Database Access using **JDBC**

Tessema Mengistu (Ph.D.)
Department of Computer Science
Virginia Tech
Mengistu@vt.edu

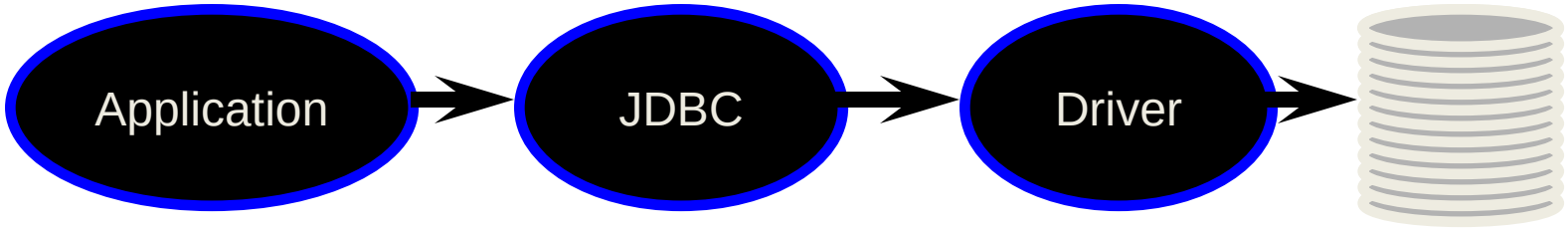
Outline

- Introduction to JDBC
- Connect to a database using Java Database Connectivity (JDBC)
- Create and execute a query using JDBC
- Process and manipulate the result of a query

Introduction to JDBC

- JDBC – Java DataBase Connectivity
  - Industry standard interface for connecting to relational databases from Java
  - Independent of any DBMS
  - Allows three things
    - Establish a connection with Relational databases
    - Send SQL statements
    - Process the results
  - Provides two APIs
    - API for application writers
    - API for driver writers

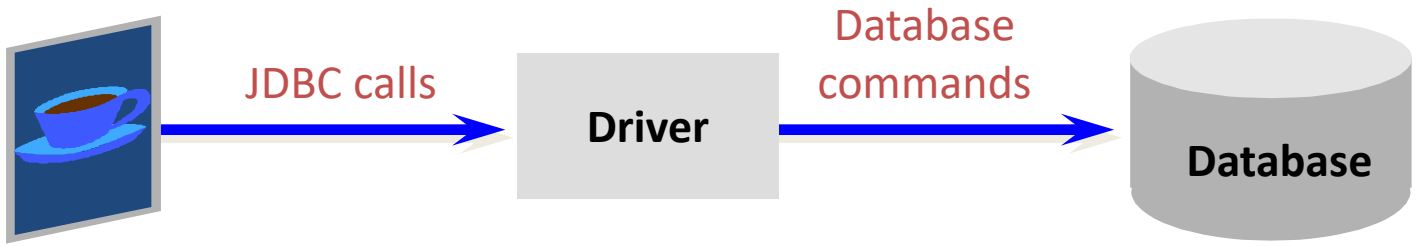
JDBC Architecture



- Java code calls JDBC library
- JDBC loads a *driver*
- Driver talks to a particular database
- Can have more than one driver -> more than one database
- Ideal: can change database engines without changing any application code

A JDBC Driver

- Is an interpreter that translates JDBC method calls to vendor-specific database commands

