Barbagallo Valerio
Da Lozzo Giordano
Mellini Giampiero

PUNTO 1

Db usato: DB2 express-C versione 9

Successivamente è riportato l'sql per la creazione della base di dati.

```
create table CLIENTE(
nome char(20),
cognome varchar(255),
id integer primary key not null
);
create table ORDINE(
id integer primary key not null,
id_cliente integer references CLIENTE (id),
stato integer
);
create table LINEA_ORDINE(
id_ordine integer references ORDINE(id),
prodotto char(20),
quantita integer
);
```

Riportiamo anche l'sql utilizzato per la creazione degli indici, attività che si è resa necessaria fare manualmente dato che db2 che creando indici con gli strumenti di db2 venivano anche automaticamente aggiornate le statistiche (giustamente per il corretto utilizzo degli indici).

```
create index cliente_indice__nome on CLIENTE (nome);
create index linea_ordine_indice_quantita on LINEA_ORDINE (quantita);
```

E si riporta anche l'sql utilizzato da db2 per l'aggiornamento delle statistiche sulle tabelle.

```
RUNSTATS ON TABLE ERROR0.CLIENTE ON ALL COLUMNS ALLOW WRITE ACCESS ;
```

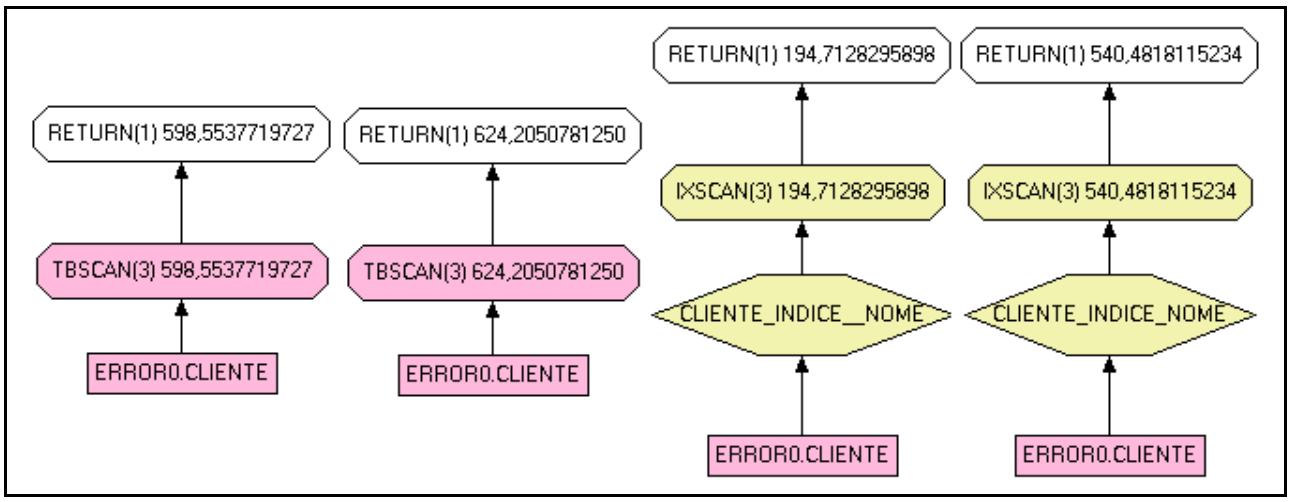
Abbiamo cosi popolato un database con un programma java, con circa 50 000 tuple per la relazione cliente e 500 000 tuple per le relazioni ordine e linea_ordine. Da questo database si è successivamente eseguito un backup e successivamente un ripristino di tale backup in 3 nuovi

database, in modo tale da poter lavorare contemporaneamente su database con stessi dati, ma diversi per statistiche aggiornate e presenza di indici, come segue:

1. database senza statistiche senza indici
2. database con statistiche senza indici
3. database senza statistiche con indici
4. database con statistiche con indici

QUERY 1

- select nome from CLIENTE;



nota: con le statistiche aumenta il tempo espresso in timeron, questo perchè senza statistiche aggiornate, ed essendo il tempo espresso dal piano di esecuzione ipoteticamente calcolato sulle statistiche, non è affidabile

nota2: nel caso della presenza di indici viene ovviamente scelto di utilizzare gli indici. Si nota come non viene neanch effettuato il fetch della tupla, dato che il risultato richiesto può essere semplicemente ottenuto dal valore del parametro utilizzato per indicizzare.

nota3: se piuttosto di una select nome, avessimo fatto una select *, anche in presenza di indici sarebbe comunque stato fatto il TBSCAN di tutta la tabella, e con lo stesso tempo stimato del relativo database senza indici (relativamente alle statistiche se aggiornate o meno).

QUERY 2

select nome
from cliente, linea_ordine
where nome = prodotto;

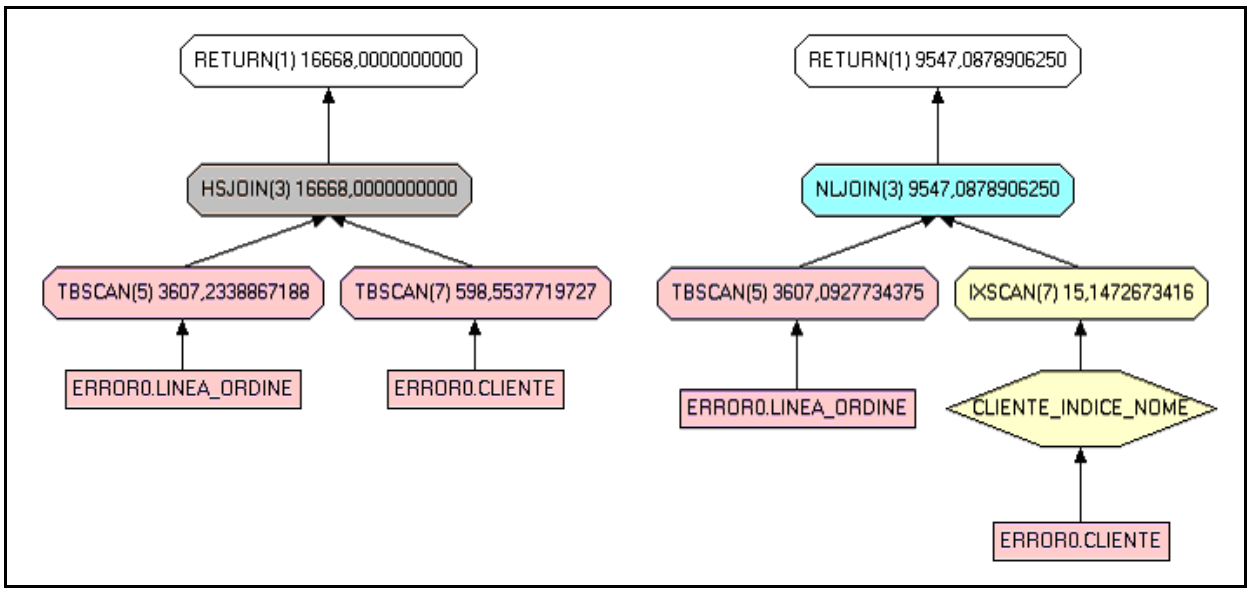


Figura 2.1 Figura 2.2

La figura 2.1 mostra il piano di accesso nel caso di assenza di indici sulla base di dati e con statistiche aggiornate e non. L'assenza dell'indice e di informazioni sull'ordinamento delle relazioni porta il dbms a preferire l'*hash-join.* Il tempo di esecuzione atteso decresce con la presenza di statistiche aggiornate, senza le quali, le dimensioni delle relazioni sono sovrastimate.

La figura 2.2 mostra, invece, l'esecuzione in presenza di un indice sull'attributo *nome* della tabella *cliente.* Le dimensioni delle tabelle suggeriscono l'impiego in un join di tipo *nested-loop.* Come nel caso precedente il piano non cambia con l'aggiornamento delle statistiche, confermando la stima per eccesso delle cardinalità delle relazioni che conferma l'utilità dell'utilizzo dell'indice, ma migliorano le sue prestazioni attese.

QUERY 3

select quantita

from ordine,linea_ordine

where id=id_ordine;

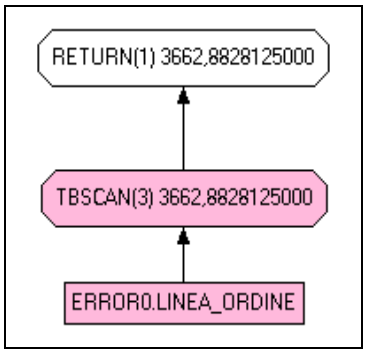


Figura 3

L'interrogazione mostra come la presenza dell'indice sull'attributo *id* della relazione *ordine* e di statistiche aggiornate non abbiano alcun effetto, fatta eccezione per la valutazione dei costi, sul piano d'accesso. Il vincolo di chiave esterna garantisce infatti che la scansione della tabella linea_ordine e la verifica "is not null" sul attributo id_ordine equivalga logicamente all'operazione di join.

