App Store Database Project

Fourth Phase

Mobin Dariush Hamedani
96521191

Summary:

This project is a Postgres implementation of the database designed in the previous phases.

- Link to the repository: https://github.com/dalisyron/PostgreAppStore

Creators:

The queries for creating the database can be found in the *“`creators/`”* directory.

Populators:

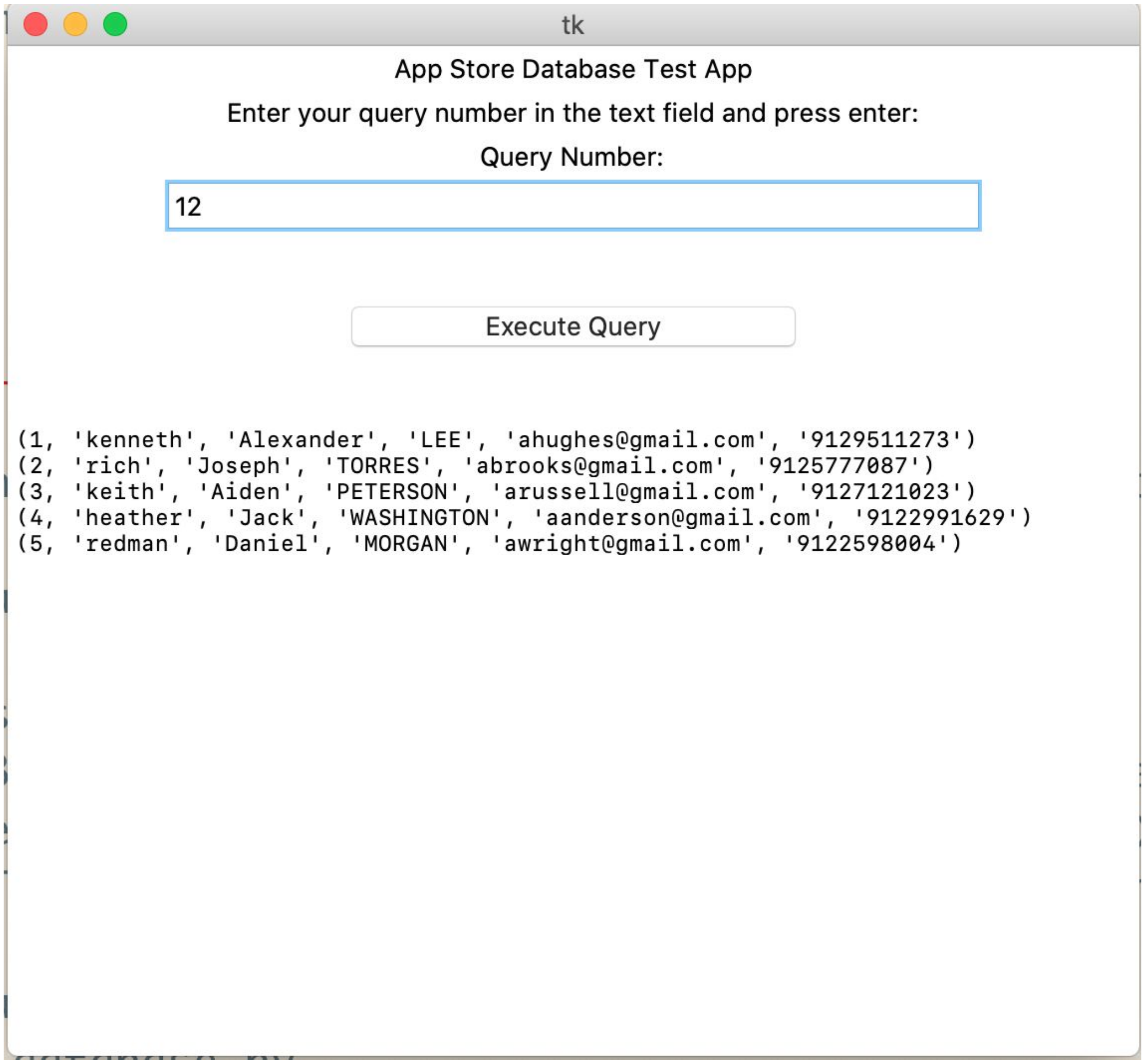
To load the sample data for testing the queries use the file in the *“`populators/`”* directory. (Generators generate the populators)

