CS669 Term Project Iteration 5
“TradenXchange market”

By Venkatesh Prasad Vadaga and Gurpinder Singh

Previous Iterations Overview

The existing use-case for tradenxchange market is for working on a web marketplace where buyers can look for buying old items that sellers that want to get rid of those items can earn quick money. This includes electronic appliances, toys, old books, non-returnable amazon products. This database also has paid user and free user versions .

Idea behind it is to make sure you earn something back when you would want to get rid of the items rather than throwing them them away.

We tried to capture the scope of development of the solution for now, however we are prepared to counter some setbacks in future and change our scope accordingly.

We will be using SQL database connectivity to record everything and gather data.

Premium version of this application will allow buyers and sellers to browse or list products without any ads.

Scope of the Marketplace

Stakeholders

1. Sellers
2. Buyers
3. Admin

The Use case as per previous iterations is as follows

Use Case 1 : -User Profile Login/sign-up

- ❖ The application is installed by the user .
- ❖ The app asks them to create a unique profile
- ❖ The user enters information and their profile gets created in the database.

Use Case 2 : - Enter user details

- The user is asked to enter his/her first and last name
- Date of birth for verification if the user tries to access from new device
- Enters phone number which needs to be verified through one time password
- Password for the account is encrypted and stored in the database

Use Case 3 : - Activity Log

- User start to post or search for items which are present in the database
- The posts are recorded with the timestamp and location .
- The customer manually enters the item details ,cost ,description for the item .

Use Case 4 : - Customer purchase

- The customer (Buyer) views the post in the feed and can contact the seller through messaging options inside the application .
- The buyer can view all the categories which are available in the area which the customer wants to make the purchase .
- The buyer has an option to notify for new items which are uploaded within the zip code location.

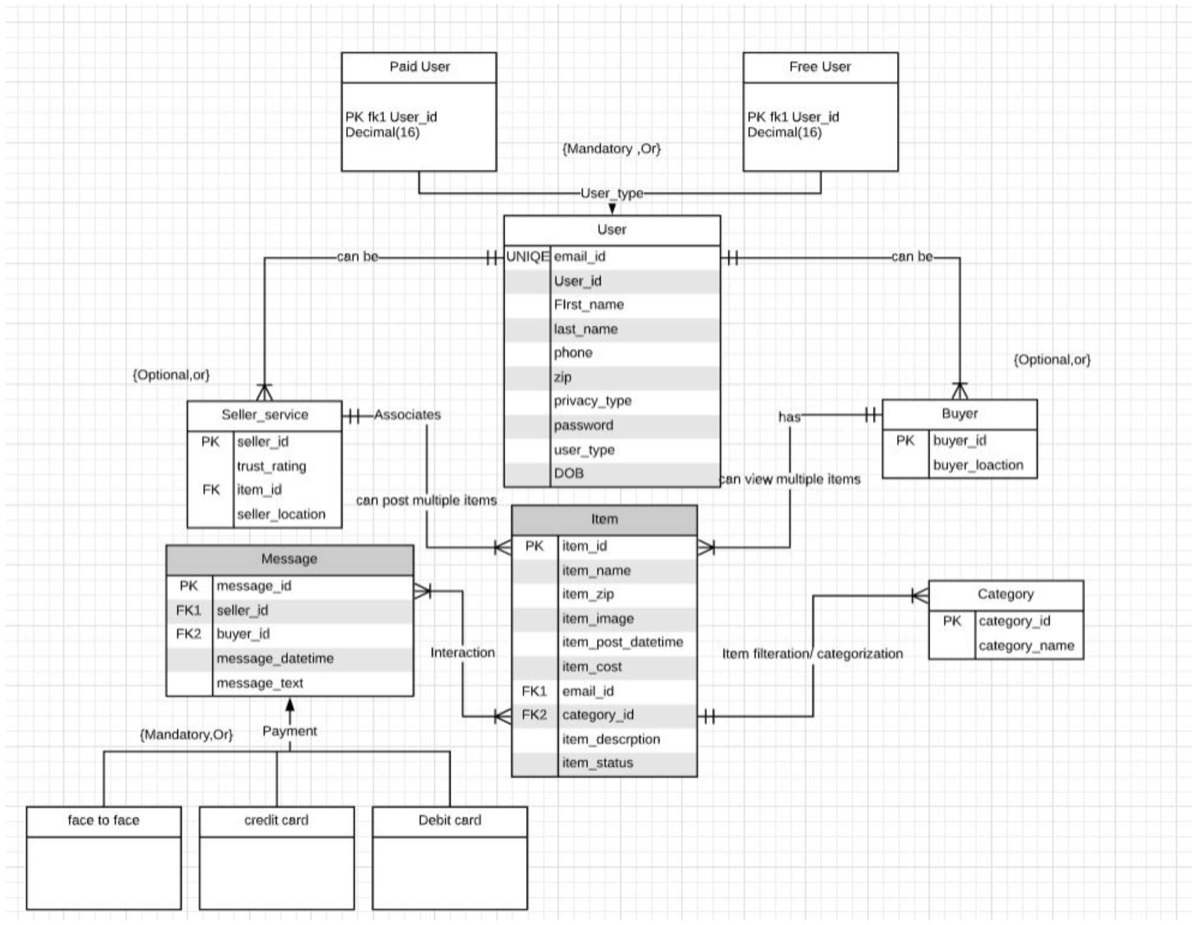
From these Use case we can derive few rules for the structural database :

