

# **Chapter 17 - Lab**

## **Transactions**

# Transaction Operators on PostgreSQL

- BEGIN
  - Start a transaction block
- COMMIT
  - Commit the current transaction
- ROLLBACK
  - Abort the current transaction
- SAVEPOINT
  - Establish a new savepoint within the current transaction
- ROLLBACK to savepoint
  - Roll back all commands that were executed after the savepoint was established

# Transaction Operators on PostgreSQL

- Example

```
postgres=# BEGIN;
BEGIN
postgres=# UPDATE account SET balance=balance+100;
UPDATE 9
postgres=# ROLLBACK;
ROLLBACK
postgres=#
```

```
postgres=# BEGIN;
BEGIN
postgres=# SELECT count(*) FROM account;
 count
-----
      9
(1개 행)

postgres=# SAVEPOINT db;
SAVEPOINT
postgres=# UPDATE account SET balance=balance+100;
UPDATE 9
postgres=# ROLLBACK to db;
ROLLBACK
postgres=# COMMIT;
COMMIT
postgres=#
```

# Lab Setup

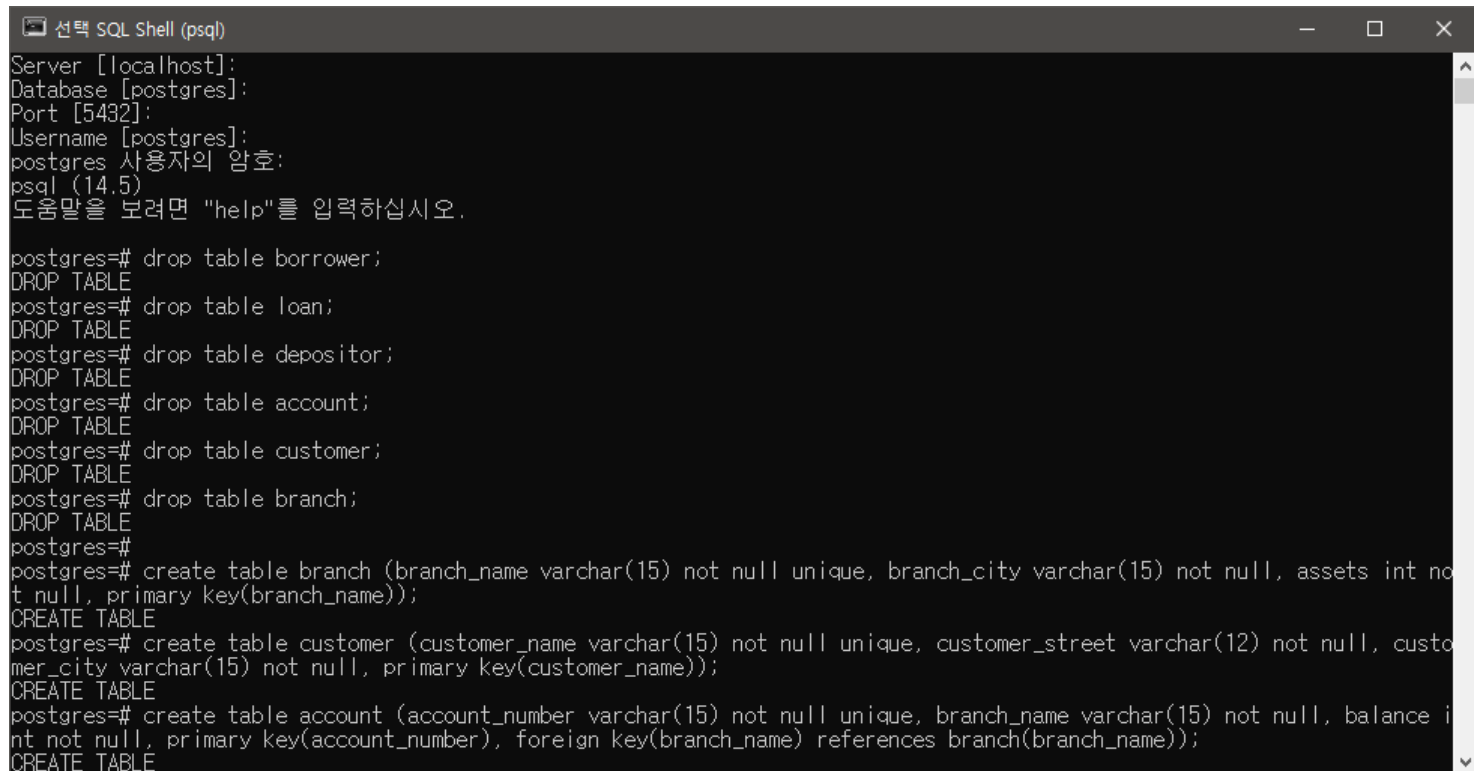
- Execute PostgreSQL **SQL Shell (psql)** and login your database
  - Server [localhost]: Press the enter key
  - Database [postgres]: Press the enter key
  - Port [5432]: Press the enter key
  - Username [postgres]: Press the enter key
  - Password for user postgres: **Type your own password**
  - **\c d{StudentID}**

```
postgres=# \c d202301234
접속정보: 데이터베이스="d202301234", 사용자="postgres".
d202301234=#
```

Your answers must be displayed along with your student ID.

