# CM0133 Internet Computing

Database Management
PHP & MySQL

## Objectives

- Transactions and Transaction Management
- Database Management System (What can it do for us)
- SQL
- PHP and MySQL
- Examples

## Database Management Systems (DBMS)

- A database management system (DBMS) supports reliable and efficient sharing of large sets of data among several users. In particular, a DBMS provides the following features:
  - persistency
  - efficient storage management
  - recovery
  - concurrency control
  - ad-hoc queries (e.g. SQL)
  - data security
- A DBMS allows to insert, retrieve and maintain data.

# Features of DBMS

- **Persistent storage** of data means that that data survive the execution of programs.
- **Efficient Storage Management:** Databases support efficient storage of large sets of data that do not fit entirely into main memory. Data is moved from a secondary storage e.g. disk to main memory using pages and buffers. There is a variety of buffering techniques that can not be covered in this course.

  Indexing techniqes are used to retrieve data from the disk. An index I associated to a data file D is an ordered file (a sequence of records) with entries ($k_i$,$p_i$) where $k_i$ is the value of the indexing field of a record in D and $p_i$ is the address of the block containing that record.

# Indices

- Indices provide fast access to our records (e.g. binary search).
- A rule for your web databases: If you do a lot of search on an attribute (column) then use an index! No matter which it will improve your access.
- There are many ways to index data and they will be covered in other lectures. You will come across clustering index, hashing, B* Trees (hierarchical multilevel index).
- MySQL mostly implements a B-Tree index, if you work with memory tables than hashing is used and if you work with spatial data MySQL uses R-Trees.

http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html

# Transaction: An Execution of a DB Program

- Key concept is *transaction,* which is an *atomic* sequence of database actions (reads/writes).
- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.
  - Users can specify some simple *integrity constraints* on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data.
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's/developer's responsibility!

## Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a transaction.
- Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of transactions:
  - Before a change is made to the database, the corresponding log entry is forced to a safe location.
  - After a crash, the effects of partially executed transactions are *undone* using the log. (the change was not applied to database but to the log itself!)

## Database Transactions - Atomicity

- **Atomicity**: Transactions are executed atomically. This means that either none of the actions of a transaction is carried out or all of them are carried out. Special commands are carried out to indicate the start of a transaction (begin transaction), the successful completion of a transaction (commit transaction), and the abort of a transaction (abort transaction).

## Consistency

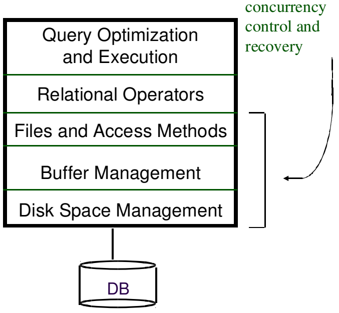
- Transactions move the database from one **consistent state** to another even if the database is accessed by several users simultaneously, executing several transactions interleaved (or in parallel). The traditional correctness criterion for executing several transactions interleaved is serializability.
- **Serializability** means that the overall effect of several transactions executed interleaved is the same as if these transaction had been executed in some serial order.

# Isolation & Durability

- Transactions are executed in **isolation**. Interim results of a transaction are not visible to other transactions. This means that effects of a transaction are visible to other transactions only after it has been completed successfully.
- **Durability** guarantees that once a transaction has been completed successfully, its effects remain persistent despite possible subsequent failures.

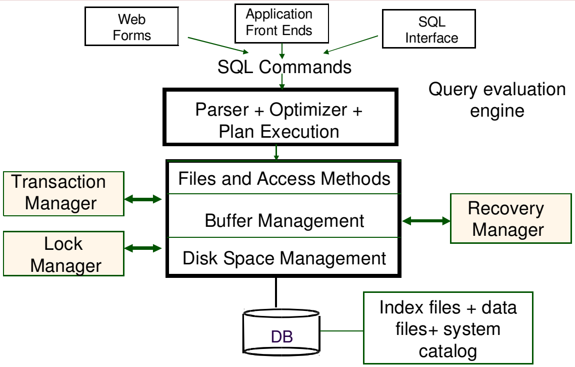
# Structure of a DBMS

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.



