

(R)DB[MS]

Relational DATA BASE Management Systems

© 2005 Giorgio Meini – ITI “Galilei” (Livorno)

un esempio testuale “storico”: il formato CSV (Comma* Separated Values)

file:///etc/passwd (UNIX/linux)

```
root:x:0:0:root:/root:/bin/bash
giorgio:x:500:500:GM:/home/giorgio:/bin/bash
```

file:///etc/shadow (UNIX/linux)

```
root:$1$xHkMhql0$utt9s/y7XsjxksihmSs381:12922:0:99999:7:::
giorgio:$1$1tuFUmA0sSAyAJ7t5qGmlVNMjtTcUlxO:12922:-1:99999:-1:::
```

* il separatore può essere un simbolo diverso (in questo caso “:”)

Cronologia storica

anni '60	data-base gerarchici e reticolari (IBM, ...)
1970	Codd, IBM: algebra relazionale dei dati
1974	Chamberlain & Boyce, IBM: linguaggio di interrogazione dichiarativo SEQUEL
1979	Ellison, Oracle (SDL/RSI): primo database relazionale commerciale basato su SQL
1986-1987	ANSI-ISO: primo standard SQL
1992	SQL-92 (SQL-2)
1995-1996	Widenius ("Monty"), MySQL AB (TCX): prima versione pubblica di MySQL con licenza GPL
1999	SQL:1999: trigger, funzionalità ad oggetti, ...
2003	SQL:2003: sequenze, funzionalità XML, ...

dal presente al futuro

OO-DB

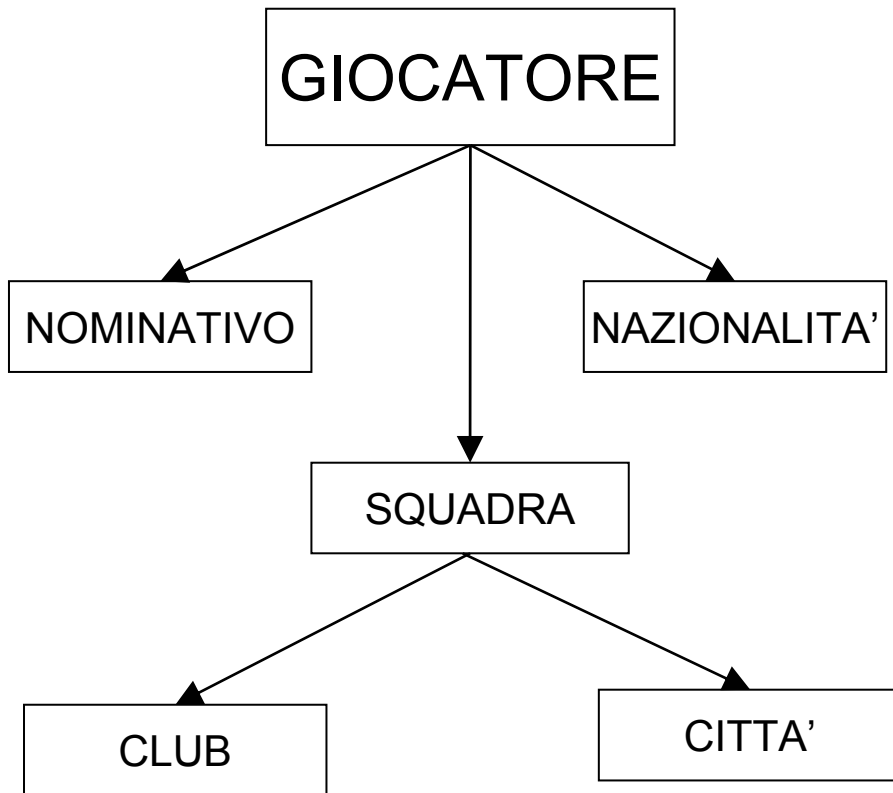
- gli *Object-Oriented* DB consentono rappresentazioni dei dati basate sui concetti di tipo definito dall'utente e di ereditarietà

XML-DB

- i DB XML* gestiscono dati in formato testuale rappresentati secondo uno schema predefinito

* *eXtended Mark-up Language* è un "metalinguaggio" standard, usato per l'implementazione dei *web-services*

un esempio XML minimale



XML

```
<giocatore>  
  <nominativo>  
    Armando Picchi  
  </nominativo>  
  <squadra>  
    <club>Inter</club>  
    <citta'>Milano</citta'>  
  </squadra>  
  <nazionalita'>  
    Italiana  
  </nazionalita'>  
</giocatore>
```

DATABASE developing

1. DB design

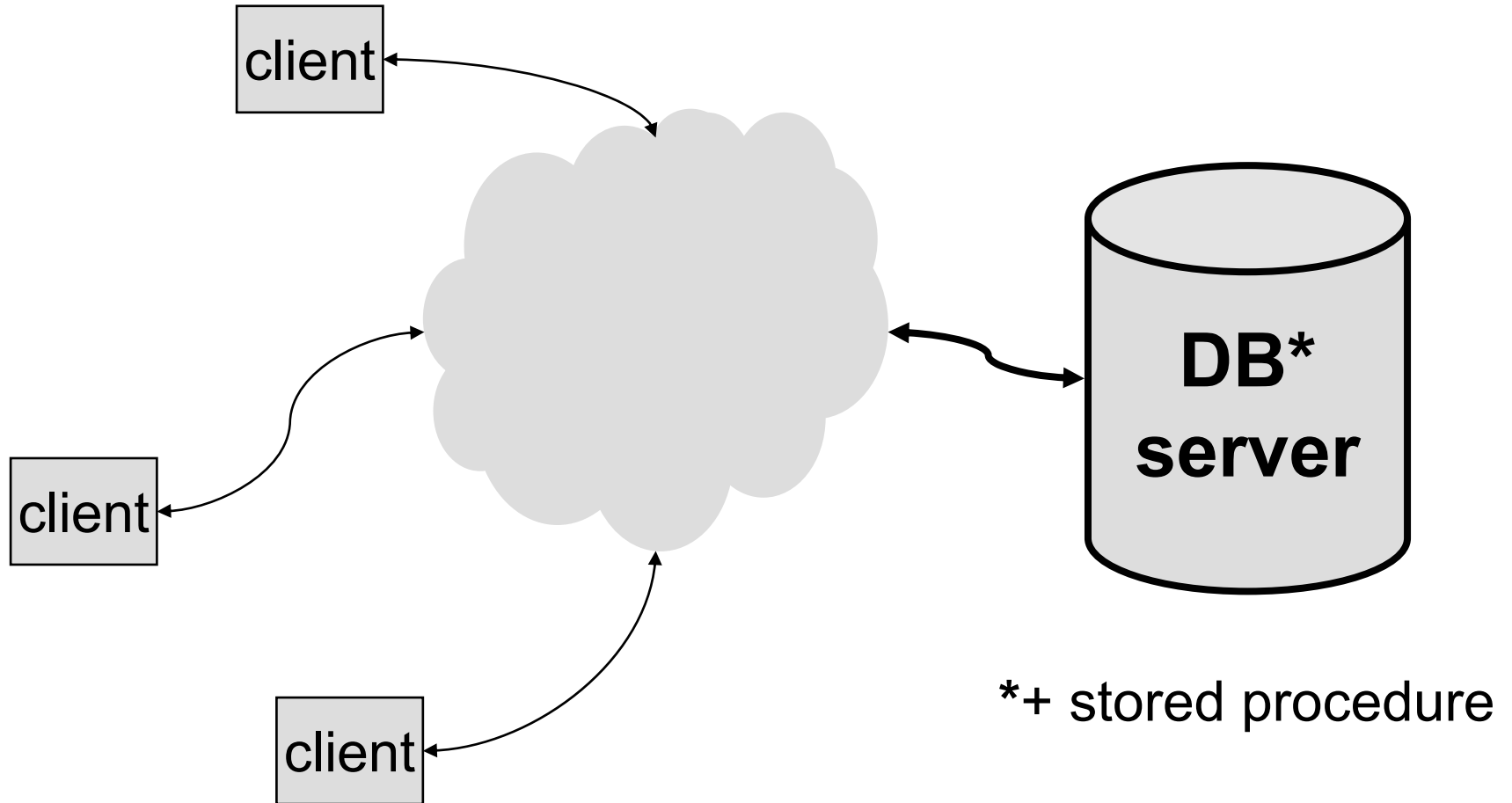
- analisi dei dati (entità, attributi, relazioni, ...)
- sintesi di uno schema di rappresentazione **logica** dei dati (diagramma *Entity-Relationship*)
- definizione di una rappresentazione **fisica** dei dati (tabelle, campi, ...)

