In this lab you will run concurrent transactions that will perform at different ANSI SQL isolation levels. These isolation levels will be abbreviated in this document as:

Read Uncommitted: RU
Read Committed: RC
Repeatable/Phantom Read: RR
Serializable: SR

The above four isolation levels allow a combination of the following four phenomena:

Lost Update, Dirty Read, Non-repeatable Read and Phantom Read.

Your task in this lab is to run two concurrent transactions and observe the behavior of their interactions (i.e., presence or absence of these phenomena) for different levels of isolations.

**Assignment:**

1. Create the following table to be used in all transactions.

```sql
create table Account (
    AcctName varchar2(30),
    AcctBal number
);
insert into Account values('John', 100);
insert into Account values('David', 200);
insert into Account values('Mary', 300);
insert into Account values('Cathy', 100);
insert into Account values('Calvin', 300);
insert into Account values('Susan', 300);
commit;
```
2. Open two windows and call them TOP and BOT (Bottom). Define two transactions in these two windows as follows. Note that the step numbers are indicated as comments. These step numbers are used in defining the schedules in the following pages.

TOP:

-- 1. & 2.

  var x number
  begin select AcctBal into :x from Account where AcctName='John'; end;
  /
  print x

-- 3.

  begin :x := :x + 10; end;
  /
  print x

-- 4.

  begin update Account set AcctBal=:x where AcctName='John'; end;
  /

-- 5.

  commit;/rollback;

BOT:

-- 1'. & 2'.

  var x number
  begin select AcctBal into :x from Account where AcctName='John'; end;
  /
  print x

-- 3'.

  begin :x := :x + 20; end;
  /
  print x

-- 4'.

  begin update Account set AcctBal=:x where AcctName='John'; end;
  /
-- 5'.
    commit;/rollback;

The above two transactions can be conceptually represented as follows.
You cannot use this syntax to run it in sqlPlus.

TOP:
1. begin transaction
2. x= select AcctBal from Account where AcctName="John"
3. x= x+10
4. update Account set AcctBal=x where AcctName="john"
5. commit/abort transaction

BOT:
1'. begin transaction
2'. x= select AcctBal from Account where AcctName="John"
3'. x= x+20
4'. update Account set AcctBal=x where AcctName="john"
5'. commit/abort transaction

Now you will be running the two transactions in an interleaved fashion
one sql statement at a time. You will write the results of these runs in
the tables given below. For tables with four rows, you will not run any
transactions. You will simply fill them out by assuming ANSI SQL
isolation level described at the end of this document. Note that the
locking method described is not part of the ANSI SQL standard. For
tables with two rows, you will run schedules in Oracle to fill them out.
3. **Lost Update**: $(r_1(x), r_2(x), w_1(x), c_1, w_2(x), c_2)$:

Fill out the following table for the schedule given below, assuming ANSI-SQL isolation level. The schedule allows lost update.

Schedule: 1,2,1',2',3,4,5c,3',4',5'c

<table>
<thead>
<tr>
<th>Runs</th>
<th>If no., db value</th>
<th>db value</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>reasons?</td>
<td>expected</td>
<td>obtained</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Give an application where lost update is allowed.
Run the above schedule in oracle and fill out the table below: Note that you need to set the isolation level before each transaction by using one of the following statements:

set transaction isolation level read committed;
set transaction isolation level serializable;

Also note that you may like to reset a table to its original values before trying the next isolation level as follows:

update Account set AcctBal=100 where AcctName='John';
commit;

| Runs | | If no, | db value | db value | comments |
|------| | ------|----------|----------|----------|
|      | |  ? | reasons? | expected | obtained |
|------| |  ---|--------|---------|----------|
| RC   | |   |      |         |          |
|      | |   |      |         |          |
|------| |  ---|------|-------|---------|
| SR   | |   |      |         |          |
|      | |   |      |         |          |
|------| |  ---|------|-------|---------|

-------------------------------------------------------------

4. **Dirty Read**: \((w_1(x), r_2(x), a_1(x))\)

Fill out the following table for the schedule given below, assuming ANSI-SQL isolation level. The schedule allows lost update.

Schedule: 1,2,3,4,1',2',5r,3',4',5'c

<table>
<thead>
<tr>
<th>Runs</th>
<th>If no</th>
<th>db value</th>
<th>db value</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>?, ?</td>
<td>reasons?</td>
<td>expected</td>
<td>obtained</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RU</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-----</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-----</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-----</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-----</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
</tbody>
</table>

Give an application where dirty read is allowed.
Run the above schedule in oracle and fill out the table below:

<table>
<thead>
<tr>
<th>Runs</th>
<th>If no,</th>
<th>db value</th>
<th>db value</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>reasons?</td>
<td>expected</td>
<td>obtained</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>----------</td>
<td>----------</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>RC</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

---

<table>
<thead>
<tr>
<th>SR</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

