

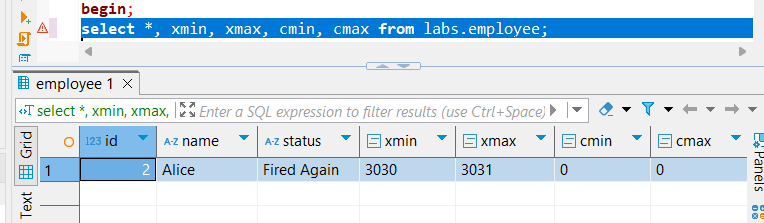
| Business Template  **Transactions** |
| --- |
|  |

### Observations & Issues Encountered

* **Rollback Requirement on Query Errors**: When I attempt to execute the required tasks, any mistake within a query causes the transaction to fail. I must explicitly roll back the transaction to proceed, as the transaction state remains failed until reset.
* **Existing Transaction Detected**: At times, I encounter the message *"there is already a transaction in progress"*, indicating that a previous transaction was not properly closed.

**No Visible Changes in System Columns**: After including the required commands, I observe that system columns such as cmin and cmax consistently return 0, even when running the same code as in previous tasks. For reference:  
  
 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SHOW TRANSACTION ISOLATION LEVEL;



* **Serializable Isolation Behavior**: While testing serializable isolation, I attempted to simulate a phantom read anomaly by inserting a new row from a separate session during an ongoing transaction. The system correctly prevented the anomaly, demonstrating the expected behavior of serializable isolation.
* **Read Committed Isolation Behavior**:  
  + When using READ COMMITTED isolation, I was unable to update an entry if another session was holding an open transaction on the same data. I had to commit the first session before proceeding with the update in the second.
  + This isolation level also appeared to introduce longer loading times in some cases.

### Task Execution Steps

#### **0) Table Creation and Row Insertion**

-- Drop existing table if it exists

DROP TABLE IF EXISTS labs.employee;

-- Begin a new transaction for table creation

BEGIN;

CREATE TABLE IF NOT EXISTS labs.employee (

ID SERIAL,

name VARCHAR(255),

status VARCHAR(255)

);

COMMIT;

-- Insert initial data

INSERT INTO labs.employee (name, status) VALUES

('Alice', 'active'),

('Bob', 'inactive');

#### **1) Serializability Anomaly (Phantom Read Simulation)**

**Session 1**

BEGIN ISOLATION LEVEL SERIALIZABLE;

-- Read current active employees

SELECT \* FROM employee WHERE status = 'active';

-- Wait...

-- Insert another active employee

INSERT INTO employee (name, status) VALUES ('Diana', 'active');

-- Inspect transaction and command metadata

SELECT \*, xmin, xmax, cmin, cmax FROM employee;

COMMIT;

*This scenario attempts to demonstrate a phantom read. Initially, only one active employee is visible. After the insert and before commit, more than one "active" employee is present—representing a potential serialization anomaly.*

**Session 2**

BEGIN ISOLATION LEVEL SERIALIZABLE;

-- Insert a new active employee

INSERT INTO employee (name, status) VALUES ('Charlie', 'active');

-- Check internal transaction metadata

SELECT \*, xmin, xmax, cmin, cmax FROM employee WHERE name = 'Charlie';

COMMIT;

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### **2) Lost Update Anomaly**

**Session 1**

BEGIN;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-- Read Alice's status

SELECT status FROM labs.employee WHERE name = 'Alice';

-- Attempt to update Alice's status

UPDATE labs.employee SET status = 'Active' WHERE name = 'Alice';

-- Wait...

COMMIT;

**Session 2**

BEGIN ISOLATION LEVEL SERIALIZABLE;

-- Insert a new employee during the other session's transaction

INSERT INTO employee (name, status) VALUES ('Charlie', 'active');

-- View metadata for inserted row

SELECT \*, xmin, xmax, cmin, cmax FROM employee WHERE name = 'Charlie';

COMMIT;