements and supports basic SQL functionality
- Using JDBC you can send SQL, PL/SQL statements to almost any relational database.



JDBC Architecture

- **Application**
  - Uses java.sql API to retrieve/query a database
- **Database**
  - A repository system for organizing data in a structured way
- **Database** Driver
  - A separate entity which provides interface between the Application and Database.



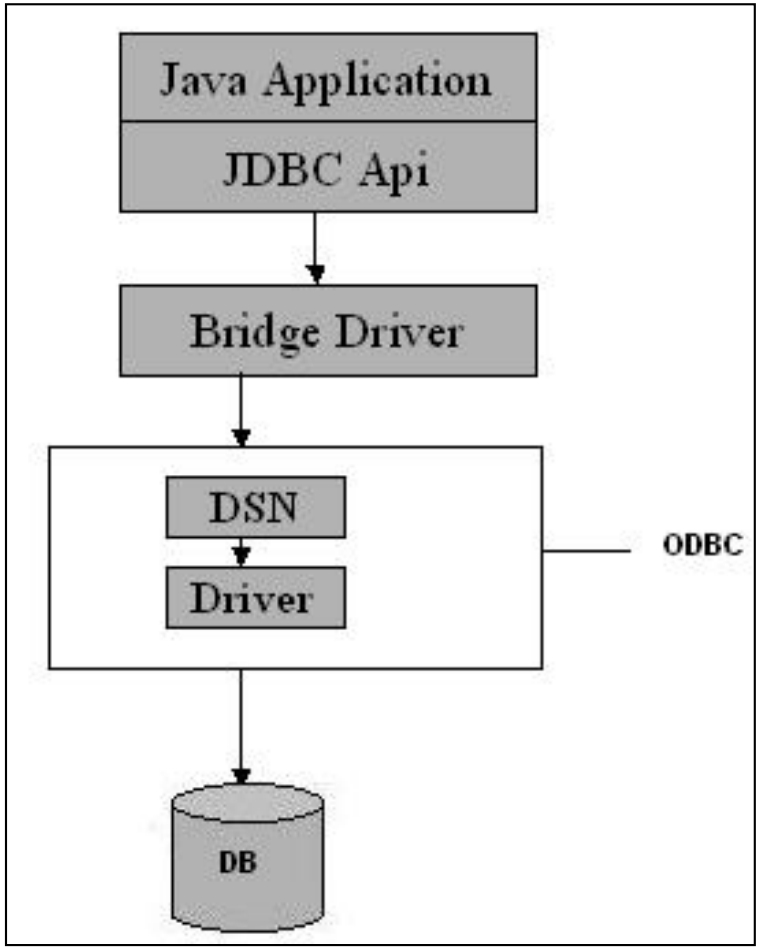
Types of JDBC Drivers

- JDBC-ODBC Bridge driver (Bridge)
- Native-API/partly Java driver (Native)
- All Java/Net-protocol driver (Middleware)
- All Java/Native-protocol driver (Pure)

Type 1: JDBC-ODBC Bridge Driver

Translates all JDBC calls into ODBC calls and sends them to the ODBC driver:

- Advantage
  - The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available
- Disadvantage
  - Type 1 drivers are not portable
  - Performance - very Slow
  - Client requires ODBC installation
  - Not good for Web

