Introduzione alle transazioni

corso di basi di dati e laboratorio

Prof. Alfio Ferrara

Anno Accademico 2020/2021

Indice

1 Transazione

1.1 Introduzione

La nozione di transazione

- L'idea di fondo di una transazione è di collocare in un'unica operazione atomica un insieme di operazioni sui dati in modo che tali operazioni siano eseguite tutte o nessuna.
- L'aspetto fondamentale di una transazione è che le operazioni interne alla transazione (e il loro esito) non sono visibili a altre transazioni.
- Quindi, se anche usa sola delle operazioni interne fallisce, nessuna delle operazioni incluse nella transazione ha effetto sullo stato della base di dati.

Utilità delle transazioni

- Mantenere corretto il comportamento della base di dati anche in presenza di singole operazioni che falliscono.

- Mantenere corretto il comportamento della basi di dati anche in presenza di operazioni concorrenti, ovvero eseguite in tempi diversi sugli stessi dati.
- Una transazione trasforma lo stato corretto di un database in un altro stato corretto del database. Durante l'esecuzione della transazione, lo stato può essere temporaneamente non corretto.
- Al termine della transazione tuttavia gli esiti possibili sono solo due: raggiungimento di un nuovo stato corretto (COMMIT) o ritorno a uno stato corretto precedente (ROLLBACK).

Esempio (1)

```
CREATE TABLE area( country CHAR(3) PRIMARY KEY REFERENCES country(iso3),
sqkm DOUBLE PRECISION);

INSERT INTO area SELECT country, value FROM measure WHERE label = 'Area
(sq km)';

SELECT country, sqkm FROM area WHERE country IN ('ITA', 'FRA');
```

AREA

country	sqkm
FRA	643801.0
ITA	301340.0

Supponiamo che la Francia e l'Italia decidano di cedere 500 kilometri quadrati del proprio territorio a testa per creare un nuovo stato indipendente chiamato 'Freedomland (FRE)'.

Esempio (2)

```
UPDATE area SET sqkm = sqkm - 500 WHERE country = 'FRA'

UPDATE area SET sqkm = sqkm - 500 WHERE country = 'ITA'

INSERT INTO area(country, sqkm) VALUES ('FRE', 1000)
```

Qual è la situazione al temine di questi aggiornamenti?

Esempio (3)

Le operazioni precedenti danno esito:

```
UPDATE area SET sqkm = sqkm - 500 WHERE country = 'FRA';
UPDATE 1

UPDATE area SET sqkm = sqkm - 500 WHERE country = 'ITA';
UPDATE 1

INSERT INTO area(country, sqkm) VALUES ('FRE', 1000);
ERROR: insert or update on table "area" violates foreign key constraint
"area_country_fkey"
DETAIL: Key (country)=(FRE) is not present in table "country".
```

Esempio (4)
Dunque l'ultima operazione ha prodotto un errore, ma ci troviamo adesso in una situazione di incoerenza dei dati, poiché abbiamo ridotto il territorio francese e italiano senza però creare un nuovo stato costituito dei territori sottratti a Francia e Italia.

```
SELECT country, sqkm FROM area WHERE country IN ('ITA', 'FRA', 'FRE');
```

AREA

country	sqkm
FRA	643301.0
ITA	300840.0

Uso delle transazioni
Con riferimento all'esempio, il comportamento desiderato prevederebbe che nel caso una delle operazioni di aggiornamento del territorio non andasse a buon fine, nessuna variazione avvenga nel database.

```
BEGIN TRANSACTION;
UPDATE measure SET value = value - 500
WHERE country = 'FRA' AND label = 'Area (sq km)';
UPDATE measure SET value = value - 500
WHERE country = 'ITA' AND label = 'Area (sq km)';
INSERT INTO measure (country, label, value)
```

```
VALUES ('FRE', 'Area (sq km)', 1000);
COMMIT;
```

Effetto della transazione
L'uso della transazione fa sì che, raggiunta l'istruzione di `COMMIT`, il DBMS verifichi l'esistenza di un errore generato dall'istruzione di inserimento e perciò effettui un `ROLLBACK`, riportando lo stato dell'intero database a prima dell'inizio della transazione.

```
SELECT country, sqkm FROM area WHERE country IN ('ITA', 'FRA', 'FRE');
```

AREA

country	sqkm
ITA	301340.0
FRA	643801.0

1.2 Problemi di concorrenza

Problemi dovuti a accesso concorrente

- Le operazioni di un DBMS sono eseguite tramite le primitive di `read` dei dati da memoria secondaria a memoria principale e `write` dei dati da memoria principale e memoria secondaria.
- Quando più utenti o applicazioni lavorano in modo concorrente sui dati i tempi di lettura e scrittura possono provocare ulteriori anomalie.
  - Lost update
  - Dirty read
  - Incorrect summary

Lost update
Si verifica quando due client che accedono agli stessi dati eseguono le proprie operazioni in tempi tali che il valore finale di qualche dato risulta errato.

