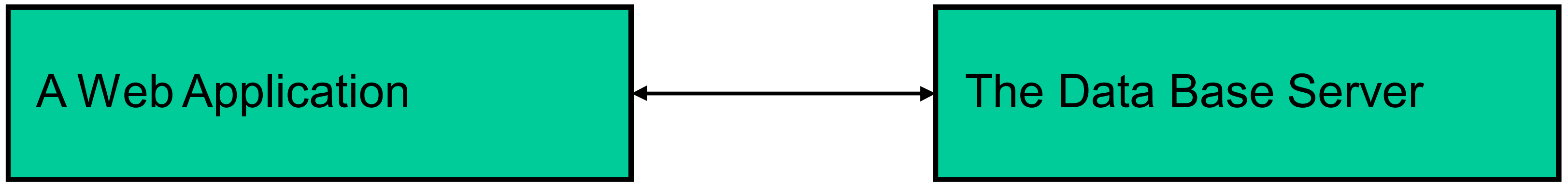
Мрежово програмиране

Връзка към бази от данни

доц. д-р Йордан Денев
denev@fmi.uni-sofia.bg

Data Base Access with a SQL interface

- The direct link



PHP + MYSQL

- ❑ Connecting to the server

```
mysql_connect($host,$user,$password);
```

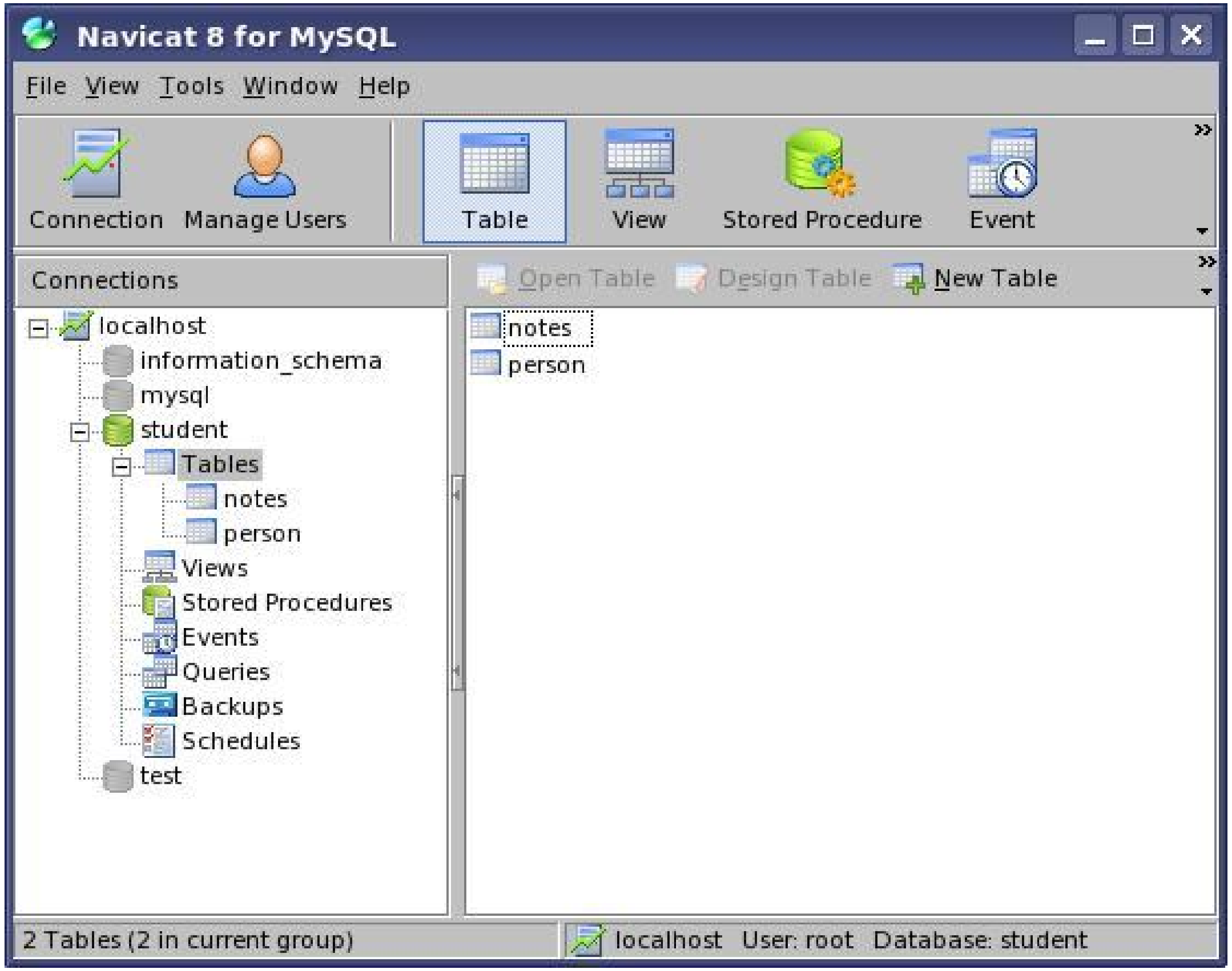
- ❑ Selecting the data base

```
mysql_select_db($database);
```

