

CAP4_SQL

Table of contents

- [Concetti base](#)
- [Interfaccia grafica](#)
- [Data Manipulation Language \(DML\)](#)
 - [Modifica degli schemi](#)
- [Data Definition Language \(DDL\)](#)
 - [`SELECT`](#)
 - [`CREATE TABLE`](#)
- [Esempi DDL e DML](#)
- [Estratti da esami passati](#)
 - [Prova itinere 11-2022](#)
 - [Tema_A](#)
 - [Tema_B](#)

Structured Query Language (SQL)

[#sql](#) [#pratica](#) [#comandi-sql](#) [#dml](#) [#ddl](#) [#transazione](#) [#sicurezza](#)

Concetti base

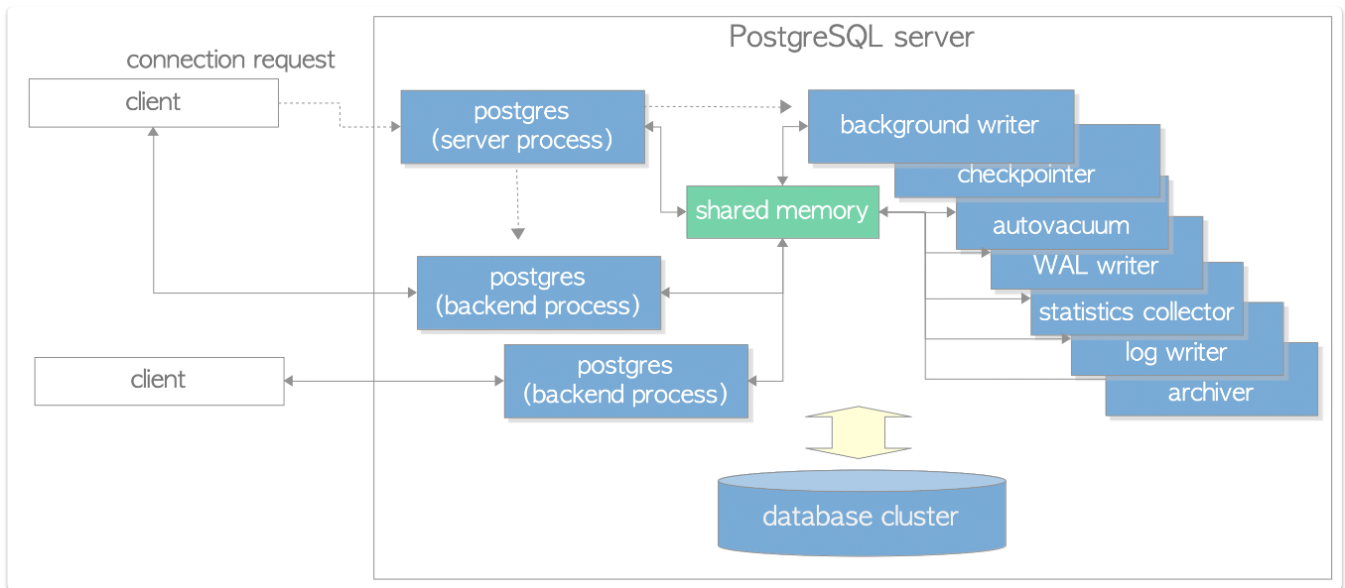
Linguaggio che contiene sia DML (che lavora solo su istanza) sia DDL.

Prima di parlare del linguaggio in sè, osserviamo i processi in esecuzione all'avvio del DBMS.

Per `psql`, parlando del comando, esistono diversi *worker* all'interno della nostra macchina linux che vengono avviati all'istanza. Il seguente comando li mostra in command line.

```
ps aux | grep mysql
```

- Il nostro server si prende la libertà di memorizzare le pagine all'interno della RAM, ma si occupa anche di possibili fallimenti, il *logging collector* è uno dei processi;
- *stats collector* ogni tanto lancia (possiamo farlo manualmente) per acquisire info sul database;
- *autovacuum launcher*, nelle mie tabelle ogni tanto faccio eliminazioni che sono logiche (alcune n -uple diventano invalide), aiuta a recuperare spazio liberato compattando le tabelle;
- *logical replication launcher* per replicare (come altro server) i contenuti del server corrente (master & slave);
- il *background writer* scrive pagine nella shared memory lentamente verso la memoria persistente;
- *WAL writer* che trasferisce dati WAL su memoria persistente;
- *archiver* che archivia il log eseguito.



Interfaccia grafica

⚠ L'interfaccia grafica a noi non interessa troppo siccome tutto quello che serve e' gia' incluso in linea di comando.

Per la GUI usiamo `pgAdmin`.