1. Each user is associated with both buyer and seller ; each seller can sell as well as buy items , and buyer can sell and buy items.
2. Each item can be sold to only one person , but each item can be viewed by many buyers .
3. Each item can be bought by one person and one person can post multiple items at a time .
4. Each item shown in the feed should have an active label and as the product or item has been sold it should be labeled inactive .
5. The seller should provide the payment flexibility proposed by the user .

The business Rules:-

- Any individual can sign up as a user
- Admin credentials will be pre-set.
- Users can be either Sellers, Buyers both buyers and sellers.
- Sellers are allowed to post an item for sale.
- Sellers are allowed to set their own price.
- Sellers are allowed to edit the price and item details at any point of time
- Buyers are allowed to browse through the categories
- Buyers are allowed to browse through the sub-categories
- Buyers are allowed to browse through the list of items
- Buyers are allowed to initiate a conversation for a particular item
- Sellers are allowed to reply to the conversation.
- If the buyer offers a price lower than the asking price it sends a negotiation message to the seller.
- Seller has the option to reject the offer, accept the offer or propose a counter offer
- Sellers are allowed to mark the item as sold.
- Sellers are allowed to mark the buyer.
- Buyers can access and see the sold items but can’t offer for them
- Buyers can rate the sellers after the sellers marked them as sold to.
- Seller/Buyer can switch between being a buyer or seller at any point of time.
- Buyers can apply various filters for a particular item like location, date since available etc.

SPECIALIZATION GENERALIZATION MAPPING:-



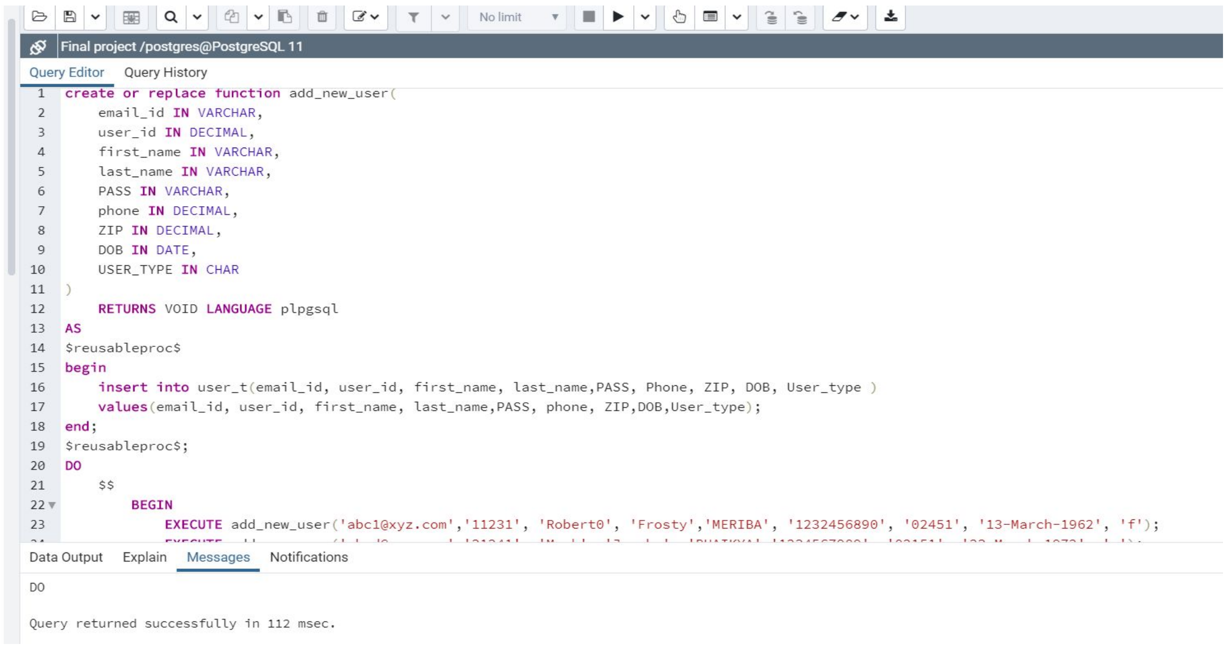
Exchange market Transactions

Use Case 1 : -User Profile Login/sign-up

- ❖ -The application is installed by the user .
- ❖ -The app asks them to create a unique profile
- ❖ -The user enters information and their profile gets created in the database.

