Anthony Jordan

Advanced Data Management-D191-PA

Submitted Body of Work

A. Summarize one real-world written business report that can be created from the DVD Dataset from the “Labs on Demand Assessment Environment and DVD Database” attachment.

After a detailed analysis of the provided project documentation, the real-world business question/report representative of the remainder of this submission’s body of work was decided to be:

**How important are ‘big named’ actors when it comes to the sale of DVD rentals?**

The idea is to determine whether or not the actors appearing in a particular movie will sway the rentals of those movies. This information can help stores efficiently invest in inventory, as well as other uses that will be further explored later in this submission.

1. Identify the specific fields that will be included in the detailed table and the summary table of the report.

Each row of the **detailed table** will have one row dedicated to each actor. The columns of this table will have these values:

**Actor Name** – This column will hold the combined values of “actor.first\_name” and “actor.last\_name” under the alias of “actor\_name”. This value will be stored in a VARCHAR(20) type variable.

**Movie Count** – This column will hold the COUNT() of every movie the actor has starred in, and will be stored in a SMALLINT

**Average Gross Sales per Movie** – This column will hold the average amount of money earned for the rental company by this actor. It will be computed by getting all rental sales of each movie the actor appears in, and averaging each of these values. This value will be stored in a NUMERIC(5, 2) type variable.

**Total Gross Sales** – This column will hold the total gross sales of rentals where the specified actor is present, and will be stored as a NUMERIC(7, 2)

**Total Rentals** – This column will hold data on the total number of rentals of movies that the specified actor has appeared in. This value will be stored in a SMALLINT.

The **summary table** will only have two fields derived from the **detailed table**:

**Actor name**, and **Average Gross Sales per Movie**.

2. Describe the types of data fields used for the report.

The **Actor Name** field will be stored in a variable named “actor\_name”, and it will be of the VARCHAR(30) type. This type was chosen because the actor’s names include both first and last, and they are in the form of a string.

The **Movie Count** field will be stored in a variable named “movie\_count”, and will be stored as a SMALLINT. There was an attempt to cast the type to TINYINT, however it appears as though this type is not supported in the PA lab, so the next best choice was made, SMALLINT.

The **Average Gross Sales per Movie** field will be stored in a variable named “average\_gross\_sales\_per\_movie”, and will be of the NUMERIC(5, 2) type. This leaves enough wiggle room that an actor can reach 999.99 with this variable, which seemed appropriate given the real world application, and the data derived from the database.

The **Total Gross Sales** field will be stored in a variable named “total\_gross\_sales”, and will be stored in a NUMERIC(7, 2) type. This was chosen for the same reason as the previous field. It leaves plenty of wiggle room for financial growth, while also seeming appropriate given the specified environment.

The **Total Rentals** field will also be stored in a SMALLINT. This is largely in part due to the respective range of values already stored in the database, while also reflecting the numeric data this column represents. The numbers aren’t likely to exceed SMALLINT’s max value of over 32,000.

3. Identify at least two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report.

Due to the nature of this database, the information needed for this business report will have to stretch across several different tables. This is a detailed explanation for the data used:

**The Payment Table** – This table represents the bottom of this report’s data-derivation-hierarchy. The ‘rental\_id’ field is a foreign key that will help in the “rental” table. The ‘amount’ field will be used to calculate varying monetary fields in the “detailed” and “summary tables” such as “Total Gross Sales”.

**The Rental Table** – This table is simply a means to an end. There is no data present that bears significance to the overall report. The ‘rental\_id’ variable is needed to link inventory to the purchase of rentals, and by extension, links purchases to actors and films.

**The Inventory Table** – Like the rental table, this table is more of a transition to other data. However, the inventory table is used significantly more than the rental table due to its possession of the “film\_id” field. This is the point at which the data can be used to construct a framework for this report.

The **Film\_actor Table** – Once data from this table is introduced to the report, most of the computing can be completed. This table critically stores both film and actor information that can then be used to pull data from those respective tables. “film\_id”, and “actor\_id” will both be used heavily.

4. Identify at least one field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of N to No and Y to Yes).

Several data transformations in the forms of aggregate functions were used on multiple values in the detailed table. In order to get the total gross sales for each actor, the SUM() function was used after selecting all purchases from the retail table. This is the most efficient methodology for data manipulation in this fashion. (SUM and COUNT were both used extensively)

The user-created function concatCWW() formats the first and last names of the actors and returns a full name value that can be applied to this report as ‘actor\_name’. This should be done to maintain professionalism and readability of the data.

