



UNIVERSITA' DEGLI STUDI ROMA TRE

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesina

Tecnologia delle Basi di Dati

a cura di
Marco Piolo

Anno Accademico 2006-2007

Indice

1.1	DataBase Utilizzato	2
1.2	Sperimentazione e considerazioni	3
1.2.1	Senza indici	4
1.2.2	Con indici senza statistiche	6
1.2.3	Con indici con statistiche	7
2.1	Sperimentazioni e considerazioni	10
2.1.1	Perdita di aggiornamento	11
2.1.2	Lecture sporche	12
2.1.3	Lecture inconsistenti	13
2.1.4	Aggiornamento fantasma	14
2.1.5	Inserimento fantasma	16
3.1	Obiettivi della sperimentazione	18
3.1.1	Struttura logica della base di dati	18
3.1.2	Requisiti	19
3.2	Sperimentazione	19
3.2.1	Progettazione dello schema dimensionale	19
3.2.2	Popolamento del data warehouse	20
3.2.3	Verifica del Data Warehouse	22
3.3	Conclusioni	23

Capitolo 1

Strutture fisiche

Sperimentare le strutture fisiche di un DBMS, definendo alcune relazioni ed alcune interrogazioni che prevedano selezioni, proiezioni e join. Utilizzare relazioni di dimensioni sufficientemente grandi da rendere conveniente l'uso di indici. Mostrare, con riferimento al DBMS scelto, il comportamento del sistema, in presenza o in assenza di indici e prima e dopo l'aggiornamento delle statistiche.

1.1 DataBase Utilizzato

Il DBMS scelto per eseguire l'esperimento è il DB2 della IBM. Per poter condurre l'esperimento è stato necessario costruire un database di dimensioni relativamente grandi per poter apprezzare gli effetti dell'utilizzo degli indici.

E' stato utilizzato un Database con informazioni relative ad acquisti di ipotetici prodotti, organizzato con le seguenti tre relazioni:

- (codiceArt, descrizione, prezzo, bool)
Numero di record: 10000
- (codiceCl, nome, cognome)
Numero di record: 2000
- (codiceCl, codiceArt, timestamp, timestamp)
Numero di record: 12

Le relazioni sono state popolate tramite un apposito programma Java con valori casuali.

1.2 Sperimentazione e considerazioni

Il Database è stato sollecitato con le interrogazioni che seguono per poter verificare i casi in cui si ha un beneficio consistente nell'utilizzo degli indici.

```
select * from clienti where codiceCl = 2007;
```

Poichè il campo codiceCl è un campo chiave DB2 crea automaticamente un indice univoco.

```
select * from articoli where descrizione = Scarpe ;
```

Questa operazione è simile alla precedente, ma il campo Descrizione non è chiave e non sarà automaticamente creato un indice.

```
select * from articoli where prezzo = 20;  
select * from articoli where prezzo/4 = 5;
```

Queste due interrogazioni servono a dimostrare come, sebbene producano lo stesso risultato, il fatto di inserire operazioni aritmetiche all'interno dell'interrogazione faccia sì che non vengano utilizzati indici da parte del DBMS

```
select * from articoli where bool = 1;
```

L'interrogazione restituisce tutti i farmaci che hanno il campo bool impostato a 1. Essendo un campo treano si vede come anche l'inserimento di un indice non porti benefici a livello di prestazioni.

```
select a.descrizione, c.nome, c.cognome from articoli a, scontrini s,  
clienti c where s.codiceArt = a.codiceArt and s.codiceCl = c.codiceCl and  
cognome = Rossi ;
```

Questa interrogazione, assai più pesante delle precedenti, ha il compito di mostrare tutte le ricette appartenenti al paziente Rossi. Per fare ciò, per come è stato strutturato il db, è necessario eseguire un join tra tre tabelle.

1.2.1 Senza indici

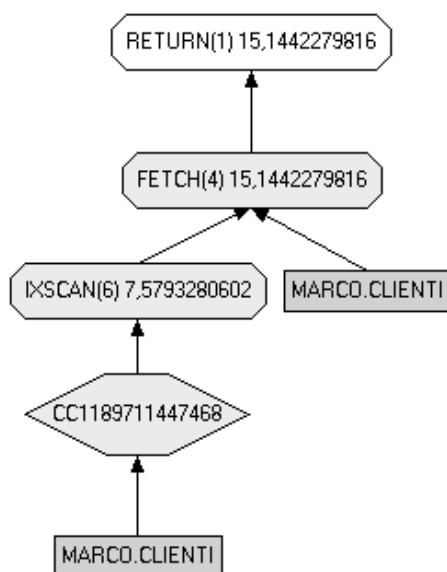


Figura 1.1: Selezione con indice

La prima interrogazione consiste in una selezione su un campo chiave. Poichè DB2 ha creato automaticamente un indice su questo campo, il DBMS ha eseguito l'operazione con un costo totale di 15,144 timeron. Il timeron è un'unità di misura del DBMS DB2 non direttamente legato al tempo di esecuzione, ma che fornisce una stima delle risorse richieste per effettuare la query. Questa stima viene calcolata in funzione del costo speso dalla CPU e il costo speso dalle operazioni di I/O.

Il piano di accesso di DB2 (vedi figura 1.1) evidenzia come il DBMS abbia prima eseguito un'operazione IXSCAN (un accesso all'indice della tabella)

tramite la quale, con il risultato ottenuto, ha potuto effettuare il fetch per estrarre i dati dalla tabella.

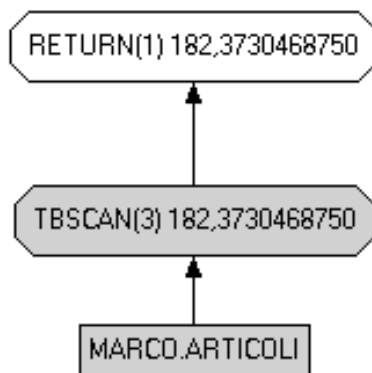


Figura 1.2: Selezione senza indice

La seconda interrogazione esegue una select su un campo non chiave. Il piano di accesso (vedi figura 1.2) rivela come il DB2 esegua direttamente un accesso su tutte le tuple della tabella interessata (TBSCAN). Il costo impiegato è nettamente maggiore (182,373 timeron).

Situazioni analoghe si verificano per la terza, la quarta e la quinta interrogazione.

La sesta interrogazione, che prevedeva l'utilizzo di un join su tre tabelle, prevede un accesso su un campo non chiave (e per il quale DB2 non ha creato automaticamente un indice). Il costo di tale operazione è di 56,87 timeron. Il piano di accesso (vedi figura 1.3) mostra come abbia eseguito un primo join tra la tabella scontrini (che beneficia dell'indice sul campo richiesto) e la tabella clienti (che non ne beneficia, poichè cerca i clienti con cognome Rossi, campo per il quale non è stato definito un indice) e il risultato viene nuovamente messo in join con la tabella dei articoli (la quale dispone anch'essa di un indice sul campo richiesto). E' possibile osservare come l'operazione più costosa sia l'accesso a tutte le righe della tabella clienti (41,69 timeron).

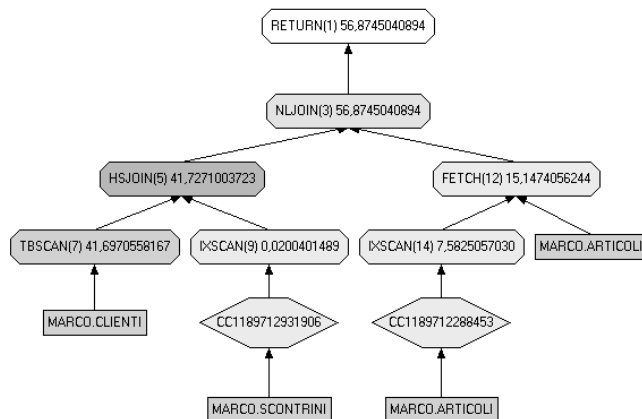


Figura 1.3: Join con selezione senza indice

1.2.2 Con indici senza statistiche

Abbiamo visto come la seconda interrogazione che selezionava i record, sfruttando un campo senza indice, era molto più lenta della prima che si avvaleva di un campo chiave con indice annesso.

```
select * from articoli where descrizione = Scarpa ;
```

Per ovviare a questo problema si può creare un indice secondario sul campo descrizione.

```
create index DescrizioneInd on articoli descrizione ;
```

Eseguire questa operazione non è sufficiente per poter beneficiare dell'indice. Infatti il DB2 continua ad eseguire la stessa interrogazione con costo 182,373 timeron senza avvalersi dell'ausilio dell'indice.