These layers must consider concurrency control and recovery

# Structure of a DBMS (cont.)



Query evaluation engine

## Question ?

How can you test if a database or information system supports data integrity and consistency?

## Reasons for a DBMS

- Changes to the type and format of data may occur frequently. Data independence is important
- Large amounts of data must be stored and be retrieved efficiently
- Data must be updated reliably. Inconsistent database states due to hardware and software failure are not tolerable.
- Date are accessed by several users simultaneously.
- Unexpected queries should be handled fast.
- Data are very sensitive. Data security is very important.

## Reasons against DBMS

- The amount of data is small.
- The application is very simple, no future changes to data types and data formats are expected.
- Concurrent access to the database is not required.
- The high costs of a data base management system (DBMS) are unjustified (although nowadays there are low cost solutions)
- The application has strict real time requirements and DBMS would be to slow.
- The application is very special and cannot be supported by a standard DBMS efficiently.

## Querying a DBMS

- A DBMS provides a Query Language.
- Query languages allow querying and updating a DMBS in a simple way.
- Most popular DML (Data Manipulation Language) : SQL(Structured Query Language).
- Queries:
  - List the name of student with sid=27373
  - Name and age of students enrolled in CM0133

  The following examples are SQL queries for MySQL. There might be a difference with another DBMS. MySQL often conforms with ANSI SQL standard.

## SQL – CREATE TABLE

```
CREATE TABLE 'CM0133'.'students' (
    'uid' BIGINT NOT NULL AUTO_INCREMENT ,
    'firstName' VARCHAR( 100 ) NOT NULL ,
    'surname' VARCHAR( 100 ) NOT NULL ,
    'address' TEXT NULL ,
   PRIMARY KEY ( 'uid' )
   );
```

**MySQL Data Types**

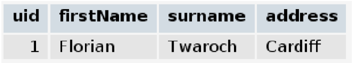
## CREATE TABLE

- Different database implementations support different data types. For our examples we can use integer (BIGINT), characters (VARCHAR (length)), Text, Date and Timestamp.
- NOT NULL indicates a constraint. Data has to be entered for this attribute. In our example key and full name has to be provided but not the address (NULL).
- AUTO_INCREMENT is a non standard SQL convenience function by MySQL that creates unique integers for you by incrementing.

# INSERT

```
INSERT INTO
CM0133.students (uid ,
firstName ,surname ,
address)
VALUES (
NULL , 'Florian', 'Twaroch',
'Cardiff'
);
```

- With INSERT we populate the created table.
- Note that NULL is entered for the uid, AUTOINCREMENT creates the value for us.



| uid | firstName | surname | address |
|---|---|---|---|
| 1 | Florian | Twaroch | Cardiff |

# UPDATE

```
UPDATE CM0133.students
SET address = 'Zurich'
WHERE students.uid =1;
```

- To change an entry we use the UPDATE command together with a condition.
- We have a number of operators at hand to support that
  - Logical operators: AND, OR, NOT
  - Equivalence op: ==, !=
  - Comparision op: >,<, etc.

# DELETE

```
DELETE * FROM
CM0133.students
WHERE students.uid =1;
```

- The DELETE command will delete entries. Again we can use conditions on which tuples we would like to delete.
- Here the user with the unique id 1 is deleted.

