Architecture of Database Application

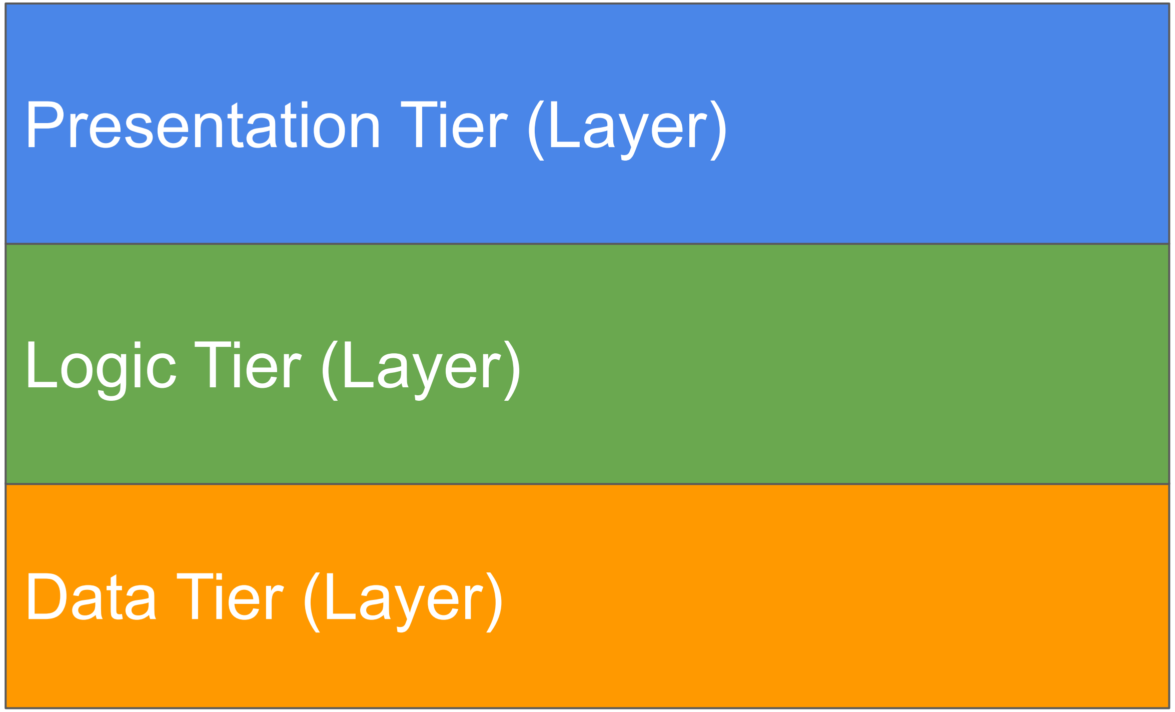
CSC 365

Multi - Tier (Layered) Architecture

- Application is split into multiple tiers / layers by concerns / responsibilities.
- “Tier” is also referred to as “Layer”, but some people make distinctions between the two.
- Often people refer to the multi-tier architecture as 3 tier architecture, splitting concerns/responsibilities into three.
- Each tier/layer should be able to be constructed separately, possibly by different teams of people with different skills.

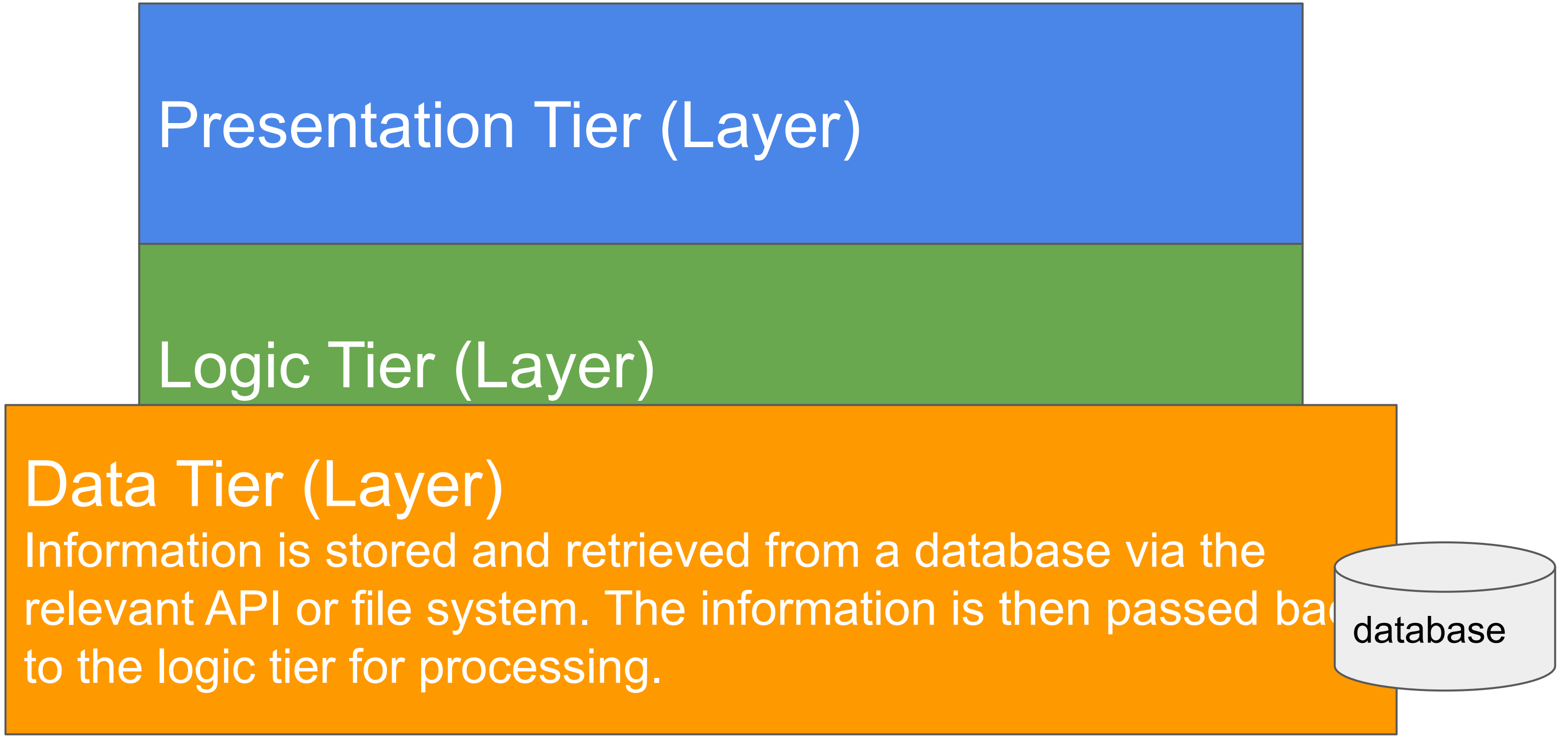
Multi - Tier (Layered) Architecture

Split application into multiple tiers / layers by concerns / responsibilities.