Questo perchè, dopo la creazione di un indice, è necessario aggiornare il catalogo con il comando `runstats` altrimenti gli indici non saranno utilizzati dal DBMS.

```
runstats on table marco.articoli on all columns and index marco.DescrizioneInd;
```

Dopo aver aggiornato le statistiche, il DBMS esegue la stessa interrogazione di prima, sfruttando l'indice e riducendo il costo dell'operazione a 38,579 timeron.

Lo stesso esperimento è analogo per tutte le altre interrogazioni.

1.2.3 Con indici con statistiche

Dopo aver verificato come un indice abbia migliorato notevolmente le prestazioni della seconda operazione (dopo aver aggiornato le statistiche) si può condurre un esperimento interessante su delle interrogazioni nelle quali sono coinvolte delle operazioni aritmetiche.

```
select * from articoli where prezzo = 20;  
select * from articoli where prezzo/4 = 5;
```

Prima di tutto è necessario definire un indice sul campo prezzo ed aggiornare successivamente le statistiche.

```
create index PrezzoInd on articoli prezzo ;  
runstats on table marco.articoli on all columns and index marco.PrezzoInd;
```

In questo modo la prima delle due interrogazioni viene svolta in un tempo molto minore. Infatti il DBMS, avvalendosi dell'indice, ha ridotto il costo da 182 timeron a 92,255 timeron.

Eseguendo la seconda interrogazione accade che l'indice in questo caso non è stato considerato. Ciò è dovuto al fatto che all'interno dell'istruzione SQL vi era l'espressione aritmetica sul campo prezzo che ha invalidato l'utilizzo dell'indice. In questo caso il costo dell'operazione è rimasto invariato.

E' possibile mostrare un altro caso in cui l'utilizzo di indici è superfluo

```
select * from articoli where bool = 1;
```

Costruendo un indice sul campo bool e rieseguendo la stessa interrogazione

il gestore delle interrogazioni del DBMS ignora l'esistenza dell'indice ed esegue l'operazione con gli stessi costi che avrebbe avuto senza l'indice definito sul campo bool. Questo perchè il campo bool può assumere solo i valori 0 o 1 e l'indice appena definito è secondario. In questo caso il DBMS deve accedere a parecchie pagine e ciò vanifica i benefici apportati dall'indice.

Esaminando la penultima interrogazione

select a.descrizione, c.nome, c.cognome from scontrini s ,articoli a,clienti c where a.codice = s.codiceArt and s.codiceCl = c.codiceCl and nome = Marco and cognome = Rossi ;

e costruendo un indice, stavolta multi-attributo, sui campi cognome e nome della relazione clienti

create index ClientiInd on clienti cognome,nome ;

l'interrogazione viene eseguita con un costo nettamente minore rispetto all'esecuzione senza l'indice definito.

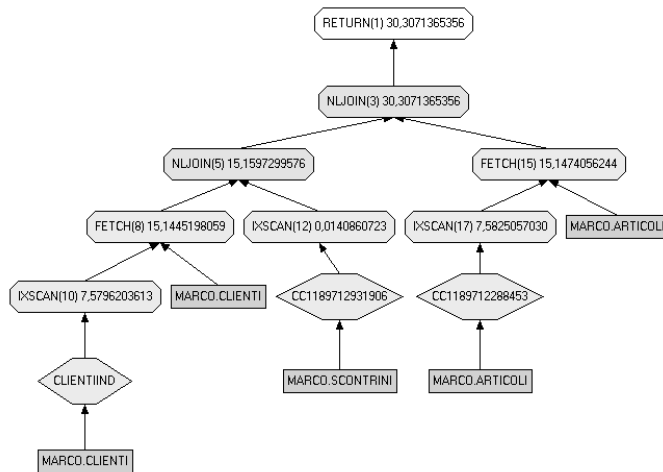


Figura 1.4: Join con selezione con indice

Infatti il piano di accesso (figura 1.4) mostra un costo di 30,307 timeron

contro i 196 dell'esecuzione precedente. Questo perchè l'operazione più pesante, in questo caso, era la selezione dei clienti, che hanno come cognome Rossi e che è stata notevolmente alleggerita dall'indice appena creato. E' buona norma costruire un indice quando si esegue un join sul campo coinvolto della relazione più grande. Così facendo tutti i campi coinvolti posseggono già un indice, poichè sono chiavi e il DB2 vi crea automaticamente un indice.

Concludendo, si possono esprimere le seguenti riflessioni sugli indici. E' infatti conveniente utilizzarli quando:

- le relazioni hanno dimensioni importanti (almeno 200 tuple)
- sono definiti su dei campi dove le selezioni sono molto frequenti
- sono definiti su dei campi dove i valori sono molto variabili
- le interrogazioni non coinvolgono operazioni aritmetiche
- le operazioni di aggiornamento non sono in misura maggiore delle operazioni di ricerca
- sono coinvolte operazioni di join

Dagli esperimenti sopra esposti si evince come l'aumento di prestazioni nell'utilizzo degli indici è considerevole, rispettando però alcune regole d'uso, pena il mancato utilizzo degli indici da parte del DBMS

Capitolo 2

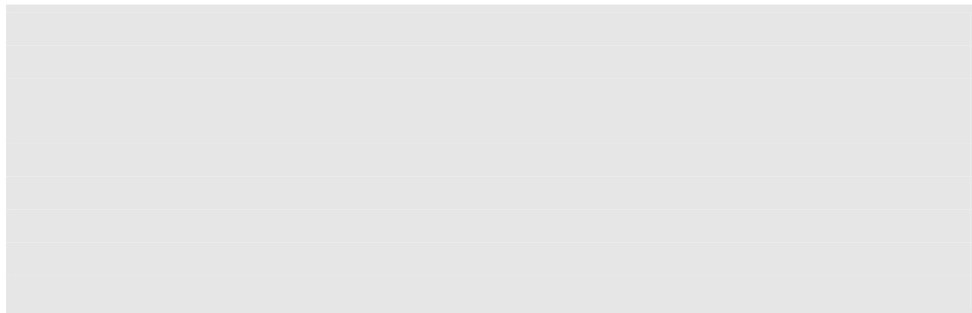
Livelli d'isolamento

Sviluppare semplici programmi che permettano di verificare i diversi livelli di isolamento previsti da SQL e da JDBC

2.1 Sperimentazioni e considerazioni

Gli esperimenti sono stati eseguiti attraverso un programma Java col compito di simulare il comportamento di ogni anomalia prima e dopo l'applicazione di un livello di isolamento. Ogni volta che il programma simulava un'anomalia il flusso di esecuzione veniva sdoppiato in due thread in modo tale che due flussi separati eseguissero le transazioni in maniera concorrente.

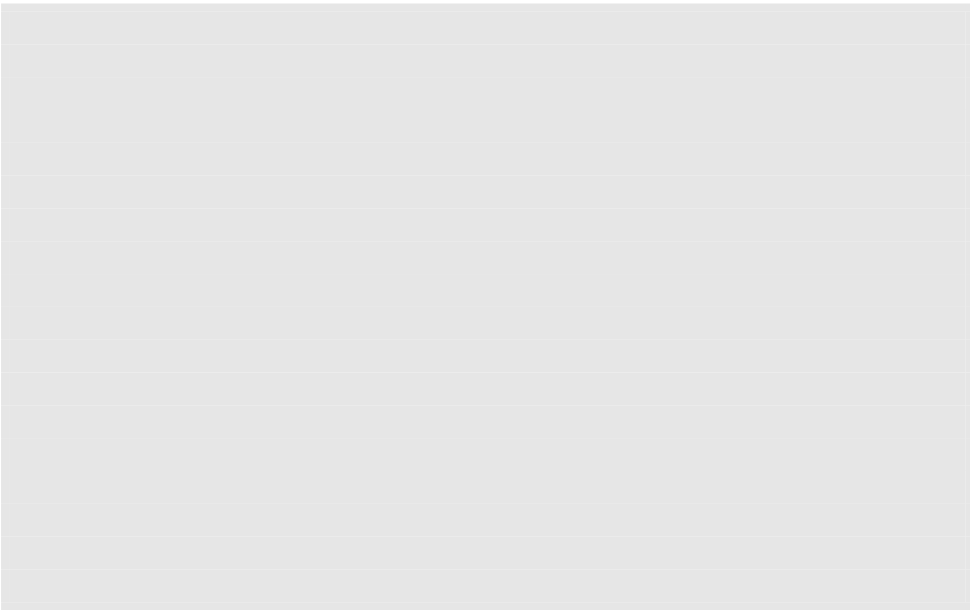
Per prima cosa è stato necessario interrogare il DBMS utilizzato (ancora DB2) sulla disponibilità della gestione dei livelli di isolamento. Infatti, non tutti i DBMS mettono a disposizione i livelli sopra citati. Il DBMS (attraverso il programma Java) ha fornito la seguente risposta.



