

Università degli Studi di Modena e Reggio Emilia  
Facoltà di Scienze della Comunicazione e dell'Economia  
Corso di Laurea in Comunicazione e Marketing

Anno Accademico 2004/05

# Metodi per la Gestione dei Dati (lezioni di laboratorio)

Titolare del corso: ing. Stefano SETTI

Lezioni di laboratorio (gruppo A-K): dott. Fabio RUINI

# Programma delle lezioni

- 12/05/05: Download ed installazione di Mysql, attraverso il pacchetto software open source EasyPhp;
- 19/05/05: Creazione Data Base, creazione tabelle e popolamento DB;
- 25/05/05: DML (data manipulation language) – principali istruzioni SQL;
- 26/05/05: ... si continua con il DML;
- 01/06/05: Ricevimento;
- 08/06/05: Supporto ai progetti;
- 09/06/05: Supporto ai progetti.

# Connessione a MySQL – step 1

- Una volta entrati nel prompt dei comandi, occorre spostarsi all'interno della directory nella quale è stato installato il client.
- Se durante l'installazione non sono stati variati i parametri di default, per accedere alla cartella dovrebbe essere sufficiente digitare l'istruzione:

```
C:\>cd \Programmi\EasyPHP\mysql\bin
```

seguita dalla pressione del tasto INVIO

# Connessione a MySQL – step 2

- Per comodità possiamo accedere a MySQL con le credenziali di “root”:

```
C:\Programmi\EasyPHP\mysql\bin\>mysql -u root
```

(il parametro “-u” indica a MySQL che la stringa seguente rappresenta il nome dell’utente che sta tentando di collegarsi al DBMS).

# L'esercizio della settimana scorsa

Nelle slides della scorsa lezione abbiamo visto un esercizio, dove veniva chiesto di creare alcune tabelle, elencate qui sotto:

- *studente = {id, id\_cdl, nome, cognome, data\_di\_nascita, crediti\_conseguiti}*
- *cdl = {id, id\_facolta, nome, classe\_di\_laurea, numero\_chiuso};*
- *facolta = {id, nome, indirizzo, citta};*
- *insegnamento = {id, id\_cdl, nome, crediti, anno};*
- *appello = {id, id\_ins, id\_stu, data};*
- *propedeuticita = {id, id\_ins, id\_ins\_prop}; //qui abbiamo una relazione ricorsiva*

# La tabella 'studente'

```
mysql> CREATE TABLE `studente` (  
  `id` smallint(5) unsigned NOT NULL auto_increment,  
  `id_cdl` smallint(5) NOT NULL default '0',  
  `nome` varchar(30) NOT NULL default "",  
  `cognome` varchar(30) NOT NULL default "",  
  `data_di_nascita` date NOT NULL default '0000-00-  
    00',  
  `crediti_conseguiti` smallint(6) default NULL,  
  PRIMARY KEY (`id`)  
);
```

# La tabella 'cdl'

```
mysql> CREATE TABLE `cdl` (  
  `id` smallint(5) NOT NULL auto_increment,  
  `id_facolta` smallint(5) NOT NULL default  
    '0',  
  `nome` varchar(60) NOT NULL default "",  
  `classe_di_laurea` smallint(3) default  
    NULL,  
  `numero_chiuso` smallint(1) default NULL,  
  PRIMARY KEY (`id`)  
);
```

# La tabella 'facolta'

```
mysql> CREATE TABLE `facolta` (  
  `id` smallint(5) NOT NULL  
    auto_increment,  
  `nome` varchar(50) NOT NULL default "",  
  `indirizzo` varchar(50) default NULL,  
  `citta` varchar(25) default NULL,  
  PRIMARY KEY (`id`)  
);
```

# La tabella 'insegnamento'

```
mysql> CREATE TABLE `insegnamento` (  
  `id` smallint(5) NOT NULL auto_increment,  
  `id_cdl` smallint(5) NOT NULL default '0',  
  `nome` varchar(50) NOT NULL default "",  
  `crediti` smallint(2) NOT NULL default '0',  
  `anno` smallint(1) default NULL,  
  PRIMARY KEY (`id`)  
);
```

# La tabella 'appello'

```
mysql> CREATE TABLE `appello` (  
  `id` smallint(5) NOT NULL auto_increment,  
  `id_ins` smallint(5) NOT NULL default '0',  
  `id_stu` smallint(5) NOT NULL default '0',  
  `data` date default NULL,  
  PRIMARY KEY (`id`)  
);
```