- ❑ Executing the SQL statements

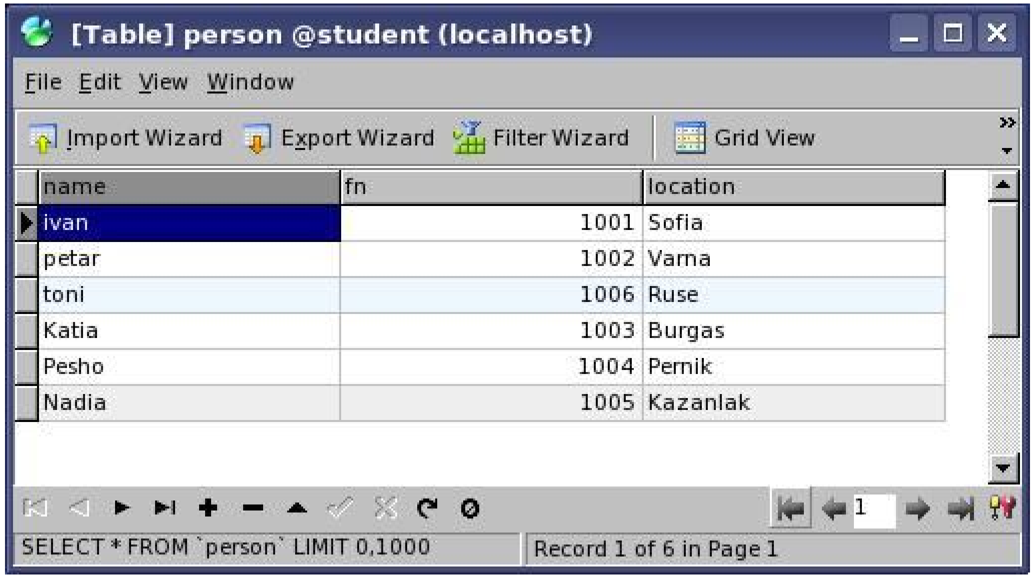
```
$result=mysql_query($SQL_expresion);
```

An example:



2 Tables (2 in current group)

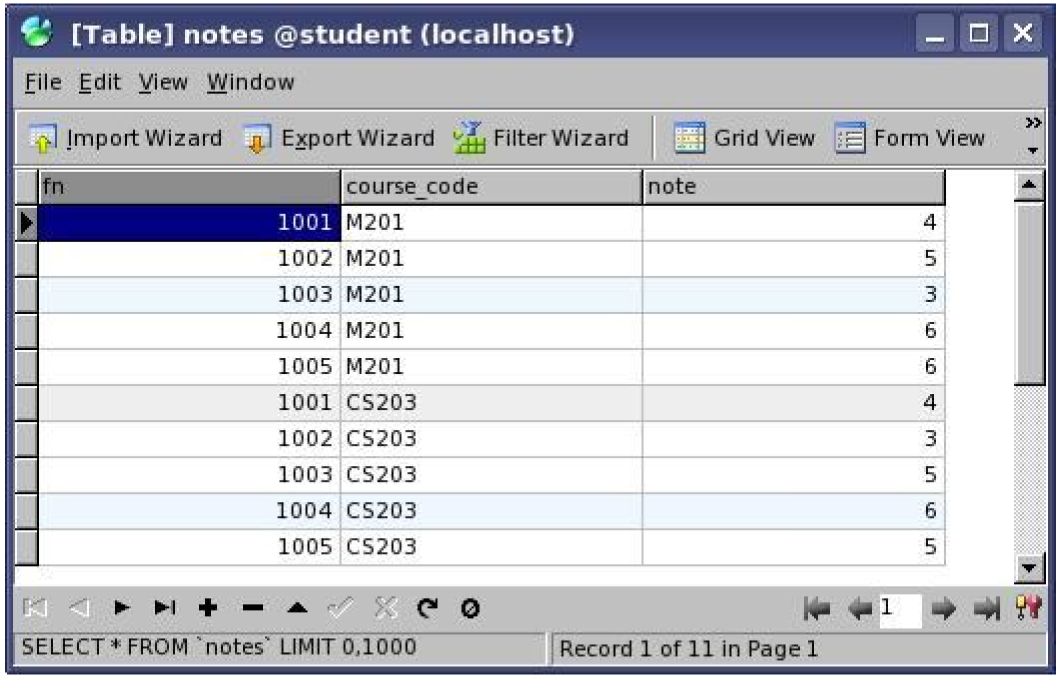
localhost User: root Database: student



name	fn	location
ivan	1001	Sofia
petar	1002	Varna
toni	1006	Ruse
Katia	1003	Burgas
Pesho	1004	Pernik
Nadia	1005	Kazanlak

```
SELECT * FROM `person` LIMIT 0,1000
```

Record 1 of 6 in Page 1



fn	course_code	note
1001	M201	4
1002	M201	5
1003	M201	3
1004	M201	6
1005	M201	6
1001	CS203	4
1002	CS203	3
1003	CS203	5
1004	CS203	6
1005	CS203	5

```
SELECT * FROM `notes` LIMIT 0,1000
```

Record 1 of 11 in Page 1

```
<?php
      $con = mysql_connect("localhost","root");
      $res=mysql_select_db("student");
      mysql_query("SELECT * FROM notes where
   fn=1001");
      echo mysql_affected_rows();
?>
```

- Affected rows:

```
echo mysql_affected_rows();
```

- Processing of the result:
  - mysql_fetch_row ($result) – next row, presented as an array;
  - mysql_fetch_object ($result) – next row, presented as an object;
  - mysql_fetch_array ($result)- next row, presented as an associative array;

An example:

```
$con = mysql_connect("localhost","root");
$res=mysql_select_db("student");
$result= mysql_query("SELECT *  FROM notes
      where fn=1001");
while ($row=mysql_fetch_array($result)) {
      echo $row["fn"],":",$row["course_code"],
      ":",$row["note"],"<BR>";
```

