Postgresql Notes

Notes:

- Postgresql - Installing
- Postgresql - Upgrading

Working with postgresql

- Postgresql documentation on psql command: http://www.postgresql.org/docs/9.0/static/app-psql.html
- How to run script from command line (\i <file_name>): http://stackoverflow.com/questions/9736085/run-a-postgresql-sql-file-using-command-line-args
- View running queries: http://www.chrismiles.info/systemsadmin/databases/articles/viewing-current-postgresql-queries/

Config files

- Find location of config file using SQL:
  - `SHOW config_file;`
- How to configure the pg_hba.conf file:
  - https://www.postgresql.org/docs/9.5/static/auth-pg-hba-conf.html

psql

- \x = extended view - easier on the eyes.
- run an SQL script file: \i <path_to_sql_file>

GUI clients

- http://www.psequel.com/
- http://www.sqltabs.com/
- http://dbeaver.jkiss.org/download/
- https://eggerapps.at/postico/
- https://www.pgadmin.org/

Databases

Connecting to database

- log in with the posgres user
  - `su - postgres` (use password set when you installed)
- start the pgsql client
  - `psql`
  - `options:`
    - `-U <username>`
    - `-W # force password prompt`
    - `-d <database>`
- first, make sure you know what databases are in your instance.
  - list databases: `\list`
- then, connect to a database and look for your table.
  - connect to a database: `\c <database_name>` OR `\connect <database_name>`
  - list tables in a database: `\dt`

Creating a database:

- `su - postgres`
- In sql client:
  - `CREATE DATABASE <new_db_name>;`
    - ex: `CREATE DATABASE socs_twitter;`
- At shell prompt:
  - `createdb <database_name>`
    - ex: `createdb socs_twitter`
- NOTE: Postgres expects database names to be all lower case. If you really want a database name with upper case, enclose the name in quotation marks when you pass it to created.
  - Yeah, not sure why that is.
- Create database by duplicating existing database (using "template"):
  - In psql client:
    - `CREATE DATABASE <new_db_name> WITH TEMPLATE <original_db_name> OWNER <db_user>;`
    - Example:
      - `CREATE DATABASE sourcenet_test WITH TEMPLATE jsm_sourcenet OWNER django_user;`
  - At shell prompt, logged in as postgres user:
    - `createdb -O <db_user> -T <original_db_name> <new_db_name>`
  - More: http://stackoverflow.com/questions/876522/creating-a-copy-of-a-database-in-postgresql

Deleting a database

- start up postgresql psql client per instructions above.
- to delete a database:
    - `DROP DATABASE <database_name>;`

Rename a database

- start up postgresql client per instructions above
- run the following SQL:
    - `ALTER DATABASE "sourcenet" RENAME TO "sourcenet_dev";`

List tables in a database

- start up postgresql client per instructions above
- connect to the database whose tables you want to list: `\c <database_name>`
- list tables in a database: `\dt`
- `list tables in a database: \d+ <table_name>`

List columns in a table in a database

- start up postgresql client per instructions above
- connect to the database whose tables you want to list: `\c <database_name>`
- list columns in tables in a database: `\d+ <table_name>`

List active sessions in a database

- start up postgresql client per instructions above
- http://www.devopsderek.com/blog/2012/11/13/list-and-disconnect-postgresql-db-sessions/
- run the following SQL:
    - `SELECT * FROM pg_stat_activity;`
- To cancel processes (pid = the contents of the field “pid” in the query above):
    - `SELECT pg_cancel_backend( <pid> );`
- To cancel sessions (pid = the contents of the field “pid” in the query above):
    - `SELECT pg_terminate_backend( <pid> );`

Schemas

List out schemas

- in psql:

- `\dn`

List out tables in schema

- in psql:
  - `\dt <schema>.*`

Create schema

- SQL:
  - `CREATE SCHEMA test_schema;`

Choose default schema(s)

- SQL:
  - `SET search_path TO <schema> (, <schema>);`

Change table of schema

- SQL:
  - `ALTER TABLE <table> SET SCHEMA <schema>;`

Scripts

Running external scripts

- at unix command line
  - `psql <database_name> < <file_path>`
  - OR `psql -d <database_name> -f <file_path>`
    - this will give you more descriptive output.
    - http://petereisentraut.blogspot.com/2010/03/running-sql-scripts-with-psql.html
- in psql:
  - base command is to use "\i <file_path>". Example:
    - `\i output_comments_as_csv.sql`
  - if the script name has spaces in it, you can surround the file name with single quotes.
  - current directory is directory you ran psql in.
  - if you want to see output, set ECHO to all.

