CS1555 Recitation 12 - Solution

Objective: 1. To get started with JDBC and demonstrate transaction concurrency control on oracle.
2. To practice Views

PART 1:

Oracle supports multi-version concurrency control. Therefore, a normal select statement does not acquire a lock on the data item it reads and never reads dirty (uncommitted) data.

Transactions are characterized by the ACID properties:
o Atomicity
o Consistency
o Isolation
o Durability

Isolation is ensured by Concurrency Control, which synchronizes the execution of transactions to ensure each executes in an isolation manner. The default, more restricted isolation level is serializability in which transactions execute as if they executed serially, one after the other.

In general, SQL transactions can execute under different isolation levels.
o Serializable
o Repeatable reads
o Read committed
o Read uncommitted (execute like OS processes)

1. Typical synchronization is based on locks: (1) read or shared and (2) write or exclusive. How do these locks work?
Answer:

- SHARED
  Row-level shared locks allow multiple transactions to read data, but do not allow any transaction to change that data. Multiple transactions can hold shared locks simultaneously.

- EXCLUSIVE
  An exclusive lock allows only one transaction to update a particular piece of data (insert, update, and delete). When one transaction has an exclusive lock on a row or table, no other lock of any type may be placed on it.

2. Oracle supports multi-version concurrency control. What does multi-version mean?
Answer:

When an MVCC database needs to update an item of data, it will not overwrite the original data item with new data, but instead creates a newer version of the data item. Thus there are multiple versions stored. The version that each transactions sees depends on the isolation level implemented.

SQL select reads the most recently committed version of a table/tuple. That is, a SQL select statement does not acquire a lock on the data item it reads and never reads dirty (uncommitted) data.

PART 2:

Before we start:

- In order to set the PATH and CLASSPATH environmental variables to point to JAVA and Oracle JDBC library, just do the following:
  - `source ~panos/1555/bash.env.class3`

- Copy the following files to your working directory:
  - `cp ~panos/1555/recitation/rec8db.sql rec8db.sql`
  - `cp ~panos/1555/recitation/TranDemo1.java TranDemo1.java`
  - `cp ~panos/1555/recitation/TranDemo2.java TranDemo2.java`

- Open 3 terminal windows, ssh to class3 and set the environment variables.
  - In the first terminal, we'll be running sqlplus to keep track of what changes are happening to the database.
  - In the second terminal, we'll be running TranDemo1.java
  - In the third terminal, we'll be running TranDemo2.java

Example 0: Getting Started

- Edit the TranDemo1.java file and TranDemo2.java, change the username and password to your username and password that you use to login to Oracle.
- Compile the files using the command: `javac TranDemo1.java; javac TranDemo2.java`
- Execute rec8db.sql under sqlplus in the first terminal.
- Run the program using the command: `java TranDemo1 0`
- Now read the demo source file to learn how it works. Note in the file:
  - How to connect to the DB.
  - How to execute an SQL statement.
  - How to iterate through the results set.

Notes:

- To run any of the examples, pass as an argument the example number.
- We will start running 2 transactions concurrently by running TranDemo1 and TranDemo2.
  - Run TranDemo1 first and then run TranDemo2 while TranDemo1 is still running.
  - Notice in the source codes how to group SQL statements into one transaction, commit/rollback the transaction and how to set isolation level for the transaction.
  - The sleep (milliseconds) function is used to force the statements in both transactions to execute in the order we want.

Example 1: Multi-version Concurrency of Oracle

TranDemo1 (read committed)	TranDemo2 (read committed)
update class set max_num_students = 5 where classid = 1 sleep... **rollback**	SELECT * FROM class where classid = 1

Question: What is the max_num_students as read by TranDemo2?
Answer: *Because Oracle supports multi version concurrency control, TranDemo2 read the committed value of max_num_students (i.e., before the update of TranDemo1). Therefore TranDemo2 does not read dirty data and also does not have to wait for TranDemo1 to release the write lock (exclusive lock).*

Example 2: (Implicit) Unrepeatable Read Problem

TranDemo1 (read committed)	TranDemo2 (read committed)
select max_num_students, cur_num_students **from** class **where** classid = 1 **sleep...** **if**(cur_num_students <max_num_students) **update** class **set** cur_num_students = cur_num_students +1 **where** classid = 1 **else** **print** 'the class is full' **commit**	**select** max_num_students, cur_num_students **from** class where classid = 1 **sleep...** **if**(cur_num_students <max_num_students) **update** class **set** cur_num_students = cur_num_students +1 **where** classid = 1 **else** **print** 'the class is full' **commit**

Question: What is the value of cur_num_students for class with classid = 1 ? Compare it to the max_num_classes.
Answer: *both of the two transactions registered for class 1, updating the cur_num_students to 3 even though the maximum number of students allowed in this class is only 2. The reason is, both of them read 1 as the current number of students in the class. This is called an implicit case of unrepeatable read: at the time TranDemo2 tried to update cur_num_students, it read that the value is still 1 while it has been updated to 2 (i.e., if it reads the value again at this point, the value will be different).*

Example 3: Serializable Isolation Level:

The same as Example 2, but each transaction has the isolation level of **serializable**
Before running example 3, reset the database by rerunning rec8db.sql in the first terminal window.