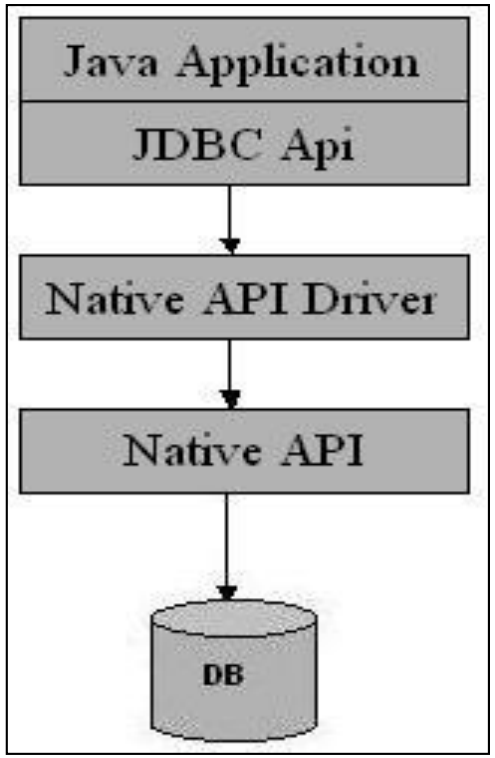


Type 2: Native-API/partly Java Driver

- Converts JDBC calls into database-specific calls
- The driver is specific to a particular database. Example: Oracle will have oracle native API.

- Advantage
  - Better performance - Less layers of communication and native drivers

- Disadvantage
  - Native API must be installed in the Client System -hence cannot be used for internet
  - Portability issue (not written in Java)
  - Native driver's are database dependent
  - Not thread safe

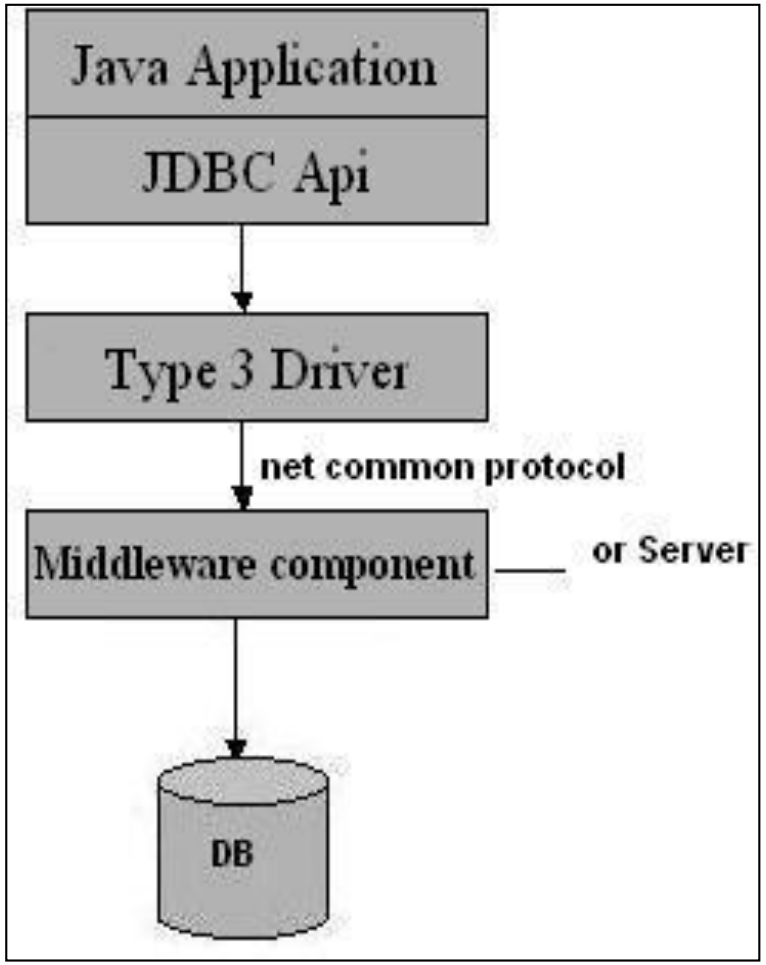


Type 3: All Java/Net-protocol Driver

- Requests are passed through the network to the middle-tier server.
- The middle-tier translates the request to the database.