5. Explain the different business uses of the detailed table section and the summary table section of the report.

The advantage of being in the retail industry, is that the supply of product can be fairly malleable based on the inherent needs of the consumer. One of the best ways to improve the relationship between business and customer is to make sure the customer is loving what the business puts out.

**The detailed table** performs this function by giving clear insight into the star power of these actors. Total gross income from rentals of movies starring these actors is included in the report, and this can provide insight into individual customers’ attitudes towards these actors. Other data is included for cross-referencing (for example, the number of movies each actor has starred in can heavily impact data regarding total net income).

**The summary table** exists primarily for stakeholders (i.e., people either employed higher up in the organization, or literal investors). This table contains easier to digest data, and it is formatted in a way that allows for questions to be readily and easily answered.

This data can assist with determining the volume of inventory to store at each location. If there is a big movie coming out, the organization can prepare by having a surplus of rentals available to distribute among eager customers. This also works in the reverse. The business can discover actors that may have fallen off based on these findings, and inventory can be adjusted to de-prioritize unpopular actors’ movies

6. Explain how frequently your report should be refreshed to remain relevant to stakeholders.

It would likely be beneficial to refresh this report monthly in order to continuously collect data on the trends of rising and falling stars. However, an imperative time to run these reports would be during movie seasons (i.e., early summer, fall, holiday season). This could be a good measure for inventory, as the business will have knowledge on upcoming releases. If there is a beloved actor in one of these movies, inventory can be adjusted to allow for the most access to these popular releases.

B. Provide original code for function(s) in text format that perform the transformation(s) you identified in part A4.

**CREATE OR REPLACE FUNCTION** concatCWW( str1 VARCHAR(20), str2 VARCHAR(20) )

**RETURNS** VARCHAR(30)

**LANGUAGE** plpgsql

**AS**

$$

**DECLARE** actor\_name VARCHAR(30);

**BEGIN**

**SELECT** str1 || ‘ ‘ || str2 **INTO** actor\_name;

**RETURN** actor\_name;

**END**;

$$

C. Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections.

**CREATE TABLE** detailed\_table (

actor\_name VARCHAR(30),

movie\_count SMALLINT,

total\_gross\_sales NUMERIC(7, 2),

average\_gross\_sales\_per\_movie NUMERIC(5, 2),

total\_rentals SMALLINT

);

**CREATE TABLE** summary\_table (

actor\_name VARCHAR(30),

average\_gross\_sales\_per\_movie NUMERIC(5, 2)

);

D. Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database.

**SELECT** concatCWW(actor.first\_name, actor.last\_name)

**AS** actor\_name, temptab.movie\_count, temptab.total\_gross\_sales,

temptab.average\_gross\_sales\_per\_movie, tempy.total\_rentals

**FROM**

(**SELECT CAST**(**SUM**(mathtab.total\_gross\_sales)

**AS** NUMERIC(7, 2)) **AS** total\_gross\_sales, **CAST**(

**COUNT**(film\_actor.actor\_id) **AS** SMALLINT) **AS** movie\_count,

film\_actor.actor\_id, **CAST**((**SUM**(mathtab.total\_gross\_sales) /

**COUNT**(film\_actor.actor\_id)) **AS** NUMERIC(5, 2))

**AS** average\_gross\_sales\_per\_movie

**FROM**

(**SELECT SUM**(tab2.amount) **AS** total\_gross\_sales,

tab2.film\_id

**FROM**

(**SELECT** payment.amount, tab1.film\_id

**FROM** payment

**JOIN**

(**SELECT** inventory.inventory\_id, rental.rental\_id,

inventory.film\_id

**FROM** inventory

**JOIN** rental

**ON** inventory.inventory\_id = rental.inventory\_id)

**AS** tab1

**ON** tab1.rental\_id = payment.rental\_id) **AS** tab2

**GROUP BY** tab2.film\_id

**ORDER BY** tab2.film\_id) **AS** mathtab,

film\_actor

D cont.