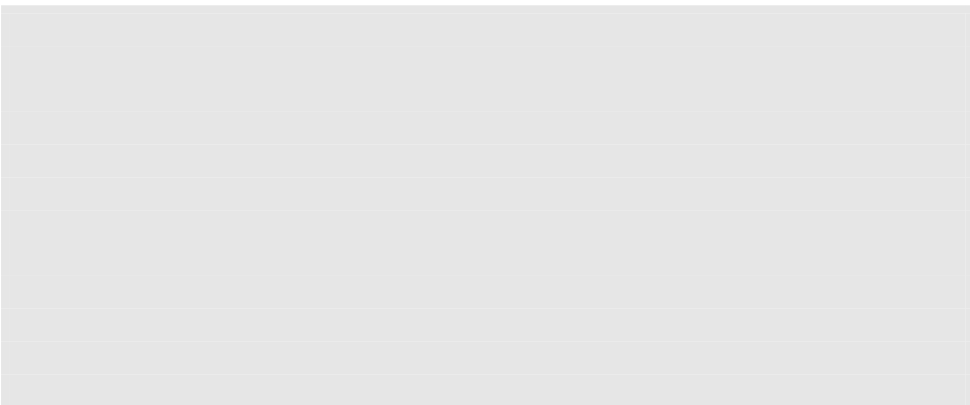
Una volta verificata la disponibilità dei livelli di isolamento è stato condotto un esperimento per ogni anomalia possibile.

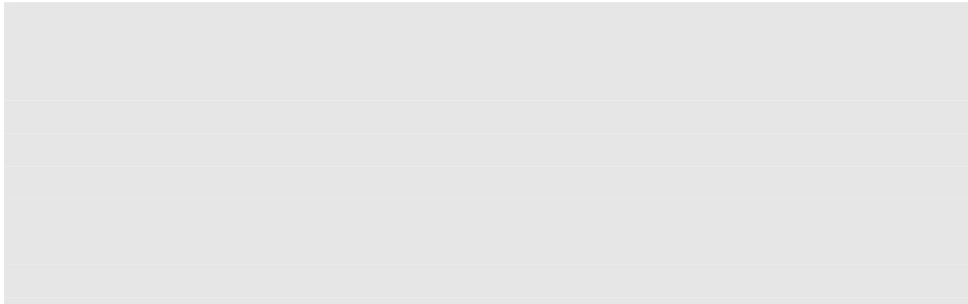
2.1.1 Perdita di aggiornamento

La perdita di aggiornamento è stata simulata con l'esecuzione di due transazioni che aumentavano di una unità il valore di un determinato oggetto.



Durante l'esecuzione di queste transazioni è stato usato il livello di isolamento READ COMMITTED (modo 2) che non risolve le anomalie di perdita di aggiornamento. Modificando il livello di isolamento a REPEATABLE READ si ha un risultato diverso.

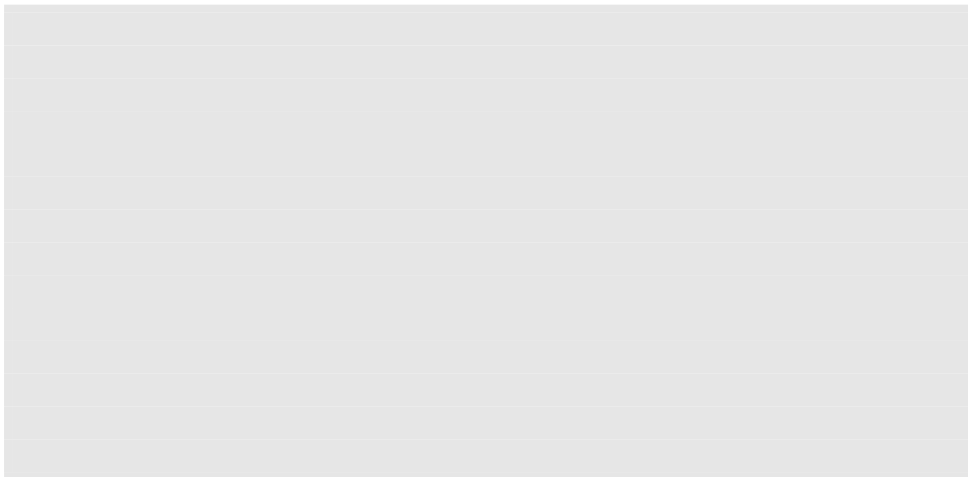




Il flusso di esecuzione viene interrotto quando la prima transazione prova ad aggiornare il valore dell'oggetto (in questo caso il prezzo dell'Scarpa). Il DBMS, poichè ha eseguito un lock, impedisce alla transazione T1 di completare le proprie operazioni intercettando l'anomalia ed annullando il proseguimento dell'operazione.

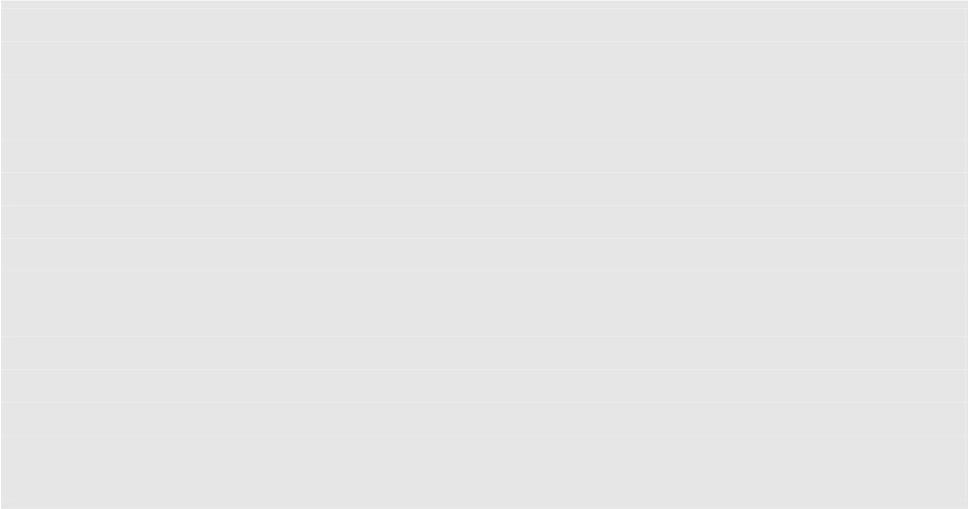
2.1.2 Letture sporche

La lettura sporca è stata simulata con l'esecuzione di una transazione con il compito di leggere e modificare il valore dell'oggetto x e di una seconda transazione col compito di leggere il valore modificato prima che la transazione precedente venisse annullata.



Il livello di isolamento utilizzato per questo primo esperimento è stato il READ UNCOMMITTED, non sufficiente per questo tipo di operazioni concorrenti. Si può vedere come la transazione T2 legga il valore 30 quando

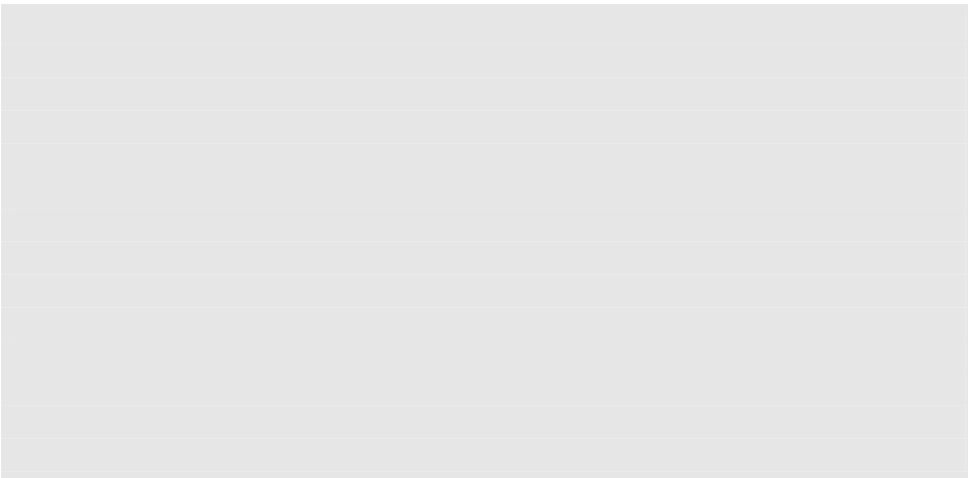
invece il valore effettivo presente sulla base di dati è 29. Utilizzando come livello di isolamento READ COMMITTED il risultato è differente.