- Advantage
  - Driver is server-based, so there is no need for any vendor database library to be present on client machines.
  - Portable and suitable for web
  - Portability, performance, and scalability can be optimized
  - Supports features such as caching, load balancing and advanced system administration such as logging and auditing
  - access to multiple databases using one driver

- Disadvantage
  - Requires another server application to install and maintain
  - Traversing the recordset may take longer, since the data comes through the backend server

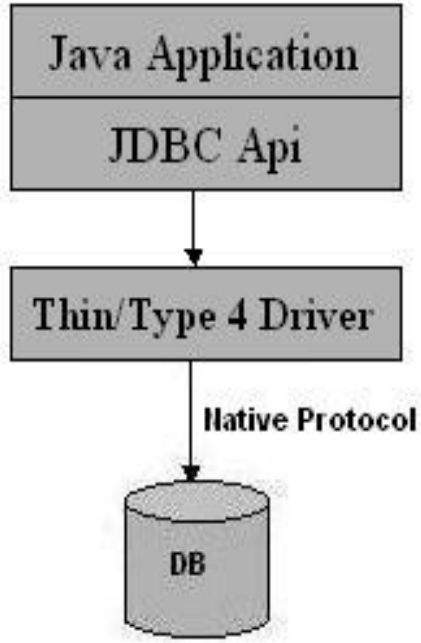


Type 4: Native-protocol/all-Java Driver

- Uses java networking libraries to communicate directly with the database server.

- Advantage
  - Platform independent since written in Java
  - Performance quite good
  - No special software on the client or server

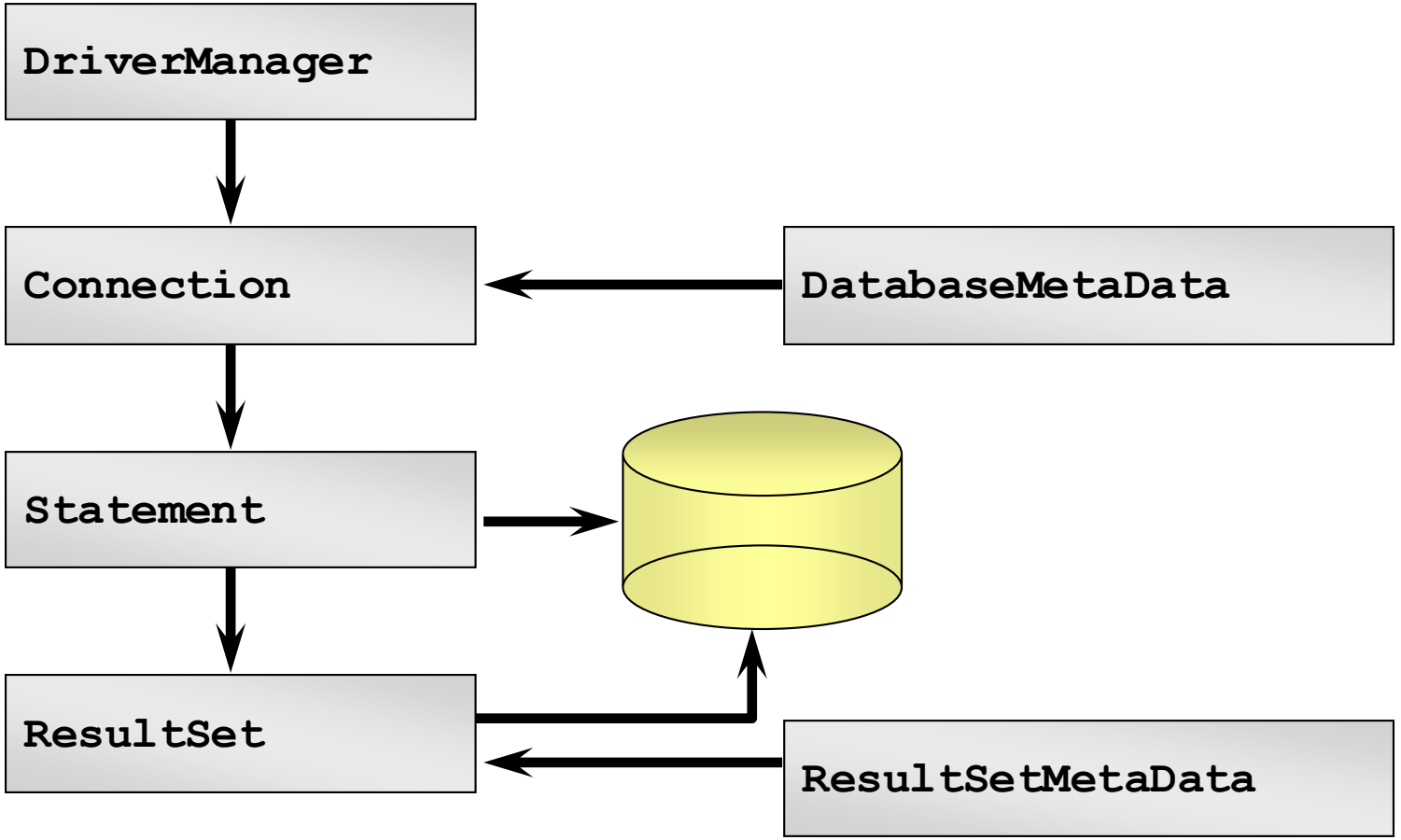
- Disadvantage
  - Different driver for each database



