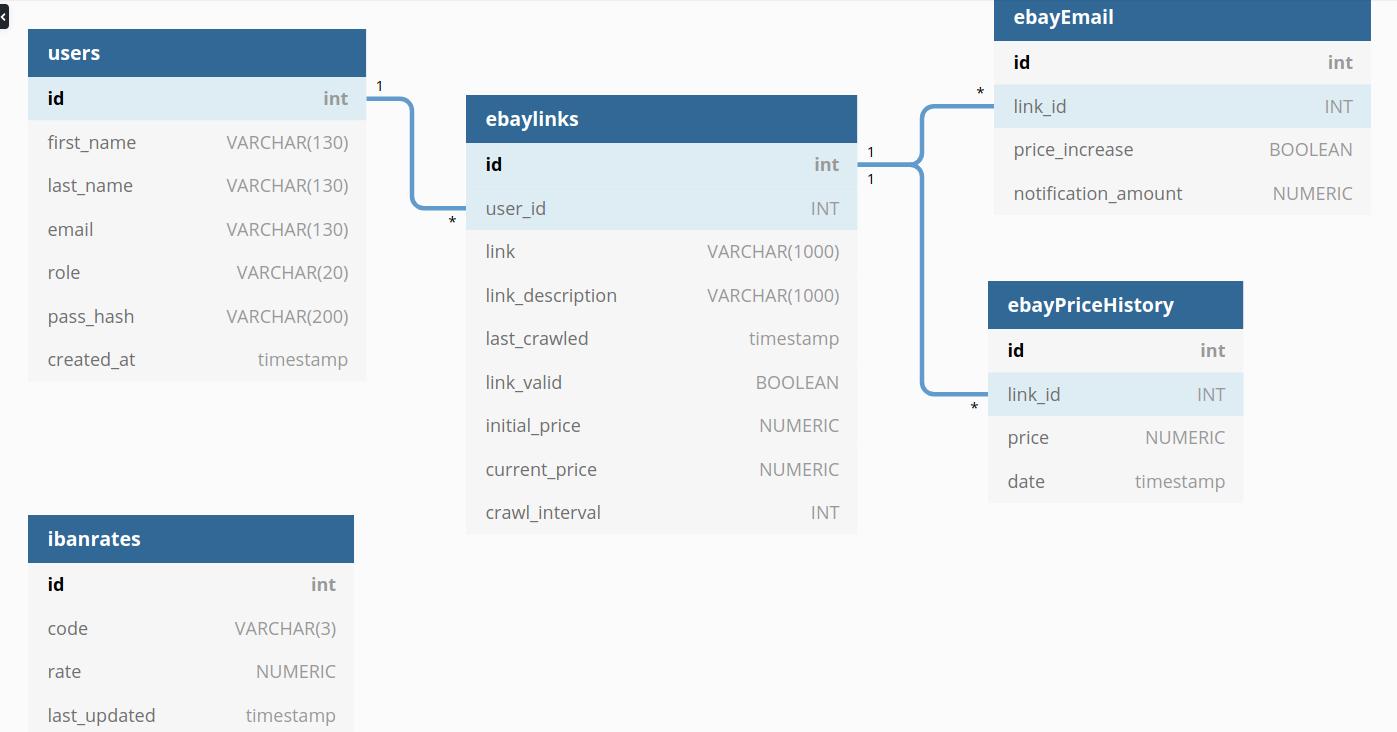
**Database Documentation**

PostgreSQL Management: <https://customer.elephantsql.com/login>

A good GUI tool to use to manage/view database: <https://www.pgadmin.org/> although I find a command line client to be much simpler to work with.

This document will cover the database schema as well as the stored functions written in PL/pgSQL and their expected inputs and outputs. Expect it to be updated as the project evolves.

Functions are provided in text file format (.sql) in the database functions subfolder here on SharePoint.