Users

List out users

- start up postgresql client per instructions above
- then, connect to the database that you want to work in.
    - connect to a database: `\c <database_name>`
    - list users in a database: `\du`

Creating a user

- Create your own database superuser (logged in as user postgres).
    - `su - postgres`
    - just create a superuser:
        - `createuser -P -s <username>`
            - enter the password for the user.
    - OR `createuser --interactive -P <username>`
        - enter a password
        - decide if you want the user to be a super-user or not.
        - if not (a django user shouldn't be, for example), then:
            - decide if you want user to be able to create databases (no).
            - decide if you want user to be able to create more users (no).
- creating a non-superuser user and giving them access to a database.
    - first, create a user in postgresql:
        - log in as postgres
        - `createuser -P <username>`
            - not superuser, don't allow creation of databases or users.
- SQL to create a non-superuser user and giving them access to a database.
    - open the postgresql client (run psql when logged in as postgres), then run:
        - `CREATE ROLE <username> WITH LOGIN;`
    - If this doesn't grant LOGIN or if you need to add it later, you can alter role:
        - `ALTER ROLE <username> WITH LOGIN;`
- then, grant privileges on your database:
    - open the postgresql client (run psql when logged in as postgres)
        - `GRANT ALL PRIVILEGES ON DATABASE <database> TO <username>;`

View all grants

- In psql, the "\z".

Alter role

- SQL to alter a role:
    - open the postgresql client (run psql when logged in as postgres) then run:
        - `ALTER ROLE <username> WITH <privilege>;`
    - Example, to make a user able to LOGIN:
        - `ALTER ROLE <username> WITH LOGIN;`

Make a user a member of a role

- start up postgresql psql client per instructions above.
- `GRANT <role> TO <role>;`
    - where `<role>` can either be a user or a group. In this case, first role should be one for a group, second should be one for a user.
- On success, will just output “GRANT”.

Grant privileges on a database to a user

- start up postgresql psql client per instructions above.
- `GRANT ALL PRIVILEGES ON DATABASE <database> TO <username>;`
- On success, will just output “GRANT”.

Grant privileges on a given schema to a user

- connect to database that contains table you want to connect to (\connect <database_name>).
- `GRANT ALL PRIVILEGES ON SCHEMA <schema_name> TO <username>;`
- example:
    - `\c sourcenet`
    - `GRANT ALL PRIVILEGES ON SCHEMA public TO django_user;`

Grant privileges on all tables in a given schema to a user

- connect to database that contains table you want to connect to (\connect <database_name>).
- `GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA <schema_name> TO <role>;`
    - where `<role>` can either be a user or a group.
- example:
    - `\c sourcenet`
    - `GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO`

```
django_user;
```

Grant privileges on all sequences in a given schema to a user

- connect to database that contains table you want to connect to (\connect <database_name>).
- `GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA <schema_name> TO <username>;`
- example:
    - `\c sourcenet`
    - `GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO django_user;`

Grant privileges on a table to a user

- connect to database that contains table you want to connect to (\connect <database_name>).
- `GRANT ALL PRIVILEGES ON TABLE <table> TO <username>;`

Revoke privileges

- To remove privileges, you use the `REVOKE` command. `REVOKE` syntax is identical to all the `GRANT` examples above, except you use "`REVOKE...FROM...`" rather than "`GRANT...TO`".

Deleting a user

- either log in as postgres user (removes user and associated tables):
    - run dropuser command:
        - `dropuser <username>`
- OR start up postgresql client per instructions above (doesn't remove associated tables, just removes user).
    - to remove a user:
        - `DROP USER <username>;`
        - *always put a semi-colon at the end of any postgresql command!* If you don't, weird things will happen involving multiple-line commands.

Allow user to login using a password:

- Add to /etc/postgresql/9.5/main/pg_hba.conf:
    - `local   all    <username>       md5`
    - `local   all    django_user      md5`

More details:

- http://stackoverflow.com/questions/6799224/pgsql-grant-every-single-right-to-a-user-on-a-schema
- https://www.digitalocean.com/community/tutorials/how-to-use-roles-and-manage-grant-permissions-in-postgresql-on-a-vps--2

Tables

Creating a table

- Example:

```
CREATE TABLE epictable
(
    mytable_key    SERIAL PRIMARY KEY,
    moobars        VARCHAR(40) not null,
    foobars        DATE
);
```

- for the primary key, “SERIAL” maps to an “int” primary key. “BIGSERIAL” corresponds to a bigint primary key.