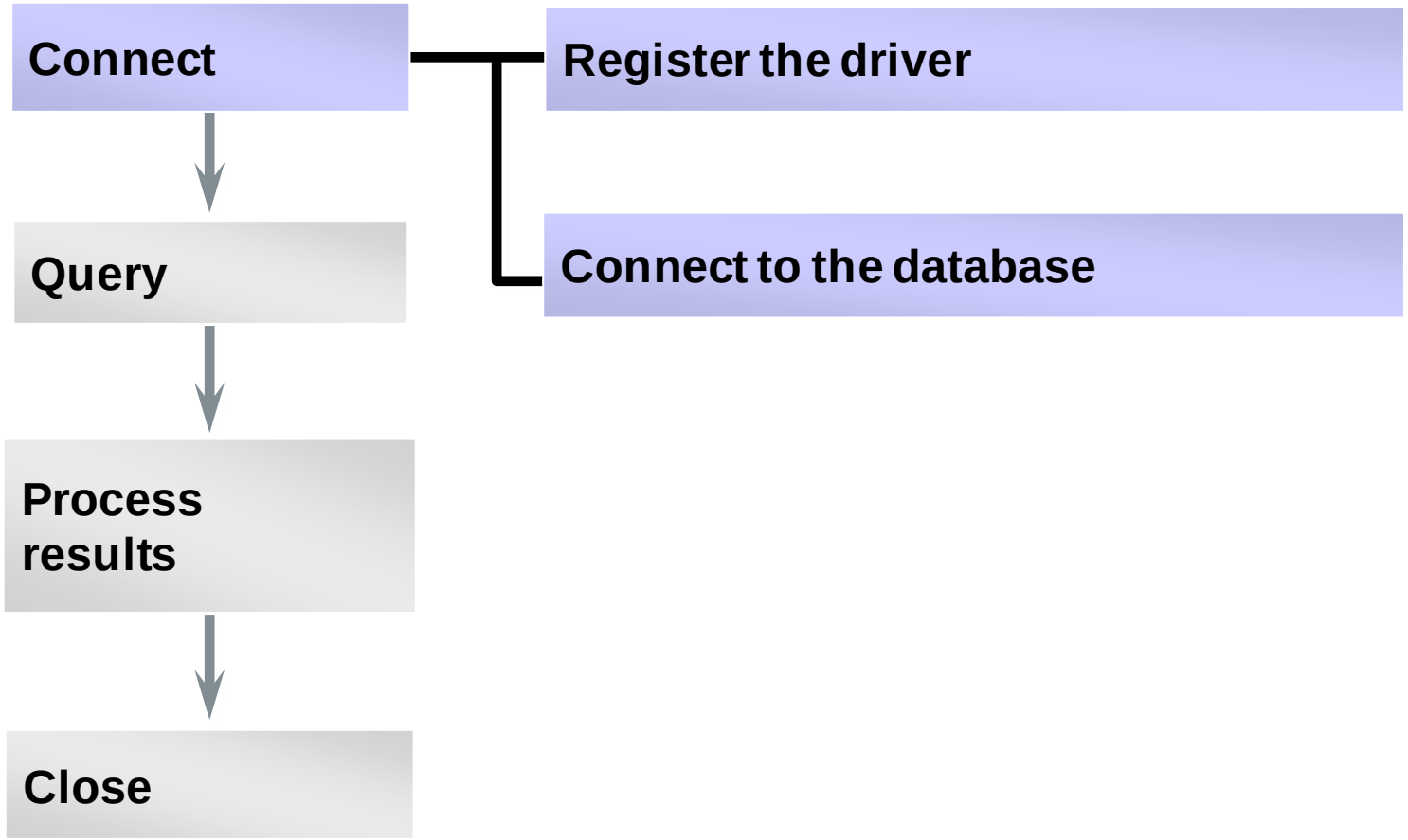
Java Database Connectivity Steps

- Import the java.sql package.
- Create a data source name using ODBC
- Register the Driver
- Establish a Connection to the database
- Create a Statement object
- Execute SQL Query statement(s)
- Retrieve the ResultSet Object
- Retrieve record/field data from ResultSet object for processing
- Close ResultSet Object
- Close Statement Object
- Close Connection Object

Java Database Connectivity Steps



Stage1: Connection establishment



Register the Driver

- Load the driver class by calling Class.forName() with the Driver class name as an argument.
- The Driver class creates an instance of itself.
- The return type of the Class.forName(String ClassName) method is "Class". Class is a class in java.lang package.

```
Class c = Class.forName(
 "oracle.jdbc.driver.OracleDriver");
```

```
Class c = Class.forName(
 ”sun.jdbc.odbc.JdbcOdbcDriver");
```