Multi - Tier (Layered) Architecture

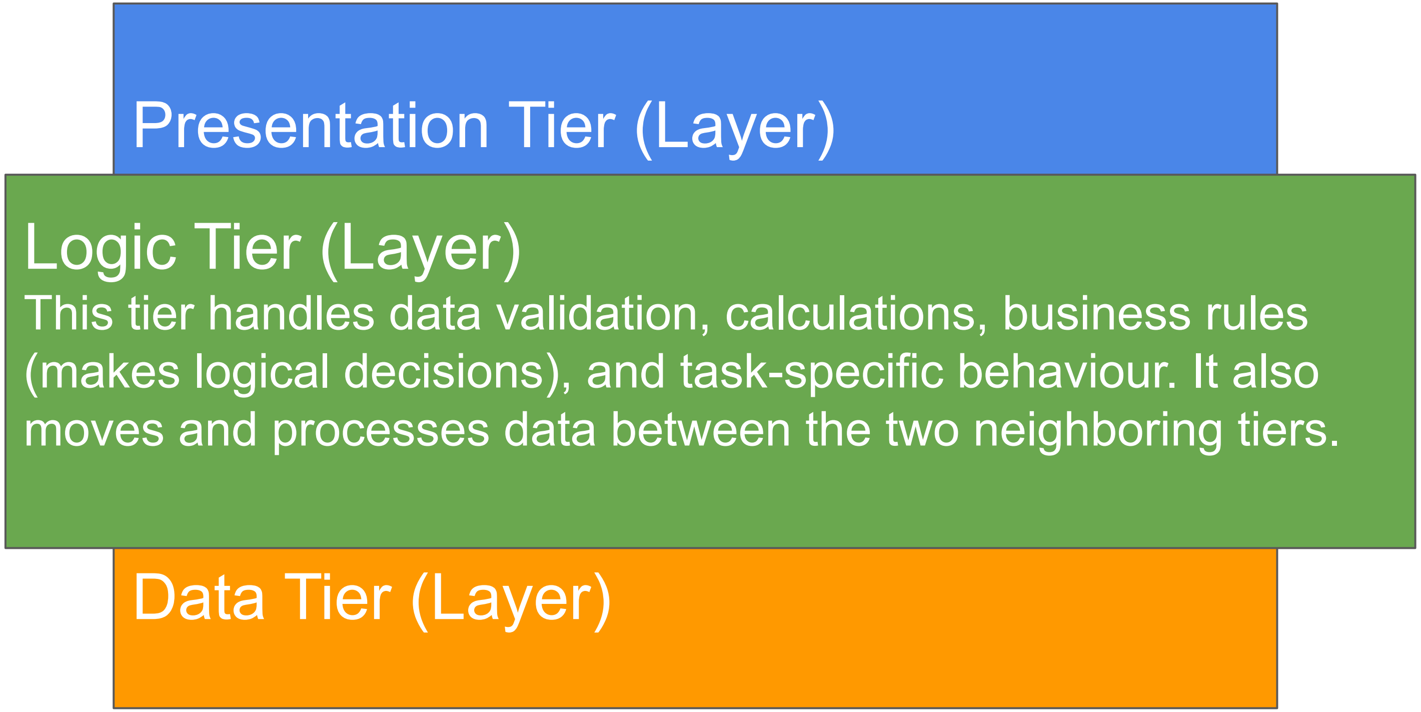
Split application into multiple tiers / layers by concerns / responsibilities.



Information is stored and retrieved from a database via the relevant API or file system. The information is then passed ba to the logic tier for processing.

Multi - Tier (Layered) Architecture

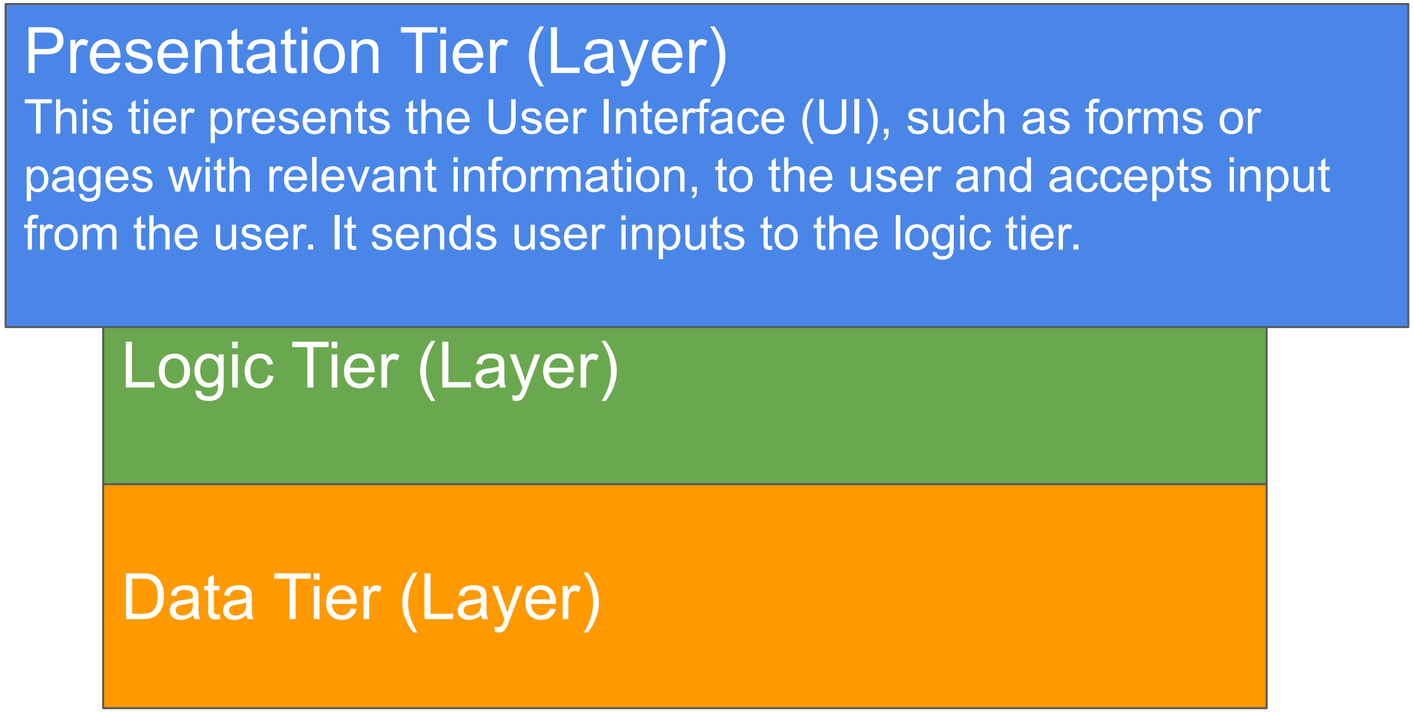
Split application into multiple tiers / layers by concerns / responsibilities.