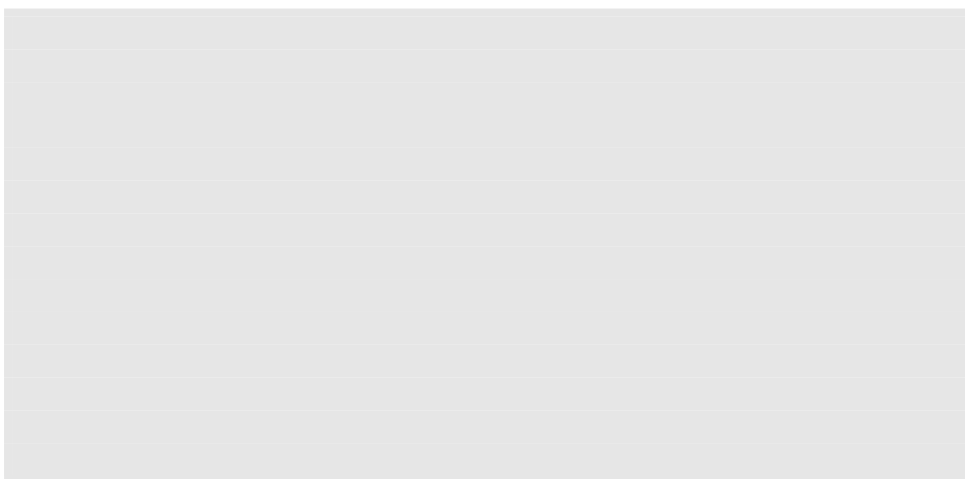
In questo caso il livello READ COMMITTED previene l'anomalia di lettura sporca. Infatti la transazione T2 legge il valore 62 nonostante sia stato incrementato a 63.

2.1.3 Letture inconsistenti

Anche qui l'anomalia è stata simulata mediante l'utilizzo di due transazioni. La prima aveva il compito di leggere due volte il valore dello stesso oggetto. La seconda quello di modificare il valore dell'oggetto tra una lettura e l'altra.



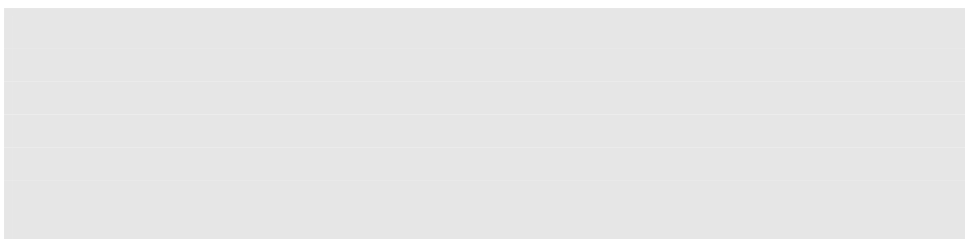
Il risultato dell'esecuzione mostra come la prima transazione legga per lo stesso oggetto due valori diversi. Tale anomalia si è verificata utilizzando come livello di isolamento READ COMMITTED, non sufficiente per questo tipo di problemi sulla concorrenza. Utilizzando anch'essa REPEATABLE READ il DBMS anche stavolta si comporta diversamente.

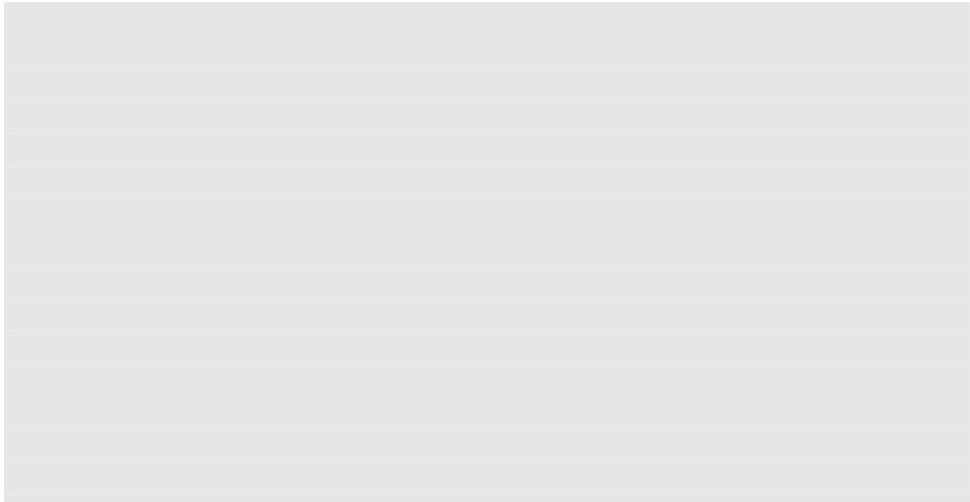


La transazione T1 con questo livello di isolamento continua a leggere il dato che aveva letto in precedenza, nonostante T2 lo abbia modificato tra le due letture.

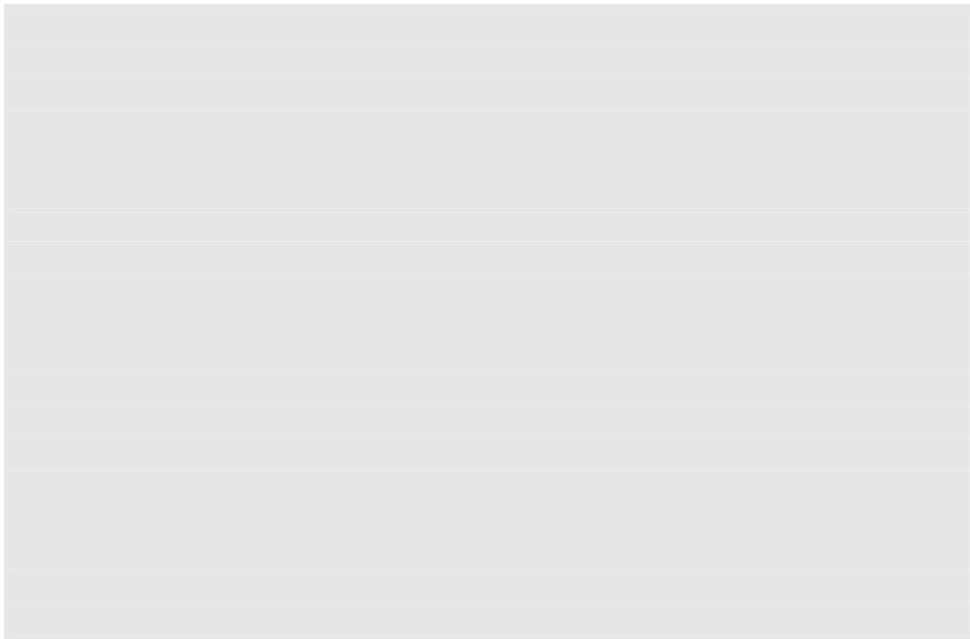
2.1.4 Aggiornamento fantasma

Tale anomalia si verifica aggiornando due valori di due oggetti letti dalla prima transazione. Il primo valore viene letto prima dell'esecuzione della seconda transazione che li modifica. Il secondo valore viene invece letto dopo l'esecuzione della seconda transazione. In questo modo la prima transazione perde l'aggiornamento del primo valore e si accorge solamente del secondo valore. Ciò porta ad un alterazione dei dati letti per la prima transazione.