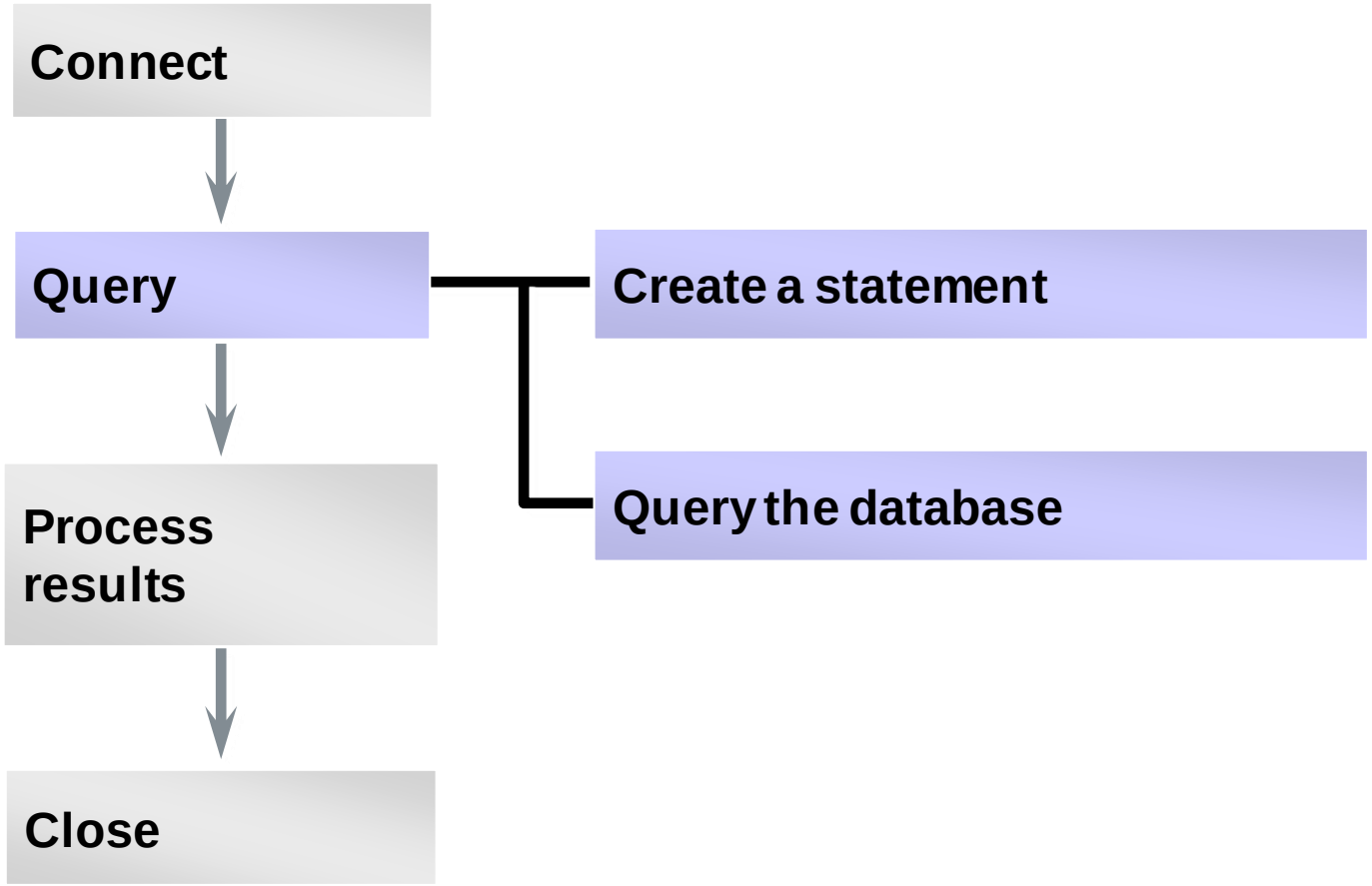
Establish a Connection

- JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver.
- The getConnection() method is used to establish a session/connection to a specific database
- An application can have one or more connections with a single database, or it can have many connections with different databases.
- A Connection object provides metadata i.e. information about the database, tables, and fields. It also contains methods to deal with transactions.

```
Connection conn =
DriverManager.getConnection(URL,userid,password);
```

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@myhost:1521:orcl",
      "scott", "tiger");
```


Stage 2: Query construction



Create a Statement

- A Statement object sends your SQL statement to the database
- You need an active connection to create a JDBC statement

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


Types of Statement

- **Statement**
  Execute simple sql queries without parameters.

```
Statement createStatement()
```

- **Prepared Statement**
  Execute precompiled sql queries with or without parameters.
  PreparedStatement objects are precompiled SQL statements.

```
PreparedStatement prepareStatement(String sql)
```

- **Callable Statement**
  Execute a call to a database stored procedure.

```
CallableStatement prepareCall(String sql)
```

Query the Database

Statement has three methods to execute a SQL statement:

- *executeQuery()* for QUERY statements
- *executeUpdate()* for INSERT, UPDATE, DELETE, or DDL statements
- *execute()* for either type of statement