This tier handles data validation, calculations, business rules (makes logical decisions), and task-specific behaviour. It also moves and processes data between the two neighboring tiers.

Multi - Tier (Layered) Architecture

Split application into multiple tiers / layers by concerns / responsibilities.



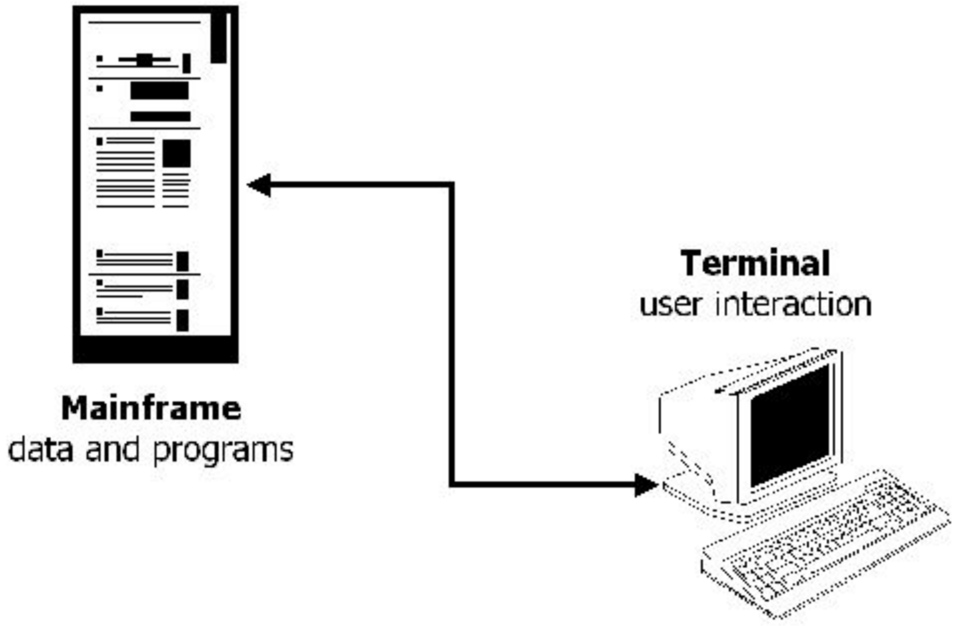
Presentation Tier (Layer)

This tier presents the User Interface (UI), such as forms or pages with relevant information, to the user and accepts input from the user. It sends user inputs to the logic tier.

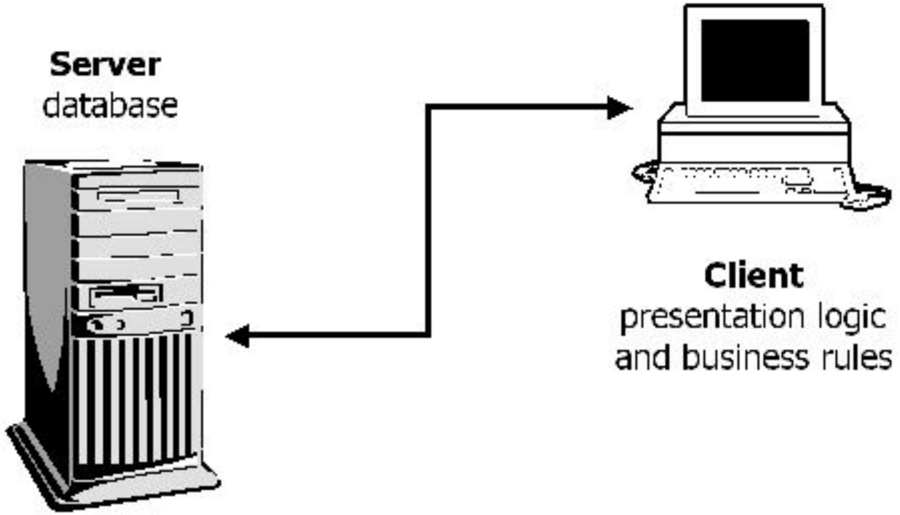
Logic Tier (Layer)

Data Tier (Layer)

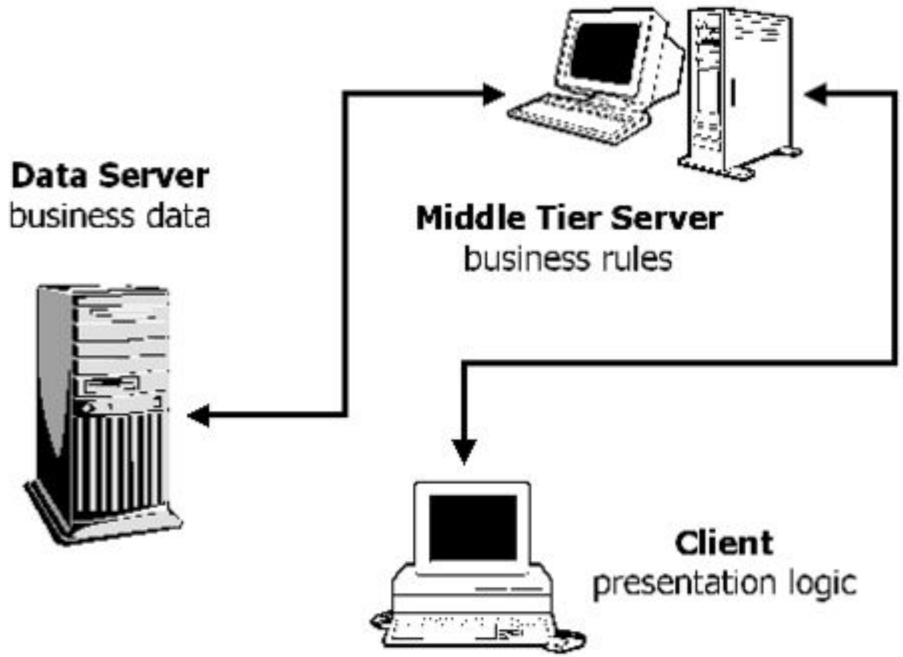
1 tier: Once upon a time ...