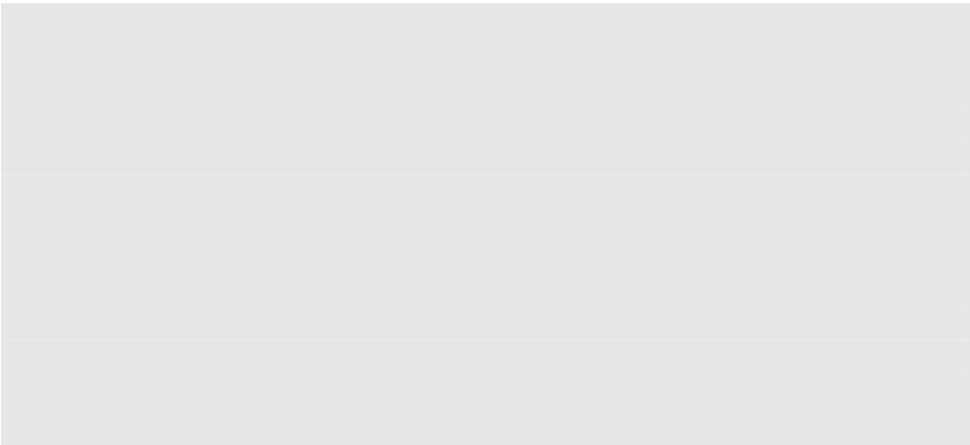
T2 decrementa di 1 il valore di y (il prezzo dell'Scarpa) e incrementa di 1 il valore di z (il prezzo dell' Orologio). Ma T1 non si è potuta accorgere del decremento del valore di y . Sebbene T2 non abbia alterato il valore della somma, (incrementare di 1 e decrementare di 1 il valore di due addendi non cambia il risultato) T1 si è accorta solo parzialmente di questo aggiornamento. Il risultato della somma finale è incrementato da 102 a 103. Il livello di isolamento utilizzato in questo caso è stato `READ COMMITTED`.



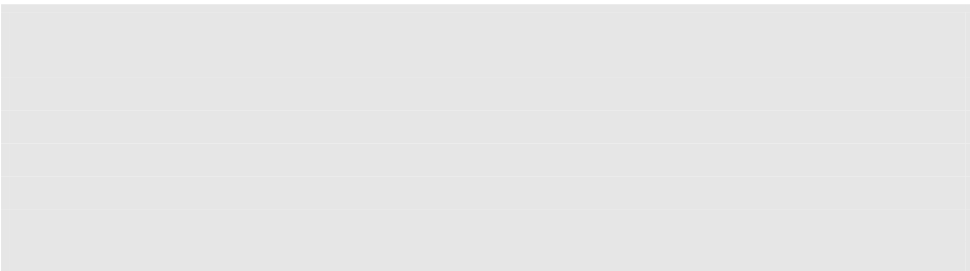
Utilizzando REPEATABLE READ si può notare come il DBMS permetta alla transazione T2 tutte le letture sugli oggetti in comune con la transazione T1. Quando invece T2 necessita di modificare uno di questi valori la transazione viene bloccata in attesa del completamento della transazione T1. Una volta che T1 è stata completata viene ridata la possibilità di scrittura a T2.

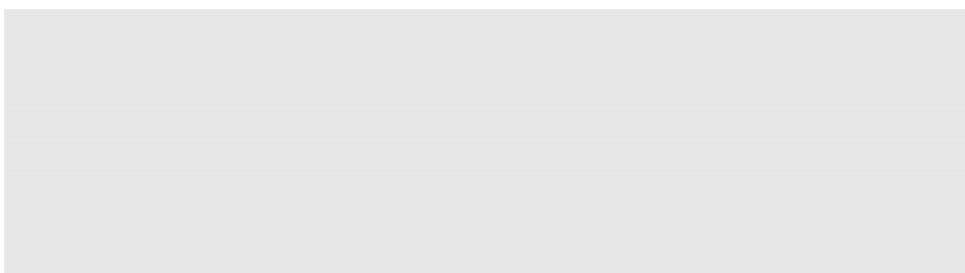
2.1.5 Inserimento fantasma

L'ultima anomalia si presenta quando vengono coinvolti degli inserimenti di nuove tuple. In questo caso la prima transazione aveva il compito di contare due volte il numero di articoli a disposizione dalla farmacia. Una seconda transazione avrebbe aggiunto tra le due letture un nuovo farmaco andando così a rendere incoerente il risultato della seconda lettura.



Si può vedere come il conteggio dei articoli tra le due letture sia differenti. Utilizzando stavolta il livello di isolamento più stringente (SERIALIZABLE) si può notare come l'anomalia di aggiornamento fantasma venga impedita.





Con il livello di isolamento `SERIALIZABLE`, che applica un lock di predicato, vengono impediti oltre all'aggiornamento dei valori anche gli inserimenti di nuovi record all'interno della stessa relazione.

E' dunque evidente la potenzialità del poter scegliere diversi livelli di isolamento all'interno di DBMS a seconda delle proprie esigenze: livelli più bassi per prestazioni migliori, qualora l'integrità dei dati non sia un fatto vitale per l'applicazione; livelli più alti di isolamento per garantire la massima integrità delle informazioni all'interno del sistema a scapito delle prestazioni.

Capitolo 3

Data Warehouse

Costruire e popolare un data warehouse a partire da una base di dati relazionale, relativa alle prescrizioni di farmaci acquisite da un insieme di farmacie.

3.1 Obiettivi della sperimentazione

3.1.1 Struttura logica della base di dati

La base di dati utilizzata per la sperimentazione ha la seguente struttura logica:

- (Numero, CodFarmacia, CFPaziente, Data)
- (CodFarmacia, Nome)
- (NumRicetta, NumLinea, CodFarmaco)
- (Codice, Descrizione, CodMolecola, CodCasa, Prezzo, Fascia)
- (CodMolecola, Descrizione)
- (CF, Cognome, Nome, DataNascita, Via, NumeroCivico, Città)
- (CodCasa, Nome)

- (Codice, Nome)
- (Via, Città, NumeroCivico, ASL)

3.1.2 Requisiti

Si vuole costruire un data warehouse, attraverso il modello multidimensionale, che permetta di analizzare complessivamente la quantità e il prezzo delle prescrizioni effettuate rispetto a concetti come:

-
- (comprensivi di molecola e casa farmaceutica)
-
-

Si vuole supporre che per ragioni di privacy non possano essere riportati dati relativi alle identità dei pazienti (nome, cognome, residenza...).

3.2 Sperimentazione

3.2.1 Progettazione dello schema dimensionale

Partendo dalle richieste da soddisfare, si può notare come il fatto principale sia l'analisi delle prescrizioni, le misure siano la quantità e l'importo e le dimensioni sulle quali svolgere l'analisi siano la data, il tipo di farmaco, l'ASL di residenza, la fascia di età dei pazienti e le farmacie in cui sono stati venduti i farmaci.

La tabella dei fatti ha la seguente forma:

- (KData, KFarmacia, KFarmaco, KASL, KFasciaEtà, Quantità, Importo)

Si può notare come nella relazione è stata creata una chiave per ogni dimensione di interesse. Le misure che si vogliono analizzare sono invece la quantità e l'importo come richiesto dalle specifiche.

Per ogni chiave si va a costruire una tabella dimensione:

- (KFarmaco, Codice, Descrizione, CodMolecola, DescrizioneMolecola, CodCasa, NomeCasa, Fascia)
- (KFarmacia, CodFarmacia, Nome)
- (KASL, CodiceASL, Nome)
- (KFasciaEtà, 0-20, 20-40, 40-60, Over60)
- (KData, Giorno, Mese, Anno)

Si possono fare quindi alcune riflessioni sulle dimensioni. L'attributo prezzo della relazione FarmaciDim non è stato inserito poichè rientra già nell'aggregazione dei prezzi nell'importo della relazione FattiPrescrizioni.

Inoltre, poichè per questione di privacy, non era concesso memorizzare le informazioni personali dei pazienti, sono state eliminate tutte quelle informazioni che potrebbero far risalire all'identità della persona. Perciò l'unica informazione dei pazienti che viene mantenuta è la data di nascita o, meglio, la fascia d'età in cui sono inquadrati.

3.2.2 Popolamento del data warehouse

Una volta costruita la struttura logica è necessario costruire le interrogazioni per estrarre dalla base di dati di partenza le informazioni di interesse da inserire nel data warehouse.

E' necessario aggregare, quindi, la quantità di prescrizione fatte e la somma dei prezzi delle singole prescrizioni sulla base del tipo di farmaco, della farmacia, della fascia d'età e dell'ASL di appartenenza. Per fare ciò si deve applicare un join tra la relazione Ricette e la relazione ElementiRicetta, effettuando un conteggio delle linee per sapere la quantità delle prescrizioni fatte e la somma dei prezzi per avere l'importo totale, e raggruppando per le dimensioni necessarie (farmaco, farmacia...).