# SELECT

The SELECT command allows you to extract tuples from your database, e.g.:

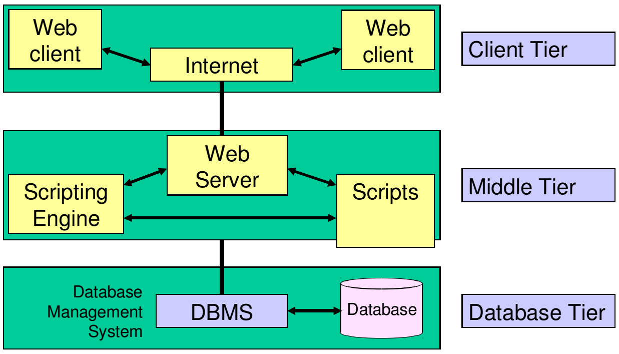
```
SELECT firstName, surname from CM0133.students
WHERE uid > 0 AND uid < 10
ORDER BY surname  DESC;
```

We will look closer at database queries in a tutorial and also how SQL queries interact with PHP in the next lectures.

# Connecting to Databases in PHP

- Connecting to databases in PHP is very straightforward
- Databases are required for storing large amounts of data and quickly retrieving large amounts of data
- Example database data may be:
  - Personal information
  - Financial details
  - Usernames/Passwords
  - Stock for an online shop
  - Web site content (content management systems)
- Before we continue, where do databases fit in with PHP and the internet?

# Three-tier model



Web client

Internet

Web client

Client Tier

Web Server

Scripting Engine

Scripts

Middle Tier

Database Management System

DBMS

Database

Database Tier

## MySQL

- MySQL is a database management system (DBMS) for relational databases, based on the Standard Query Language (SQL)
- MySQL is open source
- The focus of this course is NOT to learn SQL
- However: you can use these notes as a basis for making you sites interact with a database

## MySQL

- MySQL manages a system of relational databases
- A username and password are required to access the database system
- Each database contains tables
- Each table contains records (rows)
- Records are made up of fields
- Warning – don't use a database unless you need one!

## MySQL

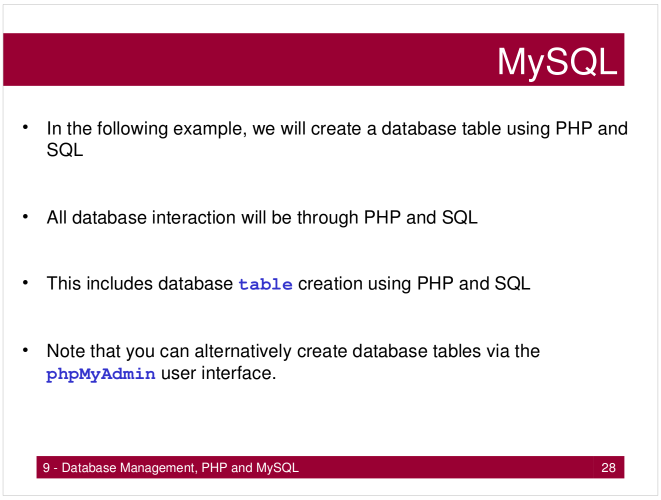
- phpAdmin provides an easy way to interact with and manage a MySQL database
  - provides an administrative interface to MySQL

You have access through http://www.cs.cf.ac.uk/phpMyAdmin/ and through program API.

- You can find notes describing how to use phpMyAdmin together with PHP at

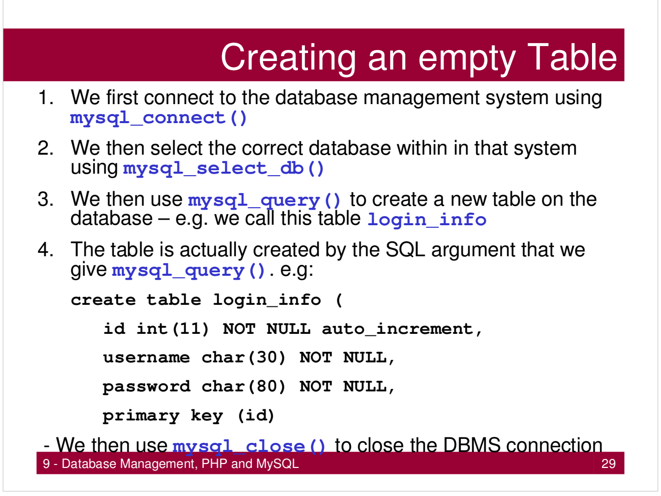
  http://docs.cs.cf.ac.uk/docs/notes/html/602

- Before you can use phpMyAdmin you first require a database to be created on the server – the administrator (e.g. Robert Evans) has to do this
- You then get a password to access the database



# MySQL

- In the following example, we will create a database table using PHP and SQL
- All database interaction will be through PHP and SQL
- This includes database `table` creation using PHP and SQL
- Note that you can alternatively create database tables via the `phpMyAdmin` user interface.

