ADVANCED DATA MANAGEMENT

D326

CSN1 - TASK 1

Ruth Sablich

007870412

WGU

A. The human resource manager for the movie store chain 'Family Flix' wants to send physical holiday cards to each store and virtual holiday greetings to employees currently employed. She needs a detailed table of each employee's full name, staff ID, active status, email addresses, and explicit details on the store they work at. She also needs a summary table of their full names, IDs, email addresses, and store IDs, and each store's full address.

A1. The specific fields included in the **detailed** table are as follows:

- staff (
  staff_id, first_name, last_name, email, store_id, active)
- store (
  store_id, manager_staff_id, address_id)
- address (
  address_id, address, address2, district, city_id, postal_code)
- city (
  city_id, city, country_id)

A1. a. The specific fields included in the **summary** table are as follows:

- staff (
  staff_id, first_name, last_name, store_id, email)
- store (
  store_id, address_id)
- address (
  address_id, address, address2, city_id, postal_code)
- city (
  city_id, city, country_id)

A2. The staff_id, store_id, manager_staff_id, city_id, country_id, and address_id fields are integer types of data, integer meaning whole numbers with no decimals, although they are also used as small integers in foreign key tables (small integers are limited to a smaller range of numbers). The first_name, last_name, address, address2, city, postal_code, and email fields are all character types of data; character types are strings of characters (not used to store numeric values as integers) and may contain letters, special characters, and numbers. Finally, the active field is a Boolean type of data—the boolean type is a representation of the values true and false and accepts 0s and 1s.

A3. Referencing directly from the given dataset, the tables needed for population of the detailed and summary tables are: Table staff (), Table store (), Table address (), and Table city ().

A4. The 'active' field in the detailed table will require a custom transformation with a user-defined function.

It is necessary to change that field identifier's default Boolean datatype to "Yes" and "No" respectively instead of "true/false" to make it easier for readers and the human resource manager to search for, and understand that, an employee is currently employed.

A5. Both the detailed and summary tables have many different business uses.

The detailed table can be used to look up a past employee's store history, find a store's district location, or identify an employee's manager. Furthermore, the summary table can be used to find an employee's email address, their store ID, and the full address of each store. Both tables will improve business by providing efficient and accurate data on the employees and the stores without having to query multiple different tables to receive the same information. Therefore, they aid in anyone's effort to find such information by cutting the effort in half.

A6. My report should be iterated every 12 hours to ensure the validity of each employee's current employment status and the store at which they are actively working, as well as any updated personal information.

B. SQL for the transformation identified in part A4:

```
CREATE FUNCTION bool_to_string(active boolean)
        RETURNS varchar
        LANGUAGE plpgsql
AS
$$
        BEGIN
SELECT
        CASE active
         WHEN true THEN 'YES'
        WHEN false THEN 'NO'
        END employed
FROM staff;
```

```
END;
$$
;
```

C. SQL for creation of the detailed table *and* the summary table:

A.
```
CREATE TABLE detailed_table (
        staff_id INTEGER PRIMARY KEY NOT NULL,
        first_name VARCHAR(30) NOT NULL,
        last_name VARCHAR(30) NOT NULL,
        email VARCHAR(40) NOT NULL,
        active BOOLEAN NOT NULL,
        store_id INTEGER NOT NULL,
        manager_staff_id INTEGER,
        address_id INTEGER NOT NULL,
        address VARCHAR(50) NOT NULL,
        address2 VARCHAR(50),
        district VARCHAR(40),
        city_id INTEGER NOT NULL,
        postal_code VARCHAR(20) NOT NULL,
        city VARCHAR(40) NOT NULL
    );
```

B.
```
CREATE TABLE summary_table (
        staff_id INTEGER PRIMARY KEY NOT NULL,
        last_name VARCHAR(30) NOT NULL,
        first_name VARCHAR(30) NOT NULL,
        email VARCHAR(40) NOT NULL,
        store_id INTEGER NOT NULL,
        address_id INTEGER NOT NULL,
        address VARCHAR(50) NOT NULL,
        address2 VARCHAR(50),
        district VARCHAR(40),
        postal_code VARCHAR(20) NOT NULL,
        city VARCHAR(40) NOT NULL
    );
```

