

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 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.

---

Materiale didattico a cura di: Fabio Ruini e Alessandro Filisetti

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

# Riferimenti semi-bibliografici:

- MySQL Reference Manual  
disponibile in vari formati all'indirizzo:  
<http://dev.mysql.com/doc/>
- In particolare:
  - Chapter 3 – “MySQL Tutorial”
  - estratti pubblicati su:  
<http://www.webalice.it/fabio.ruini>  
numerati e richiamati, dove necessario, con la sigla **[RIFxx]**

# Il linguaggio SQL

SQL – Structured Query Language, è molto di più di un linguaggio per scrivere interrogazioni. Contiene al suo interno:

DML – Data Manipulation Language

DDL – Data Definition Language

*Insieme di comandi per la modifica e l'interrogazione dell'istanza di una base di dati*

- Insert
- Select
- Update
- Delete
- ...

*Insieme di comandi per la definizione dello schema di una base di dati relazionale*

- Alter table
- Create
- Drop
- ...

# 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).

# Visualizzazione dell'elenco dei Data Base presenti nel DBMS

- Una volta connessi al DBMS, possiamo ottenere un elenco dei Data Base presenti al suo interno, con il comando:

```
mysql>SHOW DATABASES;
```

L'output che otteniamo è in forma tabellare e ci fornisce un elenco dei DB presenti nel DBMS (o, meglio, di quelli che possiamo vedere con l'utente attuale, perché in possesso dei privilegi necessari [RIF01])

# Selezione di un particolare DB

- Prima di iniziare a lavorare su di uno specifico Data Base, occorre indicare a MySQL di quale DB si tratta. Per farlo si può ricorrere all'istruzione:

```
mysql>USE nome_database;
```

(per una breve descrizione del comando, [RIF02])

# Elenco delle tabelle

- Una volta selezionato il Data Base di interesse, l'istruzione che consente di visualizzare l'elenco completo delle tabelle ivi contenute è:

```
mysql>SHOW TABLES;
```

# Creazione di un nuovo DB

- A seconda del tipo di lavoro da svolgere, si può utilizzare un Data Base già esistente, oppure crearne uno ex novo. L'istruzione per farlo è:

```
mysql>CREATE DATABASE nome_nuovo_database;
```

(per un breve approfondimento delle istruzioni  
“USE” e “CREATE DATABASE”, [RIF02])

# Selezionare il nuovo DB

- Ora che abbiamo creato il nuovo Data Base, per iniziare ad utilizzarlo occorre innanzitutto selezionarlo, secondo la modalità vista in precedenza:

```
mysql>USE nome_nuovo_database;
```

- In alternativa, è possibile “preselezionare” il DB da utilizzare direttamente all’avvio di MySQL, con l’istruzione:

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

# Creazione di una nuova tabella

- Per creare una nuova tabella, si fa riferimento all'istruzione "CREATE TABLE" [RIF03]:

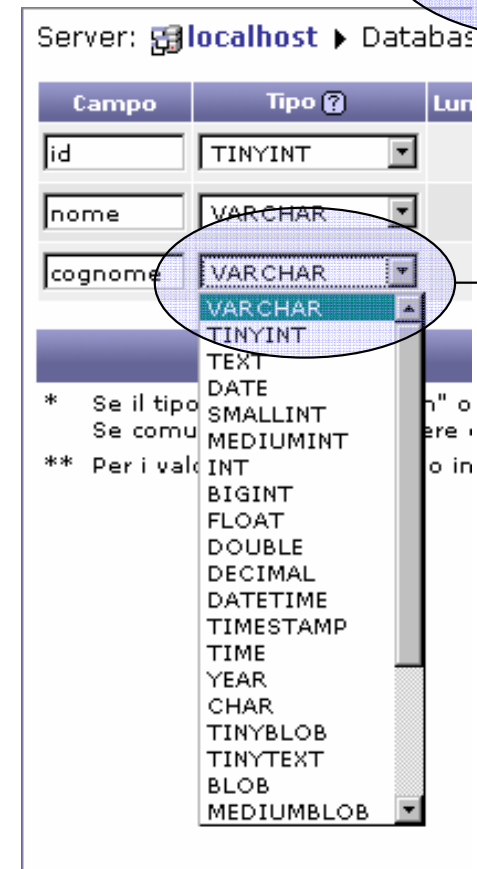
```
mysql>CREATE TABLE nome_tabella (  
    nome_campo_1  
    tipo_campo1  
    NOT NULL  
    AUTO_INCREMENT,  
    nome_campo_2  
    ... ,  
    primary key(nome_campo)  
)
```