# Tables

## demo\_table

Table name: demo\_table

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | SERIAL | PRIMARY KEY |
| name | VARCHAR(130) | NOT NULL |
| role | VARCHAR(20) | NOT NULL |

Demo table contains the team members names and roles and is proof of concept to show how the API will handle data. See API document for further information.

## users

Table name: users

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | SERIAL | PRIMARY KEY |
| first\_name | VARCHAR(130) | NOT NULL |
| last\_name | VARCHAR(130) | NOT NULL |
| email | VARCHAR(130) | NOT NULL, UNIQUE |
| role | VARCHAR(20) | NOT NULL |
| pass\_hash | VARCHAR(200) | NOT NULL |

See related stored functions: addUser, getUser

## Ibanrates

Table name: ibanrates

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | SERIAL | PRIMARY KEY |
| code | VARCHAR(3) | NOT NULL UNIQUE |
| rate | NUMERIC | NOT NULL |
| last\_updated | TIMESTAMP | NOT NULL |

This table contains currencies scraped from the Iban currency code website. Code represents the currency code, for example AUD. Rate is the corresponding rate.

Currently it only stores rates for AUD, NZD, USD, GBP, CNY.

See related stored functions: ibanUpdate, convertIban

## IbanHistory

Table name: ibanHistory

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | SERIAL | PRIMARY KEY |
| code | VARCHAR(3) | NOT NULL UNIQUE |
| rate | NUMERIC | NOT NULL |
| last\_updated | TIMESTAMP | NOT NULL |

This table contains data from every time the Iban rates table is scraped. It is intended for use by the frontend for graphing currency rate history.

## Ebaylinks

Table name: ebaylinks

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | SERIAL | PRIMARY KEY |
| user\_id | INT |  |
| link | VARCHAR(1000) | NOT NULL |
| link\_description | VARCHAR(1000) | NOT NULL |
| last\_crawled | TIMESTAMP | NOT NULL |
| Link\_valid | BOOLEAN | NOT NULL |
| Initial\_price | NUMERIC | NOT NULL |
| current\_price | NUMERIC |  |
| Crawl\_interval | INT | NOT NULL |

This table contains the ebay link that the user wants crawled alongside data to be used for crawl interval and initial price as a reference for notifiying of price changes later on if they occur. User\_id is a foreign key that links the ebay link to the user ID of the user from the users table.

See related stored functions: addEbayItem, updateEbayPrice, getEbayUpdateList

## EbayLinkHistory

Table name: ebayLinkHistory

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | SERIAL | PRIMARY KEY |
| Link\_id | INT |  |
| Time\_crawled | TIMESTAMP | NOT NULL |
| price | NUMERIC | NOT NULL |

This table contains a links entire price history, a new entry is added every time the link is crawled at the preset interval. This data can be used for historical purposes such as charting.

Link\_id is a foreign key linking to ebaylinks table id field.

## EbayEMailTable

Table name: ebayEmail

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | SERIAL | PRIMARY KEY |
| Link\_id | INT |  |
| Price\_increase | BOOLEAN | NOT NULL |
| Notification\_amount | NUMERIC | NOT NULL |

This table is used to keep track of notificion settings a user can set.

# Stored Functions

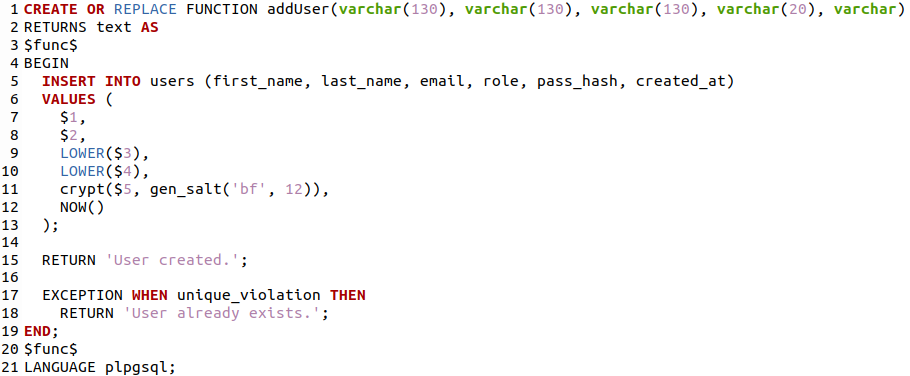
## addUser

This function adds a new user to the users table in database. It sets role to user, and salts and hashes password before saving the hash and ensures the users email address doesn’t already exist on system.