2. DB programming

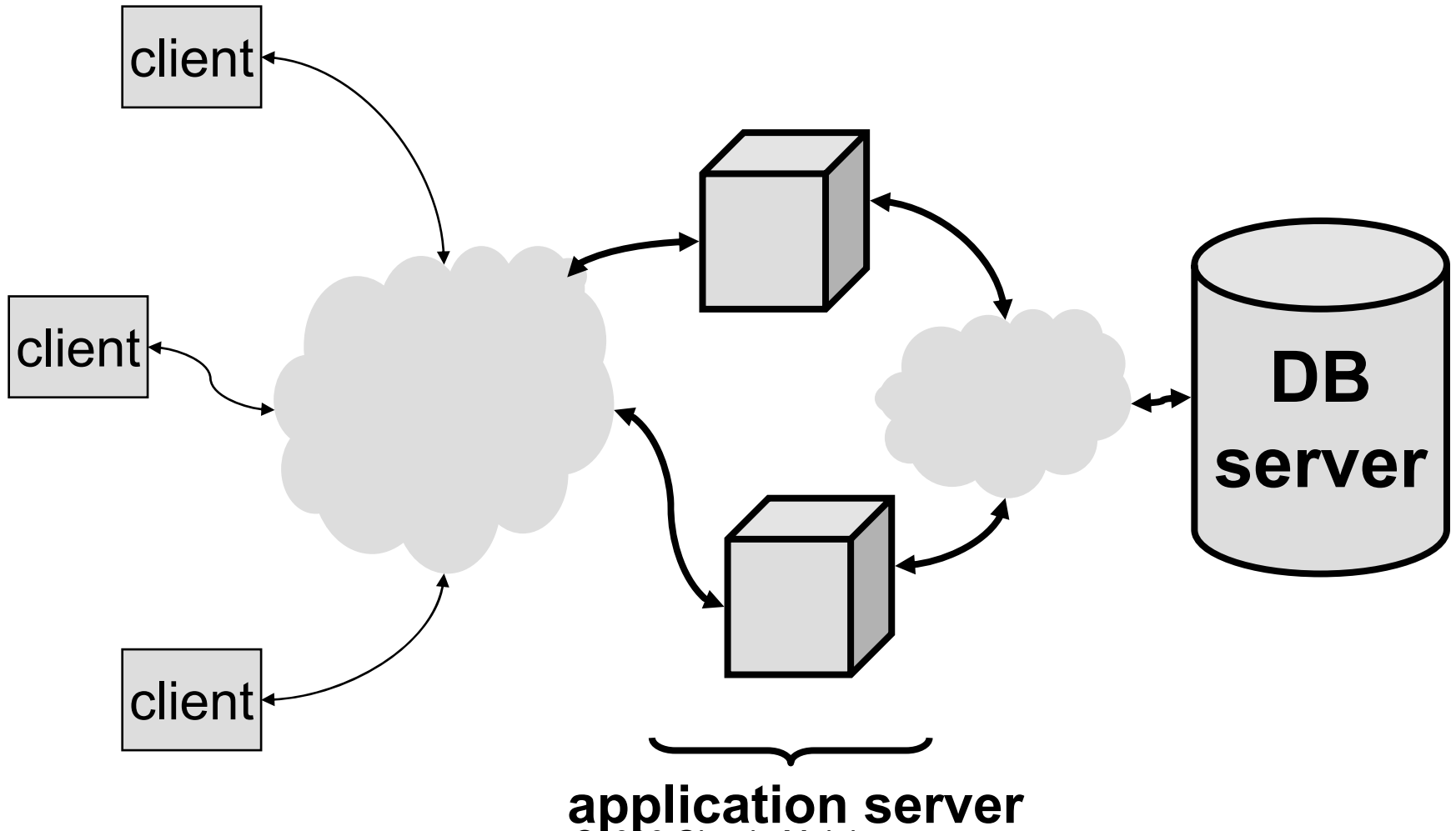
- **inserimento, eliminazione e modifica** dei dati (*data model logic*¹)
- **elaborazione** dei dati (*business control logic*²)
- “**interrogazione**” e “**presentazione**” dei dati (*data view logic*³)

1,2,3: pattern Model-View-Controller

Architettura client/server



Architettura 3-tier



application server

©2006 Giorgio Meini

**Un DB relazionale è
“fisicamente” organizzato
in tabelle costituite da
record (righe) strutturati in
“campi” (colonne):**

CAMPI

chiave primaria

R
E
C
O
R
D

Cognome	Nome	Codice Fiscale	Data di nascita	Luogo di nascita
Duck	Donald
Mouse	Mickey
Scrooge	Uncle
...

- ogni campo della tabella è di un tipo specifico (testo, numero, data, ...) e può essere soggetto a “vincoli”
- un campo della tabella rappresenta la **chiave primaria** il cui valore deve essere distinto per ogni record della tabella (esempio: codice fiscale); eventualmente la chiave primaria può essere rappresentata da un codice numerico progressivo o comprendere più campi
- i dati “impliciti” (esempio: età) non costituiscono campi, ma sono derivati in fase di interrogazione
- i record NON sono ordinati: per velocizzare le operazioni di ricerca (rallentando le operazioni di inserimento ed aggiornamento) è possibile associare ai campi della tabella un **indice**

Microsoft Access DB:

- **tabelle** per la memorizzazione dei dati
- **maschere** per l'interazione con l'utente e la visualizzazione dei dati
- ***query*** per l'interrogazione e la selezione dei dati
- ***report*** per la stampa dei dati
- **pagine** per la visualizzazione via *web* dei dati
- **macro/moduli VBA*** per l'elaborazione dei dati

in *Microsoft Access* ogni oggetto è selezionabile in due modalità distinte:

STRUTTURA

- questa modalità consente la “progettazione” (*design*) dell’oggetto

DATI

- questa modalità permette l’uso dell’oggetto per l’inserimento, l’eliminazione, la modifica e la visualizzazione o selezione dei dati

in *Microsoft Access*:

Tabelle e *query*

- sono di norma progettate nella modalità “visualizzazione struttura”

Maschere, pagine e *report*

- sono di norma generate in modalità “creazione guidata”

Una *query*...

...genera una tabella “virtuale”
contenente esclusivamente i
campi specificati e i *record*
selezionati in base ai criteri
indicati.

Regola fondamentale dei DB

evitare la “ridondanza” dei dati
distribuendo le informazioni in tabelle distinte

CAMPI

R
E
C
O
R
D


Cognome	Nome	Codice Fiscale	Città	CAP
Duck	Donald	...	Paperopoli	CB-098
Mouse	Mickey	...	Topolinia	WD-123
Scrooge	Uncle	...	Paperopoli	CB-098
...

Una “relazione” tra tabelle...

...consente di evitare la ridondanza dei dati che dipendono da altri dati mediante la decomposizione in due tabelle

Tabella "Personaggi"

chiave esterna



Cognome	Nome	Codice Fiscale	Città
Duck	Donald	...	Paperopoli
Mouse	Mickey	...	Topolinia
Scrooge	Uncle	...	Paperopoli
...

Tabella "Città"