QUERY 4

select quantita

from ordine, linea_ordine

where id=quantita

order by id;

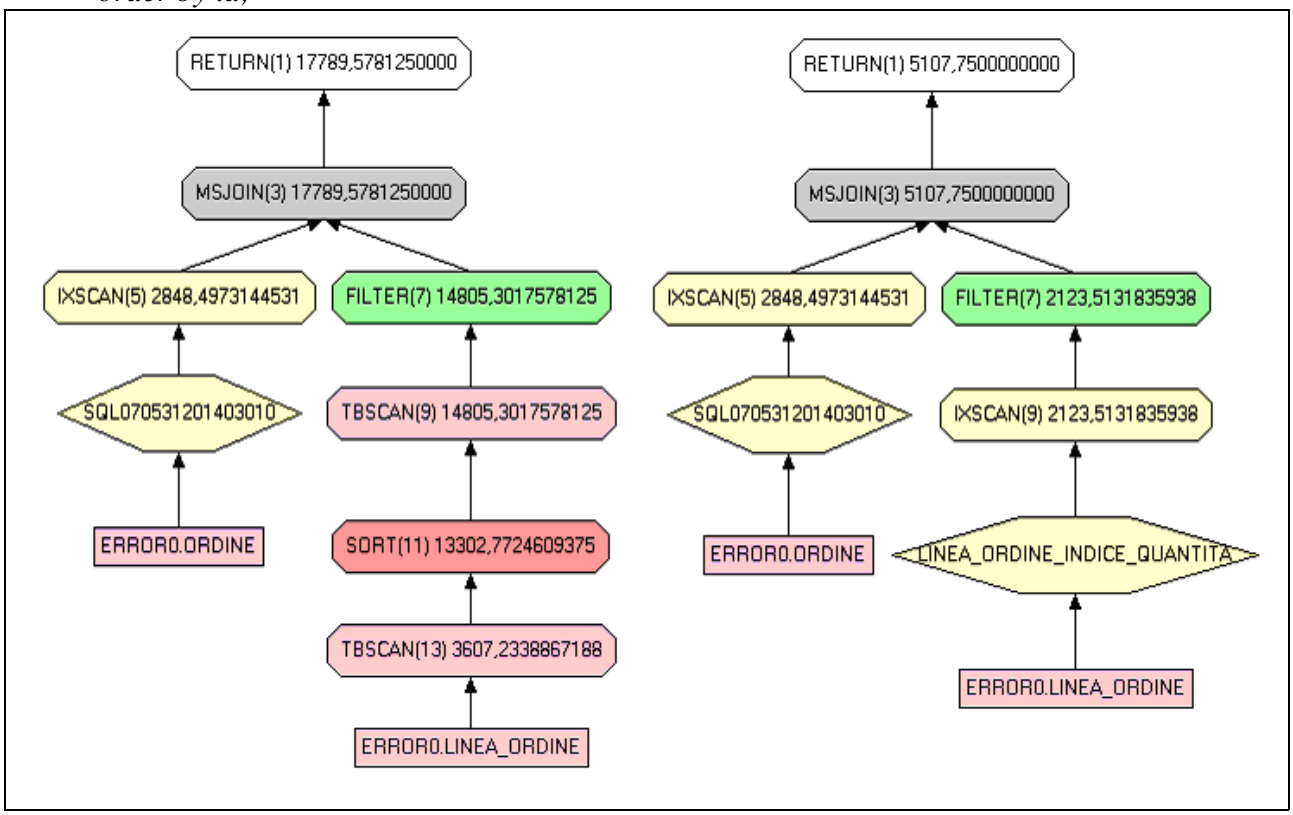


Figura 4.1 Figura 4.2

La figura 4.1 mostra il piano di accesso nel caso di assenza di indici sulla base di dati e con statistiche aggiornate e non.

La figura 4.2 mostra, invece, l'esecuzione in presenza di un indice sull'attributo *quanita* della tabella *linea_ordine.*

Si evidenzia :

- in entrambi i casi la necessità di ordinare il risultato sulla base dell' id dell'ordine porti a sfruttare l'indice definito automaticamente da DB2 su tale campo (chiave primaria), che risulta già ordinato su tale campo.
- l'input ordinato dell'indice e la necessità di un risultato ordinato porti ad utilizzare quale tecnica di join il merge scan e quindi comporta:
    - nel primo caso, l'esecuzione di un preventivo ordinamento della tabella linea ordine sull'attributo quantità;
    - nel secondo, l'impiego dell'indice (già ordinato) sull'attributo quantià.

Nuovamente, l'impiego delle statistiche aggiornate ridimensiona i costi attesi, ma la sovrastima della cardinalità delle relazioni riafferma l'utilizzo dell'indice.

QUERY 5

select quantita
from cliente, ordine, linea_ordine
where nome = prodotto and stato = quantita
group by quantita

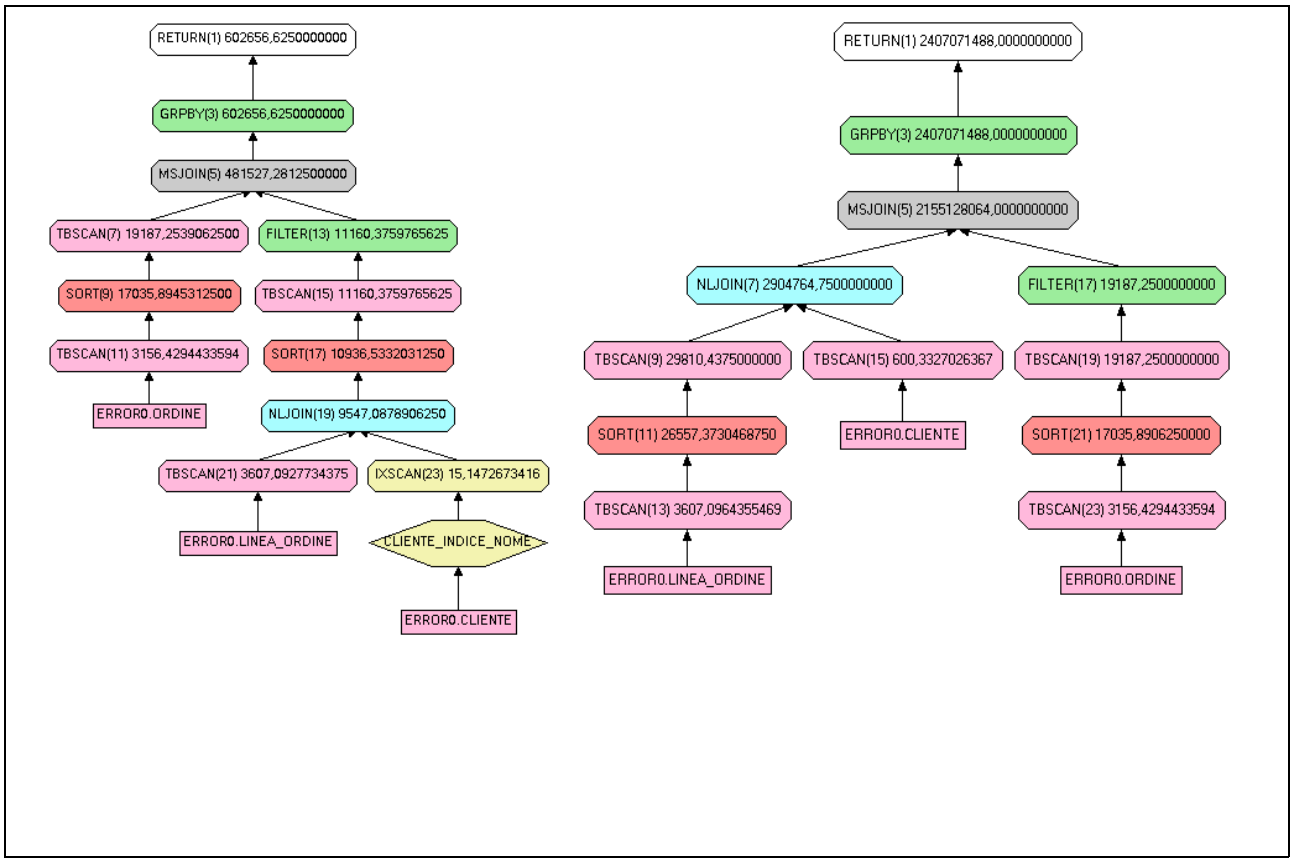


Figura 5.1 Figura 5.2

La necessità di raggruppamento sul campo quantità porta ad eseguire in tutti i piani di esecuzione un ordinamento sullo stato dell'ordine e sulla quantità delle linee ordine e quindi prepara un risultato intermedio adatto ad un un *merge-scan join*.