Correct Usage: SELECT \* FROM addUser(‘Jane’, ‘Smith’, [‘test@test.com’](mailto:‘test@test.com’), ‘user’, ‘pasword123’);

Returns “User Created” for success.

Returns “User already exists” if a record with the same email address already exists in the users table.



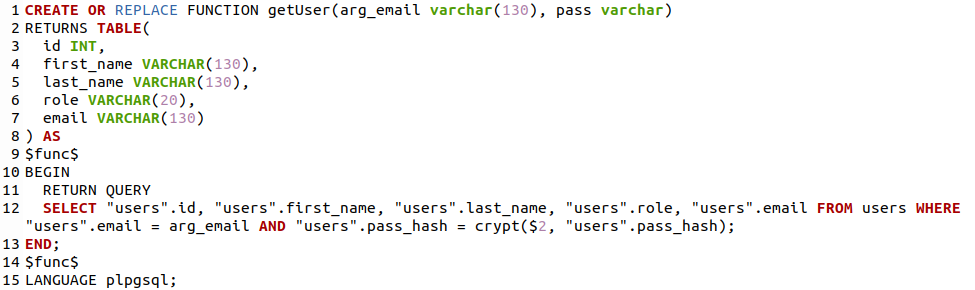
## getUser

Takes an email address and password. Checks that password hashes match and returns result for user.

Correct usage: SELECT \* FROM [getUser(‘test@test.com’](mailto:getUser(‘test@test.com’), ‘pasword123’);

Returns table with users id, first\_name, last\_name, role and email if password hashes match.

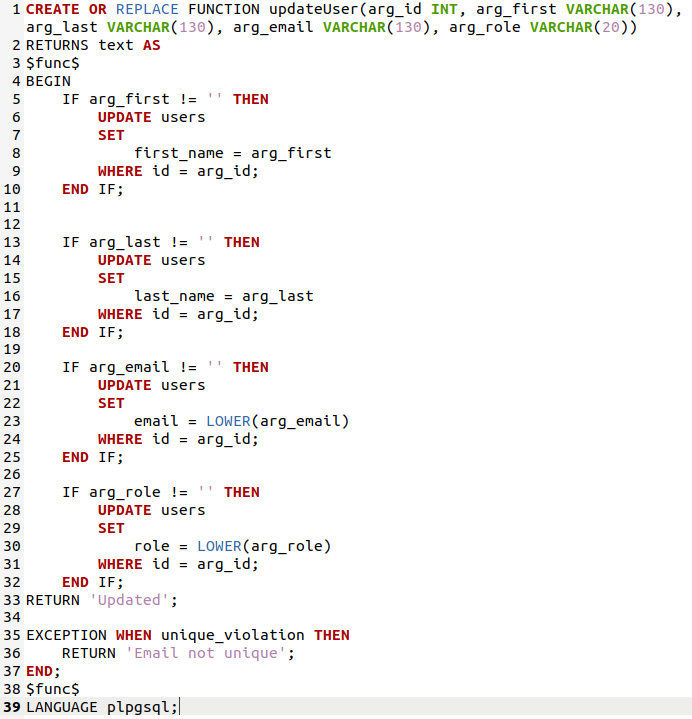
Returns 0 rows if no match. For security reasons we don’t want to specify if the password was wrong or simply the user doesn’t exist (email doesn’t match).



## UpdateUser

Takes in users ID, first name, last name, email and role and updates their record in the users table. If any of the fields are empty strings (aside from ID) then the empty field is ignored.

