SEN-209: Designing And Implementing Databases
Lab 7: Multi-User Environment & Transactions

Name: Thanawin Pattanaphol Date: 01324096

Part A – Transactions & ACID

```
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

dvdrental_new=#
BEGIN;
UPDATE customer SET first_name = 'Alice' WHERE customer_id = 1;
BEGIN
UPDATE 1
dvdrental_new=*# ROLLBACK
dvdrental_new-*# ;
ROLLBACK
dvdrental_new=#
```

```
[postgres@ ~]$ psql -U postgres -d dvdrental_new
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

dvdrental_new=# SELECT first_name FROM customer WHERE customer_id = 1;
 first_name
------------
 Mary
(1 row)

dvdrental_new=# SELECT first_name FROM customer WHERE customer_id = 1;
 first_name
------------
 Mary
(1 row)
```

Session 2 never saw uncommitted changes due to PostgreSQL's default isolation level i.e. Read Committed Level, that is, a SELECT query can only see data committed before the query began and never sees the uncommitted data or changes made by other transactions.

Part B – Lost Update Problem

Initial rental_rate: 4.99

```
lab7=# SELECT rental_rate FROM film WHERE film_id = 10;
 rental_rate
-------------
        4.99
(1 row)
```

Session 1 & 2 committed.

```
[postgres@ ~]$ psql -U postgres -d lab7
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

lab7=#
BEGIN;
UPDATE film SET rental_rate = rental_rate + 1 WHERE film_id = 10;
BEGIN
UPDATE 1
lab7=*# COMMIT;
COMMIT
lab7=# SELECT rental_rate FROM film WHERE film_id = 10;
 rental_rate
-------------
        7.99
(1 row)
```

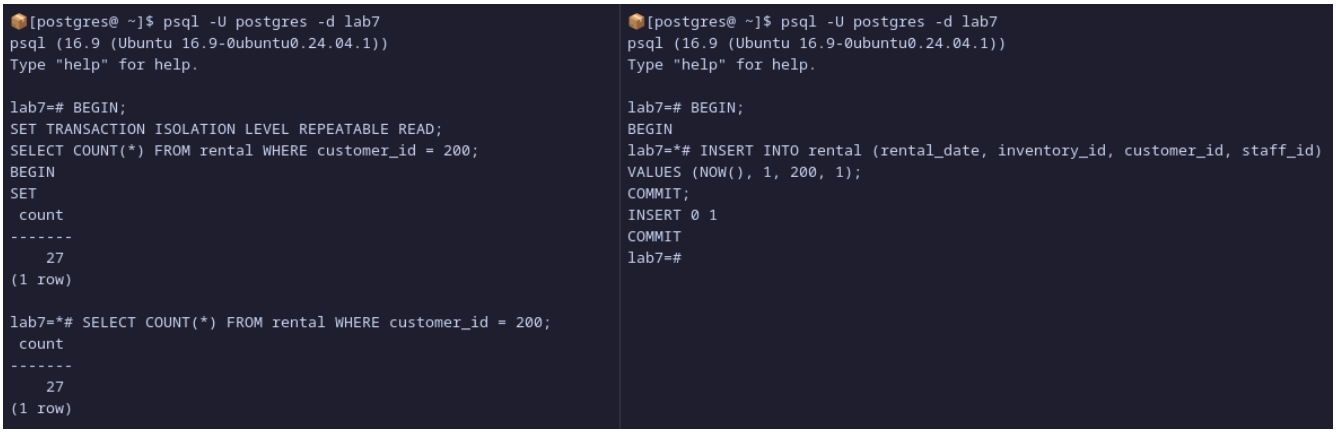
```
[postgres@ ~]$ psql -U postgres -d lab7
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

lab7=# BEGIN;
UPDATE film SET rental_rate = rental_rate + 2 WHERE film_id = 10;
BEGIN
UPDATE 1
lab7=*# COMMIT;
COMMIT
lab7=#
```

The final rental_rate is 7.99, one update did not overwrite the other since the 2nd session (session on the right), waits for the query of the 1st session to be committed (does not allow any input of any other query while waiting), once it is commited, 2nd session query continues.

