

Basi di Dati - VII

Corso di Laurea in Informatica
Anno Accademico 2022/2023

Alessandra Raffaetà
raffaeta@unive.it

Il DDL di SQL

- 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)

- 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**)

- 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`

- Uno schema può essere creato con:

CREATE SCHEMA Università **AUTHORIZATION** rossi

- Uno schema può essere eliminato mediante un comando

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

- Esempio

DROP SCHEMA Università **CASCADE**

- 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)

- 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 ( VALUE <= 30 AND VALUE >= 18 )
```


- SQL supporta un certo numero di tipi di dato atomici; i principali sono
 - tipi **interi**:
 - **INTEGER, SMALLINT ...**
 - valori **decimali**:
 - **NUMERIC(p, s)**
 - **virgola mobile**:
 - **REAL**
 - **stringhe di bit**:
 - **BIT(x), BIT VARYING(x)**
 - **booleani**:
 - **BOOLEAN**

- **stringhe** di caratteri:
 - **CHAR(x)** (o **CHARACTER(x)**)
 - **VARCHAR(x)** (o **CHAR VARYING(x)** o **CHARACTER VARYING(x)**)
- **date e ore**:
 - **DATE, TIME, TIMESTAMP**
- **intervalli temporali**:
 - **INTERVAL {YEAR, MONTH, DAY, HOUR, MINUTE, SECOND}**

- **SERIAL** serve per creare una colonna con un identificatore unico - simile a `AUTO_INCREMENT`.

```
CREATE TABLE tablename (  
    colname SERIAL,  
    name VARCHAR(10),  
    ...  
);
```


equivalente a

```
CREATE SEQUENCE tablename_colname_seq;  
CREATE TABLE tablename (  
    colname integer NOT NULL DEFAULT nextval('tablename_colname_seq')  
    ...  
);  
ALTER SEQUENCE tablename_colname_seq OWNED BY tablename.colname;
```

- 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);
 - un eventuale **vincolo**; e.g. NOT NULL, CHECK (<CONDIZIONE>)

```
CREATE TABLE Studenti (  
    Nome          VARCHAR(10) NOT NULL,  
    Cognome       VARCHAR(10) NOT NULL,  
    Sesso         CHAR(1) CHECK(Sesso IN ('M', 'F')),  
    Matricola     CHAR(6),  
    Nascita       DATE,  
    Provincia     CHAR(2) DEFAULT 'VE',  
    Tutor         CHAR(6)  
);
```

- 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);
 - **CHECK**: specifica un'espressione che produce un valore booleano.
 - interrelazionali
 - **FOREIGN KEY**: designa
 - un insieme di attributi come chiave esterna
 - un'eventuale azione da intraprendere (**NO ACTION**, **SET NULL**, **SET DEFAULT**, **CASCADE**) 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)



Nome	Cognome	Matricola	Nascita	Provincia	Tutor
Chiara	Scuri	71346	1985	VE	71347
Giorgio	Zeri	71347	1987	VE	NULL
Paolo	Verdi	71523	1986	VE	NULL
Paolo	Poli	71576	1988	PD	71523

- 
- Cosa succede se si rimuove Giorgio Zeri?

```
CREATE TABLE Studenti (  
    Nome          VARCHAR(10) NOT NULL,  
    Cognome       VARCHAR(10) NOT NULL,  
    Matricola     CHAR(6) PRIMARY KEY,  
    Nascita       YEAR,  
    Provincia     CHAR(2) DEFAULT 'VE',  
    Tutor        CHAR(6),  
    FOREIGN KEY (Tutor) REFERENCES Studenti(Matricola)  
        ON UPDATE CASCADE  
        ON DELETE SET NULL  
);
```

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

```
CREATE TABLE Esami (  
    Codice      CHAR(4) PRIMARY KEY,  
    Materia     CHAR(3),  
    Candidato   CHAR(6) NOT NULL,  
    Data        DATE,  
    Voto        INTEGER CHECK(Voto >= 18 AND Voto <= 30),  
    Lode        CHAR(1),  
    CodDoc      CHAR(3) NOT NULL,  
    UNIQUE (Materia,Candidato),  
    FOREIGN KEY (Candidato) REFERENCES Studenti(Matricola)  
        ON UPDATE CASCADE,  
  
    FOREIGN KEY (CodDoc)      REFERENCES Docenti(CodDoc)  
        ON UPDATE CASCADE  
);
```


- 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  
    ALTER COLUMN Nazionalita TYPE VARCHAR(15);
```

- Aggiungere ed eliminare vincoli

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

```
ALTER TABLE  Studenti  
    DROP CONSTRAINT nome_vincolo
```

- E molto altro ...

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

- 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**
- **CASCADE** provoca la rimozione automatica di tutte le viste che utilizzano la tabella (o la vista) cancellata.
- **RESTRICT** non viene rimossa se la tabella (o la vista) è utilizzata in altre viste.