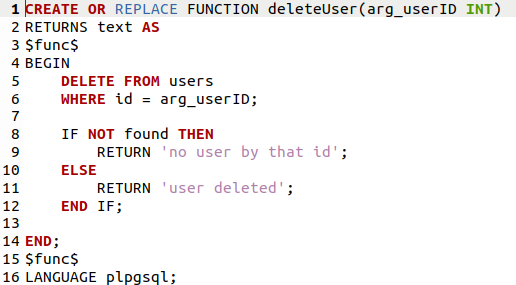
Correct usage: SELECT \* FROM update[User(‘mary’,’‘smith’](mailto:getUser(‘test@test.com’), [‘ms@outlook.com’](mailto:‘ms@outlook.com’), ‘user’);



## DeleteUser

Deletes user from the users table by their user ID.

Correct usage: SELECT \* FROM deleteUser(‘22’);



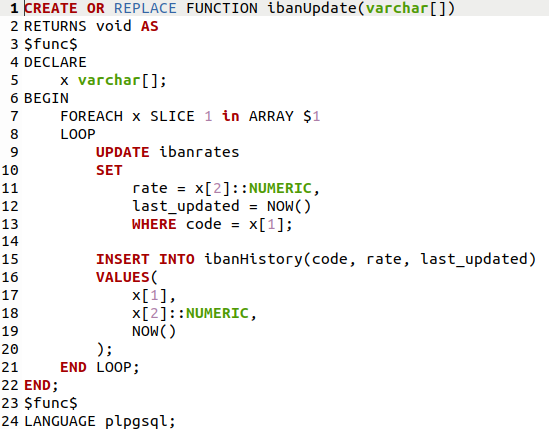
## IbanUpdate

Takes in an array of currency codes an associated rates and updates the record that matches the code in the ibanrates table.

Correct usage: SELECT \* FROM ibanUpdate(ARRAY[[‘AUD’,‘1.234’],[‘USD’,’1.2345’]]);

Returns for success: “Currency rate updated”

and for failure: ”Currency code not found in database”



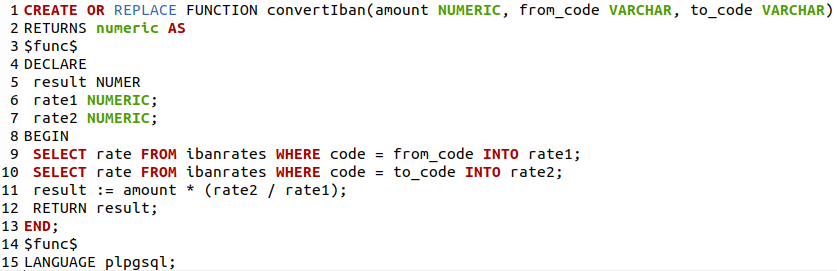
## ConvertIban

Takes in a numerical amount (money), from rate and to rate (currency code) and returns the converted amount using data stored in the ibanrates table.

Correct usage: SELECT \* FROM convertIban(‘115.24’, ‘AUD’, ‘USD’);

This is saying convert $115.24 Australian dollars (AUD) to American dollars (USD).

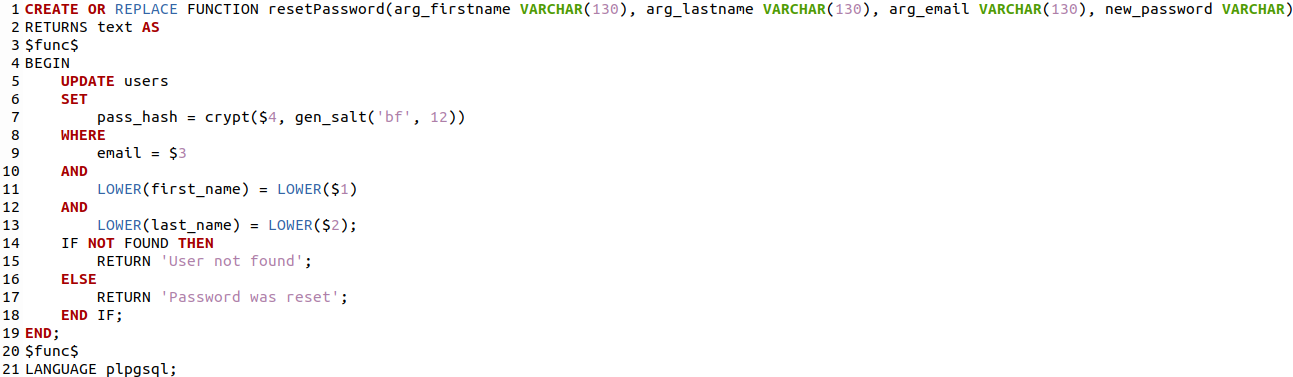
Returns: the converted amount on success. Returns nothing on failure.