List out tables

- start up postgresql client per instructions above
- then, connect to the database that you want to work in.
    - connect to a database: `\c <database_name>`
    - list tables in a database: `\dt`

List columns in a table

- start up postgresql client per instructions above
- then, connect to the database that you want to work in.
    - connect to a database: `\c <database_name>`
    - list columns in a database table: `\d+ <table_name>`

Deleting a table

- start up postgresql client per instructions above
- then, connect to the database that contains the table you want to delete.
    - connect to a database: `\c <database_name>`
    - list tables in a database: `\dt`
- to remove a table:
    - `DROP TABLE <table_name> [, <table_name>];`
    - *always put a semi-colon at the end of any postgresql command!* If you don't, weird things will happen involving multiple-line commands.

Updating a table

- PostgresSQL update syntax is different.
- http://stackoverflow.com/questions/7869592/how-to-do-an-update-join-in-postgresql
- http://stackoverflow.com/questions/13473499/update-a-column-of-a-table-with-a-column-of-another-table-in-postgresql
- http://www.postgresql.org/docs/9.2/static/sql-update.html
- Example:
    - ```
      UPDATE reddit_collect_comment rcc
      SET subreddit_id = rcp.subreddit_id
      FROM reddit_collect_post rcp
      WHERE ( ( rcc.subreddit_id IS NULL ) OR ( rcc.subreddit_id <
      0 ) )
         AND rcc.post_id = rcp.id
         AND ( ( rcp.subreddit_id IS NOT NULL ) AND (
      rcp.subreddit_id > 0 ) );
      ```
- Note:
    - in SET, you don't add the table abbreviation before column(s) you are setting (in the table that is in the UPDATE clause itself), just for any values you reference that are in tables in the FROM clause.

Sequences

- Listing sequences in a database
    - `SELECT c.relname FROM pg_class c WHERE c.relkind = 'S';`
- Reset sequence to start with X:
    - http://stackoverflow.com/questions/4678110/how-to-reset-sequence-in-postgres-and-fill-id-column-with-new-data
    - Reset sequence to start with 1:
        - `ALTER SEQUENCE <sequence_name> RESTART WITH 1;`
    - restart with the next value you want assigned for the sequence.
- For Django tables, the standard way the sequence is named is:
    - `<table_name>_id_seq`
    - so for table sourcenet_analysis_reliability_names, sequence would be named: `sourcenet_analysis_reliability_names_id_seq`
    - And you'd reset to id of 1 with SQL:
      ```

        - ALTER SEQUENCE sourcenet_analysis_reliability_names_id_seq RESTART WITH 1;
- Good troubleshooting:
    - SELECT MAX(id) as max_id, MIN(id) as min_id, COUNT(*) as record_count
    - FROM sourcenet_analysis_reliability_names
    - WHERE label = 'prelim_reliability_combined';

Add primary key:

- ALTER TABLE your_table ADD COLUMN key_column BIGSERIAL PRIMARY KEY;

Add unique constraint key:

- First, create the column to hold the ID value (same type as the ID column it references if referring to a single column).
    - ALTER TABLE <table_name> ADD COLUMN <column_name> <type>;
- Then, create a REFERENCES CONSTRAINT:

```
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name>_fk
FOREIGN KEY ( <column_name> )
REFERENCES <parent_table> ( <parent_table_pk> );
```

- From:
    - http://stackoverflow.com/questions/1194438/can-i-add-a-unique-constraint-to-a-postgresql-table-after-its-already-created
    - http://www.postgresqltutorial.com/postgresql-foreign-key/

Add foreign key column:

- ALTER TABLE the_table ADD CONSTRAINT constraint_name UNIQUE (thecolumn);
- From: http://stackoverflow.com/questions/1194438/can-i-add-a-unique-constraint-to-a-postgresql-table-after-its-already-created

Query primary key columns:

- From: https://wiki.postgresql.org/wiki/Retrieve_primary_key_columns

```
SELECT a.attname AS a_att_name, format_type( a.atttypid, a.atttypmod ) AS data_type
FROM pg_index i
JOIN pg_attribute a ON a.attrelid = i.indrelid
    AND a.attnum = ANY(i.indkey)
WHERE i.indrelid = '<tablename>'::regclass
    AND i.indisprimary;
```