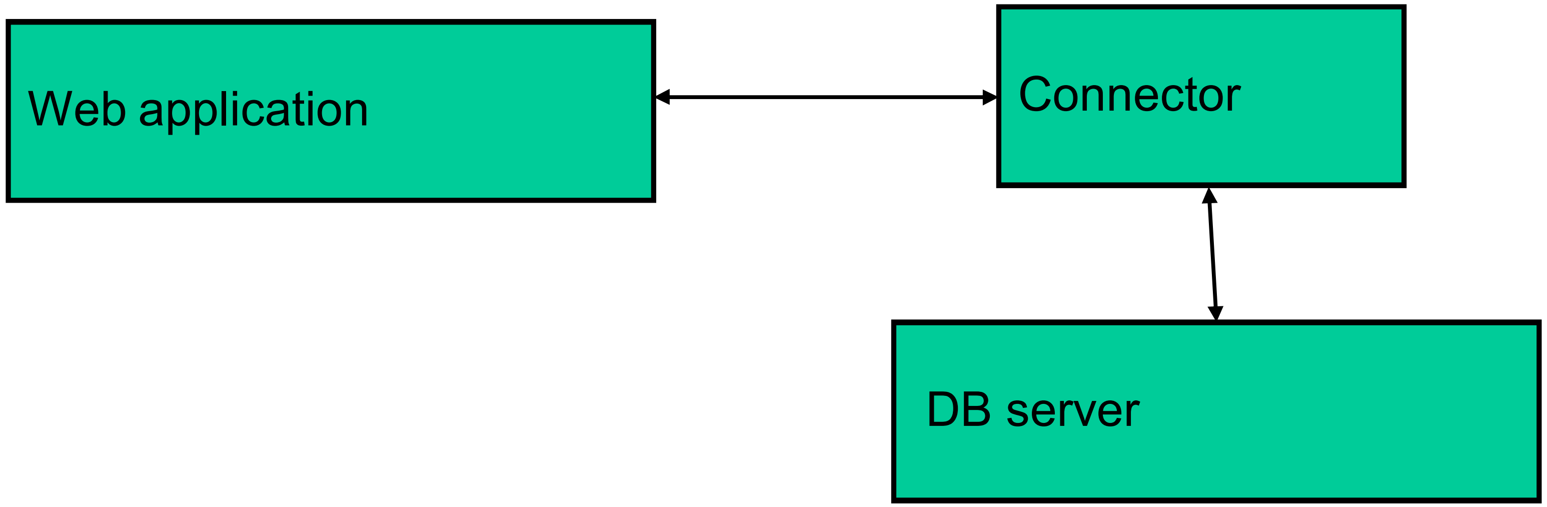
The output:

```
1001:M201:4
1001:CS203:4
```

❑ Drawbacks of the direct connection

PHP includes several specialized database-access interfaces that take the form of separate sets of functions for each database system. There is one set for MySQL, another for InterBase, another for PostgreSQL, and so forth. However, having a different set of functions for each database makes PHP scripts non-portable at the lexical (source code) level. For example, the function for issuing an SQL statement is named `mysql_query()`, `ibase_query()`, or `pg_exec()`, depending on whether you are using MySQL, InterBase, or PostgreSQL.

❑Connect trough a connector



- Most popular connectors – PDO, ODBC, JDBC

❑ PDO (PHP Data Objects)

PDO supports database access in an engine-independent manner based on a two-level architecture:

- The top level provides an interface that consists of a set of classes and methods that is the same for all database engines supported by PDO. The interface hides engine-specific details so that script writers need not think about which set of functions to use.
- The lower level consists of individual drivers. Each driver supports a particular database engine and translates between the top-level interface seen by script writers and the database-specific interface required by the engine. This provides you the flexibility of using any database for which a driver exists.