→ chiave primaria riferita esternamente

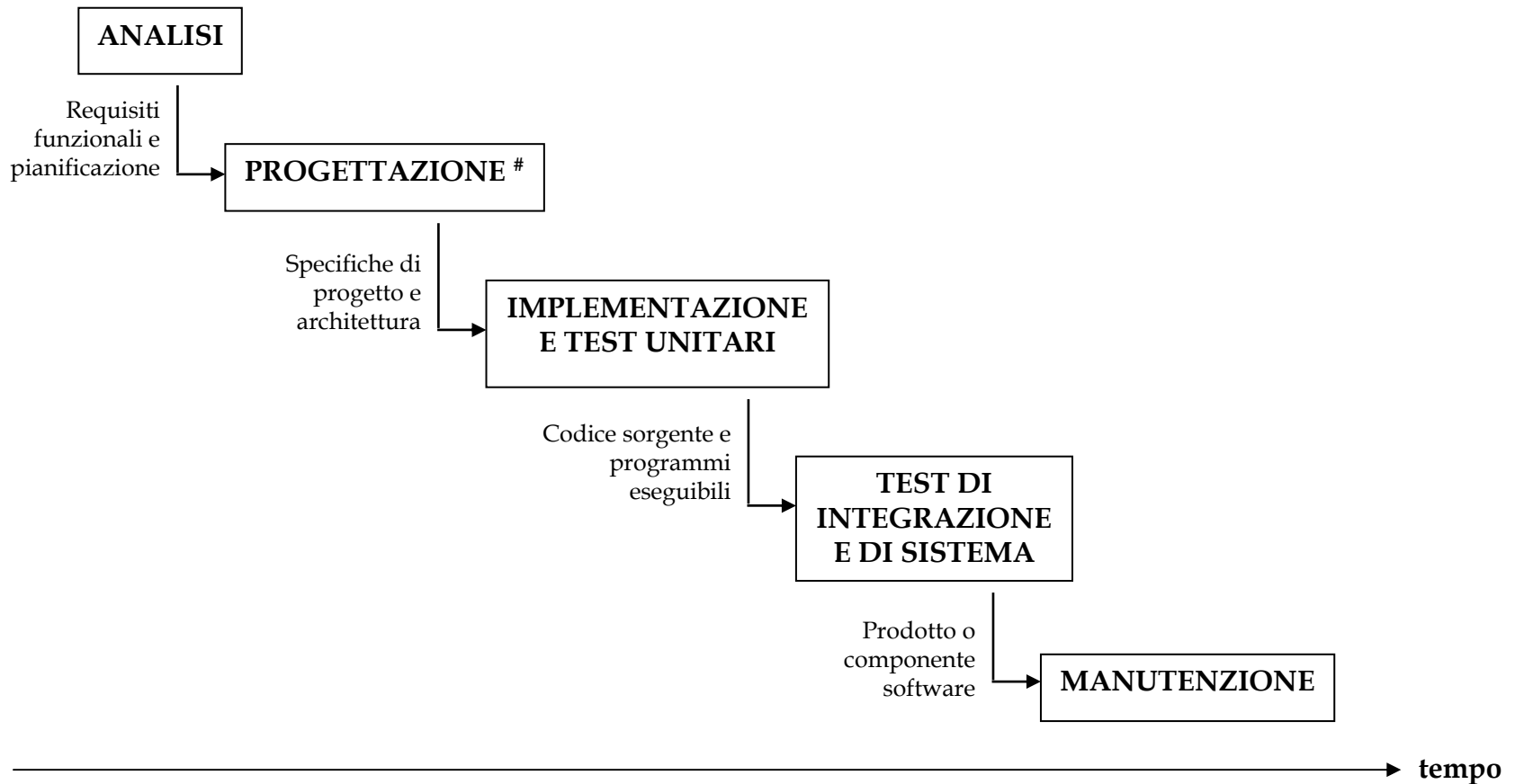
Città	CAP
Paperopoli	CB-098
Topolinia	WD-123
...	...
...	...

Nessuna
ridondanza
dei dati!

un vincolo di “integrità referenziale”:

- impedisce l’inserimento di record i cui valori di campo non riferiscano record presenti nella tabella “relazionata”
- eventualmente, elimina ed aggiorna automaticamente i record dipendenti dai record relazionati modificati e/o eliminati

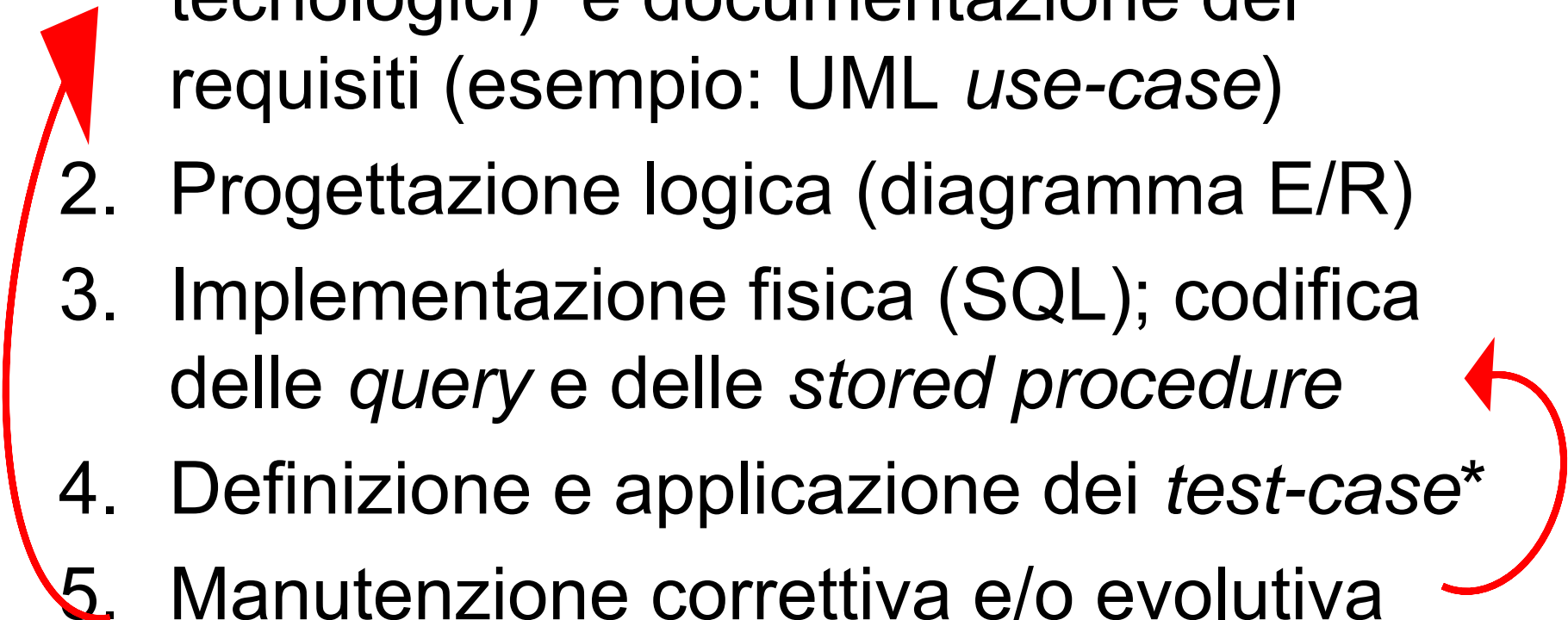
Ingegneria del software: ciclo di vita “a cascata” *



* le moderne metodologie “agili” privilegiano un ciclo di vita iterativo “a spirale”

per il design del software si ricorre al linguaggio grafico UML (Unified Modeling Language)

Il ciclo di vita di un DB relazionale:

1. Analisi (classificazione in funzionali e tecnologici) e documentazione dei requisiti (esempio: UML *use-case*)
 2. Progettazione logica (diagramma E/R)
 3. Implementazione fisica (SQL); codifica delle *query* e delle *stored procedure*
 4. Definizione e applicazione dei *test-case**
 5. Manutenzione correttiva e/o evolutiva
- 