Question: What is the value of cur_num_students for class with classid = 1 now? Do both transactions perform the update?
Answer: *Only TranDemo1 can register (i.e. update cur_num_students,) successfully. TranDemo2 got the error message: "*ORA-08177: Cannot serialize access for this transaction"*. In serializable isolation level, Oracle allows a transaction T to update a data item only if no other transaction is committing an update on that data item since T started. In this example, TranDemo1 has committed its update to the data item, preventing TranDemo2 from executing its update statement. Therefore, the data is still consistent: the cur_num_students is kept less than or equal to the max_num_students. The program should be able to catch this type of error from Oracle and re-run the transaction instead of notifying the application user of the error.*

Example 4: Using for Update of

The same as Example 2, but each transaction uses the following statement to select max and current number of students:

```
SELECT max_num_students, cur_num_students
FROM class where classid = 1
for update of cur_num_students
```

Again before running example 4, reset the database by rerunning rec8db.sql in the first terminal window.

Question: What is the value of cur_num_students for class with classid = 1 now? Do both transactions perform the update?
Answer: *Only TranDemo1 can register (i.e. update cur_num_students,) successfully. TranDemo2 got the user-friendly error message: "*the class is full". *When a transaction executes a "*select…for update*" statement, it acquires an exclusive lock on the data item. This means that when TranDemo1 read the cur_num_students of class 1, it also kept an xlock on the corresponding row(s). Later, when TranDemo2 tried to read cur_num_students, it has to first ask for the xlock. Because TranDemo1 has the x lock, TranDemo2 has to wait until TranDemo1 commits and releases the lock. Therefore, the outcome of this example is the same as when the 2 transactions run sequentially.*

Example 5: Deadlock

TranDemo1 (read committed)	TranDemo2 (read committed)
update class **set** max_num_students = 10 **where** classid = ***1*** sleep...	
	update class **set** max_num_students = 20 **where** classid = ***2***
update class **set** max_num_students = 10 **where** classid = ***2*** **commit**	sleep...
	update class **set** max_num_students = 20 **where** classid = ***1*** **commit**

Question: What is the value of max_num_students ? Do both transactions perform their updates?
Answer: *Deadlock happens. One of the two transactions is selected as the victim and rollbacks. The victim transaction receives an error message that a deadlock is detected. The other transaction runs normally.*

PART 3 (Optional Review):

Before we start:
- Copy and run the file creating the Student database using:
  host cp ~panos/1555/recitation/studentdb.sql studentdb.sql
  @ studentdb

1. Create a view called student_courses that lists the SIDs, student names, number of courses in the Course_taken table.

```
create or replace view student_courses as
select s.sid, s.name, count(course_no) as num_courses
from student s, course_taken ct
where s.sid = ct.sid
group by s.sid, s.name;
```

2. Create a materialized view called mv_student_courses that lists the SIDs, student names, number of courses in the Course_taken table.

```
drop materialized view mv_student_courses;
create materialized view mv_student_courses
--refresh complete on commit
as
select s.sid, s.name, count(course_no) as num_courses
from student s, course_taken ct
where s.sid = ct.sid
group by s.sid, s.name;
```

3. Execute the following commands. Compare the query results and time used of the two select statements.

```
insert into course_taken
values('CS1555', '129','Fall 18', null);
commit;

--execute DBMS_MVIEW.REFRESH('mv_student_courses');
set timing on
select * from mv_student_courses;
set timing on
set timing off
select * from student_courses;
set timing off
commit;
```

- The result from the materialized view is incorrect because the materialized view was not refreshed after the insert statement.
- The result from the view is correct because what a normal view does is rewriting the query. It does not store a snapshot of the query result like the materialized view.
- The running time of the materialized view is shorter, because it does not need to rewrite the query and run the rewritten query on the original Course_taken table.

4. Reset the database by running the studentdb.sql and recreate the views. Comment back the line beginning with “execute” in the above commands and execute the commands. Compare the query results of the two select statements.

- The result from the materialized view is correct this time, because we refreshed the materialized view before the select statement.
- The default refresh method of a materialized view is “Refresh on Demand”, which requires the user to execute the refresh command explicitly, in order to refresh the view (i.e. the line beginning with “execute”).
- You can make the refresh happen automatically after each commit that modifies the original table by using the “Refresh on Commit” refresh method in the declaration of the materialized view (i.e., the line commented out in the solution of Question 2).