Tecnologie di Sviluppo per il Web

Programmazione su Basi di Dati: Transazioni Parte b

versione 3.0

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Transazioni con JDBC
- Transazioni con i DAO
- Transazioni Applicative
- Locking Ottimistico

Transazioni con JDBC

- Normalmente, lavorando con JDBC
  - ⇨modalità “autoCommit”
  - ⇨una istruzione SQL, una transazione
- Per lavorare in modalità concatenata
  - ⇨disabilitare la modalità autocommit
  - ⇨impostare il livello di isolamento
  - ⇨si tratta di proprietà della connessione

Transazioni con JDBC

- Disabilitare il modo autoCommit
  - ⇨proprietà di tipo boolean di Connection
  - ⇨void setAutoCommit(boolean autoCommit)
  - ⇨esempio: conn.setAutocommit(false);
- Per concludere le transazioni
  - ⇨medodi commit e rollback di Connection
  - ⇨void commit()
  - ⇨void rollback()

Transazioni con JDBC

- Dopo aver disabilitato autocommit
  - ⇨l'esecuzione del primo statement successivo inizia una nuova transazione
  - ⇨la transazione deve essere conclusa con commit() o rollback()
  - ⇨dopo ogni commit() o rollback() comincia automaticamente una nuova transazione

Transazioni con JDBC

- Livello di Isolamento
  - ⇨proprietà di tipo intero di Connection
  - ⇨void setTransactionIsolation(int livello)
- Livello di Isolamento
  - ⇨uno dei quattro livelli di isolamento previsti da SQL
  - ⇨i valori corrispondono a costanti della classe Connection

Transazioni con JDBC

- Livelli di Isolamento
  - Connection.TRANSACTION_READ_UNCOMMITTED
  - Connection.TRANSACTION_READ_COMMITTED
  - Connection.TRANSACTION_REPEATABLE_READ
  - Connection.TRANSACTION_SERIALIZABLE
- Esempio
  - ⇨conn.setTransactionIsolation(Connection.TRANSACTION_REPEATABLE_READ);

Transazioni con JDBC

- Esempio: inserimento di un noleggio
  - ⇨richiede varie operazioni
  - ⇨inserire una ennupla nella tabella noleggi
  - ⇨aggiornare la quantità disponibile nella tabella video
  - ⇨aggiornare il numero di noleggi nella tabella tessere
  - ⇨le operazioni vanno eseguite in modo atomico

Transazioni con JDBC

- Il metodo inserisciNoleggio()
  - ⇨due proprietà della classe: il codice del video ed il codice della tessera
  - ⇨effettua la transazione corrispondente
  - ⇨viene mostrata una versione semplice


```
Connection conn;
try {
    conn = DataSource.getConnection();
    conn.setAutocommit(false);
    conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
    Statement st = conn.createStatement();
    String agg1 = "insert into noleggi values ('" + codVideo + "','" + codTess+"')";
    st.executeUpdate(agg1);
    String agg2 = "update video set qta = qta - 1 " + where cod="  + codVideo + "";
    st.executeUpdate(agg2);
    String agg3 = "update tessere set noleggi=noleggi + 1" + " where cod='" + codTess;
    st.executeUpdate(agg3);
    conn.commit();
} catch(SQLException se){
    try {conn.rollback();} catch (SQLException s) {System.out.println s);
} finally {
    if (conn != null) {
        try {
            conn.setAutocommit(true);
            conn.close();
        } catch (SQLException e) {}
    }
}
```

Transazioni con JDBC

- Attenzione
  - ⇨le transazioni sono legate alla connessione su cui vengono eseguite
  - ⇨è possibile eseguire due transazioni in serie su una connessione
  - ⇨non è possibile eseguire due transazioni contemporaneamente (es: thread) su una connessione (i metodi commit() e rollback() interferirebbero)

Transazioni con i DAO

- In un'applicazione con DAO
  - ⇨tipicamente una transazione è un metodo di un DAO
  - ⇨che richiama metodi CRUD di altri DAO
- Esempio: videonoleggio
  - ⇨supponiamo di avere DAONoleggio, DAOVideocassetta, DAOTessera
  - ⇨vediamo il metodo transazioneNoleggio() di DAONoleggio

Transazioni con i DAO

- transazioneNoleggio()
  - riceve come argomento il riferimento al noleggio, a sua volta in associazione con videocassetta e tessera
  - acquisisce una connessione
  - chiama doInsert(noleggio) di DAONoleggio
  - chiama doUpdate(videoc) di DAOVideocass
  - chiama doUpdate(tessera) di DAOTessera
  - gestendo eventuali eccezioni