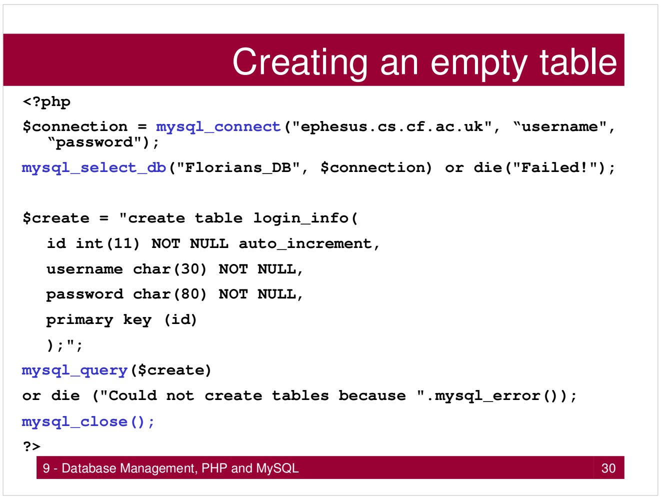


# Creating an empty Table

1. We first connect to the database management system using `mysql_connect()`
2. We then select the correct database within in that system using `mysql_select_db()`
3. We then use `mysql_query()` to create a new table on the database – e.g. we call this table `login_info`
4. The table is actually created by the SQL argument that we give `mysql_query()`. e.g:

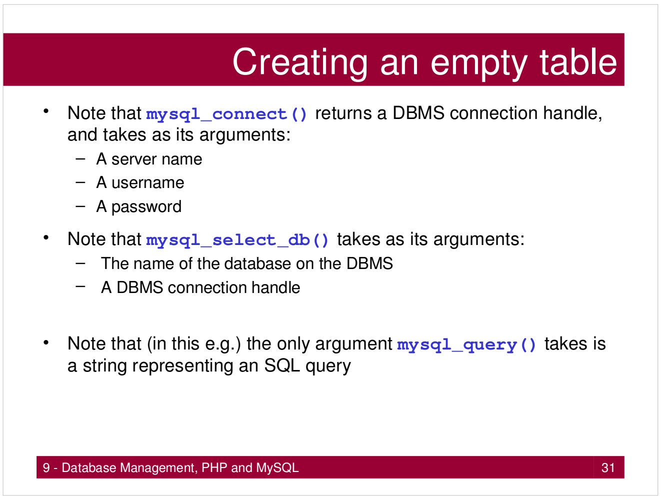
```
create table login_info (
   id int(11) NOT NULL auto_increment,
   username char(30) NOT NULL,
   password char(80) NOT NULL,
   primary key (id)
```