1. To establish a connection to a MySQL server, specify a data source name (DSN) containing connection parameters, and optionally the username and password of the MySQL account that you want to use

   ```
   $dbh = new PDO("mysql:host=localhost;dbname=test",
   "testuser", "testpass");
   ```

2. For statements that modify rows and produce no result set, pass the statement string to the database handle exec() method:

   ```
   $count = $dbh->exec ("some SQL statement");
   ```

3. For statements that select rows and produce a result set, invoke the database handle query() method, which executes the statement and returns an object of the PDOStatement class:

   ```
   $sth = $dbh->query ("some SQL statement");
   ```

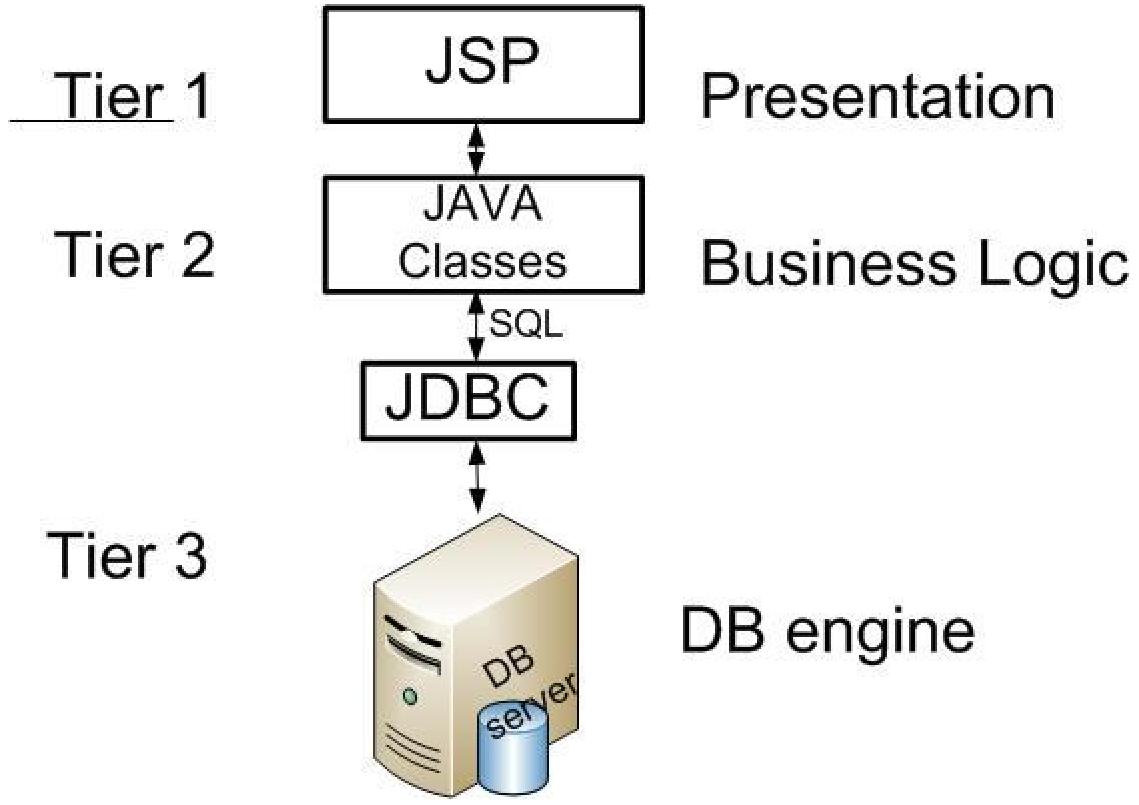
4. Work with the result

- `PDO::FETCH_NUM`
  Return each row of the result set as an array containing elements that correspond to the columns named in the SELECT statement and that are accessed by numeric indices beginning at 0:
  ```
  while ($row = $sth->fetch (PDO::FETCH_NUM))
  ```

-
  ```
  printf ("Name: %s, Category: %s\n", $row[0], $row[1]);
  ```
- `PDO::FETCH_ASSOC`
  Return each row as an array containing elements that are accessed by column name:
  ```
  while ($row = $sth->fetch (PDO::FETCH_ASSOC))

  printf ("Name: %s, Category: %s\n", $row["name"],
  $row["category"]);
  ```

Java SQL interface



JDBC – RDBMS Java Interface

- Connecting to the server
  - register the RDBMS driver (connector) you plan to use;
  - invoke its **`getConnection()`** method.