E' evidente che i piani di esecuzione hanno una struttura complementare, infatti:

- il join tra le linee ordine e i clienti utilizza in ogni caso un *nested loop* con la differenza sostanziale che la presenza di statistiche aggiornate suggerisce l'applicazione dell'indice, quando presente, sul nome del cliente. Si noti, in particolare, la diversa strategia che conduce ad un risultato ordinato del join; si ha, di fatto, che l'ordinamento sullo stato dell'ordine segue l'esecuzione del join con la presenza dell'indice perché la sua selettività fa prevedere un risultato intermedio di dimensioni inferiori a quella della tabella esterna, mentre la sua assenza anticipa l'ordinamento per migliorare le prestazione del nested loop nella scansione interna delle linee ordine (che può terminare senza scandire l'intera tabella).
- l'esecuzione dell'ordinamento sulla base dello stato della relazione ordine in ingresso all'algoritmo di merge, a meno di considerazioni sui costi dovuti alle statistiche, non differisce e svolge il medesimo ruolo in tutti i piani.

PUNTO 2

Dbms utilizzato:
db2 express-c

Soluzione attuata:
abbiamo eseguito transazioni concorrentemente, sincronizzandole per ottenere l'anomalia richiesta, nello stesso modo con cui sono state presentate a lezione. In particolare abbiamo utilizzato jdbc, usando una diversa Connection per ogni transazione, e settando il parametro Connection.setAutoCommint(false). Questo implica che ogni query non è più ritenuta una transazione atomica, ma parte di una transazione che deve essere gestita manualmente (da software) con il commit o il rollback.
Abbiamo poi impostato un timeout sui lock, dato che db2 ha di default un timeout a infinito con st.execute("SET CURRENT LOCK TIMEOUT WAIT 3") per evitare lo stallo delle transazioni, e che i lock restassero sempre attivi sulla base di dati.
Ulteriormente le istruzioni che causavano lo stallo, e che ritornavano un eccezione settando il timeout, sono state racchiuse da blocchi try catch, in modo tale da gestire tale eccezione e stampare un apposito messaggio di errore.

Risultati del test:
abbiamo visto come db2 usa di default il livello di isolamento read committed.
Oltretutto abbiamo visto come i livelli di isolamento non evitano le anomalie come dovrebbe essere in teoria. Ovvero read uncommitted non offre alcuna protezione, mentre read committed vieta solamente la lettura sporca. Per serializable e per repeatable read, invece, è come la teoria, ossia tutte le anomalie sono vietate con serializable, e solo l'inserimento fantasma è permesso con repeatable read.

Alleghiamo di seguito l'output del codice usato per testare la base di dati. Sono evidenziate in rosso le eccezioni, ossia anomalia proibita. In verde invece vengono evidenziate le anomalie non vietate.

```
INIZIO
------------------------------
INIZIALIZZAZIONE TABELLA...
nome = aydcmqwuuku
fine crezione connessione
eseguito l'insert
...INIZIALIZZAZIONE TABELLA COMPLETATA
------------------------------
INIZIALIZZAZIONE TABELLA...
nome = akgywkyavxdyk
fine crezione connessione
eseguito l'insert
```

```
...INIZIALIZZAZIONE TABELLA COMPLETATA
------------------------------
------------------------------

READ UNCOMMITTED

------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
t1 legge x = 4000
t2 legge x = 4000
x=x+5
t1 scrive x = 4005
t1 eseguito il commit
x=x+10
t2 scrive x = 4010
t2 eseguito il commit
alla fine leggo x = 4010
...FINE TEST PERDITA DI AGGIORNAMENTO
------------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 4010
x=x+5
t1 scrive x = 4015
t2 legge x = 4015
t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 4010
...FINE TEST LETTURA SPORCA
------------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 4010
t2 legge x = 4010
x=x+5
t2 scrive x = 4015
t2 esegue il commit
t2 chiude la connessione
t1 legge x = 4015
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 4015
...FINE TEST LETTURA INCONSISTENTE
------------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 4015
t2 legge y = 4015
y = y-100
t2 legge z = 4000
z = z+100
t2 scrive y = 3915
t2 scrive z = 4100
t2 esegue il commit
t1 legge z = 4100
t1 esegue il commit
alla fine leggo y = 3915 e z = 4100
...FINE TEST AGGIORNAMENTO FANTASMA
------------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 60
t2 aggiunge una tupla
t2 esegue il commit
t1 conta tuple totali = 61
t1 esegue il commit
alla fine leggo tuple totali = 61
...FINE TEST INSERIMENTO FANTASMA
------------------------------
------------------------------

READ COMMITTED
```

```
------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
t1 legge x = 3915
t2 legge x = 3915
x=x+5
t1 scrive x = 3920
t1 eseguito il commit
x=x+10
t2 scrive x = 3925
t2 eseguito il commit
alla fine leggo x = 3925
...FINE TEST PERDITA DI AGGIORNAMENTO
------------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 3925
x=x+5
t1 scrive x = 3930

t2 lettura sporca impedita: [IBM][CLI Driver][DB2/NT] SQL0911N La transazione corrente è stata
annullata a causa di un deadlock o timeout. Codice di origine errore "68". SQLSTATE=40001

t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 3925
...FINE TEST LETTURA SPORCA
------------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 3925
t2 legge x = 3925
x=x+5
t2 scrive x = 3930
t2 esegue il commit
t2 chiude la connessione
t1 legge x = 3930
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 3930
...FINE TEST LETTURA INCONSISTENTE
------------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 3930
t2 legge y = 3930
y = y-100
t2 legge z = 4100
z = z+100
t2 scrive y = 3830
t2 scrive z = 4200
t2 esegue il commit
t1 legge z = 4200
t1 esegue il commit
alla fine leggo y = 3830 e z = 4200
...FINE TEST AGGIORNAMENTO FANTASMA
------------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 61
t2 aggiunge una tupla
t2 esegue il commit
t1 conta tuple totali = 62
t1 esegue il commit
alla fine leggo tuple totali = 62
...FINE TEST INSERIMENTO FANTASMA
------------------------------
------------------------------

REPEATABLE READ
```

```
------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
t1 legge x = 3830
t2 legge x = 3830
x=x+5
```

t1 scrittura bloccata: perdita di aggiornamento evitata [IBM][CLI Driver][DB2/NT] SQL0911N La transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore "68". SQLSTATE=40001

```
x=x+10
t2 scrive x = 3840
t2 eseguito il commit
alla fine leggo x = 3840
...FINE TEST PERDITA DI AGGIORNAMENTO
------------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 3840
x=x+5
t1 scrive x = 3845
```

t2 lettura sporca impedita: [IBM][CLI Driver][DB2/NT] SQL0911N La transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore "68". SQLSTATE=40001

```
t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 3840
...FINE TEST LETTURA SPORCA
------------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 3840
t2 legge x = 3840
x=x+5
```

t2 scrittura vietata, lettura inconsistente impedita: [IBM][CLI Driver][DB2/NT] SQL0911N La transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore "68". SQLSTATE=40001

```
t2 chiude la connessione
t1 legge x = 3840
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 3840
...FINE TEST LETTURA INCONSISTENTE
------------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 3840
t2 legge y = 3840
y = y-100
t2 legge z = 4200
z = z+100
```

t2 scrittura vietata, aggiornamento fantasma vietato: [IBM][CLI Driver][DB2/NT] SQL0911N La transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore "68". SQLSTATE=40001

```
t1 legge z = 4200
t1 esegue il commit
alla fine leggo y = 3840 e z = 4200
...FINE TEST AGGIORNAMENTO FANTASMA
------------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 62
t2 aggiunge una tupla
t2 esegue il commit
```

```
t1 conta tuple totali = 63
t1 esegue il commit
alla fine leggo tuple totali = 63
...FINE TEST INSERIMENTO FANTASMA
------------------------------
------------------------------


SERIALIZABLE


------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
t1 legge x = 3840
t2 legge x = 3840
x=x+5


t1 scrittura bloccata: perdita di aggiornamento evitata [IBM][CLI Driver][DB2/NT] SQL0911N La
transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore
"68". SQLSTATE=40001


x=x+10
t2 scrive x = 3850
t2 eseguito il commit
alla fine leggo x = 3850
...FINE TEST PERDITA DI AGGIORNAMENTO
------------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 3850
x=x+5
t1 scrive x = 3855


t2 lettura sporca impedita: [IBM][CLI Driver][DB2/NT] SQL0911N La transazione corrente è stata
annullata a causa di un deadlock o timeout. Codice di origine errore "68". SQLSTATE=40001


t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 3850
...FINE TEST LETTURA SPORCA
------------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 3850
t2 legge x = 3850
x=x+5


t2 scrittura vietata, lettura inconsistente impedita: [IBM][CLI Driver][DB2/NT] SQL0911N La
transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore
"68". SQLSTATE=40001


t2 chiude la connessione
t1 legge x = 3850
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 3850
...FINE TEST LETTURA INCONSISTENTE
------------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 3850
t2 legge y = 3850
y = y-100
t2 legge z = 4200
z = z+100


t2 scrittura vietata, aggiornamento fantasma vietato: [IBM][CLI Driver][DB2/NT] SQL0911N La
transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore
"68". SQLSTATE=40001
```

```
t1 legge z = 4200
t1 esegue il commit
alla fine leggo y = 3850 e z = 4200
...FINE TEST AGGIORNAMENTO FANTASMA
-----------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 63

t2 inserimento bloccato, inserimento fantasma vietato: [IBM][CLI Driver][DB2/NT] SQL0911N La
transazione corrente è stata annullata a causa di un deadlock o timeout. Codice di origine errore
"68". SQLSTATE=40001

t1 conta tuple totali = 63
t1 esegue il commit
alla fine leggo tuple totali = 63
...FINE TEST INSERIMENTO FANTASMA
-----------------------------
FINE
```

Abbiamo poi eseguito lo stesso test anche su postgresql.