For this use case, we have implemented a transaction to create add a new user into the database using the postgresql

- Here is the screenshot of first stored procedure definition.



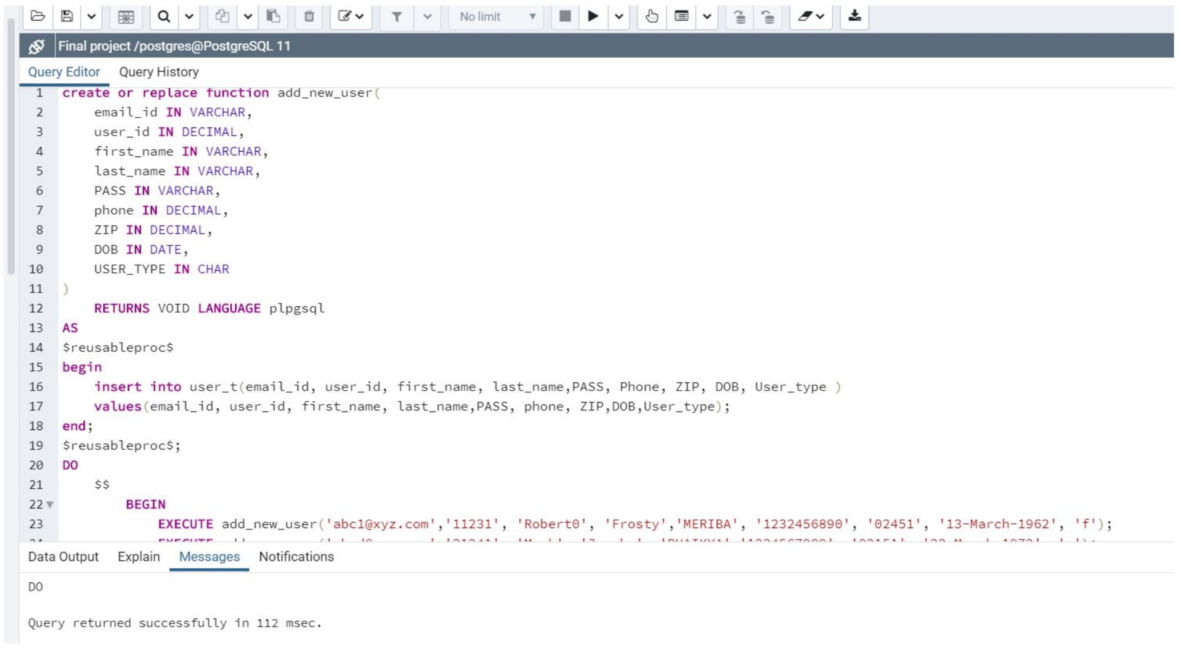
```
create or replace function add_new_user(
    email_id IN VARCHAR,
    user_id IN DECIMAL,
    first_name IN VARCHAR,
    last_name IN VARCHAR,
    PASS IN VARCHAR,
    phone IN DECIMAL,
    ZIP IN DECIMAL,
    DOB IN DATE,
    USER_TYPE IN CHAR
)
    RETURNS VOID LANGUAGE plpgsql
AS
$reusableproc$
begin
    insert into user_t(email_id, user_id, first_name, last_name,PASS, Phone, ZIP, DOB, User_type )
    values(email_id, user_id, first_name, last_name,PASS, phone, ZIP,DOB,User_type);
end;
$reusableproc$;
DO
    $$
        BEGIN
            EXECUTE add_new_user('abc1@xyz.com','11231', 'Robert0', 'Frosty','MERIBA', '1232456890', '02451', '13-March-1962', 'f');
```

```
DO

Query returned successfully in 112 msec.
```

We name the stored procedure “Add_new_User”, and give it parameters that correspond to the UserAccount table. Since CreatedDate is always the current date, we do not need a parameter for that, but instead use it to call the procedure function to add every user in Postgresql server.

Inside the stored procedure, there are two insert rows named robert frost and margarita filagi which need to be inserted into a “User” Table.