---
5. **Non-repeatable Read**: \((r_1(x), w_2(x), c_2, r_1(x))\)

update Account set AcctBal=100 where AcctName='John';
commit;

Revise the Bot transaction as follows:

**BOT:**

-- 1'.
   set serveroutput on
   var x number
-- 2'.
   begin select AcctBal into :x from Account where AcctName='John'; end;
   /
   print x
-- 3'.
   declare
       bal number;
   begin
       if :x=100 then
           select AcctBal into bal from Account where AcctName='John';
           dbms_output.put_line(bal);
       end if;
   end;
   /
-- 4.
   commit;/rollback;
Fill out the following table for the schedule given below, assuming ANSI-SQL isolation level. The schedule allows non-repeatable read.

schedule: 1',2',1,2,3,4,5c,3',4'

<table>
<thead>
<tr>
<th>Runs</th>
<th>If no</th>
<th>db value</th>
<th>db value</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>reasons?</td>
<td>expected</td>
<td>obtained</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Give an application where non repeatable read is allowed.

Run the above schedule in oracle and fill out the table below:

<table>
<thead>
<tr>
<th>Runs</th>
<th>If no</th>
<th>db value</th>
<th>db value</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>reasons?</td>
<td>expected</td>
<td>obtained</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9
6. **Phantom Read**: \( (r_1(x), w_2(x), c_2, r_1(x)) \); \( r_1(x) \): multiple tuples.

update Account set AcctBal=100 where AcctName='John'; commit;

Revise the Bot transaction as follows:

```sql
-- BOT:

-- 1'.
set serveroutput on
var x number

-- 2'
begin select AVG(AcctBal) into :x from Account where AcctBal<150; end;
/
print x

-- 3'.
declare
    avgBal number;
begin
    if :x<=100 then
        select AVG(AcctBal) into avgBal from Account where AcctBal<150;
        dbms_output.put_line(avgBal);
    end if;
end;
/
print x

-- 4'.
commit;/rollback;
```
Fill out the following table for the schedule given below, assuming ANSI-SQL isolation level. The schedule allows phantom read.

schedule: 1',2',1,2,3,4,5c,3',4'

<table>
<thead>
<tr>
<th>Runs</th>
<th>If no</th>
<th>db value</th>
<th>db value</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>RU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Give an application where phantom read is allowed.

Run the above schedule in oracle and fill out the table below:

<table>
<thead>
<tr>
<th>Runs</th>
<th>If no</th>
<th>db value</th>
<th>db value</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. You will run a long duration transaction, given below, concurrently (interleaved) from two different windows. You will run them using both read committed as well as serializable isolation levels. In the following, a high level description of the transaction is given first. A PL/sql description of the transaction is provided next. You take this PL/sql description of the transaction and run it from two different windows using both read committed and serializable isolation levels. Observe and note the differences.

    -- initialization
    create table flight(
        no number primary key,
        seats number
    );

    insert into flight values (101, 1);
    insert into flight values (102, 1);
    commit;

T1: (interleaving)
    1. select seats from flight where no=101;
    2. select seats from flight where no=102;
        -- if both x>0 and y>0
    3. select seats from flight where no=101 for update;
    4. select seats from flight where no=102 for update;
        -- if both x>0 and y>0
            -- get customer charge card number (this step may be skipped)
    5. update flight set seats=seats-1 where no=101;
    6. update flight set seats=seats-1 where no=102;
    7. commit;
    -- else
    5. rollback;

T1: (pl/sql program)

    set serveroutput on
    declare
        seats101 number;
        seats102 number;
begin
select seats into seats101 from flight where no=101;
-- do some work
select seats into seats102 from flight where no=102;
-- do some work
if seats101 > 0 and seats101 > 0 then
    select seats into seats101 from flight where no=101 for update;
select seats into seats102 from flight where no=102 for update;
-- get customer charge card number (this step may be skipped)
if seats101 > 0 and seats101 > 0 then
    update flight set seats=seats-1 where no=101;
update flight set seats=seats-1 where no=102;
commit;
dbms_output.put_line('Committed');
else
    rollback;
dbms_output.put_line('No seats, rollback');
end if;
else
    dbms_output.put_line('No seats');
end if;
end;
/
<table>
<thead>
<tr>
<th>Isolation level (Degree)</th>
<th>Read locks</th>
<th>Write locks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted (Degree 0)</td>
<td>none required</td>
<td>Well-formed writes Long duration write locks</td>
</tr>
<tr>
<td>Read Committed (Degree 1)</td>
<td>Well-formed reads, short duration read locks</td>
<td>Well-formed writes long duration write locks</td>
</tr>
<tr>
<td>Repeatable Read (Degree 2)</td>
<td>Well-formed reads, Long duration data item read lock, short duration read predicate lock</td>
<td>Well-formed writes Long duration write locks</td>
</tr>
<tr>
<td>Serializable (Degree 3)</td>
<td>Long duration read locks</td>
<td>Well-formed writes long duration write locks</td>
</tr>
</tbody>
</table>