Monday, May 23, 2011

What is Dirty Read?

Dirty read occurs wherein one transaction is changing the tuple/record, and a second transaction can read this tuple/record before the original change has been committed or rolled back. This is known as a dirty read scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid value.

To understand it better,lets take a use case where on thread is viewing the record and other thread is updating the value of the record.Since the transaction isolation attribute is set to READ UNCOMMITTED.Second thread will be able to see the changes made by other thread.

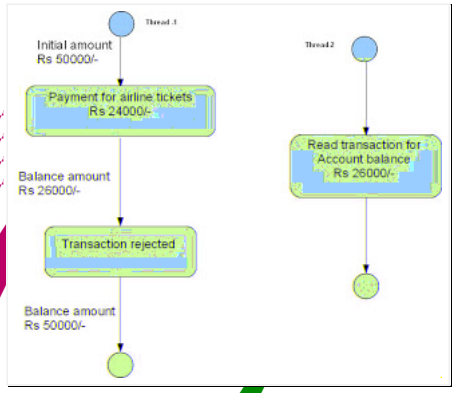
Example of Dirty Read:-

In the current use case we have a table containing account balances.
One thread is reading the data from the account balances table.Other thread is updating the data from that table.Since isolation attribute is 'READ UNCOMMITTED' .Other thread can view non committed data.

Table Definition:-

```
Create table AccountBalance
(
    id integer Primary Key,
    acctName varchar2(255) not null,
    acctBalance integer(9,2) not null,
    bankName varcahr2(255)
);
```

insert into AccountBalance values (1,'Kunaal',50000,'Bank-a');



ReaderRunImpl.java

```
package com.kunaal.poolin

impor  ja a. ql.Connec
impor  ja a. ql.Prepa      men ;
impor  ja a. ql.Re    l
impor  ja a. ql.SQLE       n;

/**
 * Reader thread wh    elects the data while other thread is updating the
 * data
 *
 * @author Kunaal A    rehan
 *
 */
p blic cla    ReaderR nImpl  implemen   Runnable{

 pri a e Connection conn;

 pri a e   a ic final String QUERY="Select balance from AccountBalance where id=1";
```

Labels

-
-
- (
- (
- (
-
-
- (
-
-

Blog Archive

(

(

(

Followers

Join th

Members (1

About Me

Kunaal A

Follow

Subscribe To

```
 p blic ReaderR nImpl(Connection conn){
   hi .conn=conn;
 }

 @Override
 p blic  oid r n() {
  PreparedStatement stmt =n ll;
  ResultSet rs =n ll;

    r  {
    stmt = conn.prepareStatement(QUERY);
    rs = stmt.executeQuery();
     hile (rs.next()){
     System.out.println("Balance is:" + rs.getDouble(1));

    }
  } ca ch (SQLException e) {
    e.printStackTrace();
  }finall {
     r  {
     stmt.close();
     rs.close();
    } ca ch (SQLException
     e.printStackTrace()
    }
   }
 }

}
```

PaymentRunImpl.java

```
package com.kunaal.p

impor  ja a. ql.Conne    ;
impor  ja a. ql.Prepa    a emen ;
impor  ja a. ql.SQL      ion;

/**
 * @author Kunaa      rehan
 *
 */
p blic cla   Pa    en R nImpl implemen   Runnable{

 pri a e Connection conn;

 pri a e   a ic final String QUERY="Update AccountBalance set balance=26000 where id=1";

 p blic Pa men R nImpl(Connection conn){
   hi .conn=conn;
 }

 @Override
 p blic  oid r n() {
  PreparedStatement stmt = n ll;

    r  {
    stmt = conn.prepareStatement(QUERY);
    stmt.execute();
    Thread.currentThread().sleep(3000);
    conn.rollback();
  } ca ch (SQLException e) {
    e.printStackTrace();
  } ca ch (InterruptedException e  {
    e.printStackTrace();
  }finall {
     r  {
     stmt.close();
    } ca ch (SQLException e
     e.printStackTrace();
    }
   }
 }

}
```