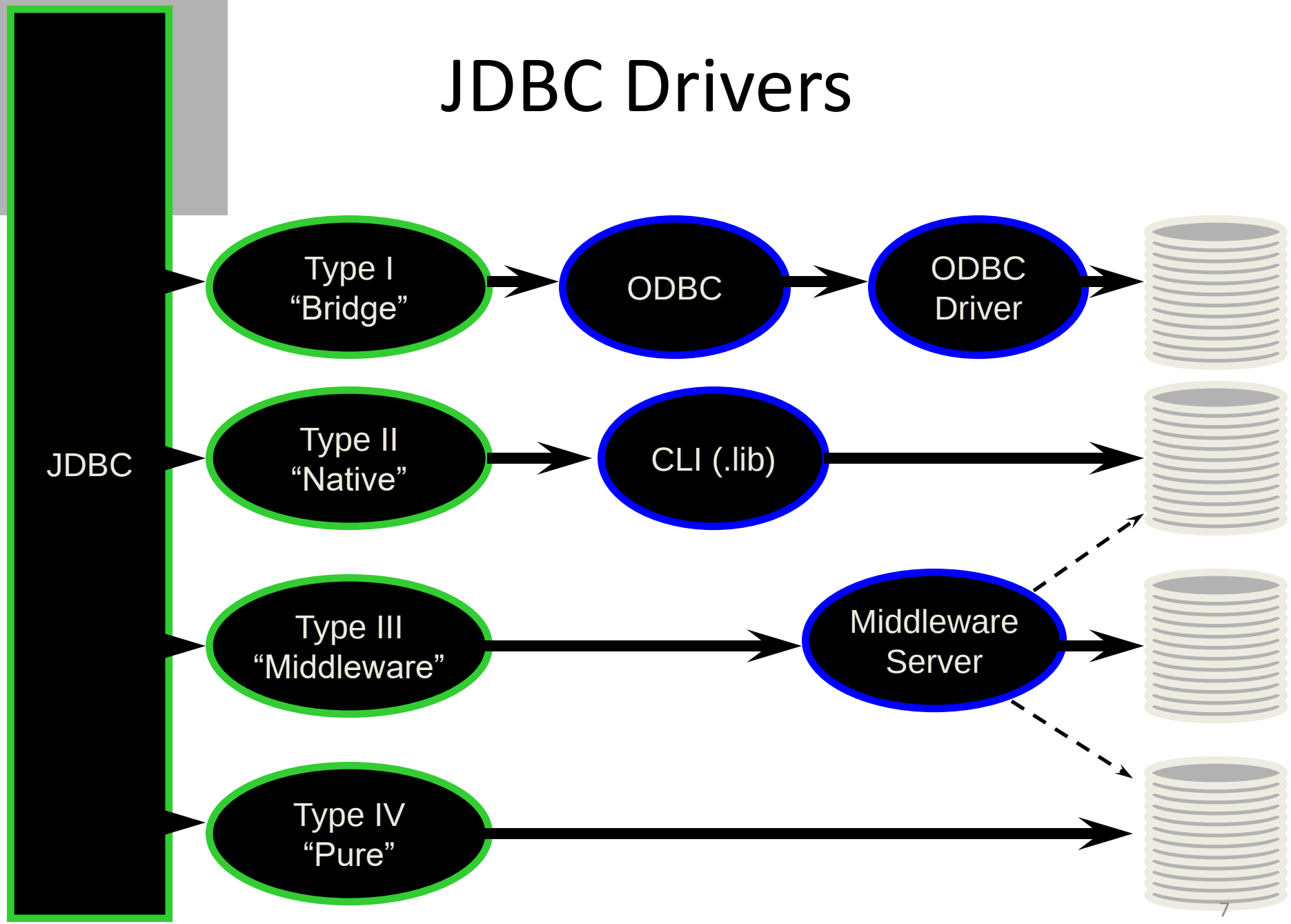


- Implements interfaces **in java.sql**
- Can also provide a vendor's extensions to the JDBC standard

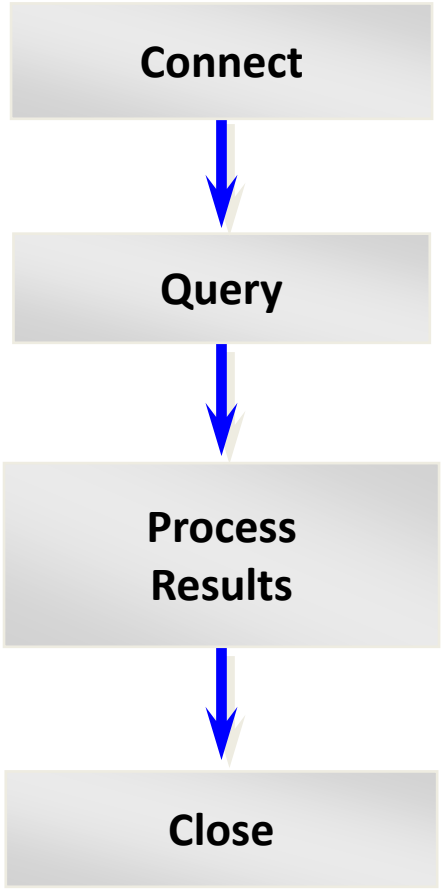
JDBC Drivers

- Type I: “Bridge”
- Type II: “Native”
- Type III: “Middleware”
- Type IV: “Pure”

