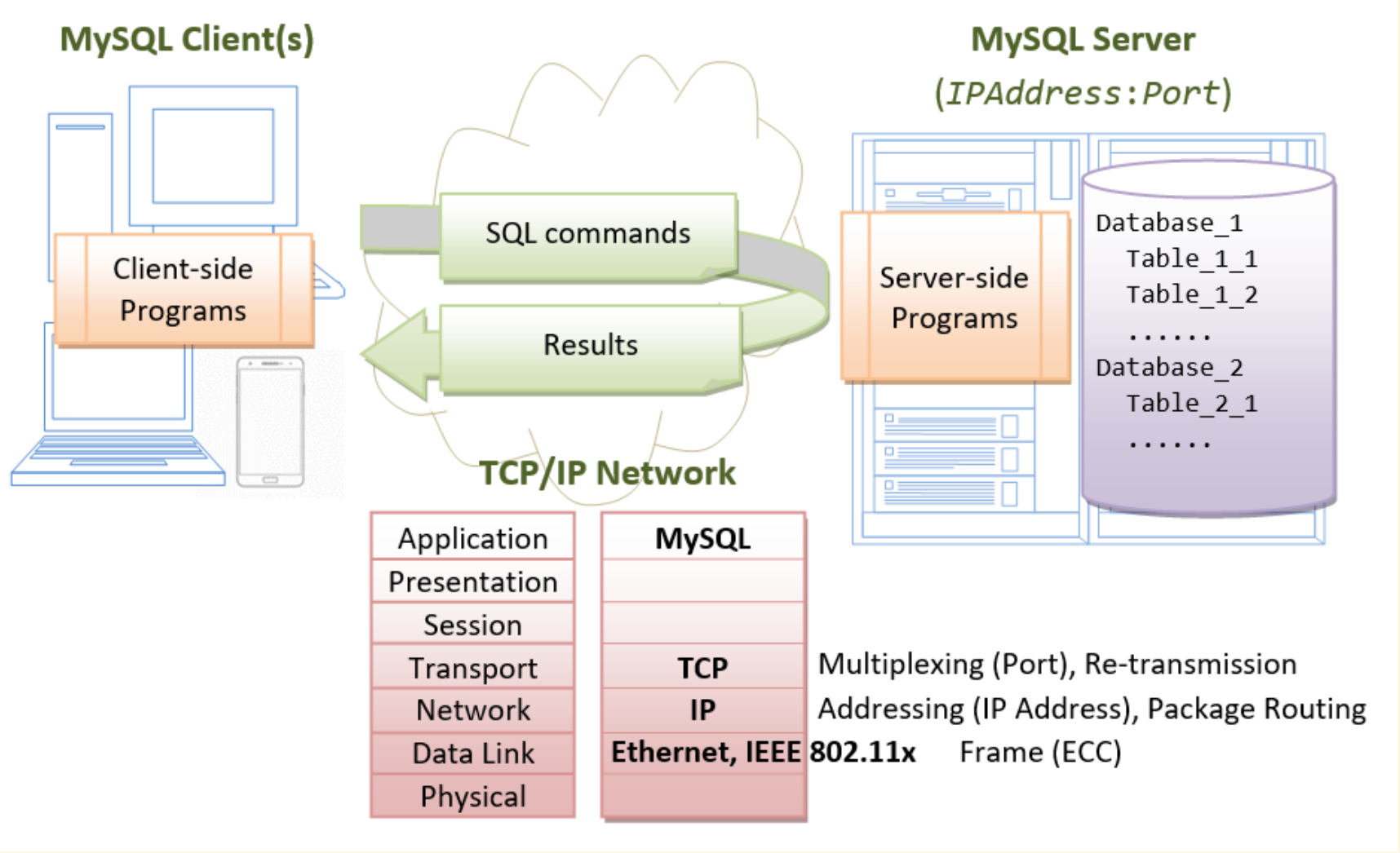
Database Connectivity

Overview

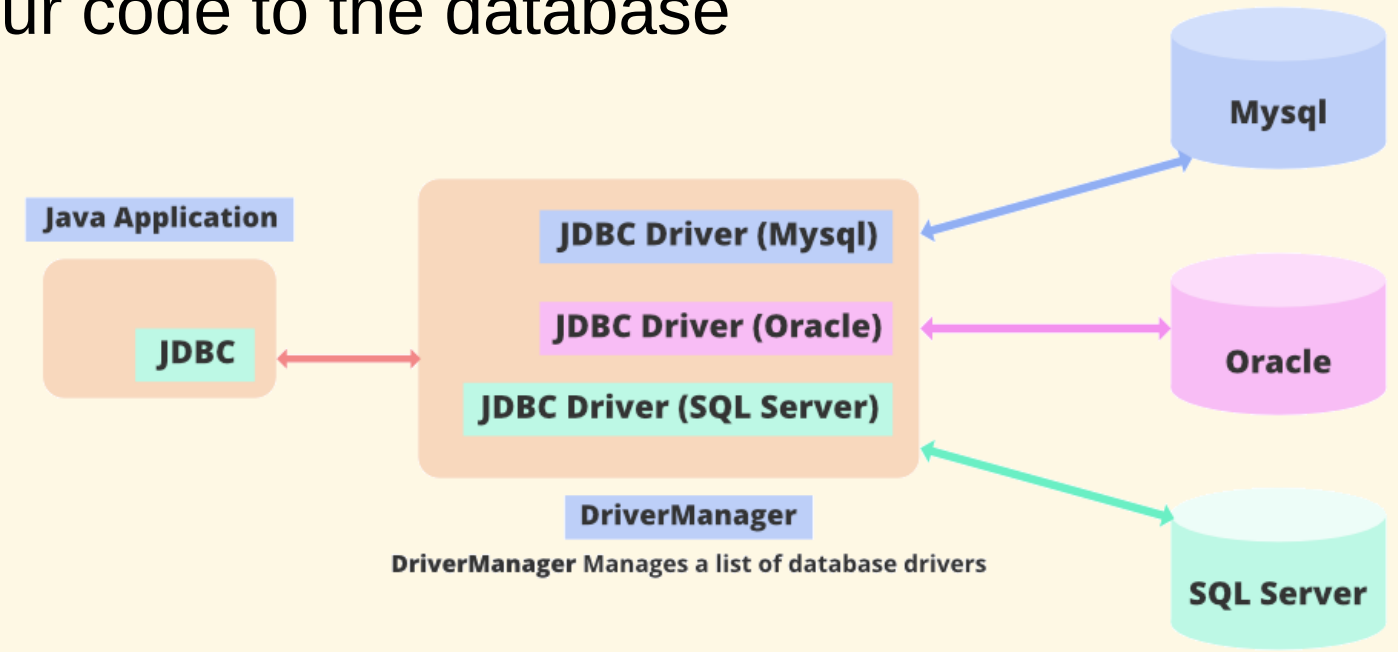
- Very often, nowadays applications are backed by a database to easily manage their persistent data
- Prerequisites:
  - Have prior knowledges on RDBMS (Relational Database Management Systems) and the SQL language
  - Have installed a RDBMS (like MySQL)
  - Import the sample database from `university-db.sql`:
    - `?> mysql -u <user-name> -p university < university-db.sql`

SQL


MySQL Client(s)
MySQL Server
(IPAddress:Port)
Client-side Programs
SQL commands
Results
Server-side Programs
Database_1
Table_1_1
Table_1_2
.....
Database_2
Table_2_1
.....
TCP/IP Network
Application
Presentation
Session
Transport
Network
Data Link
Physical
MySQL
TCP
IP
Ethernet, IEEE 802.11x
Multiplexing (Port), Re-transmission
Addressing (IP Address), Package Routing
Frame (ECC)

JDBC

- JDBC (Java Database Connectivity) is the API that manages connecting to a database, issuing queries and commands, and handling result sets
- It acts as a bridge from your code to the database
- Connecting to each DBMS from JDBC requires a correct JDBC driver