DirtyReadExample.java

```
package com.kunaal.p      g;

impor  ja a. ql.Conne     ion;
impor  ja a. ql.SQLE      ep ion;

/**
 * @author Kunaal A Trehan
 *
 */
p blic cla   Dir  ReadE ample {
```

```
/**
 * @param args
 * @throws SQLException
 * @throws InterruptedException
 */
p blic   a ic  oid main(String[] args) {
 ConnectionPool pool=ne  ConnectionPool(5  000);

 Connection connPymt = pool.getConnection();
 Connection connReader = pool.getConnection();
  r  {
  connPymt.setAutoCommit(fal e);
  connPymt.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);

  connReader.setAutoCommit(fal e);
  //connReader.setTransactionIsolation(Connection.TRANSACTION_READ_UNCOMMITTED);
  connReader.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
 } ca ch (SQLException e) {
  e.printStackTrace();
 }


 Thread pymtThread=ne  Thread(ne  PaymentRunImpl(connPymt));
 Thread readerThread=ne  Thread(ne  ReaderRunImpl(connReader));

 pymtThread.start();
  r  {
  Thread.currentThread().sleep(1000);
 } ca ch (InterruptedException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 }
 readerThread.start();

 pool.returnConnection(connPymt);
 pool.returnConnection(connReader);
}

}
```

Here reader thread views the account balance when payment thread is sleeping.So reader thread will view the balance as 24000 if the isolation level is Connection.TRANSACTION_READ_UNCOMMITTED as other transaction has roll back the transaction.

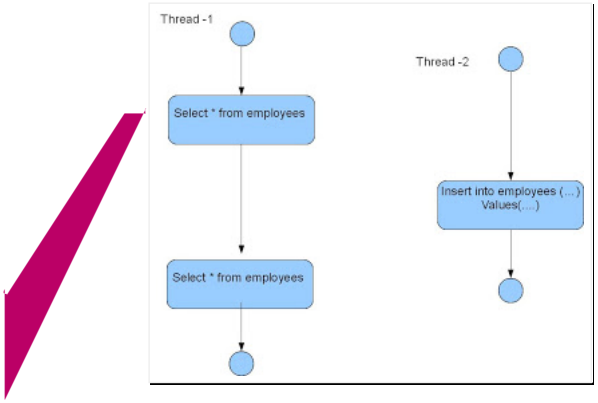
What is Phantom Read?

Phantom read occurs where in a transaction same query executes twice, and the second result set includes rows that weren't visible in the first result set. This situation is caused by another transaction inserting new rows between the execution of the two queries.

Example of Phantom Read:-

To understand it better consider a use case where one thread is inserting the data while other thread is reading the data in different transaction.Since reader thread has isolation attribute as READ COMMITTED,reader thread will see the new rows inserted when it queries again.

Table definition:-
Create table Employee
(
    id integer Primary Key,
    empName varchar2(255) not null,
    empCity varchar2(255) not null,
    empCtry varchar2(255)not null
);



PhantomReader.java

```
package com.kunaal.pooling;

impor  ja a. ql.Connec ion;
```

```
impor  ja a. ql.Prepa      a emen ;
impor  ja a. ql.Re    ;
impor  ja a. ql.SQL     ion;

/**
 * @author Kunaa      rehan
 *
 */
p blic cla   Pha  omReader implemen   Runnable{

 pri a e Connection conn;

 pri a e   a ic final String QUERY="Select * from Employee";

 p blic Phan omReader(Connection conn){
   hi .conn=conn;
 }

 @Override
 p blic  oid r n() {
  PreparedStatement stmt =n ll;
  ResultSet rs=n ll;

   r  {
   stmt=conn.prepareStatement(QUERY);
   rs=stmt.executeQuery();

    hile(rs.next()){
    System.out.println("Emp Details- "+ rs.getInt(1) + "-"+ rs.getString(2) + "-"+
        rs.getString(3)+ "-"+ rs.getString(4));
   }

   Thread.currentThread().sleep(3000);
   System.out.println("AFTER WAKING UP");
   System.out.println("=============================================");

   rs=stmt.executeQuery();

    hile(rs.next()){
    System.out.println("Emp Details- "+ rs.getInt(1) + "-"+ rs.getString(2) + "-"+
        rs.getString(3)+ "-"+ rs.getString(4));
   }
  } ca ch (SQLException e) {
   e.printStackTrace();
  } ca ch (InterruptedException e) {
   e.printStackTrace();
  }finall {
    r  {
    rs.close();
    stmt.close();
   } ca ch (SQLException e)
    e.printStackTrace();
   }
  }
 }

}
```

