Java Data Base Connectivity (JDBC)

Fresher Learning Program
January, 2012

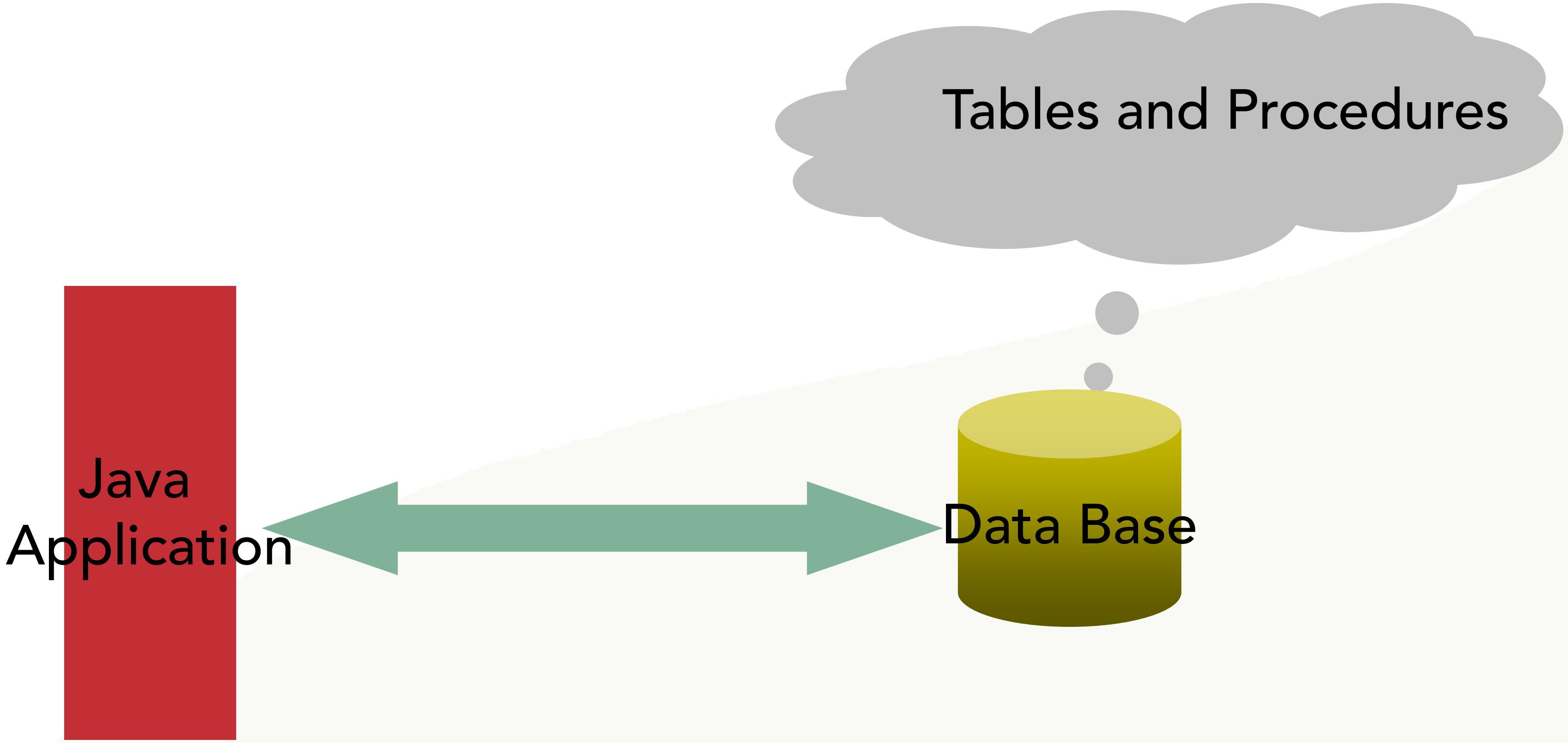
People matter, results count.