* in molti casi reali: *query* ©2006 Giorgio Meini

Un esempio: la biblioteca

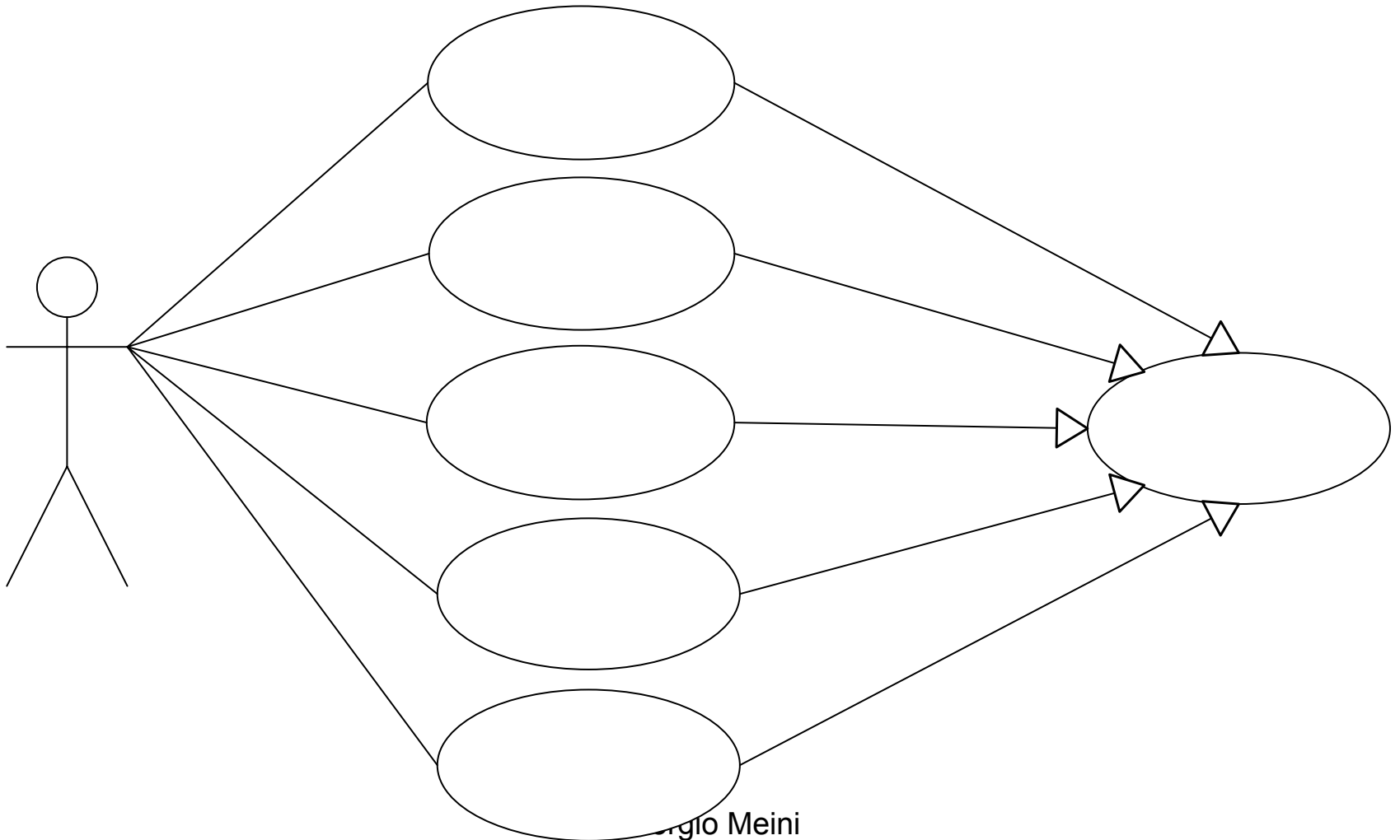
Requisiti funzionali

- documentati con *use-case* per gli **attori** “utente” e “amministratore”

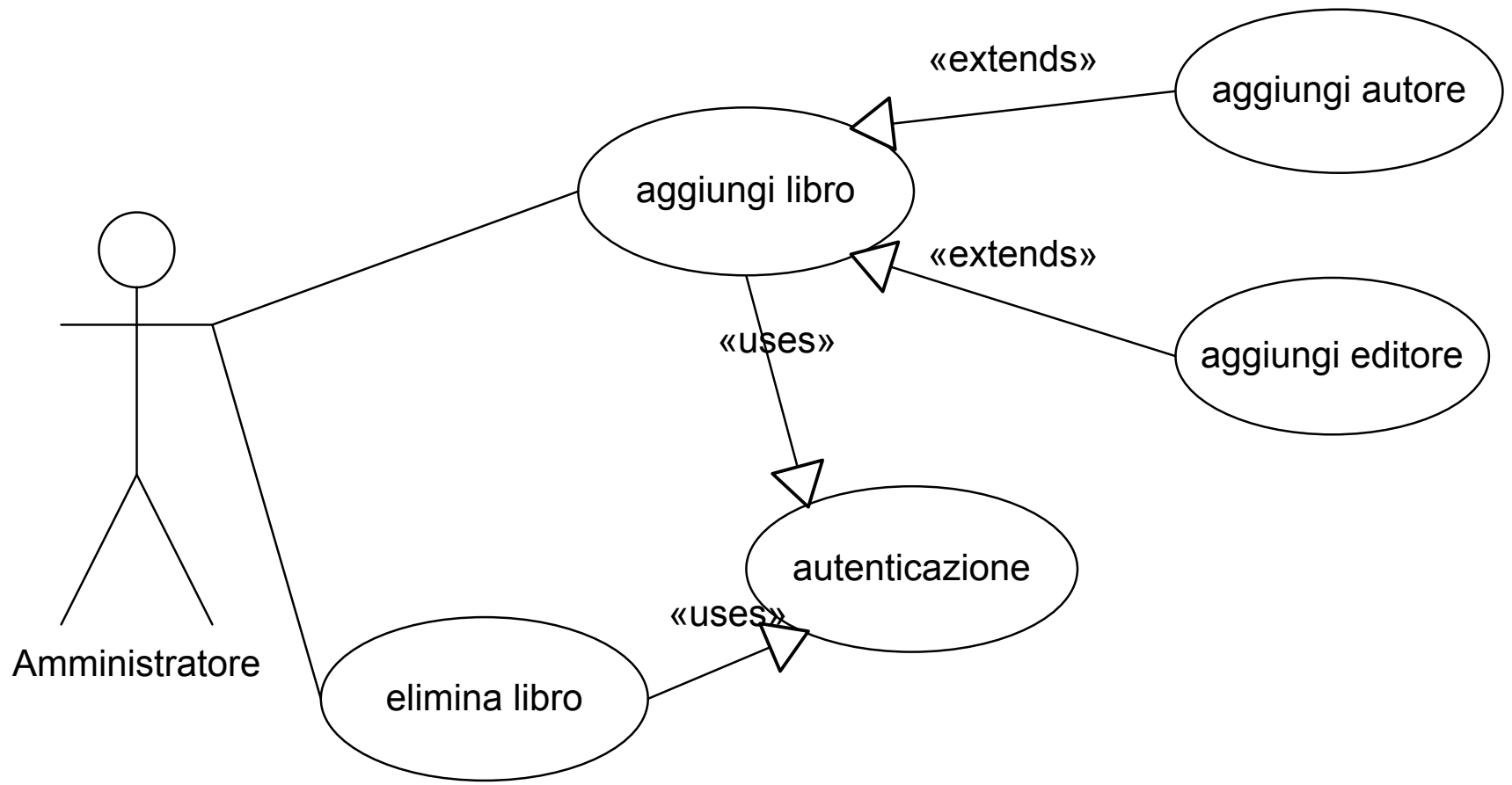
Requisiti tecnologici

- DB relazionale
- architettura *client/server*
- GUI *web-based*

Biblioteca: *use-case* #1



Biblioteca: *use-case* #2



Il diagramma E/R della biblioteca

Entità

(attributi *)

- **LIBRI** (ISBN, Titolo, Anno, ...)
- **EDITORI** (Codice SIAE, Nome, Città, ...)
- **AUTORI** (CF, Cognome, Nome, ...)

