Basi di Dati - VI

Corso di Laurea in Informatica

Anno Accademico 2012/2013

Paolo Baldan

baldan@math.unipd.it

http://www.math.unipd.it/~baldan

Il DDL di SQL

SQL per la definizione di Basi di Dati

- SQL non è solo un linguaggio di interrogazione (Query Language), ma anche un linguaggio per la **definizione** di basi di dati (Data-definition language (DDL))
  - creazione della BD e della struttura logica delle tabelle
    - **`CREATE SCHEMA`** `Nome` **`AUTHORIZATION`** `Utente`
    - **`CREATE TABLE`** o **`VIEW`**, con vincoli
  - vincoli di integrità
    - su attributi di una ennupla (es. `NOT NULL`)
    - intrarelazionali (es. chiave)
    - interrelazionali (es. integrità referenziale)

SQL per la definizione di Basi di Dati

- conoscenza procedurale
  stored procedures, trigger
- modifica dello schema
  `ALTER ...`
- struttura fisica, i.e. come memorizzare i dati e strutture per l'accesso (es. **`CREATE INDEX`**)
- controllo degli accessi ai dati (es. **`GRANT`**)

Il DDL di SQL

- In SQL il livello intensionale ha concettualmente una struttura **gerarchica**, dovuta alla compresenza di utenti multipli e alla necessità di un utente di poter creare più schemi:
  - **tabella**: insieme di colonne;
  - **schema**: insieme di tabelle (con vincoli, trigger, procedure, ecc.)
  - **catalogo**: insieme di schemi;
- Una tabella `NomeTab` dello schema `NomeSchema` può essere riferita come `NomeSchema.NomeTab`
- Schema ~ database in MySQL

Creazione di uno schema

- Uno schema può essere creato con:

```
CREATE SCHEMA Università AUTHORIZATION rossi
```

  - Ogni creazione di tabelle, viste, ecc. è per default associata allo schema creato più recentemente
  - Nota: **AUTHORIZATION** non si indica in MySQL
- Uno schema può essere eliminato mediante un comando

```
DROP SCHEMA Nome [ CASCADE | RESTRICT ]
```

- Esempio

```
DROP SCHEMA Università CASCADE
```


Schemi

- Uno schema può contenere varie tabelle delle quali esistono più tipi:
  - tabelle base (base tables)
    - i metadati appartengono allo schema;
    - i dati sono fisicamente memorizzati
  - viste (views o viewed tables)
    - i metadati sono presenti nello schema
    - i dati non sono fisicamente memorizzati (ma prodotti dalla valutazione di un'espressione)
- Le tabelle di base possono essere anche temporanee (temporary) in MySQL

Creazione di una tabella

- Una tabella (base), creata con il comando **CREATE TABLE**, è un insieme di colonne/attributi per ciascuna delle quali va specificato:
  - nome
  - tipo di dato, che può essere
    - predefinito
    - definito dall'utente (dominio) costruito con il comando **CREATE DOMAIN**; e.g.

```
CREATE DOMAIN Voto AS SMALLINT
              CHECK (Voto <= 30 AND Voto >= 18)
```

- Nota: **CREATE DOMAIN** non disponibile in MySQL.

Tipi di dato predefimiti

- SQL supporta un certo numero di tipi di dato atomici; i principali sono
  - tipi interi:
    - **INTEGER** (o **INT**), **TINYINT**, **SMALLINT**, **MEDIUMINT**, **BIGINT**, ...
  - virgola mobile:
    - **FLOAT(p)** (o **REAL**), **DOUBLE**
  - stringhe di bit:
    - **BIT(x)**, **BIT VARYING(x)**
  - booleani:
    - **BOOL**

Tipi di Dato Predefiniti

- stringhe di caratteri:
  - **CHAR(x)** (o **CHARACTER(x)**)
  - **VARCHAR(x)** (o **CHAR VARYING(x)**)
- enumerazione e insieme:
  - **ENUM(list)**, **SET(list)**
  - es. SET('a','b','c'), funzioni FIND_IN_SET('a',set)
- date e ore:
  - **DATE, YEAR, TIMESTAMP**
- intervalli temporali:
  - **INTERVAL** {**YEAR, MONTH, DAY, HOUR, MINUTE, SECOND**}

Tipi di Dato Predefiniti

- testo e oggetti binari:
  - **TEXT, BLOB, LONGTEXT, LONGBLOB, TINYTEXT**, ...
- ....