# Lab Setup

- Download the “bank.txt” file from blackboard
- Copy & paste all the contents in the “bank.txt” file on PostgreSQL
  - If you want to reset database, just copy & paste again



```
선택 SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
postgres 사용자의 암호:
psql (14.5)
도움말을 보려면 "help"를 입력하십시오.

postgres=# drop table borrower;
DROP TABLE
postgres=# drop table loan;
DROP TABLE
postgres=# drop table depositor;
DROP TABLE
postgres=# drop table account;
DROP TABLE
postgres=# drop table customer;
DROP TABLE
postgres=# drop table branch;
DROP TABLE
postgres=#
postgres=# create table branch (branch_name varchar(15) not null unique, branch_city varchar(15) not null, assets int not null, primary key(branch_name));
CREATE TABLE
postgres=# create table customer (customer_name varchar(15) not null unique, customer_street varchar(12) not null, customer_city varchar(15) not null, primary key(customer_name));
CREATE TABLE
postgres=# create table account (account_number varchar(15) not null unique, branch_name varchar(15) not null, balance int not null, primary key(account_number), foreign key(branch_name) references branch(branch_name));
CREATE TABLE
```

# “bank” Database Schema

- branch (branch\_name, branch\_city, assets)
- customer (customer\_name, customer\_street, customer\_city)
- account (account\_number, branch\_name, balance)
- depositor (customer\_name, account\_number)
- loan (loan\_number, branch\_name, amount)
- borrower (customer\_name, loan\_number)

※ Be careful regarding the primary-key and foreign-key constraints!  
(e.g. No customers have the same name, ....)

# Exercise 1

- Make transactions for each of the following queries using the transaction operators
  - a. Register a new depositor (with a new account in 'Downtown' and a new customer)
  - b. Eliminate everything created above
  - c. Transfer balance 200 from customer 'Johnson' (at branch 'Downtown') to customer 'Smith' (at branch 'Mianus'). You must update the assets of branches accordingly.

## Exercise 2

- Make transactions for the following queries using the transaction operators
  - a. Transfer balance 100 from 'Lindsay' to 'Turner'
    - You don't need consider assets of the branches in this exercise
  - b. Set all balance by 0 in the account table, and make a system crash before committing
    - You can make a crash by just closing the window
    - Then, open PostgreSQL again and check whether the update is applied



## Exercise 3

- Make a transaction of Exercise 2.a. with establishing savepoint
  - Just before committing, the recipient is changed into 'Jones'. Using “SAVEPOINT” and “ROLLBACK to savepoint” operators, make additional SQL expressions that apply the change of the recipient.
    - You don't need consider assets of the branches in this exercise

# Exercise 4

- Execute the following queries using transaction operators
  - a. Open two PostgreSQL windows simultaneously.  
In the window A, begin the transaction and increase Lindsay's balance by 300 without commit.  
In the window B, increase Lindsay's balance by 200.
    - 1) Is the query of the window B executed?
    - 2) If no, compare the result of two cases
      - Case 1: commit in the window A
      - Case 2: rollback in the window A
  - b. Open two PostgreSQL windows simultaneously.  
In the window A, begin the transaction and increase Lindsay's balance by 300 without commit.  
In the window B, increase **Smith**'s balance by 200.
    - 1) Is the query of the window B executed?
    - 2) If no, compare the result of two cases
      - Case 1: commit in the window A
      - Case 2: rollback in the window A

# Three Phenomena of Inconsistency caused by Non-serializable Concurrent Executions

- The phenomena (defined by ANSI SQL) which are prohibited at various levels are:
  - Dirty write
  - Dirty read
    - A transaction reads data written by a concurrent uncommitted transaction
  - Non-repeatable read
    - A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read)
  - Phantom read
    - A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction

# Transaction Isolation

- The SQL standard defines four levels of transaction isolation
  - Read uncommitted
    - A transaction reads data written by a concurrent uncommitted transaction
  - Read committed
    - A statement can only see rows committed before it begins
  - Repeatable read
    - All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction
  - Serializable
    - Any concurrent execution is guaranteed to produce the same effect as running them one at a time in some order

