

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



UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

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 Manipolation Language) – principali istruzioni SQL;
- **26/05/05: DML – Join ed altre istruzioni SQL;**
- 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 (ed accertato che EasyPHP sia in esecuzione), 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).

Le interrogazioni proposte ieri...

- Al termine della lezione di ieri, sono state proposte alcune interrogazioni da effettuarsi sul Data Base “scuola”.
- Vediamo nel dettaglio quali sono delle possibili query SQL che permettono di ottenere i risultati desiderati.

Query 1

- Elenco degli studenti presenti nel Data Base, con i rispettivi dati anagrafici (nome, cognome, data di nascita):

```
mysql> SELECT nome, cognome, data_di_nascita  
FROM studente;
```

Query 2

- Numero di facoltà presenti nel DB:

```
mysql> SELECT COUNT(*)  
        FROM facolta;
```

Query 3

- Media del numero di crediti conseguiti dagli studenti:

```
mysql> SELECT AVG(crediti_conseguiti)  
        FROM studente;
```

Query 4

- Numero di crediti conseguiti complessivamente dagli studenti:

```
mysql> SELECT SUM(crediti_conseguiti)  
        FROM studente;
```

Query 5

- Numero di insegnamenti del Corso di Laurea in “Economia, Reti, Informazione” (id_cdl = 1)

```
mysql> SELECT COUNT(*)  
        FROM insegnamento  
        WHERE id_cdl = 1;
```

Query 6

- Numero di studenti che di cognome fanno “Bianchi”:

```
mysql> SELECT COUNT(*)  
        FROM studente  
        WHERE cognome = 'Bianchi';
```

Query 7

- Numero di studenti nati nel 1982:

```
mysql> SELECT COUNT(*)  
        FROM studente  
        WHERE data_di_nascita >= '1982-01-01'  
        AND data_di_nascita <= '1982-12-31';
```

- oppure:

```
mysql> SELECT COUNT(*)  
        FROM studente  
        WHERE data_di_nascita LIKE '1982%';
```

Altre due istruzioni

- Ora che abbiamo visto come funzionano gli operatori logici, possiamo introdurre le ultime due istruzioni che ci potranno tornare utili in vista del progetto finale:
- DELETE;
- UPDATE.

L'istruzione DELETE [RIF12]

- L'istruzione DELETE serve per eliminare uno o più record da una tabella. La sua sintassi è la seguente:

```
mysql> DELETE  
        FROM nome_tabella  
        WHERE condizione;
```

- Dalla tabella “nome_tabella” verranno quindi eliminati tutti i record che corrispondono alla “condizione”.

L'istruzione DELETE - esempio

- Con riferimento al solito Data Base “scuola” possiamo inserire un nuovo record nella tabella “studente”:

```
mysql> INSERT INTO studente  
      (id_cdl,nome,cognome,data_di_nascita,crediti_conseguiti)  
      VALUES (2,'Alberto','Neri','1981-02-25',32);
```

- e successivamente procedere alla sua eliminazione:

```
mysql> DELETE FROM studente  
      WHERE id_cdl = 2 AND nome = 'Alberto' AND cognome = 'Neri' AND  
      data_di_nascita = '1981-02-25' AND crediti_conseguiti = 32;
```

L'istruzione DELETE - esempio

- L'interrogazione precedente, con quella clausola WHERE piuttosto complessa, può essere semplificata utilizzando un'altra query.
- Quando abbiamo creato la tabella "studente", abbiamo impostato il campo "id" come chiave primaria, con proprietà auto increment.

L'istruzione DELETE - esempio

- Possiamo quindi individuare l'id del record appena inserito, tramite l'interrogazione:

```
mysql> SELECT MAX(id) from studente;
```

- ed utilizzare tale valore all'interno della clausola WHERE dell'istruzione DELETE:

```
mysql> DELETE FROM studente  
        WHERE id = valore_restituito;
```

L'istruzione UPDATE [RIF13]

- L'istruzione UPDATE serve per aggiornare uno o più campi di una tabella. La sua sintassi è la seguente:

```
mysql> UPDATE nome_tabella  
      SET nome_campo_1 = nuovo_valore_1  
      SET nome_campo_2 = nuovo_valore_2  
      ...  
      WHERE condizione;
```