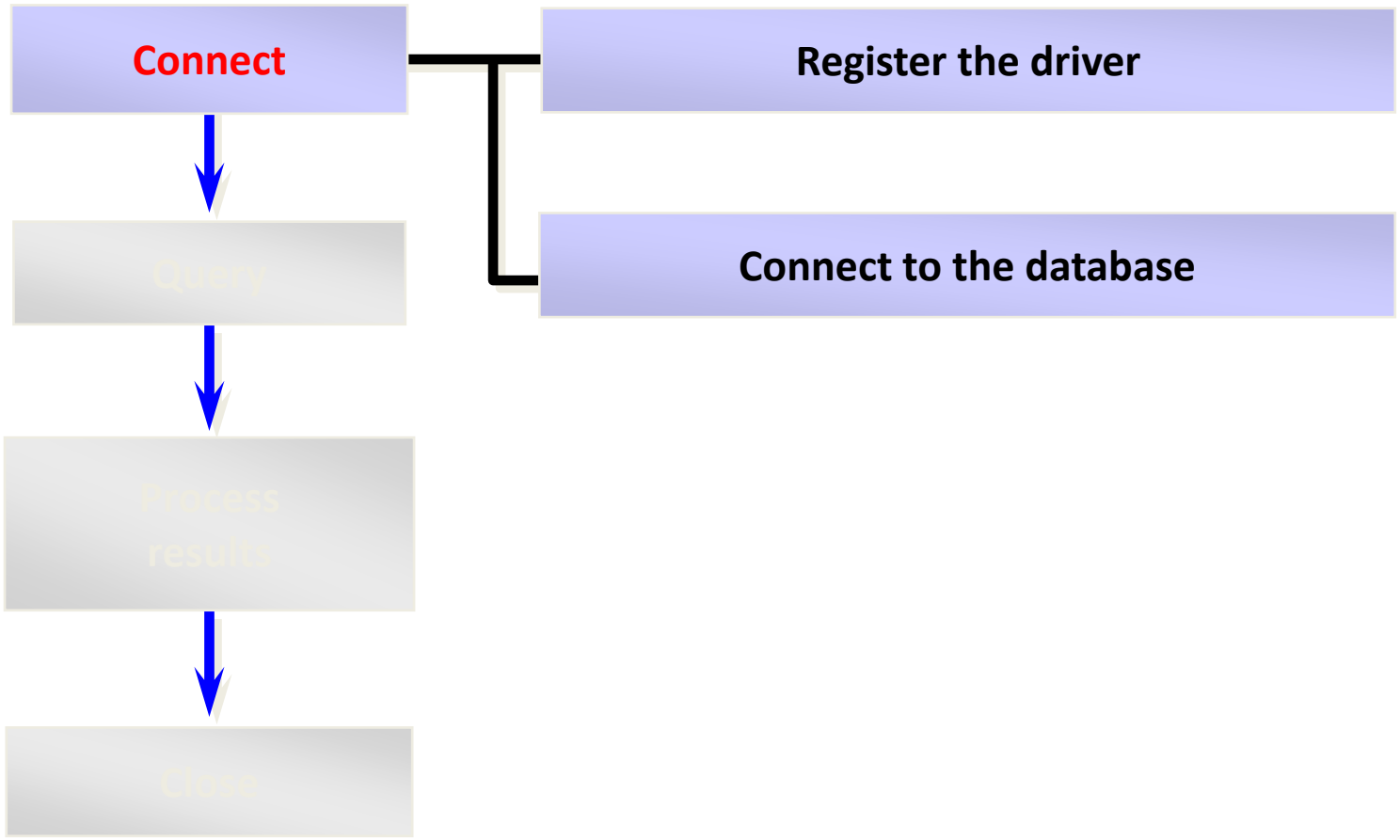
JDBC Drivers



Overview of Querying a Database With JDBC



Stage 1: Connect



Registering a Driver

- Automatically load driver
  - Before JDBC 4.0, it should be loaded statically
    - `Class.forName("Driver Name");`
- Every driver has its own name
  - For example
    - mysql - com.mysql.jdbc.Driver
    - Oracle - oracle.jdbc.driver.OracleDriver
- Use the *jdbc.drivers* system property

