Appunti Basi di Dati

Luca Seggiani

16 Maggio 2024

Regole aziendali

Le regole aziendali (*business rule*) sono determinate regole della realtà d'interesse da modelizzare nel DBMS. Il meccanismo dei trigger fornisce un modo per assicurarsi che tali regole vengano rispettate. Poniamo ad esempio la regola aziendale: "Ogni mese, le visite mutuate di un medico non possono essere più di 10".

```
DROP TRIGGER IF EXISTS blocca_non_mutuate;
DELIMITER $$
CREATE TRIGGER blocca_non_mutuate
BEFORE INSERT ON Visite
FOR EACH ROW
BEGIN

  DECLARE non_mutuate_mese INTEGER DEFAULT 0;

  SET non_mutuate_mese = (
    SELECT COUNT(*)
    FROM Visita V
    WHERE M.Medico = NEW.Medico
      AND MONTH(V.Data) = MONTH(CURRENT_DATE)
      AND YEAR(V.Data) = YEAR(CURRENT_DATE)
      AND V.Mutuata = 1
  );
  IF non_mutuate_mese = 10 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Visita non inseribile';
  END IF;

END $$
DELIMITER ;
```

Il comando "SIGNAL SQLSTATE '45000'" serve a lanciare un'errore (segnale), di cui impostiamo subito dopo il messaggio d'errore, e impedire l'inserimento. Nello specifico, il codice 45000 sta per "errore generico". In ogni caso, l'*exception handling* è al di fuori degli argomenti del corso.

Event

Gli **event** sono procedure simili ai trigger, ma la loro causa scatenante è il raggiungimento di uno specifico istante temporale. Ad esempio possono essere usati per eseguire azioni periodice (giornalmente, mensilmente, ecc...). Poniamo per esempio di voler creare e mantenere giornalmente aggiornata una ridondanza nella tabella Medico contenente il totale delle visite effettuate. Potremo dire:

```
CREATE EVENT AggiornaTotaliVisite
ON SCHEDULE EVERY 1 DAY
STARTS '2023-05-25 23:55:00'
DO
  UPDATE Medico
  SET TotaleVisite = TotaleVisite +
                      (SELECT COUNT(*)
                       FROM Visita V
                       WHERE V.Medico = Matricola
                        AND V.Data = CURRENT_DATE);
```

Notiamo la parola chiave STARTS per impostare la data di inizio. Esiste inoltre la variante STARTS - ENDS per impostare anche la data di fine della schedulazione.

Attivazione dello scheduler

L'attivazione dello scheduler si ottiene impostando la varibile di sistema event_scheduler, come:

```
SET GLOBAL event_scheduler = ON;
```

1 Materialized view

Le materialized view sono viste ridondanti, cioè ricavabili dai dati nel database (detti *raw data*), che scegliamo di precalcolare. Sostanzialmente, una materialized view è il risultato precalcolato di una query (spesso formata da un join molto corposo). A differenza delle normali view (viste), la materialized view non viene calcolata ad ogni accesso, ma precalcolata periodicamente, accumulando inevitabilmente un certo scarto temporale. Si usano quando il risultato deve essere ottenuto velocemente, ma la query necessaria a calcolarla richiede molte risorse (e tempo).
La dichiarazione di una materialized view corrisponde a quella di una tabella:

```
CREATE TABLE MATERIALIZED_VIEW(
  Paziente CHAR(100) NOT NULL,
  NumVisite INT(11) NOT NULL DEFAULT 0,
  UltimaVisita DATE,
  PRIMARY KEY(Paziente)
```

```
) ENGINE = InnoDB DEFAULT CHARSET=latin1;
```

Politiche di refresh

Esistono più politiche di refresh (ricalcolo) delle materialized view:

- **Immediate:** dopo ogni evento, quindi equiparabili ad un **trigger**;
- **Deferred:** su base temporale, quindi equiparabili ad un **event**;
- **On demand:** avviate manualmente, quindi equiparabili ad una **stored procedure**.

Esistono inoltre due modalità di aggiornamento della materialized view:

- **Full refresh:** si aggiorna tutta la vista in blocco, da zero.
- **Incremental refresh:** si aggiornano solamente le componenti non più aggiornate. Può quindi essere un'aggiornamento sia totale che parziale.

Vediamo alcuni esempi:

- **Immediate refresh** (sync):
  Una vista materializzata ad aggiornamento immediato dopo ogni aggiornamento si può implementare come:

```
DELIMITER $$
DROP TRIGGER IF EXISTS immediate_refresh_visita $$
CREATE TRIGGER immediate_refresh_visita
AFTER INSERT ON Visita
FOR EACH ROW
BEGIN
  UPDATE MATERIALIZED_VIEW
  SET NumVisite = NumVisite + 1
    UltimaVisita = CURRENT_DATE
  WHERE Paziente = NEW.Paziente
END $$
```

  questo assumendo che ad ogni paziente corrisponderà un record in MATERIALIZED_VIEW, creato con il trigger:

```
DROP TRIGGER IF EXISTS immediate_refresh_paziente $$
CREATE TRIGGER immediate_refresh_paziente
AFTER INSERT ON Paziente
FOR EACH ROW
BEGIN
  INSERT INTO MATERIALIZED_VIEW(Paziente)
  VALUES(NEW.CodFiscale)
END $$
DELMITER ;
```

- **On demand refresh** (full):
  Vediamo una vista materializzata on demand (ad aggiornamento manuale, con procedura) di tipo full:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS on_demand_refresh $$
CREATE PROCEDURE on_demand_refresh
BEGIN
  TRUNCATE MATERIALIZED_VIEW;
  INSERT INTO MATERIALIZED_VIEW
  SELECT P.CodFiscale,
    IF(V.Paziente IS NOT NULL, COUNT(*), 0),
    IF(V.Paziente IS NOT NULL, MAX(V.Data), NULL)
  FROM Visita V
    RIGHT OUTER JOIN
    Paziente P ON P.CodFiscale = V.Paziente
  GROUP BY P.CodFiscale
END $$
DELMITER ;
```

- **Deferred refresh** (full):
  Vediamo infine come implementare il refresh in differita, appoggiandoci alla procedura on demand appena creata:

```
DELIMITER $$
DROP EVENT IF EXISTS deferred_refresh $$
CREATE EVENT deferred_refresh
ON SCHEDULE EVERY 1 WEEK
BEGIN
  CALL on_demand_refresh;
END $$
DELIMITER ;
```