GUI App (Bonus):

It’s possible to test the database with the gui app too. The app is inside the “app/” directory and can be run with the following command:

```
python3 database.py
```

psycopg2 and tkinter dependencies are needed. To run them you have to download them based on your OS. You can use brew on macOS.


tk
App Store Database Test App
Enter your query number in the text field and press enter:
Query Number:
12
Execute Query

```
(1, 'kenneth', 'Alexander', 'LEE', 'ahughes@gmail.com', '9129511273')
(2, 'rich', 'Joseph', 'TORRES', 'abrooks@gmail.com', '9125777087')
(3, 'keith', 'Aiden', 'PETERSON', 'arussell@gmail.com', '9127121023')
(4, 'heather', 'Jack', 'WASHINGTON', 'aanderson@gmail.com', '9122991629')
(5, 'redman', 'Daniel', 'MORGAN', 'awright@gmail.com', '9122598004')
```

Queries:

1 - Name of newest apps added to the store:

```
SELECT
    apt.id, display_name, date_created
FROM
    "Apps" AS apt
    LEFT JOIN "Uploads" AS upt ON apt.id = upt.id
ORDER BY
    date_created DESC
LIMIT 20
```

2 - List of apps downloaded by a user:

```
SELECT
    display_name
FROM
    "Apps" AS apt
    LEFT JOIN "Downloads" AS dt ON apt.package_id = dt.package_id
WHERE
    user_id = 5
```

3 - All comments from a user across different apps and games:

```
SELECT
    user_id,
    package_id,
    rate,
    text
FROM ("Reviews" AS rt
    LEFT JOIN "Users" AS ut ON rt.user_id = ut.id)
WHERE
    user_id = 3
```

4 - List of apps published by a developer:

```
SELECT
    display_name
FROM
    "Apps" AS apt
    LEFT JOIN "Packages" AS pt ON apt.package_id = pt.id
WHERE
    dev_id IN (
        SELECT
            id
        FROM
            "Developers" AS dt
        WHERE
            dt.first_name = 'David'
            AND dt.last_name = 'RICHARDSON')
```

5 - Total income of a developer:

```
SELECT
    sum(price) AS TotalSum
FROM
    "Payments" AS payt
    LEFT JOIN "Purchases" AS purt ON payt.purchase_id = purt.id
WHERE
    package_id IN (
        SELECT
            package_id
        FROM
            "Packages"
        WHERE
            dev_id = 2)
```

6 - Users who have downloaded a specific app in the last week:

```
SELECT
    user_id,
    date_created
FROM
    "Downloads"
WHERE
    package_id = 335
    AND date_created BETWEEN NOW()::date - EXTRACT(DOW FROM
NOW())::integer - 7
    AND NOW()::date - EXTRACT(DOW FROM NOW())::integer
```

7 - Users who have paid for an app or game:

```
SELECT
    id
FROM
    "Users"
WHERE
    id IN (
        SELECT
            user_id
        FROM
            "Purchases")
```

8 - List of latest comments on a package:

```
SELECT
    *
FROM
    "Reviews"
WHERE
    package_id = 12
    AND date_created BETWEEN NOW()::date - EXTRACT(DOW FROM
NOW())::integer - 7
    AND NOW()::date - EXTRACT(DOW FROM NOW())::integer
```

9 - Users search for ‘Jack’:

```
SELECT
    *
FROM
    "Users"
WHERE
    first_name LIKE '%Jack%'
```

10 - List of lowest rated apps and games:

```
SELECT
    package_id
FROM
    "Reviews"
GROUP BY
    package_id
ORDER BY
    avg(rate) ASC
```

11 - Developer’s income in the last month:

```
SELECT
    sum(price) AS TotalPaid
FROM
    "Purchases" AS purt
    LEFT JOIN "Payments" AS payt ON purt.id = payt.purchase_id
WHERE
    package_id IN (
        SELECT
            package_id
        FROM
            "Packages"
        WHERE
            dev_id = 1)
    AND date_completed BETWEEN NOW()::date - EXTRACT(DOW FROM
NOW())::integer - 30
    AND NOW()::date - EXTRACT(DOW FROM NOW())::integer
```