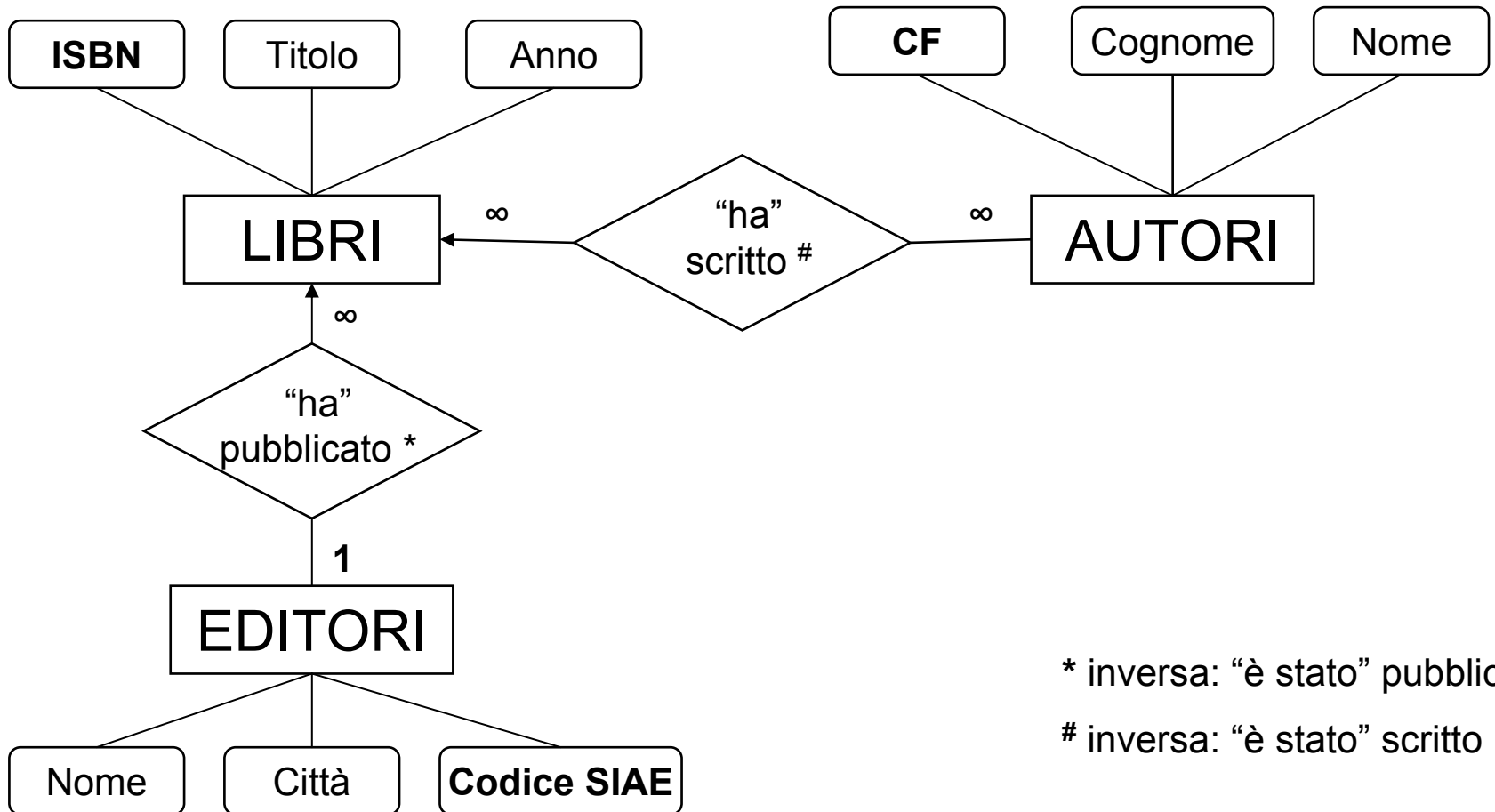
Relazioni

(tra 2 entità)

- “ha” pubblicato
- “è” autore

* informazioni, identificatori, descrittori di relazioni

Il diagramma E/R della biblioteca



- una *super-chiave* è un insieme di attributi di una entità che la identificano univocamente
- una *chiave candidata* per una entità è una *super-chiave* minimale: non esiste nessun suo sottoinsieme proprio di attributi che identificano univocamente l'entità
- una *chiave primaria* di una entità è una specifica chiave candidata (può essere costituita da più attributi)

Le relazioni tra entità *:

uno – a – molti

→ **esempio: un qualsiasi editore “ha pubblicato” molti libri, ma ogni singolo libro “è stato pubblicato” da un solo editore** (a una specifica entità LIBRO corrisponde sempre una specifica entità EDITORE, ma a una specifica entità EDITORE possono corrispondere più entità LIBRO)

molti – a – molti

→ **esempio: un singolo autore “ha scritto” molti libri e un singolo libro “è stato scritto” da più autori** (a una specifica entità LIBRO possono corrispondere più entità AUTORE e a una specifica entità AUTORE possono corrispondere più entità LIBRO)

* una eventuale relazione “uno – a – uno” rappresenta la corrispondenza tra attributi e entità

Dal modello logico alla rappresentazione fisica

Entità	Tabelle
Attributi	Campi (tipi, vincoli, ...)
Relazioni $1/\infty$	2 tabelle distinte aventi un campo comune (chiave esterna e chiave primaria)
Relazioni ∞/∞	NON rappresentabili direttamente: trasformazione in 2 relazioni $1/\infty$

La trasformazione di una relazione ∞/∞ in 2 relazioni $1/\infty$

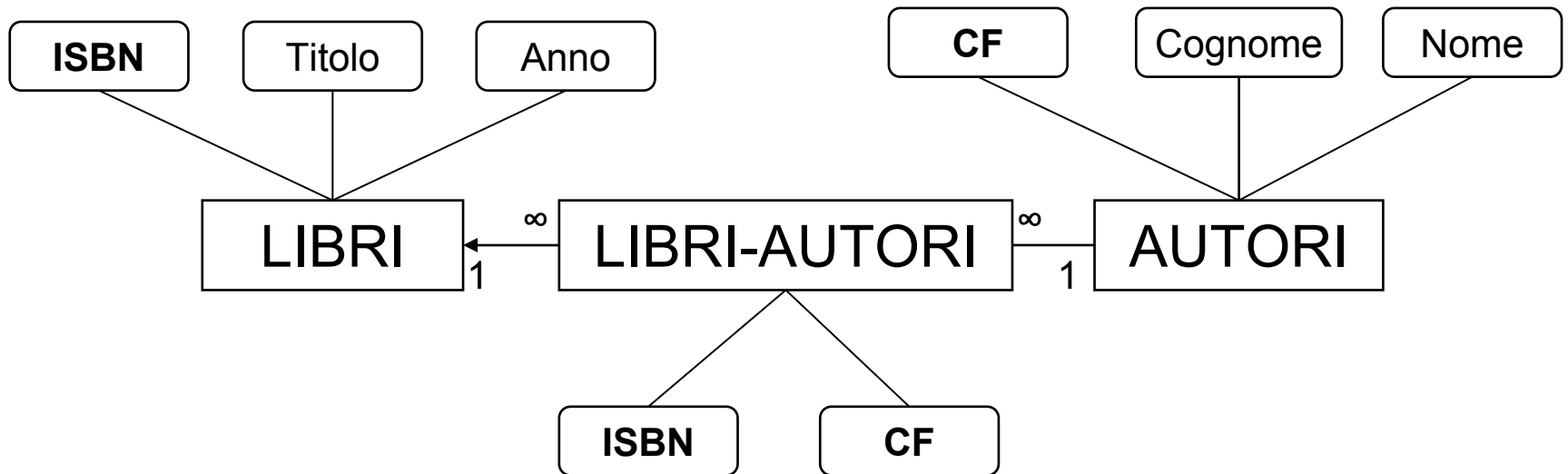


Tabella "EDITORI"

chiave primaria



Nome	Città	Codice SIAE
Mondadori	Milano	0123
Hoepli	Milano	9876
Zanichelli	Bologna	6745

Tabella "AUTORI"

chiave primaria



Cognome	Nome	CF
Neri	Mario	NRRMRR00X99Y123Q
Rossi	Luigi	RSSLGG99Y00X000T
Verdi	Maria	VRDMRR90Z09Q987P
Bianchi	Elena	BNCELN00A99B123K

Tabella "LIBRI"

chiave primaria

chiave esterna

Titolo	Anno	ISBN	Editore (codice SIAE)
RDBMS	1998	88-0000-123-1	0123
SQL	1994	0-12345-000-9	0123
XML	2001	66-9999-987-2	9876
UML	2000	9-09876-999-0	6745

Tabella "LIBRI-AUTORI"

chiave
primaria
(2 campi)



ISBN	CF
88-0000-123-1	NRRMRR00X99Y123Q
88-0000-123-1	BNCELN00A99B123K
0-12345-000-9	RSSLGG99Y00X000T
66-9999-987-2	VRDMRR90Z09Q987P
9-09876-999-0	BNCELN00A99B123K

©2006 Giorgio Meini

Forme normali

La decomposizione* delle tabelle finalizzata alla riduzione e/o eliminazione della ridondanza è stata classificata in **forme normali** che definiscono le proprietà delle tabelle generate dalla decomposizione

- 1^a FN
- 2^a FN
- 3^a FN
- FN di Boyce-Codd
- 4a FN
- ...

* la decomposizione delle tabelle può causare perdita di informazioni e/o di dipendenze
©2006 Giorgio Meini

Definizioni per normalizzazione

- **attributo strettamente identificativo**: attributo che appartiene almeno ad una chiave
- **attributo strettamente informativo**: attributo che non appartiene a nessuna chiave
- **dipendenza funzionale**: relazione tra attributi informativi e attributi identificativi che esplicita la sempre possibile determinazione univoca dei primi a partire dai secondi (esempio: l'editore di un libro è univocamente determinato dal codice ISBN del libro)

1^a forma normale

→ una tabella è in 1^a forma normale se i valori dei campi sono “atomici” (indivisibili)

CONTROESEMPIO: una tabella LIBRI con il campo “Autori” che contenga valori come “Neri, Bianchi” o come “Verdi e Rossi”

2^a forma normale

→ una tabella è in 2^a forma normale se è in 1^a forma normale e se tutti gli attributi strettamente informativi sono attributi dell'entità rappresentata dalla tabella

CONTROESEMPIO: una tabella LIBRI con il campo “Indirizzo” riferito all'editore

3^a forma normale

→ una tabella è in 3^a forma normale se è in 2^a forma normale e se tutti gli attributi strettamente informativi dipendono esclusivamente da attributi identificativi

CONTROESEMPIO: una tabella LIBRI con i campi informativi “Pagine” e “Prezzo” nel caso che Prezzo dipenda esclusivamente da Pagine (secondo un coefficiente stabilito dall’editore)

Il linguaggio* SQL (*sequel*) è composto da 3 sotto-linguaggi:

- **DDL** (Data Definition Language): per creare, modificare e distruggere tabelle
- **DML** (Data Manipulation Language): per inserire, aggiornare o eliminare dati
- **QL** (Query Language): per interrogare il data-base

* il linguaggio SQL è dichiarativo e non procedurale

Comandi DDL e DML fondamentali

DDL

- **CREATE**
(creare)
- **ALTER**
(modificare)
- **DROP**
(distruggere)

DML

- **INSERT**
(inserire)
- **UPDATE**
(aggiornare)
- **DELETE**
(eliminare)

Esempi di comando CREATE

→ CREATE DATABASE *BIBLIOTECA*;

NOT NULL
impedisce
di non
specificare
i valori dei
campi

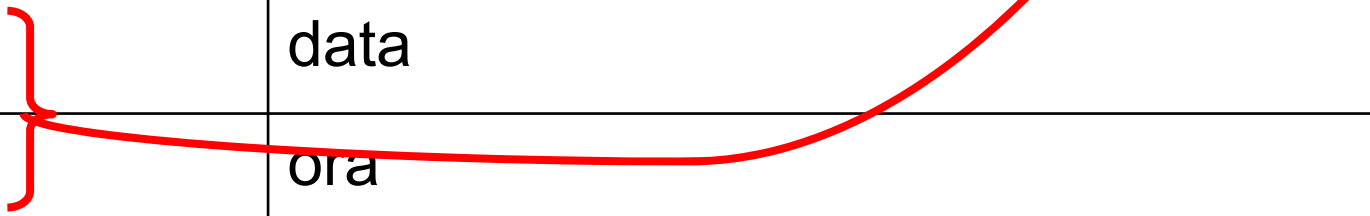
→ CREATE TABLE *AUTORI*
(*Cognome* VARCHAR(32) NOT NULL,
Nome VARCHAR(32) NOT NULL,
CF CHAR(16) NOT NULL PRIMARY KEY)
TYPE = InnoDB; { il tipo predefinito delle tabelle
My-SQL è My-ISAM

il comando **DROP nome-database** elimina una database dal DBMS
il comando **DROP TABLE nome-tabella** elimina una tabella dal database

Tipi di dato (My)SQL

esiste anche il tipo di dato DATETIME

Tipo	Descrizione
INT	valore numerico intero
REAL	valore numerico <i>floating-point</i>
CHAR	stringa di lunghezza fissa
VARCHAR	stringa di lunghezza variabile (limitata)
TEXT	testo di lunghezza variabile (limitata)
DATE	data
TIME	ora
BLOB	dati binari (non modificati)



Comando CREATE TABLE con relazioni e chiavi esterne

→ CREATE TABLE *EDITORI*
(*Nome* VARCHAR(64) NOT NULL,
Città VARCHAR(32),
Codice CHAR(4) NOT NULL PRIMARY KEY)
TYPE = InnoDB;

→ CREATE TABLE *LIBRI*
(*Titolo* VARCHAR(256) NOT NULL,
Anno INT,
ISBN CHAR(13) NOT NULL PRIMARY KEY,
Editore CHAR(4) NOT NULL
REFERENCES *EDITORI*(*Codice*))
TYPE = InnoDB;

la clausola di
“integrità
referenziale”
vincola i valori
del campo
a riferire valori
identici nel
campo
associato
nella tabella
in relazione

Comando CREATE TABLE con chiave primaria estesa su più campi che riferenzia chiavi esterne

→ CREATE TABLE *LIBRI-EDITORI*
(*ISBN* CHAR(13) NOT NULL PRIMARY KEY
REFERENCES *LIBRI*(*ISBN*),
CF CHAR(16) NOT NULL PRIMARY KEY
REFERENCES *AUTORI*(*CF*))
TYPE = InnoDB;

il comando **USE nome-database** seleziona uno dei database del DBMS
il comando **SHOW TABLES** visualizza l'elenco delle tabelle di un database
il comando **DESCRIBE nome-tabella** visualizza la struttura di una tabella

Il comando ALTER TABLE consente di modificare una tabella preesistente

→ ALTER TABLE *LIBRI-EDITORI*
DROP COLUMN *ISBN*;

→ ALTER TABLE *LIBRI-EDITORI*
ADD COLUMN *ISBN* INT;

→ ALTER TABLE *LIBRI-EDITORI*
MODIFY COLUMN *ISBN* CHAR(13) NOT NULL
REFERENCES *LIBRI*(*ISBN*);

→ ALTER TABLE *LIBRI-EDITORI*
DROP PRIMARY KEY;

→ ALTER TABLE *LIBRI-EDITORI*
ADD PRIMARY KEY(*ISBN*, *CF*);

La chiave primaria “perfetta” e i valori predefiniti: un esempio

```
→ CREATE TABLE OGGETTI  
(Codice INT AUTO_INCREMENT,  
Descrizione VARCHAR(64) NOT NULL,  
Data DATE DEFAULT '2001-01-01')  
AUTO_INCREMENT=1;
```

nel caso di non
specificazione
del campo esso
assume il valore
predefinito

primo numero da generare

per ogni singolo record inserito viene generato un nuovo valore successivo al precedente

Esempi di comando INSERT

→ INSERT INTO *AUTORI*

SET *Cognome*='Neri', *Nome*='Mario',
CF='NRRMRRR00X99Y123Q';

→ INSERT INTO *AUTORI* (*Cognome*, *Nome*,
CF) VALUES ('Rossi', 'Luigi',
'RSSLGG99Y00X000T'), ('Bianchi', 'Elena',
'BNCELN00A99B123K');

il comando **INSERT** fallisce se i dati violano le restrizioni delle clausole NOT NULL e REFERENCES, o l'unicità della chiave primaria

Esempio di comando UPDATE

```
→ UPDATE AUTORI  
   SET Cognome='Verdi', Nome='Maria',  
      CF='VRDMRR90Z09Q987P'  
   WHERE CF='NRRMRR00X99Y123Q';
```

identifica uno o più record

il comando **UPDATE** fallisce se i dati violano le restrizioni delle clausole NOT NULL e REFERENCES, o l'unicità della chiave primaria

Esempio di comando DELETE

→ DELETE FROM *EDITORI*
WHERE *Città*='Milano';

identifica uno o più record da eliminare

il comando **DELETE** privo di clausola **WHERE** elimina tutti i record di una tabella

Il Query Language (QL) è composto da un unico comando: **SELECT**


→ **SELECT** [DISTINCT | ALL] *col-1, col-2, ...**
FROM *table-1, table-2, ...**
[WHERE *condition*]
[GROUP BY *col-1, col-2, ...*]
[HAVING *criteria*]
[ORDER BY *col-1* [ASC | DESC], ...]
[UNION ...];

* per i nomi delle colonne e delle tabelle è possibile definire
“alias” con la clausola **AS**

Esempi di comando SELECT

→ SELECT * FROM *EDITORI*;

* individua
TUTTE le
colonne



→ SELECT *Titolo* FROM *LIBRI*
WHERE (*Anno* = 2000);

→ SELECT *Cognome*, *Nome* FROM *AUTORI*
WHERE ((*Cognome* > 'B') AND (*Cognome*
< 'C'));

Query e tabelle

Come conseguenza della normalizzazione le *query* coinvolgono normalmente più tabelle: nell'esempio per costruire un elenco di libri con i nominativi dell'editore e dell'autore (o degli autori) è necessario ricercare le informazioni in 3 tabelle distinte. Gli operatori **dell'algebra relazionale** si applicano alle tabelle di un database.