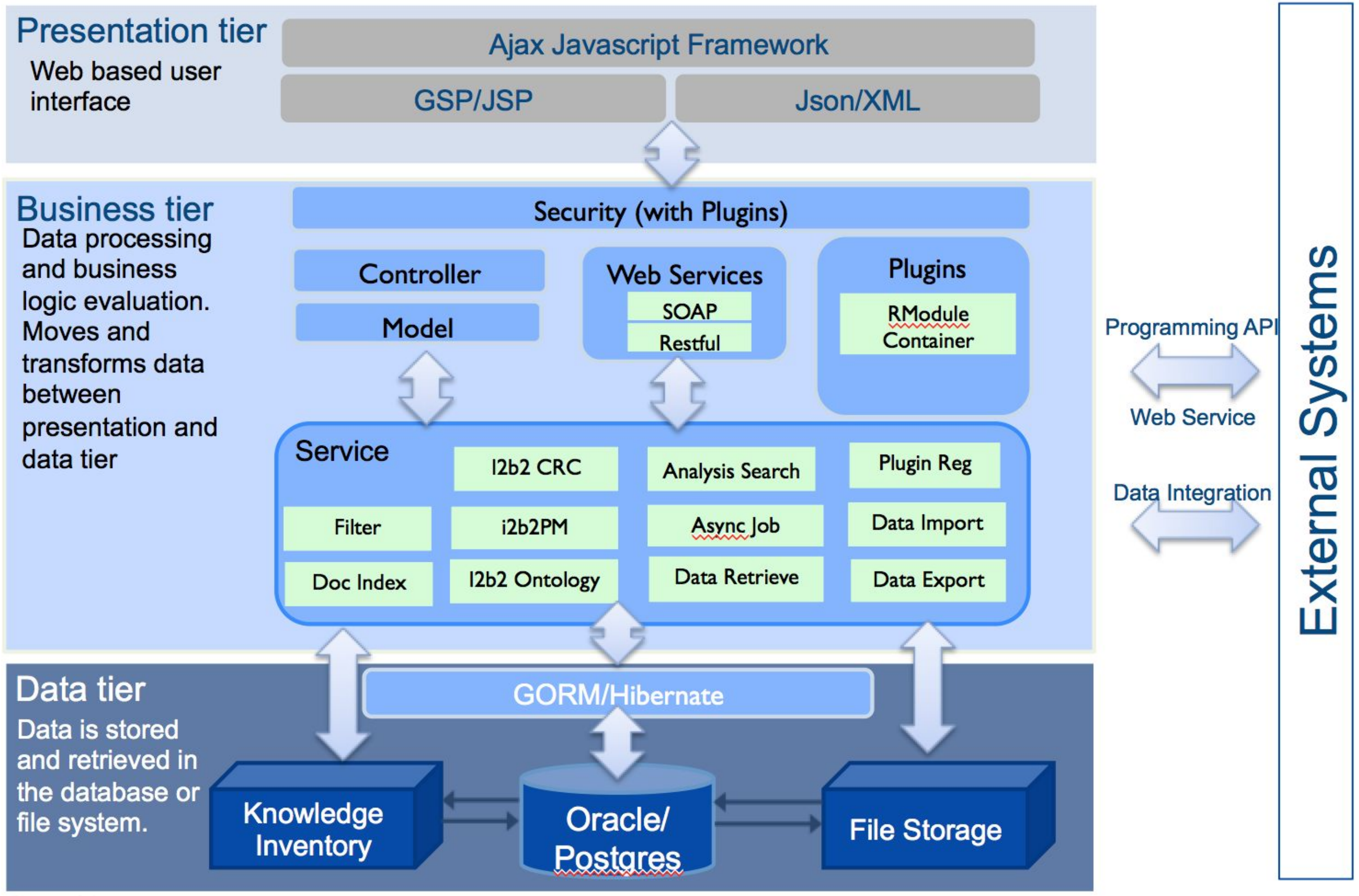
2 tier: 1980’s


Server
database
Client
presentation logic
and business rules

3 tier

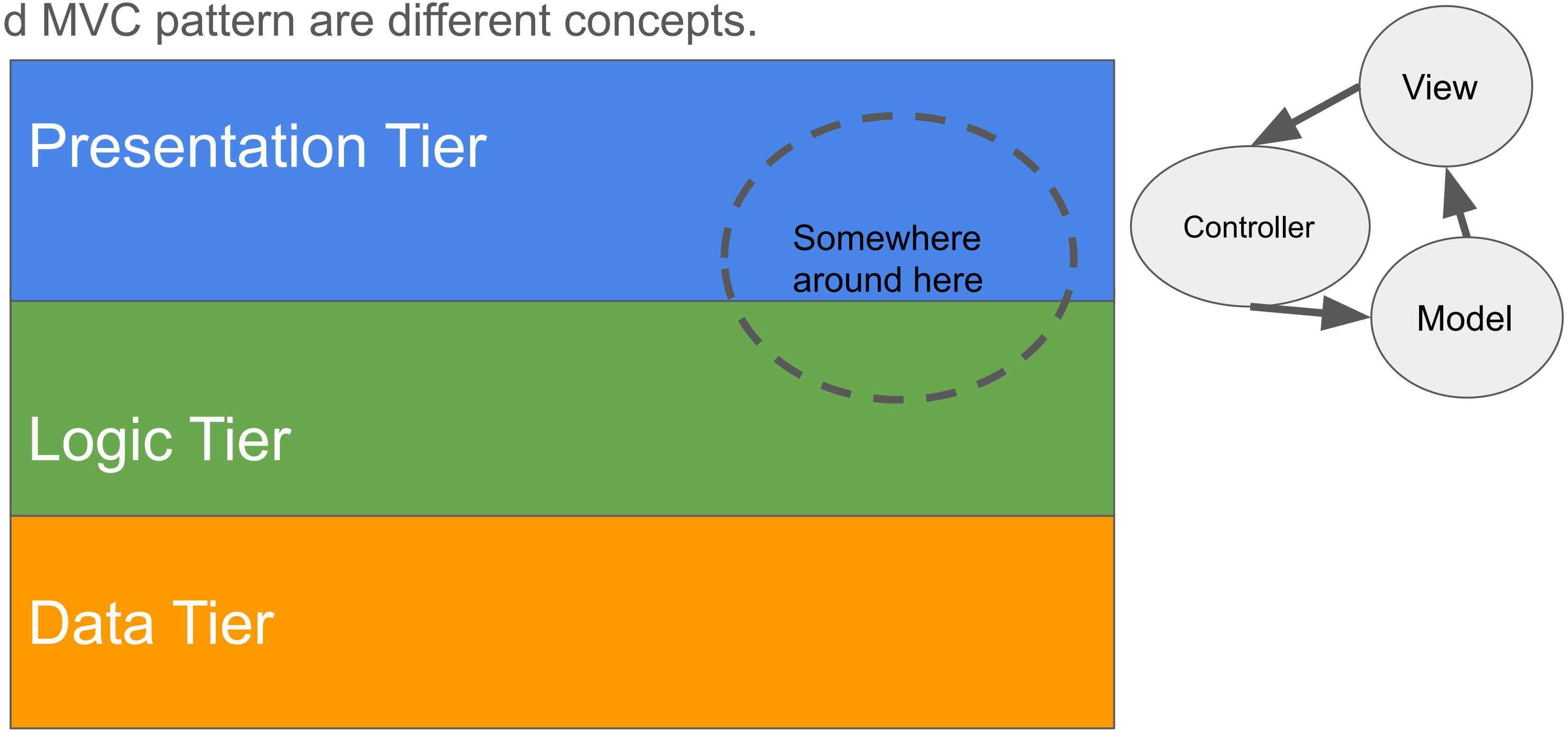

Data Server
business data
Middle Tier Server
business rules
Client
presentation logic

N - tier


Presentation tier
Web based user interface
Ajax Javascript Framework
GSP/JSP
Json/XML
Business tier
Data processing and business logic evaluation. Moves and transforms data between presentation and data tier
Security (with Plugins)
Controller
Model
Web Services
SOAP
Restful
Plugins
RModule Container