Transazioni con i DAO

- Il problema
  - fino ad ora i metodi dei DAO acquisivano ciascuno una propria connessione
  - in questo modo i vari statement sarebbero eseguiti su connessioni diverse, e quindi considerati parte di transazioni diverse (peraltro eseguiti in modalità autocommit)
  - non sarebbe garantita l'atomicità

Transazioni con i DAO

- Soluzione
  - ⇨raddoppiare i metodi CRUD dei DAO
  - ⇨per ciascun metodo CRUD è necessario una versione sovraccarica, capace di eseguire lo statement su una connessione fornita
- Esempio: in DAONoleggio
  - ⇨doInsert(Noleggio n, Connection c)
  - ⇨esegue lo statement sulla connessione c

Transazioni con i DAO

- L'altra versione del metodo CRUD
  - ⇨serve ad effettuare operazioni in modalità non concatenata
  - ⇨per evitare di duplicare il codice, si limita ad acquisire una connessione dalla DataSource e a chiamare l'altra versione per eseguire lo statement
  - ⇨al termine chiude la connessione

```
public class DAONoleggio {

  public static void doInsert(Noleggio noleggio) throws DAOException {
      Connection connection = null;
      try {
          connection = DataSource.getConnection();
          doInsert(noleggio, connection);
      } catch (SQLException sqle) {
          throw new DAOException(sqle);
      } finally {
          DataSource.close(connection);
      }
  }

  public static void doInsert(Noleggio noleggio, Connection connection) throws DAOException {
      Statement statement = null;
      try {
          statement = connection.createStatement();
          String query = "insert into noleggi values (" + ... + ")";
          statement.executeUpdate(query);
      } catch (SQLException sqle) {
          throw new DAOException(sqle);
      } finally {
          DataSource.close(statement);
      }
  }
  ...
```


Transazioni con i DAO

>> aci2

- Un esempio di applicazione
  - ⇨aci2, versione evoluta del Sistema Informativo dell'ACI
  - ⇨organizzata con DAO
  - ⇨esegue una transazione per effettuare lo scambio di targhe tra due automobili
  - ⇨i metodi dei DAO sono tutti duplicati

Transazioni Applicative

- Riassumendo
  - ⇨JDBC offre un supporto alla programmazione con le transazioni
  - ⇨i DAO possono essere riorganizzati in modo da ospitare metodi che eseguono transazioni
- Ma, in realtà...
  - ⇨il problema della programmazione con le transazioni è molto più complesso
  - ⇨molti programmatori lo ignorano del tutto

Transazioni Applicative

- Una importante distinzione
  - ⇨transazione sul DBMS
  - ⇨transazione applicativa
- Transazione sul DBMS
  - ⇨“database transaction”
  - ⇨sequenza di operazioni iniziata con START TRANSACTION e conclusa con COMMIT oppure ROLLBACK; es: noleggio

Transazioni Applicative

- Transazione applicativa
  - “application transaction”
  - caso d’uso implentato nell’operazione che porta all’esecuzione di una transazione sul DBMS
  - esempio: “Utente noleggia videocassetta”
  - tipicamente richiede l’esecuzione di vari schermi e varie azioni, e un’interazione più o meno complessa con l’utente

Transazioni Applicative

- Una tipica transazione applicativa
  - l’utente avvia il caso d’uso
  - il sistema legge dei dati dalla base di dati (es: disponibilità delle videoc.) e li visualizza
  - l’utente effettua delle scelte e fornisce dei dati (es: seleziona il titolo e fornisce il numero di giorni)
  - il sistema effettua un aggiornamento sulla base di dati (es: registrazione noleggio)

Transazioni Applicative

- Attenzione
  - per garantire la correttezza delle operazioni finali è necessario che anche le query precedenti vengano eseguite nella stessa transazione
  - eventualmente con l'opzione SELECT ... FOR UPDATE
  - altrimenti nessuno impedisce ad altre transazioni di modificarli rendendo impossibile l'operazione finale

Transazioni Applicative

- Teoricamente
  - sarebbe possibile avviare la transazione all'inizio del caso d'uso
  - tenerla aperta per la durata del caso d'uso
  - eseguire il commit o il rollback al termine del caso d'uso
  - questa strategia si chiama "locking pessimistico"

Transazioni Applicative

- Locking pessimistico
  - ⇨strategia per cui la durata delle transazioni applicative viene fatta coincidere con quella delle transazioni sul DBMS
  - ⇨in questo modo il DBMS stesso garantisce la serializzabilità e la consistenza delle operazioni attraverso la concessione dei lock
  - ⇨ma a prezzo di seri problemi di prestazioni