Vincoli di ennupla

- Per una colonna si possono specificare anche
  - un eventuale **valore di default**, con la clausola **DEFAULT**; può essere
    - un valore costante o NULL
    - il risultato di una chiamata di funzione 0-aria (*e.g.* CURRENT_DATE());
  - AUTO_INCREMENT
  - un eventuale vincolo; *e.g.* NOT NULL, CHECK (<CONDIZIONE>)

```
CREATE TABLE Studenti (
       Nome       VARCHAR(10) NOT NULL,
       Cognome    VARCHAR(10) NOT NULL,
       Matricola  CHAR(6),
       Nascita    YEAR,
       Provincia  CHAR(2) DEFAULT 'PD',
       Tutor      CHAR(6)
);
```

Vincoli intra- e inter-relazionali

- In una tabella sono anche inclusi vincoli
  - intrarelazionali
    - **PRIMARY KEY**: designa un insieme di attributi come chiave primaria;
    - **UNIQUE**: designa un insieme di attributi come chiave (non primaria);
  - interrelazionali
    - **FOREIGN KEY**: designa
      - un insieme di attributi come chiave esterna
      - un'eventuale azione da intraprendere (**SET NULL**, **SET DEFAULT**, **CASCADE**, **RESTRICT**) se il vincolo viene violato a causa di cancellazione (**ON DELETE**) o modifica (**ON UPDATE**) della riga riferita
- Ai vincoli di tabella può essere dato un nome (ad esempio per poterli eliminare)

Esempio (quasi) completo: base di dati Università


```
CREATE TABLE Studenti (
       Nome       VARCHAR(10) NOT NULL,
       Cognome    VARCHAR(10) NOT NULL,
       Matricola  CHAR(5) PRIMARY KEY,
       Nascita    YEAR,
       Provincia  CHAR(2) DEFAULT 'PD',
       Tutor      CHAR(5),
       FOREIGN KEY (Tutor) REFERENCES Studenti(Matricola)
                           ON UPDATE CASCADE
                           ON DELETE SET NULL
) ENGINE=InnoDB;


CREATE TABLE Docenti (
       CodDoc     CHAR(3) PRIMARY KEY,
       Nome       VARCHAR(8),
       Cognome    VARCHAR(8)
) ENGINE=InnoDB;
```


Esempio Completo


```
CREATE TABLE Esami (
       Codice     CHAR(4) PRIMARY KEY,
       Materia    CHAR(3),
       Candidato  CHAR(5) NOT NULL,
       Data       DATE,
       Voto       INT,
       Lode       CHAR(1),
       CodDoc     CHAR(3) NOT NULL,
       UNIQUE KEY (Materia,Candidato),
       FOREIGN KEY (Candidato) REFERENCES Studenti(Matricola)
                               ON UPDATE CASCADE,

       FOREIGN KEY (CodDoc)    REFERENCES Docenti(CodDoc)
                               ON UPDATE CASCADE
) ENGINE=InnoDB;
```


Riassunto

- SQL come linguaggio di definizione della BD (DDL)
- creazione e cancellazione di database
  - **CREATE SCHEMA** Università
  - **DROP SCHEMA** Università
- Creazione di tabelle: **nome** della **tabella**, **nome** e **tipo** degli **attributi** e ...
  vincoli

Definizione di Tabelle

- Ciò che si crea con un **CREATE** si può cambiare con il comando **ALTER** ed eliminare con il comando **DROP**.

- Aggiungere nuovi attributi

```
ALTER TABLE  Studenti
      ADD COLUMN Nazionalita VARCHAR(10) DEFAULT 'Italiana';
```

- Eliminare attributi

```
ALTER TABLE  Studenti
      DROP COLUMN Provincia;
```

- Modificare il tipo di una colonna

```
ALTER TABLE  Studenti
      MODIFY COLUMN Nazionalita VARCHAR(15);
```


Definizione di Tabelle

- Aggiungere ed eliminare vincoli

```
ALTER TABLE Docenti
      ADD UNIQUE KEY (RecapitoTel);

ALTER TABLE  Studenti
      DROP FOREIGN KEY nome_vincolo
```

- E molto altro …

```
ALTER TABLE Studenti
      ALTER COLUMN Provincia DROP DEFAULT;
```


Eliminare tabelle

- Le tabelle possono essere anche distrutte, mediante il comando **DROP TABLE**, con cui si rimuovono dallo schema la definizione della tabella e dai dati tutte le righe che la istanziano; *e.g.*

- **DROP TABLE** Studenti **CASCADE**

- **DROP TABLE** Docenti **RESTRICT**

Tabelle Inizializzate

- **Tabelle inizializzate**:
  - **CREATE TABLE** Nome [**AS**] EspressioneSELECT

- **Esempio**: Tutor degli studenti di Padova

```
CREATE TABLE TutorPD AS
      SELECT  t.Matricola, t.Nome, t.Cognome
      FROM    Studenti t
      WHERE   t.Matricola IN (SELECT s.Tutor
                              FROM   Studenti s
                              WHERE  s.Provincia='PD');
```

Tabelle Inizializzate

- Creazione dello storico degli esami

```
CREATE TABLE EsamiFino2006 AS
       SELECT  *
       FROM    Esami e
       WHERE   YEAR(e.Data) <= '2006';

DELETE FROM Esami
WHERE YEAR(e.Data) <= '2006';
```


Viste (o tabelle calcolate)

