Basi di Dati - Linguaggio SQL

Introduzione

SQL è il linguaggio più diffuso per le Basi di Dati relazionali.

SQL è un linguaggio **dichiarativo** basato sul **Calcolo relazionale su Ennuple** e **Algebra Relazionale.**

Il linguaggio comprende:

- **DML (Data Manipulation Language):**
  ricerche e/o modifiche interattive, come l'interrogazione o le query
- **DDL (Data Definition Language)**
  definizione e amministrazione delle Basi di Dati

DML:

1) Comando SELECT:

Il comando SELECT del linguaggio SQL mi **permette di effettuare delle interrogazioni in una tabella**.
Gli attributi sono la lista delle colonne da visualizzare, separate tra loro da una virgola. Per vederle tutte basta indicare l'asterisco *.

```
SELECT attributo1, attributo2 FROM tabelle
// In questo modo vado a selezionare gli attributi delle tabelle
```

```
SELECT * FROM tabelle
// Così vado a selezionare tutti gli attributi delle tabelle senza condizioni
```

2) Comando FROM:

Indica la tabella o le tabelle in cui deve operare il comando SELECT.

3) Comando WHERE:

Indica la condizione logica con cui vengon filtrate le colonne. Tra parentesi quadre [] si può indicare un parametro da chiedere prima della selezione. Nel caso vi siano due o più tabelle selezionate le si può mettere in relazione indicando in un'eguaglianza i campi voluti.

Ad esempio, data la seguente tabella denominata 'studenti' strutturata in tre campi o colonne (nome, cognome e classe) e contenente 5 record (righe):

Tabella Studenti

nome	cognome	classe
Mario	Rossi	3
Giuseppe	Bianchi	3
Luca	Verdi	2
Mario	Russo	2
Paolo	Rossi	1

record

campi

```
SELECT *
FROM Studenti
WHERE classe='3'
// In questo modo andiamo a selezionare tutti gli studenti che hanno classe = 3
```

Mario	Rossi	3
Giuseppe	Bianchi	3

Questo sarà il risultato ottenuto.

Qualificazione: notazione con il punto e alias:

Per alias si intende l'associazione delle relazioni ad un identificatore.

Essenzialmente viene utilizzato se si sta operando su più copie della stessa relazione.

Es: Generare una tabella che contenga cognomi e matricole degli studenti e dei loro tutor:

```
SELECT s.Cognome, s.Matricola, t.Cognome, t.Matricola
FROM Studenti s, Studenti t
WHERE s.Tutor = t.Matricola
```

La clausola **AS** in SQL serve per rinominare un attributo, quindi assegnarli un alias:

```
SELECT north_east_user_subscriptions AS ne_subs // Rinominato a ne_subs
FROM users
WHERE ne_subs > 5;

// Questo alias value per la durata della query
```

Clausola DISTINCT:

La clausola **Distinct** serve a non ripetere nei risultati della SELECT quelli con lo stesso valore.
Permette di eliminare le ripetizioni dei dati uguali in una query. Si tratta di una clausola **e** non di un comando **SQL**.

```
SELECT DISTINCT City
FROM Station
WHERE ID%2=0 // Oppure MOD(ID,2)=0

// In questo caso vado a selezionare le città senza ripetizioni
```

Funzioni matematiche/aritmetiche:

Ovviamente SQL dispone di diverse funzioni che permettono di fare operazioni aritmetiche, come ad esempio la funzione SUM, MIN, MAX, COUNT, AVG.

```
// In questo modo andiamo a contare ciò che si trova all'interno di Qualcosa:
SELECT COUNT(*)
FROM Qualcosa

// Supponiamo di dover sommare tutti gli attributi Sium della tabella Qualcosa:
```

```
SELECT SUM(Sium)
FROM Qualcosa

// Troviamo anno di nascita minimo, massimo e medio degli Studenti:
SELECT MIN(Nascita), MAX(Nascita), AVG(Nascita)
FROM Studenti

// C'è anche la funzione per controllare il resto di una divisione:
MOD(Sium,2) // -> ovviamente va utilizzata in un certo modo, come nello scorso
            // paragrafo
```

Operazione JOIN:

In SQL l’operazione JOIN combina le righe di due o più tabelle in base ai valori contenuti nelle colonne.