# Standard SQL Transaction Isolation Level

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

# Transaction Isolation in PostgreSQL

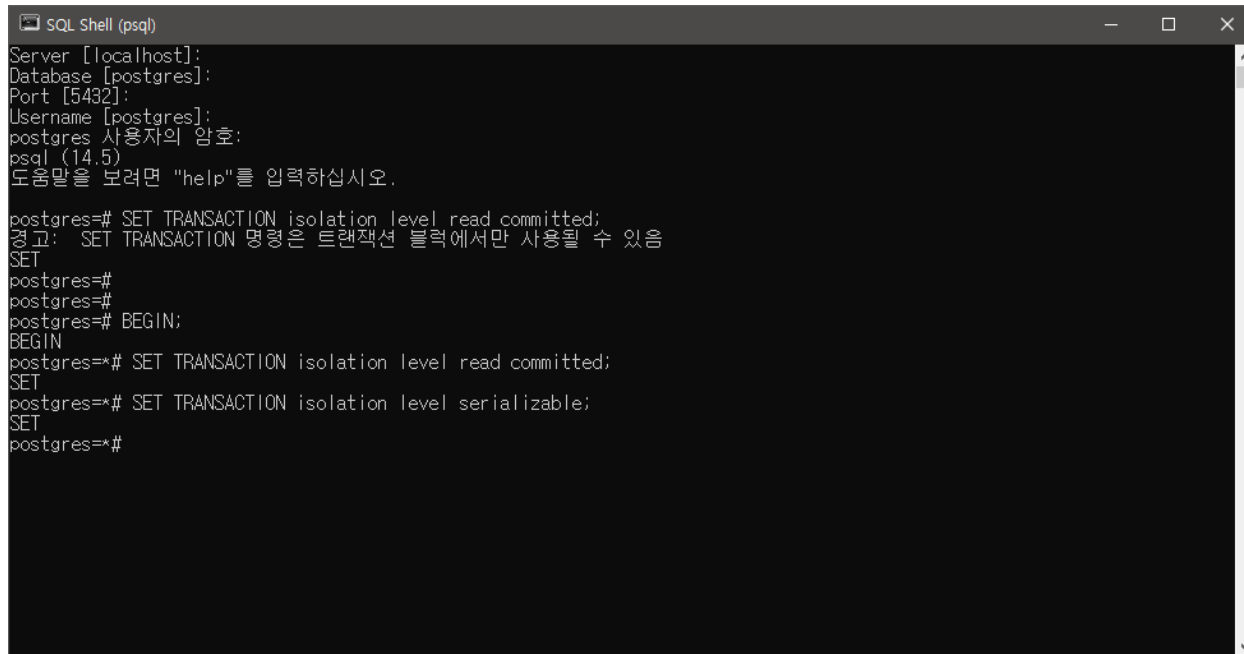
- User can request four standard isolation levels in PostgreSQL
- However, internally, there are only two distinct isolation levels
  - Read committed (default)
    - Read uncommitted is treated as read committed
  - Serializable
    - Repeatable read is treated as serializable

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Not possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Not possible
Serializable	Not possible	Not possible	Not possible

Transaction Isolation Level in PostgreSQL

(This is permitted by the SQL standard: The four isolation levels only define which phenomena must not happen; they do not define which phenomena must happen.)

# Example – SET TRANSACTION



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
postgres 사용자의 암호:
psql (14.5)
도움말을 보려면 "help"를 입력하십시오.

postgres=# SET TRANSACTION isolation level read committed;
경고: SET TRANSACTION 명령은 트랜잭션 블록에서만 사용될 수 있음
SET
postgres=#
postgres=#
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION isolation level read committed;
SET
postgres=# SET TRANSACTION isolation level serializable;
SET
postgres=#
```

Note that if SET TRANSACTION is executed without a prior START TRANSACTION or BEGIN, it emits a warning and otherwise has no effect.

# Exercise 5

- Make scenarios for each of the followings.
  - a. Dirty reads are possible vs. Dirty reads are not possible.
  - b. Non-repeatable reads are possible vs. Non-repeatable reads are not possible
  - c. Phantom reads are possible vs. Phantom reads are not possible
  
- Hint: Open two PostgreSQL windows simultaneously. For each window, set the transaction isolation level and compose SQL sentences appropriately.



# Homework

- Complete today's practice exercises
- Write your queries and take screenshots of execution results
- Submit your report on blackboard
  - 10:29:59, November 28th, 2024
  - **Only PDF files** are accepted
  - **No late submission**

**End of Lab**