- **Tabelle inizializzate:**

- **CREATE TABLE** Nome [**AS**] Espressione**SELECT**

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

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

- Creazione dello storico degli esami

```
CREATE TABLE EsamiFino2006 AS  
  
    SELECT    *  
  
    FROM      Esami e  
  
    WHERE     e.Data <= '31/12/2006';
```

```
DELETE FROM Esami  
  
WHERE e.Data <= '31/12/2006';
```

- 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;
```

- 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='PD';
```

- Potrebbe diventare

```
SELECT s.Cognome, s.Matricola, AVG(e.Voto)
FROM    Studenti s JOIN Esami e ON s.Matricola=e.Candidato
WHERE    s.Provincia='PD'
GROUP BY s.Matricola, s.Cognome;
```

- 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, né funzioni di aggregazione)
 2. FROM una sola tabella modificabile
 3. GROUP BY e HAVING non sono presenti nella definizione.
 4. Non deve contenere operatori insiemistici

- 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

- Trovare la media dei voti massimi ottenuti nelle varie province
- 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;
```

- Le province dove la media dei voti degli studenti è massima.
Restituire tale/i provincia/ce e la media.

```
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 ...

```
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);
```

Cani(Cod, Nome, Razza*, Madre*, Padre*, AnnoNasc, AnnoMorte, Istruttore*)

Madre FK(Cani), Padre FK(Cani), Razza FK(Razze), Istruttore FK(Istruttori)

Dare il nome di ogni cane che ha entrambi i genitori e i loro genitori della sua stessa razza

```
CREATE VIEW GenStessaRazza(Figlio, Padre, Madre, Razza)
AS SELECT c.Cod, p.Cod, m.Cod, c.Razza
FROM Cani c JOIN Cani p ON c.Padre = p.Cod JOIN Cani m ON c.Madre = m.Cod
WHERE m.Razza = p.Razza AND m.Razza = c.Razza

SELECT c.Nome
FROM GenStessaRazza f JOIN GenStessaRazza m ON f.Madre = m.Figlio
JOIN GenStessaRazza p ON f.Padre = p.Figlio JOIN Cani c ON c.Cod = f.Figlio
```

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

```
Default := DEFAULT { valore | NULL }
```

```
VincoloAttr := [NOT] NULL | CHECK (Condition) | PRIMARY KEY
```

- Vincoli su tabella
 - VincoloTabella := **UNIQUE** (Attributo {, Attributo})
| **CHECK** (Condizione) |
| **PRIMARY KEY** (Attributo {, Attributo})
| **FOREIGN KEY** (Attributo {, Attributo})
| **REFERENCES** Tabella [(Attributo {, Attributo})]
[**ON DELETE CASCADE** | **NO ACTION** | **SET DEFAULT** | **SET NULL**]
[**ON UPDATE CASCADE** | **NO ACTION** | **SET DEFAULT** | **SET NULL**]

Considerazioni varie

- **WITH** fornisce un modo alternativo per scrivere sottoquery da utilizzare in query più complesse.

```
WITH ProvinceMedia AS (  
    SELECT s.Provincia, AVG(e.Voto) AS Media  
    FROM Studenti s JOIN Esami e ON s.Matricola=e.Candidato  
    GROUP BY s.Provincia  
)  
SELECT Provincia, Media  
FROM ProvinceMedia  
WHERE Media = (SELECT MAX(Media) FROM ProvinceMedia);
```

Può essere pensata come una **tabella temporanea** che esiste **SOLO** per questa query.

- **CASE** è un'espressione condizionale che può essere usata in qualsiasi contesto dove un'espressione è valida.

CASE WHEN condizione **THEN** risultato

[**WHEN** ...]

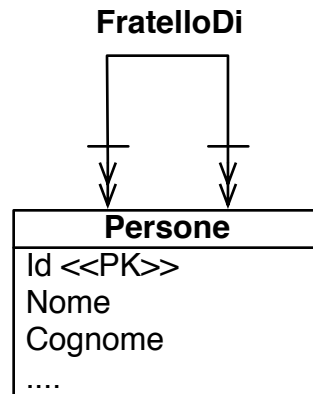
[**ELSE** risultato]

END

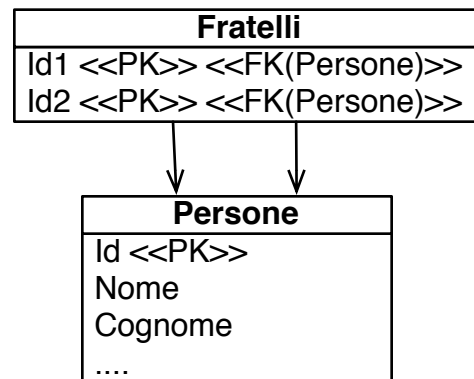
condizione è un'espressione che restituisce un valore booleano.

Se la condizione è vera, restituisce risultato e le altre condizioni non sono valutate. Se la condizione non è vera, le successive clausole **WHEN** sono esaminate in ordine. Se nessuna di queste è vera, allora restituisce il risultato dell'**ELSE**. Se l'**ELSE** non è presente, il risultato è **NULL**.

- Supponiamo di avere uno schema concettuale con una associazione simmetrica



- Questo può essere tradotto nello schema relazionale come:



- 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 ...

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

```
SELECT p1.*, p2.*
FROM   Persone p1 JOIN Fratelli f ON p1.Id = f.Id1
      JOIN Persone p2 ON f.Id2 = p2.Id
```

restituisce

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

- Devo modificare la query

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

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

- 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)
```

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

- 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 può fare senza sottoselect?

- 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;
```

- 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;
```