## ResetPassword

Takes in a first and last name, email and new password value. If the email is found in the users table with corresponding name fields, the function will hash the new password update it for the associated email.

Note: first and last name are case insensitive to make the process easier.

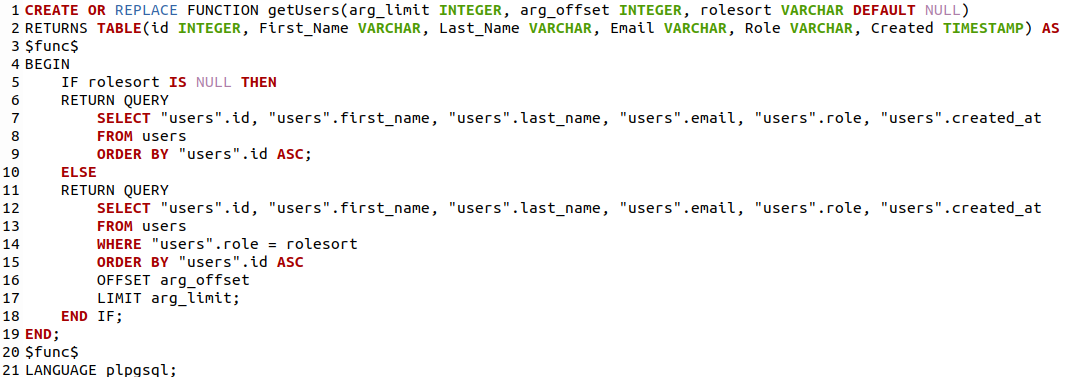


Correct usage: SELECT \* FROM [resetPassword(‘johny’, ‘droptables’, ‘test@test.com’](mailto:resetPassword(‘test@test.com’), ‘password1234’);

Returns: ‘Password was reset’ on success and ‘User not found’ if no matching email found in the users table.