Service
I2b2 CRC
Analysis Search
Plugin Reg
Filter
i2b2PM
Async Job
Data Import
Doc Index
I2b2 Ontology
Data Retrieve
Data Export
Data tier
Data is stored and retrieved in the database or file system.
GORM/Hibernate
Knowledge Inventory
Oracle/ Postgres
File Storage
Programming API
Web Service
Data Integration
External Systems

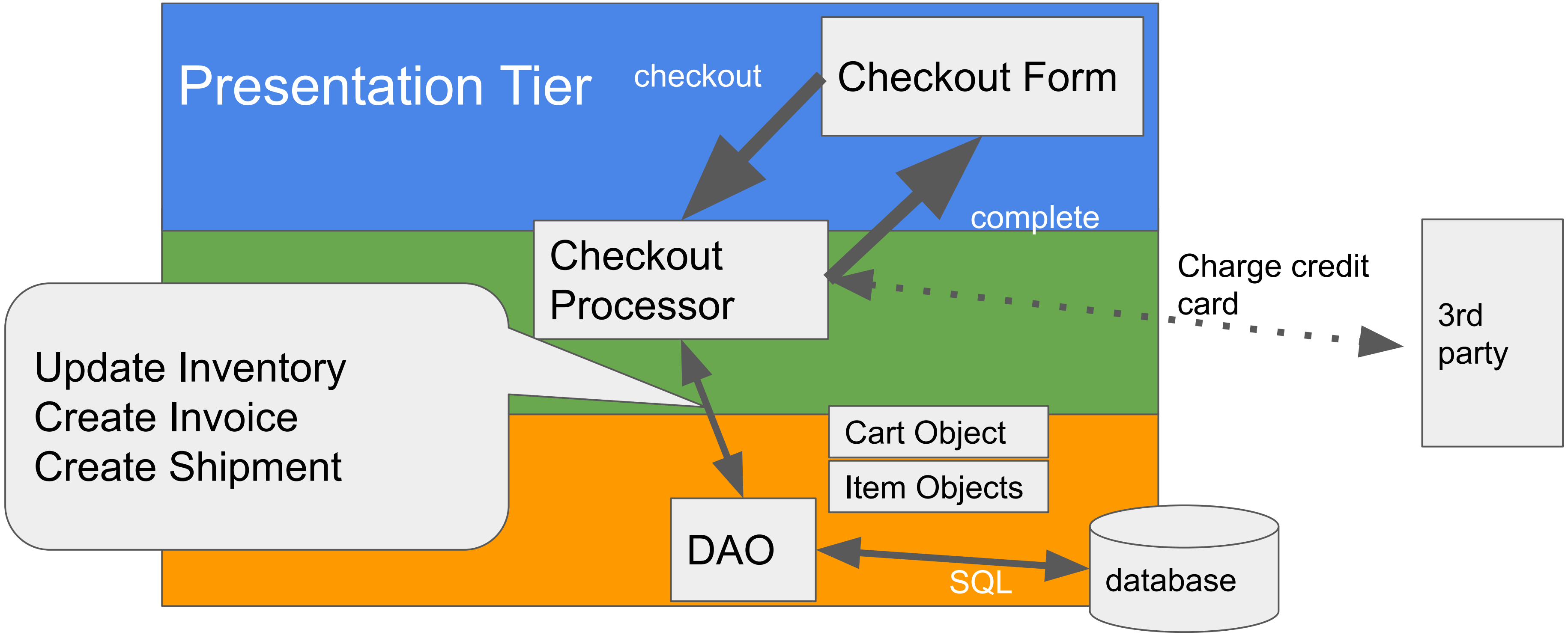
Where Model View Controller Pattern fits in

3 Tier and MVC pattern are different concepts.



3 tier Application Example

Split application into multiple tiers / layers by concerns / responsibilities.



