

Basi di Dati - VII

Corso di Laurea in Informatica
Anno Accademico 2013/2014

Alessandra Raffaetà
raffaeta@dsi.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**)

- 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

- 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, FLOAT(p)**
 - **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}**
- **testo e oggetti binari**:
 - **BLOB (BINARY LARGE OBJECT), CLOB (CHARACTER LARGE OBJECT), ...**
-

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

```
CREATE TABLE tablename (  
  
    colname SERIAL);
```

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,  
    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**:
 - 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

```
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**

● Tabelle inizializzate:

- **CREATE TABLE** Nome [**AS**] EspressioneSELECT

● **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, Esami e  
WHERE   s.Matricola=e.Candidato AND 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)
 2. FROM una sola tabella modificabile
 3. UNION non presente
 4. GROUP BY e HAVING non sono presenti nella definizione.

- 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 provincie
- Non si può fare

```
SELECT AVG(MAX(e.Voto))  
FROM Studenti s, Esami e  
WHERE s.Matricola = e.Candidato  
GROUP BY s.Provincia;
```

- Invece

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

```
SELECT AVG(Max) FROM ProvMax;
```


- 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 ...

- 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)
AS SELECT c.Cod, p.Cod, m.Cod
FROM Cani c, Cani p, Cani m
WHERE c.Padre = p.Cod AND c.Madre = m.Cod AND
      m.Razza = p.Razza AND m.Razza = c.Razza

SELECT c.Nome
FROM GenStessaRazza f, GenStessaRazza m,
      GenStessaRazza p, Cani c
WHERE f.Padre = p.Figlio AND f.Madre = m.Figlio AND
      c.Cod = f.Figlio
```

```
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 su attributi

- VincoloAttributo := **NOT NULL** | **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** | **NO ACTION** | **SET DEFAULT** | **SET NULL**]
[**ON UPDATE CASCADE** | **NO ACTION** | **SET DEFAULT** | **SET NULL**]