- Esempio:
  1. U1: `X := read(X) + 100`

2. U2: `X := read(X) + 200`
3. U1: `write(X)`
4. U2: `write(X)`

- Poiché U2 legge X prima che U1 scriva, l'aggiornamento effettuato da U1 viene perso.

Dirty read

Si verifica quando un client U1 aggiorna un dato X e successivamente fallisce e il dato X (temporaneamente) aggiornato da U1 viene letto da un altro client U2 prima che X venga riportato al suo valore originale precedente la modifica effettuata da U1.

- Esempio:

1. U1: `X := read(X) + 100`
2. U1: `write(X)`
3. U2: `X := read(X) + 200`
4. U2: `write(X)`
5. U1: `Abort`

- Poiché U2 legge X prima che U1 fallisca, il dato letto da U2 è potenzialmente errato.

Incorrect summary

Se un client calcola una funzione aggregata su un insieme di record mentre un altro client effettua un aggiornamento sui record, il primo client può calcolare la funzione aggregata considerando alcuni valori prima dell'aggiornamento e alcuni valori dopo l'aggiornamento.

- Esempio:

1. U2: `sum := 0`
2. U1: `X := read(X) - 200`
3. U1: `write(X)`
4. U2: `sum := sum + read(X)`
5. U2: `sum := sum + read(Y)`
6. U1: `Y := read(Y) + 200`
7. U1: `write(Y)`

- Poiché U2 legge X dopo che è stato aggiornato e Y prima che venga aggiornato, la somma delle due quantità non è coerente con i valori finali di X e Y.

Proprietà delle transazioni
Il DBMS esegue transazioni concorrenti in modo tale da garantire che le transazioni abbiano le seguenti proprietà, dette proprietà ACID:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Comportamento di default

- Modalità AUTOCOMMIT: ogni operazione SQL viene considerata una transazione (non servono né BEGIN né COMMIT)
- Quando viene lanciato il comando COMMIT/ROLLBACK la transazione corrente termina e ne inizia una nuova.

Atomicity
Se si verifica un errore durante l'esecuzione di una transazione prima del `COMMIT`, gli effetti parziali della transazione non sono memorizzati nel database.

La cancellazione delle modifiche parziali di una transazione si definisce `ROLLBACK`.

Il `ROLLBACK` può essere eseguito automaticamente in caso di errore o anche invocato esplicitamente dall'utente nella definizione della transazione per catturare eventuali condizioni logiche non soddisfatte:

```
BEGIN TRANSACTION;
(operazioni effettuate entro la transazione)
IF (condizione) THEN
    COMMIT;
ELSE
    ROLLBACK [TO SAVEPOINT];
END IF;
```

Consistency

Per ogni client, ogni transazione: (1) Può assumere di lavorare su un DB dove sono verificati tutti i vincoli di integrità; (2) Deve lasciare un DB che soddisfa tutti i vincoli di integrità. Ai fini delle transazioni, i vincoli possono essere specificati come `IMMEDIATE` o `DEFERRED`.

- IMMEDIATE: verifica eseguita durante l'esecuzione della transazione. L'istruzione che causa violazione viene cancellata (UNDO) senza imporre un abort della transazione.
- DEFERRED: verifica al termine della transazione, dopo il COMMIT. Se qualche vincolo viene violato, viene eseguito il ROLLBACK della transazione disfacendo (UNDO) l'intera sequenza di comandi che costituiscono la transazione.

Es. `CREATE TABLE e(a INTEGER PRIMARY KEY INITIALLY DEFERRED, ...);`

Isolation

- Si può garantire alternanza nell'esecuzione dei comandi di transazioni diverse conservando uno stato del DB coerente.
- Le operazioni sono alternate in modo che l'esecuzione sia equivalente a qualche ordine sequenziale (seriale) delle transazioni (nozione di serializzabilità).
- In altri termini il DBMS garantisce che diverse transazioni siano indipendenti e isolate le une dalle altre.

Durability

- Se si verifica un crash dopo che una una transazione ha eseguito `COMMIT`. gli effetti della transazione restano nel DB (non vengono persi).