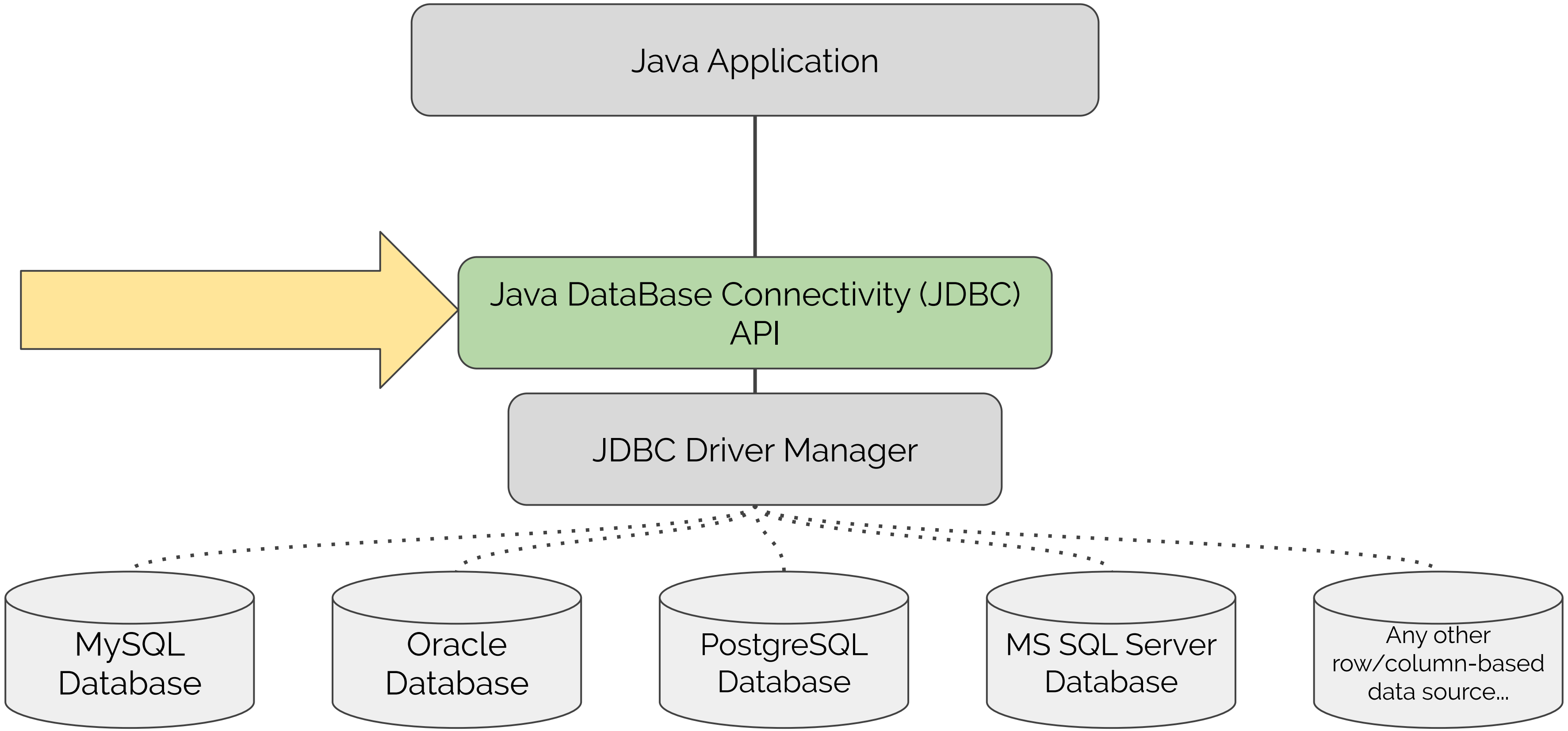
DAO

- Data Access Object
    - An object that provides an abstract interface to some type of database or other persistence mechanism.

Java DataBase Connectivity (JDBC)

- an application program interface (API) specification for connecting programs written in Java to the data in popular databases.
- The application program interface lets you encode access request statements in Structured Query Language (SQL) that are then passed to the program that manages the database.
- The results are returned through a similar interface.

JDBC


Java Application
Java DataBase Connectivity (JDBC) API
JDBC Driver Manager
MySQL Database
Oracle Database
PostgreSQL Database
MS SQL Server Database
Any other row/column-based data source...

Similar APIs available in other languages

Microsoft Open Database Connectivity (ODBC) (C language, predates JDBC)

Microsoft Object Linking and Embedding (OLE DB)

Python Database API Specification

PHP Data Objects

JDBC Driver

Database-specific details (networking protocols, connection mechanisms, data conversion, etc.) are handled by a **JDBC Driver**.

A JDBC Driver implementation is typically provided by the RDBMS vendor/developer.

JDBC MySQL Driver

To make use of the JDBC API, an appropriate JDBC Driver (provided as a JAR file) must be added to the Java application's classpath.

MySQL driver, known as Connector/J may be downloaded from:
https://dev.mysql.com/downloads/connector/j/

For Windows

1. Download the "latest" MySQL JDBC driver from http://dev.mysql.com/downloads ⇒ "MySQL Connectors" ⇒ "Connector/J" ⇒ Connector/J 8.0.{xx}, where {xx} is the latest update number ⇒ In "Select Operating System", choose "Platform Independent" ⇒ ZIP Archive (e.g., "mysql-connector-java-8.0.{xx}.zip" ⇒ "No thanks, just start my download".
2. UNZIP the download file into your project directory "C:\myWebProject".
3. Open the unzipped folder. Look for the JAR file "mysql-connector-java-8.0.{xx}.jar".
   a. The Absolute Full-path Filename for this JAR file is :
      i. "C:\myWebProject\mysql-connector-java-8.0.{xx}\mysql-connector-java-8.0.{xx}.jar".
   b. Take note of this super-long filename that you will need later. COPY and SAVE in a scratch pad so that you can retrieve later.

For Mac

1. Download the latest MySQL JDBC driver from http://dev.mysql.com/downloads ⇒ MySQL Connectors ⇒ Connector/J ⇒ Connector/J 8.0.{xx}, where {xx} is the latest update number ⇒ In "Select Operating System", choose "Platform Independent" ⇒ Compressed TAR Archive (e.g., mysql-connector-java-8.0.{xx}.tar.gz.
2. Double-click on the downloaded TAR file to expand into folder "mysql-connector-java-8.0.{xx}".
3. Move the expanded folder into your project directory "~/myWebProject".
4. Open the expanded folder. Look for the JAR file "mysql-connector-java-8.0.{xx}.jar".
   The Absolute Full-path Filename for this JAR file is :
   a. "~/myWebProject/mysql-connector-java-8.0.{xx}/mysql-connector-java-8.0.{xx}.jar".
   b. Take note of this super-long filename that you will need later. COPY and SAVE in a scratch pad so that you can retrieve later.

Using JDBC

For JDK 8 and lower, you can copy the JAR file into JDK's extension directory at:

- "<JAVA_HOME>\jre\lib\ext" (Windows)
- "/Library/Java/Extensions" (Mac OS X).