- We then use `mysql_close()` to close the DBMS connection



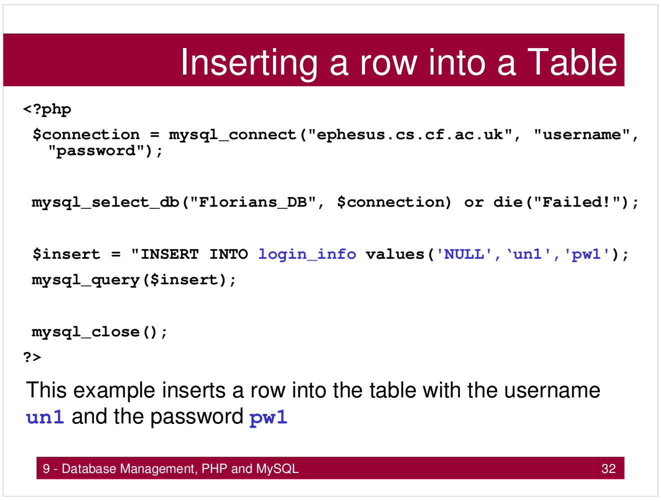
# Creating an empty table

```
<?php
$connection = mysql_connect("ephesus.cs.cf.ac.uk", "username",
   "password");
mysql_select_db("Florians_DB", $connection) or die("Failed!");

$create = "create table login_info(
   id int(11) NOT NULL auto_increment,
   username char(30) NOT NULL,
   password char(80) NOT NULL,
   primary key (id)
   );";
mysql_query($create)
or die ("Could not create tables because ".mysql_error());
mysql_close();
?>
```



## Creating an empty table

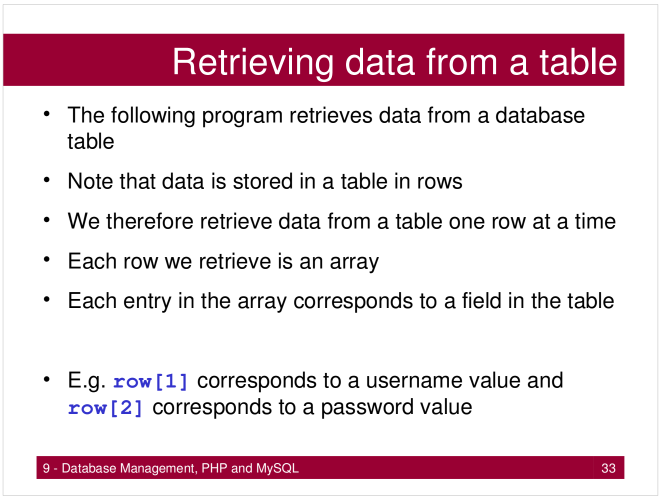
- Note that mysql_connect() returns a DBMS connection handle, and takes as its arguments:
  - A server name
  - A username
  - A password
- Note that mysql_select_db() takes as its arguments:
  - The name of the database on the DBMS
  - A DBMS connection handle
- Note that (in this e.g.) the only argument mysql_query() takes is a string representing an SQL query



## Inserting a row into a Table

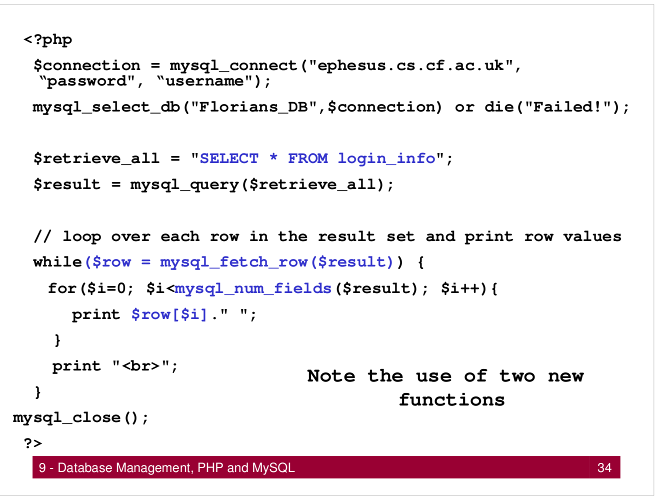
```
<?php
 $connection = mysql_connect("ephesus.cs.cf.ac.uk", "username",
   "password");

 mysql_select_db("Florians_DB", $connection) or die("Failed!");

 $insert = "INSERT INTO login_info values('NULL',‘un1','pw1');
 mysql_query($insert);

 mysql_close();
?>
```

This example inserts a row into the table with the username un1 and the password pw1