An example

```
import java.sql.*;
```

………………………………

```
Connection conn = null;
try {
   String userName = "testuser";
   String password = "testpass";
   String url = "jdbc:mysql://localhost/test";
   Class.forName ("com.mysql.jdbc.Driver").newInstance ();
   conn = DriverManager.getConnection (url, userName,
   password);
   System.out.println ("Database connection established");
} catch (Exception e) {
    System.err.println ("Cannot connect to database
   server");
}
```

- Issuing queries that return no result set
  - obtain a `Statement` object from the `Connection` object;
  - `executeUpdate()` is the appropriate method for issuing SQL statements, that modify the database.

An example

```
Statement s = conn.createStatement ();
int count;
s.executeUpdate ("DROP TABLE IF EXISTS animal");
s.executeUpdate
   ( "CREATE TABLE animal (‟
       + "id INT UNSIGNED NOT NULL AUTO_INCREMENT,"
       + "PRIMARY KEY (id),"
       + "name CHAR(40), category CHAR(40))");
count = s.executeUpdate (
    "INSERT INTO animal (name, category)"
       + " VALUES"
       + "('snake', 'reptile'),"
       + "('frog', 'amphibian'),"
       + "('tuna', 'fish'),"
       + "('racoon', 'mammal')");
s.close ();
System.out.println (count + " rows were inserted");
```

❑Issuing queries that return a result set

- For statements such as SELECT queries that retrieve information from the database, use `executeQuery()`.
- After calling this method, create a `ResultSet` object and use it to iterate through the rows returned by your query.
- To obtain the column values from each row, invoke `get`*`XXX`*`()` methods that match the column data types

