Django and Materialized View

Django is known for its object-relational mapping (ORM) which is a super powerful mechanism that greatly makes development productive. As each model is associated with each physical table in database, I start wondering if it’s possible to make ORM deal with more advanced database feature, that is view and materialized view. The result turns out to be more than great!

View and materialized view are results of lookup SQL query. The main difference between them is that materialized view stores result as an actual table while view does not. From technical perspective, view and materialized view are good for repetitive query. They enhance query speed. From business perspective, they can be used to declare data scope for the usages of different teams/departments.

Here are conclusions of my research -- ORM works well with materialized view but view. It makes sense, does it? Because materialized view is a physical table. ORM should be able to associate it. Regarding view, ORM has a similar mechanism called “manager” on which I will write another article to walk you through in the future. Let’s continue our tutorial on materialized view.

Dependencies

1. Python 3.7
2. Django 2.2.x
3. Psycopg2-binary 2.8.6

In this tutorial, I am going to use PostgreSQL database. Definitely choose whatever relational database you would like to use. In order to let Python “talks” to PostgreSQL database, we need to pip install psycopg2-binary. In addition, you may need to set up a local PostgreSQL database on your own machine so that models in your Django project can connect.

Step 1. Build model to create an origin table

In models.py in **polls** app, declare a **Question** model as below.



Run migration commands as below. Await first command making impact, before we run second command.



That should create a **polls\_question** table in database. In my case, my local database name is **postgres**. This table is located under **public** schema. Columns are illustrated as below.

Graphical user interface, application

Description automatically generated

Let’s populate some data into table.

Run command below that should bring us to a Django console.



Copy and paste following codes in console. Hit **Enter**. It should insert 1000 rows. Now we have enough data to test!

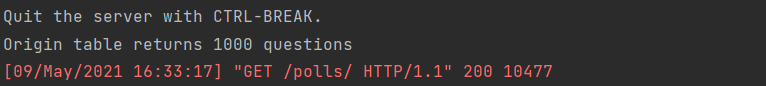


Step 2. Create lookup query on origin table

In **views.py** of **polls** app, add a lookup query by selecting all data in **polls\_question** table.



Launch project. Open up <http://127.0.0.1:8000/polls/> in browser. We should see a print as below in console. It returns 1000 rows. It’s what we would expect.



Step 3. Create a materialized view in database

In PostgreSQL console, run following SQL command to generate a materialized view that only contains data whose pub\_date is in or after year 2021.



We should see it under Materialized Views category as below.

Graphical user interface, text, application

Description automatically generated

Step 4. Build and associate model with materialized view

In **models.py** in **polls** app, create a model **Question2021** as below.



It’s important to declare db\_table explicitly so that Django can make connection between model and materialized view. Please do not run migration. Because materialized view is already created manually through SQL command. Plus, we need to make sure **managed** attribute is False. It makes sure Django does not create table by mistake when we run migrations on **polls** app in the future.

Step 5. Create lookup query on materialized model. Compare number of return

Add lookup query on **Question2021** model and print number of returns in console. Codes should look like below.



Run