Here is a screenshot of the stored procedure execution



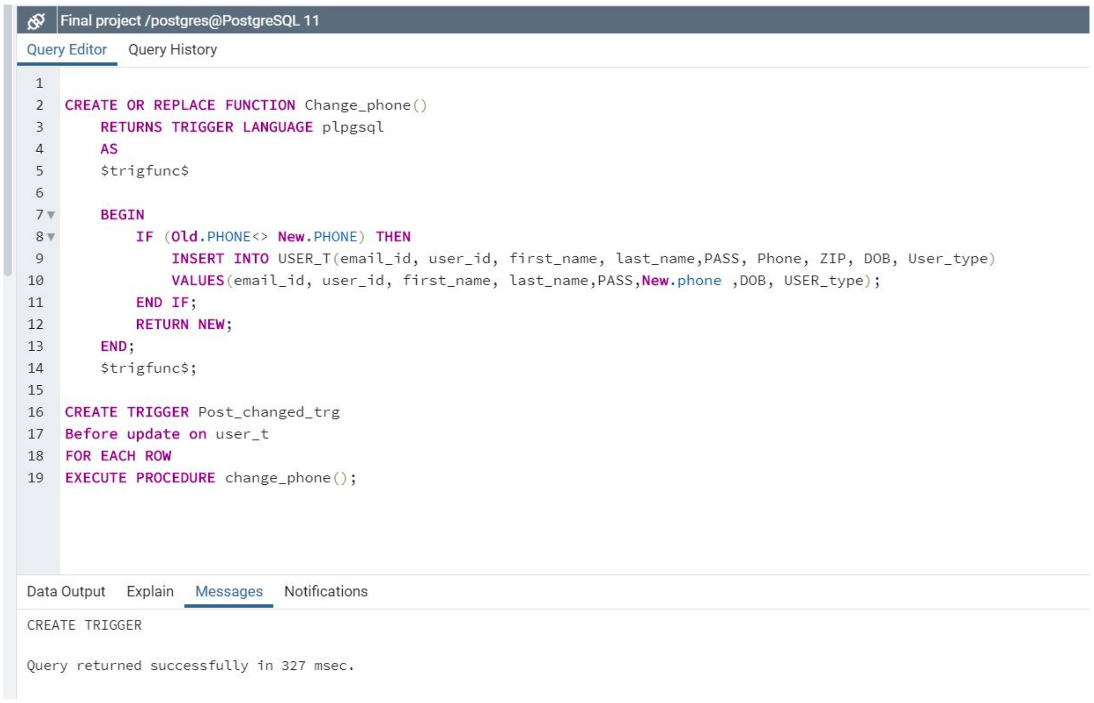
```
create or replace function add_new_user(
    email_id IN VARCHAR,
    user_id IN DECIMAL,
    first_name IN VARCHAR,
    last_name IN VARCHAR,
    PASS IN VARCHAR,
    phone IN DECIMAL,
    ZIP IN DECIMAL,
    DOB IN DATE,
    USER_TYPE IN CHAR
)
    RETURNS VOID LANGUAGE plpgsql
AS
$reusableproc$
begin
    insert into user_t(email_id, user_id, first_name, last_name,PASS, Phone, ZIP, DOB, User_type )
    values(email_id, user_id, first_name, last_name,PASS, phone, ZIP,DOB,User_type);
end;
$reusableproc$;
DO
    $$
        BEGIN
            EXECUTE add_new_user('abc1@xyz.com','11231', 'Robert0', 'Frosty','MERIBA', '1232456890', '02451', '13-March-1962', 'f');
```

```
DO

Query returned successfully in 112 msec.
```

Add a fictional person named Roberto Frosty with other fictional but realistic information. I nested the stored procedure call between transaction control statements to ensure the transaction is committed.

Inside the stored procedure with the trigger to edit the phone number , there are two insert statements to insert into the two respective tables.

Here is a screenshot of stored trigger function execution .