D. SQL for the extraction of data from existing tables into the detailed table:

```
INSERT INTO detailed_table (
        staff_id,
        first_name,
        last_name,
        email,
        active,
        store_id,
        manager_staff_id,
        address_id,
        address,
        address2,
        district,
        postal_code,
        city_id,
        city
)

SELECT
        s1.staff_id,
        s1.first_name,
        s1.last_name,
        s1.email,
        s1.active,
        s2.store_id,
        s2.manager_staff_id,
        s2.address_id,
        a.address,
        a.address2,
        a.district,
        a.postal_code,
        c1.city_id,
        c1.city
FROM staff s1
LEFT JOIN store s2
                ON s2.store_id = s1.store_id
LEFT JOIN address a
```

```
        ON a.address_id = s1.address_id
LEFT JOIN city c1
        ON c1.city_id = a.city_id
```

E. SQL for the creation of a trigger that will continually update the summary table as data is added to *and* updated in the detailed table:

```
CREATE OR REPLACE FUNCTION insert_st()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
    BEGIN
        DELETE FROM summary_table;
INSERT INTO summary_table
    SELECT staff_id, first_name, last_name, email, store_id, address_id,
address, address2, district, postal_code, city_id
    FROM detailed_table;
END;
$$
;

CREATE TRIGGER added_data_dt
    AFTER INSERT
    ON detailed_table
    FOR EACH STATEMENT
    EXECUTE PROCEDURE insert_st()
;

CREATE OR REPLACE FUNCTION updated_st()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
    BEGIN
        DELETE FROM summary_table;
INSERT INTO summary_table
```

```
        SELECT staff_id, first_name, last_name, email, store_id, address_id,
address, address2, district, postal_code, city_id
        FROM detailed_table;
END;
$$
;

CREATE TRIGGER updated_data_dt
    AFTER UPDATE
    ON detailed_table
    FOR EACH STATEMENT
    EXECUTE PROCEDURE updated_st()
;
```

F. SQL for original stored procedure that refreshes both the detailed and summary tables by clearing their contents and repopulates the detailed table using the raw data extraction from part D:

```
CREATE OR REPLACE PROCEDURE refresh_dt_st()
    LANGUAGE plpgsql
AS
$$
    BEGIN
        DROP TABLE IF EXISTS detailed_table;
        DROP TABLE IF EXISTS summary_table;

CREATE TABLE detailed_table AS
    SELECT  s1.staff_id, s1.first_name, s1.last_name, s1.email, s1.active,
s2.store_id, s2.manager_staff_id, s2.address_id, a.address, a.address2, a.district,
a.postal_code, c1.city_id, c1.city
    FROM staff s1
    LEFT JOIN store s2
            ON s2.store_id = s1.store_id
    LEFT JOIN address a
            ON a.address_id = s1.address_id
    LEFT JOIN city c1
            ON c1.city_id = a.city_id;
```

```
CREATE TABLE summary_table AS
        SELECT  s1.staff_id, s1.first_name, s1.last_name, s1.email, s2.store_id,
s2.address_id, a.address, a.address2, a.district, a.postal_code, c1.city
        FROM staff s1
        LEFT JOIN store s2
                        ON s2.store_id = s1.store_id
        LEFT JOIN address a
                        ON a.address_id = s1.address_id
        LEFT JOIN city c1
                        ON c1.city = a.city;

END;
$$
;
```

F1. To automate the stored procedure, I would suggest a job scheduling tool such as Cron—a time-based scheduler that is uncomplicated and great for regularly deployed procedures.

G. Panopto Video Link:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c8529852-ee38-4d0d-9dc6-b11d018b61e7

H. RESOURCES:
Working with Triggers and Stored Procedures WGU 2024
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cdeef825-e36a-4e9a-a157-aeed0139b592