JDBC

Java Database Connectivity

Three-tier Architecture

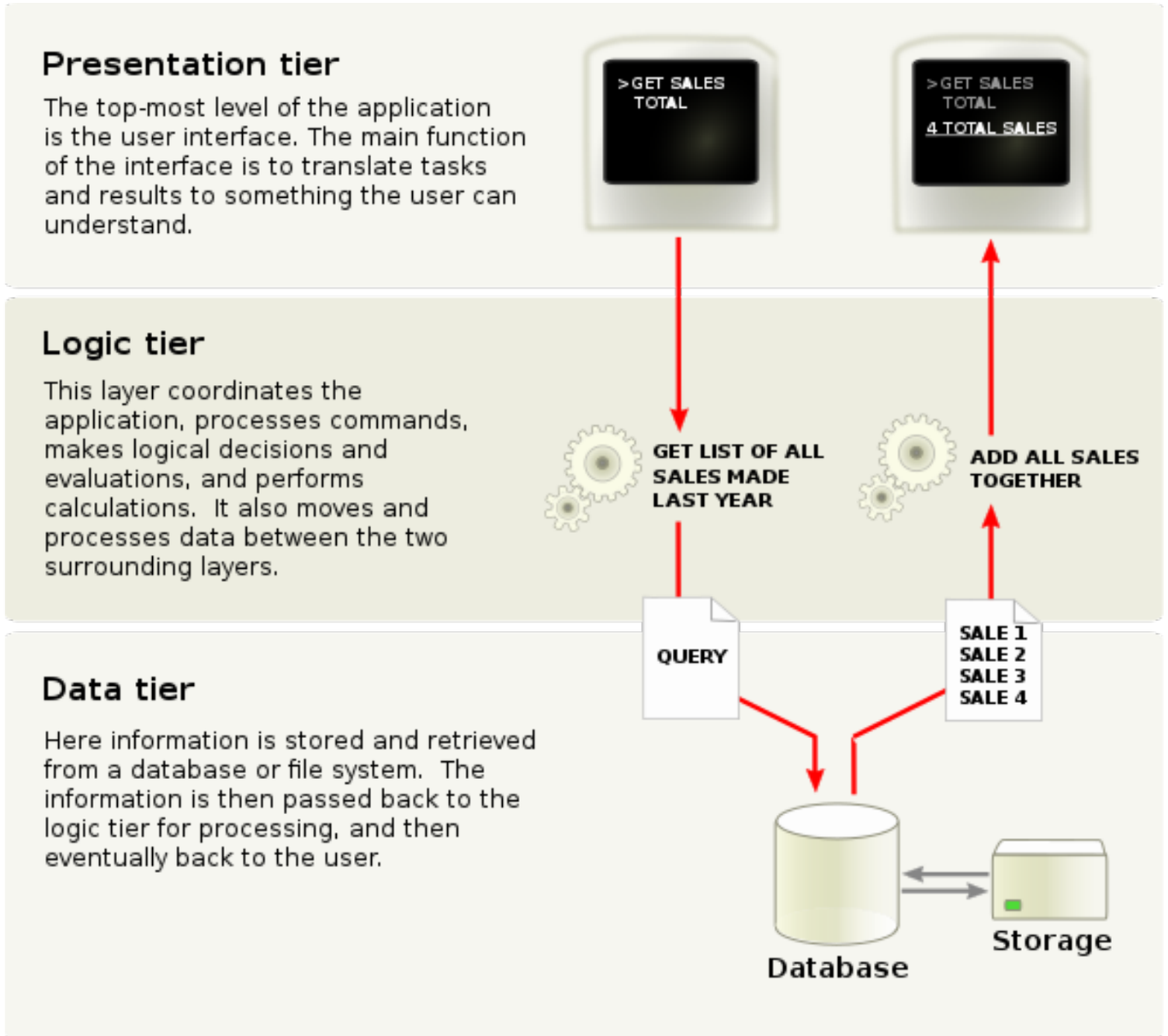

Image source: https://en.wikipedia.org/wiki/Multitier_architecture

Database Commands

- Structured Query Language (SQL)
  - select
  - insert
  - update
  - delete
- Data Definition Language
  - create table
  - drop table

SQL Select

- Used to retrieve data from a database

  select [fields]
  from [tablename]
  where [condition];

  select * from customers;

  select name, phone from customers where state=“ND”;

  select distinct city from customers;

  select count(*) from customers;

- The data returned is a table called a result set
  - java.sql.ResultSet

SQL Insert

- Used to add new records to a table.
- Add a full record

```
insert into [tablename]
values (val1, val2, val3, …);
```

- Add a partial record (only specified fields)

```
insert into [tablename] (col1, col4)
values (val1, val4);
```

SQL Update

- To modify existing records in a table

```
update [tablename]
set col1=val1, col2=val2
where condition;
```

- Update a single record

```
update customers
set contact="George O", phone="701-555-1234"
where custid=1;
```

- Update multiple records

```
update sales_staff set sales_rep="Julie"
where territory="MN";
```

SQL Delete

- Delete existing records from a table

```
delete from [tablename]
where [condition];
```

- If you omit “where…”, all records will be deleted!