- Definite da

```
CREATE VIEW Nome [(Attributo {, Attributo})]
            AS (EspressioneSELECT);
```

- Risultato di un'espressione SQL che riferisce tabelle di base e altre viste
- Dati non fisicamente memorizzati

```
CREATE VIEW VotiMedi(Matricola, Media) AS
       SELECT e.Candidato,avg(Voto)
       FROM   Esami e
       GROUP BY e.Candidato;
```


Viste

- Calcolate ad ogni interrogazione (modulo caching)
- L'ottimizzatore può decidere di combinare la loro definizione con la query
  - Query:

```
SELECT s.Cognome, vm.Matricola, vm.Media
FROM   Studenti s NATURAL JOIN VotiMedi vm
WHERE  s.Provincia='VE';
```

  - Potrebbe diventare

```
SELECT s.Cognome, s.Matricola, avg(e.Voto)
FROM   Studenti s, Esami e
WHERE  s.Matricola=e.Candidato AND s.Provincia='VE'
GROUP BY s.Matricola;
```


Viste Modificabili

- Le viste si interrogano come le altre tabelle, ma in generale non si possono modificare.
- Deve esistere una corrispondenza biunivoca fra le righe della vista e un sottoinsieme di righe di una tabella di base, ovvero:
  1. SELECT senza DISTINCT e solo di attributi (non calcolati)
  2. FROM una sola tabella modificabile
  3. UNION non presente
  4. GROUP BY e HAVING non sono presenti nella definizione.

Viste (riassunto)

- Creazione

```
CREATE VIEW NomeVista AS
SELECT ...
```

- Interrogabili come normali tabelle
- Modificabili solo se le (parti di) righe che si modificano sono 1-1 con quelle di una tabella base

Utilità delle viste

- Per nascondere certe modifiche dell'organizzazione logica dei dati (indipendenza logica).
  - Es. Divisione di Studenti in `Matricole` e `NonMatricole`
- Per proteggere i dati
  - Es. si può dare ad un utente accesso solo ad una parte limitata/aggregata dei dati
- Per offrire visioni diverse degli stessi dati senza ricorrere a duplicazioni (es. Vedi `VotiMedi`)
- Per rendere più semplici, o per rendere possibili, alcune interrogazioni

Viste e interrogazioni impossibili

- Trovare la media dei voti massimi ottenuti nelle varie provincie

Viste e interrogazioni impossibili

- Trovare la media dei voti massimi ottenuti nelle varie provincie
- Non si può fare

```
SELECT avg(max(e.Voto))
FROM Studenti s JOIN Esami e ON s.Matricola=e.Candidato
GROUP BY s.Provincia;
```

Viste e interrogazioni impossibili

- Trovare la media dei voti massimi ottenuti nelle varie provincie
- Non si può fare

```
SELECT avg(max(e.Voto))
FROM Studenti s JOIN Esami e ON s.Matricola=e.Candidato
GROUP BY s.Provincia;
```

- Invece

```
CREATE VIEW ProvMax(Provincia, Max) AS
     SELECT s.Provincia, max(e.Voto)
     FROM Studenti s JOIN Esami e ON s.Matricola=e.Candidato
     GROUP BY s.Provincia;
SELECT AVG(Max) FROM ProvMax;
```


Viste e interrogazioni difficili

- Le province dove la media dei voti degli studenti è massima
  -
```
CREATE VIEW ProvMedia (Provincia, Media) AS
     SELECT s.Provincia, AVG(e.Voto)
     FROM Studenti s JOIN Esami e ON s.Matricola=e.Candidato
     GROUP BY s.Provincia;
```
  -
```
SELECT Provincia, Media
     FROM ProvMedia
     WHERE Media = (SELECT MAX(Media) FROM ProvMedia);
```
- equivalente a ...

Viste e interrogazioni difficili

- equivalente a
  -
```
SELECT s.Provincia, AVG(e.Voto)
FROM   Studenti s JOIN Esami e ON s.Matricola=e.Candidato
GROUP BY s.Provincia
HAVING avg(e.voto) >=ALL (SELECT avg(e.Voto)
                          FROM Studenti s JOIN Esami e
                                 ON s.Matricola=e.Candidato
                          GROUP BY s.Provincia);
```
- talvolta non ammessa perché `HAVING` coinvolge una sottoselect (ammesso in MySQL).