# Tipi di dati

Ogni attributo all'interno di una relazione è associato ad un tipo di dato (datatype) o dominio.

I tipi di dati si distinguono in:

- Caratteri
- Tipi numerici esatti
- Tipi numerici approssimati
- Date e ore



# I vari tipi di dati - testo

Un tipo di dato stringa è un campo che può contenere qualunque tipo di carattere: numerico, alfanumerico, simboli ecc.

<b>Nome del tipo</b>	<b>Dimensioni massime</b>	<b>Memoria occupata</b>
CHAR	255 byte	X byte (*)
VARCHAR	255 byte	X+1 byte (*)
TINYTEXT	255 byte	X+1 byte (*)
TINYBLOB	255 byte	X+2 byte (*)
TEXT	65535 byte	X+2 byte (*)
BLOB	65535 byte	X+2 byte (*)
MEDIUMTEXT	1,6 MB	X+3 byte (*)
MEDIUMBLOB	1,6 MB	X+3 byte (*)
LONTEXT	4,2 GB	X+4 byte (*)
LONGBLOB	4,2 GB	X+4 byte (*)

(\*) X è lo spazio occupato dal testo all'interno del campo

# I vari tipi di dati - testo

## CHAR E VARCHAR

Questi due tipi di campi, nonostante la somiglianza nel nome, si comportano in maniera molto diversa. Il primo ha una lunghezza fissa, mentre il secondo è variabile.

Ciò significa che se creassimo un campo CHAR(9) e al suo interno specificassimo "ciao", questo campo occuperebbe comunque 9 byte invece di 4. Con VARCHAR(9) invece, scrivendo al suo interno "ciao" il campo occuperebbe 5 byte (guardare la tabella superiore X+1 dove in questo caso è X=4, quindi 4+1=5).

NB: All'interno di una tabella non è possibile utilizzarli assieme, automaticamente il sistema li renderà tutti e due varchar.

# I vari tipi di dati - testo

## TEXT E BLOB

TEXT e BLOB sono i campi di MySQL dedicati a contenere grandi quantità di dati. Fino a 4,2 GB con i LONGTEXT e LONGBLOB.

Il secondo in particolare, il campo di tipo BLOB sta per Binary Large Object e consente il salvataggio di interi file nel formato binario. Utile per nascondere file dietro username e password, senza così riuscire a rintracciare il percorso fisico del file (che infatti non esiste, essendo incluso direttamente nel database).

# I vari tipi di dati - testo

## I Vincoli

I vincoli previsti da questi tipi di campi sono:

- BINARY**: ammesso dai campi CHAR e VARCHAR: i dati salvati saranno trattati come stringhe binarie.
- DEFAULT**: ammesso da tutti i tipi di campi: imposta un valore predefinito nel caso il campo fosse lasciato vuoto.
- NOT NULL**: ammesso da tutti i tipi di campi: impone che il campo non sia lasciato vuoto.
- NULL**: ammesso da tutti i tipi di campi: se il campo non contiene un valore, sarà vuoto.
- PRIMARY KEY**: ammesso da tutti i campi, ma è consigliabile impostarlo su dati di tipo numerico. Serve a impostare un indice i quali dati non devono essere vuoti.