12 - Users who have not updated ‘Telegram’ for at least a year:

```
SELECT
    *
FROM
    "Users"
WHERE
    id NOT IN ( SELECT DISTINCT
            user_id
        FROM
            "Downloads"
        WHERE
            package_id IN (
                SELECT
                    package_id
                FROM
                    "Apps"
                WHERE
                    display_name = 'Telegram')
                AND date_created BETWEEN NOW()::date - EXTRACT(DOW FROM
NOW())::integer - 365
                AND NOW()::date - EXTRACT(DOW FROM NOW())::integer)
```

13 - Search for game ‘Clash of’:

```
SELECT
    *
FROM
    "Games"
WHERE
    display_name LIKE '%Clash Of%'
```

14 - Users who have made most in app payments on “Clash Of Clans”:

```
SELECT
    user_id,
    sum(price) AS TotalPaid
FROM
    "Purchases" AS purt
    LEFT JOIN "Payments" AS payt ON (purt.id = payt.purchase_id
            AND purt.type = '1')
WHERE
    package_id IN (
        SELECT
            package_id
        FROM
            "Games"
        WHERE
            display_name = 'Clash Of Clans')
GROUP BY
    user_id
ORDER BY
    TotalPaid DESC
LIMIT 10
```

15 - Comments for a specific game:

```
SELECT
    user_id,
    rate,
    text
FROM ("Reviews" AS rt
    LEFT JOIN "Users" AS ut ON rt.user_id = ut.id) AS urt
    LEFT JOIN "Games" AS apt ON urt.package_id = apt.package_id
WHERE
    display_name = 'Clash Of Titans'
```

16 - Edit a comment:

```
UPDATE
    "Reviews"
SET
    text = 'This is the next text'
WHERE
    id = 12
```

17 - List of apps and games with most rated reviews:

```
SELECT
    package_id, avg(rate)
FROM
    "Reviews"
GROUP BY
    package_id
ORDER BY
    avg(rate) DESC
LIMIT 100
```

18 - List of developers using gmail:

```
SELECT
    *
FROM
    "Developers"
WHERE
    email LIKE '%gmail.com'
```

19 - Number of apps in each category:

```
SELECT
    name AS CategoryName,
    count(package_id) AS NumberOfApps
FROM
    "Apps" AS apt
    LEFT JOIN "Categories" AS ct ON apt.category_id = ct.id
GROUP BY
    name
```

Triggers:

Check game name does not contain the word ‘blood’:

```
DROP TRIGGER IF EXISTS check_game_name ON "Games";

DROP FUNCTION IF EXISTS fun_check_game_name;

CREATE FUNCTION fun_check_game_name ()
    RETURNS TRIGGER
    AS $$
BEGIN
    IF (NEW.display_name LIKE '%blood%') THEN
        RAISE exception 'Name cannot contain that word';
    END IF;
END
$$
LANGUAGE plpgsql;

CREATE TRIGGER check_game_name
    BEFORE INSERT ON "Games"
    FOR EACH ROW
    EXECUTE PROCEDURE fun_check_game_name ();
```

Make sure payment prices are not negative:

```
DROP TRIGGER IF EXISTS check_price_not_negative ON "Payments";

DROP FUNCTION IF EXISTS fun_check_price_not_negative;

CREATE FUNCTION fun_check_price_not_negative ()
    RETURNS TRIGGER
    AS $$
BEGIN
    IF (NEW.price <= 0) THEN
        RAISE exception 'Price must be positive';
    END IF;
END
$$
```

```
LANGUAGE plpgsql;

CREATE TRIGGER check_price_not_negative
    BEFORE INSERT OR UPDATE ON "Payments"
    FOR EACH ROW
    EXECUTE PROCEDURE fun_check_price_not_negative ();
```

Remove previously failed payments when a successful payment is made for a purchase:

```
DROP TRIGGER IF EXISTS delete_failed_status_payment ON "Payments";

DROP FUNCTION IF EXISTS fun_delete_failed_status_payment;

CREATE FUNCTION fun_delete_failed_status_payment ()
    RETURNS TRIGGER
    AS $$
BEGIN
    IF (NEW.status = 'SUCCESS') THEN
        DELETE FROM "Payments"
        WHERE purchase_id = NEW.purchase_id
            AND status = 'FAILURE';
    END IF;
END
$$
LANGUAGE plpgsql;

CREATE TRIGGER delete_failed_status_payment
    BEFORE INSERT ON "Payments"
    FOR EACH ROW
    EXECUTE PROCEDURE fun_delete_failed_status_payment ();
```