Chadia Chammout

Advanced Data Management D191

ID #010306379

11/20/2022

VDM1 Performance Assessment

1. The attached data sets will be used to assess the number of rentals per category of film. This will answer the following questions: What category or categories are rented the most? Which are rented the least? This data can be used for a variety of relevant business purposes for its benefit. It can be used for marketing(this is further explained in part A5). This expands the customer base while also effectively and efficiently using the money spent on advertising. The business can also reduce costs by tailoring its inventory to best fit the results found.

**A1**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variable Name | Table to | Table from | Datatype | Description |
| Rental ID | Detailed | Rental | INT | Key that leads to rental information |
| Inventory ID | Detailed | Rental | INT | Key that leads to inventory information |
| Film ID | Detailed | Inventory | INT | Key that leads to film information |
| Category ID | Detailed | Film Category | INT | Integer that ties film category to a name. Used to match IDs with another table to get accurate category |
| Category Name | Detailed | Category | VARCHAR | Name of the film category |
| Category Name | Summary | Detailed | VARCHAR | Name of film category |
| Count | Summary | Detailed | INT | The number of rentals in the database for each category |

**A2.** The tables **rental, inventory, film category, and category** will be used to extract the necessary data for the report.

**A3.** A1’s table effectively addresses this section of the report as well. All the fields will be used to either refer to the table they are from so as to provide the means to access any relevant information needed from the ID, used to connect information from one table to another for accuracy, or used directly to draw relevant data for the business report.

**A4**. The field ‘name’ in the detailed table will be used alongside the COUNT function to transform the categories into relevant numbers to answer the business question. This is the basis of the creation of the summary table. Each one of the categories of film will be listed alongside their number of rentals to provide a visual which can be drawn upon for further use as specified in part A of this report.

**A5.** The detailed table contains a log of each rental organized by the fields mentioned previously. These fields contain the keys relevant to expanding upon the information further. The summary table logs the number of rentals per available category. Those fields directly answer the business questions posed.

The stakeholders could use the information in the summary table to then expand upon or reduce inventory. Categories that are most popular can be expanded upon using the money earned from selling the least popular categories. They could also ensure they don’t waste money buying inventory for the less popular categories. They could tailor advertisements to bring in customers by using the more popular categories. They could conversely focus their marketing efforts on the less popular categories so they could increase revenue.

The stakeholders could use the information in the detailed table to build upon the initial business questions. For example, if the ‘Action’ category is popular then efforts can be focused on marketing to the demographic that rented said category using the ‘rental id’ field. ‘Film id’ can be used to find the most popular films per category, and these can be used to tailor inventory or to draw in customers with advertisements. ‘Film id’ can also match big actors to their respective categories who can be used to promote the business.

**A6**. For inventory purposes, a refresh frequency **quarterly** is suggested to ensure the data remains relevant. For marketing, the same frequency applies. New movies, actors waxing and waning with relevance, revivals of old films, and so forth; all occur daily but capturing data quarterly can ensure that there is enough data available to draw out trends while not being too outdated.

1. CREATE TABLE detailed AS

SELECT rental\_id, inventory\_id

FROM rental;

ALTER TABLE detailed

ADD film\_id integer;

ALTER TABLE detailed

ADD category\_id integer;

ALTER TABLE detailed

ADD category\_name VARCHAR (60);

**(to be done AFTER part C)**

CREATE TABLE summary AS

SELECT category\_name, COUNT(category\_name)

FROM detailed

GROUP BY category\_name;

1. UPDATE detailed

SET film\_id = inventory.film\_id

FROM inventory

WHERE inventory.inventory\_id = detailed.inventory\_id;

UPDATE detailed

SET category\_id = film\_category.category\_id

FROM film\_category

WHERE detailed.film\_id = film\_category.film\_id;

UPDATE detailed

SET category\_name = category.name

FROM category

WHERE detailed.category\_id = category.category\_id;