# I vari tipi di dati – numeri

<b>Nome del tipo</b>	<b>Memoria occupata</b>	<b>Intervallo di valori consentito</b>	<b>Se solo positivi (UNSIGNED)</b>
TINYINT	1 byte	da -128 a +127	da 0 a +255
SMALLINT	2 byte	da -32768 a +32767	da 0 a +65535
MEDIUMINT	3 byte	da -8388608 a +8388607	da 0 a +16777215
INT	4 byte	da -2147483648 a +2147483647	da 0 a +4294967295
BIGINT	8 byte	da -9223372036854775808 a +9223372036854775807	da 0 a +18446744073709550615
FLOAT(I,D)	4 byte	A seconda dei valori	
DOUBLE(I,D)	8 byte	A seconda dei valori	
DECIMAL(I,D)	Il peso di I + 2 Byte	A seconda dei valori	

I e D rappresentano i numeri decimali ammessi

Importante porre l'attenzione sui limiti massimi e minimi quando si sceglie un tipo di dato ma occorre anche tenere conto della memoria occupata.

# I vari tipi di dati – le date

I tipi di campi data sono tutti quei campi che contengono come valore la data e/o l'ora.

<b>Nome del tipo</b>	<b>Formato</b>	<b>Se vuoto (zero)</b>
DATETIME	AAAA-MM-GG hh:mm:ss	0000-00-00 00:00:00
DATE	AAAA-MM-GG	0000-00-00
TIME	hh:mm:ss	00:00:00
YEAR	AAAA	0000
TIMESTAMP(2)	AA	00
TIMESTAMP(4)	AAMM	0000
TIMESTAMP(6)	AAMMGG	000000
TIMESTAMP(8)	AAAAMMGG	00000000
TIMESTAMP(10)	AAMMGGhhmm	0000000000
TIMESTAMP(12)	AAMMGGhhmmss	000000000000
TIMESTAMP(14)	AAAAMMGGhhmmss	00000000000000

A=anno, M=mese, G=giorno, h=ora, m=minuti, s=secondi

# I vari tipi di dati – le date

## **DATETIME**

E' il formato più completo e preciso a nostra disposizione. Varia da 1000-01-01 00:00:00 a 9999-12-31 23:59:59

## **DATE**

Uguale al precedente, ma senza l'ora. Ammette infatti dati a partire da 1000-01-01 al 9999-12-31.

## **TIME**

Salva l'ora. I valori vanno da 00:00:00 a 23:59:59. E' possibile però salvare intervalli di valore tra un evento e un altro e quindi, ammettere ore differenti. In questo caso, i dati vanno da -838:59:59 a 838:59:59.

MySQL legge i valori partendo da destra, quindi, salvando un campo con il contenuto 8:32, nel database verrà interpretato come 00:08:32.

Oltre ai due punti ( : ) MySQL ammette altri segni di interpunzione senza particolari difficoltà. L'immissione di un valore sbagliato sarà salvato come mezzanotte in punto (00:00:00).

# I vari tipi di dati – le date

## **YEAR**

Salva l'anno ed è il campo più leggero: 1 solo byte. Ammette valori dal 1901 al 2155. Gli anni possono essere salvati a due o a quattro cifre. MySQL, in caso di anni a due cifre, interpreterà i valori da 70 a 99 come dal 1970 al 1999. Quelli dall'1 al 69, come dal 2001 al 2069.

Per evitare fraintendimenti quindi, è consigliabile impostare l'anno a quattro cifre.

## **TIMESTAMP**

Questo campo salva il momento esatto in cui la tabella viene modificata. Quindi può essere utile per visualizzare (senza doverlo calcolare ogni volta) il momento dell'ultima modifica del record in cui il campo **TIMESTAMP** appartiene. Ammette anni compresi tra il 1970 e il 2037. Tutti i tipi di **TIMESTAMP** occupano lo stesso spazio: 4 byte. Perché questo?