Change owner of a table

- Use ALTER TABLE command.
- Change owner of table:
    - `ALTER TABLE <schema>.<table_name> OWNER TO <role>;`
- http://www.postgresql.org/docs/current/static/sql-delete.html

Deleting from a table

- Use DELETE command.
- Delete all:
    - `DELETE FROM <table_name> *;`
- http://www.postgresql.org/docs/current/static/sql-delete.html

---

Indexes

Creating an index

- `CREATE [UNIQUE] INDEX index-name ON table-name( column [,...] );`
- Example:
    - `CREATE INDEX ildoc_admit_person_id_index ON ildoc_admit( person_id );`

List out indexes

- http://stackoverflow.com/questions/2204058/list-columns-with-indexes-in-postgresql

List indexes in a table

- start up postgresql client per instructions above
- then, connect to the database that you want to work in.
    - connect to a database: `\c <database_name>`
    - list columns in a database table: `\d+ <table_name>`

Deleting an index

- start up postgresql client per instructions above
- then, connect to the database that contains the table you want to delete.
    - connect to a database: `\c <database_name>`
- to remove an index:
    - `DROP INDEX <schema>.<index_name> [, <index_name>];`
    - *always put a semi-colon at the end of any postgresql command!* If you don't, weird things will happen involving multiple-line commands.

Rebuild an index

- REINDEX INDEX <index_name>;
- Can also rebuild all indexes for a table:
    - REINDEX TABLE <table_name>
- https://www.postgresql.org/docs/current/static/sql-reindex.html

---

Import/Export

Exporting

- To CSV:
    - from psql command line:
        - `\copy my_table to 'filename' csv header`
    - SQL:
        - `COPY products_273 TO '/tmp/products_199.csv' DELIMITER ',' CSV HEADER;`
    - More complex example:
        - ```
          COPY
          (

             SELECT id, reddit_id, reddit_full_id, link_id,
          parent_reddit_id, author_name, post_reddit_id, subreddit_name,
          subreddit_reddit_id, upvotes, downvotes, score, created_utc,
          created_utc_dt

             FROM reddit_collect_comment

             ORDER BY subreddit_reddit_id ASC, post_reddit_id ASC,
          parent_reddit_id ASC
          ```
          ```

```
)

TO '/var/lib/postgresql/1_hour_subreddit-comments.csv' DELIMITER
',' CSV HEADER;
```

- SQL:
  - pg_dump command line command:
    - `pg_dump <database> > <output_file_name>`
    - `pg_dump <database> -f <output_file_name>`
    - `pg_dump -U <username> <database> -f <output_file_name>`
  - options you should be aware of:
    - -O / --no-owner
      - Do not output commands to set ownership of objects to match the original database. By default, pg_dump issues ALTER OWNER or SET SESSION AUTHORIZATION statements to set ownership of created database objects. These statements will fail when the script is run unless it is started by a superuser (or the same user that owns all of the objects in the script). To make a script that can be restored by any user, but will give that user ownership of all the objects, specify -O.
    - -c / --clean
      - Output commands to clean (drop) database objects prior to outputting the commands for creating them. (Unless --if-exists is also specified, restore might generate some harmless error messages, if any objects were not present in the destination database.)
    - -C / --create
      - Begin the output with a command to create the database itself and reconnect to the created database. (With a script of this form, it doesn't matter which database in the destination installation you connect to before running the script.) If --clean is also specified, the script drops and recreates the target database before reconnecting to it.
    - -t / --table
      - Dump only tables with names matching table. For this purpose, "table" includes views, materialized views, sequences, and foreign tables. Multiple tables can be selected by writing multiple -t switches. Also, the table parameter is interpreted as a pattern according to the same rules used by psql's \d commands (see Patterns), so multiple tables can also be selected by writing wildcard characters in the pattern. When using wildcards, be careful to quote the pattern if needed to prevent the shell from expanding the wildcards; see Examples.
    - examples:
      - `pg_dump -O -c -C <database> > <output_file_name>`
      - `pg_dump -O -c -C -t <table_name_1> -t <table_name_2> ... <database> > <output_file_name>`
      - `pg_dump -O -c -C -f <output_file_name> <database>`