Pensiam per esempio di avere due tabelle. Da queste due tabelle devo ricavare una tabella in cui per ciascuna persona compaia il reddito e il nome del padre.

Persone

Nome	Reddito
Mario	25000
Giovanni	15000
Paolo	30000
Giuseppe	20000
Maria	35000

Paternità

Padre	Figlio
Giovanni	Maria
Mario	Paolo
Mario	Giuseppe

```
SELECT Nome, Reddito, Padre
FROM Persone JOIN Paternità
ON Nome = Figlio // Con ON vado a dare una condizione su cui effettuare il JOIN

// Praticamente sono andato a unire li dove il nome delle persone corrisponde
// al nome del figlio nella tabella Paternità.
```

E questa è la tabella generata:

JOIN Nome=Figlio

Nome	Reddito	Padre
Paolo	30000	Mario
Giuseppe	20000	Mario
Maria	35000	Giovanni

Diversi tipi di JOIN:

Ovviamente l'operazione JOIN può essere fatta in modo diverso a seconda di ciò che viene richiesto. C'è il NATURAL JOIN, CROSS JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN:

```
// Il cross join viene utilizzato per fare una combinazione di tutte le righe
// della prima tabella con tutte le righe della seconda:
// Quindi se ho nella prima tabella A,B e nella seconda a,b otterrò:
// (A,a),(A,b),(B,a),(B,b).

// Ora utilizzeremo la consegna dello scorso paragrafo per fare esempi:

// Natural Join:
SELECT * FROM Persone NATURAL JOIN Paternità;
// Nel caso di join naturale non va specificata la condizione di join, il
// natural join infatti verifica automaticamente se esistono colonne con lo
// stesso nome nelle due tabelle.

// Left Join:
SELECT Nome, Reddito, Padre
FROM Persone LEFT JOIN Paternità
ON Nome = Figlio
// Voglio mostrare il nome, il reddito e, se presente, il nome del padre della
// persona. Quindi per non escludere le righe della prima tabella senza
// corrispondenza nella seconda utilizzo il LEFT JOIN. Ottenendo di fatto le
// corrispondenze e le non corrispondenze.

// Right Join:
// Mantiene tutte le righe della seconda tabella a cui aggiunge le righe della
// prima che soddisfano la condizione di join.

// Full Join:
// E' la combinazione del LEFT e RIGHT OUTER JOIN. Mantiene tutte le tuple di
// entrambe le tabelle, estendendole con valori nulli se necessario.
```

LEFT JOIN Nome=Figlio

Nome	Reddito	Padre
Mario	25000	
Giovanni	15000	
Paolo	30000	Mario
Giuseppe	20000	Mario
Maria	35000	Giovanni

Ecco la tabella generata dal Left Join.

Operatori insiemistici:

Clausola ORDER BY:

Esercizi SQL:

1]

Trovare il nome e cognome degli atleti italiani o svedesi che hanno vinto una medaglia d’argento nella disciplina Short track.

```
SELECT a.Nazione, a.Nome, a.Cognome, m.Anno-a.AnnoNascita AS Età
FROM Atleti a NATURAL JOIN Medaglie m
WHERE m.Anno-a.AnnoNascita=(SELECT MIN(m1.Anno-a1.AnnoNascita)
                            FROM Atleti a1 NATURAL JOIN Medaglia m1
                            WHERE a1.Nazione = a.Nazione)
```

2]

Cancellare le nazioni che non hanno mai vinto alcuna medaglia e cancellare anche gli atleti di quella nazione.

```
SELECT COUNT(*)
FROM Atleta a NATURAL JOIN Medaglie m
WHERE a.Nazione = ’Italia’ AND m.Anno=2016
```

3]

Trovare gli atleti che hanno vinto almeno una medaglia per tipo e restituire il nome e cognome, la nazione e il numero di medaglie totale vinte.

```
SELECT DISTINCT a.IdAtleta, a.Nome, a.Cognome, a.Nazione
FROM Atleti a JOIN Medaglie m1 USING(IdAtleta)
     JOIN Medaglie m2 USING (IdAtleta)
WHERE m1.Sport<>m2.Sport
```

4]

