Rebecca Wells
D326
August 26, 2024

A1.

The detailed report will focus on the monthly rentals for each film, categorized by film genre. The summary report will concentrate on the total number of rentals by film category and month. Both reports will be completed by utilizing three months' worth of data. In this case, the months will be June, July, and August.

A2.

The detailed table will contain the following fields: film_id (INT, primary and foreign key), title (VARCHAR(225)), category (VARCHAR(25)), category_id (INT, primary and foreign key), month (VARCHAR(9), primary key), rentals_count (INT).

The summary table will contain the following fields: month (VARCHAR(9), primary key), category_id (INT, primary key), category (VARCHAR(25)), total_rentals (INT).

A3.

The data for the detailed table will be sourced from the film category, category, film, inventory, and rental tables. The summary table will use data from the category table.

A4.

The timestamp rental_date field will be changed into a text field that will display only the month of the rental date. This will increase the comprehensibility of the data.

A5.

The summary table can be used to see which film categories are being rented the most. With this information, the stores can adjust their inventory to contain a larger portion of DVDs in the more frequently rented film categories.

The detailed table provides information that can be utilized to discover which DVDs are rented the most frequently in each film category. This data can then be compared with the inventory to make sure there are enough copies of the popular DVDs available for customers to rent.

A6.

Both reports should be refreshed within the first week of each month**.**

B.

```
CREATE OR REPLACE FUNCTION get_month(rental_date TIMESTAMP)

    RETURNS TEXT

    LANGUAGE plpgsql
```

```
AS

$$

DECLARE month TEXT;

BEGIN

    SELECT TO_CHAR(rental_date, 'Month') INTO month;

    RETURN month;

 END;

 $$

SELECT get_month('2005-06-30');
```

C.

```
CREATE TABLE summary_rentals (
    month VARCHAR(9),

    category_id INT,

    category VARCHAR(25),

    total_rentals INT,

    PRIMARY KEY (month, category_id),

    FOREIGN KEY (category_id) REFERENCES category (category_id)

);

SELECT * FROM summary_rentals;

CREATE TABLE detail_rentals (

    film_id INT,

    title VARCHAR(225),

    category VARCHAR(25),

    category_id INT,

    month VARCHAR(9),

    rentals_count INT,
        PRIMARY KEY (film_id, category_id, month),
        FOREIGN KEY (film_id) REFERENCES film (film_id),
        FOREIGN KEY (category_id) REFERENCES category (category_id)
);

SELECT * FROM detail_table;
```

D.

```
    INSERT INTO detail_rentals
    SELECT f.film_id AS film_id, f.title AS title, c.name AS category,  c.category_id AS
    category_id, get_month(r.rental_date) AS month, COUNT(r.inventory_id) AS rentals_count
    FROM film_category AS fc
    INNER JOIN category AS c
    ON fc.category_id = c.category_id
    INNER JOIN film AS f
    ON f.film_id = fc.film_id
    INNER JOIN inventory AS i
    ON f.film_id = i.film_id
    INNER JOIN rental AS r
    ON i.inventory_id = r.inventory_id
    WHERE r.rental_date BETWEEN '2005-06-30 00:00:00' AND '2005-08-31 23:59:59'
    GROUP BY month, f.film_id, c.category_id, category, title
    ORDER BY month, category, rentals_count DESC;


    SELECT * FROM detail_rentals;
```

E.

```
CREATE OR REPLACE FUNCTION insert_trigger_function()
RETURNS TRIGGER
 LANGUAGE plpgsql
 AS
 $$
 BEGIN
 DELETE FROM summary_rentals;
 INSERT INTO summary_rentals
 SELECT month, category_id, category, SUM(rentals_count)
 FROM detail_rentals
 GROUP BY month, category_id, category;
 RETURN NEW;
 END;
 $$

 CREATE TRIGGER updated_summary
 AFTER INSERT
 ON detail_rentals
 FOR EACH STATEMENT
 EXECUTE PROCEDURE insert_trigger_function();
```

F.

```
 --TRIGGER WILL AUTOMATICALLY CLEAR AND REFRESH SUMMARY TABLE
CREATE OR REPLACE PROCEDURE refresh_tables()
LANGUAGE plpgsql
AS
$$
BEGIN
```

```
DELETE FROM summary_rentals;

DELETE FROM detail_rentals;

INSERT INTO detail_rentals

SELECT f.film_id AS film_id, f.title AS title, c.name AS category,  c.category_id AS
category_id, get_month(r.rental_date) AS month, COUNT(r.inventory_id) AS rentals_count

FROM film_category AS fc

INNER JOIN category AS c

ON fc.category_id = c.category_id

INNER JOIN film AS f

ON f.film_id = fc.film_id

INNER JOIN inventory AS i

ON f.film_id = i.film_id

INNER JOIN rental AS r

ON i.inventory_id = r.inventory_id

WHERE r.rental_date BETWEEN '2005-06-01 00:00:00' AND '2005-08-31 23:59:59'

GROUP BY month, f.film_id, c.category_id, category, title

ORDER BY month, category, rentals_count DESC;

RETURN;

END;

$$


CALL refresh_tables();
```

F1.
I would install the pg_cron extension for PostgreSQL to automate the stored procedure.

G.

rw d326 pa (panopto.com)

H.

Dias, H. (2020, February 3). *An Overview of Job Scheduling Tools for PostgreSQL | Severalnines*. Severalnines. https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/