Laboratorio di Basi di Dati 2
Secondo progetto

Cosa consegnare: Un documento compresso contenente:

1. Un SINGOLO documento cartaceo contenente: (i) descrizione software utilizzato, come richiesto al punto 1; (ii) per ogni domanda contenuta al Punto 2, descrizione della soluzione implementative adottata e risposta alle domande contenute nel testo.
2. Il codice delle applicazioni realizzate.

Software da utilizzare. Un DSMS a scelta tra PostgreSQL, Oracle e Microsoft SQL Server. La documentazione prodotta deve essere commisurata alla quantità di documentazione e informazioni disponibili relativamente ai protocolli di controllo della concorrenza nel DBMS scelto.

Punto 1: Comprensione dello strumento. Basandosi sulla documentazione disponibile e su prove pratiche, acquisire familiarità sui protocolli di controllo della concorrenza disponibili nel sistema prescelto. Produrre una breve relazione in cui si descrivano tali elementi evidenziando le principali caratteristiche differenze rispetto a quanto visto a lezione, con particolare riferimento a: (i) livelli di isolamento; (ii) lock escalation. Riportare le fonti delle informazioni inserite nella documentazione.

Punto 2: Transazioni in Java e JDBC.

1. *Transazioni singole.* Considerare un programma Java che si interfacci al database tramite JDBC (potete adattare quanto proposto nel file `labo.java` disponibile su AulaWeb). Realizzate un programma Java per eseguire ciascuna delle transazioni specificate nei punti 2,3,4, dell'Esercitazione 4, cancellando preventivamente le tuple nella tabella `Account`. Provare quindi a rieseguirle utilizzando settaggi diversi del parametro `AUTOCOMMIT`.

   Attenzione: In JDBC, data una connessione `conn`**:** `conn.setAutoCommit(false|true)`**). Se tale parametro è settato a** `true`**, il commit viene eseguito automaticamente dopo l'esecuzione di ogni comando SQL.**
   Commentare i risultati ottenuti, anche in confronto a quanto accade per le transazioni SQL, in riferimento all'autocommit.

2. *Transazioni concorrenti.* Cancellare tutte le tuple eventualmente presenti a questo punto nelle due tabelle, riferite ai conti con numero da 1 a 100 (lasciate la tupla corrispondente al conto con numero 0). Dopo aver esaminato e provato ad eseguire con diversi valori dei parametri il file `ConcurrentTransactions.java` disponibile su Aulaweb, modificarlo opportunamente in modo che, invocato con valori in input 100 100, inserisca i 100 conti con saldo 0 in 100 thread concorrenti differenti. (Provare a cancellare le tuple e reinserire i conti eseguendolo con valori 100 1 e osservare la differenza).

   Modificare ora il file `ConcurrentTransactions.java` disponibile su Aulaweb, generando una connessione distinta per ogni thread. Che cosa cambia? Riuscite ad indicare il numero di transazioni eseguita dal server nel primo e nel secondo caso? Qual é nei due casi la relazione tra thread e transazioni? Quali prove avete fatto per rispondere alla domanda?

3. *Isolamento.* Provare ora modificando il programma `ConcurrentTransactions.java` a eseguire per ogni conto 1..100 la seguente transazione

   ```
   e ← SELECT balance FROM Account WHERE number=i
   UPDATE Account SET balance=e + 1 WHERE number=i
   c ← SELECT balance FROM Account WHERE number=0
   UPDATE Account SET balance=c - 1 WHERE number=0
   ```

   Dopo l'esecuzione tutti i conti 1..100 contengono correttamente 1, mentre il conto 0 non é detto sia a 0. Perché? Giustificare il comportamente del sistema, tenendo anche in considerazione il livello di isolamento impostato di default.

4. *Isolamento: variazione.* Provare ora modificando il programma `ConcurrentTransactions.java` a eseguire per ogni conto 1..100 la seguente transazione

   ```
   UPDATE Account SET balance=balance+1 WHERE number=i
   UPDATE Account SET balance=balance-1 WHERE number=0
   ```

   Provare a rieseguire le transazioni (dopo aver ripristinato i valori iniziali dei conti). Commentare i risultati ottenuti, giustificando il comportamento del sistema, tenendo anche in considerazione il livello di isolamento impostato di default.

5. *Isolamento: modifiche.* Provare a rieseguire i programmi al punto 3 e al punto 4 (ripristinando ogni volta i saldi iniziali dei conti) con diversi livelli di isolamento per le transazioni e osservare i comportamenti ottenuti.
   Attenzione: Per settare il livello di isolamento utilizzare `conn.setTransactionIsolation (level)` **dove** `level` **puó assumere uno tra i valori**

   - `Connection.TRANSACTION_READ_UNCOMMITTED`,
   - `Connection.TRANSACTION_READ_COMMITTED`,
   - `Connection.TRANSACTION_REPEATABLE_READ`,
   - `Connection.TRANSACTION_SERIALIZABLE`.

   Commentare i risultati ottenuti.

Punto 4: Tuning del livello di isolamento. Considerare ora i seguenti tipi di transazione, che devono essere eseguite concorrentemente sulla base di dati, popolata con almeno 50 filiali e 1000 conti.

- T_1: Addebita/accredita dei soldi su uno o piú conti e aggiorna il saldo delle filiali corrispondente. Ogni transazione di tipo T_1 agisce su un insieme di conti differente.
- T_2: Legge il saldo di un insieme di conti. Ogni transazione di tipo T_2 agisce su un insieme di conti differente.
- T_3: Confronta il saldo di ogni filiale con la somma dei saldi dei conti in quella filiale.

Definire un opportuno concetto di throughput e correttezza relativi all'esecuzione concorrente delle transazioni in oggetto.

Selezionare il livello di isolamento che si ritiene piú opportuno per le varie tipologie di transazione, motivando le scelte effettuate.

Mostrare i livelli di throughput e correttezza che si ottengono con l'esecuzione concorrente di: (i) una transazione di tipo T_1 **ogni** $x\%$ **conti**; (ii) una di tipo T_2 **ogni** $x\%$ **conti**; (iii) una di tipo T_3. Considerare a questo proposito almeno tre valori diversi di $x > 0$.

Se si ritiene che piú di un livello di isolamento possa essere ragionevole per un tipo di transazione, modificare i livelli di isolamento e confrontare i risultati, in termini di throughput e correttezza, generati in base alle due alternative.