```
ResultSet rset = stmt.executeQuery(statement);
int count = stmt.executeUpdate(statement);
boolean isquery = stmt.execute(statement);
```


Query the Database: Examples

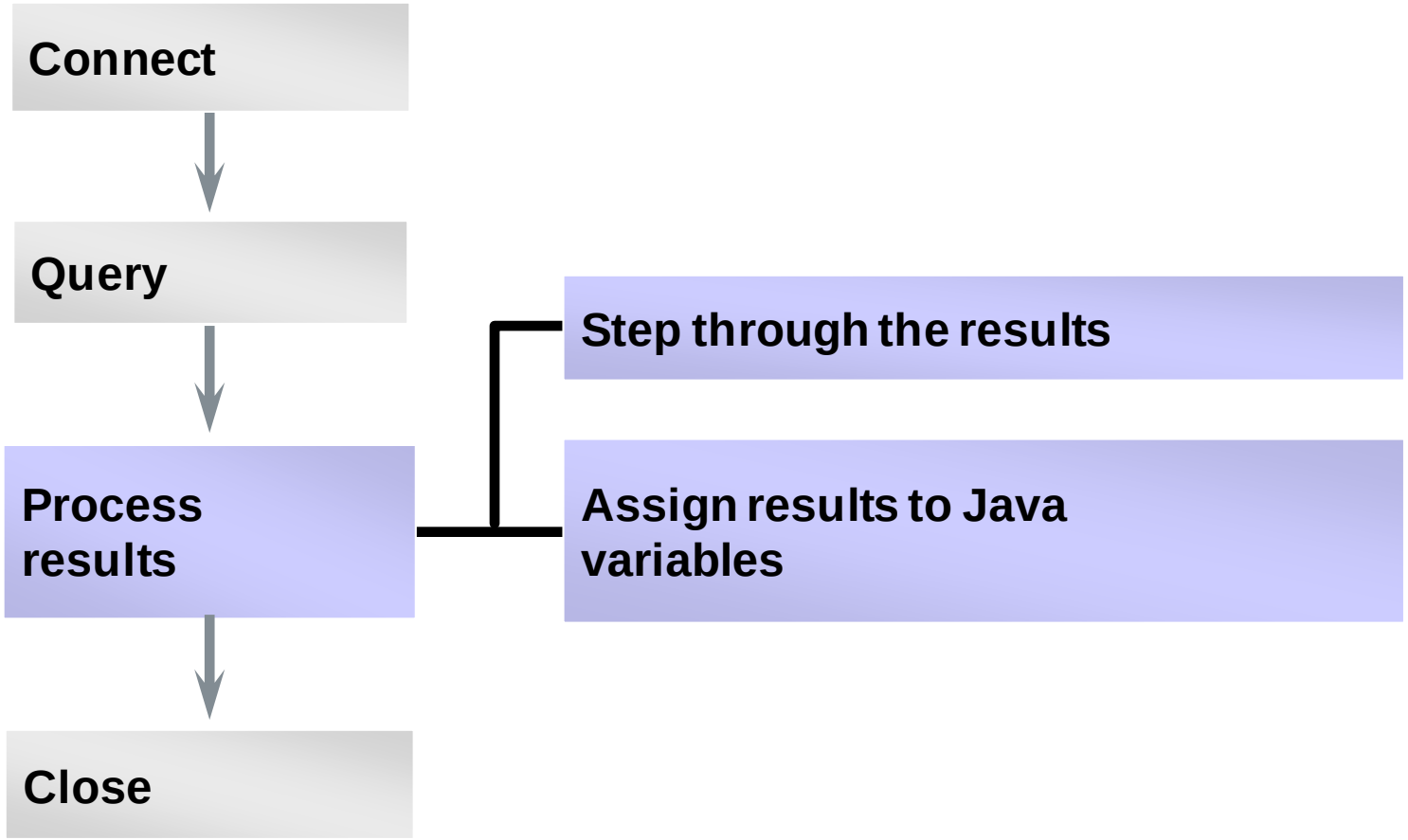
- Execute a select statement

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery
  ("select RENTAL_ID, STATUS from ACME_RENTALS");
```

Execute a delete statement

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
  ("delete from ACME_RENTAL_ITEMS
      where rental_id = 1011");
```

Stage 3: Process the Results



ResultSet

- JDBC returns the results of a query in a ResultSet object
- A ResultSet maintains a cursor pointing to its current row of data
- Use next() to step through the result set row by row
- getString(), getInt(), and so on assign each value to a Java variable

Process the Results

- Step through the result set