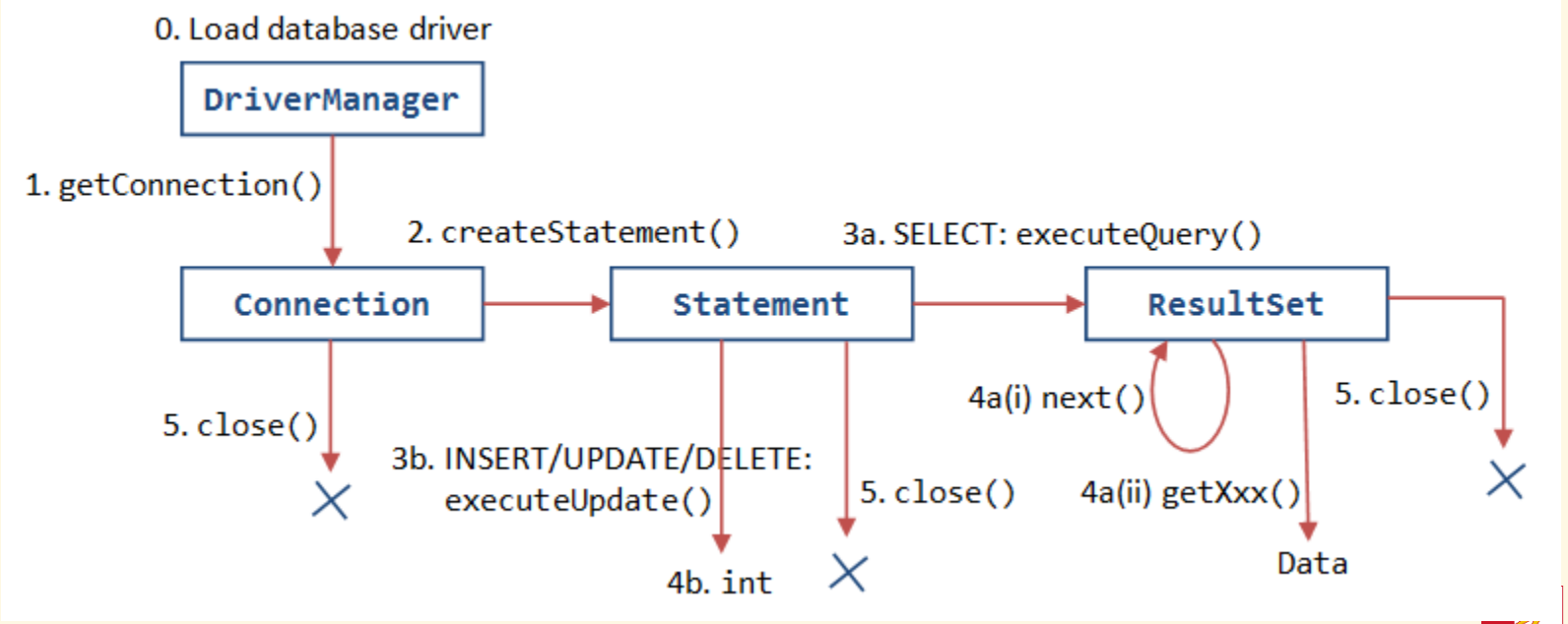


Install the MySQL JDBC Driver

1. Go to: https://dev.mysql.com/downloads/connector/j/
2. In Operating System, choose “Platform Independent”
3. Click on “Download” button
4. Click on the small link “No thanks, just start my download.”
5. Save the zip file and extract the .jar file inside it

Working with JDBC

- A JDBC program comprises the following 5 steps:
  - Step 0 (optional): Register the driver class
  - Step 1: Create the `Connection` object
  - Step 2: Create the `Statement` object
  - Step 3: Execute the statement
  - Step 4: Process the query result
  - Step 5: Close the connection



SELECT Statement (1)

- Use `executeQuery()` method and access results using column indices (first column is 1)
- Example:
  - ```
    try (
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/university",
            "username", "password");
        Statement stmt = conn.createStatement();
    ) {
        String query = "select id, name, tot_cred from student where id between 100
    and 200";
        ResultSet rset = stmt.executeQuery(query);
        while (rset.next()) {
            System.out.printf("%d, %s, %d%n",
                rset.getInt(1),
                rset.getString(2),
                rset.getInt(3));
        }
    }
    ```
    ```

SELECT Statement (2)

- Use `executeQuery()` method and access results using column names
- Example:

```
try (
    Connection conn = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/university",
        "username", "password");
    Statement stmt = conn.createStatement();
) {
    String query = "select id, name, tot_cred from student where id
between 100 and 200";
    ResultSet rset = stmt.executeQuery(query);
    while (rset.next()) {
        System.out.printf("%d, %s, %d%n",
            rset.getInt("id"),
            rset.getString("name"),
            rset.getInt("tot_cred"));
    }
}
```

INSERT, UPDATE and DELETE Statements

- These statements make updates to data, and return the number of affected rows
  - Use `executeUpdate()` method
- Example:
  - ```
    String query = "update student set tot_cred = 50 where id = 20";
    int numUpdated = stmt.executeUpdate(query);
    ```
    ```

Prepared Statements

- For better performance and security
- Example:
  - ```
    String query = "select dept_name from student where id between ? and ?";
    try (
        Connection conn = DriverManager.getConnection(...);
        PreparedStatement stmt = conn.prepareStatement(query)
    ) {
        stmt.setInt(1, 100);
        stmt.setInt(2, 200);
        ResultSet rset = stmt.executeQuery(query);
        // ...

        stmt.setInt(1, 300);
        stmt.setInt(2, 400);
        rset = stmt.executeQuery(query);
        // ...
    }
    ```
    ```

Batching

- Is to repeat a statement multiple times with different parameters, by collecting them together, then issue them all at once
- Example:

```
String query = "insert into department(dept_name, building) values (?, ?)";
try (
    Connection conn = DriverManager.getConnection(...);
    PreparedStatement stmt = conn.prepareStatement(query)
) {
    stmt.setString(1, "...");
    stmt.setString(2, "...");
    stmt.addBatch();

    stmt.setString(1, "...");
    stmt.setString(2, "...");
    stmt.addBatch();

    int[]  rowsAffected = stmt.executeBatch();
    // ...
}
```

Transactions

- Use transactions to wrap a set of updates in an interaction that either succeeds or fails altogether
  - By default, auto-commit is on, which means whenever an `executeUpdate()` is run, the command is committed
  - For a transaction, turn off auto-commit, then manually call `commit()` all is done
- Example:

```
conn.setAutoCommit(false);
stmt.executeUpdate(query1);
stmt.executeUpdate(query2);
stmt.executeUpdate(query3);
conn.commit();
```

Exercise

- Write a simple console program to add, remove, edit and search students