Sintassi per la definizione di tabelle

-
```
CREATE TABLE  Nome (
    Attributo  Tipo [Default] [VincoloAttributo]
   {, Attributo  Tipo [Default] [VincoloAttributo]}
   {, VincoloTabella}
)

Default := DEFAULT { valore | NULL }

VincoloAttributo := [NOT] NULL | CHECK (Condition)

VincoloTabella := PRIMARY KEY | UNIQUE |
                  FOREIGN KEY (Attr) REFERENCES Tab(Attr)
```

Vincoli di Integrità

- Vincoli su attributi
  - VincoloAttributo :=
    [**NOT NULL** [**UNIQUE**]] | [**CHECK** (Condizione)]
- Vincoli su tabella
  - VincoloTabella := **UNIQUE** (Attributo {, Attributo})
    | **CHECK** (Condizione) |
    | **PRIMARY KEY** [Nome] (Attributo {, Attributo})
    | **FOREIGN KEY** [Nome] (Attributo {, Attributo})
    REFERENCES Tabella [(Attributo {, Attributo})]
    [**ON DELETE CASCADE** | **RESTRICT** | **SET DEFAULT** | **SET NULL**]
    [**ON UPDATE CASCADE** | **RESTRICT** | **SET DEFAULT** | **SET NULL**]

Comandi utili

- SHOW DATABASES (o SHOW SCHEMAS)
- USE <nome del database>
- SOURCE <script>
- SHOW tables
- SHOW CREATE TABLE <nome tabella>
- DESCRIBE <nome tabella>
- SHOW WARNINGS
- LOAD DATA LOCAL INFILE 'Studenti.txt' INTO TABLE Studenti;

Wednesday, May 1, 2013

Storage engine

- Il default e` MyIsam
- Si puo` cambiare con
  - SET storage_engine=InnoDB
  - mysqld --default-storage-engine=InnoDB

Aspetti procedurali

- Uno schema può comprendere anche informazione procedurale codificata come
  - **Stored procedure**
    - espressi in un linguaggio interno (es. PL/SQL)
    - eseguite dal DBMS su richiesta delle applicazioni
    - funzioni e procedure
  - **Trigger**
    - procedure memorizzate
    - eseguite automaticamente dal DBMS al verificarsi di dati eventi

Wednesday, May 1, 2013

Controllo degli accessi

- Basato sui concetti di
  - utente
  - privilegio/autorizzazione (create, select, delete, update, ...)
  - (profilo,ruolo)
- Chi crea lo schema (autorizzato dall'amministratore) della BD può fare **CREATE**, **ALTER** e **DROP**
- Chi crea una tabella stabilisce i modi in cui altri possono farne uso:
  - **GRANT** Privilegi **ON** Oggetto **TO** Utenti [**WITH GRANT OPTION**]

Controllo degli Accessi

- Tipi di privilegi:
  - **SELECT** [(Attributi)]: lettura di dati
  - **INSERT** [(Attributi)]: inserire record (con valori non nulli per gli attributi)
  - **UPDATE** [(Attributi)]: modificare record (o solo gli attributi)
  - **DELETE**: cancellazione di record
  - **REFERENCES** [(Attributi)]: definire chiavi esterne in altre tabelle che riferiscono gli attributi.
- **WITH GRANT OPTION**: si possono trasferire i privilegi ad altri utenti.

Controllo degli accessi (cont.)

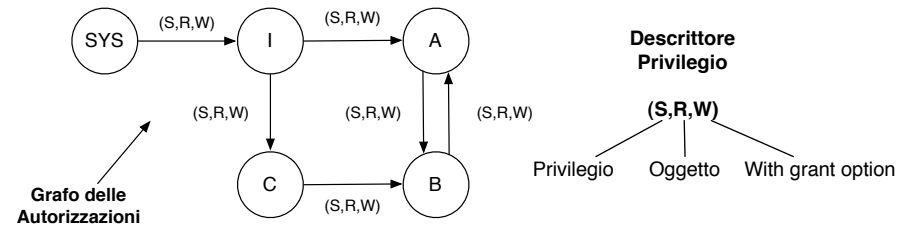
- Chi definisce una tabella o una VIEW ottiene automaticamente tutti i privilegi su di essa, ed è l'unico che può fare DROP e può autorizzare altri ad usarla con GRANT.
- Nel caso di viste, il "creatore" ha i privilegi che ha sulle tabelle usate nella definizione.
- Le autorizzazioni si annullano con il comando:
  - **REVOKE** Privilegi, [**GRANT OPTION**] **ON** Oggetto **FROM** Utenti [**CASCADE**]
- Quando si toglie un privilegio a U, lo si toglie anche a tutti coloro che lo hanno avuto solo da U.

Esempi di GRANT

- **GRANT INSERT, SELECT ON** Esami **TO** rossi
- **GRANT DELETE ON** Esami **TO** verdi **WITH GRANT OPTION**
  - verdi può cancellare record e autorizzare altri a farlo.
- **GRANT UPDATE** (Voto) **ON** Esami **TO** rossi
  - l'utente rossi può modificare solo il voto degli esami.
- **GRANT SELECT ON** VistaEsamiBD **TO** baldan
  - l'utente baldan può interrogare solo i suoi esami.