Using JDBC - Setting CLASSPATH

Once you have the JDBC Driver, there are a few ways to add it to your working Java classpath. Two simple options, using the MySQL Connector/J driver:

Option 1: Set the CLASSPATH environment variable:

```
  % export
CLASSPATH=$CLASSPATH:mysql-connector-java-5.1.44-bin.jar:.
  % java MyMainClass
```

Option 2: Use the -cp command line switch when running java:

```
  % java -cp mysql-connector-java-5.1.44-bin.jar:. MyMainClass
```

With Maven it is much easier

Create a maven project and put the following snippet in the pom.xml under inside <dependencies></dependencies>

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>

    <version>8.0.15</version>

</dependency>
```

JDBC Core Components

The core JDBC components are comprised of the following:

- **JDBC Driver**: This is a collection of classes that enables you to connect to a database and perform CRUD operations against it.
- **Connection**: This class is used to connect to a database using the JDBC API. Developers can obtain a connection to a database only after the JDBC driver for that database is loaded and initialized in JVM memory.
- **Statement**: A statement is used to execute the CRUD operations.
- **Result Set**: After developers have executed a query using the JDBC API, the result of the query is returned in the form of a ResultSet.

JDBC - Setting URL

A **JDBC URL** specifies database location and other connection parameters. Each vendor/driver extends the URL structure in its own way. Generally:

jdbc:<vendor|driver>:<server>[:<port>]/<database name>[?<param1=val1>...]

MySQL Example:
jdbc:mysql://*db.labthreesixfive.com*/*dbName*?autoReconnect=true

JDBC Classes

DriverManager	Service for managing JDBC drivers and establishing connections to a database
Connection	Represents a connection (session) with a specific database server.
Statement	Used to execute SQL statements and retrieve results. All SQL statements are executed within the context of a connection.
ResultSet	Represents a database result as a table of data. Maintains an internal cursor that allows you to move through the results.
PreparedStatement	A precompiled (and possibly parameterized) SQL statement that may be reused.
SQLException	Exception subclass that provides detail about database-related errors.

JDBC Extended Classes

DataSource	Operates as a connection "factory," supporting several connection modes: basic, pooled, distributed transaction.
RowSet	Disconnected, serializable version of a ResultSet (extends ResultSet class, operates without an active connection with the database, versus ResultSet's connected cursor mode)
PooledConnection	Connection with support for pooling (sharing a single connection across multiple clients, to avoid the cost of re-establishing a connection for each client)
XAConnection	Connection that provides support for distributed transactions (a topic for another lecture/course, or several…)

Sequence of Basic JDBC Program Steps

A JDBC program comprises the following steps:

1. Create a Connection object, for connecting to the database server.
2. Create a PreparedStatement object, under the Connection created earlier, for storing a SQL command.
3. Write a SQL query and execute the query, via the PreparedStatement and Connection created.
4. Process the query result using the ResultSet object returned by the execution of the SQL command.
5. Close the PreparedStatement, ResultSet and Connection to free up the resources.

Examples With Prepared Statement

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class PreparedStmtMoviesExample {
 // JDBC Driver Name & Database URL
 static final String JDBC_DRIVER =
"com.mysql.jdbc.Driver";
 static final String JDBC_DB_URL =
"jdbc:mysql://toshikuboi.net:3306/mydb";

 // JDBC Database Credentials
 static final String JDBC_USER = "yyy”;
 static final String JDBC_PASS = "xxx";
```

```
public static void main(String[] args) {
  try {
    //checks if the driver class is available
    Class.forName(JDBC_DRIVER);
   //get connection
    Connection connObj = DriverManager.getConnection(JDBC_DB_URL,
JDBC_USER, JDBC_PASS);

    //create prepared sql statement
    PreparedStatement prepStatement = connObj.prepareStatement(
      "SELECT * FROM Movies,StarsIn WHERE Movies.mid=StarsIn.mid
AND director=?");
    //plug in a parameter
    prepStatement.setString(1, "James Cameron");

    //execute the statement
    ResultSet resObj = prepStatement.executeQuery();
    //loop through the result set
    while(resObj.next()) {
      System.out.println(
        //you have to know the data type of each attribute
        "title: " + resObj.getString("title") + ", year: " + resObj.getInt("year")
        + ", actor: " + resObj.getString("sname"));
    }
   connObj.close();
  } catch (Exception sqlException) {
    sqlException.printStackTrace();
  }
 }
}
```

Examples With Prepared Statement

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class PreparedStmtMoviesExample {
 // JDBC Driver Name & Database URL
 static final String JDBC_DRIVER =
"com.mysql.jdbc.Driver";
 static final String JDBC_DB_URL =
"jdbc:mysql://toshikuboi.net:3306/mydb";

 // JDBC Database Credentials
 static final String JDBC_USER = "yyy";
 static final String JDBC_PASS = "xxx";
```

Driver Class

```
public static void main(String[] args) {
  try {
    //checks if the driver class is available
    Class.forName(JDBC_DRIVER);
                                                        ion(JDBC_DB_URL,

                                                        pareStatement(
       SELECT * FROM Movies,StarsIn WHERE Movies.mid=StarsIn.mid
AND director=?");
    //plug in a parameter
    prepStatement.setString(1, "James Cameron");

    //execute the statement
    ResultSet resObj = prepStatement.executeQuery();
    //loop through the result set
    while(resObj.next()) {
      System.out.println(
        //you have to know the data type of each attribute
        "title: " + resObj.getString("title") + ", year: " + resObj.getInt("year")
        + ", actor: " + resObj.getString("sname"));
    }
   connObj.close();
  } catch (Exception sqlException) {
    sqlException.printStackTrace();
  }
 }
}
```

Examples With Prepared Statement

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class PreparedStmtMoviesExample {
 // JDBC Driver Name & Database URL
 static final String JDBC_DRIVER =
"com.mysql.jdbc.Driver";
 static final String JDBC_DB_URL =
"jdbc:mysql://toshikuboi.net:3306/mydb";

 // JDBC Database Credentials
 static final String JDBC_USER = "yyy";
 static final String JDBC_PASS = "xxx";
```

```
public static void main(String[] args) {
  try {
    //checks if the driver class is available
    Class.forName(JDBC_DRIVER);
   //get connection
    Connection connObj = DriverManager.getConnection(JDBC_DB_URL,
                                                          eStatement(
                                                          .mid=StarsIn.mid
    prepStatement.setString(1, "James Cameron");

    //execute the statement
    ResultSet resObj = prepStatement.executeQuery();
    //loop through the result set
    while(resObj.next()) {
      System.out.println(
        //you have to know the data type of each attribute
        "title: " + resObj.getString("title") + ", year: " + resObj.getInt("year")
        + ", actor: " + resObj.getString("sname"));
    }
   connObj.close();
  } catch (Exception sqlException) {
    sqlException.printStackTrace();
  }
 }
}
```

JDBC DB URL

Examples With Prepared Statement

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class PreparedStmtMoviesExample {
 // JDBC Driver Name & Database URL
 static final String JDBC_DRIVER =
"com.mysql.jdbc.Driver";
 static final String JDBC_DB_URL =
"jdbc:mysql://toshikuboi.net:3306/myd

 // JDBC Database Credentials
 static final String JDBC_USER = "yyy";
 static final String JDBC_PASS = "xxx";
```

```
public static void main(String[] args) {
  try {
   //checks if the driver class is available
   Class.forName(JDBC_DRIVER);