Nonostante i vari formati? Perché MySQL salva comunque tutti i dati e poi ne visualizza solo quelli richiesti. Ad esempio, con **TIMESTAMP(2)** il database visualizza solo due cifre dell'anno, ma in memoria ha tutti gli altri dati (anno a 4 cifre, mese, giorno, ora, minuti e secondi). Quando infatti modifichiamo il tipo di **TIMESTAMP**, ad esempio con **TIMESTAMP(8)** lui ha tutti i dati in memoria. La stessa cosa avviene quando abbassiamo il valore di **TIMESTAMP**, visualizzando quindi meno dati. MySQL non cancellerà i vari valori, semplicemente li nasconderà.

# I vari tipi di dati – altri tipi

## **ENUM**

Indica a MySQL le varie possibilità previste.

Ad esempio, con: `ENUM('mare','montagna','lago')` Si impone l'utente la scelta di uno di queste tre possibilità. Altri valori, saranno trattati come valori vuoti (NULL), a meno che non sia definito un valore di default.

## **SET**

Questo tipo di dato è uguale a ENUM, con la differenza di poter effettuare una scelta multipla. Il campo ENUM infatti, consente di scegliere un solo valore alla volta.

Per un ulteriore approfondimento dei tipi di dati supportati da MySQL, si veda [**RIF04**]

# Valori nulli

- In qualunque tabella del DB possono esserci campi particolari, per i quali non è obbligatorio che sia sempre inserito un valore.
- Con il termine “valore nullo” si intende “mancanza di dati”. E’ un concetto diverso rispetto allo 0 (zero) dei tipi numerici ed alla stringa vuota (“”) dei tipi stringa.

(per un approfondimento del concetto di valore nullo e sui problemi che possono conseguirne dall’utilizzo, si vedano [RIF05] e [RIF06])

# Il vincolo NULL

NULL assume quindi tre possibili significati:

1. Valore sconosciuto
2. Valore inesistente
3. Mancanza di informazione (rappresenta un OR logico tra le due precedenti)

Se attribuiamo il vincolo `Not null` ad un attributo e non inseriamo il dato corrispondente, il sistema restituirà un messaggio di errore, a meno che non sia stato specificato un valore di default da inserire in caso di mancato inserimento.

# Il vincolo NULL

L'attribuzione del vincolo NOT NULL è obbligatoria nella definizione di un attributo all'interno di una relazione è obbligatoria mentre quella del NULL (accetta valori nulli) è facoltativa.

```
Cognome varchar(20) not null
```

```
Cognome varchar(20)
```

# Proprietà AUTO\_INCREMENT

- La proprietà AUTO\_INCREMENT di un campo può essere utilizzata per attribuire un'identità univoca alle nuove righe (records) inserite nel Data Base.
- NB: attribuire ad un campo la proprietà AUTO\_INCREMENT implica che questo campo dovrà essere impostato come chiave!
- La proprietà AUTO\_INCREMENT renderà leggermente più agevole la successiva fase di inserimento dei dati.

# Chiavi primarie

- Ogni relazione può (deve) avere **una ed una sola CHIAVE PRIMARIA** = insieme (minimo) degli attributi che individuano univocamente una riga della relazione.

In MySQL il vincolo PK può essere definito direttamente su di un singolo attributo, oppure essere definito elencando più attributi che definiscono la chiave primaria. Gli attributo che fanno parte della chiave ovviamente non possono assumere valore NULL.

NB: è buona norma inserire all'interno delle relazioni un attributo ID che non ha un valore semantico particolare se non quello di servire da identificativo univoco del record

# Chiavi esterne (foreign keys)

- Una chiave esterna (*foreign key*) è costituita da uno o più attributi della relazione che rappresentano la chiave primaria di un'altra relazione.