Objectives of JDBC

- **Purpose**:
  - Awareness of JDBC, types of drivers, how to create database connection and execute query using JDBC API, callable statement, prepared statement and transaction.
- **Product**:
  - To know how to load database driver
  - How to create database connection, execute query, and handle the result using JDBC
  - To understand Statement, Callable and Prepared statement
  - Awareness of transaction, and batch query execution
- **Process**:
  - Theory Sessions along with assignments
  - A recap at the end of the session in the form of Quiz

Table of Contents

What is JDBC



What is JDBC

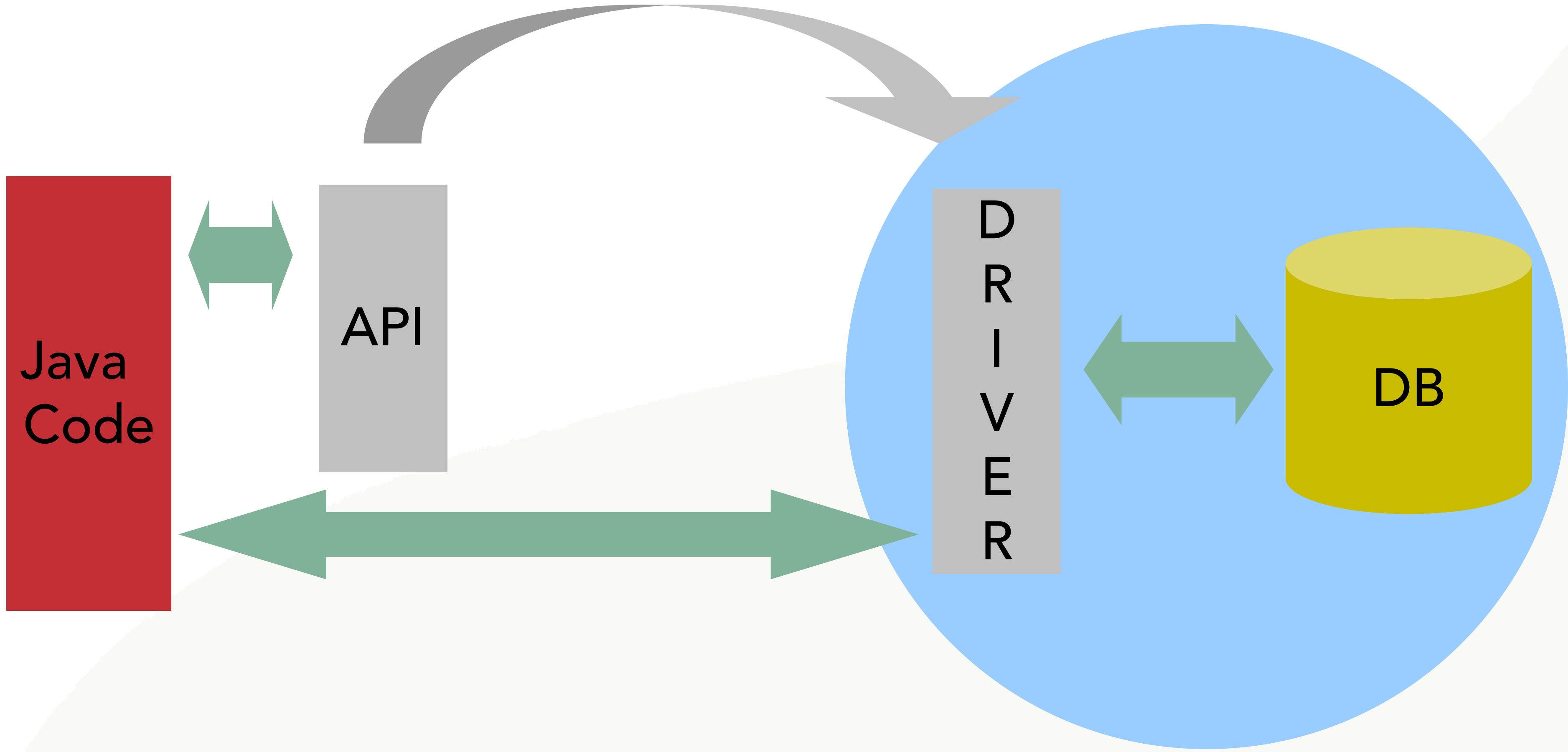
- A pure java API interface for database communication

- A set of classes that performs database transaction.
  - Connection to relational DB
  - Send SQL commands
  - Process results

Benefits of JDBC

- Don’t have to rely on single DB vendor .
- Easier for DB vendors
- Don’t need to provide a query language, only implement API.