## GetUsers

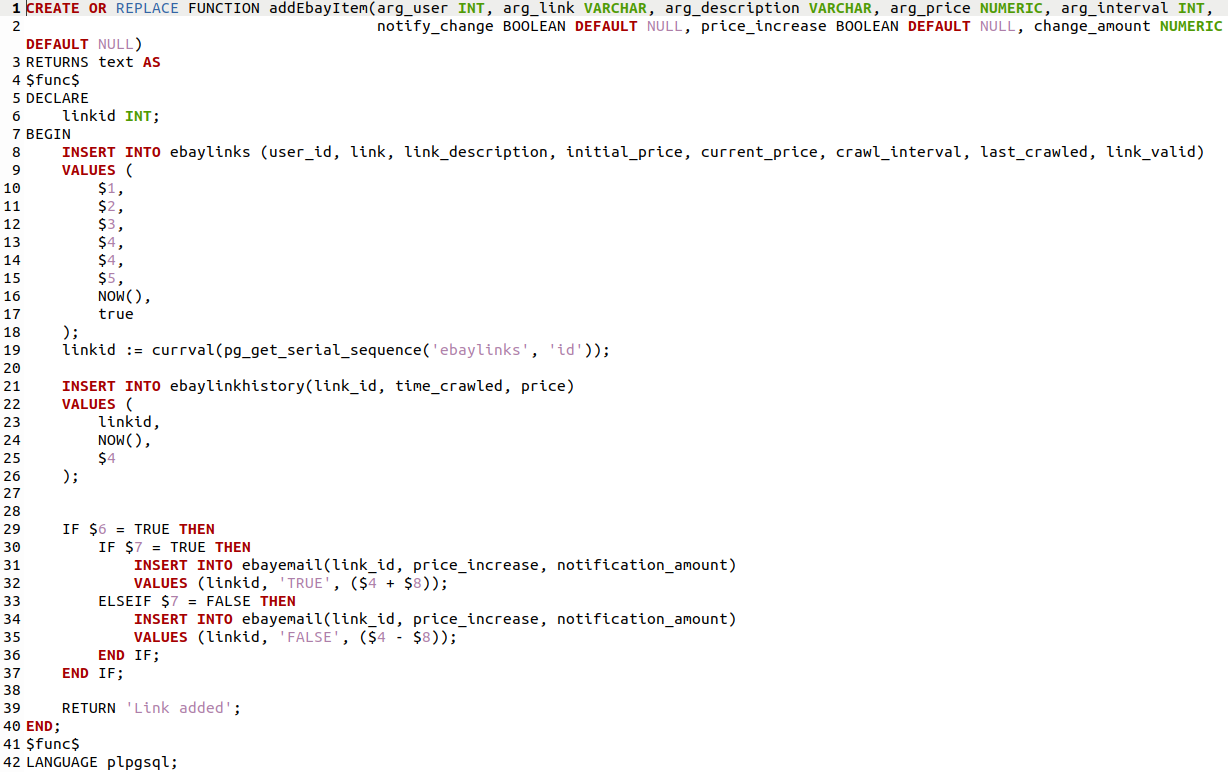
This function takes a limit and offset with an optional rolesort field and applies a SQL query on it depending on input values and returns the result.



## AddEbayItem

This function takes in the eBay items details (link, price, description, update interval and user id) and stores it in the ebaylinks table. It also sets the link to valid, and add the initial price to the ebayLinkHistory table.

It also has optional fields to take price change parameters that are used for notify a user regarding price increase/decrease.

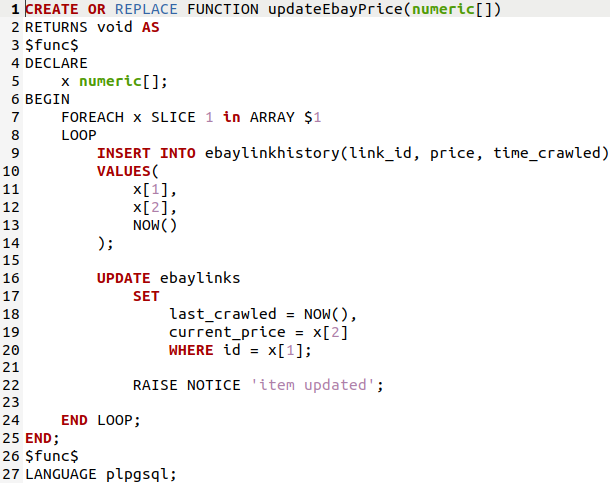


## UpdateEbayPrice

This function is used by the backend to update the price associated with a saved eBay link. This price data is from the ebay crawler. It also updates the last\_crawled in ebaylinks (which is show in frontend) and time\_crawled in ebayLinkHistory which is used for scheduling.

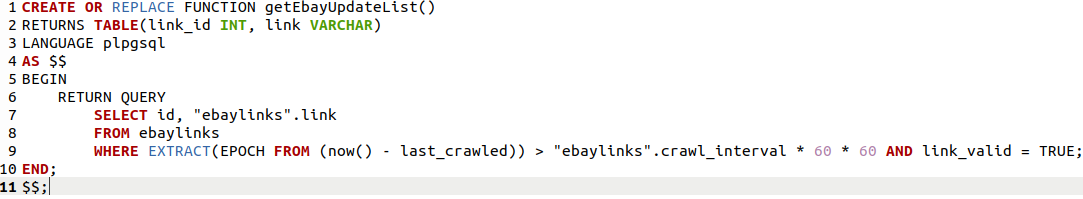
It takes in an array of values so multiple items can be updated with one query.