Connection to the Database

- Handled:
  - *DriverManager* Object
  - *DataSource* Object
    - Preferred way of getting a connection
    - Provide Connection pooling

DeviceManager- Connection to the Database

- Calls `getConnection()` Method
  - Accepts
    - JDBC URL,
    - username
    - password
  - Returns a connection object
- Throws `java.sql.SQLException`

DataSource- Connection to the Database

- Deploying DataSource object need three tasks:
  - Creating an instance of the DataSource class
  - Setting its properties
    - Username, password, url, etc.
  - Registering it with a naming service (JNDI API)

JDBC URLs

- JDBC uses a URL to identify the database connection

  jdbc:*subprotocol*:*source*

- Each driver has its own subprotocol
- Each subprotocol has its own syntax for the source

RDBMS	Database URL format
MySQL	`jdbc:mysql://`*hostname*`:`*portNumber*`/`*databaseName*
ORACLE	`jdbc:oracle:thin:@`*hostname*`:`*portNumber*`:`*databaseName*
DB2	`jdbc:db2:`*hostname*`:`*portNumber*`/`*databaseName*
PostgreSQL	`jdbc:postgresql://`*hostname*`:`*portNumber*`/`*databaseName*
Java DB/Apache Derby	`jdbc:derby:`*dataBaseName* (embedded) `jdbc:derby://`*hostname*`:`*portNumber*`/`*databaseName* (network)
Microsoft SQL Server	`jdbc:sqlserver://`*hostname*`:`*portNumber*`;databaseName=`*dataBaseName*
Sybase	`jdbc:sybase:Tds:`*hostname*`:`*portNumber*`/`*databaseName*