Java Data Base Connectivity

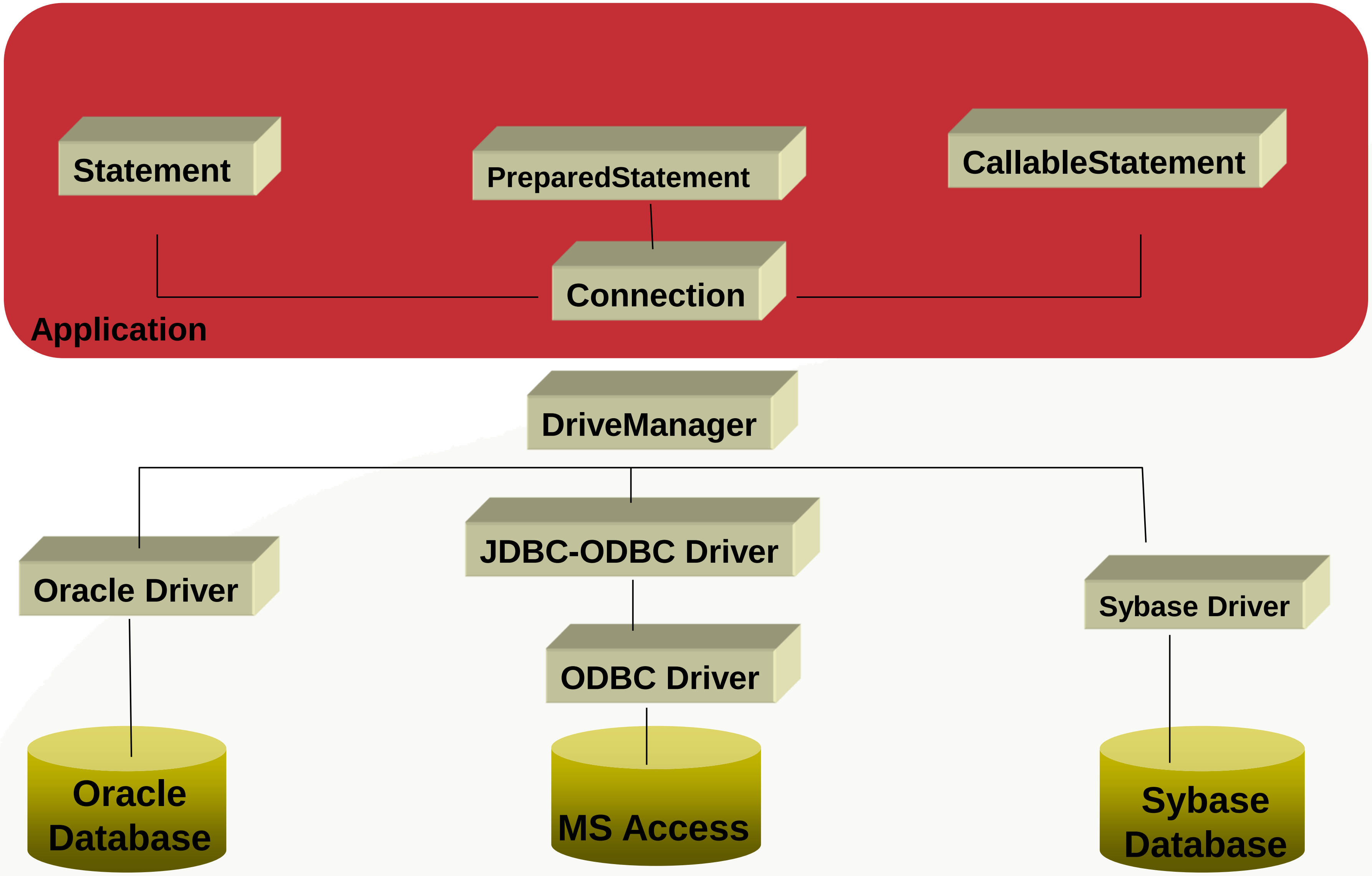


JDBC API

API layer has2 levels of interface.

- **Application layer**: developer uses API to make calls to DB via SQL & retrieve results.
- **Driver layer** : handles all communication with a specific Driver implementation