- Tutti i record della tabella “nome_tabella”, che corrispondono alla “condizione”, verranno quindi modificati, impostando un “nuovo_valore” in corrispondenza dei campi specificati

L'istruzione UPDATE - esempio

- Con riferimento al solito Data Base “scuola” possiamo modificare un record contenuto all’interno della tabella “studente”:

```
mysql> UPDATE studente  
      SET nome = 'Serenò'  
      WHERE id = 2;
```

- e poi ancora un altro:

```
mysql> UPDATE studente  
      SET crediti_conseguiti = 0  
      WHERE crediti_conseguiti IS NULL;
```



Relazioni tra due tabelle

- Fino a questo momento abbiamo visto come fare interrogazioni su singole tabelle.
- Ma...
- ... nella pratica, interrogazioni di questo genere sono piuttosto rare.

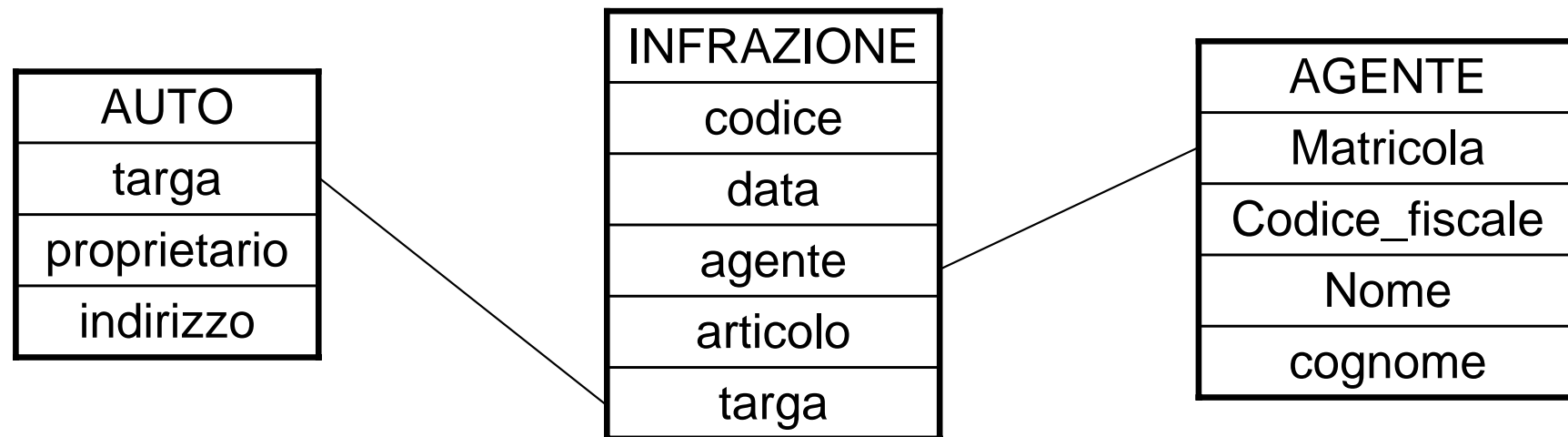
Un esempio di Data Base

- Vediamo quindi un Data Base, che chiameremo “multe”, definito dal seguente schema di base di dati:

R = { AGENTE (matricola,codice_fiscale,nome,cognome),
AUTO (targa,proprietario,indirizzo)
INFRAZIONE (codice,data,agente,articolo, targa) }

Un esempio di Data Base – rappresentazione grafica

- Da un punto di vista “grafico”, il Data Base “multe” può essere rappresentato nel seguente modo:



Un esempio di Data Base – la creazione del DB multe

- Come sempre, possiamo creare il Data Base e le rispettive tabelle, sia attraverso la shell di MySQL, sia attraverso l'interfaccia grafica phpMyAdmin.
- Creiamo un nuovo DB da console e lo selezioniamo:

```
mysql> CREATE DATABASE multe;
```

```
mysql> USE multe;
```

Un esempio di Data Base – la creazione della tabella “agente”

- Creiamo la tabella “agente”:

```
mysql> CREATE TABLE agente (  
    'matricola' char(3) NOT NULL,  
    'codice_fiscale' varchar(16) NOT NULL,  
    'nome' varchar(20) NOT NULL,  
    'cognome' varchar(20) NOT NULL,  
    PRIMARY KEY ('matricola')  
);
```