- pg_dump documentation page:
  - https://www.postgresql.org/docs/current/static/app-pgdump.html
  - https://www.postgresql.org/docs/devel/static/app-pgdump.html
- examples:
  - http://www.postgresql.org/docs/current/static/backup-dump.html
  - http://www.postgresql.org/docs/devel/static/backup-dump.html

Importing

- SQL:
  - use psql command line tool:
    - `psql <dbname> < <infile>`
    - more complicated options: http://www.postgresql.org/docs/9.1/static/backup-dump.html
  - in psql:
    - connect to database, then base command is to use "\i <file_path>". Example:
      - `\c <database>`
      - `\i output_comments_as_csv.sql`
    - if the script name has spaces in it, you can surround the file name with single quotes.
    - current directory is directory you ran psql in.
    - if you want to see output, set ECHO to all.
  - probably will need to wipe and recreate database, then set up permissions again.
    - to delete a database:
      - `DROP DATABASE <database_name>;`
    - then, create database:
      - In sql client:
        - `CREATE DATABASE <new_db_name>;`
          - ex: `CREATE DATABASE socs_twitter;`
      - At shell prompt:
        - `su - postgres`
        - `createdb <database_name>`
          - ex: `createdb socs_twitter`
    - Import the data.
      - eventually, should figure out how to configure django_user so that it can connect to psql on the command line and run the import, so it owns everything.
    - Grant privileges on the database as before:
      - start up postgresql psql client per instructions above.
      - Grant privileges on database, schema, tables, and sequences to any django users:
        - `GRANT ALL PRIVILEGES ON DATABASE <database> TO <username>;`
          - On success, will just output "GRANT".

        - `\c <database>`
        - `GRANT ALL PRIVILEGES ON SCHEMA public TO <username>;`
        - `GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO <username>;`
        - `GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO <username>;`
- In strings, single quotation marks must be escaped by transforming each single quote into two single quotes. So, “Hi I’m pleased to meet you” would need to be “Hi I”m pleased to meet you”.

Tuning postgresql

Optimizing for high-volume of writes

- postgresql thinks it is writing too frequently to disk if you are getting a warning like the following in the postgresql log:
    - LOG: checkpoints are occurring too frequently (10 seconds apart)
    - HINT: Consider increasing the configuration parameter "checkpoint_segments".
- to "fix", you need to update the checkpoint configuration properties in postgresql.conf so postgresql writes to diskless frequently.
- Those properties are:
    - **checkpoint_segments**: defaults to "3" - in logfile segments, min 1, 16MB each
    - **checkpoint_timeout**: defaults to "5min" - range 30s-1h
    - **checkpoint_completion_target**: defaults to "0.5" - checkpoint target duration, 0.0 - 1.0, where 1 = write 100% of changes to disk every time.
    - **checkpoint_warning**: defaults to "30s" - 0 disables - time between checkpoints below which is too much.
    - **log_checkpoints**: defaults to "off" - default is to not log each checkpoint.
- First step is to increase the "checkpoint_segments", so more changes are cached up between writes of the change log file.
    - recommend starting with 16 or 32, then increasing in increments of 32.
    - 100 or so is not unheard of.
    - http://www.postgresql.org/message-id/1456c1b29ef1ba986cda9b23e70632db@2ndquadrant.it
- More discussion of next steps:
    - http://dba.stackexchange.com/questions/10208/configuring-postgresql-for-write-performance

---

Installing postgresql

- Postgresql - Installing

Upgrading Postgresql

- Postgresql - Upgrading

Using postgresql with django

- for django, install psycopg2 - python connector for postgresql (make sure you installed the libpq-dev package first).
    - `sudo pip install psycopg2`
- create a django postgresql user that is not a superuser, can't create databases or users.
- create a database and add your django_user as an admin for that database.
    - `GRANT ALL PRIVILEGES ON DATABASE <database> TO <username>;`
    - `GRANT ALL PRIVILEGES ON DATABASE <database> TO django_user;`
- it should just work if you created a user and a database.
    - migrations:
    - if you load an import file, either
        - load it as your django database user (I use django_user) - you'll have to make sure you configure pg_hba.conf so that your django user can connect from localhost using psql.
        - or make sure that you assign all privileges on the database, the public schema, and all tables and sequences to your django database user once the import is done. Example using user "django_user":
            - for database: `GRANT ALL PRIVILEGES ON DATABASE <database> TO django_user;`
            - while connected to database:
                - for "public" schema:
                    - `GRANT ALL PRIVILEGES ON SCHEMA public TO django_user;`
                - for all tables:
                    - `GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO django_user;`
                - for sequences:
                    - `GRANT ALL PRIVILEGES ON ALL SEQUENCES IN`

```
SCHEMA public TO django_user;
```