Transazioni Applicative

- I problemi del locking pessimistico
  - ⇨le transazioni applicative tendono ad avere lunga durata per via dell'interazione con l'utente (i cosiddetto "think time")
  - ⇨questo aumenta il ciclo di vita dei lock sul DBMS e riduce drasticamente le prestazioni
  - ⇨può accadere che l'utente abbandoni l'applicazione senza chiudere il caso d'uso; in questo caso i lock non verrebbero rilasciati

Transazioni Applicative

- Linea guida
  - ⇨per ragioni di prestazioni, le transazioni sul DBMS dovrebbero durare al più pochi secondi
- La soluzione
  - ⇨spezzare la transazione applicativa in più transazioni sul DBMS (letture prima e scritture poi)
  - ⇨e adottare un “locking ottimistico”

Locking Ottimistico

- Locking ottimistico
  - ⇨si basa sul presupposto per cui la probabilità che due transazioni applicative vadano in conflitto è relativamente bassa
  - ⇨es: due noleggi della stessa cassetta insieme
  - ⇨di conseguenza adotta la strategia di rinunciare ad eseguire tutta la transazione applicativa in modo atomico, preparandosi a rimediare in caso sorgano problemi

Locking Ottimistico

- Una transazione con il locking ottimistico
  - ⇨legge i dati iniziali (es: disponibilità videoc.) in modalità non concatenata
  - ⇨interagisce con l'utente
  - ⇨effettua gli aggiornamenti finali in modalità concatenata
  - ⇨ma prima di farlo verifica che NON ci siano stati nel frattempo cambiamenti nelle ennuple coinvolte nella transazione finale

Locking Ottimistico

- Se non ci sono stati aggiornamenti
  - ⇨la transazione finale viene eseguita regolarmente
- Se ci sono stati aggiornamenti
  - ⇨la transazione finale non può essere eseguita (produrrebbe una perdita di agg.)
  - ⇨il problema può essere gestito in vari modi, a seconda dell'applicazione

Locking Ottimistico

- Tipiche soluzioni in caso di problemi
  - ⇨eseguire il rollback della transazione, segnalare il problema all'utente e chiedere di rieseguire il caso d'uso
  - ⇨segnalare le modifiche intervenute nei dati all'utente, chiedendogli di risolvere esplicitamente il conflitto tra gli aggiornamenti
  - ⇨eseguire la transazione finale, registrando nel log di sistema il problema per segnalarlo all'amministratore del sistema

Locking Ottimistico

- Un aspetto centrale
  - ⇨come riconoscere modifiche nei dati ?
- Anche qui ci sono varie soluzioni
  - ⇨una soluzione consiste nel confrontare i dati letti all'inizio del caso d'uso con quelli nella base di dati (problematico: richiede di conservare i dati originali)
  - ⇨una soluzione più semplice consiste nell'utilizzare un numero di versione

Locking Ottimistico

- Numero di versione
  - a ciascuna ennupla potenzialmente coinvolta in una transazione applicativa lunga viene aggiunto un attributo intero “versione”
  - l’attributo viene previsto anche nel bean corrispondente, e caricato durante l’esecuzione del metodo doSelect()
  - il codice dei DAO viene scritto in modo da incrementare il numero di versione ad ogni aggiornamento

Locking Ottimistico

- Numero di versione (continua)
  - prima della transazione finale, il DAO confronta il numero di versione in ciascun bean coinvolto con i numeri presenti nella base di dati
  - in caso siano tutti uguali procede all’aggiornamento
  - altrimenti attiva la procedura scelta per gestire il problema

Locking Ottimistico

- Vantaggio del locking ottimistico
  - ⇨l'impatto sulle prestazioni è modesto e non altera significativamente il volume di transazioni eseguite dal DBMS
- Svantaggio del locking ottimistico
  - ⇨la programmazione è decisamente più complicata

Locking Ottimistico

>> aci2

- Un esempio
  - ⇨aci2 e lo scambio di targhe
  - ⇨alla classe Automobile è stato aggiunto il numero di versione
  - ⇨il numero di versione viene aggiornato dai DAO
  - ⇨la transazione effettua un locking ottimistico e lancia eccezione nel caso in cui i numeri di versione non corrispondano

Riassumendo

- Transazioni con JDBC
- Transazioni con i DAO
- Transazioni Applicative
- Locking Ottimistico

Termini della Licenza