Dbms utilizzato:
postgresql

Soluzione attuata:
abbiamo utilizzato lo stesso codice di test, con piccole modifiche, per testare i livelli di isolamento di postgresql. A differenza di db2, non è necessario settare il timeout, e gestire le eccezioni, tranne nel caso della perdita di aggiornamento, unica anomalia in cui la scrittura è bloccata.

Risultati del test:
postgresql utilizza solamente 2 livelli di isolamento, ossia utilizzare read uncommitted ha gli stessi effetti di read committed, mentre repeatable read si comporta come serializable. Quindi il 2° livello di isolamento vieta qualsiasi anomalia, mentre il primo livello vieta solamente la lettura sporca. Ossia read committed si comporta allo stesso modo che in db2. A differenza di questultimo si può notare come, eccetto per la perdita di aggiornamento dove l'integrità del db deve essere garantita e la scrittura viene bloccata ritornando un eccezione, in tutti gli altri casi postgre usa un'ottimizzazione del gestore della concorrenza, dove viene mantenuto il penultimo valore dell'area di memoria utilizzata dalle transazioni, in modo da restituire questo "vecchio" valore, invece che l'ultimo, in modo da far figurare la lettura della transazione, come se fosse avvenuta prima, e quindi non causando nessun conflitto.

Alleghiamo di seguito l'output del codice usato per testare la base di dati. Sono evidenziate in rosso le eccezioni, ossia anomalia proibita. In verde invece vengono evidenziate le anomalie non vietate.

```
INIZIO
-----------------------------
INIZIALIZZAZIONE TABELLA...
nome = yeouir
fine crezione connessione
eseguito l'insert
```

```
...INIZIALIZZAZIONE TABELLA COMPLETATA
------------------------------
INIZIALIZZAZIONE TABELLA...
nome = yxjmkmhm
fine crezione connessione
eseguito l'insert
...INIZIALIZZAZIONE TABELLA COMPLETATA
------------------------------
------------------------------


READ UNCOMMITTED


------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
t1 legge x = 4000
t2 legge x = 4000
x=x+5
t1 scrive x = 4005
t1 eseguito il commit
x=x+10
t2 scrive x = 4010
t2 eseguito il commit
alla fine leggo x = 4010
...FINE TEST PERDITA DI AGGIORNAMENTO
------------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 4010
x=x+5
t1 scrive x = 4015
t2 legge x = 4010
t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 4010
...FINE TEST LETTURA SPORCA
------------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 4010
t2 legge x = 4010
x=x+5
t2 scrive x = 4015
t2 esegue il commit
t2 chiude la connessione
t1 legge x = 4015
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 4015
...FINE TEST LETTURA INCONSISTENTE
------------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 4015
t2 legge y = 4015
y = y-100
t2 legge z = 4000
z = z+100
t2 scrive y = 3915
t2 scrive z = 4100
t2 esegue il commit
t1 legge z = 4100
t1 esegue il commit
alla fine leggo y = 3915 e z = 4100
...FINE TEST AGGIORNAMENTO FANTASMA
------------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 24
t2 aggiunge una tupla
t2 esegue il commit
t1 conta tuple totali = 25
t1 esegue il commit
alla fine leggo tuple totali = 25
```

```
...FINE TEST INSERIMENTO FANTASMA
------------------------------
------------------------------


READ COMMITTED


------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
t1 legge x = 3915
t2 legge x = 3915
x=x+5
t1 scrive x = 3920
t1 eseguito il commit
x=x+10
t2 scrive x = 3925
t2 eseguito il commit
alla fine leggo x = 3925
...FINE TEST PERDITA DI AGGIORNAMENTO
------------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 3925
x=x+5
t1 scrive x = 3930
t2 legge x = 3925
t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 3925
...FINE TEST LETTURA SPORCA
------------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 3925
t2 legge x = 3925
x=x+5
t2 scrive x = 3930
t2 esegue il commit
t2 chiude la connessione
t1 legge x = 3930
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 3930
...FINE TEST LETTURA INCONSISTENTE
------------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 3930
t2 legge y = 3930
y = y-100
t2 legge z = 4100
z = z+100
t2 scrive y = 3830
t2 scrive z = 4200
t2 esegue il commit
t1 legge z = 4200
t1 esegue il commit
alla fine leggo y = 3830 e z = 4200
...FINE TEST AGGIORNAMENTO FANTASMA
------------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 25
t2 aggiunge una tupla
t2 esegue il commit
t1 conta tuple totali = 26
t1 esegue il commit
alla fine leggo tuple totali = 26
...FINE TEST INSERIMENTO FANTASMA
------------------------------
------------------------------


REPEATABLE READ
```

```
------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
t1 legge x = 3830
t2 legge x = 3830
x=x+5
t1 scrive x = 3835
t1 eseguito il commit
x=x+10


t2 scrittura bloccata: perdita di aggiornamento evitata ERROR: could not serialize access due to
concurrent update
alla fine leggo x = 3835
...FINE TEST PERDITA DI AGGIORNAMENTO
------------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 3835
x=x+5
t1 scrive x = 3840
t2 legge x = 3835
t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 3835
...FINE TEST LETTURA SPORCA
------------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 3835
t2 legge x = 3835
x=x+5
t2 scrive x = 3840
t2 esegue il commit
t2 chiude la connessione
t1 legge x = 3835
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 3840
...FINE TEST LETTURA INCONSISTENTE
------------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 3840
t2 legge y = 3840
y = y-100
t2 legge z = 4200
z = z+100
t2 scrive y = 3740
t2 scrive z = 4300
t2 esegue il commit
t1 legge z = 4200
t1 esegue il commit
alla fine leggo y = 3740 e z = 4300
...FINE TEST AGGIORNAMENTO FANTASMA
------------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 26
t2 aggiunge una tupla
t2 esegue il commit
t1 conta tuple totali = 26
t1 esegue il commit
alla fine leggo tuple totali = 27
...FINE TEST INSERIMENTO FANTASMA
------------------------------
------------------------------


SERIALIZABLE


------------------------------
------------------------------
INIZIO TEST PERDITA DI AGGIORNAMENTO...
```

```
t1 legge x = 3740
t2 legge x = 3740
x=x+5
t1 scrive x = 3745
t1 eseguito il commit
x=x+10


t2 scrittura bloccata: perdita di aggiornamento evitata ERROR: could not serialize access due to
concurrent update
alla fine leggo x = 3745
...FINE TEST PERDITA DI AGGIORNAMENTO
-----------------------------
INIZIO TEST LETTURA SPORCA...
t1 legge x = 3745
x=x+5
t1 scrive x = 3750
t2 legge x = 3745
t1 esegue l'abort\rollback
t2 esegue il commit
alla fine leggo x = 3745
...FINE TEST LETTURA SPORCA
-----------------------------
INIZIO TEST LETTURA INCONSISTENTE...
t1 legge x = 3745
t2 legge x = 3745
x=x+5
t2 scrive x = 3750
t2 esegue il commit
t2 chiude la connessione
t1 legge x = 3745
t1 esegue esegue il commit
t1 chiude la connessione
alla fine leggo x = 3750
...FINE TEST LETTURA INCONSISTENTE
-----------------------------
INIZIO TEST AGGIORNAMENTO FANTASMA...
t1 legge y = 3750
t2 legge y = 3750
y = y-100
t2 legge z = 4300
z = z+100
t2 scrive y = 3650
t2 scrive z = 4400
t2 esegue il commit
t1 legge z = 4300
t1 esegue il commit
alla fine leggo y = 3650 e z = 4400
...FINE TEST AGGIORNAMENTO FANTASMA
-----------------------------
INIZIO TEST INSERIMENTO FANTASMA...
t1 conta tuple totali = 27
t2 aggiunge una tupla
t2 esegue il commit
t1 conta tuple totali = 27
t1 esegue il commit
alla fine leggo tuple totali = 28
...FINE TEST INSERIMENTO FANTASMA
-----------------------------
FINE
```

