Exercise 3

Nasrul Huda, Aldo Sula

May 13, 2025

1 3. Synchronization with Locking Protocols

1.1 3.1. Isolation Levels

a) The current isolation level is **READ COMMITTED**.The isolation levels supported by the PostgreSQL are:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

b) Sample table named sheet3 with columns id and name is created. Sample data is inserted into the table.

c) The autocommit is set to off with the command

$$\textbf{\textbackslash set} \ \text{AUTOCOMMIT} \ \text{off};$$

Query one row from table sheet3 and find out the currently held locks:

Result of querying a row and checking locks:

SELECT ∗ **FROM** sheet3 **WHERE** id = 1;

```
 id |  name
----+--------
  1 |  John
```

```
SELECT relation::regclass, mode, granted
FROM pg_locks
WHERE relation::regclass = 'sheet3'::regclass;
```

The result shows no locks on the sheet3 table. This is because in READ COMMITTED isolation level, PostgreSQL acquires row-level read locks during query execution but releases them immediately after the statement completes. Since our SELECT query finished before we checked for locks, any locks that were held had already been released.

d) The transaction is started with the command

```
--- Transaction 1
BEGIN;
SELECT * FROM sheet3 WHERE id = 1;
COMMIT;
```

– Serializable Transaction

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SELECT * FROM sheet3 WHERE id = 1;
COMMIT;
```

Steps repeated for the serializable transaction:

The locks held by the transaction are:

```
relation |       mode       | granted
-----------+------------------+----------
sheet3    | AccessShareLock  | t
sheet3    | SIReadLock       | t
(2 rows)
```

Locks held by the transaction:

- Serializable Read Lock

Serializable transaction acquires a row-level read lock on the sheet3 table and holds it until the transaction is committed.

1.2 3.2 Lock Conflicts

a) First connection with isolation level **READ COMMITTED** and **AUTOCOMMIT off**

```
SELECT * FROM sheet3 WHERE id > 3;
```

```
id |  name
----+---------
  4 | Nasrul
  5 | Raj
  6 | Aldo
  7 | Alia
  8 | Ayesha
(5 rows)
```

Second connection with isolation level **READ COMMITTED** and **AUTOCOMMIT on**

Inserts a new row into the sheet3 table;

2

INSERT INTO sheet3 (id, name) **VALUES** (9, 'Bill');

First connection repeating the query;

SELECT * **FROM** sheet3 **WHERE** id > 3;

```
id |  name
----+--------
4  | Nasrul
5  | Raj
6  | Aldo
7  | Alia
8  | Ayesha
9  | Bill
(6 rows)
```

The new row is visible. Even though the first connected has not yet committed, when the SELECT query is run again in the same transaction, the new row is visible. This is because the second connection has committed the new row and the first connection is in READ COMMITTED isolation level.

b) First connection's isolation level is set to **REPEATABLE READ** with **AUTOCOMMIT off**

SET TRANSACTION ISOLATION LEVEL REPEATABLE **READ**;

Repeats the query:

SELECT * **FROM** sheet3 **WHERE** id > 3;

```
id |  name
----+--------
 4 | Nasrul
 5 | Raj
 6 | Aldo
 7 | Alia
 8 | Ayesha
 9 | Bill
(6 rows)
```

Whereas in the second connection, a new row is inserted at the same time:

INSERT INTO sheet3 (id, name) **VALUES** (10, 'Chris');

Repeats the query in the first connection:

SELECT * **FROM** sheet3 **WHERE** id > 3;

```
id |   name
---+--------
 4 | Nasrul
 5 | Raj
 6 | Aldo
 7 | Alia
 8 | Ayesha
 9 | Bill
(6 rows)
```

The new row is not visible to the first connection. This is because the first connection is in REPEATABLE READ isolation level and the second is in READ COMMITTED isolation level. The second connection has committed the new row but the first connection has not yet committed.

REPEATABLE READ takes a snapshot of the database at the start of the transaction, not at the start of each statement (unlike READ COMMITTED)

The locks held before the commit of the first transaction are:

SELECT relation::regclass, mode, granted
FROM pg_locks
WHERE relation::regclass = 'sheet3'::regclass;

```
relation | mode | granted
---------+------+--------
sheet3   | AccessShareLock | t
sheet3   | SIReadLock      | t
(2 rows)
```

After the commit of the first transaction, the newly inserted row is visible in the first transaction.

```
 id |   name
----+--------
  4 | Nasrul
  5 | Raj
  6 | Aldo
  7 | Alia
  8 | Ayesha
  9 | Bill
 10 | Chris
(7 rows)
```

c) Update one row in the first connection with isolation level **REPEATABLE READ** and **AUTOCOMMIT off**

UPDATE sheet3 **SET** name = 'Brock' **WHERE** id = 1;

Haven't committed the transaction yet. While in the second connection, another row is updated with id 2:

```sql
UPDATE sheet3 SET name = 'Shayne' WHERE id = 2;
```

And again in the second connection, the same row with id = 1 is updated with name = 'Shayne':

```sql
UPDATE sheet3 SET name = 'Shayne' WHERE id = 1;
```

The transaction will now be blocked, because the first connection has not yet committed.

The isolation level does not matter for this particular blocking behavior. Whether using READ COMMITTED or REPEATABLE READ, PostgreSQL always prevents concurrent updates to the same row (this prevents "lost updates").

Only after connection 1 commits or rolls back, releasing its lock on row id=1, will connection 2's update proceed. At that point, connection 2's update will overwrite connection 1's changes to that row.