Migrating from mysql to postgresql

- If you are moving to a brand new database, create the database in postgresql:
  - log in as postgres user.
  - at command line:
    - `createdb <database_name>`
  - If the user you will log your import process into postgresql with is not a superuser (say, it is your django_user), grant that user all privileges on the database:
    - open psql client.
    - `GRANT ALL PRIVILEGES ON DATABASE <database_name> TO <username>;`
- With py-mysql2pgsql
  - install libyaml
    - `(sudo) apt-get install libyaml-dev`
  - install py-mysql2pgsql using pip:
    - `(sudo) py-mysql2pgsql`
    - dependencies are psycopg2, pyyaml (this one outputs lots of warnings), and termcolor.
  - try running command with help flag:
    - `py-mysql2pgsql -h`
  - if that works, then you need to create a YAML configuration file telling the program where the source and destination are.
    - cd into a good, comfortable working directory.
    - to create a default file named mysql2pgsql.yml, just run the py-mysql2pgsql command with no flags:
      - `py-mysql2pgsql`
    - rename the file to something descriptive of the migration it specs (for example, sourcenet-m2p.yml).
    - Edit the configuration so that the connection information for mysql and postgresql matches your environment (for more specific information on this, see https://github.com/philipsoutham/py-mysql2pgsql). You can configure it to migrate over a network, but that will be slow. Probably better to have both source and destination on the same machine.
      - you will probably just be entering the standard connection information for mysql and postgresql - username, password, database name, host, port, etc.
      - if you are going to use these tables with django, consider configuring this script to use the django_user on the destination machine, so the tables are accessible to it once they are migrated.

If you don't do this, you'll end up having to GRANT ALL PRIVILEGES on the database and each individual table to the django postgresql user once the import is done.
        - if you leave the socket set to the default ("/tmp/mysql.sock") and you get an error like "_mysql_exceptions.OperationalError: (2002, "Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)")", you need to find the location of the mysql socket:
            - `mysqladmin -u <mysql_username> -p variables | grep socket`
            - on ubuntu 13.10, the mysql socket path is: `/var/run/mysqld/mysqld.sock`
        - if your postgresql or mysql server is configured to answer requests at a certain IP address, you have to connect at that IP address, even if you are on the server.
    - Run the migration:
        - if you are migrating a large database, consider doing this in a screen session, so you can leave it to run in the background.
        - run the migration, pointing it at your configuration file:
            - `py-mysql2pgsql -v -f sourcenet-m2p.yml`
- Once you run the migration, if you are using this with django, if you didn't use the django postgresql user to do the migration, you'll need to grant access to the django postgresql user to the database, and then to each table:
    - for database: `GRANT ALL PRIVILEGES ON DATABASE <database> TO django_user;`
    - while connected to database:
        - for "public" schema:
            - `GRANT ALL PRIVILEGES ON SCHEMA public TO django_user;`
        - for all tables:
            - `GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO django_user;`
        - for sequences:
            - `GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO django_user;`

Notes:

- PostgreSQL has native column types for key-value pair and JSON document stores, making it more like a NoSQL database:
    - http://www.linuxjournal.com/content/postgresql-nosql-database
- there is an actual boolean type in PostgreSQL, not just a tinyint. You have to cast them to int ("::int") to call numeric functions like sum(), max() on them.

- GROUP BY works differently in PostgreSQL than it does in MySQL (I think I like it better, too - more explicit how to treat each column unlike essentially random behavior of mysql): If column is in SELECT, it must either be in an aggregate function or in the GROUP BY clause. Have to test SQL in postgres, MySQL to see how this changes results. In MySQL, it is fine to put all the columns in - the result is the same. So, better to go with PostgreSQL on this one.
- PostgreSQL doesn't like back-ticks in queries (`) - it isn't in the SQL standard, either. MySQL uses them to surround identifiers that use non-standard identifiers, or that are named the same as reserved names. So, for cross-platform, leave back-ticks out of raw queries (and don't name things the same as reserved names).
- PostgreSQL only supports the following INSERT syntax, not the kind where you use a SET clause:
    - `INSERT INTO <table_name> ( <column_name_list> ) VALUES ( <value_list> );`