Grafo delle Autorizzazioni

- L'utente I ha creato la tabella R e innesca la seguente successione di eventi:
  - I: GRANT SELECT ON R TO A WITH GRANT OPTION
  - A: GRANT SELECT ON R TO B WITH GRANT OPTION
  - B: GRANT SELECT ON R TO A WITH GRANT OPTION
  - I: GRANT SELECT ON R TO C WITH GRANT OPTION
  - C: GRANT SELECT ON R TO B WITH GRANT OPTION

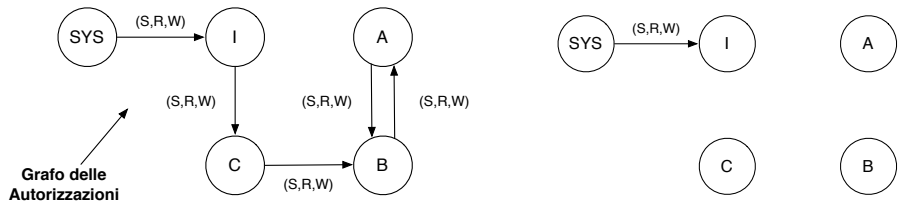


Grafo delle autorizzazioni: proprietà

- Se un nodo N ha un arco uscente con un privilegio, allora esiste un cammino da SYS a N con ogni arco etichettato dallo stesso privilegio + W(ith grant option).
- Effetto del REVOKE, ad es.

  I: **REVOKE SELECT ON** R **FROM** A **CASCADE**

- e poi I: **REVOKE SELECT ON** R **FROM** C **CASCADE**



Privilegi e MySQL

- Il modello MySQL è più semplice
  - privilegi a utenti (con la possibilità di GRANT OPTION), senza proprietà implicita del creatore e soprattutto senza CASCADE
  - **SHOW GRANTS [FOR** account]

```
+-----------------------------------------------------------+
| Grants for @localhost                                     |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO 'baldan'@'localhost'                 |
| GRANT ALL PRIVILEGES ON 'Univ'.* TO 'baldan'@'localhost'  |
|                      IDENTIFIED BY PASSWORD '...'         |
|                      WITH GRANT OPTION                    |
+-----------------------------------------------------------+
```


Privilegi e MySQL

- Creazione di un utente
  - **CREATE USER** user@host **IDENTIFIED BY** "password"
  - **GRANT** privs **ON** obj **TO** user@host **IDENTIFIED BY** "password"
- Cambio della password
  - **SET PASSWORD FOR** user@'%' = PASSWORD('...')
- Rinomina
  - **RENAME USER** user@host **TO** user1@host
- Elimina
  - **DROP USER** user@host

Indici

- Indice su un attributo A (o più) della tabella R
  - insieme ordinato di coppie (a, {r1, ..., rk})
  - ogni rj riferisce una ennupla di R
- Memorizzato in una struttura ad albero (B-albero)
- Non influenza la formulazione delle query, ma è usato per l'ottimizzazione delle stesse
  - es. natural o equi-join

Creazione di indici

- Esempio:

R

A	....	C
pippo		4
pluto		2
minnie		3
paperino		1
paperone		3

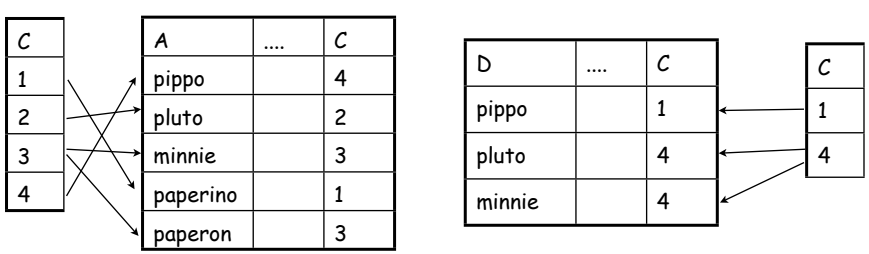
S

D	....	C
pippo		1
pluto		4
minnie		4

Creazione di indici

- Esempio:

R



S

Creazione di indici (cont.)

- Non è un comando standard SQL e quindi ci sono differenze nei vari sistemi
  - `CREATE [UNIQUE] INDEX NomeIdx ON Tabella(Attributi)`
  - `DROP INDEX NomeIdx`

Come scegliere gli indici

- Attenzione
  - Occupano spazio (media ~ 20% della relazione)
  - Devono essere aggiornati ad ogni transizione di stato
- Regole del pollice: indici su
  - attributi selettivi
  - attributi non modificati frequentemente
  - chiavi esterne per le operazioni di giunzione
  - attributi usati frequentemente in ORDER BY, DISTINCT, GROUP BY
  - attributi in condizioni WHERE (con uguaglianza/confronti)

Catalogo

- Schema che contiene i metadati (anche dello schema stesso!)
- Es. in MySQL:
  - `mysql`
    - tabella `User(host, user password, privilegi, ...)`
    - …
  - `information_schema`
    - Tabella degli schemi

      `Schemata(..., schema_name, ...)`

    - Tabella delle tabelle (type = view or table):