   ResultSet resObj = prepStatement.executeQuery();
   //loop through the result set
   while(resObj.next()) {
    System.out.println(
     //you have to know the data type of each attribute
     "title: " + resObj.getString("title") + ", year: " + resObj.getInt("year")
     + ", actor: " + resObj.getString("sname"));
   }
  connObj.close();
  } catch (Exception sqlException) {
   sqlException.printStackTrace();
  }
 }
}
```

User and password for a database account

Step 1 : Create a Connection object

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class PreparedStmtMovi
 // J
 sta
"co
 sta
"jd

 // J
 sta
 sta
```

```
public static void main(String[] args) {
  try {
    //checks if the driver class is available
    Class.forName(JDBC_DRIVER);
   //get connection
    Connection connObj = DriverManager.getConnection(JDBC_DB_URL,
JDBC_USER, JDBC_PASS);

    //create prepared sql statement
```

```
          + ", actor: " + resObj.getString("sname"));
    }
   connObj.close();
  } catch (Exception sqlException) {
    sqlException.printStackTrace();
  }
 }
}
```

```
 //checks if the driver class is available
 Class.forName(JDBC_DRIVER);
//get connection
 Connection connObj = DriverManager.getConnection(
                    JDBC_DB_URL, JDBC_USER, JDBC_PASS);
```

Step 2 : Create a PreparedStatement object

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class PreparedStmtMoviesExample {
 // JDBC Driver Name & Database URL
 static final String JDBC_DRIVER =
"com.mysql.jdbc.Driver";
 static final String JDBC_DB_URL =
"jdbc:mysql://toshik
```

```java
public static void main(String[] args) {
  try {
    //checks if the driver class is available
    Class.forName(JDBC_DRIVER);
   //get connection
    Connection connObj = DriverManager.getConnection(JDBC_DB_URL,
JDBC_USER, JDBC_PASS);

    //create prepared sql statement
    PreparedStatement prepStatement = connObj.prepareStatement(
      "SELECT  * FROM Movies,StarsIn WHERE Movies.mid=StarsIn.mid
AND director=?");
    //plug in a parameter
    prepStatement.setString(1, "James Cameron");
```

```
}
}
```

```java
//create prepared sql statement
PreparedStatement prepStatement = connObj.prepareStatement(
      "SELECT  * FROM Movies,StarsIn
       WHERE Movies.mid=StarsIn.mid AND director=?");
//plug in a parameter
 prepStatement.setString(1, "James Cameron");
```

Step 3 : Write a SQL query and execute the query

```
//execute the statement
ResultSet resObj = prepStatement.executeQuery();
```

```
"jdbc:mysql://toshikubol.net:3306/mydb";

// JDBC Database Credentials
static final String JDBC_USER = "yyy";
static final String JDBC_PASS = "xxx";
```

```
public static void main(String[] args) {
      //execute the statement
      ResultSet resObj = prepStatement.executeQuery();
      //loop through the result set
      while(resObj.next()) {
        System.out.println(
          //you have to know the data type of each attribute
          "title: " + resObj.getString("title") + ", year: " + resObj.getInt("year")
          + ", actor: " + resObj.getString("sname"));
      }
     connObj.close();
    } catch (Exception sqlException) {
      sqlException.printStackTrace();
    }
  }
}
```

Step 4 : Process the query result via ResultSet

```
//loop through the result set
while(resObj.next()) {
  System.out.println(
    //you have to know the data type of each attribute
    "title: " + resObj.getString("title") + ", year: " + resObj.getInt("year")
    + ", actor: " + resObj.getString("sname"));
}
```

```
// JDBC Database Credentials
static final String JDBC_USER = "yyy";
static final String JDBC_PASS = "xxx";
```

```
    //loop through the result set
    while(resObj.next()) {
      System.out.println(
        //you have to know the data type of each attribute
        "title: " + resObj.getString("title") + ", year: " + resObj.getInt("year")
        + ", actor: " + resObj.getString("sname"));
    }
```

```
    connObj.close();
  } catch (Exception sqlException) {
    sqlException.printStackTrace();
  }
 }
}
```

Step 5 : Free up the resources created

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class PreparedStmtMoviesExample {
 // JDBC Driver Name & Database URL
 static final String JDBC_DRIVER =
"com.mysql.jdbc.Driver"
 static final String JDBC_
"jdbc:mysql://toshikubo

 // JDBC Database Creder
 static final String JDBC_
 static final String JDBC_
```

```
public static void main(String[] args) {
  try {
    //checks if the driver class is available
    Class.forName(JDBC_DRIVER);
   //get connection
    Connection connObj = DriverManager.getConnection(JDBC_DB_URL,
JDBC_USER, JDBC_PASS);

    //create prepared sql statement
    PreparedStatement prepStatement = connObj.prepareStatement(
      "SELECT * FROM Movies,StarsIn WHERE Movies.mid=StarsIn.mid
                                                                r")
    }
   connObj.close();
  } catch (Exception sqlException) {
    sqlException.printStackTrace();
  }
 }
}
```

```
connObj.close();
```

SQL / Java Data Type Mapping

ANSI SQL Data Type	**Java Type**
`CHAR, VARCHAR`	`String`
`INTEGER, FLOAT, DOUBLE`	`int, float, double`
`DECIMAL(p,s)`	`java.math.BigDecimal`
`BIT`	`boolean`
`DATE`	`java.sql.Date` (subclass of `java.util.Date`)
`TIME`	`java.sql.Time` (subclass of `java.util.Date`)
`DATETIME`	`java.sql.Timestamp` (subclass of `java.util.Date`)
`BINARY`	`byte[]`

ORM

- Object - Relational Mapping
  - A programming technique to bridge gap between the object model and the relational data model.
  - It converts data between relational databases and object oriented programming languages such as Java, C#, etc.

ORM Examples

- Hibernate (Java)
- EclipseLink (Java)
- SQLAlchemy (Python)
- Django (Python)
- DataMapper (Ruby)

Java Persistence API (JPA)

"The Java Persistence API (JPA) is a Java specification for accessing, persisting, and managing data between Java objects / classes and a relational database."

- API defined in the javax.persistence package
- Java Persistence Query Language (JPQL)
- Metadata / configuration

JPA standardized several earlier Java ORM libraries (notably: Hibernate, which continues to be in wide use as an implementation of the JPA standard)

ORM Query Languages (JPQL, HQL)

Java Persistence Query Language (JPQL) is the standard query language defined by JPA. Hibernate Query Language (HQL) is a similar language defined by the Hibernate ORM library, which predates the JPA standard.

JPQL and HQL are quite similar to SQL, but operate on class names, fields and class relationships instead of tables and columns.

Query in JPQL Example

```
List<Bill> findAllByStateAndTypeAndNumberOrderBySessionYearDesc(
        State state, String type, String number);
```