An example

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT id, name, category FROM animal");
ResultSet rs = s.getResultSet ();
int count = 0;
while (rs.next ()) {
       int idVal = rs.getInt ("id");
       String nameVal = rs.getString ("name");
       String catVal = rs.getString ("category");
       System.out.println (
              "id = " + idVal + ",
              name = " + nameVal + ",
              category = " + catVal);
   ++count;
}
rs.close ();
s.close ();
System.out.println (count + " rows were retrieved");
```

An Object approach to the Data Base Access

- ❑ **Persistence** refers to the characteristic of data that outlives the execution of the program that created it. Without this capability, data only exists in RAM, and will be lost when the memory loses power, such as on computer shutdown.
- ❑ This is achieved in practice by storing the data in non-volatile storage such as a file system or a relational database or an object database.

Persistance data in object systems

- ❑In object oriented systems,we represent entities as objects and classes and use database to persist those objects. Most of the data-driven applications today,are written using object oriented technologies. The idea of representing entities as set of classes is to re-use the classes and objects once written.

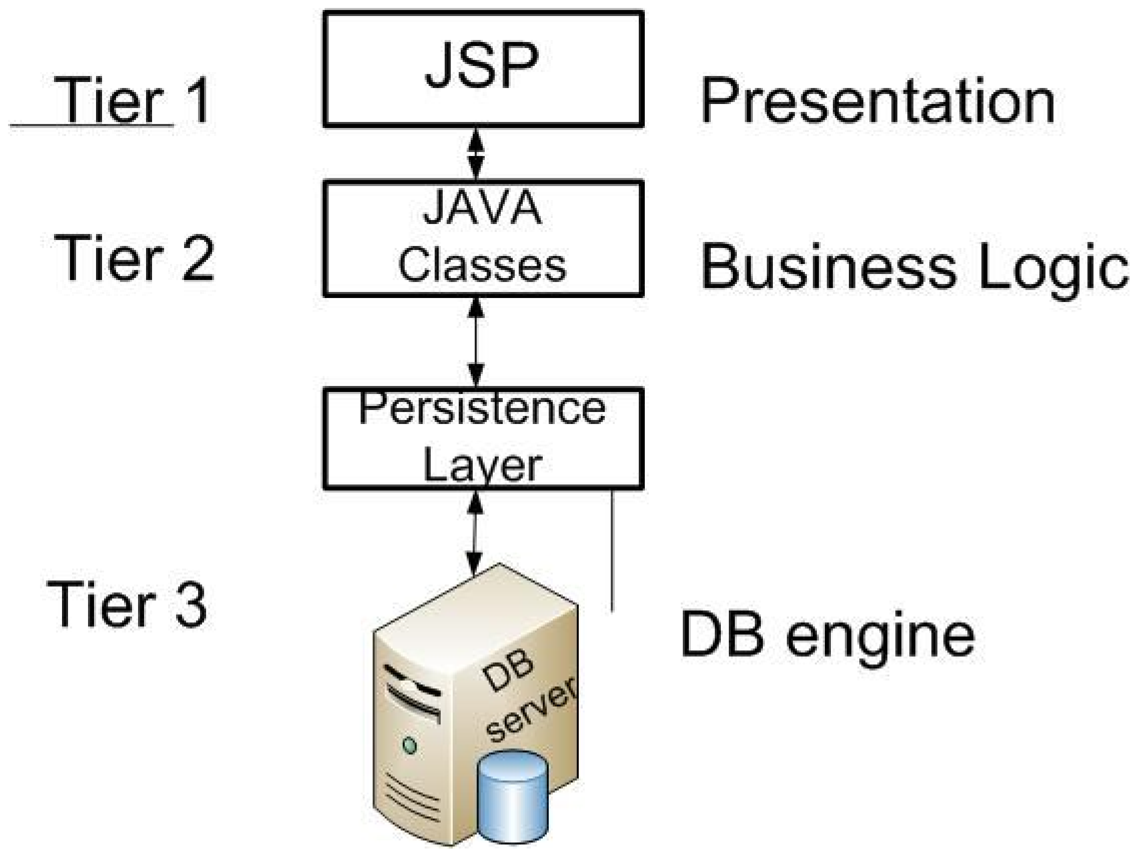
Object vs. Relational Model

- Objects usualy are not scalars.
- The object data are stored in several instances.
- The granularity problem comes when the number of classes mapping to number of tables in the database do not match.

(more detailed in http://www.lalitbhatt.com/tiki-index.php?page=Introduction+to+ORM)

The **Java Persistence API (JPA)**

- ❑It is a Java programming language framework that allows developers to manage relational data in Java Platforms, Standard Edition..
- ❑Persistence consists of three areas:
  - the API, defined in the javax.persistence package
  - the Java Persistence Query Language
  - object/relational metadata


Tier 1
JSP
Presentation
Tier 2
JAVA Classes
Business Logic
Persistence Layer
Tier 3
DB server
DB engine

Entities

- An entity is a lightweight persistence domain object.
- Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table.

An example

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity      // java annotation
@Table(name="EMPLOYEE TABLE")
public class Employee {
@Id
private int id;
private String name;

public Employee() {   }
public Employee(int id) {
this.id = id;   }

public int getId() {
return id;  }
public void setId(int id) {
this.id = id;  }

public String getName() {
return name;  }
public void setName(String name) {
this.name = name;  }
}
```

❑ Entity manager

- If you want the JPA framework to manage a particular entity instance, you have to put it explicitly under the control of a JPA component called *entity manager.*
- As long as the entity manager controls the entity, you can expect that changes to the entity will be synchronized with the database.
- Once this control ends, however, the entity is again nothing but a regular Java object.

An example - persisting a new entity

```
EntityManager em;
// set up a new entity instance
Employee person = new Employee(10);
person.setName("Miller");
// put it under the management of the entity manager
em.persist(person);
```

An example - finding an entity by Its unique identifier

```
EntityManager em;
// retrieve a managed entity instance
Employee person = em.find(Employee.class,
  Integer.valueOf(10));
if (person != null) {
// schedule the entity for removal
em.remove(person);
}
```