

Practice 6

Transaction Processing



hyunsikyon@korea.ac.kr

Transaction Operators on PostgreSQL

- Transaction operations 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

- Begin;

Work; (including update, insert and delete)

Commit; or Rollback;

- Begin;

Work *A*;

Savepoint *sp*;

Work *B*;

Rollback to *sp*;

Commit; or Rollback;

DB setup

- Import the given SQL script file (“data_generation.txt”)
 - psql: \i [filepath]/data_generation.txt
 - cmd: psql -U postgres postgres < data_generation.txt
 - Else, Copy & paste the contents of the script file on the psql
- Consider the “bank” database schema
 - *branch (branch-name, branch-city, assets)*
 - *account (account-number, branch-name, balance)*
 - *depositor (customer-name, account-number)*
 - *customer (customer-name, customer-street, customer-city)*
 - *loan (loan-number, branch-name, amount)*
 - *borrower (customer-name, loan-number)*
- Be careful regarding the foreign-key constraints!

Exercise 1

- Make transactions for the following queries using SQL expressions
 1. Begin;
register a new depositor (with arbitrary values)
Commit;
 2. Begin;
eliminate the depositor
Commit;
 3. Begin;
transfer balance 200 from depositor A(at branch 'a') to B(at branch 'b')
(You must consider the assets of the branches)
Commit;

Exercise 2

- Using the transaction operations, execute following queries on PostgreSQL

1. Begin;

Transfer balance 500 from 'Johnson' to 'Turner'

Commit;

- Note. You don't have to consider assets of the branches in this exercise. In addition, you don't need to consider the numbers of '*Johnson*' and '*Turner*' tuples

2. Begin;

Increase all balance by 500 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

Transaction Isolation

- The SQL standard defines four levels of transaction isolation.
 - Read uncommitted
 - Read committed
 - Repeatable read
 - Serializable

Read Phenomena

- The phenomena which are prohibited at various levels are:
 - Dirty read
 - Non-repeatable read
 - Phantom read