**WHERE** mathtab.film\_id = film\_actor.film\_id

**GROUP BY** film\_actor.actor\_id) **AS** temptab,

actor

**JOIN**

(**SELECT CAST**(**COUNT**(film\_actor.actor\_id) **AS** SMALLINT)

**AS** total\_rentals, film\_actor.actor\_id

**FROM** rental, inventory, film\_actor

**WHERE** rental.inventory\_id = inventory.inventory\_id

**AND** inventory.film\_id = film\_actor.film\_id

**GROUP BY** film\_actor.actor\_id) **AS** tempy

**ON** actor.actor\_id = tempy.actor\_id

**WHERE** temptab.actor\_id = actor.actor\_id

**ORDER BY** temptab.average\_gross\_sales\_per\_movie **DESC**;

E. Provide original SQL code in a text format that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table.

**CREATE OR REPLACE FUNCTION** update\_summary\_table()

**RETURNS TRIGGER**

**LANGUAGE** plpgsql

**AS**

$$

**BEGIN**

**DELETE FROM** summary\_table;

**INSERT INTO** summary\_table

**SELECT** actor\_name, average\_gross\_sales\_per\_movie

**FROM** detailed\_table;

**RETURN NEW;**

**END;**

E cont.

$$;

**CREATE TRIGGER** update\_to\_detailedTB

**AFTER INSERT**

**ON** detailed\_table

**FOR EACH STATEMENT**

**EXECUTE PROCEDURE** update\_summary\_table();

F. Provide an original stored procedure in a text format that can be used to refresh the data in both the detailed table and summary table. The procedure should clear the contents of the detailed table and summary table and perform the raw data extraction from part D.

**CREATE OR REPLACE PROCEDURE** refresh \_tables()

**LANGUAGE** plpgsql

**AS**

$$

**BEGIN**

**DELETE FROM** summary\_table;

**DELETE FROM** detailed\_table;

**INSERT INTO** detailed\_table

**SELECT** concatCWW(actor.first\_name, actor.last\_name)

**AS** actor\_name, temptab.movie\_count, temptab.total\_gross\_sales,

temptab.average\_gross\_sales\_per\_movie, tempy.total\_rentals

**FROM**

(**SELECT CAST**(**SUM**(mathtab.total\_gross\_sales)

**AS** NUMERIC(7, 2)) **AS** total\_gross\_sales, **CAST**(

**COUNT**(film\_actor.actor\_id) **AS** SMALLINT) **AS** movie\_count,

film\_actor.actor\_id, **CAST**((**SUM**(mathtab.total\_gross\_sales) /

**COUNT**(film\_actor.actor\_id)) **AS** NUMERIC(5, 2))

F cont.

**AS** average\_gross\_sales\_per\_movie

**FROM**

(**SELECT SUM**(tab2.amount) **AS** total\_gross\_sales,

tab2.film\_id

**FROM**

(**SELECT** payment.amount, tab1.film\_id

**FROM** payment

**JOIN**

(**SELECT** inventory.inventory\_id, rental.rental\_id,

inventory.film\_id

**FROM** inventory

**JOIN** rental

**ON** inventory.inventory\_id = rental.inventory\_id)

**AS** tab1

**ON** tab1.rental\_id = payment.rental\_id) **AS** tab2

**GROUP BY** tab2.film\_id

**ORDER BY** tab2.film\_id) **AS** mathtab,

film\_actor

**WHERE** mathtab.film\_id = film\_actor.film\_id

**GROUP BY** film\_actor.actor\_id) **AS** temptab,

actor

**JOIN**

(**SELECT CAST**(**COUNT**(film\_actor.actor\_id) **AS** SMALLINT)

**AS** total\_rentals, film\_actor.actor\_id

**FROM** rental, inventory, film\_actor

**WHERE** rental.inventory\_id = inventory.inventory\_id

**AND** inventory.film\_id = film\_actor.film\_id

**GROUP BY** film\_actor.actor\_id) **AS** tempy

**ON** actor.actor\_id = tempy.actor\_id

F cont.

**WHERE** temptab.actor\_id = actor.actor\_id

**ORDER BY** temptab.average\_gross\_sales\_per\_movie **DESC**;

1 Identify a relevant job scheduling tool that can be used to automate the stored procedure

Linux Crontab could be used to automate the ‘refresh\_tables’ procedure. A crontab file would need to be configured on a Linux distribution. As previously mentioned, the best times to run this automation would depend on the movie season, however if it were to be run monthly, let’s, say at noon, the configuration file would look like this:

0 12 1 \* \* [path-to-stored-procedure]/refresh\_tables.sql

Then I could run ‘systemctl status cron’ to check the status of this file to make sure it’s functioning properly.

Where 0 = minutes, 12 = hours [12:00 pm], 1 = day [first of the month], and the two asterisks are month and day of the week respectively. Since their values are asterisks, this means the job will run regardless of month, and regardless of what day of the week it is.

H. Acknowledge all utilized sources, including any sources of third-party code, using in-text citations and references. If no sources are used, clearly declare that no sources were used to support your submission.

No third party code was used in this submission, no sources need to be cited.