NB: Con MySQL non è possibile definire la chiave esterna direttamente nella creazione della relazione. Questa andrà a definirsi nel momento in cui creeremo delle join tra relazioni

# Un esempio completo - teoria

- Per chiarire i concetti appena esposti, proviamo a creare, all'interno di un database chiamato "scuola", una tabella di nome "studente" con la seguente "struttura":

id	nome	cognome	data_di_ nascita	crediti_ conseguiti
----	------	---------	---------------------	------------------------

- Secondo il modello relazionale, la formalizzazione di questo schema di relazione è:

*studente = {id,nome,cognome,data\_di\_nascita,crediti\_conseguiti}*

# Un esempio completo - pratica

- Un esempio di codice SQL che implementa lo schema visto nella slide precedente potrebbe essere:

```
mysql>CREATE DATABASE scuola;
mysql>CREATE TABLE studente (
    id int not null auto_increment,
    nome text not null,
    cognome text not null,
    data_di_nascita date not null,
    crediti_conseguiti int,
    primary key(id)
);
```

# Ottimizzare... sempre!

- Il codice SQL che abbiamo appena visto è funzionante, ma vi si potrebbero apportare innumerevoli modifiche.
- In particolare, per i vari campi, si sarebbe potuta effettuare una scelta più oculata dei tipi, in modo di risparmiare memoria allocata!

# Popolamento di un Data Base

- Ora che abbiamo creato un nuovo Data Base e vi abbiamo inserito al suo interno una tabella, possiamo passare alla fase di popolamento.
- Per inserire nuovi records (o tuple, in accordo al modello relazionale) si utilizza l'istruzione:

```
mysql>INSERT INTO nome_tabella (campo1,campo2,...)  
VALUES(valore1,valore2,...);
```

(per ulteriori informazioni sull'istruzione "INSERT", si veda  
[RIF07])

# Uno studente dentro al Data Base

- A titolo esemplificativo, vediamo come inserire un record all'interno della tabella creata in precedenza.
- Una sintassi per farlo può essere:

```
mysql>INSERT INTO studente  
      (id,nome,cognome,data_di_nascita,crediti_conseguiti)  
      VALUES(1,'Mario','Bianchi','1982-08-28',16);
```

# Il vantaggio di un campo AUTO\_INCREMENT

- Ancora una volta, seppur la sintassi utilizzata porti al risultato atteso (inserimento di un record all'interno del Data Base), è possibile un miglioramento.
- Dimenticavamo la proprietà AUTO\_INCREMENT?

```
mysql>INSERT INTO studente  
      (nome,cognome,data_di_nascita,crediti_conseguiti)  
      VALUES('Mario','Bianchi','1982-08-28',16);
```

# Drop [RIF08] e [RIF09]

- Ora che abbiamo visto come creare Data Base e tabelle, non ci resta che vedere come fare per eliminarli.
- Per rimuovere una tabella da un Data Base, si usa l'istruzione:

```
mysql>DROP TABLE nome_tabella;
```

- Per eliminare interamente un database, si utilizza invece l'istruzione:

```
mysql>DROP DATABASE nome_database;
```

# Eseguire l'SQL contenuto in un file

- Spesso può rivelarsi comodo utilizzare un editor esterno per scrivere il codice SQL.
- La shell di MySQL dispone di una funzione per eseguire i comandi contenuti all'interno di un file. La sua sintassi è:

```
mysql>SOURCE nome_file;
```

- NB: deve essere specificato anche il percorso (es: mysql>SOURCE c:\Documenti\miofile.sql).

Per ulteriori informazioni sull'esecuzione di codice contenuto in files di testo si veda [RIF10].

# Le stesse operazioni in phpMyAdmin

- Tutte le operazioni che abbiamo visto in questa lezione (creazione di Data Base e tabelle, popolamento, drop) possono essere comodamente effettuate attraverso l'interfaccia grafica di phpMyAdmin.
- Una peculiarità di questo strumento è che, in seguito ad ogni operazione svolta, mostra all'utente il codice SQL che ha generato automaticamente.