In sostanza, quest'operazione può essere riassunta in un'unica interrogazione, la quale, ovviamente, può essere scomposta in più interrogazioni lavorando soltanto sui codici delle dimensioni.

```
select count numero as quantit , sum prezzo as importo, frm.codfarmacia, frm.nome, f.codice, f.descrizione, year data as anno, month data as mese,
```

day data as giorno, a.codice as codice, a.nome as nome, 0-20, 20-40, 40-60, over60, m.codmolecola as codmolecola, m.descrizione as descrizione molecola, c.codcasa as codcasa, c.nome as nome casa, fascia from ricette r, elementiricetta, farmaci f, farmacia frm, pazienti p, territorio t, asl a, pazientiaux paux, molecole m, casefarmaceutiche c where numero = NumRicetta and r.codfarmacia = frm.codfarmacia and codfarmaco = f.codice and cfpaziente = p.cf and p.via = t.via and p.numerocivico = t.numerocivico and p.citt = t.citt and t.asl = a.codice and p.cf = paux.cf and f.codmolecola = m.codmolecola and f.codcasa = c.codcasa group by frm.codfarmacia, frm.nome, f.codice, f.descrizione, data, a.codice, a.nome, 0-20, 20-40, 40-60, over60, m.codmolecola, m.descrizione, c.codcasa, c.nome, fascia;

Tale interrogazione effettua una proiezione su tutti gli attributi che andranno inseriti nella tabella fatti e nelle tabelle dimensioni, eseguendo i vari join per rendere coerenti tra loro tutte le informazioni e raggruppando secondo le varie dimensioni.

Prima di questa interrogazione è stata necessaria un'operazione preliminare che convertisse le date di nascita dei pazienti in fasce di età. Per far questo ci si è appoggiati su una relazione ausiliaria:

- (CF, 0-20, 20-40, 40-60, Over60)

la quale è stata popolata estraendo le età dei pazienti tramite una sottrazione tra l'anno della ricetta e l'anno della data di nascita.

```
insert into pazientiaux cf,0-20 select distinct cf, year data - year datanascita
as anni from pazienti, ricette where cfpaziente = cf and year data - year datanascita
20;
```

```
insert into pazientiaux cf,20-40 select distinct cf, year data - year datanascita
as anni from pazienti, ricette where cfpaziente = cf and year data - year datanascita
40 and year data - year datanascita = 20;
```

```
insert into pazientiaux cf,40-60 select distinct cf, year data - year datanascita
```

```
as anni from pazienti, ricette where cfpaziente = cf and year data - year datanascita
60 and year data - year datanascita = 40;
```

```
insert into pazientiaux cf,over60 select distinct cf, year data - year datanascita
as anni from pazienti, ricette where cfpaziente = cf and year data - year datanascita
60;
```

In questo modo è stato possibile organizzare preventivamente i pazienti per fascia d'età.

Una volta estratti i dati della base di dati è stato possibile inserirli nel data warehouse tramite delle semplici interrogazioni

```
insert into farmaciadim codice, descrizione, codmolecola, descrizionemolecola,
codcasa, nomecasa, fascia select codice, descrizione, codmolecola, descrizionemolecola,
codcasa, nomecasa, fascia from aggrbuffer;
```

dove aggrbuffer è una vista relativa alla interrogazione per estrarre i dati. Questa stessa interrogazione è stata riutilizzata con le dovute modifiche per tutte le altre dimensioni.

Una volta popolate le dimensioni ed ottenute le chiavi (autoincrementanti) da inserire nella tabella dei fatti è stato possibile popolare quest'ultima sempre sfruttando la vista aggrbuffer, facendo una proiezione sugli attributi aggregati quantità ed importo e ricostruendo le chiavi da inserire nella tabella dei fatti tramite apposite tabelle di conversione.

3.2.3 Verifica del Data Warehouse

Una volta popolato il data warehouse sono state eseguite un paio di interrogazioni molto semplici per controllare se i dati forniti fossero coerenti con quelli della base di dati di partenza.

```
select sum importo as importo,f.nome from farmaciadim f, fattiprescrizioni
p where f.kfarmacia = p.kfarmacia and f.nome = Farmacia Rossi group by
f.nome
```

```
select sum importo ,f.descrizione from farmacidim f, fattiprescrizioni p
where f.kfarmaco = p.kfarmaco and f.descrizione = Aspirina group by f.descrizione
```

I risultati forniti da queste interrogazioni hanno confermato come il valori forniti dal datawarehouse fossero coerenti: infatti la Farmacia Bianchi aveva venduto farmaci per un totale di 55 euro ed erano state vendute 3 aspirine in tutte le farmacie per un totale di 10 euro. Tale risultato è stato ottenuto sia interrogando il data warehouse sia interrogando la base di dati.

3.3 Conclusioni

La costruzione del data warehouse ha permesso di separare le informazioni storiche per analisi statistiche (e non solo) dalla base di dati applicativa. Questo permette a chi ha il compito di fornire supporto decisionale, di poter interrogare la base di dati spesso con query complesse ed esose, senza dover minare le prestazioni della base di dati applicativa. Inoltre, una base di dati costruita ad hoc per il supporto delle decisioni permette di organizzare i dati in modo tale da semplificare le operazioni in lettura.

Appendice

Nella seguente appendice c'è il codice java utilizzato durante la sperimentazione sui livelli d'isolamento.

- `ConcurrentTest.java` programma che esegue due transazioni concorrenti che forzano il presentarsi di anomalie.

```
/*
 * ConcurrentTest.java
 * Created on 26 maggio 2007, 15.15
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */
package concurrentTest;
import java.sql.*;
import java.lang.Thread;
/**
 *
 * @author marco
 */
public class ConcurrentTest extends Thread{
    /**
     * Creates a new instance of ConcurrentTest
     */
    public ConcurrentTest() {
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Connection conn = getConnection();
        getMetaData(conn);
        try {
            conn.close();
        } catch (SQLException e){
            System.err.println(e.toString());
            System.err.println("SQL Error");
            System.exit(1);
        }
        ConcurrentTest t2 = new ConcurrentTest();
        t2.start();
        int mode;
        //Prova supporto livelli d'isolamento
        mode = Connection.TRANSACTION_REPEATABLE_READ;
        perditaAggiornamento(mode);
        mode = Connection.TRANSACTION_READ_COMMITTED;
        letturaSporca(mode);
        mode = Connection.TRANSACTION_REPEATABLE_READ;
        letturaInconsistente(mode);
        mode = Connection.TRANSACTION_REPEATABLE_READ;
        aggiornamentoFantasma(mode);
        mode = Connection.TRANSACTION_SERIALIZABLE;
        inserimentoFantasma(mode);
    }
    private static void perditaAggiornamento(int mode){
        try {
            Thread t = Thread.currentThread();
            System.out.println("PERDITA AGGIORNAMENTO T1");
            Connection conn1 = getConnection();
            conn1.setAutoCommit(false);
            System.out.println("T1 MODE: "+mode);
            conn1.setTransactionIsolation(mode);
            PreparedStatement stm =
            conn1.prepareStatement("SELECT prezzo FROM articoli" +
            " WHERE descrizione = ?");
            System.out.println("T1 : SELECT prezzo FROM" +
            " articoli WHERE descrizione = Scarpa");
            stm.setString(1,"Scarpa");
            ResultSet rs = stm.executeQuery();
            rs.next();
            int prezzol = rs.getInt("prezzo");
            System.out.println("PREZZO letto T1: "+prezzol);
            prezzol++;
            try {
                t.sleep(1500);
            } catch (InterruptedException e){
            }
            PreparedStatement stm5 =
            conn1.prepareStatement("UPDATE articoli SET prezzo" +
            "= ? WHERE descrizione = ?");
            System.out.println("T1: UPDATE articoli SET prezzo" +
            "= "+prezzol+" WHERE descrizione = 'Scarpa'");
            stm5.setInt(1,prezzol);
        }
    }
}
```

```

stm5.setString(2,"Scarpa");
stm5.execute();
conn1.commit();
//lettura finale
PreparedStatement stm3 =
conn1.prepareStatement("SELECT prezzo FROM articoli" +
    " WHERE descrizione = ?");
System.out.println("VERIFICA : SELECT prezzo FROM" +
    " articoli WHERE descrizione = Scarpa");
stm3.setString(1,"Scarpa");
rs = stm3.executeQuery();
rs.next();
int prezzo = rs.getInt("prezzo");
System.out.println("PREZZO letto VERIFICA: "+prezzo);
conn1.commit();
conn1.close();
} catch (SQLException e){
System.err.println(e.toString());
System.err.println("SQL Error");
System.exit(1);
}
}
private static void letturaSporca(int mode){
try {
Thread t = Thread.currentThread();
System.out.println("LETTURA SPORCA T1");
Connection conn1 = getConnection();
conn1.setAutoCommit(false);
System.out.println("T1 MODE: "+mode);
conn1.setTransactionIsolation(mode);
PreparedStatement stm =
conn1.prepareStatement("SELECT prezzo FROM articoli" +
    " WHERE descrizione = ?");
System.out.println("T1 : SELECT prezzo FROM articoli" +
    " WHERE descrizione = Scarpa");
stm.setString(1,"Scarpa");
ResultSet rs = stm.executeQuery();
rs.next();
int prezzo1 = rs.getInt("prezzo");
System.out.println("PREZZO letto T1: "+prezzo1);
System.out.println("incremento...");
prezzo1++;
PreparedStatement stm5 =
conn1.prepareStatement("UPDATE articoli SET prezzo" +
    " = ? WHERE descrizione = ?");
System.out.println("T1: UPDATE articoli SET prezzo" +
    " = "+prezzo1+" WHERE descrizione = 'Scarpa'");
stm5.setInt(1,prezzo1);
stm5.setString(2,"Scarpa");
stm5.execute();
try {
t.sleep(1500);
} catch (InterruptedException e){
}
conn1.rollback();
System.out.println("Abort...");
//lettura finale
PreparedStatement stm3 =
conn1.prepareStatement("SELECT prezzo FROM articoli" +
    " WHERE descrizione = ?");
System.out.println("VERIFICA : SELECT prezzo FROM articoli" +
    " WHERE descrizione = Scarpa");
stm3.setString(1,"Scarpa");
rs = stm3.executeQuery();
rs.next();
int prezzo = rs.getInt("prezzo");
System.out.println("PREZZO letto VERIFICA: "+prezzo);
conn1.commit();
conn1.close();
} catch (SQLException e){
System.err.println(e.toString());
System.err.println("SQL Error");
System.exit(1);
}
}
private static void letturaInconsistente(int mode){
try {