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

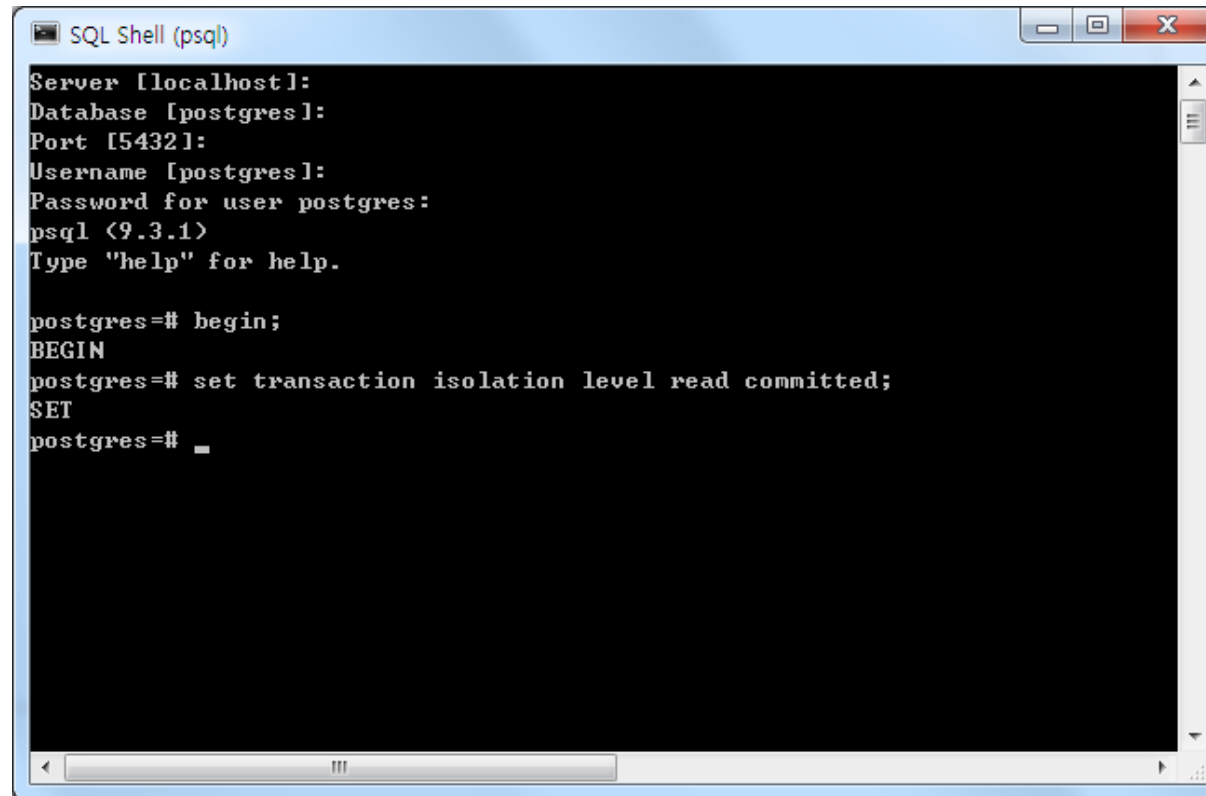
Transaction Isolation in PostgreSQL

- 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

(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]:
Password for user postgres:
psql <9.3.1>
Type "help" for help.

postgres=# begin;
BEGIN
postgres=# set transaction isolation level read committed;
SET
postgres=#
```

- Note that if *SET TRANSACTION* is executed without a prior *START TRANSACTION* or *BEGIN*, it will appear to have no effect, since the transaction will immediately end.

Exercise 3

- Open two PostgreSQL windows simultaneously.
In the window A, begin a transaction with the isolation level *Read Committed*.
 1. *Dirty read*
In the window B, begin a transaction and increase A-215's balances by 1000 **WITHOUT** Commit.
 - Is the update in the window B applied to the transaction of the window A?
If yes, why? If no, why?
 2. *Non-repeatable read*
In the window A, Check the A-201's balance using an appropriate SQL query and,
In the window B, begin a new transaction and increase A-201's balance by 1000 **WITH** Commit.
 - Is the update applied to the A-201's balance in the transaction of the window A?
If yes, why? If no, why?

Exercise 3 (cont'd)

- Open two PostgreSQL windows simultaneously.
In the window A, begin a transaction with the isolation level *Read Committed*.
 - 3. *Phantom read*
In the window A, Check all tuples in the Account relation using an appropriate SQL query and,
In the window B, begin a transaction and insert an account into account relation with values ('A-999', 'Anam', 1000).
 - Is the insertion applied to the Account relation in the transaction of the window A?
If yes, why? If no, why?

Exercise 4

- Make two transactions with the isolation level *Serializable*.
Open two PostgreSQL windows simultaneously.
In the window A, begin a transaction with the isolation level *Serializable*.
 1. *Dirty read*
In the window B, begin a transaction and increase A-215's balances by 1000 **WITHOUT** Commit.
 - Is the update in the window B applied to the transaction of the window A?
If yes, why? If no, why?
 2. *Non-repeatable read*
In the window A, Check the A-201's balance using an appropriate SQL query and,
In the window B, begin a new transaction and increase A-201's balance by 1000 **WITH** Commit.
 - Is the update applied to the A-201's balance in the transaction of the window A?
If yes, why? If no, why?

Exercise 4 (cont'd)

- Open two PostgreSQL windows simultaneously.
In the window A, begin a transaction with the isolation level *Read Committed*.
 - 3. *Phantom read*
In the window A, Check all tuples in the Account relation using an appropriate SQL query and,
In the window B, begin a transaction and insert an account into account relation with values ('A-999', 'Anam', 1000).
 - Is the insertion applied to the Account relation in the transaction of the window A?
If yes, why? If no, why?

Homework 7

- Follow the directives in this slides
- Take screenshots of your queries and execution results
- Submit your homework online (blackboard)
 - Deadline
 - 10:30 am, December 17th, 2019
 - **Only PDF files** are accepted
 - **No late submissions!**
 - **No plagiarism!**

End



KOREA UNIVERSITY
DATABASE LAB

hyunsikyon@korea.ac.kr