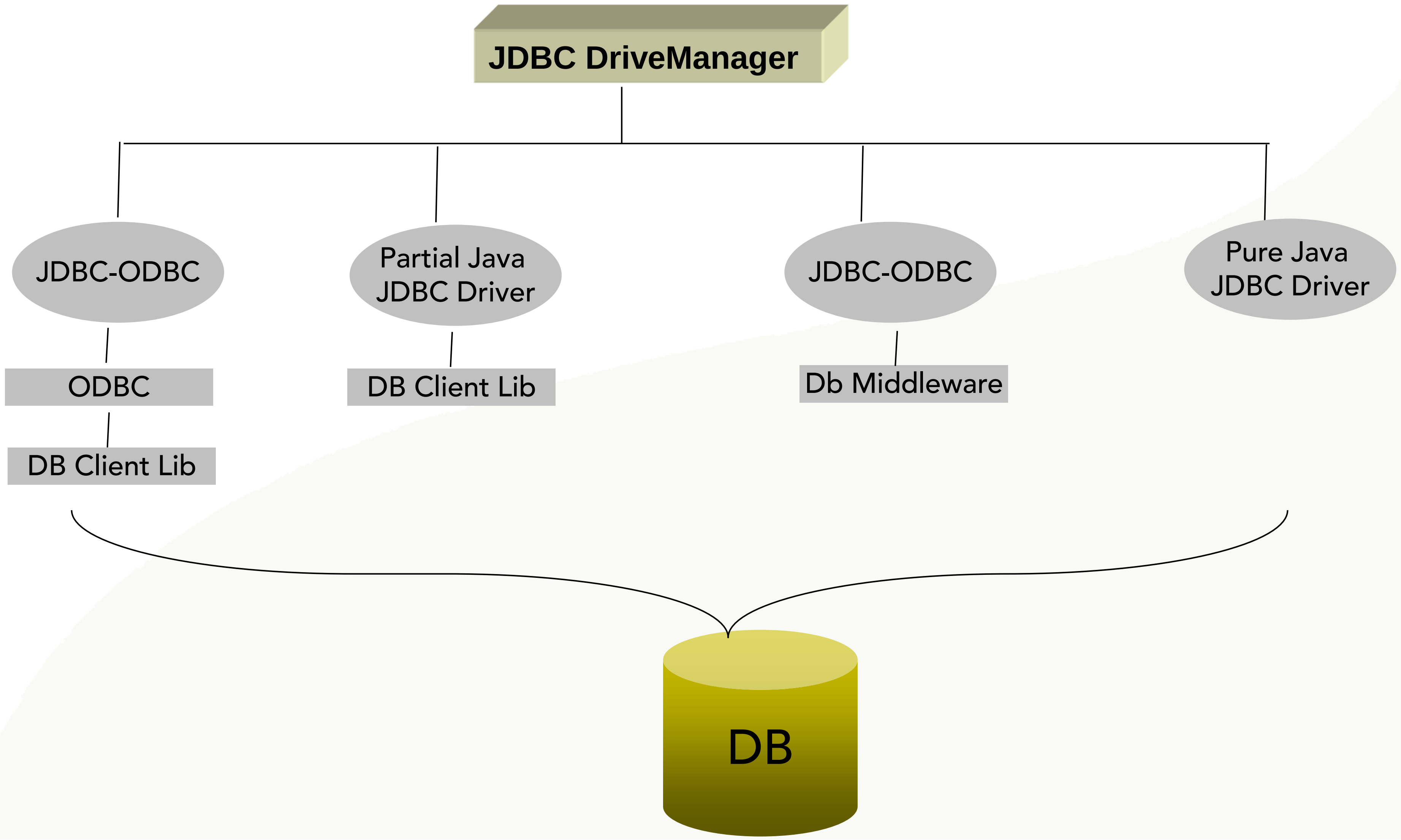
JDBC API


Statement
PreparedStatement
CallableStatement
Connection
Application
DriveManager
JDBC-ODBC Driver
Oracle Driver
Sybase Driver
ODBC Driver
Oracle Database
MS Access
Sybase Database

JDBC Drivers

- **JDBC-ODBC Bridge** (Type 1 Driver)
- **Partial Java driver** (Type 2 Driver)
- **Pure Java driver for database middleware** (Type 3 Driver)
- **direct to database pure Java driver** (Type 4 Driver)

JDBC Drivers



Steps

1. Loading Drivers
2. Establishing Connection
3. Creating Statements
4. Executing Statements Queries
5. Handle result (result set)
6. Close connection

Loading Drivers

- Class.forName(“sun.jdbc.driver.JdbcOdbcDriver”);
- Class.forName loads the class in the memory and Makes the class(Driver) available to the DriverManager

Making a Connection

- The second step is to have the appropriate driver connect to the DBMS.

```
Connection con =
  DriverManager.getConnection("jdbc:mysql://cannings.org:3306/test", "myLogin", "myPassword");
```

- The URL consists of the protocol, type of database, hostname, port, and name of the database. In this case we are using JDBC to connect to a mysql database, on host cannings.org, port 3306, to database test.