Nell'interfaccia grafica possiamo fare JOIN tra *schemi* (collezione di tabelle), possiamo vedere gli utenti con accesso al DB, possiamo vedere le relazioni. Le operazioni sono molteplici ma equivalgono alle stesse operazioni possibili tramite linea di comando (useremo soltanto da linea di comando).

Data Manipulation Language (DML)

Modifica degli schemi

Usiamo `INSERT`, `DELETE`, `UPDATE`, `ALTER` da una sola tabella per 0, 1, n istanze, sulla base di una condizione coinvolgente anche altre relazioni.

Per inserire n -uple:

```
INSERT INTO nomeTabella(colonna1, colonna2, ...)
VALUES (valore1, valore2, ...);
```

oppure

```
INSERT INTO nomeTabella(nomeColonne)
SELECT ();
```

Per cancellare n -uple:

```
DELETE FROM nomeTabella
WHERE condizione; -- senza questa, la tabella verrebbe svuotata
```

Se la politica di reazione per vincoli referenziali è specificata `CASCADE`, allora istanze di altre tabelle correlate vengono eliminate; significa che tutte le righe legate con chiave esterna, n -uple soddisfacenti il predicato della `DELETE`, verranno eliminate.

Per modificare n -uple:

```
UPDATE nomeTabella
SET nomeColonna = [ espressione | SELECT | NULL | DEFAULT | ... ]
```

```
WHERE condizione;
```

Per alterare tabelle:

```
ALTER TABLE nomeTabella  
[ ADD | DROP | ALTER ] nomeColonna | nomeConstraint;
```

Data Definition Language (DDL)

SELECT

[SELECT](#) → vedere PDF associato

CREATE TABLE

Per creare porzioni di schema usiamo l'istruzione `CREATE TABLE`:

- definisce uno schema di relazione e ne crea un'istanza vuota;
- specifica attributi, domini e vincoli.

```
CREATE TABLE [ IF NOT EXISTS ] nomeTabella (  
    nomeColonna dominio [ CONSTRAINT ],  
    ...  
);
```

Ci è possibile creare tabelle anche al risultato di un'espressione.

Domini elementari

- di *carattere*, singoli caratteri o stringhe;
- *numerici* esatti e approssimati;
- *data*, *ora*, *intervalli di tempo*;
- *boolean*, scritto per esteso
- *BLOB*(binary long object), *CLOB*(character long object)

Definiamo dei tipi di dato semplice con `CREATE DOMAIN`.

Ci permette di portarci dietro i vincoli ogni qual'ora ci serve scrivere lo stesso dato in più tabelle.

```
CREATE DOMAIN nomeDominio  
AS dominio [ CONSTRAINT ... ];
```

Vincoli Intrarelazionali

Chiamiamo un vincolo *intrarelazionale*, un vincolo riferito agli attributi della tabella su cui stiamo lavorando.

- `NOT NULL`, un attributo non può essere nullo;
- `UNIQUE` per creare una chiave, significa "ce ne può essere uno solo";
- `PRIMARY KEY` la chiave primaria (una soltanto, implica `NOT NULL` ma possiamo scriverlo per chiarezza);
- `CHECK` per vincolo di *n*-upla, sarebbe un *constraint*.

Differenze tra `UNIQUE` e `PRIMARY KEY`

`UNIQUE` e `PRIMARY KEY` differiscono:

- chiavi `UNIQUE` possono essere definite in diverso numero, mentre di `PRIMARY KEY`, associata o meno a più attributi, ne può esistere una soltanto;
- i valori `NULL` in chiavi `UNIQUE` possono esistere, per la `PRIMARY KEY` non sono ammessi.

```
CREATE nomeTabella (
    -- un caso di creazione chiave primaria
    colonna1 dominio PRIMARY KEY,
    colonna2 dominio,
    colonna3 dominio,
    -- due attributi insieme formano una chiave
    UNIQUE(colonna2,colonna3),
    -- caso alternativo di creazione chiave primaria
    colonna1 dominio,
    PRIMARY KEY (colonna1)
);
```

```
CREATE TABLE persone (
    ...
    sesso CHAR(1) NOT NULL CHECK (sesso IN ('M','F')),
    ...
);
```

Vincoli Interrelazionali

Riguardano la tabella che abbiamo in oggetto, ma si riferiscono anche ad altre tabelle.

Serve per indicare che un *attributo* della tabella corrente, fa *riferimento* a un altro attributo di un'altra tabella, e che quindi non c'è bisogno di riscrivere siccome già presente.

- CHECK;
- REFERENCES e FOREIGN KEY per definire vincoli d'integrità referenziale; sono due sintassi per scrivere la stessa cosa
 - per singoli attributi
 - su più attributi
- è possibile definire politiche di reazione alla violazione del vincolo imposto.

```
CREATE TABLE nomeTabella (
    -- primo modo per creare vincolo esterno
    colonna1 dominio
        REFERENCES altraTabella(colonna),
    colonna2 dominio,
    colonna3 dominio,
    -- modo alternativo di vincolo esterno
    FOREIGN KEY (colonna2,colonna3)
        REFERENCES altraTabella(colonna, colonna)
);
```

Per condizioni complicate, per vincoli, invece che CHECK usiamo ASSERTION.

In `psql` non sono supportate, perché non ci è garantita la loro efficienza.

La vista creata con CREATE VIEW NomeVista.

Sono come normali relazioni.

Esempi DDL e DML

```
-- esempio di tabella con chiavi
CREATE TABLE Impiegati (
    matricola CHAR(6) PRIMARY KEY,
    nome VARCHAR(20) NOT NULL,
    cognome VARCHAR(20) NOT NULL,
    dipart VARCHAR(30),
```

```

    stipendio NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(dipart)
        REFERENCES Dipartimenti(nomeDipart),
    UNIQUE (cognome, nome)
);

```

```

-- creo un dominio da riutilizzare in piu' tabelle se mi serve
CREATE DOMAIN voto
    AS SMALLINT DEFAULT NULL,
    CHECK (value ≥ 18 AND value ≤ 30)

```

```

-- il numero di figli di ciascun padre
SELECT Padre, COUNT(*) AS NumFigli
FROM paternita
GROUP BY Padre

```

```

INSERT INTO matricole
VALUES ('Marco', 'Rondelli', '306706');

```

```

DELETE FROM paternita
WHERE figlio NOT IN (SELECT nome
                     FROM persone);

```

```

UPDATE persone
SET reddito = reddito * 1.1
WHERE eta < 30

```

Estratti da esami passati

Prova itinere 11-2022

- Fornire una istanza della tabella $R(A, B)$ per la quale la query `SELECT COUNT(A), COUNT(B) FROM R` calcola due valori diversi.

A	B
mario	rossi
NULL	neri

- Data la relazione $R(A, B, C)$, la query `SELECT COUNT(*), B*C AS PROD FROM R ORDER BY B*C` è errata; per quale motivo? Come deve essere corretta?

Vedere file [SELECT](#)

E' errata perché la query utilizza una funzione aggregata (`COUNT`) senza l'operatore `GROUP BY` per raccogliere i risultati.

```

SELECT COUNT(*), B*C AS PROD
FROM R
GROUP BY B*C
ORDER BY B*C

```

Biblioteche (codice, nome, citta, indirizzo)

Libri (codice, titolo, edizione, anno, pagine)

Autori (codice, nome, cognome, anno_nascita, biografia)

Autori_libri (libro_{fk}, autore_{fk})

Copie_libri (seriale, libro_{fk}, biblioteca_{fk}, collocazione)

Prestiti (codice, data_inizio, data_fine_prevista, data_fine_effettiva*, copia_libro_{fk})

1. Scrivere l'istruzione DDL per la definizione della relazione autori_libri includendo, oltre ai vincoli indicati nello schema, il vincolo che impone che per ogni libro non vi possano essere più autori con la stessa posizione (ordine sequenza) nella sequenza degli autori.

```
CREATE TABLE autore_libri (  
    libro INTEGER NOT NULL,  
    autore INTEGER NOT NULL,  
    ordine_sequenza INTEGER NOT NULL,  
    UNIQUE(libro, autore, ordine_sequenza),  
    PRIMARY KEY (libro, autore),  
    FOREIGN KEY (libro) REFERENCES libri(codice),  
    FOREIGN KEY (autore) REFERENCES autori(codice)  
);
```

2. Definire la vista relazionale libri_con_prestiti_scaduti(codice_libro, titolo) che elenca i codici e i titoli dei libri per i quali esiste almeno un prestito in corso la cui data prevista di restituzione è precedente alla data odierna.

```
CREATE OR REPLACE VIEW libri_con_prestiti_scaduti AS  
SELECT DISTINCT l.codice AS codice_libro, l.titolo AS titolo  
FROM prestiti p, copie_libri cl, libri l  
WHERE p.data_fine_prevista < CURRENT_DATE  
    AND p.copia_libro = cl.seriale  
    AND cl.libro = l.codice
```

3. Modificare i prestiti in corso per le copie di libri della biblioteca di nome "Biblioteca Pavese" di Parma, spostando in avanti di 30 giorni la data fine prevista.

```
UPDATE prestiti  
SET data_fine_prevista = data_fine_prevista + 30  
WHERE copia_libro IN (  
    SELECT cl.seriale  
    FROM prestiti p, copie_libri cl, biblioteche b, libri l  
    WHERE p.copia_libro = cl.seriale  
        AND cl.biblioteca = b.codice  
        AND cl.libro = l.codice  
        AND b.nome = 'Biblioteca Pavese'  
        AND b.citta = 'Parma'  
        AND data_fine_prevista > current_date  
);
```

4. Per ogni città e per ogni autore, calcolare il numero di prestiti registrati, dall'inizio del 2015 alla fine del 2019, in una biblioteca di quella città e che hanno riguardato (una copia di) un libro di quell'autore.

```
SELECT b.citta, a.cognome, COUNT(p.codice) AS prestiti_redistrati  
FROM biblioteche b, autori a, prestiti p,  
    copie_libri cl, libri l, autori_libri al  
WHERE p.data_inizio ≥ '2015-01-01'
```

```

AND p.data_inizio ≤ '2019-12-31'
AND p.copia_libro = cl.seriale
AND cl.libro = l.codice
AND cl.biblioteca = b.codice
AND al.libro = l.codice
AND al.autore = a.codice
GROUP BY b.citta, a.cognome
ORDER BY b.citta, a.cognome;

```

5. Modificare lo schema della tabella Prestiti, aggiungendo il vincolo di integrità che impedisce di avere una data inizio superiore alla data fine prevista e alla data fine effettiva.

```

ALTER TABLE prestiti
ADD CONSTRAINT valid_date
CHECK (
    data_inizio < data_fine_prevista
    AND
    data_inizio < data_fine_effettiva
);

```

Tema_B

Biblioteche (codice, nome, citta, indirizzo)

Libri (codice, titolo, edizione, anno, pagine)

Autori (codice, nome, cognome, anno_nascita, biografia)

Autori_libri (libro_{fk}, autore_{fk})

Copie_libri (seriale, libro_{fk}, biblioteca_{fk}, collocazione)

Prestiti (codice, data_inizio, data_fine_prevista, data_fine_effettiva*, copia_libro_{fk})

1. Scrivere l'istruzione DDL per la definizione della relazione copie libri includendo, oltre ai vincoli indicati nello schema, il vincolo che impone che ogni biblioteca non possa avere più copie dello stesso libro.

```

CREATE TABLE copie_libri (
    seriale SERIAL NOT NULL,
    libro INTEGER NOT NULL,
    biblioteca INTEGER NOT NULL,
    collocazione INTEGER NOT NULL,
    PRIMARY KEY (seriale),
    FOREIGN KEY (libro) REFERENCES libri(codice),
    FOREIGN KEY (biblioteca) REFERENCES biblioteche(codice),
    UNIQUE(libro, biblioteca)
);

```

2. Definire la vista relazionale autori ignorati(codice, cognome, nome) che elenca gli autori per i cui libri non sono stati registrati prestiti nel corso dell'anno 2021 (considerare la data d'inizio del prestito).

```

CREATE OR REPLACE VIEW autori_ignorati AS
SELECT a.codice, a.cognome, a.nome
FROM prestiti p, copie_libri cl, autori_libri al, autori a
WHERE p.copia_libro = cl.seriale
    AND cl.libro = al.libro
    AND al.autore = a.codice
    AND a.codice NOT IN (
        SELECT a.codice
        FROM prestiti p, copie_libri cl, autori_libri al, autori a

```

```

        WHERE p.data_inizio ≥ '2021-01-01'
              AND p.data_inizio ≤ '2021-12-31'
              AND p.copia_libro = cl.seriale
              AND cl.libro = al.libro
              AND al.autore = a.codice
    );

```

3. Eliminare i libri per i quali non sono presenti copie nelle biblioteche.

```

DELETE FROM libri
WHERE codice NOT IN (
    SELECT l.codice
    FROM copie_libri cl, libri l, biblioteche b
    WHERE cl.libro = l.codice
          AND cl.biblioteca = b.codice
);

```

4. Per ogni biblioteca e per ogni autore, calcolare il numero di copie di libri di quell'autore presenti nella biblioteca (nota: una copia si considera presente anche se è al momento in prestito).

```

SELECT b.nome AS biblioteca, a.cognome AS autore, COUNT(cl.seriale)
FROM copie_libri cl, autori_libri al, biblioteche b, autori a
WHERE cl.libro = al.libro
      AND cl.biblioteca = b.codice
      AND al.autore = a.codice
GROUP BY b.nome, a.cognome
ORDER BY b.nome, a.cognome

```

5. Modificare lo schema della tabella Prestiti, aggiungendo il vincolo di integrità che impedisce di avere una data inizio superiore alla data fine prevista e alla data fine effettiva.

```

ALTER TABLE prestiti
ADD CONSTRAINT valid_date
CHECK (
    data_inizio < data_fine_prevista
    AND
    data_inizio < data_fine_effettiva
);

```