CREATE TABLE summary AS

SELECT category\_name, COUNT(category\_name)

FROM detailed

GROUP BY category\_name;

**For validation of accuracy:**

Select \* from detailed

Select \* from rental

Select distinct name

from category

Select \* from summary

1. create function get\_category\_count()

returns int

language plpgsql

as

$$

declare

    category\_count integer;

begin

    select count(category\_name)

    into category\_count

    from detailed;

    return category\_count;

end;

$$;

The following code above is created for the assignment but is not used in the extraction of data for either of the tables created. Instead, the pre-built COUNT function is used.

1. CREATE OR REPLACE FUNCTION update\_number()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS $$ BEGIN

if TG\_OP=’INSERT’ then

UPDATE summary

SET count = count + 1 where category\_name = new.category\_name; end if;

RETURN NEW;

END;

$$

CREATE TRIGGER update\_count AFTER INSERT ON detailed FOR EACH ROW EXECUTE PROCEDURE update\_number();

1. create or replace procedure create\_category\_tables()

language plpgsql

as $$

begin

DROP TABLE detailed;

DROP TABLE summary;

CREATE TABLE detailed AS

SELECT rental\_id, inventory\_id

FROM rental;

ALTER TABLE detailed

ADD film\_id integer;

ALTER TABLE detailed

ADD category\_id integer;

ALTER TABLE detailed

ADD category\_name VARCHAR (60);

UPDATE detailed

SET film\_id = inventory.film\_id

FROM inventory

WHERE inventory.inventory\_id = detailed.inventory\_id;

UPDATE detailed

SET category\_id = film\_category.category\_id

FROM film\_category

WHERE detailed.film\_id = film\_category.film\_id;

UPDATE detailed

SET category\_name = category.name

FROM category

WHERE detailed.category\_id = category.category\_id;

CREATE TABLE summary AS

SELECT category\_name, COUNT(category\_name)

FROM detailed

GROUP BY category\_name;

return;

End;

$$

**F1.** Since the virtual access environment has pgAdmin 4 already installed then the use of pgAgent for scheduling is ideal. The first step would be to install pgAgent using written code, followed by writing the code to create its extension. The extension will build all the tables and functions for pgAgent. The interface will now be accessible through pgAdmin by simply choosing it in the side bar. A right click will enable the user to choose the ‘create’ option. This will pull up the screen where the dialog box with four tabs is located (Dias, 2020).

In the steps tab we will find both the general and code sections. The general section will be used to name the job, with ours being ‘category\_tables\_job’. We’ll click to enable the job and choose a local connection since our database is in-house. The steps section is where we will include the code for the procedure in part F. Finally, in the schedules tab we can scroll down to the month’s section and set it to occur every three months at the beginning of March, June, September, and December (Tiwari, 2022).

**G.** URL: <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=3bccc277-4642-4115-baad-af5901859b0b>

The environment used is a virtual access lab emulating a windows type environment. The pgAdmin application and the Microsoft Edge browser were both used while completing the lab. Upon opening pgAdmin one can access the DVD rental database by simply clicking on the words ‘server’ on the left followed by ‘DVD rental’. At the top of the application is a button with an icon that illustrates several round objects atop each other and a play button. If this is pressed after the DVD rental database is opened the interface for typing in SQL code will appear. All one must do is simply type or paste in code and press the play button above the interface.

Microsoft Edge is used to access the google document created with the code that was used. Simply going to the google website, typing in documents in the search bar, and pressing on the first link will take you to the google docs website. From there, after signing in, the document with the code will appear.

**H.**

Dias, H. (2020, February 3). *An Overview of Job Scheduling Tools for PostgreSQL.* Several Nines.

<https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/>

Tiwari, J. (2022, July 12). *How to Schedule a Job in PostgreSQL.* Free Code Camp.

<https://www.freecodecamp.org/news/how-to-schedule-a-job-in-postgresql/>