Final project /postgres@PostgreSQL 11

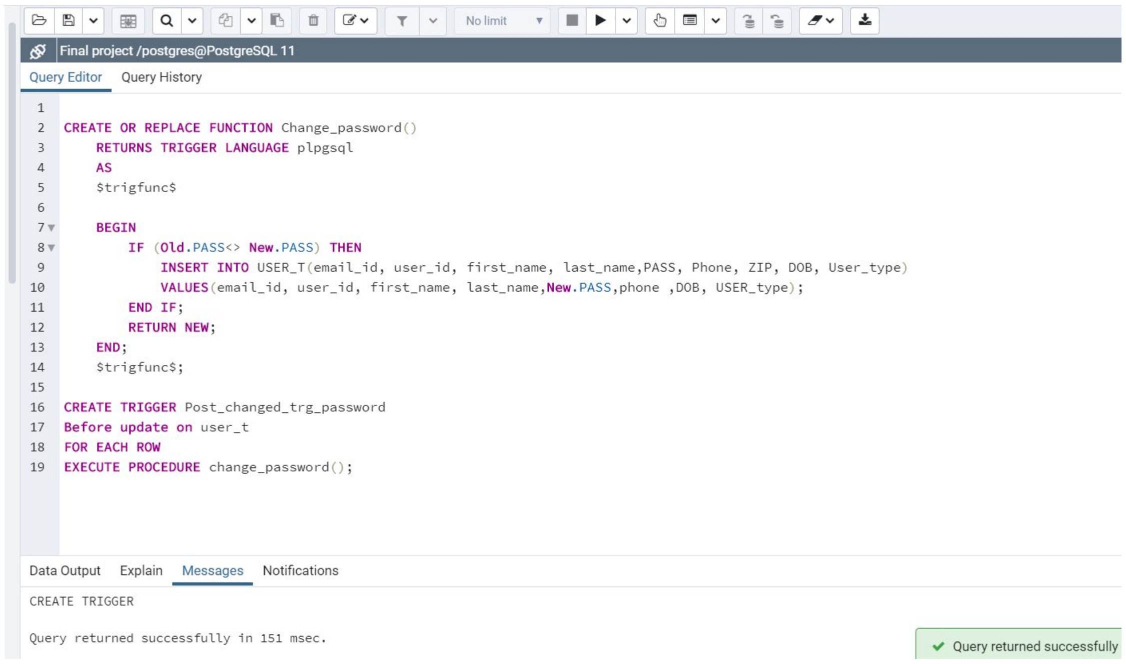
Query Editor   Query History

```sql

CREATE OR REPLACE FUNCTION Change_phone()
    RETURNS TRIGGER LANGUAGE plpgsql
    AS
    $trigfunc$

    BEGIN
        IF (Old.PHONE<> New.PHONE) THEN
            INSERT INTO USER_T(email_id, user_id, first_name, last_name,PASS, Phone, ZIP, DOB, User_type)
            VALUES(email_id, user_id, first_name, last_name,PASS,New.phone ,DOB, USER_type);
        END IF;
        RETURN NEW;
    END;
    $trigfunc$;

CREATE TRIGGER Post_changed_trg
Before update on user_t
FOR EACH ROW
EXECUTE PROCEDURE change_phone();
```

Data Output   Explain   Messages   Notifications

```
CREATE TRIGGER

Query returned successfully in 327 msec.
```