      `Tables(schema, name, ..., rows, ...)`

Catalogo

- Tabella delle views

  `Views(...,schema, name, definition, definer, ...)`

- Tabella dei trigger
- Tabella delle procedure
- …

Backup

- È possibile creare una copia di backup di un database con

  `mysqldump -u` *`account`* `-p -r` *`dbname`* `[`*`tab1 ... tabn`*`]`

  - genera su std output uno script per generare e popolare il DB
  - ripristino mediante esecuzione dello script
- In modo interattivo

  `SELECT * FROM` `tabella` **`INTO OUTFILE`** *`'file'`*

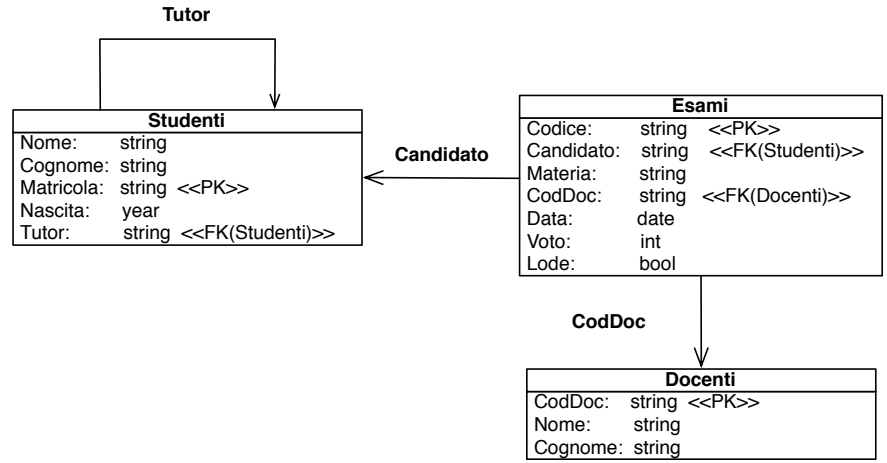
  - genera un file (TAB-separated), con il contenuto della tabella
  - ripristino con **`LOAD DATA INFILE`**

Riepilogo

- DDL consente la definizione di database, tabelle, viste e indici.
- Le tabelle si possono modificare aggiungendo o togliendo attributi e vincoli.
- Le viste si possono interrogare come ogni altra tabella, ma in generale non consentono modifiche dei dati.
- I comandi GRANT / REVOKE + viste offrono ampie possibilità di controllo degli uso dei dati.
- SQL consente di dichiarare altri tipi di vincoli, oltre a quelli fondamentali di chiave e referenziale.
- Oltre alle tabelle fanno parte dello schema le procedure e i trigger.

Esercizio

- Consideriamo ancora lo schema (Vedi definizione SQL precedente)



Esercizio

1. Matricola e cognome degli studenti che hanno sostenuto BD e non FIS
2. Matricola, cognome e voto medio degli studenti, la cui media è superiore alla media dei voti di tutti gli studenti
3. Data/e in cui è stato sostenuto il massimo numero di esami
4. Numero degli studenti che non hanno fatto esami
5. Per ogni materia, il numero degli esami sostenuti nel 2006 e la media nel 2005
6. Cancellare lo studente con matricola 71523 e tutti i suoi esami.
7. Gli studenti che hanno preso tutti trenta con il docente la cui media dei voti assegnati è minima.

Considerazioni varie

For all senza for all

- Gli studenti che hanno preso tutti trenta

```
SELECT s.*
FROM   Studenti s
WHERE  s.Matricola NOT IN (SELECT  e.Candidato
                           FROM    Esami e
                           WHERE   e.Voto <> 30);
```

- Si puo fare senza sottoselect?

For all senza for all: errori

- Gli studenti che hanno preso tutti trenta

```
SELECT s.*
FROM   Studenti s, Esami e
WHERE  s.Matricola <> e.Candidato AND
       e.Voto <> 30;
```

```
SELECT s.*
FROM   Studenti s, Esami e
WHERE  s.Matricola = e.Candidato AND
       e.Voto = 30;
```


For all senza for all: complemento

- Con complemento

```
SELECT s.*
FROM   Studenti s
  EXCEPT
SELECT s.*
FROM   Studenti s JOIN Esami e ON (s.Matricola=e.Candidato)
WHERE  e.Voto <> 30;
```

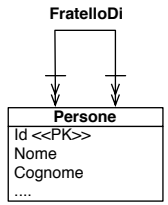

For all senza for all: giunzione esterna

- Con giunzione esterna

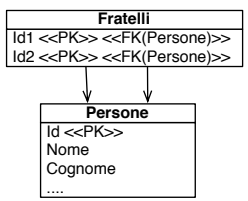
```
SELECT s.*
FROM   Studenti s LEFT JOIN Esami e
       ON (s.Matricola=e.Candidato AND e.Voto <> 30)
WHERE  e.Voto IS NULL;
```

Associazioni simmetriche

- Supponiamo di avere uno schema concettuale con una associazione simmetrica



- Questo può essere tradotto nello schema relazionale come:



Associazioni simmetriche (cont.)