```

```
Thread t = Thread.currentThread();
System.out.println("LETTURA INCONSISTENTE T1");
Connection conn1 = getConnection();
conn1.setAutoCommit(false);
System.out.println("T1 MODE: "+mode);
conn1.setTransactionIsolation(mode);
PreparedStatement stm =
conn1.prepareStatement("SELECT prezzo FROM articoli" +
" WHERE descrizione = ?");
System.out.println("T1 : SELECT prezzo FROM articoli" +
" WHERE descrizione = Scarpa");
stm.setString(1, "Scarpa");
ResultSet rs = stm.executeQuery();
rs.next();
int prezzol = rs.getInt("prezzo");
System.out.println("PREZZO letto T1: "+prezzol);
try {
t.sleep(1500);
} catch (InterruptedException e){
}
PreparedStatement stm3 =
conn1.prepareStatement("SELECT prezzo FROM articoli" +
" WHERE descrizione = ?");
System.out.println("T1 : SELECT prezzo FROM articoli" +
" WHERE descrizione = Scarpa");
stm.setString(1, "Scarpa");
rs = stm.executeQuery();
rs.next();
prezzol = rs.getInt("prezzo");
System.out.println("PREZZO letto T1: "+prezzol);
conn1.commit();
conn1.close();
} catch (SQLException e){
System.err.println(e.toString());
System.err.println("SQL Error");
System.exit(1);
}
}
private static void aggiornamentoFantasma(int mode){
try {
Thread t = Thread.currentThread();
System.out.println("AGGIORNAMENTO FANTASMA T1");
Connection conn1 = getConnection();
conn1.setAutoCommit(false);
System.out.println("T1 MODE: "+mode);
conn1.setTransactionIsolation(mode);
PreparedStatement stm =
conn1.prepareStatement("SELECT prezzo FROM articoli" +
" WHERE descrizione = ?");
System.out.println("T1 : SELECT prezzo FROM articoli" +
" WHERE descrizione = Scarpa");
stm.setString(1, "Scarpa");
ResultSet rs = stm.executeQuery();
rs.next();
int prezzox1 = rs.getInt("prezzo");
System.out.println("PREZZO letto x T1: "+prezzox1);
try {
t.sleep(700);
} catch (InterruptedException e){
}
PreparedStatement stm3 =
conn1.prepareStatement("SELECT prezzo FROM articoli " +
"WHERE descrizione = ?");
System.out.println("T1: SELECT prezzo FROM articoli " +
"WHERE descrizione = Palla");
stm3.setString(1, "Palla");
rs = stm3.executeQuery();
rs.next();
int prezzoy1 = rs.getInt("prezzo");
System.out.println("PREZZO letto y T2: "+prezzoy1);
try {
t.sleep(2800);
} catch (InterruptedException e){
}
PreparedStatement stm7 =
conn1.prepareStatement("SELECT prezzo FROM articoli" +
" WHERE descrizione = ?");
```

```
System.out.println("T1: SELECT prezzo FROM articoli" +
    " WHERE descrizione = Orologio");
stm3.setString(1,"Orologio");
rs = stm3.executeQuery();
rs.next();
int prezzoz1 = rs.getInt("prezzo");
System.out.println("PREZZO letto z T1: "+prezzoz1);
conn1.commit();
int somma = prezzox1 + prezzoy1 + prezzoz1;
System.out.println("Somma T1: "+somma);
conn1.close();
} catch (SQLException e){
    System.err.println(e.toString());
    System.err.println("SQL Error");
    System.exit(1);
}
}
private static void inserimentoFantasma(int mode){
    try {
        Thread t = Thread.currentThread();
        System.out.println("INSERIMENTO FANTASMA T1");
        Connection conn1 = getConnection();
        conn1.setAutoCommit(false);
        System.out.println("T1 MODE: "+mode);
        conn1.setTransactionIsolation(mode);
        PreparedStatement stm =
        conn1.prepareStatement("SELECT count(codice) as num FROM articoli");
        System.out.println("T1 : SELECT count(codice) as num FROM articoli");
        ResultSet rs = stm.executeQuery();
        rs.next();
        int count = rs.getInt("num");
        System.out.println("Numero Articoli T1: "+count);
        try {
            t.sleep(1600);
        } catch (InterruptedException e){
        }
        PreparedStatement stm3 =
        conn1.prepareStatement("SELECT count(codice) as num FROM articoli");
        System.out.println("T1 : SELECT count(codice) as num FROM articoli");
        rs = stm.executeQuery();
        rs.next();
        count = rs.getInt("num");
        System.out.println("Numero Articoli T1: "+count);
        conn1.commit();
        conn1.close();
    } catch (SQLException e){
        System.err.println(e.toString());
        System.err.println("SQL Error");
        System.exit(1);
    }
}
private static void perditaAggiornamento2(int mode){
    try {
        Thread t = Thread.currentThread();
        System.out.println("PERDITA AGGIORNAMENTO T2");
        Connection conn2 = getConnection();
        conn2.setAutoCommit(false);
        System.out.println("T2 MODE: "+mode);
        conn2.setTransactionIsolation(mode);
        try {
            t.sleep(500);
        } catch (InterruptedException e){
        }
        PreparedStatement stm2 =
        conn2.prepareStatement("SELECT prezzo FROM articoli " +
            "WHERE descrizione = ?");
        System.out.println("T2: SELECT prezzo FROM articoli " +
            "WHERE descrizione = Scarpa");
        stm2.setString(1,"Scarpa");
        ResultSet rs = stm2.executeQuery();
        rs.next();
        int prezzo2 = rs.getInt("prezzo");
        System.out.println("PREZZO letto T2: "+prezzo2);
        prezzo2++;
        PreparedStatement stm4 =
```

```

        conn2.prepareStatement("UPDATE articoli SET prezzo " +
            "= ? WHERE descrizione = ?");
        System.out.println("T2: UPDATE articoli SET prezzo " +
            "= "+prezzo2+" WHERE descrizione = 'Scarpa'");
        stm4.setInt(1,prezzo2);
        stm4.setString(2,"Scarpa");
        stm4.execute();
        conn2.commit();
        conn2.close();
    } catch (SQLException e){
        System.err.println(e.toString());
        System.err.println("SQL Error");
        System.exit(1);
    }
}
}
private static void letturaSporca2(int mode){
    try {
        Thread t = Thread.currentThread();
        System.out.println("LETTURA SPORCA T2");
        Connection conn2 = getConnection();
        conn2.setAutoCommit(false);
        System.out.println("T2 MODE: "+mode);
        conn2.setTransactionIsolation(mode);
        try {
            t.sleep(500);
        } catch (InterruptedException e){
        }
        PreparedStatement stm2 =
        conn2.prepareStatement("SELECT prezzo FROM articoli " +
            "WHERE descrizione = ?");
        System.out.println("T2: SELECT prezzo FROM articoli " +
            "WHERE descrizione = Scarpa");
        stm2.setString(1,"Scarpa");
        ResultSet rs = stm2.executeQuery();
        rs.next();
        int prezzo2 = rs.getInt("prezzo");
        System.out.println("PREZZO letto T2: "+prezzo2);
        conn2.commit();
        conn2.close();
    } catch (SQLException e){
        System.err.println(e.toString());
        System.err.println("SQL Error");
        System.exit(1);
    }
}
}
private static void letturaInconsistente2(int mode){
    try {
        Thread t = Thread.currentThread();
        System.out.println("LETTURA INCONSISTENTE T2");
        Connection conn2 = getConnection();
        conn2.setAutoCommit(false);
        System.out.println("T2 MODE: "+mode);
        conn2.setTransactionIsolation(mode);
        try {
            t.sleep(500);
        } catch (InterruptedException e){
        }
        PreparedStatement stm2 =
        conn2.prepareStatement("SELECT prezzo FROM articoli " +
            "WHERE descrizione = ?");
        System.out.println("T2: SELECT prezzo FROM articoli " +
            "WHERE descrizione = Scarpa");
        stm2.setString(1,"Scarpa");
        ResultSet rs = stm2.executeQuery();
        rs.next();
        int prezzo2 = rs.getInt("prezzo");
        System.out.println("PREZZO letto T2: "+prezzo2);
        System.out.println("incremento...");
        prezzo2++;
        PreparedStatement stm5 =
        conn2.prepareStatement("UPDATE articoli SET prezzo " +
            "= ? WHERE descrizione = ?");
        System.out.println("T2: UPDATE articoli SET prezzo " +
            "= "+prezzo2+" WHERE descrizione = 'Scarpa'");
        stm5.setInt(1,prezzo2);
        stm5.setString(2,"Scarpa");
        stm5.execute();
    }
}
}