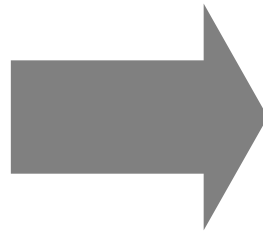
Operatori fondamentali dell'algebra relazionale:

- UNIONE
- INTERSEZIONE
- DIFFERENZA
- PRODOTTO CARTESIANO
- PROIEZIONE
- SELEZIONE (RESTRIZIONE)
- JOIN

operatore relazionale UNIONE

C1	C2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

C1	C2
<i>x</i>	<i>y</i>
<i>z</i>	<i>t</i>

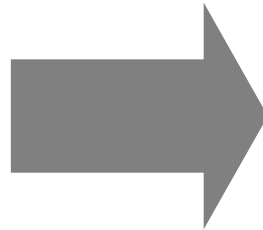


C1	C2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>
<i>x</i>	<i>y</i>
<i>z</i>	<i>t</i>

operatore relazionale INTERSEZIONE

C1	C2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

C1	C2
<i>e</i>	<i>f</i>
<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>



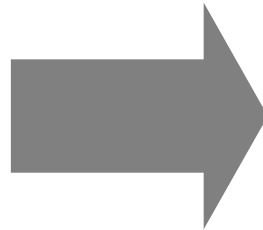
C1	C2
<i>a</i>	<i>b</i>
<i>e</i>	<i>f</i>

operatore relazionale

DIFFERENZA

C1	C2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

C1	C2
<i>e</i>	<i>f</i>
<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>

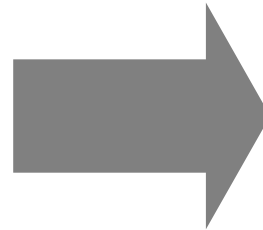


C1	C2
<i>c</i>	<i>d</i>
<i>x</i>	<i>y</i>

operatore relazionale PRODOTTO CARTESIANO

A1	A2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

B1	B2	B3
<i>x</i>	<i>y</i>	<i>z</i>
<i>i</i>	<i>j</i>	<i>k</i>

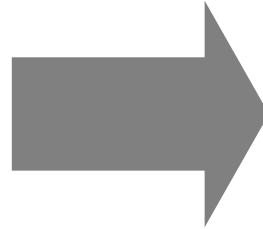


A1	A2	B1	B2	B3
<i>a</i>	<i>b</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>c</i>	<i>d</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>e</i>	<i>f</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>a</i>	<i>b</i>	<i>i</i>	<i>j</i>	<i>k</i>
<i>c</i>	<i>d</i>	<i>i</i>	<i>j</i>	<i>k</i>
<i>e</i>	<i>f</i>	<i>i</i>	<i>j</i>	<i>k</i>

operatore relazionale

PROIEZIONE

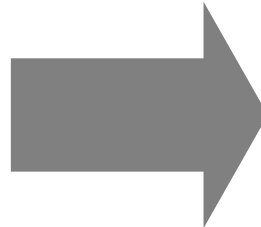
C1	C2	C3	C4	C5
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>
<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>
<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>	<i>t</i>



C1	C3	C5
<i>a</i>	<i>c</i>	<i>e</i>
<i>f</i>	<i>h</i>	<i>j</i>
<i>k</i>	<i>m</i>	<i>o</i>
<i>p</i>	<i>r</i>	<i>t</i>

operatore relazionale SELEZIONE (RESTRIZIONE)

C1	C2	C3	C4	C5
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>	<i>b</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>o</i>	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>
<i>a</i>	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>v</i>	<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>

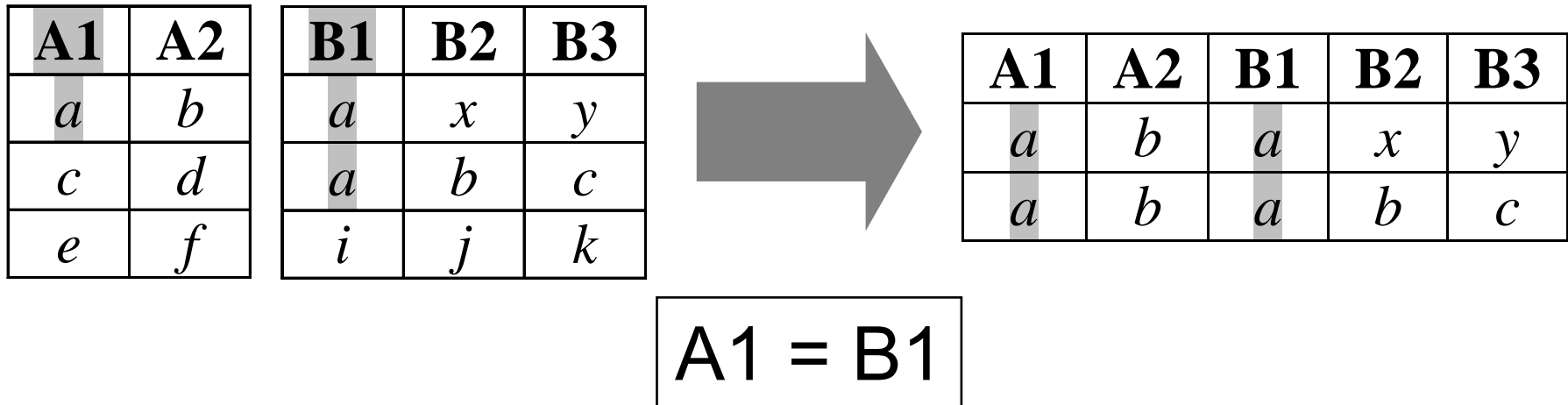


C1	C2	C3	C4	C5
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
<i>a</i>	<i>b</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>v</i>	<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>

$$((C1 = a) \wedge (C2 = b)) \vee (C5 = z)$$

operatore relazionale

INNER JOIN^{1,2}



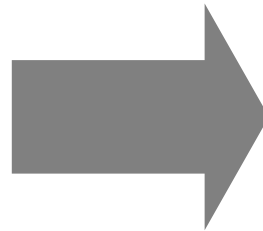
- ¹ l'operatore di INNER JOIN è realizzabile con l'applicazione dell'operatore di selezione al prodotto cartesiano
- ² "natural join" è il JOIN di 2 tabelle fondato sull'uguaglianza di tutte le colonne aventi lo stesso nome

operatore relazionale

SEMI-JOIN

A1	A2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

B1	B2	B3
<i>a</i>	<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>	<i>c</i>
<i>i</i>	<i>j</i>	<i>k</i>

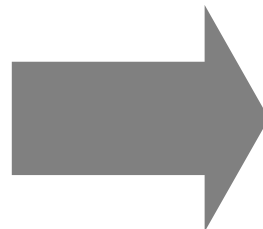


$A1 = B1$

A1	A2
<i>a</i>	<i>b</i>
<i>a</i>	<i>b</i>

A1	A2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

B1	B2	B3
<i>a</i>	<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>	<i>c</i>
<i>i</i>	<i>j</i>	<i>k</i>



$A1 = B1$

B1	B2	B3
<i>a</i>	<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>	<i>c</i>

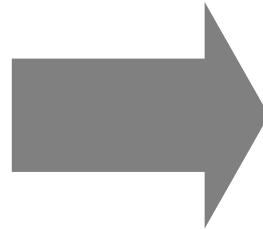
left semi-join: join con proiezione delle colonne A1 e A2

right semi-join: join con proiezione delle colonne B1, B2 e B3

operatore relazionale LEFT OUTER JOIN

A1	A2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

B1	B2	B3
<i>a</i>	<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>	<i>c</i>
<i>i</i>	<i>j</i>	<i>k</i>



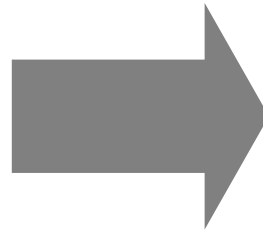
A1	A2	B1	B2	B3
<i>a</i>	<i>b</i>	<i>a</i>	<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>c</i>	<i>d</i>			
<i>e</i>	<i>f</i>			

A1 = B1

operatore relazionale RIGHT OUTER JOIN

A1	A2
<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>
<i>e</i>	<i>f</i>

B1	B2	B3
<i>a</i>	<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>	<i>c</i>
<i>i</i>	<i>j</i>	<i>k</i>



A1	A2	B1	B2	B3
<i>a</i>	<i>b</i>	<i>a</i>	<i>x</i>	<i>y</i>
<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>
		<i>i</i>	<i>j</i>	<i>k</i>

A1 = B1

Esempio di comando SELECT con 2 tabelle

- *SELECT LIBRI.Titolo, EDITORI.Nome
FROM LIBRI, EDITORI
WHERE (LIBRI.Editore = EDITORI.Codice);*
- *SELECT Titolo, Nome
FROM LIBRI INNER JOIN EDITORI
ON (Editore = Codice);*

è possibile evitare la qualificazione dei nomi delle colonne
con i nomi delle tabelle se non si incorre in ambiguità

Esempio di comando **SELECT con 3 tabelle**

→ *SELECT AUTORI.Cognome, AUTORI.Nome, LIBRI.Titolo
FROM AUTORI, LIBRI, LIBRI-AUTORI
WHERE (LIBRI.ISBN = LIBRI-AUTORI.ISBN) AND
(AUTORI.CF = LIBRI-AUTORI.CF) AND
(LIBRI.Anno > 2000)
ORDER BY AUTORI.Cognome, AUTORI.Nome;*

Esempio di comando **SELECT con OUTER JOIN**

→ *SELECT EDITORI.Nome, LIBRI.Titolo*
FROM EDITORI
LEFT JOIN LIBRI
ON LIBRI.Editore = EDITORI.Codice;

*l'elenco include gli Editori a cui
non corrisponde nessun Libro*

Le funzioni SQL consentono di costruire *query* con attributi impliciti o clausole complesse

→ SELECT COUNT(*) FROM *LIBRI*
WHERE (*Anno* = 2000);

numero di Libri pubblicati
nell'anno 2000

→ SELECT *Titolo*, YEAR(CURDATE())-*Anno*
FROM *LIBRI*;

elenco di tutti i Titoli
dei Libri con la loro
"età"

→ SELECT *Titolo* FROM *LIBRI*
WHERE (YEAR(CURDATE())-*Anno*) < 10;

→ SELECT *Titolo* FROM *LIBRI*
WHERE LENGHT(*Titolo*) < 20

elenco di tutti i Titoli dei Libri
pubblicati negli ultimi 10 anni

elenco di tutti i Libri con Titoli
più corti di 20 caratteri

Il *pattern-matching* delle stringhe nelle *query* è possibile con l'operatore **LIKE***

→ *SELECT* *Cognome*, *Nome*
FROM *AUTORI*
WHERE *Cognome* **LIKE** 'A%';

elenco degli
Autori con
Cognome che
inizia con "A"

→ *SELECT* *Nome*
FROM *EDITORI*
WHERE *Codice* **LIKE** '999_';

elenco degli
Editori con
codice SIAE del
tipo "999?"

