

PSQL TP5 (Transactions & Logging)

Subject: PostgreSQL

Students: BOURBAI Ismail

I. Transaction:

Definition:

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

Properties of Transactions:

Transactions have the following four standard properties, usually referred to by the acronym ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability).

Transaction Control:

The following commands are used to control transactions:

1. **BEGIN TRANSACTION:** To start a transaction.
2. **COMMIT:** To save the changes, alternatively you can use END TRANSACTION command.
3. **ROLLBACK:** To rollback (cancel) the changes.

PS: Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

Example:

Consider the **COMPANY** table is having the following records:

```
transaction=# select * from company;
```

id	name	age	address	salary
1	Paul	32	California	20000
2	Allen	25	Texas	15000
3	Teddy	23	Norway	20000
4	Mark	25	Rich-Mond	65000
5	David	27	Texas	85000

(5 rows)

Figure 1 - Table Rows

Now, let us start a transaction and delete records from the table having age = 25 and finally we use ROLLBACK command to undo all the changes.

```
transaction=# BEGIN;
BEGIN
transaction=# DELETE FROM company WHERE age = 25;
DELETE 2
transaction=# ROLLBACK;
ROLLBACK
transaction=# select * from company;
```

id	name	age	address	salary
1	Paul	32	California	20000
2	Allen	25	Texas	15000
3	Teddy	23	Norway	20000
4	Mark	25	Rich-Mond	65000
5	David	27	Texas	85000

(5 rows)

Transaction Control

The Result : nothing change

Figure 2 - Use Rollback

Now, let us start another transaction and delete records from the table having age = 25 and finally we use COMMIT command to commit all the changes.

```

transaction=# BEGIN;
BEGIN
transaction=# DELETE FROM company WHERE age = 25;
DELETE 2
transaction=# COMMIT;
COMMIT
transaction=# SELECT * FROM company;

```

id	name	age	address	salary
1	Paul	32	California	20000
3	Teddy	23	Norway	20000
5	David	27	Texas	85000

(3 rows)

Transaction Control

The Result: 2 records deleted!

Figure 3 - Use Commit

II. Logging:

Enabling logging within PostgreSQL is made quite easy by altering a handful of configuration settings and then restarting the server. While these settings can be altered “in memory”, thereby enabling temporary logging for only that particular client session, we’ll see how to configure postgres to permanently create iterative log files for all sessions and connections.

Locating the Configuration File:

The first thing we must know where the `postgresql.conf` config file is located, the simplest method for finding the location is to execute this command:

```

transaction=# SHOW config_file;
config_file

```

C:/Program Files/PostgreSQL/11/data/postgresql.conf

(1 row)

The Command

Location of config file

Figure 4 - Show Config_File Command

Now just open that file with your favorite text editor and we can start changing settings.

Configuring PostgreSQL to Generate Log Output:

With the `postgresql.conf` file open, scroll down to the **ERROR REPORTING AND LOGGING** section and you’ll likely see a number of configuration options commented out. The most critical of these settings are **log_destination** and **logging_collector**. Below are the recommended settings, though feel free to change these to suit your own needs:

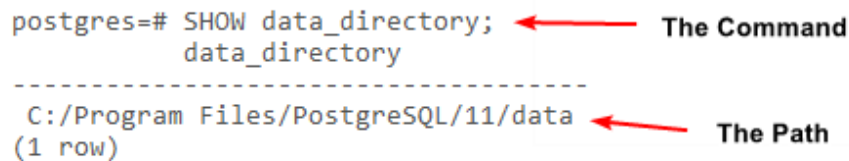
- `log_destination = 'csvlog'` (*where to log*)
- `logging_collector = on` (*Enable capturing of stderr and csvlog into log files*)
- `log_directory = 'pg_log'` (*directory where log files are written*)
- `log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'` (*log file name pattern*)

Here we’re telling postgres to generate logs in the CSV format and to output them to the `pg_log` directory (within the data directory). We’ve also uncommented the `log_filename` setting to produce some proper name including timestamps for the log files.

The final step is to restart the PostgreSQL service so that these settings, in particular logging_collector, will take effect

Verifying Log Generation

First we need the path of the **data** directory for your postgres installation, and retrieving the path is a matter of another simple SHOW statement:



```
postgres=# SHOW data_directory;
data_directory
-----
C:/Program Files/PostgreSQL/11/data
(1 row)
```

The Command

The Path

Figure 5 - Show Directory

Once the system has been restarted logging should begin immediately. To ensure this is the case, navigate to the data/pg_log directory of your postgres installation, so simply navigate to that directory by adding /pg_log to the end to get into the log directory.

Now you should see a log file has been created following the previous service restart.

There we have it; automatically generated log files are enabled with PostgreSQL by changing just a few configuration settings.

THANKS!