Another trigger to stored trigger function execution to change the password



Final project /postgres@PostgreSQL 11

Query Editor   Query History

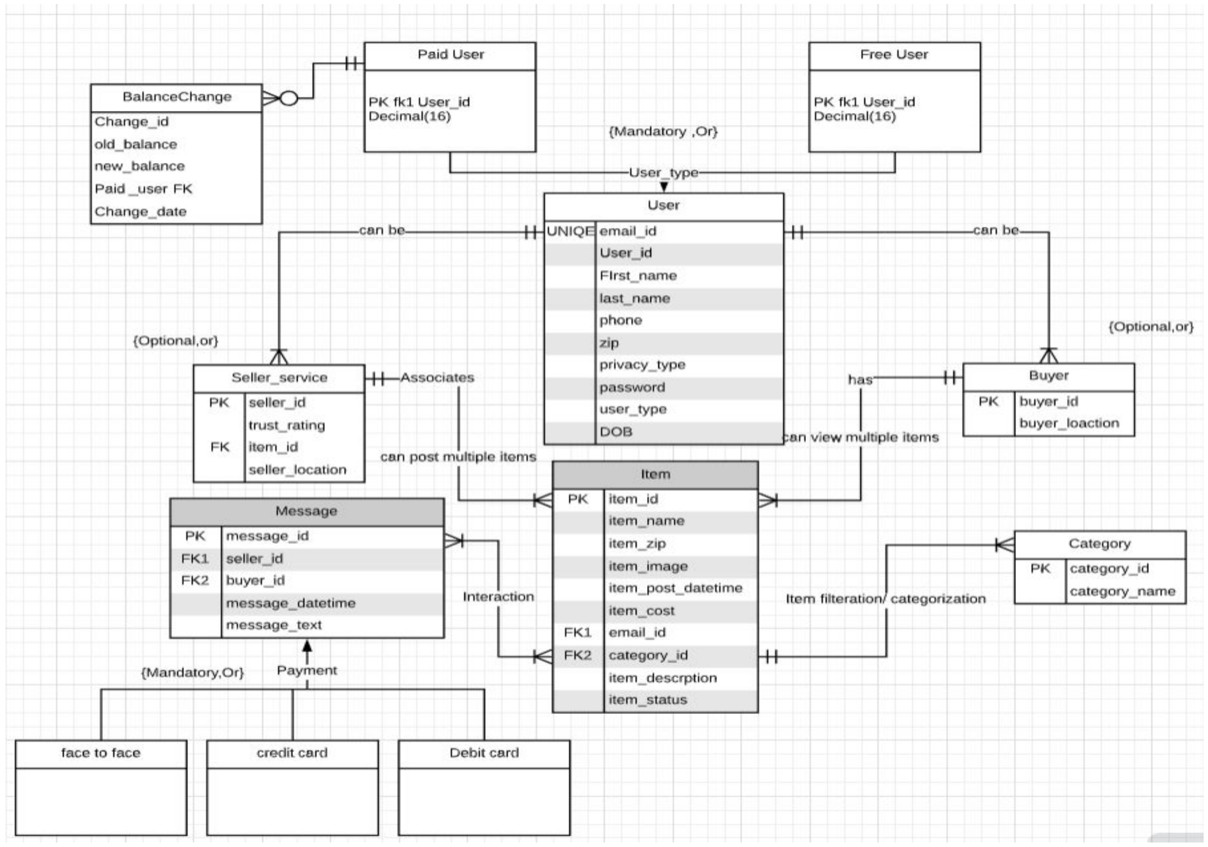
```sql

CREATE OR REPLACE FUNCTION Change_password()
    RETURNS TRIGGER LANGUAGE plpgsql
    AS
    $trigfunc$

    BEGIN
        IF (Old.PASS<> New.PASS) THEN
            INSERT INTO USER_T(email_id, user_id, first_name, last_name,PASS, Phone, ZIP, DOB, User_type)
            VALUES(email_id, user_id, first_name, last_name,New.PASS,phone ,DOB, USER_type);
        END IF;
        RETURN NEW;
    END;
    $trigfunc$;

CREATE TRIGGER Post_changed_trg_password
Before update on user_t
FOR EACH ROW
EXECUTE PROCEDURE change_password();
```

Data Output   Explain   Messages   Notifications

```
CREATE TRIGGER

Query returned successfully in 151 msec.
```

Exchange Market History

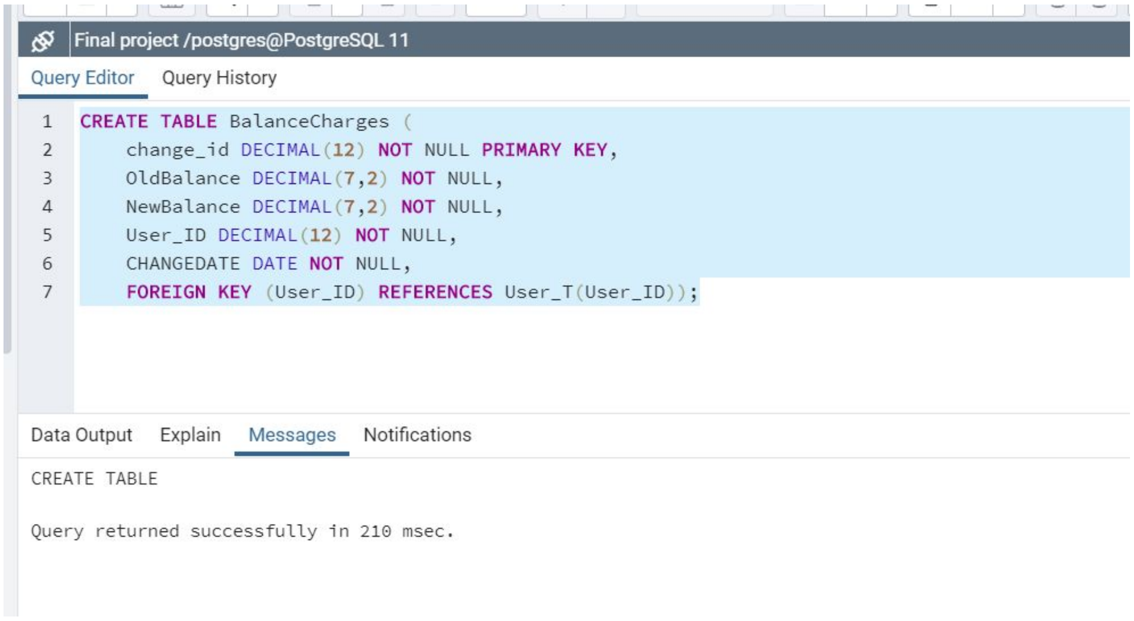
In reviewing the DBMS physical ERD, one piece of data that would benefit from a historical record is a person’s balance in the PaidAccount table. Such a history would help me calculate statistics about account balances that are accurate over time. First, the new structural database rule is ; balances change for individuals trackable , and different plans would have different balance charges for the paid/ premium user .