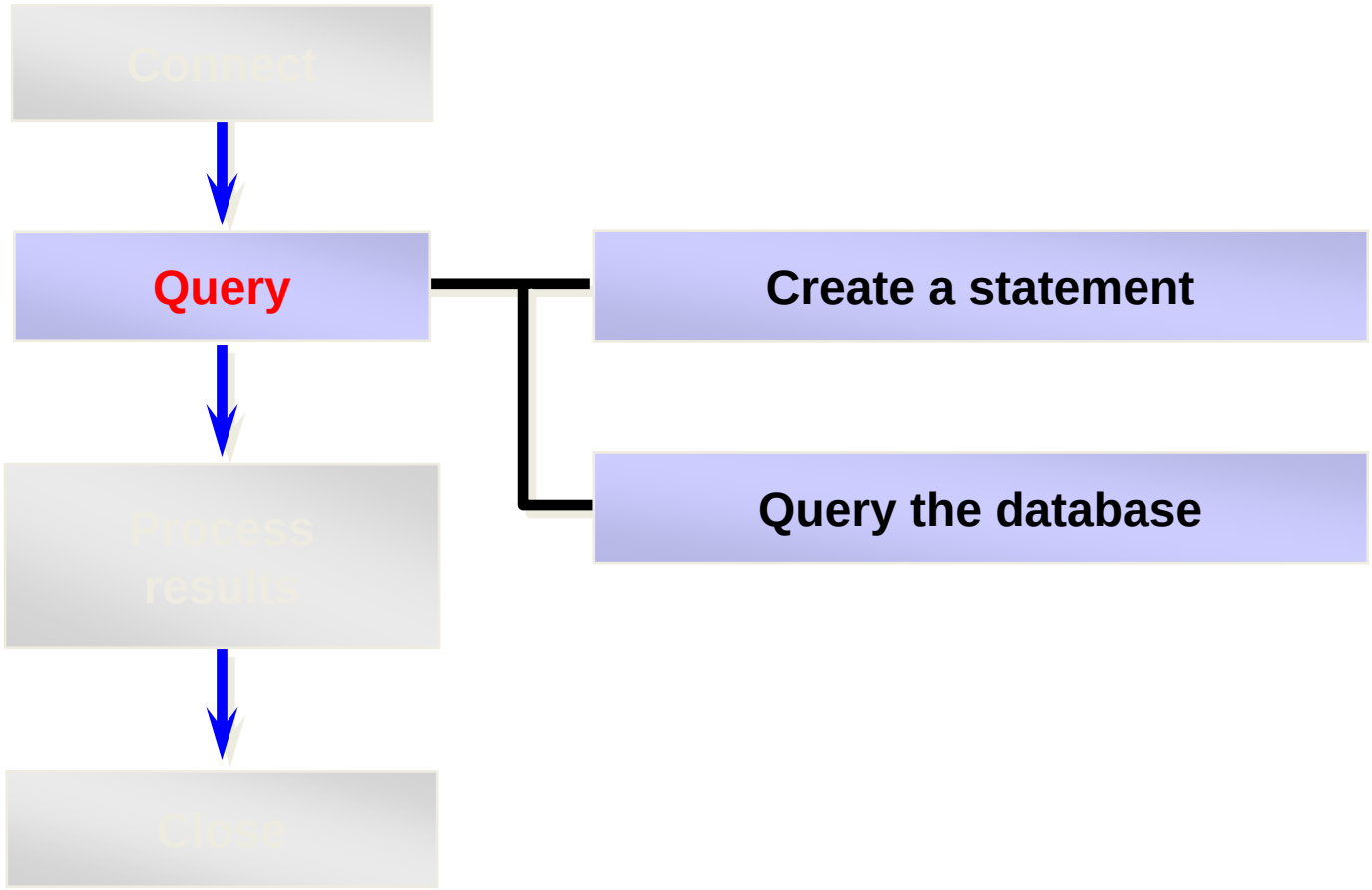
Connection

- A Connection represents a session with a specific database.
- Within the context of a Connection, SQL statements are executed and results are returned.
- Can have multiple connections to a database
- Also provides “metadata” -- information about the database, tables, and fields
- Also methods to deal with transactions

Obtaining a Connection

```
String url =
"jdbc:oracle:thin:@dbserv.cs.siu.edu:1521:cs";
    try {
  Connection con = DriverManager.getConnection(url);
 }
  catch (ClassNotFoundException e)
       {
           e.printStackTrace();
         }
  catch (SQLException e)
        {
           e.printStackTrace();
        }
```

Stage 2: Query



Statement

- A Statement object is used for executing a SQL statement and obtaining the results produced by it
- Different types of Statements
  - *Statement*
  - *PreparedStatement*
  - *CallabaleStatment*

Connection Methods

- Statement
  - Created by `createStatement()` method on the connection object
  - returns a new Statement object

PreparedStatment

- Creates precompiled SQL statements - more efficient than Statement class
- Can also allow specifying parameters that can be filled during run time
- Usually used if the statements is going to execute more than once
- Created by `prepareStatement()` method on the connection object
  - Accepts the query
  - returns a new PreparedStatement object

