YOUR NAME:

OBJECTIVES: The goal of this lab is to gain an understanding of transactions. We're going to create transactions to build dolls by multiple users simultaneously.

1. Two windows
   You should open two sqlPlus windows on your screen. We are going to pretend that there are two users, one window for each user. Open a notepad or other type of editor window to write your statements in this window and cut/paste to the sqlPlus windows.

2. Create tables
   Execute the following in either window. Thus, the following creates two tables to keep track of inventories of parts of a Doll and an assembled doll.

   
   ```sql
   create table DollParts ( 
       name varchar2(10),
       cnt number,
       constraint PKdollParts primary key(name),
       constraint ck1 check (cnt >= 0) 
   );
   ```

   ```sql
   insert into DollParts(name, cnt) values('HEAD', 17);
   insert into DollParts(name, cnt) values('BODY', 17);
   insert into DollParts(name, cnt) values('ARM', 17);
   insert into DollParts(name, cnt) values('LEG', 17);
   commit;
   ```

   ```sql
   create table Dolls ( 
       name varchar2(20),
       cnt number,
       constraint PKdolls primary key(name)
   )
   ```
insert into Dolls(name, cnt) values('Barbie', 0);
insert into Dolls(name, cnt) values('Ken', 0);
commit;

3. Setting the isolation level

SQL has several "isolation levels". They provide the degree of independence between concurrently executing transactions. We will use the default isolation level in Oracle, read committed, for now. You do not need to know any more about isolation levels in this lab. Use of various isolation levels in transactions will be studied in the next lab.

4. Using transactions:

In one window do an SQL update command that sets the count of heads in DollParts to 9. In the other window set the count to 23. You should be able to do a select * from DollParts and see cnt=23 in both windows, now.

The transaction we're going to be interested in doing is to deduct from inventory everything we need to make a doll and increment the count of that doll by 1. The problem we can get into is this: what if two users are doing this at the same time? Put your sqlPlus windows one over the other, so I can refer to them as top and bottom and do the following:

TOP: Update DollParts set cnt=cnt-1 where name='HEAD';
TOP: Update DollParts set cnt=cnt-1 where name='BODY';
TOP: commit;
BOT: Select * from DollParts

What we get in the bottom window is an inventory in a partial condition. We might be checking to see if we have enough parts for a doll, but we would be wrong, because there's a doll-build in progress right now and two arms and legs are already spoken for.

Dealing with this completely is complicated, so we're going to take it one step at a time.
5. **A transaction with writes**

Now, do the following steps:

TOP: Update DollParts set cnt=cnt-1 where name='HEAD';
TOP: Update DollParts set cnt=cnt-1 where name='BODY';
TOP: Select cnt from DollParts where name='HEAD';
TOP: Select cnt from DollParts where name='BODY';
BOT: Update DollParts set cnt=cnt-1 where name='HEAD';
BOT: commit;
BOT: Select cnt from DollParts where name='HEAD';

This duplicated the condition of two doll-builds starting at the same time. Notice that BOT has hung! That is because TOP has an exclusive lock on the HEAD and BODY rows of DollParts. This will suspend BOT until TOP is done. Go ahead and finish in TOP:

TOP: Update DollParts set cnt=cnt-2 where name='ARM';
TOP: Update DollParts set cnt=cnt-2 where name='LEG';
TOP: commit;

Now we indicate to SQL that the transaction is complete using this statement in TOP:

commit;

Note that BOT finished successfully. It was not allowed to do anything until TOP was done. Note that it wasn’t even doing a transaction at the time, just update statements.

6. **Write Locks**

What happened? TOP acquired an exclusive lock on the rows it wanted to write. An exclusive lock means nobody else can read or write the rows at all. In a transaction, every write attempt will try to acquire an exclusive lock. If the object is already locked by another transaction, this transaction will patiently wait for the other to end.

When are the locks released? When the transaction commits!

7. **A transaction with reads**

Do the following:
TOP: select * from Dolls where name='Barbie' for update;
BOT: update Dolls set cnt=cnt+1 where name='Barbie';
BOT: Select cnt from Dolls where name='Barbie';

What happened? TOP acquired a **shared lock** on the rows it read. A shared lock is a "read-only" lock. When BOT tried to write to the record it tried to acquire an exclusive lock, it is forced to wait. Let’s free it up:

TOP: commit;

And finish the transaction in BOT:

BOT: commit;

8. **Deadlock!**

In oracle transactions acquire exclusive locks on specific rows and shared lock on the table as a result of sql DML. For example, select for update will result in acquiring exclusive lock on particular rows to be updated and shared lock on the table containing the rows. For now we will focus on exclusive lock for the specific rows. Let us try this:

TOP: Select * from Dolls where name='Barbie' for update;
TOP: update Dolls set cnt=cnt+1 where name='Barbie';
TOP: commit;
TOP: select cnt from Dolls where name='Barbie';

When the select for update occurred, the transaction acquired an exclusive lock on the rows for 'Barbie'. Now try this version:

BOT: select * from Dolls where name='Barbie' for update;
TOP: select * from Dolls where name='Barbie' for update;
TOP: update Dolls set cnt=cnt+1 where name='Barbie';
TOP: select cnt from Dolls where name='Barbie';

Note that TOP is now waiting on BOT to finish it’s transaction. Why? Let’s finish this transaction:
BOT: commit;  
TOP: commit;  

What if TOP must wait for BOT and BOT must wait for TOP? Can that happen? Yes it can. The condition is called \textbf{deadlock}. Try this:  

TOP: select * from Dolls where name='Barbie' for update;  
BOT: select * from Dolls where name='Ken' for update;  
TOP: select * from Dolls where name='Ken' for update;  
BOT: select * from Dolls where name='Barbie' for update;  

This is an error. What happened?  
What caused the deadlock?  
Write an explanation of why deadlock occurred in this case and include it in what you turn in at the end of the lab. (Print your turn-in on paper)  

9. Rollback  
There are actually two possible ways to end a transaction: we either commit, meaning we are happy with the transaction, or we can rollback. A rollback command is like this: 
rollback;  
Try the following steps in TOP:  

\[
\begin{align*}  
\text{select * from Dolls;}  
\text{update Dolls set cnt=cnt+1 where name='Barbie';}\  
\text{update Dolls set cnt=cnt+1 where name='Ken';}\  
\text{select * from Dolls;}  
\text{rollback;}  
\end{align*}
\]

select * from Dolls;  

Note that the table was not changed. Neither of the steps were taken. Why would we want to do this? Suppose we are building a doll. We try to get items from the inventory, but one item is not available. We can rollback all of the other items we took. Basically, when a transaction fails you rollback so it is either an all or nothing affair.