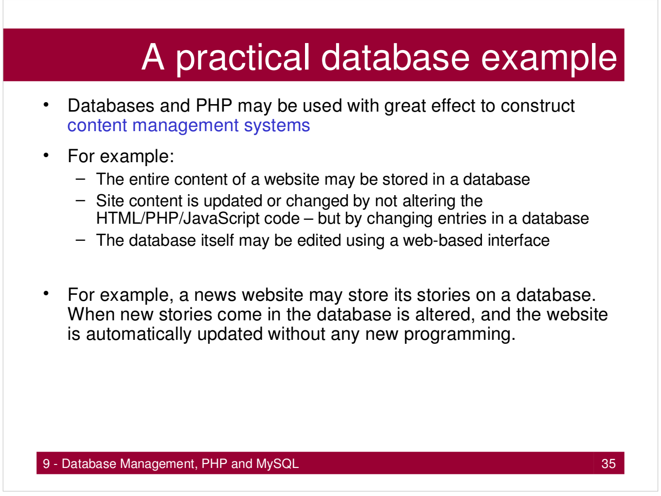
## Retrieving data from a table

- The following program retrieves data from a database table
- Note that data is stored in a table in rows
- We therefore retrieve data from a table one row at a time
- Each row we retrieve is an array
- Each entry in the array corresponds to a field in the table
- E.g. row[1] corresponds to a username value and row[2] corresponds to a password value



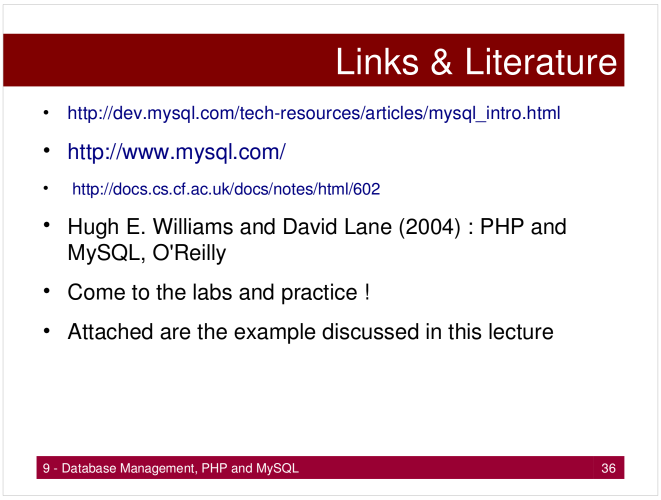
```
<?php
 $connection = mysql_connect("ephesus.cs.cf.ac.uk",
  "password", "username");
 mysql_select_db("Florians_DB",$connection) or die("Failed!");

 $retrieve_all = "SELECT * FROM login_info";
 $result = mysql_query($retrieve_all);

 // loop over each row in the result set and print row values
 while($row = mysql_fetch_row($result)) {
   for($i=0; $i<mysql_num_fields($result); $i++){
      print $row[$i]." ";
    }
    print "<br>";
 }
mysql_close();
?>
```

Note the use of two new functions



## A practical database example

- Databases and PHP may be used with great effect to construct content management systems
- For example:
  - The entire content of a website may be stored in a database
  - Site content is updated or changed by not altering the HTML/PHP/JavaScript code – but by changing entries in a database
  - The database itself may be edited using a web-based interface
- For example, a news website may store its stories on a database. When new stories come in the database is altered, and the website is automatically updated without any new programming.

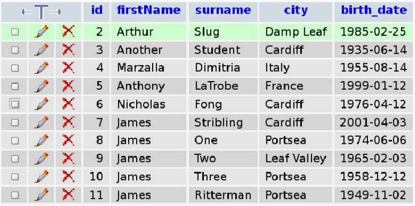


## Links & Literature

- http://dev.mysql.com/tech-resources/articles/mysql_intro.html
- http://www.mysql.com/
- http://docs.cs.cf.ac.uk/docs/notes/html/602
- Hugh E. Williams and David Lane (2004) : PHP and MySQL, O'Reilly
- Come to the labs and practice !
- Attached are the example discussed in this lecture