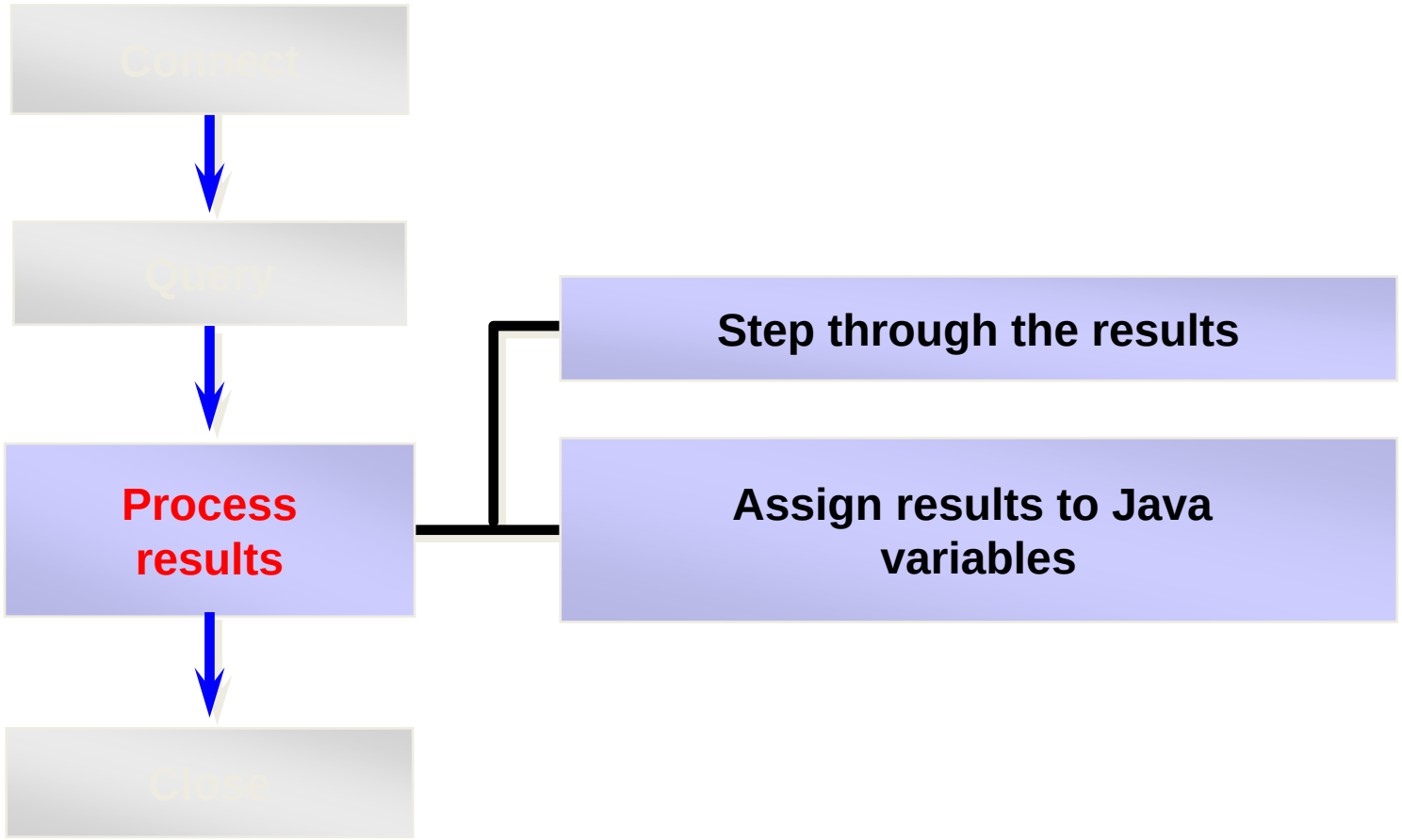
CallableStatment

- Many DBMS can store individual or set of SQL statements in a database
  - Stored Procedures
- JDBC allows programs to invoke stored procedures using CallableStatment interface
- A Callable statement object holds parameters for calling stored procedures
- CallableStatement `prepareCall(String sql)`
  - Accepts the query
  - returns a new CallableStatement object

Statement Methods

- *executeQuery(String)*
  - Accepts SQL statement
  - Execute a SQL statement that returns a single ResultSet.
- *executeUpdate(String)*
  - Execute a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows changed.
- *execute(String)*
  - Execute a SQL statement that may return multiple results.

Stage 3: Process the Results



ResultSet

- A *`ResultSet`* provides access to a table of data generated by executing a Statement.
- Only one *`ResultSet`* per Statement can be open at once.
- The table rows are retrieved in sequence.
- A *`ResultSet`* maintains a cursor pointing to its current row of data.
- The 'next' method moves the cursor to the next row.

ResultSet Methods

- boolean `next()`
  - activates the next row
  - the first call to next() activates the first row
  - returns false if there are no more rows