Example input, link ID and associated price: [2, 159.24],[3, 88.00]



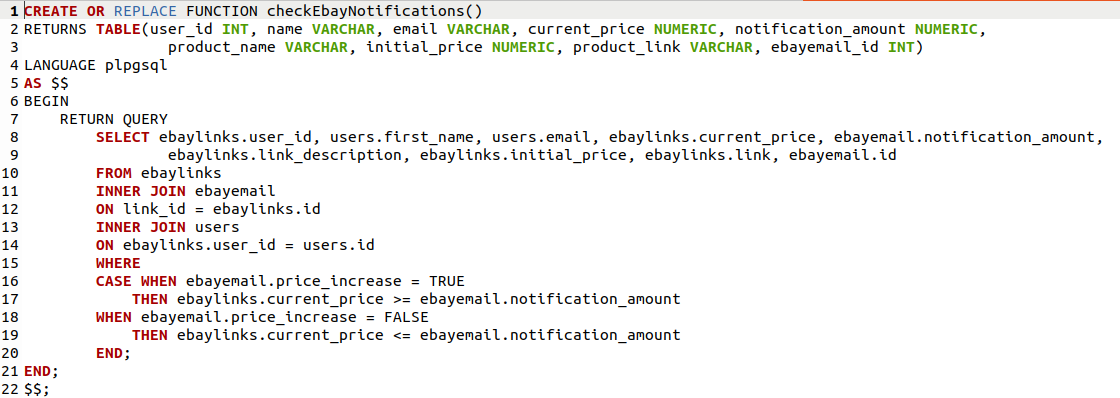
## GetEbayUpdateList

This function will go through all links saved for all users and determine if they are due to be updated. This is done by comparing the amount time since the last update against the set update interval. If it is due to be updated the function returns a list of links to be run through the crawler which are then fed back to the updateEbayPrice method via the backend(API).



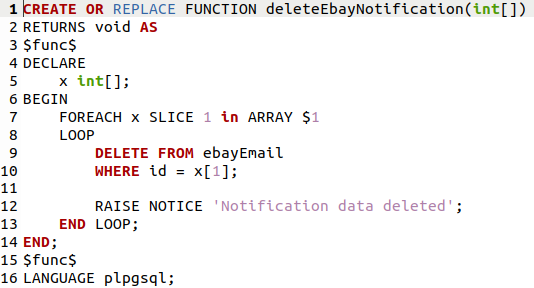
## CheckEbayNotifcations

This function checks to see if any items current prices match associated settings for any stored price alert settings.



## DeleteEbayNotifaction

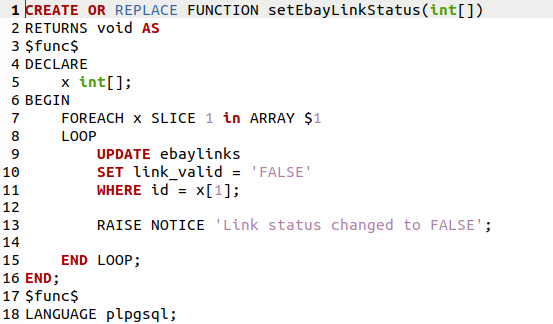
This function deletes an ebay notification triggers from the ebayemail table. This is designed to be run after a notification email has been triggered and sent. It takes in an array of ebayemail ID’s.



## SetEbayLinkStatus

When a stored link expires, for example the product is removed by seller or auction ends, The link status is set to false and this prevents it from being processed in the crawler.

This method takes an array of Link IDs and sets the link status for each to false.



## DeleteEbayLink

This function deletes a link via its ID, it also takes in a user ID, both must match the link for it to be deleted.

