

CAP05_SQL_EVOLUTO

Table of contents

- [Vincoli in SQL](#)
 1. [Vincoli d'integrita' generici](#)
 1. [CHECK](#)
 2. [ASSERTION](#)
- [Viste](#)
- [Controllo dell'accesso](#)
- [Transazioni](#)
 1. [Transaction Isolation Levels](#)
 1. [Anomalie sui livelli](#)

Vincoli in SQL

I tipi di dato (domini) sono un modo per dividere il numero di rappresentazioni che il nostro dato puo' assumere. Tuttavia nella vita reale sono necessari *vincoli* piu' forti: non esiste in SQL un modo per esprimere un dato in riguardo a numeri positivi (improvvisamente la nostra confezione di pasta vale -3€).

Vincoli d'integrita' generici

CHECK

Una n -upla all'interno della nostra tabella deve rispettare un'espressione booleana (`TRUE`) perche' questa possa essere inserita.

```
-- sintassi generica
CHECK ([ condizione ])

-- sintassi con nome
CONSTRAINT nomeVariabile CHECK ([ condizione ])
```

☰ DDL la pasta non ha piu' valore negativo

```
CREATE TABLE prodotti (
    numero INTEGER PRIMARY KEY,
    nome VARCHAR(100),
    -- sintassi generica
    prezzo NUMERIC CHECK (prezzo > 0)
    -- sintassi con nome
    prezzo NUMERIC CONSTRAINT prezzoPositivo CHECK (prezzo > 0)
);
```

`CHECK` non puo' assicurare la correttezza se chiamato su un'altra tabella diversa da quella



su cui il vincolo viene imposto.

Questo per il susseguirsi di cambiamenti non garantiti che si propagherebbero se la tabella a cui facciamo riferimento col `CHECK` subisse cambiamenti.

```
CREATE TABLE Impiegati (  
    -- ...  
    stipendio INTEGER CHECK (stipendio > 0),  
    superiore INTEGER,  
    CHECK (stipendio ≤ (SELECT stipendio  
                                FROM Impiegati J  
                                WHERE superiore = J.matricola))  
);
```

ASSERTION

⚡ Le `ASSERTION` non sono implementate in PostgreSQL, ma possiamo emularle utilizzando `TRIGGER` con `CHECK` per controllare prima di ogni inserimento se il vincolo è rispettato.

I vincoli possono essere specificati a livello di schema.

Nel caso in cui volessimo assicurarci che una condizione venga soddisfatta su un'intera tabella, prima ancora di prendere atto nel modificarla, utilizziamo un'asserzione.

```
-- sintassi di creazione asserzione  
CREATE ASSERTION assName CHECK ([ condizione ])  
  
-- esempio 'ci deve essere almeno 1 impiegato'  
CREATE ASSERTION almenoUno  
    CHECK (1 ≤ (SELECT COUNT(*)  
                    FROM Impiegati));
```

Viste

Una *vista* è il prodotto di una query che abbiamo eseguito.

Non è materializzata, bensì viene rieseguita ogni qual volta viene chiamata in una query.

Usiamo le viste per convenienza e ordine: nel caso ci interessassero dati precisi di una tabella, ma non volessimo farne riferimento nell'insieme (magari ci sono milioni di n -uple da controllare), creiamo una vista con solo i dati che vogliamo.

```
CREATE [ OR REPLACE ] VIEW name [ ( column_name [, ...] ) ] AS query  
[ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

≡ VIEW **dove gli impiegati lavorano nel dipartimento amministrativo**

```
CREATE VIEW impiegatiAmministrazione (nome, cognome, stipendio) AS  
    SELECT nome, cognome, stipendio  
    FROM impiegato  
    WHERE dipart = 'amministrazione' AND  
           stipendio > 10
```

Sulle viste possiamo:

- interrogare come se fossero relazioni di base;
- aggiornare, ma soltanto su viste di una sola relazione (niente nidificazioni);
- imporre verifiche (`CHECK`).

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
```

Transazioni

🕒 Questa voce dell'argomento verterà ripresa nei capitoli successivi: non contenuta negli argomenti della prova in itinere.

Operazioni considerate indivisibili, corrette anche in presenza di concorrenza con effetti definitivi:

- 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;
```

Transaction Isolation Levels

Anomalie sui livelli

- dirty read
Una transazione legge dati scritti da una transazione concorrente non ancora committed
- nonrepeatable read
Una transazione rilegge dati precedentemente letti e trova che i dati sono stati modificati da un'altra
- phantom read
Una transazione riesegue una query che soddisfa una condizione di ricerca e trova che la condizione è stata modificata causa un'altra transazione
- serialization anomaly
Il risultato di una transazione o gruppo, è inconsistente rispetto alle transazioni eseguite una a una

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, not in <code>psql</code>	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, not in <code>psql</code>	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

In `psql` possiamo richiedere uno dei quattro *isolamenti di livello*, anche se ne vengono implementati solo 3 (Read uncommitted = Read Committed).