# Esportare l'SQL da phpMyAdmin ad un file di testo

- phpMyAdmin dispone inoltre di un'efficace funzione di esportazione dei dati.
- Per usufruirne, dall'interfaccia principale del programma, effettuare le seguenti operazioni:
  - scegliere il database da esportare;
  - fare clic su "esporta";
  - alla voce "struttura" selezionare: "Aggiungi DROP TABLE", "Aggiunge IF NOT EXISTS", "Aggiungi valore AUTO\_INCREMENT", "Usa i backquotes con i nomi delle tabelle e dei campi";
  - alla voce "dati" selezionare: "Inserimenti completi", "Inserimenti estesi", "Usa l'esadecimale per i dati binari";
  - selezionare "Salva con nome" e scegliere il nome del file in cui esportare tutto il contenuto del Data Base;
  - fare clic sul pulsante "Esegui" per avviare l'esportazione.








































# Salvare il proprio lavoro

- Floppy;
- Chiave USB;
- e-mail (soluzione migliore!)

# Esercizio 1

Proviamo adesso a modificare la tabella studente vista in precedenza in modo da ottimizzarla.

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra	Azione					
<input type="checkbox"/>	id	smallint(5)		UNSIGNED	No		auto_increment						
<input type="checkbox"/>	nome	text	latin1_swedish_ci		No								
<input type="checkbox"/>	cognome	text	latin1_swedish_ci		No								
<input type="checkbox"/>	data_di_nascita	date			No	0000-00-00							
<input type="checkbox"/>	crediti_conseguiti	int(11)			Si	NULL							

 [Seleziona tutti](#) / [Deseleziona tutti](#)    *Se selezionati:*                        

# Esercizio 1

Proviamo adesso a modificare la tabella studente vista in precedenza in modo da ottimizzarla.

Il risultato potrebbe essere questo:

Ci bastano numeri fino a 65535

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra	Azione					
<input type="checkbox"/>	id	smallint(5)		UNSIGNED	No		auto_increment						
<input type="checkbox"/>	nome	varchar(30)	latin1_swedish_ci		No								
<input type="checkbox"/>	cognome	varchar(30)	latin1_swedish_ci		No								
<input type="checkbox"/>	data_di_nascita	date			No	0000-00-00							
<input type="checkbox"/>	crediti_conseguiti	smallint(6)			Si	NULL							

Seleziona tutti / Deseleziona tutti    Se selezionati:

30 caratteri ci bastano

Usare un tinyint era poco (max 255) ma un int era troppo

# Esercizio 2

Dopo aver creato la tabella studente si potrebbe pensare di creare anche altre tabelle relative ad una segreteria didattica.

Le tabelle potrebbero essere:

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

# Esercizio 3 – da fare a casa

Possiamo provare a svolgere l'esercizio presente su Dolly come esempio. Riportiamo qui sotto il testo, riguardante un'officina.

Nell'esempio viene chiesto di preparare lo schema ERD. Dopo aver creato tale schema, si potrebbe procedere alla creazione del relativo database.

Le relazioni coinvolte nell'esempio:

- Le **officine**, con Nome, indirizzo, telefono.
- Le **automobili**, con targa, modello e proprietario.
- I **clienti** (proprietari di automobili) con codice fiscale, cognome, nome, telefono. Ogni cliente può essere proprietario di più automobili.
- Gli **interventi** di manutenzione, ognuno effettuato presso una officina e con un numero progressivo (unico nell'ambito della singola officina), data inizio e fine, pezzi di ricambio utilizzati (con le quantità) e numero di ore di manodopera.
- I **pezzi di ricambio**, con codice, nome e costo unitario.

---

Materiale didattico a cura di: Fabio Ruini e Alessandro Filisetti

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

# Fine della seconda lezione...