The BalanceChange entity is present and linked to PaidUser. Below are the attributes I added and why.

ATTRIBUTE	DESCRIPTION
ChangeID	This is the primary key of the history table. It is a DECIMAL(12) to allow for many values.
Old Balance	This is the balance of the account before the change. The datatype mirrors the Balance datatype in the PaidAccount table.
New Balance	This is the balance of the account after the change. The datatype mirrors the Balance datatype in the PaidAccount table.
PaidAccountID	This is a foreign key to the PaidAccount table, a reference to the account that had the change in balance.
ChangeDate	This is the date the balance change occurred, with a DATE datatype.

Here is a screenshot of the table creation, which has all of the same attributes and datatypes as indicated in the DBMS physical ERD.

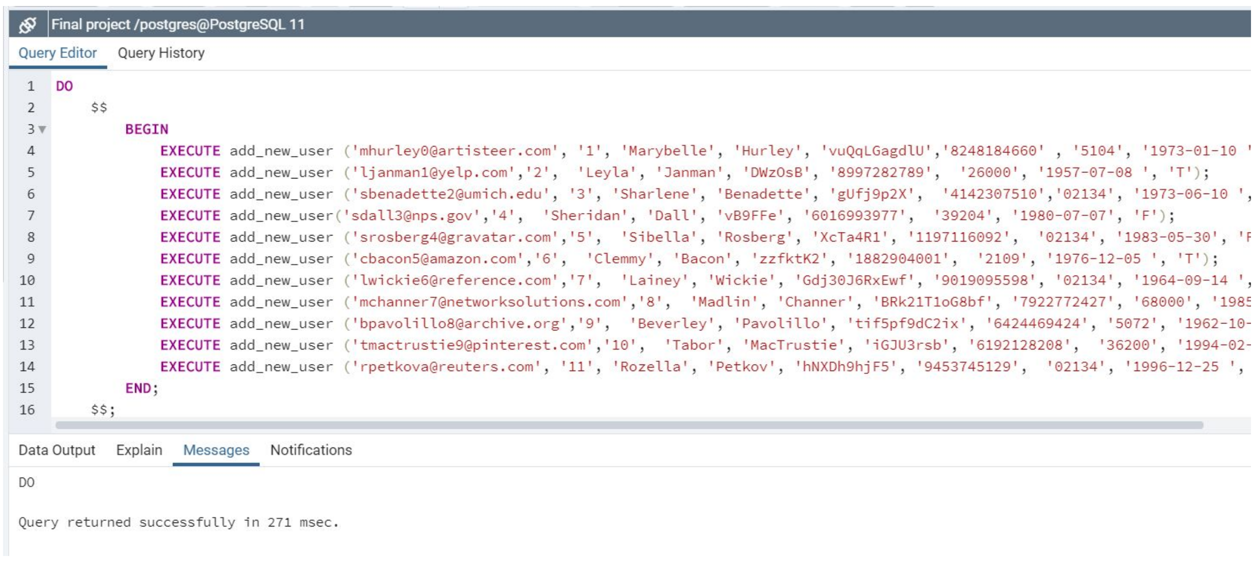


```
CREATE TABLE BalanceCharges (
    change_id DECIMAL(12) NOT NULL PRIMARY KEY,
    OldBalance DECIMAL(7,2) NOT NULL,
    NewBalance DECIMAL(7,2) NOT NULL,
    User_ID DECIMAL(12) NOT NULL,
    CHANGEDATE DATE NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES User_T(User_ID));
```

```
CREATE TABLE

Query returned successfully in 210 msec.
```

Question and Query

Now we need to insert the data inside the fields which needs to be a important role in understanding the and drawing the data out into a table and derive understanding from the portions