# La tabella 'propedeuticita'

```
mysql> CREATE TABLE `propedeuticita` (  
  `id` smallint(5) NOT NULL auto_increment,  
  `id_ins` smallint(5) NOT NULL default '0',  
  `id_ins_prop` smallint(5) NOT NULL default '0',  
  PRIMARY KEY (`id`)  
);
```

# Le stesse tabelle create con phpMyAdmin (versione 2.2.x)

Per creare le tabelle senza ricorrere alla shell, ma sfruttando l'interfaccia grafica phpMyAdmin, occorre seguire il seguente procedimento:

- alla voce “Crea un nuovo database”, digitare “scuola” e fare clic sul pulsante “Crea”;
- alla voce “Crea una nuova tabella nel database scuola”, inserire:
  - il nome della tabella da creare nella casella “Nome”;
  - il numero di campi di cui è composta nella casella “Campi”;
- fare clic sul pulsante “Esegui”;
- nella nuova schermata, compilare le varie caselle di testo e, una volta riempite tutte, fare clic sul pulsante “Salva”.

# Download del file SQL completo

Per velocizzare l'operazione di creazione delle tabelle dell'esempio, invece che crearle manualmente è possibile scaricare il file 'scuola.sql' disponibile on line (<http://www.webalice.it/fabio.ruini/MGD/scuola.sql>) ed eseguirlo:

- da shell:
  - `mysql>CREATE scuola;`
  - `mysql>USE scuola;`
  - `mysql>SOURCE nome_file_con_percorso;`
- da phpMyAdmin (versione 2.2.x):
  - creare un nuovo database (chiamato “scuola”);
  - eseguire la query sul database “scuola” (selezione del file SQL con il pulsante “Sfoggia” e quindi clic sul pulsante “Esegui”)

# Interrogazione della Base di Dati

## - sintassi [RIF11]

SELECT

Seleziona le  
colonne  
(attributi)

```
[ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr, ...
  [INTO outfile 'file_name' export_options
   | INTO DUMPFILE 'file_name']
[FROM table_references
  [WHERE where_definition]
  [GROUP BY {col_name | expr | position}
            [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_definition]
  [ORDER BY {col_name | expr | position}
            [ASC | DESC], ... ]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
  [PROCEDURE procedure_name(argument_list)]
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

Seleziona le  
tabelle

(relazioni) Condizioni

# Seguire la sintassi

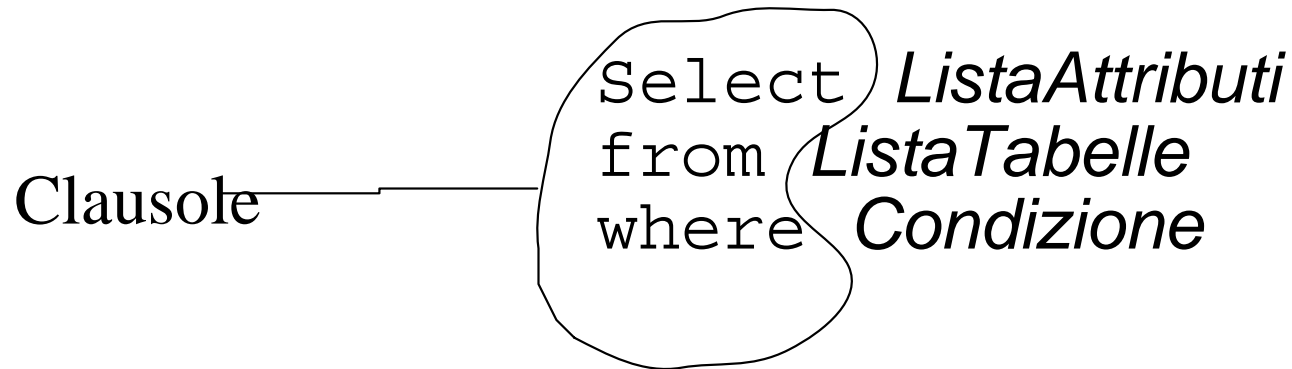
La sintassi dell'istruzione SELECT, per quanto possa sembrare articolata, è importante per ricordare l'esatta sequenza degli operatori di selezione, condizione, ... ad esempio il comando HAVING dovrà essere utilizzato dopo la clausola GROUP BY e prima di ORDER BY.

MA NON SPAVENTATEVI...

Possiamo vedere l'istruzione SELECT secondo la sua struttura essenziale

```
Select ListaAttributi  
from ListaTabelle  
where Condizione
```

# Interrogazioni semplici



L'interrogazione SQL seleziona, tra le righe che appartengono al prodotto cartesiano delle tabelle elencate nella clausola *from*, quelle che soddisfano le condizioni presenti nella clausola *where*. Il risultato di un'interrogazione SQL è così una tabella con una riga per ogni riga che soddisfa la clausola *where* e le cui colonne sono quelle contenute nella clausola *select*.

# La clausola SELECT

La clausola select specifica gli elementi dello schema della tabella risultato. Come argomento può comparire anche il carattere speciale \*(asterisco) che identifica la selezione di tutti gli attributi delle tabelle indicati nella clausola *from*

```
mysql> Select * from studente
```

Selezione tutti gli attributi dalla tabella studente (tutte le righe)

# La clausola FROM

La clausola FROM si utilizza per specificare quali sono le tabelle coinvolte nell'interrogazione e queste vengono poste come argomento di tale clausola. Vedremo nella lezione successiva cosa implica ciò.

```
mysql> Select * from studente
```

# La clausola WHERE

La clausola WHERE ammette come argomento un'espressione booleana costruita combinando predicati semplici con gli operatori *and*, *or* e *not*

**AND** :: Saranno selezionate le righe per cui **tutti** i predicati sono veri

**OR** :: Saranno selezionate le righe che soddisfano **almeno un** predicato

**NOT** :: Saranno selezionate le righe che **non soddisfano** alcun predicato

# La clausola WHERE

Ciascun predicato viene confrontato attraverso gli operatori semplici:

- = uguale
- <> diverso
- < minore
- > maggiore
- <= minore uguale
- >= maggiore uguale

```
SELECT nome, cognome  
FROM studente  
WHERE crediti > 20  
      AND crediti < 30
```



Mostra il nome ed il cognome di tutti gli studenti con nr. di crediti conseguiti compresi tra 20 e 30

# La clausola WHERE - tipi di dati non numerici

Abbiamo appena visto gli operatori semplici che ci permettono di applicare la clausola WHERE su attributi appartenenti a domini numerici ma se avessimo bisogno di porre condizioni su attributi appartenenti a domini di altro tipo?

Abbiamo due modi per eseguire dei confronti quando abbiamo a che fare con dati non numerici.

Il primo è quello che va a confrontare il valore esatto contenuto del database:

```
select * from studente where nome (= | <>) 'giuseppe'
```

In questo caso abbiamo usato gli operatori = e <>

Ma se noi avessimo bisogno di rintracciare solamente gli studenti che hanno il nome che ha come prima lettera la G?

# L'operatore LIKE

L'operatore LIKE permette di effettuare confronti tra stringhe in cui compaiono i caratteri speciali `_` (underscore) e `%` (percentuale). Il primo carattere speciale rappresenta nel confronto un carattere arbitrario mentre il secondo una stringa di un numero arbitrario (eventualmente anche nullo) di caratteri di qualsiasi tipo. Vediamo con degli esempi:

```
select * from studente where cognome like  
'_ossi'
```

Questa query ci restituirà tutti gli studenti che hanno un cognome formato da 5 caratteri dei quali gli ultimi 4 sono 'ossi'. Ci tornerà quindi cognomi tipo 'Rossi', 'Bossi', 'Gossi', 'Lossi', etc...

```
select * from studente where cognome like  
'%ossi'
```

Questa query ci restituirà tutti gli studenti che hanno un cognome che ha come ultimi 4 caratteri la stringa 'ossi'. Ci tornerà quindi cognomi tipo 'Barbarossi', 'Marcandossi', 'Rossi', 'Risossi', etc...

```
select * from studente where cognome like  
'_o%'
```

Questa query ci restituirà tutti gli studenti che hanno un cognome che ha come vincolo di avere il carattere 'o' in seconda posizione. Ci tornerà quindi 'Bonelli', 'Corelli', 'Bonifati', 'Bonisoli', etc...

# Gestione dei valori NULL – NOT NULL

Come abbiamo visto in precedenza, il valore NULL significa mancanza di informazione.

Per selezionare i termini con valori NULL in MySQL si usa il predicato IS NULL, IS NOT NULL, ecco un esempio:

```
select * from studente where  
crediti_conseguiti IS NULL
```

Questa query ci restituirà tutti gli studenti non hanno ancora fatto degli esami. Al contrario:

```
select * from studente where  
crediti_conseguiti IS NOT NULL
```

restituirà tutti gli studenti che hanno fatto almeno un esame

# AS

All'interno della clausola SELECT può tornare comodo ridenominare gli attributi delle tabelle solo all'interno dell'interrogazione in corso.

AS serve appunto a creare un alias dell'attributo su cui viene usato, ad esempio:

```
Select nome as nome_studente from studente
```

Questa query darà come risultato una relazione che avrà come nome dell'attributo non più nome ma nome\_studente. E' bene ricordare che questo non va a modificare il nome degli attributi della tabella studente all'interno del database ma semplicemente ci permette di visualizzare all'interno dell'output il nome dell'attributo che noi preferiamo. Vedremo con il proseguimento delle lezioni la comodità di questo operatore.

# Gli operatori aggregati sulla clausola SELECT - AVG()

L'operatore AVG ci permette di ottenere la media dei valori contenuti all'interno dell'attributo da noi scelto:

```
SELECT AVG( crediti_conseguiti ) AS 'media crediti'  
FROM `studente`
```

Seleziona la media dei crediti degli studenti e crea su di essa l'alias  
MEDIA STUDENTI

# Gli operatori aggregati sulla clausola SELECT - SUM()

L'operatore SUM ci permette di ottenere la somma dei valori restituiti dalla query SQL

```
SELECT SUM( crediti_conseguiti ) AS 'totale crediti'  
FROM `studente`  
WHERE id_cdl <> 1
```

Seleziona la somma dei crediti degli studenti non appartenenti al corso di laurea 1 e crea l'alias totale crediti

# Gli operatori aggregati sulla clausola SELECT - MAX() e MIN()

Gli operatori MAX e MIN si comportano in maniera analoga, ma fanno sì che l'interrogazione restituisca soltanto il valore massimo o minimo dell'attributo specificato.

```
SELECT nome, cognome, MAX(crediti_conseguiti)
      AS 'numero crediti'
FROM `studente`
WHERE id_cdl = 1
```

Questa query ci mostra lo studente che ha ottenuto il massimo numero di crediti all'interno del corso di laurea con id 1.

# Gli operatori aggregati sulla clausola select COUNT()

L'operatore COUNT ci permette di ottenere il numero di record che una query SQL ci restituisce:

```
SELECT COUNT(*) AS 'numero di studenti del cdl 1'  
FROM `studente`  
WHERE id_cdl = 1
```

Mostra il numero di studenti appartenenti al corso di laurea #1 e crea un alias che mostra il risultato con il nome dell'attributo “Numero di studenti del cdl 1”

# Ordinamento dei risultati

L'istruzione "ORDER BY" consente di ordinare, in base al valore di uno o più campi, i risultati di una query.

La sua sintassi è la seguente:

```
mysql>SELECT *  
      FROM nome_tabella  
      ORDER BY attributo1 [ASC | DESC],  
              attributo2 [ASC | DESC], ... ;
```

# Esempio di ordinamento

Ad esempio, la query seguente:

```
mysql>SELECT *  
      FROM studente  
      ORDER BY cognome, nome;
```

restituisce le tuple presenti all'interno della tabella “studente”, ordinate alfabeticamente in funzione del cognome e, in caso di due cognomi identici, del nome.

# Esercizi

Utilizzando come riferimento il database “scuola”, effettuare le seguenti interrogazioni in linguaggio SQL:

- elenco degli studenti presenti nel Data Base, con i rispettivi dati anagrafici (nome, cognome, data di nascita);
- numero di facoltà presenti nel DB;
- media del numero di crediti conseguiti dagli studenti;
- numero di crediti conseguiti complessivamente dagli studenti;
- numero di insegnamenti del corso di laurea in “Economia, Reti, Informazione” (*id\_cdl = 1*);
- numero di studenti che di cognome fanno “Bianchi”;
- numero di studenti nati nel 1982.

# Fine della terza lezione...