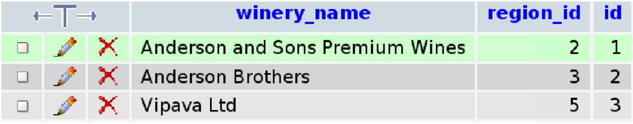
## Used Tables

customer



| ←T→ | id | firstName | surname | city | birth_date |
|---|---|---|---|---|---|
| □ ✎ ✕ | 2 | Arthur | Slug | Damp Leaf | 1985-02-25 |
| □ ✎ ✕ | 3 | Another | Student | Cardiff | 1935-06-14 |
| □ ✎ ✕ | 4 | Marzalla | Dimitria | Italy | 1955-08-14 |
| □ ✎ ✕ | 5 | Anthony | LaTrobe | France | 1999-01-12 |
| □ ✎ ✕ | 6 | Nicholas | Fong | Cardiff | 1976-04-12 |
| □ ✎ ✕ | 7 | James | Stribling | Cardiff | 2001-04-03 |
| □ ✎ ✕ | 8 | James | One | Portsea | 1974-06-06 |
| □ ✎ ✕ | 9 | James | Two | Leaf Valley | 1965-02-03 |
| □ ✎ ✕ | 10 | James | Three | Portsea | 1958-12-12 |
| □ ✎ ✕ | 11 | James | Ritterman | Portsea | 1949-11-02 |

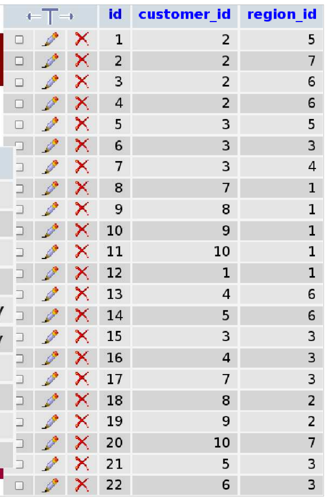
winery



| ←T→ | winery_name | region_id | id |
|---|---|---|---|
| □ ✎ ✕ | Anderson and Sons Premium Wines | 2 | 1 |
| □ ✎ ✕ | Anderson Brothers | 3 | 2 |
| □ ✎ ✕ | Vipava Ltd | 5 | 3 |

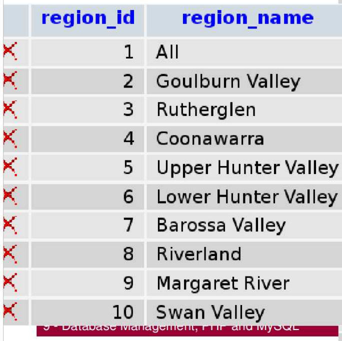
## Used Tables

orders



| ←T→ | id | customer_id | region_id |
|---|---|---|---|
| □ ✎ ✕ | 1 | 2 | 5 |
| □ ✎ ✕ | 2 | 2 | 7 |
| □ ✎ ✕ | 3 | 2 | 6 |
| □ ✎ ✕ | 4 | 2 | 6 |
| □ ✎ ✕ | 5 | 3 | 5 |
| □ ✎ ✕ | 6 | 3 | 3 |
| □ ✎ ✕ | 7 | 3 | 4 |
| □ ✎ ✕ | 8 | 7 | 1 |
| □ ✎ ✕ | 9 | 8 | 1 |
| □ ✎ ✕ | 10 | 9 | 1 |
| □ ✎ ✕ | 11 | 10 | 1 |
| □ ✎ ✕ | 12 | 1 | 1 |
| □ ✎ ✕ | 13 | 4 | 6 |
| □ ✎ ✕ | 14 | 5 | 6 |
| □ ✎ ✕ | 15 | 3 | 3 |
| □ ✎ ✕ | 16 | 4 | 3 |
| □ ✎ ✕ | 17 | 7 | 3 |
| □ ✎ ✕ | 18 | 8 | 2 |
| □ ✎ ✕ | 19 | 9 | 2 |
| □ ✎ ✕ | 20 | 10 | 7 |
| □ ✎ ✕ | 21 | 5 | 3 |
| □ ✎ ✕ | 22 | 6 | 3 |