Creating Statements

- A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it.

- Statements can be used to update the database:

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

Creating Statements, Queries

- Statements are also used to Query the database:

```
ResultSet rs = stmt.executeQuery(
    "SELECT COF_NAME, PRICE FROM COFFEES");
stmt.executeUpdate("CREATE TABLE COFFEES " +
    "(COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE
  FLOAT, " +
    "SALES INTEGER, TOTAL INTEGER)");
```

Result Sets

- JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results.

```
String query = "SELECT COF_NAME, PRICE FROM
  COFFEES";
ResultSet rs = stmt.executeQuery(query);
while (rs.next()) {
    String s = rs.getString("COF_NAME");
    float n = rs.getFloat("PRICE");
    System.out.println(s + "   " + n);
}
```

Result Sets, methods

- The getXXX method is used to retrieve data by column name:
  - `String s = rs.getString("COF_NAME");`
- We may also use a column index to access the data:
  - `String s = rs.getString(1);`
  - `float n = rs.getFloat(2);`
- The next() method is used to access the next row in the resultset:
  - `while (rs.next()) {/* get current row */}`

Closing connection

- It is always a better practice to close the connection, if it is no longer in use
- How – by calling close() method of connection object

```
con.close();
```

- Best place to write close code is finally block of try catch (if any)

Callable and Prepared Statements

- Prepared Statement
  - pre-compiled query
  - reduces execution time
  - given sql query when it is created
  - used for queries that are used many times

- Callable Statement
  - a stored procedure inside the DBMS
  - can be created and accessed by JDBC
  - used to encapsulate a set of operations or queries to execute on a database server

Prepared Statement Example

- `PreparedStatement updateSales = con.prepareStatement(`
  `"UPDATE COFFEE SET SALES = ? WHERE COF_NAME LIKE ? ");`
- `updateSales.setInt(1, 75);`
- `updateSales.setString(2, ”Columbian");`
- `updateSales.executeUpdate():`

Callable Statement Example

- Syntax for defining a stored procedure is different for each DBMS

```
String createProcedure = "create procedure
  SHOW_SUPPLIERS " + "as " + "select
  SUPPLIERS.SUP_NAME, COFFEES.COF_NAME " + "from
  SUPPLIERS, COFFEES " + "where SUPPLIERS.SUP_ID =
  COFFEES.SUP_ID " + "order by SUP_NAME";

Statement stmt = con.createStatement();

stmt.executeUpdate(createProcedure);

CallableStatement cs = con.prepareCall("{call
  SHOW_SUPPLIERS}");
ResultSet rs = cs.executeQuery();
```

Transaction

- A transaction is a collection of DB modifications, which is treated as an atomic DB operation. All take place or none do.

- Transactions are used to make sure that a collection of updates leaves the database in a consistent state

  (as defined by the application program).

- By default the Connection automatically commits; changes after executing each statement. If auto commit has been disabled, the method commit must be called explicitly; otherwise, database changes will not be saved.

ACID properties of Transactions

- **Atomicity**
  The transaction completes (commits) or if it fails (aborts) then all effects are undone (rollback)
- **Consistency**
  Transactions produce consistent results
- **Isolation**
  Intermediate results are not visible, and transactions appear to execute serially even if done concurrently
- **Durability**
  The effects of a committed transaction are never lost

