NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES

CL 203-Database Systems

Lab Session 12

MAIN OBJECTIVES:

- **Transaction Isolation Levels**
- **READ UNCOMMITTED**
- **READ COMMITTED**
- **REPEATABLE READ**
- **SERIALIZABLE**
- **ANOMALIES IN TRANSACTIONS**
- **Dirty read**
- **Nonrepeatable read**
- **Phantom read**

Transaction Isolation Levels

There are four levels of transaction isolation defined in ANSI/ISO SQL standard, with different possible outcomes for the same transaction scenario. That is, the same work performed in the same fashion with the same inputs may result in different answers, depending on your isolation level. These levels are defined in terms of three phenomena that are either permitted or not at a given isolation level:

- **Dirty read:** The meaning of this term is as bad as it sounds. You're permitted to read uncommitted, or dirty, data. You can achieve this effect by just opening an OS file that someone else is writing and reading whatever data happens to be there. Data integrity is compromised, foreign keys are violated, and unique constraints are ignored.
- **Nonrepeatable read:** This simply means that if you read a row at time T1 and try to reread that row at time T2, the row may have changed. It may have disappeared; it may have been updated, and so on.
- **Phantom read:** This means that if you execute a query at time T1 and re-execute it at time T2, additional rows may have been added to the database, which may affect your results. This differs from a nonrepeatable read in that with a phantom read, data you already read hasn't been changed, but instead, more data satisfies your query criteria than before.

Anomaly	Example
Dirty Reads A dirty read happens when a transaction reads data that is being modified by another transaction that has not yet committed.	Transaction A begins. UPDATE emp SET salary = 31650 WHERE empno = '7782' Transaction B begins. SELECT * FROM emp (Transaction B sees data updated by transaction A. Those updates have not yet been committed.)
Non-Repeatable Reads Non-repeatable reads happen when a query returns data that would be different if the query were repeated within the same transaction. Non-repeatable reads can occur when other transactions are modifying data that a transaction is reading.	Transaction A begins. SELECT * FROM emp WHERE empno = '7782' Transaction B begins. UPDATE empl SET salary = 30100 WHERE empno = '7782' (Transaction B updates rows viewed by transaction A before transaction A commits.) If Transaction A issues the same SELECT statement, the results will be different.
Phantom Reads Records that appear in a set being read by another transaction. Phantom reads can occur when other transactions insert rows that would satisfy the WHERE clause of another transaction's statement.	Transaction A begins. SELECT * FROM emp WHERE sal > 30000 Transaction B begins. INSERT INTO emp (empno, ename,job , , sal) VALUES ('7356', 'NICK', 'CLERK',’35000’) Transaction B inserts a row that would satisfy the query in Transaction A if it were issued again.

The isolation levels defined by the standard ANSI/ISO SQL are listed as follows.

Serializable

This is the highest isolation level.

With a lock-based concurrency control DBMS implementation, serializability requires read and write locks (acquired on selected data) to be released at the end of the transaction. Also range-locks must be acquired when a SELECT query uses a ranged WHERE clause, especially to avoid the phantom reads phenomenon .

When using non-lock based concurrency control, no locks are acquired; however, if the system detects a write collision among several concurrent transactions, only one of them is allowed to commit.

Repeatable reads

In this isolation level, a lock-based concurrency control DBMS implementation keeps read and write locks (acquired on selected data) until the end of the transaction. However, range-locks are not managed, so the phantom reads phenomenon can occur (see below).

Read committed

In this isolation level, a lock-based concurrency control DBMS implementation keeps write locks (acquired on selected data) until the end of the transaction, but read locks are released as soon as the SELECT operation is performed (so the non-repeatable reads phenomenon can occur in this isolation level, as discussed below). As in the previous level, range-locks are not managed.

Putting it in simpler words, read committed is an isolation level that guarantees that any data read is committed at the moment it is read. It simply restricts the reader from seeing any intermediate, uncommitted, 'dirty' read. It makes no promise whatsoever that if the transaction re-issues the read, it will find the same data; data is free to change after it is read.

Read uncommitted

This is the lowest isolation level. In this level, dirty reads are allowed, so one transaction may see not-yet-committed changes made by other transactions.

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	Permitted	Permitted	Permitted
READ COMMITTED	--	Permitted	Permitted
REPEATABLE READ	--	--	Permitted
SERIALIZABLE	--	--	--

COMMAND TO SET ISOLATION LEVEL:

The command to set the isolation level of a transaction, at its beginning, is:

SET TRANSACTION ISOLATION LEVEL {SERIALIZABLE | READ COMMITTED};

OR

SET TRANSACTION READ ONLY;

User User1

SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction set.

SQL> CREATE TABLE Primes (p INT);

Table created.

SQL> GRANT SELECT, UPDATE, INSERT ON Primes to pagh2;

Grant succeeded.

SQL> SELECT * FROM Primes;

no rows selected

SQL> INSERT INTO Primes VALUES (41);

1 row created.

SQL> SELECT * FROM Primes;

```
        P
----------
       41
```

SQL> COMMIT;

Commit complete.

User User2

SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction set.

SQL> SELECT * FROM user1.Primes;

no rows selected

SQL> INSERT INTO user1.Primes VALUES (43);

1 row created.

SQL> SELECT * FROM user1.Primes;

```
        P
----------
       43
```

SQL> SELECT * FROM user1.Primes;

```
         P
----------
        43
```

SQL> COMMIT;

Commit complete.

SQL> SELECT * FROM user1.Primes;

```
         P
----------
        41
        43
```

SQL> SELECT * FROM user1.Primes;

```
         P
----------
        41
        43
```

SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction set.

SQL> INSERT INTO Primes VALUES (2);

1 row created.

SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction set.

SQL> SELECT * FROM user1.Primes;

```
         P
----------
        41
        43
```

SQL> INSERT INTO user1.Primes VALUES (2003);

1 row created.

SQL> SELECT * FROM user1.Primes;

```
         P
----------
        41
        43
      2003
```

SQL> COMMIT;

Commit complete.

SQL> SELECT * FROM user1.Primes;

```
         P
----------
        41
        43
         2
      2003
```

SQL> SELECT * FROM Primes;

```
         P
----------
        41
        43
         2
```

SQL> COMMIT;

Commit complete.

SQL> SELECT * FROM Primes;

```
         P
----------
        41
        43
         2
      2003
```

SQL> INSERT INTO Primes VALUES (3);

1 row created.

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT * FROM Primes;

```
         P
----------
        41
        43
         2
      2003
```

EXERCISE

STEP 1:

Create a table t with an attribute tno.

STEP 2:

Create a table t_log with attribute t_count.

STEP 3:

Implement a PL/SQL procedure `p1` that writes into a table `t` the numbers 1 to 10000 (each in a separate tuple).

STEP 4:

Write a procedure `p2` that counts 20 times the number of tuples in `it`.

STEP 5:

Inserts each of the counting results into a tuple in a log table.

STEP 6:

Set isolation transaction mode to read committed with the following command.

```
Set transaction isolation level read committed
```

STEP 7:

Execute `p1` and `p2` concurrently on two pc's separately, where both transactions are in `read committed` isolation.

STEP 8:

Check the content of the log table. How do you justify the obtained result??

_____

_____