```
while (rset.next()) { … }
```

- Use getXXX() to get each column value

```
String val =
rset.getString(colname);
```

```
String val =
rset.getString(colIndex);
```

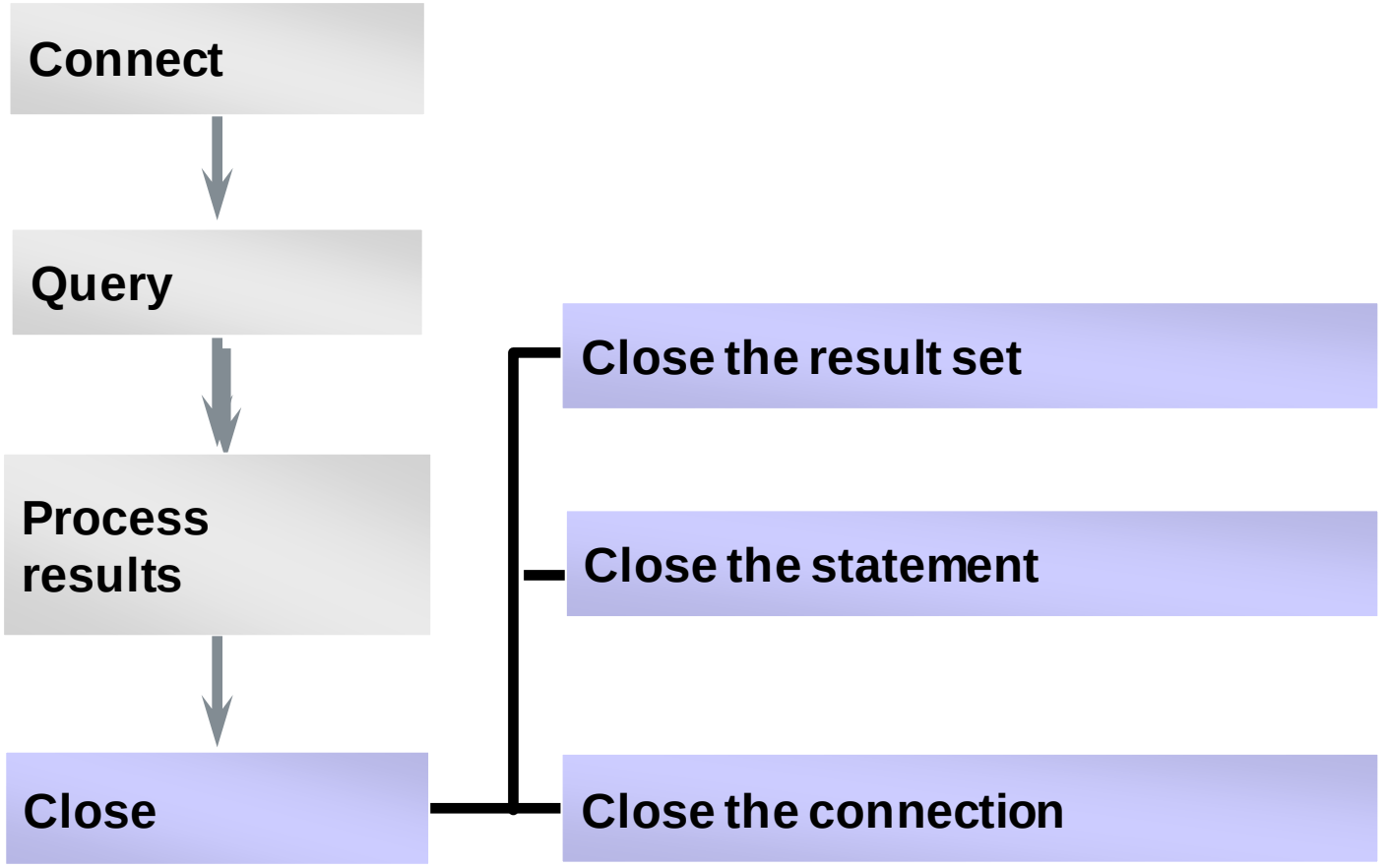
```
while (rset.next()) {
  String title = rset.getString("TITLE");
  String year = rset.getString("YEAR");
  … // Process or display the data
}
```

Handle SQL Null Values

- Java primitive types cannot have null values
- Do not use a primitive type when your query might return a SQL null
- Use ResultSet.wasNull() to determine whether a column has a null value

```
while (rset.next()) {
    String year = rset.getString("YEAR");
    if (rset.wasNull() {
        … // Handle null value}
…}
```


Stage 4: Close



Thank you

Innovative Services

Passionate Employees

Delighted Customers