Transaction

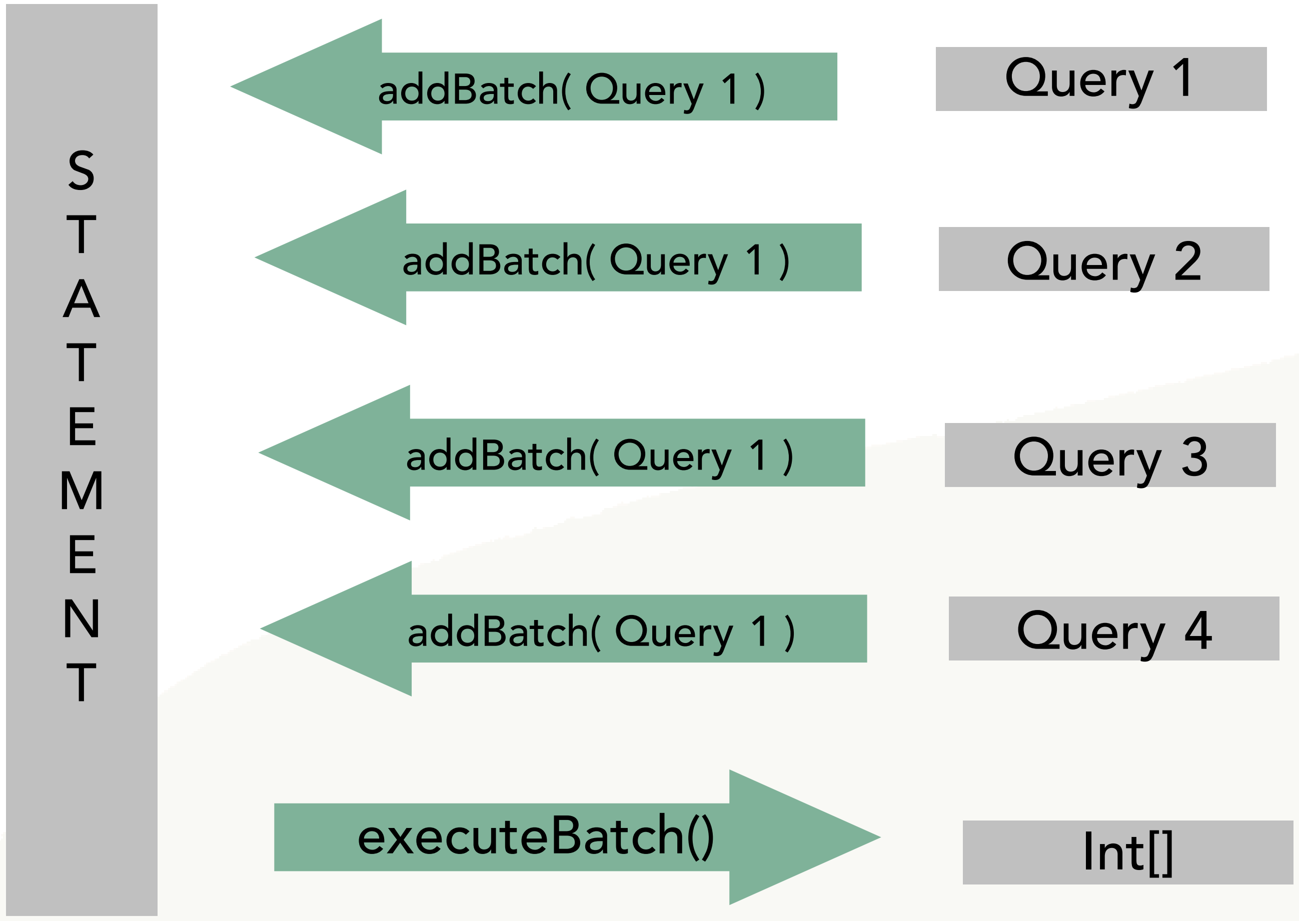
Example:

- // Change to transactional mode
- con.setAutoCommit(false);

- // Transaction A begins here
- stmt.executeUpdate("DELETE * FROM ACCOUNT...;"); // 1
- stmt.executeUpdate("INSERT INTO ACCOUNT ...."); // 2
- stmt.executeUpdate("INSERT INTO ACCOUNT ...."); // 3
- stmt.executeUpdate("INSERT INTO ACCOUNT ...."); // 4
- con.commit();

- // Commit changes to database
- // All of 1,2,3,4 take effect

Batch Updates



Batch Updates

```
dbCon.setAutoCommit(false);
Statement stmt= dbCon.createStatement();
stmt.addBatch(" Query 1" );
stmt.addBatch(" Query 2" );
stmt.addBatch(" Query 3" );
stmt.addBatch(" Query 4" );
int[] updCnt = stmt.executeBatch();
dbCon.commit();
```

Summary

- JDBC’s goal is:
  - DBMS-independent interface.
  - Can access any data source without recoding.
- JDBC Drivers:

  1)JDBC-ODBC Bridge

  2) Partial Java driver

  3)Pure Java driver for database middleware

  4)direct to database pure Java driver.
- JDBC supports basic SQL functionality
- JDBC is flexible, easy to manage and inexpensive.
- Transactions
- Batch Updates
- Connection pooling

Recap

DriverManager

ResultSet

Type-1

Callable statement

ACID

Statement

Connection

Prepared statement

Batch Update

Thank You For Your Time

People matter, results count.