Di seguito in riportiamo il codice utilizzato per testare i livelli di isolamento del DBMS. In rosso sono riportati i nomi delle classi per facilitarne la lettura.

```
public class Tester {

    private static String url = "jdbc:db2:TEST";
    private static String user = "";
```

```
    private static String password = "";
    private static String driver = "COM.ibm.db2.jdbc.app.DB2Driver";
    private static String nome = "atzeni";
       private static String nome2 = "";

       public static void main(String []arg){
        System.out.println("INIZIO");
        int livelloIsolamento = Connection.TRANSACTION_READ_UNCOMMITTED;
        nome = inizializzaTabella();
        nome2 = inizializzaTabella();
        System.out.println("-----------------------------");
        System.out.println("-----------------------------");
        System.out.println("\n\tREAD UNCOMMITTED\n");
        System.out.println("-----------------------------");
        testPerditaDiAggiornamento(livelloIsolamento);
        testLetturaSporca(livelloIsolamento);
        testLetturaInconsistente(livelloIsolamento);
        testAggiornamentoFantasma(livelloIsolamento);
        testInserimentoFantasma(livelloIsolamento);
        livelloIsolamento = Connection.TRANSACTION_READ_COMMITTED;
        System.out.println("-----------------------------");
        System.out.println("-----------------------------");
        System.out.println("\n\tREAD COMMITTED\n");
        System.out.println("-----------------------------");
        //inizializzaTabella();
        testPerditaDiAggiornamento(livelloIsolamento);
        testLetturaSporca(livelloIsolamento);
        testLetturaInconsistente(livelloIsolamento);
        testAggiornamentoFantasma(livelloIsolamento);
        testInserimentoFantasma(livelloIsolamento);
        livelloIsolamento = Connection.TRANSACTION_REPEATABLE_READ;
        System.out.println("-----------------------------");
        System.out.println("-----------------------------");
        System.out.println("\n\tREPEATABLE READ\n");
        System.out.println("-----------------------------");
        //inizializzaTabella();
        testPerditaDiAggiornamento(livelloIsolamento);
        testLetturaSporca(livelloIsolamento);
        testLetturaInconsistente(livelloIsolamento);
        testAggiornamentoFantasma(livelloIsolamento);
        testInserimentoFantasma(livelloIsolamento);
        livelloIsolamento = Connection.TRANSACTION_SERIALIZABLE;
        System.out.println("-----------------------------");
        System.out.println("-----------------------------");
        System.out.println("\n\tSERIALIZABLE\n");
        System.out.println("-----------------------------");
        //inizializzaTabella();
        testPerditaDiAggiornamento(livelloIsolamento);
        testLetturaSporca(livelloIsolamento);
        testLetturaInconsistente(livelloIsolamento);
        testAggiornamentoFantasma(livelloIsolamento);
        testInserimentoFantasma(livelloIsolamento);
        System.out.println("-----------------------------");
               System.out.println("FINE");
       }


       private static void testInserimentoFantasma(int livelloIsolamento) {
               InserimentoFantasmaTester tester = new
InserimentoFantasmaTester(driver,url,user,password);
        try {
            tester.eseguiTest(livelloIsolamento);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
       }


       private static void testAggiornamentoFantasma(int livelloIsolamento) {
```

```
              AggiornamentoFantasmaTester tester = new
AggiornamentoFantasmaTester(driver,url,user,password,nome,nome2);
        try {
            tester.eseguiTest(livelloIsolamento);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
       }


       private static void testLetturaInconsistente(int livelloIsolamento) {
        LetturaInconsistenteTester tester = new
LetturaInconsistenteTester(driver,url,user,password,nome);
        try {
            tester.eseguiTest(livelloIsolamento);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    private static void testLetturaSporca(int livelloIsolamento) {
        LetturaSporcaTester tester = new LetturaSporcaTester(driver,url,user,password,nome);
        try {
            tester.eseguiTest(livelloIsolamento);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    private static void testPerditaDiAggiornamento(int livelloIsolamento) {
              PerditaDiAggiornamentoTester p1 = new
PerditaDiAggiornamentoTester(driver,url,user,password,nome);
              try {
                     p1.eseguiTest(livelloIsolamento);
              } catch (ClassNotFoundException e) {
                     e.printStackTrace();
              } catch (SQLException e) {
                     e.printStackTrace();
              } catch (InterruptedException e) {
                     e.printStackTrace();
              }
       }

       private static String inizializzaTabella() {
              String nome = null;
              try {
            System.out.println("-----------------------------");
            System.out.println("INIZIALIZZAZIONE TABELLA...");
            Class.forName(driver);
                     Connection con = DriverManager.getConnection(url, user, password);
                     Statement st = con.createStatement();
            st.setQueryTimeout(5);
            ResultSet result;
            do {
                nome = randomStringa(5,15).toString();
                System.out.println("nome = "+nome);
                result = st.executeQuery("select stipendio from test where nome='"+nome+"'");
            } while (result.next());
            System.out.println("fine crezione connessione");
                     //st.executeUpdate("UPDATE test SET stipendio = 4000 WHERE nome = '"+nome
+"'");
            st.executeUpdate("insert into test values ('"+nome+"',4000)");
            System.out.println("eseguito l'insert");
            con.close();
```

```
            System.out.println("...INIZIALIZZAZIONE TABELLA COMPLETATA");
               } catch (ClassNotFoundException e) {
                      e.printStackTrace();
               } catch (SQLException e) {
                      e.printStackTrace();
               }
               return nome;
        }

    private static StringBuffer randomStringa(int da,int a){
        Random r = new Random();
        int lunghezza = r.nextInt(a-da)+da;
        StringBuffer result = new StringBuffer();
        for (int i = 0 ; i<lunghezza; i++){
            result.append((char)(r.nextInt(26)+97));
        }
        return result;
    }

}

public class PerditaDiAggiornamentoTester {

        private String url;
        private String user;
        private String password;
        private String driver;
    private String nome;

        public PerditaDiAggiornamentoTester(String driver,String url,String user,String password,
String nome){
               this.url = url;
               this.user = user;
               this.password = password;
               this.driver = driver;
        this.nome = nome;
        }

        public synchronized void eseguiTest(int livelloIsolamento) throws ClassNotFoundException,
SQLException, InterruptedException{
        System.out.println("-----------------------------");
               System.out.println("INIZIO TEST PERDITA DI AGGIORNAMENTO...");
               Class.forName(driver);
               Connection con1 = DriverManager.getConnection(url, user, password);
        con1.setAutoCommit(false);
        con1.setTransactionIsolation(livelloIsolamento);
               Connection con2 = DriverManager.getConnection(url, user, password);
        con2.setAutoCommit(false);
        con2.setTransactionIsolation(livelloIsolamento);

               MonitorPerditaAggiornamento monitor = new MonitorPerditaAggiornamento(nome);
               PerditaDiAggiornamentoThread runnable = new
PerditaDiAggiornamentoThread(con2,monitor);
               Thread thread = new Thread(runnable);
        thread.start();

        try {
            monitor.metodoFlussoUno(con1);
        }catch (Exception e) {
            System.out.println("errore sul flusso 1, transazione 1 abortita");
            e.printStackTrace();
        }

               thread.join();
        Connection con3 = DriverManager.getConnection(url, user, password);
        con3.setAutoCommit(false);
        con3.setTransactionIsolation(livelloIsolamento);
        Statement st = con3.createStatement();
        st.setQueryTimeout(10);
        ResultSet result = st.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x = result.getInt(1);
```

```
        System.out.println("alla fine leggo x = "+x);
        con3.commit();
        con3.close();
              System.out.println("...FINE TEST PERDITA DI AGGIORNAMENTO");
       }

}

class PerditaDiAggiornamentoThread implements Runnable{

       private Connection connection;
       private MonitorPerditaAggiornamento monitor;

       public PerditaDiAggiornamentoThread(Connection con2,MonitorPerditaAggiornamento monitor) {
              connection = con2;
              this.monitor = monitor;
       }

       public synchronized void run() {
              try {
            monitor.metodoFlussoDue(connection);
        } catch (Exception e) {
            System.out.println("errore sul flusso 2, transazione 2 abortita");
            e.printStackTrace();
        }
       }

}

class MonitorPerditaAggiornamento {

    private boolean turnoDelFlussoUno = true;
    private String nome;

       public MonitorPerditaAggiornamento(String nome) {
        this.nome = nome;
    }

    public synchronized void metodoFlussoUno(Connection con) throws InterruptedException,
SQLException {
           while(turnoDelFlussoUno == false) wait();
        turnoDelFlussoUno = false;
        Statement st = con.createStatement();
        st.setQueryTimeout(2);
        st.execute("SET CURRENT LOCK TIMEOUT WAIT 3");
        //eseguo r1(x)
        ResultSet result = st.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x = result.getInt(1);
        System.out.println("t1 legge x = "+x);
        notify();

        while(turnoDelFlussoUno == false) wait();
        turnoDelFlussoUno = false;

        //eseguo w1(x)
        x = x+5;
        System.out.println("x=x+5");
        try{
            st.execute("UPDATE test SET stipendio = "+x+" WHERE nome = '"+nome+"'");
            System.out.println("t1 scrive x = "+x);
            con.commit();
            System.out.println("t1 eseguito il commit");

        }catch (SQLException e){
            System.out.println("\nt1 scrittura bloccata: perdita di aggiornamento evitata
"+e.getMessage());
            //con.rollback();
            //System.out.println("t1 esegue un rollback");
        }
        notify();
        st.close();
```

```
        con.close();
        //System.out.println("t1 chiude la connessione");

       }

       public synchronized void metodoFlussoDue(Connection con) throws InterruptedException,
SQLException {
               while(turnoDelFlussoUno == true) wait();
        turnoDelFlussoUno = true;
        Statement st = con.createStatement();
        st.setQueryTimeout(2);
        st.execute("SET CURRENT LOCK TIMEOUT WAIT 3");
        //eseguo r2(x)
        ResultSet result = st.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x = result.getInt(1);
        System.out.println("\t\tt2 legge x = "+x);
        notify();

        while(turnoDelFlussoUno == true) wait();
        turnoDelFlussoUno = true;

        //eseguo w1(x)
        x = x+10;
        System.out.println("\t\tx=x+10");
        try{
            st.execute("UPDATE test SET stipendio = "+x+" WHERE nome = '"+nome+"'");
            System.out.println("\t\tt2 scrive x = "+x);
            con.commit();
            System.out.println("\t\tt2 eseguito il commit");

        }catch(SQLException e){
            System.out.println("\n\t\tt2 scrittura bloccata: perdita di aggiornamento evitata
"+e.getMessage());
            //con.rollback();
            //System.out.println("\t\tt2 esegue un rollback");
        }
        st.close();
        con.close();
        //System.out.println("\t\tt2 chiude la connessione");
       }

}

public class LetturaSporcaTester {

    private String url;
    private String user;
    private String password;
    private String driver;
    private String nome;

    public LetturaSporcaTester(String driver, String url, String user, String password, String nome)
{
        this.url = url;
        this.user = user;
        this.password = password;
        this.driver = driver;
        this.nome = nome;
    }

    public void eseguiTest(int livelloIsolamento) throws ClassNotFoundException, SQLException {