PhantomInsert.java

```
package com.kunaal.p

impor  ja a. ql.Conne    ;
impor  ja a. ql.Prep     a eme
impor  ja a. ql.SQL     ion;

/**
 * @author Kunaa      rehan
 *
 */
p blic cla   Pha  omIn er  mplemen   Runnable{

 pri a e Connection conn;

 pri a e   a ic final String QUERY="Insert into Employee values(?,?,?,?)";

 p blic Phan omIn er (Connection conn){
   hi .conn=conn;
 }

 @Override
 p blic  oid r n() {
  PreparedStatement stmt =n ll;

   r  {
   stmt = conn.prepareStatement(QUERY);
   stmt.setInt(1, 3);
```

```
    stmt.setString(2, "ABC");
    stmt.setString(3,"DELHI");
    stmt.setString(4, "INDIA");

    stmt.execute();
    conn.commit();
   } ca ch (SQLException e) {
    e.printStackTrace();
   }finall {
     r  {
     stmt.close();
    } ca ch (SQLException e) {
     e.printStackTrace();
    }
   }

 }

}
```

PhantomReadExample.java

```
package com.kunaal.p

impor  ja a. ql.Conn  on;
impor  ja a. ql.SQL  ion;

/**
 * @author Kunaal  rehan
 *
 */
p blic cla   Phan omReadE ample {

 /**
  * @param args
  */
 p blic   a ic  oid main(String[] args) {
  ConnectionPool pool=ne  ConnectionPool(  000);

  Connection connInsert = pool.getConnection();
  Connection connReader = pool.getConnection();
   r  {
   connInsert.setAutoCommit(fal e);
   connInsert.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);

   connReader.setAutoCommit(fal e);
   //connInsert.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
   connReader.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
  } ca ch (SQLException e) {
   e.printStackTrace();
  }

  Thread readThread=ne  Thread(ne  PhantomReader(connReader));
  Thread insertThread=ne  Thread(ne  PhantomInsert(connInsert));
  readThread.start();
  insertThread.start();

  pool.returnConnection(connReader);
  pool.returnConnection(connInsert);

 }

}
```

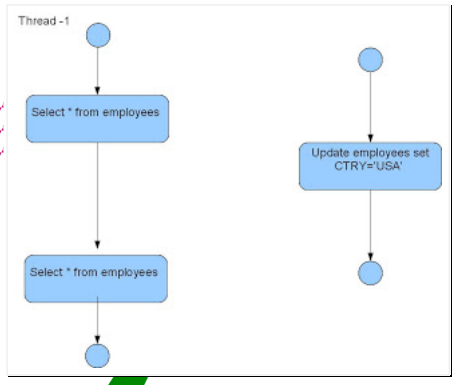
Here reader thread with isolation level as READ COMMITTED queries the employee table twice while other thread is inserting the data.As a result of which number of rows returned is different.

What is Non Repeatable Read?

Non Repeatable Reads happen when in a same transaction same query yields different results. This happens when another transaction updates the data returned by other transaction.

Example of Non Repeatable Read:-

To understand it better lets take a use case where one thread is viewing the data and other thread is updating the data.Since isolation level is READ COMMITED,other thread will be able to view the updated changes.So in the same transaction,same query will yield different data.



Updater.java

```
package com.kunaal.p

impor  ja a. ql.Conne    ;
impor  ja a. ql.Prep     a emen ;
impor  ja a. ql.SQL      ion;

/**
 * @author Kuna      rehan
 *
 */
p blic cla   Upd   er implemen   Runnable{

pri a e Connection conn;

 pri a e   a ic final String QUERY="Update Employee set empCountry='USA'";

 p blic Upda er(Connection conn){
   hi .conn=conn;
 }

 @Override
 p blic  oid r n() {
  PreparedStatement stmt =n ll;

   r  {
   stmt = conn.prepareStatement(QUERY);
   stmt.executeUpdate();
   conn.commit();
  } ca ch (SQLException e) {
   e.printStackTrace();
  }finall {
    r  {
    stmt.close();
   } ca ch (SQLException e)
    e.printStackTrace();
   }
  }

 }

}
```