Per ogni atleta, trovare il numero di edizioni in cui ha vinto una medaglia nello Sport ”Sci di fondo” e restituire anche il nome e il cognome dell’atleta. Se l’atleta non ha vinto alcuna medaglia in tale sport, per quell’atleta si deve restituire 0.

```
SELECT a.Nome, a.Cognome, COUNT(DISTINCT m.Anno) AS NumEdizioni
FROM Atleta LEFT JOIN Medaglie m ON a.IdAtleta=m.IdAtleta AND m.Sport=’Sci’
WHERE
GROUP BY a.IdAtleta, a.Nome, a.Cognome
```

5]

Trovare gli atleti che hanno vinto meno di 5 medaglie e restituire il nome, cognome e la nazione di tali atleti.

```
SELECT
FROM Atleti a NATURAL LEFT JOIN Medaglie m
WHERE a.Nome, a.Cognome, a.Nazione
GROUP BY a.IdAtleta, a.Nome, a.Cognome, a.Nazione
HAVING COUNT(m.Codice) < 5
```

6]

Trovare nome e anno di nascita dei dipendenti del progetto Master.

```
SELECT d.Nome, d.AnnoNascita
FROM Dipendenti d NATURAL JOIN Staff s JOIN Progetti p USING(Cod)
WHERE p.Nome=’Master’
```

7]

Trovare nome e data degli esami per studenti che hanno superato l’esame di BD con 30.

```
SELECT Nome, Data
FROM Studenti JOIN Esami ON Matricola = Candidato
WHERE Materia = 'DB' AND Voto = 30
```

DDL:

SQL viene usato, oltre che per l’interrogazione delle query, anche per la definizione di basi di dati.

Creazione di uno schema:

Uno schema può essere creato con:

```
CREATE SCHEMA Università AUTHORIZATION rossi
// AUTHORIZATION mi dice chi è il proprietario dello schema
```

Uno schema può essere eliminato attraverso il costrutto drop:

```
DROP SCHEMA Nome [ CASCADE | RESTRICT ]

// Di default se non scriviamo nulla viene lasciato RESTRICT:
DROP SCHEMA NomeDelloSchema RESTRICT

// Per realizzare il dropschema forzato si una CASCADE:
DROP SCHEMA NomeDelloSchema CASCADE
```

Schemi:

Uno schema può contenere varie tabelle delle quali esistono più tipi:

- Tabelle base (base table)
  - ▼ i metadati appartengono allo schema
  - ▼ i dati fisicamente memorizzati
- Viste
  - ▼ i metadati sono presenti nello schema
  - ▼ i dati non sono fisicamente memorizzati

Creazione di una Tabella:

Una tabella base viene creata con il comando **CREATE TABLE**, è un insieme di colonne/attributi per ciascuna delle quali va specificato:

- nome
- tipo di dato, quindi i valori che possono essere assunti
  - ▼ predefinito
  - ▼ definito dall'utente (dominio), costruito con il comando **CREATE DOMAIN**

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

// Come valore di valutazione posso assumere soltanto un intero
// compreso tra 18 e 30
```

Tipi di dato predefiniti:

Ce ne sono tanti, i più importanti sono:

- tipi **interi:**
  - ▼ **INTEGER, SMALLINT…**
- **valori decimali:**
  - ▼ **NUMERIC(p,s)**
- **valori in virgola mobile:**

    - ▼ **REAL**
- **stringhe di bit:**
    - ▼ **BIT(x), BIT VARYING(x)**
- **booleani:**
    - ▼ **BOOLEAN**
- **stringhe** di caratteri:
    - ▼ **CHAR(x)** oppure **CHARACTER(x)**
    - ▼ **VARCHAR(x)** oppure **CHAR VARYING(x) / CHARACTER VARYING(x)**
- **date** e **ore:**
    - ▼ **DATE, TIME, TIMESTAMP**
- **intervalli temporali:**
    - ▼ **INTERVAL{YEAR, MONTH, DAY, HOUR, MINUTE, SECOND}**

Tipi di dato:

- **SERIAL** serve per scrivere ID sintetici, spesso è utile avere un attributo che genera identificatori. In questo caso SERIAL genera interi. Quando andiamo a definire una colonna SERIAL, creiamo una nuova tabella SEQUENZA, questa nuova tabella è costituita da un’unica riga e contiene il valore appena assegnato.

```
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;
```

Vincoli di ennupla: