CAP4_SQL

Table of contents:

Structured Query Language (SQL)

#sql #pratica #comandi-sql #dml #ddl #transazione #sicurezza

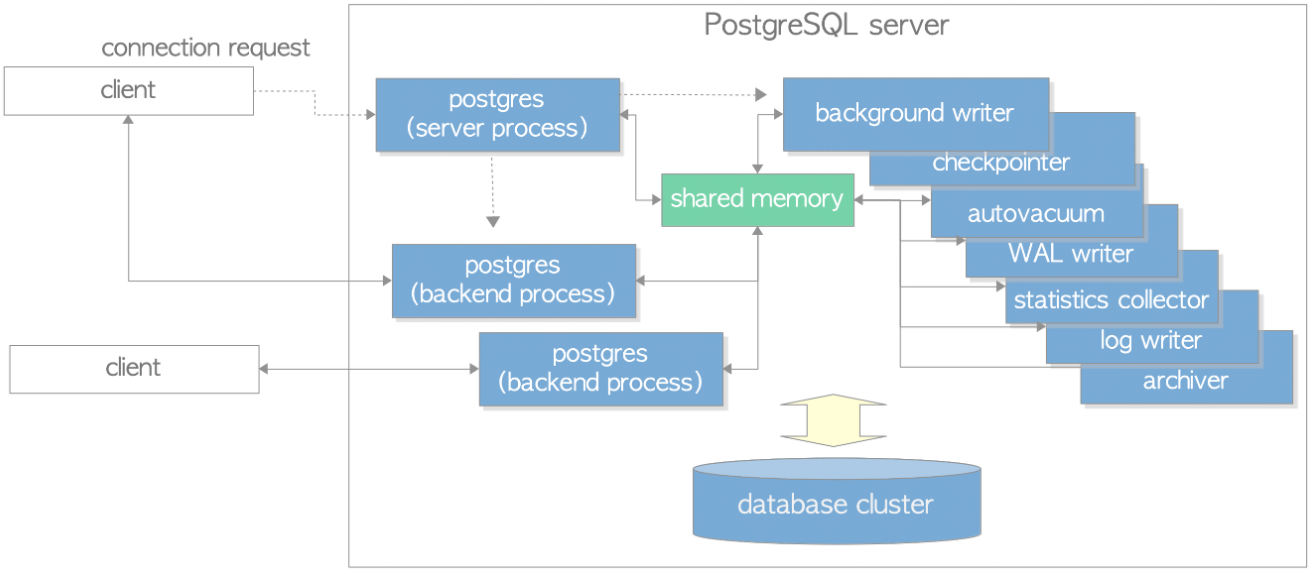
Concetti base

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

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).

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

Data Manipulation Language (DML)

Modifica degli schemi

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

Per inserire n-uple nello schema:

```
INSERT INTO nomeTabella(attributo1, attributo2, ...)
        VALUES (valore1, valore2, ...);
```

oppure

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

Per cancellarle:

```
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 modificarle:

```
UPDATE nomeTabella
SET attributo = [ espressione | SELECT (); | NULL | DEFAULT | ... ]
```

```
        WHERE condizione;
```

Transazione

> **Questa voce dell'argomento verra' ripresa nei capitoli successivi: non contenuta negli argomenti della prova in itinere.**

Operazioni considerate indivisibili, corrette anche in presenza di concorrenza con effetti definitivi.
Applichiamo la proprietà **ACID**:

- Atomicità, una sola, unico oggetto, se fallisce una parte allora fallisce l'intero tentativo;
- Consistenza;
- Isolamento, transazioni iniziate nel passato e che termineranno nel futuro, non vedo mai la transazione diversa dalla mia;
- Durabilità.

```
-- per cominciare la transazione
START TRANSACTION;
-- START TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- applica la proprietà di serializzazione della teoria

-- operazioni da eseguire sulla base di dati
COMMIT WORK;
```

> ⚠ **Esempio di uso di transazioni**
>
> Leggiamo dati di una esecuzione parziale: dirty read.
> Esempio in cui un utente comincia una transazione e decide poi di fare *rollback*.
>
> Avviene invece un'anomalia di concorrenza, una *non-repeatable read* quando la proprietà di isolamento, che è quella di serializzabilità di transazioni (costoso per il sistema ad essere garantita sempre), viene violata.

Data Definition Language (DDL)

`SELECT`

SQL_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.

Controllo dell'accesso

Controllare l'accesso di chi ha permesso per DDL.
In tutti i sistemi c'è un **amministratore** che nel caso di `psql` si chiama `postgres`. Sarebbe come l'account `root`, e può fare qualsiasi cosa sul DB. Per sicurezza è sempre meglio creare un utente che abbia privilegi minimi necessari per il suo lavoro.

L'utente amministratore `_system` può dare **privilegi** ad altri utenti:

- risorsa di riferimento;
- utente concedente la risorsa;
- l'utente ricevente la risorsa;
- l'azione che viene permessa;
- trasmissibilità del privilegio.

Tipi di privilegi:

- `INSERT` inserire dati
- `UPDATE` permette modifica del contenuto
- `DELETE` eliminare oggetti
- `SELECT` permette risorsa
- `REFERENCES` definizione di vincoli d'integrità referenziale, diritto se l'utente vuole creare chiave esterna
- `USAGE` utilizzo in una definizione

```
-- concessione
GRANT < Privileges | All privileges > on Resource
        TO Users [ WITH GRANT OPTION ]
        -- per specificare se altri utenti possono trasmettere il privilegio
```

```
-- revoca
REVOKE Privileges ON Resource FROM Users
        [ RESTRICT | CASCADE ]
        -- per specificare per altri utenti a cui trasmessi i privilegi
```

Esempi DDL e DML

```sql
-- 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)
);
```

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

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

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

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

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

---

lezione: 2022-10-21