Un esempio di Data Base – la creazione della tabella “auto”

- Creiamo la tabella “auto”:

```
mysql> CREATE TABLE agente (  
    'targa' varchar(8) NOT NULL,  
    'proprietario' varchar(20) NOT NULL,  
    'indirizzo' varchar(20) NOT NULL,  
    PRIMARY KEY ('targa')  
);
```

Un esempio di Data Base – la creazione della tabella “infrazione”

- Creiamo la tabella “agente”:

```
mysql> CREATE TABLE infrazione (  
    'codice' varchar(6) NOT NULL,  
    'data' date NOT NULL default '0000-00-00',  
    'agente' char(3) NOT NULL,  
    'articolo' char(2) NOT NULL,  
    'targa' varchar(8) NOT NULL,  
    PRIMARY KEY ('codice')  
);
```

Download del file SQL completo

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

■ da shell:

- mysql> CREATE multe;
- mysql> USE multe;
- mysql> SOURCE nome_file_con_percorso;

■ da phpMyAdmin (versione 2.2.x):

- creare un nuovo database (chiamato "multe");
- eseguire la query sul database "multe" (selezione del file SQL con il pulsante "Sfoggia" e quindi clic sul pulsante "Esegui").

JOIN tra tabelle

Utilizzando il Database appena creato proviamo ad esempio ad effettuare la seguente interrogazione alla base di dati:

Visualizza i proprietari delle automobili che hanno commesso un'infrazione in data 26/10/2004

A questo punto non ci è più sufficiente effettuare una select su di una sola tabella, perchè ad esempio la tabella *auto* contiene i nomi dei proprietari degli autoveicoli ma la data delle infrazioni è contenuta nella relazione *infrazione*.

Come possiamo fare?

JOIN

Semplice, basta inserire all'interno della query non più una ma due tabelle, auto ed infrazione

```
Select proprietario from auto [, |inner join] infrazione
```

```
where data='2004-10-26'
```

Questo risultato non è però corretto...
cosa manca?

	proprietario		
<input type="checkbox"/>			Piero Verdi
<input type="checkbox"/>			Piero Verdi
<input type="checkbox"/>			Luca Bini
<input type="checkbox"/>			Gino Luci
<input type="checkbox"/>			Piero Verdi
<input type="checkbox"/>			Piero Verdi
<input type="checkbox"/>			Luca Bini
<input type="checkbox"/>			Gino Luci

In realtà noi non abbiamo messo in relazione le tue tabelle, abbiamo semplicemente creato il rapporto cartesiano tra le relazioni.

L'utilità di una join è mettere in relazione due o più tabelle aventi valori uguali sugli attributi da noi decisi.

Questo vincolo lo andiamo a specificare all'interno della clausola WHERE.

JOIN – L'identificatore .

```
SELECT proprietario FROM auto, infrazione
```

```
WHERE targa=targa
```

Cosa vuol dire targa? a chi si riferisce?
entrambe le relazioni contengono l'attributo targa

```
AND data = 2004-10-26
```

```
SELECT auto.proprietario FROM `auto`, infrazion
```

```
WHERE auto.targa=infrazione.targa
```

PK e FK

```
AND infrazione.data = 2004-10-26
```

L'identificatore '.' serve ad assegnare ogni attributo alla propria relazione. In questo modo il sistema è in grado di riconoscere la differenza tra i due attributi targa, il primo è chiave primaria della relazione auto, mentre il secondo è chiave esterna della relazione infrazione.

A questo punto il sistema ci restituirà la risposta corretta che è Piero Verdi e Luca Bini

Riassumendo

```
SELECT auto.proprietario FROM auto, infrazione
WHERE auto.targa=infrazione.targa
AND infrazione.data = 2004-10-26
```

tabelle coinvolte

↓

Identificatore

Relazione tra chiave primaria
e chiave esterna

Un'ulteriore ottimizzazione

Abbiamo visto nella lezione precedente l'utilizzo del concetto di alias (AS), questo possiamo applicarlo anche alle tabelle:

```
SELECT a.proprietario AS 'Nome e cognome'  
FROM auto [AS] a, infrazione [AS] b  
WHERE a.targa=b.targa  
AND data = 2004-10-26
```

Il sistema nel processare una query SQL andrà prima a recuperare le tabelle coinvolte all'interno della clausola FROM, questo ci permette di poter quindi usare gli alias delle tabelle anche all'interno della clausola SELECT, malgrado questa venga prima e ovviamente anche all'interno della clausola WHERE

NB: nella creazione dell'alias per le tabelle è opzionale mettere AS.

