A)

Video store rental palace has requested an analysis of the most rented actors and their movies in their store. The goal is to find a popular actor and use that information to promote more movies from them and thus increase their sales.

The data used in this report include VARCHAR, INT, and DATE. To gather that data, several tables from the DVD rental database will be used. These tables include Actor, Rental, Film_Actor, Inventory, and Film. To make the summary table more readable, a transformation must be made to the actor table. First a new column called actor_name must be created, then the transformation concatenates the first and last name field into one column and inserts it into the new column that was created. This helps with readability and will ultimately help when creating the summary and detailed table.

With the detailed table, a query can be run to check which actor appears the most, thus table will include all the actor names, movies, and dates rented. A simple query can then be run to count the number of appearances an actor makes, which then gives the answer we are looking for. Using that information, the summary table can be made to include only that actor, the movies they appeared in, and the number of times that movie was rented.

For this to supply the most up to date information, the tables should be refreshed every two weeks to make sure the information doesn't become stale and so engagement can remain high. The reason to have this refresh is to keep customer engagement high and to cycle through the actors and promote different films to capture the popularity of said actor.

B)

```
CREATE OR REPLACE FUNCTION add_actor_name()

RETURNS VOID

AS

$$

BEGIN

ALTER TABLE actor ADD COLUMN actor_name VARCHAR(144);

UPDATE actor SET actor_name = CONCAT(first_name,' ',last_name);

END;

$$

LANGUAGE PLPGSQL;


--CALLING FUNCTION
```

```
SELECT add_actor_name();


--VERIFYING FUNCTION WORKS
select actor_name
FROM actor;



C)
CREATE TABLE detailed (
actor_name VARCHAR(144),
movie_title VARCHAR(144),
date_rented DATE
);
CREATE TABLE summary (
actor_name VARCHAR(144),
movie_title VARCHAR(144),
total_times_rented INT
        );


D)


INSERT INTO detailed
SELECT actor.actor_name, film.title, rental.rental_date
FROM actor
INNER JOIN film_actor as fa
ON
```

```
actor.actor_id = fa.actor_id
INNER JOIN film
ON fa.film_id = film.film_id
INNER JOIN inventory as i
ON film.film_id = i.film_id
INNER JOIN rental
ON i.inventory_id = rental.inventory_id;


select actor_name, COUNT(actor_name)
FROM detailed
GROUP BY actor_name
ORDER BY COUNT(actor_name) DESC;

INSERT INTO summary
SELECT actor_name, movie_title, COUNT(actor_name) as total_times_rented
FROM detailed
WHERE actor_name = 'Susan Davis'
GROUP BY actor_name, movie_title
ORDER BY COUNT(actor_name) DESC
        LIMIT 5;


SELECT *
FROM summary
LIMIT 5;
```

E)

```
CREATE OR REPLACE FUNCTION detailed_updates_summary()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN
DELETE FROM summary;
INSERT INTO summary
SELECT actor_name, movie_title, COUNT(actor_name) as total_times_rented
FROM detailed
WHERE actor_name = 'Susan Davis'
GROUP BY actor_name, movie_title
ORDER BY COUNT(actor_name) DESC
LIMIT 5;
RETURN NEW;
END;
$$;


CREATE TRIGGER detailed_updates_summary
AFTER INSERT
ON detailed
FOR EACH STATEMENT
EXECUTE PROCEDURE detailed_updates_summary();


--TESTING TRIGGER
INSERT INTO detailed VALUES ('Susan Davis', 'Goodfellas Salute', '2002-01-01');
```

```
select actor_name, COUNT(actor_name)
FROM detailed
GROUP BY actor_name
ORDER BY COUNT(actor_name) DESC;

SELECT *
FROM summary
        LIMIT 5;
```

F)

```
CREATE OR REPLACE PROCEDURE refresh_summary_detail()
LANGUAGE PLPGSQL
AS $$
BEGIN
DROP TABLE IF EXISTS detailed;
DROP TABLE IF EXISTS summary;

CREATE TABLE detailed AS
SELECT actor.actor_name, film.title, rental.rental_date
FROM actor
INNER JOIN film_actor as fa
ON
actor.actor_id = fa.actor_id
INNER JOIN film
ON fa.film_id = film.film_id
INNER JOIN inventory as i
ON film.film_id = i.film_id
```

```
INNER JOIN rental
ON i.inventory_id = rental.inventory_id;

CREATE TABLE summary AS
SELECT actor_name, title, COUNT(actor_name) as total_times_rented
FROM detailed
WHERE actor_name = 'Susan Davis'
GROUP BY actor_name, title
ORDER BY COUNT(actor_name) DESC
LIMIT 5;
RETURN;
END;
$$;

CALL refresh_summary_detail();

--VERIFYING PROCEDURE WORKED
select actor_name, COUNT(actor_name)
FROM detailed
GROUP BY actor_name
ORDER BY COUNT(actor_name) DESC;

SELECT *
FROM summary
LIMIT 5;
```

F1)

The scheduling software I recommend using is PAGEANT. This software works with POSTGRESQL and will allow the data to be refreshed biweekly.

H)

Sources:

- https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=5a59d26a-6cee-4a5e-8e64- adcc0127cf51&start=0
- https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cdeef825-e36a-4e9a- a157-aeed0139b592&start=0
- https://postgresqltutorial.com/
- https://www.w3schools.com/sql/default.asp
- http://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0eabc1aa-1e70-43ac-bb1e-addb013a26e8