  - The structure of the table remains.

DDL Create Table

- Create a new table in a database

```
create table [tablename] (
    col1 datatype constraint,
    col2 datatype constraint,
    ...
);
```

- Data types
    - int for a Java integer
    - varchar([size]) for a Java String
- Constraints
    - not null, primary key, foreign key, …
    - Some constraints have DBMS specific formats

DDL Drop Table

- Completely delete a table from a database

drop table [tablename];

- Contents and structure

Getting Started

- Install your preferred DBMS
- Install a JDBC driver from your DBMS vendor on the application host
- 4 driver types
    - Type 1. Implements the JDBC API as a bridge to another database access method (ODBC). Dependent on native code.
    - Type 2. Written partially in Java, partially in native code. Limited portability.
    - Type 3. Pure Java. Communicate with a middleware server to relay requests to the DBMS.
    - Type 4. Pure Java. Implement a network protocol for a specific DBMS.
- The JDBC driver is in a Jar file. Include the Jar file in your CLASSPATH.

JDBC Overview

- Package java.sql;
- DriverManager - easier to get started
- Connection
- DatabaseMetaData
- Statement
    - Submit SQL commands to the database
- ResultSet
    - A “table” with as many columns as you specified in your query.
    - ResultSet.next() returns true if there is another row in the “table”
    - A curser (pointer) to the next record in the ResultSet
    - Retrieve values from fields in a record.
        - getInt(String), getString(String), getFloat(String)

Getting a Connection

```
public Connection connect(
            String dbUrl, String userId, String passwd)
            throws SQLException {
    Connection conn = DriverManager.getConnection(dbUrl, userId, passed);
    return conn;
}
```

getConnection() is a static method

dbUrl format is DBMS-specific

"jdbc:mysql://localhost:3306/salesproject"

SQLException

- JDBC throws SQLException instead of Exception
- An SQLException has
  - getMessage() — a description of the error
  - getSQLState() — A standardized alphanumeric string code
  - getErrorCode() — An implementation-specific integer code.
  - getCause() — Possible causal relation to this exception.
  - getNextException() — Chained exceptions.

Statement Types

- Statement: Used to implement simple SQL statements with no parameters.

- PreparedStatement: For precompiled SQL statements that might have input parameters.

- CallableStatement: To execute stored procedures that may have both input and output parameters.

Getting a Statement

```
Statement stmt = null;
try {
    ...
    stmt = conn.createStatement();
    ...
} catch (SQLException e) {
    System.err.println(e.getMessage());
} finally {
    if (stmt != null) {
        stmt.close();
    }
}
```

```
// try-with-resource, Java SE 7 and later
// auto closes resource when exiting block
try (Statement stmt = conn.createStatement()) {
    ...
} catch (SQLException e) {
    System.err.println(e.getMessage());
}
```

Executing Queries

- Statement.executeQuery()
  - Returns one ResultSet object.
- Statement.executeUpdate()
  - For submitting SQL INSERT, UPDATE, or DELETE statements.
  - Returns an integer for the number of rows affected

Statement.executeQuery

```
String query = “select * from customers”;

try (Statement stmt = conn.createStatement()) {
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()) {
        int cId = rs.getInt(“CustomerID”);
        String name = rs.getString(“CustomerName”);
        String contact = rs.getString(“CustomerContact”);
        String phone = rs.getString(“Phone”);
        String sales = rs.getFloat(“PreviousSales”);

        System.out.println(cId + “\t” + name);
        System.out.println(“\t\t” + sales);
        System.out.println(“\t\t” + contact + “\t” + phone);
    }
} catch (SQLException e) {
    System.err.println(e.getMessage());
}
```

Statement.executeUpdate

```
float sales = 2000.25f;
int custId = 23;
…
StringBuilder sb = “update customers set sales=”;
sb.append(sales);
sb.append(“ where customerId=“);
sb.append(custId);

try (Statement stmt = conn.createStatement()) {
    int count = stmt.executeUpdate(sb.toString());
    …
} catch (SQLException e) {
    System.err.println(e.getMessage());
}
```

PreparedStatements

- For SQL statements executed many times
- Precompiled. Better performance.

```
StringBuilder sb = “update customers”;
sb.append(“ set sales = ? where custom = ?”);

conn.setAutoCommit(false);
try (PreparedStatement updatePS = conn.prepareStatement) {
      for (Map.Entry<int, float> e : weeklySales.entrySet()) {
            updatePS.setFloat(1, e.getValue());
            updatePS.setInt(2, e.getKey());
            updatePS.executeUpdate();
      }
} catch (SQLException e) {
      System.err.println(e.getMessage());
}
conn.commit();
conn.setAutoCommit(true);
```