**Joshua Washington**

**Advanced Data Management – D191**

**Business Report**

**PART G – Video Link**

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=23ef53d8-0d2a-4589-9a6e-b0c7014ab4ce>

**PART A**

A. How many times is a specific movie rented at DVDRentals? DVDRentals needs to

know which movies are rented the most amongst their customers. The reason for

this is so they can choose the highest rented movies to advertise on their

storefront, website, and on TV. Displaying or advertising the most popular

movies that they have in stock, should increase consumer interest. More

consumer interest means more potential to rent more movies and a result, increased

revenue (and by technicality, advertising as well. Show the consumers that you have

the popular movies in stock and pair that with low rent fees, they'll advertise for

you by telling others about your consumer-friendly business)

**A1.**

Detailed Table -> film\_id, movie\_title, description, category\_name, release\_year, age\_rating, last\_update, rent\_count

Summary Table -> film\_id, movie\_title, release\_year, category\_name, age\_rating, rent\_count

**A2.**

Detailed Table -> INT, VARCHAR(75), VARCHAR(150), VARCHAR(35), INT, VARCHAR(10), VARCHAR(30), INT

Summary Table -> INT, VARCHAR(75), INT, VARCHAR(35), VARCHAR(10), INT, INT

**A3.**

The *film, film\_category, category, rental, and inventory* tables will be required for the detailed and summary sections of this report.

**A4.**

last\_update can be simplified to only show the month, day, and year a movie was updated. (Remove the timestamp it was updated; Hour:Minute:Second). This should be transformed because it is easier to read and the exact time may be unnecessary information. If that information was required, you can make a new query for that case specifically, but overall, that might be unnecessary information.

**A5.**

DVDRentals can use the detailed table for more precise information such as the last time a movie was updated, how many times a movie was rented, the description, genre, etc... The business can use this so they know which movies need another update, know which movies are the most popular, and have more specific information on each movie.

DVDRentals can use the summary table if they want to see is how many times a movie was rented. The business can use this so they know which movies were rented the most amongst their customers. They can promote or show ads about movies that are similar to the most rented ones.

**A6.**

My report should be refreshed every 2-3 months because movies usually release at that rate. There is an exception to this- When a movie is available that is super popular (Revenge of the Sith), the report should be updated to reflect the high demand a movie has.

**PART F1**

A job scheduling tool you could use for refreshing the data in the detailed and summary tables is pg\_cron. I choose pg\_cron because it's lightweight and easy to use. pgAgent is for more robust scheduling jobs. The data should be refreshed every 2-3 months because movies usually release at that rate. There is an exception to this- When a movie is available that is super popular (Revenge of the Sith), the data should be updated to reflect the high demand a movie has.

**--PART C**

CREATE TABLE rented\_movies (

film\_id INT PRIMARY KEY,

movie\_title VARCHAR(75),

description VARCHAR (150),

category\_name VARCHAR(35),

release\_year INT,

age\_rating VARCHAR(10),

last\_update VARCHAR(30),

rent\_count INT

);

CREATE VIEW summary\_rented\_movies AS

SELECT

film\_id AS filmID,

movie\_title AS Title,

release\_year AS releaseYear,

category\_name AS Genre,

age\_rating AS Rating,

rent\_count AS rentCount

FROM rented\_movies;

**--PART B**

CREATE OR REPLACE FUNCTION simplifyTimeStamp()

RETURNS VOID

LANGUAGE PLPGSQL

AS $$

BEGIN

UPDATE rented\_movies

SET last\_update = SUBSTRING(last\_update, 1, 10)

WHERE film\_id >= 1;

END;

$$;

**--PART E**

CREATE OR REPLACE FUNCTION add\_summary\_trigger()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

BEGIN

IF (SELECT COUNT(\*) FROM rented\_movies WHERE film\_id = NEW.film\_id) = 0 THEN

INSERT INTO summary\_rented\_movies

(film\_id, movie\_title, category\_name, release\_year, age\_rating, last\_update, rent\_count)

VALUES (NEW.film\_id, NEW.movie\_title, NEW.category\_name, NEW.release\_year, NEW.age\_rating, NEW.last\_update, NEW.rent\_count);

END IF;

RETURN NEW;

END;

$$;

DROP TRIGGER IF EXISTS add\_summary\_trigger on rented\_movies;

CREATE TRIGGER movie\_summary\_trigger

AFTER INSERT

ON "rented\_movies"

FOR EACH ROW

WHEN (pg\_trigger\_depth() < 10)

EXECUTE PROCEDURE add\_summary\_trigger();

**--PART D**

INSERT INTO rented\_movies

(film\_id, movie\_title, description, category\_name, release\_year, age\_rating, last\_update, rent\_count)

SELECT film.film\_id, film.title, film.description, category.name, film.release\_year, film.rating, film.last\_update, count(rental.rental\_id)

FROM category

INNER JOIN film\_category on category.category\_id = film\_category.category\_id

INNER JOIN film on film\_category.film\_id = film.film\_id

INNER JOIN inventory on film.film\_id=inventory.film\_id

INNER JOIN rental on inventory.inventory\_id = rental.inventory\_id

GROUP BY 1,2,3,4

ORDER BY film\_id ASC;

**--PART F**

CREATE OR REPLACE PROCEDURE refresh\_movie\_tables()

LANGUAGE PLPGSQL

AS $$

BEGIN

TRUNCATE TABLE rented\_movies;

DELETE FROM summary\_rented\_movies WHERE summary\_rented\_movies.filmID > 0;

INSERT INTO rented\_movies

(film\_id, movie\_title, description, category\_name, release\_year, age\_rating, last\_update, rent\_count)

SELECT film.film\_id, film.title, film.description, category.name, film.release\_year, film.rating, film.last\_update, count(rental.rental\_id)

FROM category

INNER JOIN film\_category on category.category\_id = film\_category.category\_id

INNER JOIN film on film\_category.film\_id = film.film\_id

INNER JOIN inventory on film.film\_id=inventory.film\_id

INNER JOIN rental on inventory.inventory\_id = rental.inventory\_id

GROUP BY 1,2,3,4

ORDER BY film\_id ASC;

END;

$$;

SELECT \* FROM rented\_movies ORDER BY film\_id ASC;