NonRepeatableReader.java

```
package com.kunaal.po

impor  ja a. ql.Conne
impor  ja a. ql.Prepa     a emen ;
impor  ja a. ql.Re    ;
impor  ja a. ql.SQL      ion;

/**
 * @author Kuna      rehan
 *
 */
p blic cla   Non epea ableReader implemen   Runnable{

 pri a e Connection conn;

 pri a e   a ic final String QUERY="Select * from Employee";

 p blic NonRepea ableReader (Connection conn){
   hi .conn=conn;
 }

 @Override
```

```
 p blic  oid r n() {
  PreparedStatement stmt =n ll;
  ResultSet rs=n ll;

   r  {
   stmt=conn.prepareStatement(QUERY);
   rs=stmt.executeQuery();

    hile(rs.next()){
    System.out.println("Emp Details- "+ rs.getInt(1) + "-"+ rs.getString(2) + "-"+
        rs.getString(3)+ "-"+ rs.getString(4));
   }

   Thread.currentThread().sleep(3000);
   System.out.println("AFTER WAKING UP");
   System.out.println("=============================================");

   rs=stmt.executeQuery();

    hile(rs.next()){
    System.out.println("Emp Details- "+ rs.getInt(1) + "-"+ rs.getString(2) + "-"+
        rs.getString(3)+ "-"+ rs.getString(4));
   }
  } ca ch (SQLException e) {
   e.printStackTrace();
  } ca ch (InterruptedException e) {
   e.printStackTrace();
  }finall {
    r  {
    rs.close();
    stmt.close();
   } ca ch (SQLException e
    e.printStackTrace();
   }
  }
 }

}
```

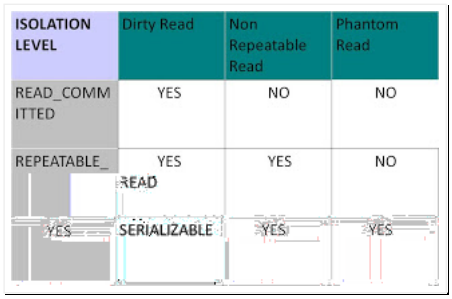
NonRepeatbleExample.java

```
package com.kunaal.p

impor  ja a. ql.Conne  on;
impor  ja a. ql.SQL  ion;

/**
 * @author Kunaal  rehan
 *
 */
p blic cla   NonRepea bleE ample {

 /**
  * @param args
  */
 p blic   a ic  oid main(String[] args) {
  ConnectionPool pool=ne  ConnectionPool(  000);

  Connection connUpdt = pool.getConnection();
  Connection connReader = pool.getConnection();
   r  {
   connUpdt.setAutoCommit(fal e);
   connUpdt.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);

   connReader.setAutoCommit(fal e);
   //connReader.setTransactionIsolation(Connection.TRANSACTION_READ_UNCOMMITTED);
   connReader.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
  } ca ch (SQLException e) {
   e.printStackTrace();
  }


  Thread updtThread=ne  Thread(ne  Updater(connUpdt));
  Thread readerThread=ne  Thread(ne  NonRepeatableReader(connReader));

  readerThread.start();
   r  {
   Thread.currentThread().sleep(1000);
  } ca ch (InterruptedException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
  updtThread.start();

  pool.returnConnection(connUpdt);
  pool.returnConnection(connReader);
 }
```

```
}
```

Here while reading same query will yield different results as other thread has updated the data

Following table depicts the isolation level mapping with dirty read ,phantom read and others.



ISOLATION LEVEL	Dirty Read	Non Repeatable Read	Phantom Read
READ_COMMITTED	YES	NO	NO
REPEATABLE_READ	YES	YES	NO
SERIALIZABLE	YES	YES	YES

Anonymous

Sergey

Reply

CaR

Nitesh Porwal

Алексей Кузнецов

!

Anonymous

!

Manish

Anonymous

Anonymous

Reply

Comment as:

Publish Preview

(