Final project /postgres@PostgreSQL 11

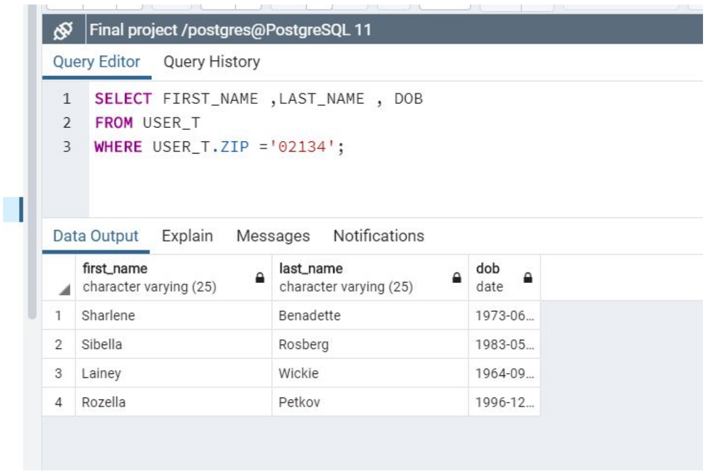
Query Editor Query History

```
DO
    $$
        BEGIN
            EXECUTE add_new_user ('mhurley0@artisteer.com', '1', 'Marybelle', 'Hurley', 'vuQqLGagdlU','8248184660' , '5104', '1973-01-10 '
            EXECUTE add_new_user ('ljanman1@yelp.com','2',  'Leyla', 'Janman', 'DWzOsB', '8997282789',  '26000', '1957-07-08 ', 'T');
            EXECUTE add_new_user ('sbenadette2@umich.edu', '3', 'Sharlene', 'Benadette', 'gUfj9p2X',  '4142307510','02134', '1973-06-10 ',
            EXECUTE add_new_user('sdall3@nps.gov','4',  'Sheridan', 'Dall', 'vB9FFe', '6016993977',  '39204', '1980-07-07', 'F');
            EXECUTE add_new_user ('srosberg4@gravatar.com','5',  'Sibella', 'Rosberg', 'XcTa4R1', '1197116092',  '02134', '1983-05-30', 'F
            EXECUTE add_new_user ('cbacon5@amazon.com','6',  'Clemmy', 'Bacon', 'zzfktK2', '1882904001',  '2109', '1976-12-05 ', 'T');
            EXECUTE add_new_user ('lwickie6@reference.com','7',  'Lainey', 'Wickie', 'Gdj30J6RxEwf', '9019095598', '02134', '1964-09-14 ',
            EXECUTE add_new_user ('mchanner7@networksolutions.com','8',  'Madlin', 'Channer', 'BRk21T1oG8bf', '7922772427', '68000', '1985
            EXECUTE add_new_user ('bpavolillo8@archive.org','9',  'Beverley', 'Pavolillo', 'tif5pf9dC2ix', '6424469424', '5072', '1962-10-
            EXECUTE add_new_user ('tmactrustie9@pinterest.com','10',  'Tabor', 'MacTrustie', 'iGJU3rsb', '6192128208',  '36200', '1994-02-
            EXECUTE add_new_user ('rpetkova@reuters.com', '11', 'Rozella', 'Petkov', 'hNXDh9hjF5', '9453745129',  '02134', '1996-12-25 ',
        END;
    $$;
```

Data Output Explain Messages Notifications

```
DO

Query returned successfully in 271 msec.
```

The following query helps in understanding our user base better, as you can see in the screenshot, once we know we are getting a lot of traction from one area, it's highly relevant information for our marketing campaign if we wish to spend money on google ads.



Final project /postgres@PostgreSQL 11

Query Editor Query History

```
SELECT FIRST_NAME ,LAST_NAME , DOB
FROM USER_T
WHERE USER_T.ZIP ='02134';
```

Data Output Explain Messages Notifications

	first_name character varying (25)	last_name character varying (25)	dob date
1	Sharlene	Benadette	1973-06...
2	Sibella	Rosberg	1983-05...
3	Lainey	Wickie	1964-09...
4	Rozella	Petkov	1996-12...

Summary and Reflection

Database for tradenxchange marketplace where customers can purchase game controllers, electronics items at their best price ; the users can post about the item which they want to put out in the market which in return would be viewed by multiple users across close regions for compelling prices.When user purchases are made, the transaction information is printed on each customer receipt. The central database will track all user information, store inventory items (products) as well as purchase transaction data. The database stores all the aforementioned information, in real time, keeping all product information current as transactions occur. The structural database rules and ERD for the database design include the following entities : User , Buyer, Seller, Item ,message , Category paid user , free user ,balance change . Additionally , the relationships between the entities are included. The DBMS physical ERD contains the same entities and relationships, uses the best practices of various keys, and contains the meaningful attributes needed by the database to support the system.

The SQL Server as well as postgresql scripts contain all table creations that follow the specification from the DBMS physical ERD. Several indexes have been created to speed up access to the Exchange market database and are also available in an SQL index script. Additionally, stored procedures have been created and executed transactionally to fill in some data into the database. Some questions useful to The Exchange market have been asked and completed with SQL queries. A history of customer phone number changes has been created as well as a query which tracks changes to phone changes for customers, based on their customer IDs.

Now that the database Iterations are complete, Planning to incorporate this database information to the frontend website. Hopefully, combining the two would create a comprehensive project .