region



|  | region_id | region_name |
|---|---|---|
| ✕ | 1 | All |
| ✕ | 2 | Goulburn Valley |
| ✕ | 3 | Rutherglen |
| ✕ | 4 | Coonawarra |
| ✕ | 5 | Upper Hunter Valley |
| ✕ | 6 | Lower Hunter Valley |
| ✕ | 7 | Barossa Valley |
| ✕ | 8 | Riverland |
| ✕ | 9 | Margaret River |
| ✕ | 10 | Swan Valley |

## INSPECT DB (command line)

```
SHOW databases;    # show all available databases
USE CM0133;        # select one

SHOW tables;       # show tables of selected database

DESCRIBE customer;    # describe one of the tables
```

## SQL EXAMPLES

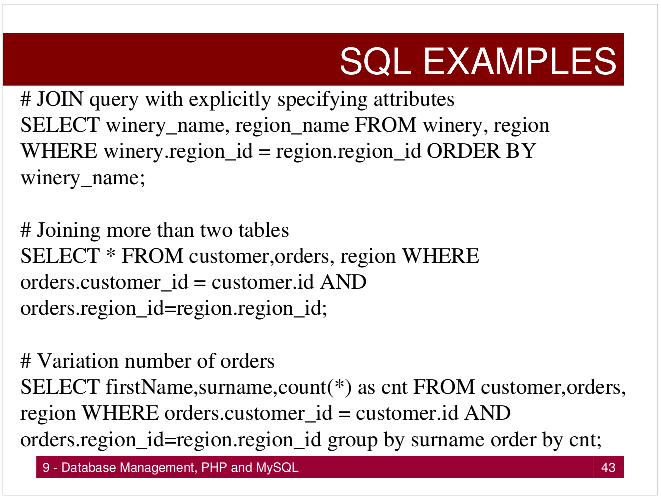
```
SELECT surname, firstname FROM customer;
SELECT * FROM region ;
SELECT curtime();
SELECT pi()*(4*4);

SELECT * FROM region WHERE region_id <= 3;
SELECT region_name FROM region WHERE region_id <= 3;

SELECT id FROM customer WHERE (surname='Marzalla' AND firstname
LIKE 'M%' ) OR birth_date='1980-07-14';

SELECT * FROM customer WHERE birth_date > '1989-01-01';
SELECT * FROM customer WHERE birth_date < '1989-01-01';
```

## SQL EXAMPLES

```
SELECT surname, firstname FROM customer WHERE city =
'Portsea' and firstname = 'James' ORDER by surname DESC;

SELECT city, COUNT(*) AS cnt FROM customer GROUP BY
city;

SELECT city, count(*) as cnt from customer GROUP BY city
HAVING cnt > 2

SELECT city, MAX(birth_date) FROM customer GROUP BY
city;

SELECT city FROM customer GROUP BY city;    equivalent to
SELECT DISTINCT city from customer ;
```

## SQL EXAMPLES

```
# Querying details without JOIN - would have to be stored in php
arrays and then be further processed. Tables can be
# matched up using JOINS - see next examples.

SELECT surname FROM customer WHERE id=2;
SELECT * FROM region WHERE id=5;

# Natural Join via identical elements
SELECT * FROM winery NATURAL JOIN region ORDER BY
winery_name;
```



## SQL EXAMPLES

```
# JOIN query with explicitly specifying attributes
SELECT winery_name, region_name FROM winery, region
WHERE winery.region_id = region.region_id ORDER BY
winery_name;

# Joining more than two tables
SELECT * FROM customer,orders, region WHERE
orders.customer_id = customer.id AND
orders.region_id=region.region_id;

# Variation number of orders
SELECT firstName,surname,count(*) as cnt FROM customer,orders,
region WHERE orders.customer_id = customer.id AND
orders.region_id=region.region_id group by surname order by cnt;
```