```

```

        conn2.commit();
        conn2.close();
    } catch (SQLException e){
        System.err.println(e.toString());
        System.err.println("SQL Error");
        System.exit(1);
    }
}
private static void aggiornamentoFantasma2(int mode){
    try {
        Thread t = Thread.currentThread();
        System.out.println("AGGIORNAMENTO FANTASMA T2");
        Connection conn2 = getConnection();
        conn2.setAutoCommit(false);
        System.out.println("T2 MODE: "+mode);
        conn2.setTransactionIsolation(mode);
        try {
            t.sleep(600);
        } catch (InterruptedException e){
        }
        PreparedStatement stm2 =
        conn2.prepareStatement("SELECT prezzo FROM articoli" +
            " WHERE descrizione = ?");
        System.out.println("T2: SELECT prezzo FROM articoli" +
            " WHERE descrizione = Palla");
        stm2.setString(1,"Palla");
        ResultSet rs = stm2.executeQuery();
        rs.next();
        int prezzoy2 = rs.getInt("prezzo");
        System.out.println("PREZZO letto y T2: "+prezzoy2);
        try {
            t.sleep(600);
        } catch (InterruptedException e){
        }
        System.out.println("decremento y di 1...");
        prezzoy2--;
        PreparedStatement stm4 =
        conn2.prepareStatement("SELECT prezzo FROM articoli " +
            "WHERE descrizione = ?");
        System.out.println("T2: SELECT prezzo FROM articoli " +
            "WHERE descrizione = Orologio");
        stm4.setString(1,"Orologio");
        rs = stm4.executeQuery();
        rs.next();
        int prezzoz2 = rs.getInt("prezzo");
        System.out.println("PREZZO letto z T2: "+prezzoz2);
        System.out.println("incremento z di 1...");
        prezzoz2++;
        PreparedStatement stm5 =
        conn2.prepareStatement("UPDATE articoli SET prezzo" +
            " = ? WHERE descrizione = ?");
        stm5.setInt(1,prezzoy2);
        stm5.setString(2,"Palla");
        stm5.execute();
        System.out.println("T2: UPDATE articoli SET prezzo" +
            " = "+prezzoy2+" WHERE descrizione = 'Palla'");
        PreparedStatement stm6 =
        conn2.prepareStatement("UPDATE articoli SET prezzo" +
            " = ? WHERE descrizione = ?");
        System.out.println("T2: UPDATE articoli SET prezzo" +
            " = "+prezzoz2+" WHERE descrizione = 'Orologio'");
        stm5.setInt(1,prezzoz2);
        stm5.setString(2,"Orologio");
        stm5.execute();
        conn2.commit();
        conn2.close();
    } catch (SQLException e){
        System.err.println(e.toString());
        System.err.println("SQL Error");
        System.exit(1);
    }
}
private static void inserimentoFantasma2(int mode){
    try {
        Thread t = Thread.currentThread();

```

```

        System.out.println("INSERIMENTO FANTASMA T2");
        Connection conn2 = getConnection();
        conn2.setAutoCommit(false);
        System.out.println("T2 MODE: "+mode);
        conn2.setTransactionIsolation(mode);
        try {
            t.sleep(500);
        } catch (InterruptedException e){
        }
        PreparedStatement stm2 =
        conn2.prepareStatement("INSERT INTO articoli VALUES (?, ?, ?, ?)");
        System.out.println("T2: INSERT INTO articoli " +
            "VALUES (10004, Notebook, 25, 0)");
        stm2.setInt(1, 10004);
        stm2.setString(2, "Notebook");
        stm2.setInt(3, 25);
        stm2.setInt(4, 0);
        stm2.execute();
        conn2.commit();
        conn2.close();
    } catch (SQLException e){
        System.err.println(e.toString());
        System.err.println("SQL Error");
        System.exit(1);
    }
}
}
private static Connection getConnection(){
    Connection conn1 = null;
    try{
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
        conn1 = DriverManager.getConnection("jdbc:db2:tbd");
    } catch (Exception e) {
        System.out.println(e.toString());
        System.out.println("Connection failed");
        System.exit(1);
    }
    System.out.println("Connection successfull");
    return conn1;
}
private static void getMetaData(Connection conn){
    try{
        DatabaseMetaData dbMetaData = conn.getMetaData();
        if(dbMetaData.supportsTransactions())
            System.out.println("Transazioni supportate");
        if(dbMetaData.supportsTransactionIsolationLevel(
            Connection.TRANSACTION_READ_UNCOMMITTED))
            System.out.println("READ UNCOMMITTED supportato");
        if(dbMetaData.supportsTransactionIsolationLevel(
            Connection.TRANSACTION_READ_COMMITTED))
            System.out.println("READ COMMITTED supportato");
        if(dbMetaData.supportsTransactionIsolationLevel(
            Connection.TRANSACTION_REPEATABLE_READ))
            System.out.println("REPEATABLE READ supportato");
        if(dbMetaData.supportsTransactionIsolationLevel(
            Connection.TRANSACTION_SERIALIZABLE))
            System.out.println("SERIALIZABLE supportato");
    } catch (SQLException e){
        System.err.println("Errore lettura metadati");
    }
}
}
public void run(){
    int mode;
    mode = Connection.TRANSACTION_REPEATABLE_READ;
    perditaAggiornamento2(mode);
    mode = Connection.TRANSACTION_READ_COMMITTED;
    letturaSporca2(mode);
    mode = Connection.TRANSACTION_REPEATABLE_READ;
    letturaInconsistente2(mode);
    mode = Connection.TRANSACTION_REPEATABLE_READ;
    aggiornamentoFantasma2(mode);
    mode = Connection.TRANSACTION_REPEATABLE_READ;
    inserimentoFantasma2(mode);
}
}
}

```