- Cosa si inserisce nella tabella Fratelli?

Id	Nome	Cognome	...
13	Giorgio	Conte	...
...	...	...	...
21	Paolo	Conte	...

Persone

  - Tutte le ennuple (Id1,Id2) tali che Id1 è fratello di Id2?
    - es. se 13 e 21 sono fratelli, inseriamo sia (13,21) che (21,13)
  - Solo una ennupla per ciascuna coppia di fratelli
    - es. se 13 e 21 sono fratelli, inseriamo solo (13,21)
- Ambedue le soluzioni hanno problemi ...

Associazioni simmetriche (cont.)

- Se inserisco **tutte** le ennuple ...
  - Ridondanza
  - "Difficile" ottenere la lista dei fratelli senza ripetizioni. Es. la query

```
SELECT p1.*, p2.*
FROM   Persone p1, Fratelli f, Persone p2
WHERE  p1.Id=f.Id1 AND f.Id2=p2.Id
```

restituisce

```
+----+---------+---------+----+---------+---------+
| Id | Nome    | Cognome | Id | Nome    | Cognome |
+----+---------+---------+----+---------+---------+
| 13 | Giorgio | Conte   | 21 | Paolo   | Conte   |
| 21 | Paolo   | Conte   | 13 | Giorgio | Conte   |
+----+---------+---------+----+---------+---------+
```


Associazioni simmetriche (cont.)

  - Devo modificare la query

```
SELECT p1.*, p2.*
FROM   Persone p1, Fratelli f, Persone p2
WHERE  p1.Id=f.Id1 AND f.Id2=p2.Id
       AND f.Id1 < f.Id2
```

```
+----+---------+---------+----+---------+---------+
| Id | Nome    | Cognome | Id | Nome    | Cognome |
+----+---------+---------+----+---------+---------+
| 13 | Giorgio | Conte   | 21 | Paolo   | Conte   |
+----+---------+---------+----+---------+---------+
```

Associazioni simmetriche (cont.)

- Se inserisco **una singola ennupla per coppia di fratelli** ...
  - bisogna fare attenzione e complicare le query.
    Es. "i fratelli di Paolo Conte (id=21)" non si realizza con

```
SELECT p.*
FROM   Persone p, Fratelli f
WHERE  f.Id1='21' AND p.Id=f.Id2
```

  - ma invece

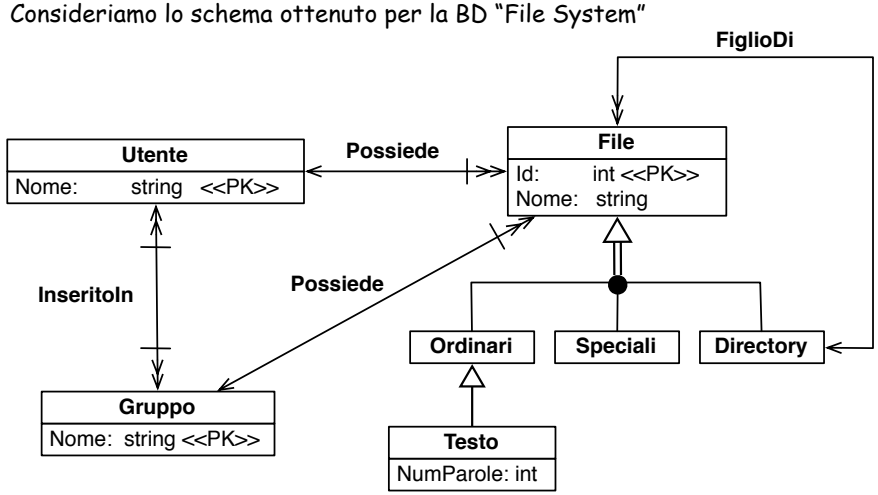
```
SELECT p.*
FROM   Persone p, Fratelli f
WHERE  (f.Id1='21' AND p.Id=f.Id2) OR
       (f.Id2='21' AND p.Id=f.Id1)
```


Associazioni simmetriche (cont.)

- Problema del modello relazionale ...
  - non ha soluzione ovvia.
  - molti preferirebbero la seconda soluzione perché priva di ridondanze.