        System.out.println("-----------------------------");
        System.out.println("INIZIO TEST LETTURA SPORCA...");
        Class.forName(driver);
        Connection con1 = DriverManager.getConnection(url, user, password);
        con1.setAutoCommit(false);
        con1.setTransactionIsolation(livelloIsolamento);
        Connection con2 = DriverManager.getConnection(url, user, password);
        con2.setAutoCommit(false);
        con2.setTransactionIsolation(livelloIsolamento);
```

```
        Statement st1 = con1.createStatement();
        Statement st2 = con2.createStatement();
        st1.setQueryTimeout(1);
        st2.setQueryTimeout(1);
        st1.execute("SET CURRENT LOCK TIMEOUT WAIT 3");
        st2.execute("SET CURRENT LOCK TIMEOUT WAIT 3");

        ResultSet result = st1.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x1 = result.getInt(1);
        System.out.println("t1 legge x = "+x1);
        x1 = x1+5;
        System.out.println("x=x+5");
        st1.execute("UPDATE test SET stipendio = "+x1+" WHERE nome = '"+nome+"'");
        System.out.println("t1 scrive x = "+x1);
        try{
            result = st2.executeQuery("select stipendio from test where nome='"+nome+"'");
            result.next();
            int x2 = result.getInt(1);
            System.out.println("\t\tt2 legge x = "+x2);
        }catch(Exception e){
            System.out.println("\n\t\tt2 lettura sporca impedita: "+e.getMessage());
        }
        con1.rollback();
        System.out.println("t1 esegue l'abort\\rollback");
        con2.commit();
        System.out.println("\t\tt2 esegue il commit");
        st1.close();
        st2.close();
        con1.close();
        //System.out.println("t1 chiude la connessione");
        con2.close();
        //System.out.println("\t\tt2 chiude la connessione");

        Connection con3 = DriverManager.getConnection(url, user, password);
        con3.setAutoCommit(false);
        con3.setTransactionIsolation(livelloIsolamento);
        Statement st = con3.createStatement();
        st.setQueryTimeout(5);
        result = st.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x3 = result.getInt(1);
        System.out.println("alla fine leggo x = "+x3);
        con3.commit();
        con3.close();
        System.out.println("...FINE TEST LETTURA SPORCA");
    }
}

public class LetturaInconsistenteTester {

    private String url;
    private String user;
    private String password;
    private String driver;
    private String nome;

    public LetturaInconsistenteTester(String driver, String url, String user, String password,
String nome) {
        this.url = url;
        this.user = user;
        this.password = password;
        this.driver = driver;
        this.nome = nome;
    }

    public void eseguiTest(int livelloIsolamento) throws ClassNotFoundException, SQLException {
        System.out.println("-----------------------------");
        System.out.println("INIZIO TEST LETTURA INCONSISTENTE...");
        Class.forName(driver);
        Connection con1 = DriverManager.getConnection(url, user, password);
        con1.setAutoCommit(false);
```

```
        con1.setTransactionIsolation(livelloIsolamento);
        Connection con2 = DriverManager.getConnection(url, user, password);
        con2.setAutoCommit(false);
        con2.setTransactionIsolation(livelloIsolamento);

        Statement st1 = con1.createStatement();
        Statement st2 = con2.createStatement();
        st1.setQueryTimeout(1);
        st2.setQueryTimeout(1);
        st1.execute("SET CURRENT LOCK TIMEOUT WAIT 3");
        st2.execute("SET CURRENT LOCK TIMEOUT WAIT 3");

        ResultSet result = st1.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x1 = result.getInt(1);
        System.out.println("t1 legge x = "+x1);
        result = st2.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x2 = result.getInt(1);
        System.out.println("\t\tt2 legge x = "+x2);
        x2 = x2+5;
        System.out.println("\t\tx=x+5");
        try{
            st2.execute("UPDATE test SET stipendio = "+x2+" WHERE nome = '"+nome+"'");
            System.out.println("\t\tt2 scrive x = "+x2);
            con2.commit();
            System.out.println("\t\tt2 esegue il commit");
        }catch (SQLException e) {
            System.out.println("\n\t\tt2 scrittura vietata, lettura inconsistente impedita:
"+e.getMessage());
        }
        st2.close();
        con2.close();
        System.out.println("\t\tt2 chiude la connessione");
        result = st1.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        x1 = result.getInt(1);
        System.out.println("t1 legge x = "+x1);



        con1.commit();
        System.out.println("t1 esegue esegue il commit");
        st1.close();
        con1.close();
        System.out.println("t1 chiude la connessione");


        Connection con3 = DriverManager.getConnection(url, user, password);
        con3.setAutoCommit(false);
        con3.setTransactionIsolation(livelloIsolamento);
        Statement st = con3.createStatement();
        st.setQueryTimeout(5);
        result = st.executeQuery("select stipendio from test where nome='"+nome+"'");
        result.next();
        int x3 = result.getInt(1);
        System.out.println("alla fine leggo x = "+x3);
        con3.commit();
        con3.close();
        System.out.println("...FINE TEST LETTURA INCONSISTENTE");
    }

}

public class AggiornamentoFantasmaTester {

       private String url;
    private String user;
    private String password;
    private String driver;
    private String nomey;
       private String nomez;
```

```
       public AggiornamentoFantasmaTester(String driver, String url, String user, String password,
String nome, String nome2) {
              this.url = url;
        this.user = user;
        this.password = password;
        this.driver = driver;
        this.nomey = nome;
        this.nomez = nome2;
       }

       public void eseguiTest(int livelloIsolamento) throws SQLException, ClassNotFoundException {
              System.out.println("-----------------------------");
        System.out.println("INIZIO TEST AGGIORNAMENTO FANTASMA...");
        Class.forName(driver);
        Connection con1 = DriverManager.getConnection(url, user, password);
        con1.setAutoCommit(false);
        con1.setTransactionIsolation(livelloIsolamento);
        Connection con2 = DriverManager.getConnection(url, user, password);
        con2.setAutoCommit(false);
        con2.setTransactionIsolation(livelloIsolamento);

        Statement st1 = con1.createStatement();
        Statement st2 = con2.createStatement();
        st1.setQueryTimeout(1);
        st2.setQueryTimeout(1);
        st1.execute("SET CURRENT LOCK TIMEOUT WAIT 3");
        st2.execute("SET CURRENT LOCK TIMEOUT WAIT 3");

        int delta = 100;

        ResultSet result = st1.executeQuery("select stipendio from test where nome='"+nomey+"'");
        result.next();
        int y1 = result.getInt(1);
        System.out.println("t1 legge y = "+y1);
        result = st2.executeQuery("select stipendio from test where nome='"+nomey+"'");
        result.next();
        int y2 = result.getInt(1);
        System.out.println("\t\tt2 legge y = "+y2);
        y2 = y2-delta;
        System.out.println("\t\ty = y-"+delta);
        result = st2.executeQuery("select stipendio from test where nome='"+nomez+"'");
        result.next();
        int z2 = result.getInt(1);
        System.out.println("\t\tt2 legge z = "+z2);
        z2 = z2+delta;
        System.out.println("\t\tz = z+"+delta);
        try{
            st2.execute("UPDATE test SET stipendio = "+y2+" WHERE nome = '"+nomey+"'");
            System.out.println("\t\tt2 scrive y = "+y2);
            st2.execute("UPDATE test SET stipendio = "+z2+" WHERE nome = '"+nomez+"'");
            System.out.println("\t\tt2 scrive z = "+z2);
            con2.commit();
            System.out.println("\t\tt2 esegue il commit");
        }catch (SQLException e){
            System.out.println("\n\t\tt2 scrittura vietata, aggiornamento fantasma vietato:
"+e.getMessage());
        }
        result = st1.executeQuery("select stipendio from test where nome='"+nomez+"'");
        result.next();
        int z1 = result.getInt(1);
        System.out.println("t1 legge z = "+z1);
        con1.commit();
        System.out.println("t1 esegue il commit");
        /*try{
            st2.execute("UPDATE test SET stipendio = "+y2+" WHERE nome = '"+nomey+"'");
            System.out.println("\t\tt2 scrive x = "+y2);
            con2.commit();
            System.out.println("\t\tt2 esegue il commit");
        }catch (SQLException e) {
            System.out.println("\n\t\tt2 scrittura vietata, lettura inconsistente impedita:
"+e.getMessage());
```

```
        } */
        st2.close();
        con2.close();
        //System.out.println("\t\tt2 chiude la connessione");



        st1.close();
        con1.close();
        //System.out.println("t1 chiude la connessione");


        Connection con3 = DriverManager.getConnection(url, user, password);
        con3.setAutoCommit(false);
        con3.setTransactionIsolation(livelloIsolamento);
        Statement st = con3.createStatement();
        st.setQueryTimeout(5);
        result = st.executeQuery("select stipendio from test where nome='"+nomey+"'");
        result.next();
        int y3 = result.getInt(1);
        result = st.executeQuery("select stipendio from test where nome='"+nomez+"'");
        result.next();
        int z3 = result.getInt(1);
        System.out.println("alla fine leggo y = "+y3+" e z = "+z3);
        con3.commit();
        con3.close();
        System.out.println("...FINE TEST AGGIORNAMENTO FANTASMA");
       }

}

public class InserimentoFantasmaTester {

       private String url;
    private String user;
    private String password;
    private String driver;

       public InserimentoFantasmaTester(String driver, String url, String user, String password) {
              this.url = url;
        this.user = user;
        this.password = password;
        this.driver = driver;
       }

       public void eseguiTest(int livelloIsolamento) throws SQLException, ClassNotFoundException {
              System.out.println("-----------------------------");
        System.out.println("INIZIO TEST INSERIMENTO FANTASMA...");
        Class.forName(driver);
        Connection con1 = DriverManager.getConnection(url, user, password);
        con1.setAutoCommit(false);
        con1.setTransactionIsolation(livelloIsolamento);
        Connection con2 = DriverManager.getConnection(url, user, password);
        con2.setAutoCommit(false);
        con2.setTransactionIsolation(livelloIsolamento);

        Statement st1 = con1.createStatement();
        Statement st2 = con2.createStatement();
        st1.setQueryTimeout(1);
        st2.setQueryTimeout(1);
        st1.execute("SET CURRENT LOCK TIMEOUT WAIT 3");
        st2.execute("SET CURRENT LOCK TIMEOUT WAIT 3");

        ResultSet result = st1.executeQuery("select count(*) from test");
        result.next();
        int x = result.getInt(1);
        System.out.println("t1 conta tuple totali = "+x);
        try{
            st2.execute("insert into test values ('tupla aggiunta',4000)");
            System.out.println("\t\tt2 aggiunge una tupla");
            con2.commit();
            System.out.println("\t\tt2 esegue il commit");
```

```
        }catch (SQLException e){
            System.out.println("\n\t\tt2 inserimento bloccato, inserimento fantasma vietato:
"+e.getMessage());
        }

        result = st1.executeQuery("select count(*) from test");
        result.next();
        x = result.getInt(1);
        System.out.println("t1 conta tuple totali = "+x);
        con1.commit();
        System.out.println("t1 esegue il commit");

        st2.close();
        con2.close();
        //System.out.println("\t\tt2 chiude la connessione");

        st1.close();
        con1.close();
        //System.out.println("t1 chiude la connessione");


        Connection con3 = DriverManager.getConnection(url, user, password);
        con3.setAutoCommit(false);
        con3.setTransactionIsolation(livelloIsolamento);
        Statement st = con3.createStatement();
        st.setQueryTimeout(5);
        result = st.executeQuery("select count(*) from test");
        result.next();
        int x3 = result.getInt(1);

        System.out.println("alla fine leggo tuple totali = "+x3);
        con3.commit();
        con3.close();
        System.out.println("...FINE TEST INSERIMENTO FANTASMA");
       }

}
```

