Azzolini Riccardo 2019-11-29

Transazioni

1 Accessi concorrenti e ripristino

Le basi di dati sono spesso ambienti multiutente, nei quali diversi utenti possono operare contemporaneamente sulla stessa porzione di dati. Per questo, un DBMS deve essere in grado di ricevere ed eseguire comandi contemporaneamente da più utenti. Serve allora un meccanismo di *controllo degli accessi concorrenti*.[1]

Inoltre, in caso di errori, sono necessari dei meccanismi per il *ripristino* (*recovery*) dello stato corretto della base di dati.

2 Transazione

Una **transazione** è una sequenza di operazioni su una base di dati alla quale si vogliono associare particolari caratteristiche di *correttezza*, *robustezza* e *isolamento*, anche in presenza di accessi concorrenti.

- Costituisce un'unità logica di lavoro, non ulteriormente scomponibile.
- La sequenza di operazioni di modifica dei dati (istruzioni SQL) porta la base di dati da uno stato consistente (cioè nel quale sono soddisfatti tutti i vincoli di integrità) a un altro stato consistente, ma non è necessario avere consistenza anche negli stati intermedi, durante l'esecuzione della transazione.

3 Transazione in SQL

In SQL, una transazione è una sequenza di istruzioni, contrassegnata da un *inizio* (BoT) e da una *fine* (EoT).

L'inizio di una transazione viene definito con il comando `START TRANSACTION`. La fine, invece, può essere determinata da due istruzioni diverse:

[1] Tale sistema deve essere realizzato in modo da minimizzare i "disagi" che comporta agli utenti (ad esempio, deve bloccare le operazioni concorrenti sugli stessi dati con una granularità sufficientemente fine, e non sulle intere tabelle).

- L'istruzione `COMMIT [WORK]`, da eseguire quando la transazione termina con successo (cioè la base di dati si trova nello stato finale corretto), rende *permanenti* e *visibili agli altri utenti* le modifiche effettuate.
- L'istruzione `ROLLBACK [WORK]` "annulla" tutte le modifiche eseguite durante la transazione, riportando la base di dati nello stato precedente all'inizio di tale transazione (quest'azione prende il nome di **abort**), e rendendo i dati nuovamente visibili agli altri utenti. Essa viene eseguita quando la transazione termina a causa di un errore, è può anche essere effettuata implicitamente dal DBMS, senza bisogno di specificarla manualmente.

3.1 Esempi

- Trasferimento della somma 100 da un conto corrente bancario a un altro:

```
START TRANSACTION;

UPDATE ContoCorrente
SET Saldo = Saldo + 100
WHERE IBAN = 'IT92X0108201004300000322229';

UPDATE ContoCorrente
SET Saldo = Saldo - 100
WHERE IBAN = 'IT32L0201601002410000278976';

COMMIT;
```

- Trasferimento della somma 100 da un conto corrente bancario a un altro, effettuato solo se essa è effettivamente disponibile:

```
START TRANSACTION;

UPDATE ContoCorrente
SET Saldo = Saldo + 100
WHERE IBAN = 'IT92X0108201004300000322229';

UPDATE ContoCorrente
SET Saldo = Saldo - 100
WHERE IBAN = 'IT32L0201601002410000278976';

SELECT Saldo INTO A
FROM ContoCorrente
WHERE IBAN = 'IT32L0201601002410000278976';

IF A >= 0
```

```
THEN COMMIT WORK;
ELSE ROLLBACK WORK;
```

4 Proprietà delle transazioni

L'insieme di operazioni che costituiscono una transazione deve soddisfare le seguenti proprietà, note come proprietà **ACIDe**:

- **Atomicity** (atomicità);
- **Consistency** (consistenza);
- **Isolation** (isolamento);
- **Durability** (persistenza, o durabilità).

4.1 Atomicità

Le operazioni di una transazione devono essere trattate come una singola unità: o vengono eseguite tutte, oppure non ne viene eseguita alcuna. Per questo, l'atomicità è anche detta proprietà "tutto o niente".

Essa è assicurata dal sottosistema di ripristino (recovery) del DBMS.

4.2 Consistenza

Una transazione deve agire sulla base di dati in modo corretto, cioè rispettare i vincoli di integrità: l'esecuzione della transazione deve portare la base di dati da uno stato iniziale consistente (corretto) a uno stato finale anch'esso consistente.

Come caso particolare, anche l'esecuzione contemporanea di un insieme di transazioni corrette deve mantenere consistente la base di dati. Ciò è garantito dal sottosistema di controllo della concorrenza, che sincronizza le transazioni in modo da evitare interferenze tra di esse.

4.3 Isolamento

L'esito di una transazione non deve essere influenzato dall'esecuzione contemporanea di altre transazioni.

Anche questa proprietà è assicurata dal sottosistema di controllo della concorrenza, impedendo a ciascuna transazione di leggere i risultati intermedi delle altre (i quali potrebbero, inoltre, essere annullati da un successivo rollback).

4.4 Persistenza

I risultati di una transazione che ha effettuato il commit devono essere resi permanenti nella base di dati, nonostante possibili malfunzionamenti del sistema.

La persistenza è assicurata dal sottosistema di ripristino.

5 Transazioni in JDBC

Di default, JDBC implementa l'*auto-commit*: ogni singolo comando SQL viene gestito come una transazione. Per poter effettuare transazioni composte da più comandi, è necessario disabilitare questo comportamento, usando il metodo `setAutoCommit` della classe `Connection`.

Quando l'auto-commit è disattivato, una transazione può essere terminata:

- esplicitamente, tramite i metodi `commit` e `rollback` della classe `Connection`;
- implicitamente, al termine dell'esecuzione.

5.1 Esempio

```
con.setAutoCommit(false);

Statement st = con.createStatement();
ResultSet rs = st.executeQuery("SELECT AVG(valutaz) FROM Film");
rs.next();
if (rs.getDouble(1) < 4) {
    st.executeUpdate("UPDATE Film SET valutaz = valutaz * 1.05");
} else {
    st.executeUpdate("UPDATE Film SET valutaz = valutaz * 0.95");
}

PreparedStatement pst = con.prepareStatement(
    "SELECT titolo, regista, valutaz" +
    "FROM Film" +
    "WHERE genere = ?"
);
pst.setString(1, genere);
rs = pst.executeQuery();

// ...
```

```
con.commit();
con.close();
```

6 Transazioni con SQL ospitato

In un programma embedded SQL, è possibile definire i limiti di una transazione usando direttamente gli appositi comandi SQL (racchiusi tra il prefisso `EXEC SQL` e il terminatore `;`):

- inizio: `EXEC SQL BEGIN TRANSACTION;`
- termine con successo: `EXEC SQL COMMIT;`
- termine con abort: `EXEC SQL ROLLBACK;`