Sottoclassi senza attributi propri

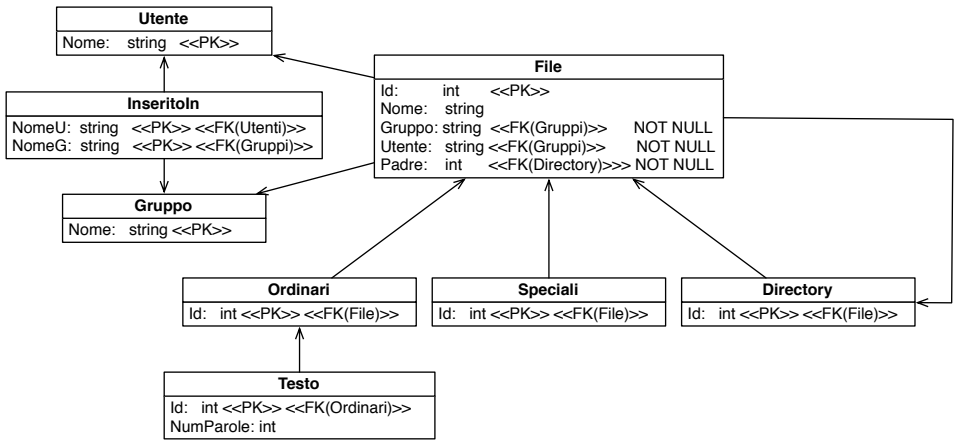
- Consideriamo lo schema ottenuto per la BD "File System"



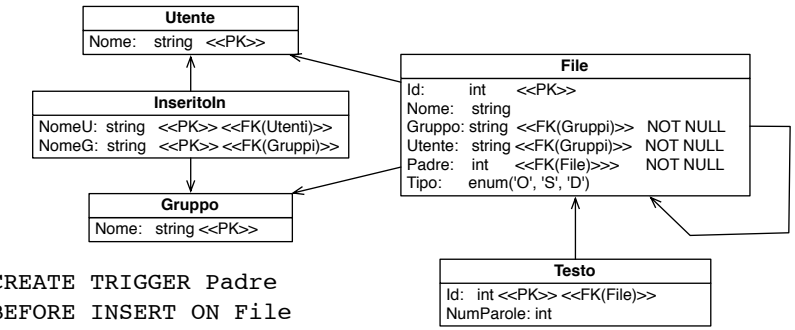
Sottoclassi senza attributi propri (cont.)

- Può essere sensato rappresentare le sottoclassi con partizionamento verticale, anche se prive di attributi propri?
  - Si, per rappresentare nello schema il fatto che un file ha come padre una directory
  - Non cattura il vincolo di disgiunzione
- Alternativamente, tabella unica, con discriminatore
  - Vincolo padre in directory espresso come vincolo semantico (trigger?)
  - Cattura il vincolo di disgiunzione

Sottoclassi senza attributi propri



Sottoclassi senza attributi propri

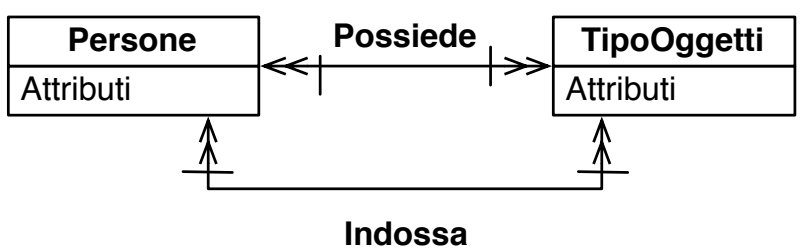


-
```
CREATE TRIGGER Padre
BEFORE INSERT ON File
FOR EACH ROW
BEGIN
  IF ((SELECT f.Tipo FROM File f WHERE f.Id=NEW.Padre)<>'D')
  THEN ERROR
  END IF;
END;
```

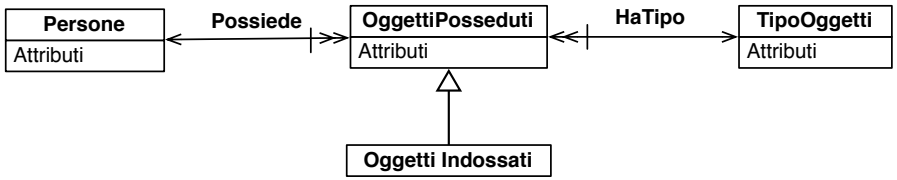

Sotto-associazioni?

- Si consideri la seguente descrizione:
  - Si vogliono rappresentare un insieme di personaggi
  - Ogni personaggio può possedere degli oggetti.
  - Alcuni degli oggetti posseduti possono essere indossati
  - Non si possono indossare due oggetti dello stesso tipo (es. due cappelli).

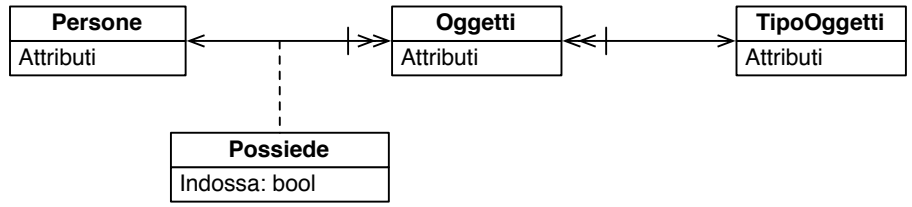
Soluzione 1



Soluzione 2



Soluzione 3



Wednesday, May 1, 2013