PUNTO 3

Riportiamo lo schema logico della base di dati descritta nell'homework.

Ricette(Numero, CodFarmacia, CFPaziente, Data)
Farmacia(CodFarmacia, Nome)
ElementiRicetta(NumeroRicetta, NumeroLinea, CodFarmaco)
Farmaci(Codice, Descrizione, CodMolecola, CodCasa, Prezzo, Fascia)
Molecole(CodMolecola, Descrizione)
Pazienti(CF, Cognome, Nome, DataNascita, Via, NumeroCivico, Citta)
CaseFarmaceutiche(CodCasa, Nome)
ASL(Codice, Nome)
Territorio(Via, Citta, NumeroCivico, ASL)

Base di dati di cui si deve definire lo schema.

```
create table farmacia(
codfarmacia varchar(50) primary key not null,
nome varchar(50));
create table asl(
codice varchar(50) primary key not null,
nome varchar(50));
create table territorio(
via varchar(50) not null,
citta varchar(50) not null,
numerocivico varchar(50) not null,
asl varchar(50) references asl (codice),
primary key (via,citta,numerocivico));
create table pazienti(
cf varchar(50) primary key not null,
cognome varchar(50),
nome varchar(50),
datanascita date,
via varchar(50),
numerocivico varchar(50),
citta varchar(50),
foreign key (via,numerocivico,citta) references territorio
(via,numerocivico,citta));
create table ricette(
numero integer primary key not null,
codfarmacia varchar(50) references farmacia (codfarmacia),
cfpaziente varchar(50) references pazienti (cf),
data date);
create table casefarmaceutiche(
codcasa varchar(50) primary key not null,
nome varchar(50));
create table molecole(
codmolecola varchar(50) primary key not null,
descrizione varchar(50));
create table farmaci(
codice varchar(50) primary key not null,
descrizione varchar(50),
codmolecola varchar(50) references molecole (codmolecola),
codcasa varchar(50) references casefarmaceutiche (codcasa),
prezzo integer,
fascia char(1));
create table elementiricetta(
numeroricetta integer references ricette (numero) not null,
numerolinea integer not null,
codfarmaco varchar(50) references farmaci (codice),
```

```
primary key (numeroricetta,numerolinea));
```

Abbiamo cosi generato l'sql per la creazione della base di dati. Si può notare come abbiamo utilizzato il campo date sia per la data della ricetta, sia per la data di nascita del paziente. Questo per permettere successivamente opportune operazioni su questi campi, quali ad esempio estrarre l'età dalla data di nascita del paziente.

Per gli altri campi si è utilizzato integer nel caso in cui fosse decisamente chiaro che il tipo dell'attributo in questione fosse semplicemente numerico, altrimenti varchar.

Ad ogni modo non era nei nostri obiettivi creare una base di dati efficiente, per il tipo di operazione che doveva svolgere, ma piuttosto studiare il processo di creazione del DW e di popolamento di esso.

```
create table tempodimension(
tempo_key integer primary key not null GENERATED ALWAYS AS IDENTITY (START WITH
1, INCREMENT BY 1),
data date);

create table farmacodimension(
farmaco_key integer primary key not null GENERATED ALWAYS AS IDENTITY (START
WITH 1, INCREMENT BY 1),
codmolecola varchar(50),
descrizione varchar(50),
codcasa varchar(50),
nome varchar(50));

create table fasciaetadimension(
fasciaeta_key integer primary key not null GENERATED ALWAYS AS IDENTITY (START
WITH 1, INCREMENT BY 1),
fascia varchar(50),
eta integer);

create table asldimension(
asl_key integer primary key not null GENERATED ALWAYS AS IDENTITY (START WITH 1,
INCREMENT BY 1),
nome varchar(50),
codice varchar(50));

create table farmaciadimension(
farmacia_key integer primary key not null GENERATED ALWAYS AS IDENTITY (START
WITH 1, INCREMENT BY 1),
nome varchar(50),
codfarmacia varchar(50));

create table prescrizione(
tempo_key integer not null references tempodimension(tempo_key),
farmaco_key integer not null references farmacodimension(farmaco_key),
farmacia_key integer not null references farmaciadimension(farmacia_key),
asl_key integer not null references asldimension(asl_key),
fasciaeta_key integer not null references fasciaetadimension(fasciaeta_key),
quantita integer,
```

```
prezzototale integer,
primary key(tempo_key,farmaco_key,farmacia_key,asl_key,fasciaeta_key)
);
```

Abbiamo riportato l'sql per la creazione del data warehouse. Si fa notare come le dimensioni da noi create siano 5, in relazione alle richieste dell'esercizio: fasciaetadimension, farmacodimension, tempodimension, farmaciadimension, asldimension.