Altri tipi di join

NATURAL JOIN

Questo tipo di join viene utile quando dobbiamo mettere in relazione delle tabelle attraverso un attributo che ha lo stesso nome in entrambe le relazioni. Nell'esempio fatto nelle slides precedenti l'uso del natural join avrebbe dato lo stesso risultato

```
SELECT * FROM auto natural join  
infrazione
```

=

```
SELECT * FROM auto a, infrazione b WHERE a.targa=b.targa
```



I due attributi sono uguali

Attenzione!

Per utilizzare il natural join abbiamo detto che i due attributi che indentificano la PK e la FK delle relazioni coinvolte devono avere lo stesso nome, questo non capita spesso perciò stare molto attenti nell'utilizzare tale costrutto.

Ad esempio nel nostro vecchio esempio relativo al database SCUOLA tutte le relazioni avevano come PK l'attributo ID. Se noi usassimo il NATURAL JOIN per mettere in relazione due o più tabelle sbaglieremmo perchè andremmo a unire le tabelle sui valori simili delle rispettive chiavi primarie quando sappiamo benissimo che la chiave primaria di una relazione deve corrispondere alla chiave esterna a lei riferita dell'altra relazione. Ad esempio se noi volessimo sapere quali sono gli studenti che appartengono al corso di laurea in Comunicazione e Marketing, la sintassi della query esatta sarebbe:

```
SELECT s.nome, s.cognome, c.nome as CDL
      FROM studente s, cdl c
      WHERE s.id_cdl = c.id
            AND c.nome = 'Comunicazione e Marketing'
```

e non

```
SELECT s.nome, s.cognome, c.nome as CDL
      FROM studente s natural join cdl c
      WHERE c.nome = 'Comunicazione e Marketing'
```

Group by

La clausola `group by` permette di raggruppare i record che contengono gli stessi valori per l'insieme di attributi contenuti all'interno della clausola stessa. Ad esempio:

```
SELECT proprietario FROM auto
```

Ipotizziamo di voler conoscere i nomi dei proprietari delle automobili multate, contenuti all'interno della relazione `auto`.

Se usassimo la sintassi

```
SELECT proprietario FROM auto group by proprietario
```

avremmo sì la risposta che cerchiamo, ma il nome `Piero Verdi` comparirebbe due volte. Al fine della nostra interrogazione noi vogliamo soltanto sapere i nomi dei singoli proprietari: non ci interessa che i loro nomi compaiono più volte. La clausola `GROUP BY` ci permette appunto di raggruppare i nomi che contengono la stessa stringa.



Distinct

La parola chiave distinct permette di eliminare i duplicati

```
SELECT distinct (data) FROM infrazione
```

Questa interrogazione restituirà i valori contenuti all'interno della relazione “infrazione” escludendo i duplicati.

Distinct VS. Group By

La differenza tra DISTINCT e GROUP BY sta nel fatto che il primo si limita semplicemente ad eliminare i duplicati mentre il secondo viene utilizzato per creare dei sottogruppi aventi lo stesso valore sul medesimo attributo sui quale poter effettuare operazioni aggregati attraverso gli operatori mostrati nelle lezioni passate, ad esempio se volessimo fare la somma dei crediti di tutti gli studenti raggruppati per cognome faremmo:

```
SELECT nome, cognome, sum(crediti_conseguiti)
      FROM `studente`
      group by cognome
```

Having

La clausola HAVING descrive le condizioni che si devono applicare al termine dell'esecuzione di una interrogazione che fa uso della clausola GROUP BY. Se prendiamo l'esempio precedente:

```
SELECT nome, cognome, sum(crediti_conseguiti)
FROM `studente`
      group by cognome
```

e lo modifichiamo dicendo che vogliamo solo i gruppi che hanno una somma di crediti minore di 50 dovremo scrivere:

```
SELECT nome, cognome
FROM `studente`
      group by cognome
      having sum(crediti_conseguiti) < 50
```

Fine della quarta lezione...