SEN-209: Designing And Implementing Databases
Lab 7: Multi-User Environment & Transactions

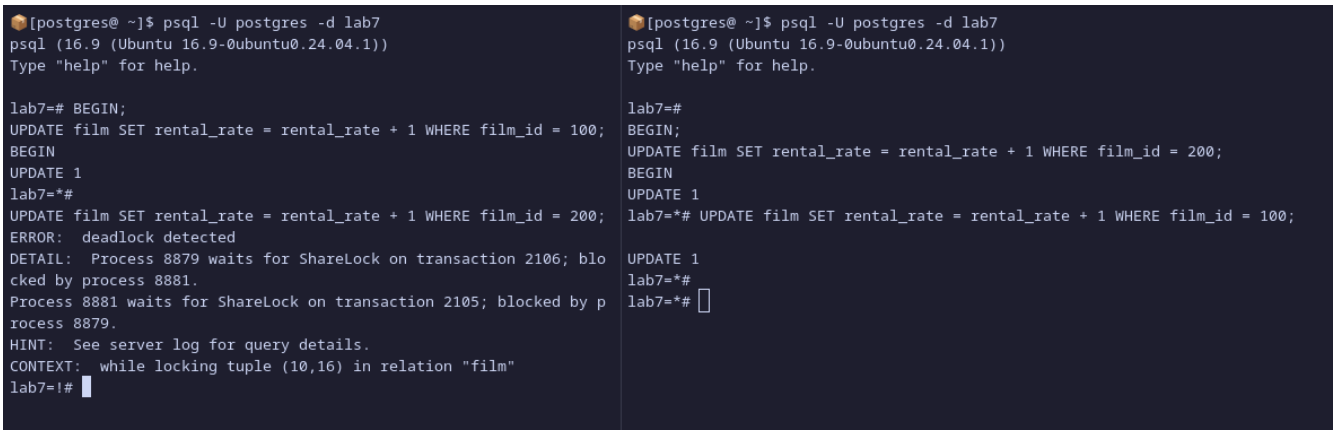
Name: Thanawin Pattanaphol Date: 01324096

Part C – Isolation Levels Experiment



The count did not change as seen in the 1st session on the left, this is due to the properties of Repeatable Read isolation level, in which, when enabled, the level only sees data committed before the transaction began and never sees uncommitted data or changes committed by other transactions during its execution.

Part D – Deadlock Simulation



The deadlock occurs when two transactions are in a conflict, in this case, the error occurs when Session 1 tries to update film_id 200 (which was being updated by Session 2) and Session 2 tries to update film_id 100 (which was being updated by Session 1). PostgreSQL waits for the deadlock_timeout and triggers the deadlock.

SEN-209: Designing And Implementing Databases
Lab 7: Multi-User Environment & Transactions

Name: Thanawin Pattanaphol Date: 01324096

Part E – Thought Exercise

1. Which isolation level would you choose (Read Committed, Repeatable Read, Serializable)? Why?

   The isolation level that would be the most ideal in this context would be the Serializable level as it the strictest or most restrictive isolation level in PostgreSQL since it essentially turns transactions that are running simultaneously to behave as it is running sequentially; eliminates all anomalies that can come up.

2. Would you allow dirty reads if it meant faster performance? Why or why not?

   It would the most ideal if we can achieve faster performance while maintaining the most minimal amount of dirty reads. This is due to the fact that dirty reads can lead to inconsistent readings of data or “anomalies” and since, in this context, the data that is being stored are financial related records, therefore, it is best to minimize the amount of anomalies as much as possible. Even though, this level sacrifices performance as it requires re-execution of the transactions, it is more desirable for a financial-related database to favor accuracy over faster performance.

3. How would you prevent deadlocks in such a system?

   The serializable isolation level already prevents deadlocks by, when encountering one, it re-executes the transaction sequentially. Though, this isolation level may sacrifice fast performance, it is more ideal to reduce as many deadlocks as possible.

Short Essay - What trade-offs exist between strict consistency and system performance in real-world multi-user systems?

In real-world multi-user systems, there can be several trade-offs when balancing strict consistency and system performance, this is due to the fact that, in a large-scale multi-user system, there consists of a large amount of load as multiple operations or transactions are being executed constantly or simultaneously, e.g. user A updating their information on table X while user B is retrieving their information from table Y & Z; usually referred to as “concurrency”, as concurrency increases, chances of data inconsistencies also increase as well; as more data is needed to be processed in a limited amount of time.

Due to the increase of data concurrency, issues of data inconsistencies rise along with other issues, such as, race conditions, in which the output of the user operations depend on the order in which operations or transactions are executed first. These issues can lead to major consequences especially in systems related to finance, healthcare and several other industries that require data accuracy and consistency. Therefore, it is important to put in place measures that prevent the aforementioned issues.

SEN-209: Designing And Implementing Databases
Lab 7: Multi-User Environment & Transactions

Name: Thanawin Pattanaphol Date: 01324096

However, when it comes to implementing the said measures, it brings up the main trade-off between performance and consistency, that is, in order to ensure that data is accurate and consistent, multi-user systems, e.g. database management systems, implement techniques, such as, locks, isolation levels, transaction management i.e. ACID properties, and others; with these measure implemented, transactions or operations would have to be slowed down along with possible increased need in computing power. This can be seen in isolation levels, such as the serializable isolation level, in which, queries will be put into sequence to ensure that every query is executed properly. If a sudden error, such as, deadlocks, were to occur, the DBMS will retry the query that caused the deadlock which requires more resources and slow down the other queries that are being queued to be executed.