In questo modo la privacy dei pazienti è garantita. Non vengono riportati in nessuna dimensione gli indirizzi e i nomi dei pazienti.

Il cambiamento degli attributi prezzi e fasce dei farmaci non è stato un problema, dato che tali campi non vengono riportati nel DW. Invece, per consentire il cambiamento delle fasce di età, abbiamo dovuto inserire, oltre alla fascia d'età di interesse (0-3, 4-17 …) anche l'età, in modo tale da rendere possibile il cambiamento della fascia. D'altronde, tale cambiamento, comporta che la massima grandezza di fasciaetadimension consista in al massimo 100-120 tuple. Il che non è praticamente ininfluente dal punto di vista dello spazio occupato.

Altra cosa da notare è l'attributo della chiave delle dimensione che è di tipo auto incrementante.

Riportiamo di seguito anche le varie insert utilizzate per popolare il database.

```
insert into asl values ('rew22','asldiroma');
insert into asl values ('asd21','asldinapoli');
insert into asl values ('lmk12','asldireggio');

insert into territorio values ('via giulietta','torino', '32','rew22');
insert into territorio values ('via romeo', 'roma', '32','asd21');
insert into territorio values ('via aladdin', 'brindisi','19','lmk12');

insert into pazienti values ('fgrdln64d01c333h','mario', 'rossi',
'01/01/2001','via giulietta','32','torino');
insert into pazienti values ('ffrdln64d01c323h','luca', 'di
giaconmo','01/04/2002','via romeo','32','roma');
insert into pazienti values ('fgrdln64d01c129g','francesco','napoletano',
'12/12/2001','via aladdin','19','brindisi');

insert into farmacia values ('a35d','proietti');
insert into farmacia values ('a36g','scarpati');
insert into farmacia values ('q25d','di mario');

insert into casefarmaceutiche values ('byr23', 'bayern');
insert into casefarmaceutiche values ('fr45', 'farma');
insert into casefarmaceutiche values ('udnm14','friulana');

insert into molecole values ('cha12','valeriana');
insert into molecole values ('fre14','metadone');
insert into molecole values ('gia14','striscia');

insert into farmaci values ('1','antidolorifico','cha12','byr23', 12,'A');
insert into farmaci values ('2','antidolorifico','fre14','fr45', 12,'B');
```

```
insert into farmaci values ('3','antidolorifico','cha12','byr23', 12,'C');
insert into farmaci values ('4','antibiotico', 'fre14','udnm14',12,'C');
insert into farmaci values ('5','antibiotico', 'cha12','udnm14',12,'C');
insert into farmaci values ('6','analgesico', 'gia14','fr45', 12,'A');

insert into ricette values (1,'q25d','fgrdln64d01c333h','12/12/2006');
insert into ricette values (2,'a36g','ffrdln64d01c323h','12/03/2005');
insert into ricette values (3,'q25d','fgrdln64d01c129g','12/09/2005');
insert into ricette values (4,'a35d','fgrdln64d01c129g','27/01/2006');
insert into ricette values (5,'a36g','ffrdln64d01c323h','30/05/2004');
insert into ricette values (6,'a35d','fgrdln64d01c333h','12/12/2006');
insert into ricette values (7,'a35d','fgrdln64d01c333h','12/12/2006');

insert into elementiricetta values (1,1,'1');
insert into elementiricetta values (1,2,'2');
insert into elementiricetta values (1,3,'3');
insert into elementiricetta values (1,4,'4');
insert into elementiricetta values (2,1,'5');
insert into elementiricetta values (2,2,'6');
insert into elementiricetta values (3,1,'1');
insert into elementiricetta values (3,2,'2');
insert into elementiricetta values (3,3,'6');
insert into elementiricetta values (3,4,'6');
insert into elementiricetta values (4,1,'3');
insert into elementiricetta values (4,2,'4');
insert into elementiricetta values (5,1,'5');
insert into elementiricetta values (5,2,'6');
insert into elementiricetta values (5,3,'1');
insert into elementiricetta values (6,1,'1');
insert into elementiricetta values (6,2,'2');
insert into elementiricetta values (7,2,'2');
insert into elementiricetta values (7,3,'3');
```

A questo punto abbiamo potuto riempire le tabelle dimensione.

```
insert into farmaciadimension (nome,codfarmacia)(
select distinct nome,codfarmacia
from farmacia
where (nome,codfarmacia) not in (
select nome,codfarmacia
from farmaciadimension));
insert into farmacodimension (codmolecola,descrizione,codcasa,nome)(
select distinct farmaci.codmolecola,molecole.descrizione,farmaci.codcasa,nome
from farmaci,molecole,casefarmaceutiche
where molecole.codmolecola=farmaci.codmolecola
and casefarmaceutiche.codcasa=farmaci.codcasa
and (farmaci.codmolecola,farmaci.codcasa) not in (
select codmolecola,codcasa
from farmacodimension));
insert into asldimension (nome,codice)(
select distinct nome,codice
from asl
where (nome,codice) not in (
select nome,codice
from asldimension));
```

```
insert into tempodimension (data) (
select distinct data
from ricette
where (data) not in (
select data
from tempodimension));

insert into fasciaetadimension (fascia,eta) values ('0-3',0);
insert into fasciaetadimension (fascia,eta) values ('0-3',1);
insert into fasciaetadimension (fascia,eta) values ('0-3',2);
insert into fasciaetadimension (fascia,eta) values ('0-3',3);
insert into fasciaetadimension (fascia,eta) values ('4-7',4);
insert into fasciaetadimension (fascia,eta) values ('4-7',5);
insert into fasciaetadimension (fascia,eta) values ('4-7',6);
insert into fasciaetadimension (fascia,eta) values ('4-7',7);
insert into fasciaetadimension (fascia,eta) values ('8-11',8);
insert into fasciaetadimension (fascia,eta) values ('8-11',9);
insert into fasciaetadimension (fascia,eta) values ('8-11',10);
insert into fasciaetadimension (fascia,eta) values ('8-11',11);
```

Con queste query sql abbiamo cosi riempito le dimensioni in modo tale da evitare di scrivere nuovamente campi già presenti della dimensione stessa. A tal proposito si è utilizzato l'operatore "not in". Discussione a parte, invece, per la fascia di età, che deve essere riempita manualmente, o con un programma apposito.

Per riempire la tabella dei fatti ci siamo però avvalsi di una vista, per gestire più facilmente l'età dei pazienti.

```
create view pazienti2 as
select cf,year(datanascita) as anno,via,numerocivico,citta from pazienti;
```

Si è utilizzato il metodo year(<data>) che estrae l'anno da un campo data fornito come input. Infine ci siamo occupati del popolamento della tabella dei fatti: prescrizione.

```
insert into prescrizione
(fasciaeta_key,farmaco_key,tempo_key,asl_key,farmacia_key,quantita,prezzototale)
(
select
(select ETA.fasciaeta_key from fasciaetadimension as ETA where
ETA.eta=(year(R.data)-P.anno)),
(select FAD.farmaco_key from farmacodimension FAD where
FAD.codmolecola=FA.codmolecola and FAD.codcasa=FA.codcasa),
(select DD.tempo_key from tempodimension DD where DD.data=R.data),
(select AD.asl_key from asldimension AD where AD.codice=A.codice),
(select FD.farmacia_key from farmaciadimension FD where
FD.codfarmacia=F.codfarmacia),
count(*),
(sum(FA.prezzo))
from farmacia F,asl A,territorio T,pazienti2 P,ricette R,casefarmaceutiche
C,molecole M,farmaci FA,elementiricetta E
where F.codfarmacia = R.codfarmacia
and A.codice = T.asl
and T.via = P.via
```

```
and T.citta = P.citta
and T.numerocivico = P.numerocivico
and P.cf = R.cfpaziente
and R.numero = E.numeroricetta
and C.codcasa = FA.codcasa
and M.codmolecola = FA.codmolecola
and FA.codice = E.codfarmaco
-- and R.data > 'ultima volta'
group by F.codfarmacia, A.codice,FA.codmolecola,FA.codcasa,P.anno,R.data);
```

Abbiamo utilizzato interrogazioni sql avvalendoci delle variabili. Ci sono infatti query nidificate per selezionare le chiavi esterne della tabella fatti, che si avvalgono proprio delle variabili della select "padre".

Altra cosa, importantissima da notare in tale query è l'utilizzo dell'operatore "group by" che determina la grana della tabella prescrizione, che si va a popolare con l'insert. E in più l'utilizzo degli operatori di aggregazione, come count e sum, che servono per ricavare la quantità e il prezzo totale.

A questo punto il DW è pronto per essere interrogato per operazioni di analisi come la seguente:

```
select fascia,sum(prezzototale) as prezzotot,sum(quantita) as quantita
from prescrizione P,fasciaetadimension F
where P.fasciaeta_key=F.fasciaeta_key
group by fascia;
```

Ad esempio è così possibile visionare tutte le prescrizioni per fascia di età, e quindi si è fatta un operazione di roll up rispetto a tutte le altre dimensioni, che non vengono neanche considerate in questa query.