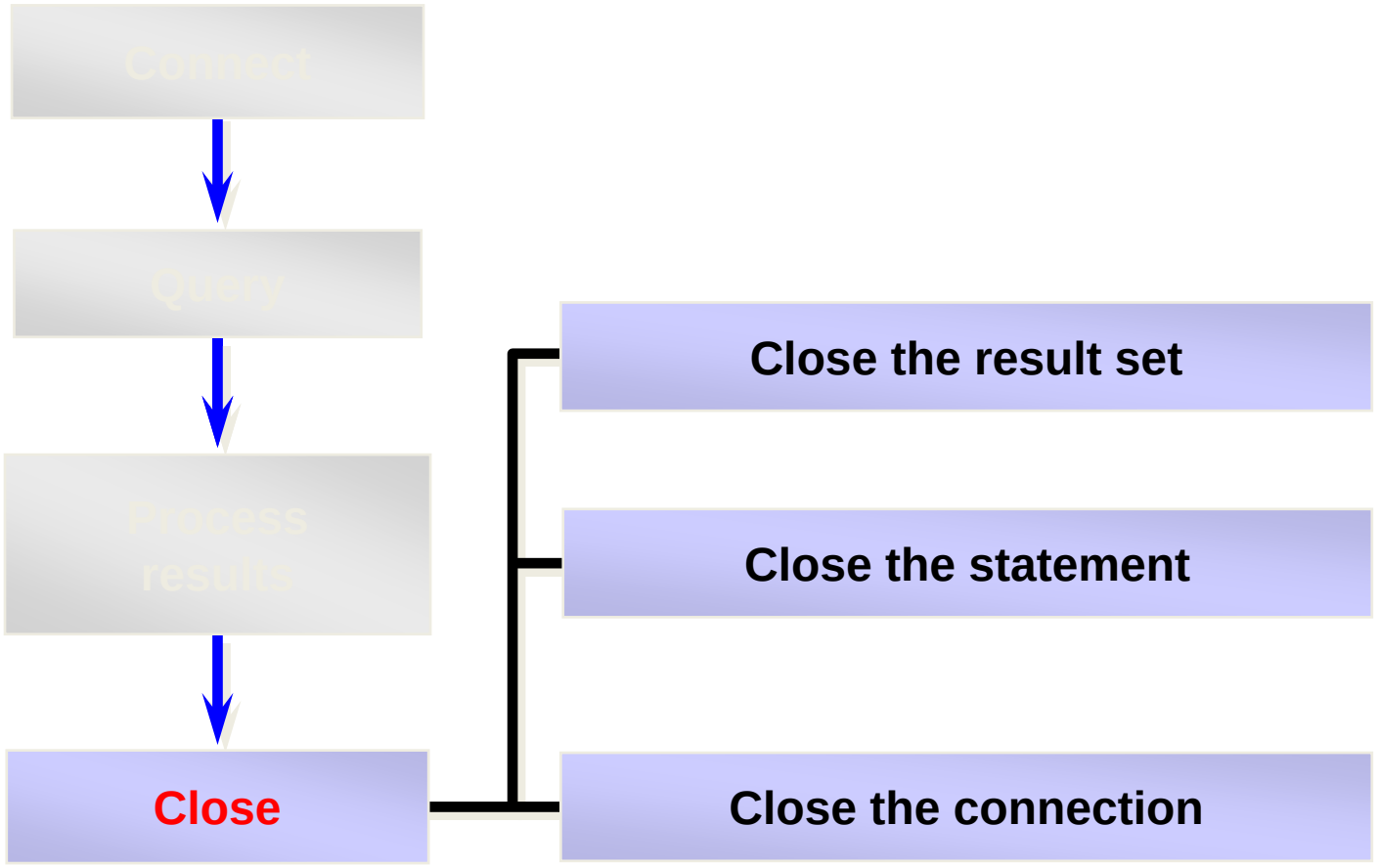
ResultSet Methods

- *`Type getType(int columnIndex)`*
  - returns the given field as the given type
  - fields indexed starting at 1 (not 0)
- *`Type getType(String columnName)`*
  - same, but uses name of field
  - less efficient
- *`int findColumn(String columnName)`*
  - looks up column index given column name

ResultSet Methods

- String getString(String columnName)
- boolean getBoolean(String columnName)
- byte getByte(String columnName)
- short getShort(String columnName)
- int getInt(String columnName)
- long getLong(String columnName)
- float getFloat(String columnName)
- double getDouble(String columnName)
- Date getDate(String columnName)
- Time getTime(String columnName)
- Timestamp getTimestamp(String columnName)

Stage 4: Close



Close

- Close the ResultSet object
  - void `close()`
    - disposes of the ResultSet
    - allows you to re-use the Statement that created it
    - automatically called by most Statement methods
- Close the Statement Object
  - void `close()`
- Close the connection
  - void `close()`
-

- Step 1: import java.sql Package
- Step2: Load and register the driver *// if needed*
- Step 3:Connect to the database
- Step 4: Create statements
- Step 5: Execute statements & Process the Result
- Step 6: close Statement
- Step 7: Close the connection

JDBC Object Classes

- DriverManager/ResourceObject
  - Loads, chooses drivers
- Driver
  - connects to actual database
- Connection
  - a series of SQL statements to and from the DB
- Statement
  - a single SQL statement
- ResultSet
  - the records returned from a Statement

Mapping Java Types to SQL Types

SQL type	Java Type
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

isNull

- In SQL, NULL means the field is empty
- Not the same as 0 or “”
- In JDBC, you must explicitly ask if a field is null by calling `ResultSet.isNull(column)`

References

- Oracle Documentation
  - https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html