* MySQL estende l'operatore SQL standard LIKE con funzioni di ricerca ed elaborazione delle "espressioni regolari"

Esempio di comando SELECT con clausola GROUP BY

Titolo	Editore	ISBN	Pagine	Costo
RDBMS	O'Reilly	88-0000-123-1	128	12,50
SQL	Addison Wesley	0-12345-000-9	1024	99,90
XML	O'Reilly	66-9999-987-2	512	50,00

→ *SELECT Editore, AVG(Costo)*
FROM LIBRI
WHERE Pagine > 500
GROUP BY Editore;

©2006 Giorgio Meini

elenco degli Editori
con associato costo
medio dei Libri aventi
più di 500 Pagine

Esempio di comando SELECT con clausole GROUP BY e HAVING

→ SELECT *Editore*
FROM *LIBRI*
GROUP BY *Editore*
HAVING COUNT(*) > 100;

elenco degli Editori
che pubblicano più di
100 Libri

→ SELECT *Editore*
FROM *LIBRI*
GROUP BY *Editore*
HAVING MAX(*Pagine*) > 1000;

elenco degli Editori che
pubblicano Libri con
più di 1000 Pagine

Esempio di comando **SELECT** con *query* nidificata *

→ **SELECT** *Titolo*
FROM *LIBRI*
WHERE *Editore* **IN**
(**SELECT** *Codice SIAE*
FROM *EDITORI*
WHERE *Città* = 'Milano');

elenco dei libri
pubblicati a
Milano

•alcune query innestate possono essere riscritte in forma lineare utilizzando operatori JOIN

Esempio di comando **SELECT** con *query* nidificata scalare

```
→ SELECT Titolo  
FROM LIBRI  
WHERE Pagine >  
(SELECT AVG(Pagine)  
FROM LIBRI);
```

elenco dei libri
aventi più pagine
della media

query scalare

Esempi di comando SELECT con *query* nidificata NON scalare

→ SELECT *Titolo*
FROM *LIBRI*
WHERE *Pagine* <=
ALL(SELECT *Pagine*
FROM *LIBRI*);

elenco dei libri
aventi il minimo
numero di pagine

→ SELECT *Titolo*
FROM *LIBRI*
WHERE *Editore* =
ANY (SELECT *Codice SIAE*
FROM *EDITORI*
WHERE *Città* = 'Milano');

elenco dei libri
pubblicati a
Milano

Esempio di comando **SELECT** con *query* nidificata correlata

```
→ SELECT Nome
FROM EDITORI
WHERE EXIST
(SELECT *
FROM LIBRI
WHERE Pagine > 1000
AND Editore = EDITORI.Codice SIAE);
```

elenco degli editori
che stampano
almeno un libro con
più di 1000 pagine

query correlata

4 funzionalità avanzate dei RDBMS:

- **Integrità:** vincoli, [trigger]
- **Prestazioni:** indici, [partizioni]
- **Sicurezza:** privilegi, ...
- **Concorrenza:** transazioni

Esempi di comando CREATE TABLE con vincoli* sui valori delle colonne

- CREATE TABLE *LIBRI*
(*Titolo* VARCHAR(256) NOT NULL,
Anno INT,
ISBN CHAR(13) NOT NULL PRIMARY KEY,
Editore CHAR(4) NOT NULL
REFERENCES *EDITORI*(*Codice*),
CHECK (*Anno* > 1900));
- CREATE TABLE *AUTORI*
(*Cognome* VARCHAR(32) NOT NULL,
Nome VARCHAR(32) NOT NULL,
CF CHAR(16) NOT NULL PRIMARY KEY,
Sesso CHAR(1),
CHECK (*Sesso* IN ('M', 'F'))),

***My-SQL non**
supporta i
vincoli, ma il
tipo di dati
ENUM
consente in
molti casi di
evitarne l'uso

Esempi di creazione di indici sui valori delle colonne di una tabella

- CREATE TABLE *AUTORI*
(*Cognome* VARCHAR(32) NOT NULL,
Nome VARCHAR(32) NOT NULL,
CF CHAR(16) NOT NULL PRIMARY KEY,
INDEX *ind1* (*Cognome*));
- CREATE INDEX *ind1* ON *AUTORI* (*Cognome*);
- CREATE TABLE *EDITORI*
(*Nome* VARCHAR(64) NOT NULL,
Città VARCHAR(32),
Codice CHAR(4) NOT NULL PRIMARY KEY,
UNIQUE INDEX *ind2* (*Nome*));

nella tabella Editori non possono essere inseriti più record con il campo "Nome" uguale

Ogni utente My-SQL dispone o meno di privilegi sugli oggetti del database

	COLONNA	TABELLA	DATABASE
ALTER	X	X	
CREATE		X	X
DELETE	X	X	
DROP		X	X
INDEX		X	
INSERT	X	X	
SELECT	X	X	
UPDATE	X	X	

©2006 Giorgio Meini

Esempi di concessione/revoca dei privilegi sugli oggetti ad un utente*

- GRANT SELECT (*Titolo, Anno*) ON *LIBRI* TO *Giorgio*;
- GRANT DELETE, INSERT ON *AUTORI* TO *Giorgio*;
- REVOKE UPDATE *ISBN* ON *LIBRI* FROM *Giorgio*;

***My-SQL** memorizza i privilegi di accesso agli oggetti del database in tabelle speciali

Transazioni e proprietà ACID

[www.wikipedia.org]

- Una **transazione** è una sequenza di operazioni, che può concludersi con un successo o un insuccesso: in caso di successo, il risultato delle operazioni deve essere permanente, mentre in caso di insuccesso si deve tornare allo stato precedente all'inizio della transazione.
- Una transazione, per essere tale, deve godere delle cosiddette proprietà **ACID**, particolarmente significative nei sistemi in cui possono essere eseguite più transazioni contemporaneamente.
- I database che supportano le transazioni sono denominati **database transazionali**: molti database moderni appartengono a questa categoria.

Le proprietà ACID delle transazioni:

- **Atomicità (Atomicity):** la transazione è indivisibile e la sua esecuzione deve essere totale o nulla
- **Consistenza (Consistency):** all'inizio ed alla fine di una transazione il database deve essere in uno stato consistente
- **Isolamento (Isolation):** ogni transazione deve essere eseguita in modo isolato ed indipendente dalle altre
- **Persistenza (Durability):** una volta che una transazione termina con un *commit-work* i cambiamenti apportati non devono essere persi

I comandi My-SQL per la gestione delle transazioni*

BEGIN;	inizia una nuova transazione
COMMIT;	conclude con successo una transazione salvando permanentemente il risultato
ROLLBACK;	conclude con insuccesso una transazione annullando tutte le operazioni eseguite

* è necessario utilizzare tabelle di tipo InnoDB e disabilitare la modalità auto-commit con il comando **SET AUTOCOMMIT=0**; il comando **SET TRANSACTION** consente di modificare il livello di isolamento delle transazioni concorrenti

My-SQL DB programming:

- **API C/C++**
- **ODBC (Open DB Connectivity)**
- **JDBC (Java DB Connectivity)**
- **API PHP**
- **API Perl (DBD/DBI), ...**
- **...**

Riferimenti bibliografici

- S. Roman, “Access database design & programming”, 2nd edition, O’Reilly, 2002
- G. Reese, R.J. Yarger, T. King & H.E. Williams, “Managing & using MySQL”, 2nd edition, O’Reilly, 2002