Ho Chi Minh University of Technology
Department of Computer Science & Engineering

DATABASE SYSTEM

Lab Assignment Report

Convenient Store Database

Instructor: Tran Minh Quang

Student:

Huynh Truong Tu	1852847
Nguyen Tan Tai	1852725
Lam Trinh Long	1811044

Ho Chi Minh City, December 2020

Contents

1 Requirement description

1.1 Requirement overview

The database will be used to help a convenience store company managing and keeping data on their multiple branches of stores.

The database will keep data on the staff members at every branches, along with the inventory of items being sold at that branch.

The database can be used by staff members, branch managers and higher ups in the company.

The customer accounts can be created and modified by the staff members, the customers would not directly enter their account into the system.

A public user may use the system to search for items they need and write reviews for the shop they bought items at.

1.2 Database analysis

The database should contain the following data:

1. The data on *branches*: Each branch has its unique team of staff members, so each staff should not be able to work at 2 branches at the same time. The staff is identified by their ID numbers, and the system will also keep track of their name, phone number, email address, their work hours, and their current position in the promotional system (i.e. manager, cashier, ...).

- Identified by unique ID numbers
- Keep information on what are the branches' names
- The location where each branch is at
- Contain a record of several evaluations, each customer can only leave one review, which they can update if they change their mind.

2. The data on *staff members*:

Each branch has its unique team of staff members, so each staff should not be able to work at 2 branches at the same time. The staff is identified by their ID numbers, and the system will also keep track of their name, phone number, email address, their work hours, and their current position in the promotional system (i.e. manager, cashier, . . .).

Each entry of data on Staff will have the following attribute fields:

- Unique ID number
- Name
- Phone number
- Email address

- Work hour
- The ranking at their working branch

3. The data on *merchandise*:

Items sold at the stores have their data stored in batches imported in at the same time and have the attributes of their product name, manufacturing date, expiration date, quantity, their retailing price, all identified by their unique ID code. The system should also support optional limited-time discounts.

Each entry of data on Item will have the following attribute fields:

- An unique ID for each item entry
- The item's name
- The quantity of that item type
- When the item shipment arrived at the store
- Their manufacturing day
- The expiration date
- Sale rate(percentage of the original pricing)
- Duration of the sale

4. The data on *customers account*:

Customers buying at the store can optionally create an account for extra services and discounts. The customer accounts details their unique customer ID, their name, phone number, address, email, date of birth, gender, and their account ranking used for various benefits at different tiers depending on their spending.

So each entry of data on Customers Account will have the following attribute fields:

- Unique ID number for each account
- The customer's name
- Their phone number
- Their address
- Their email address
- Their date of birth
- Their gender
- And a stored data on the account rank

5. The data on *purchase history*: A purchases-record system, identified by unique ID numbers, which contains where the customer made that purchase, how they paid for it (cash or credit), what list of items they purchased, who was the current cashier at that time, when that purchase was made and the tax rate for that purchase. And if it was done using a registered customer account, keep track of that also.

So the table on purchase history will have these attributes:

- Unique ID number
- Which branch it was made at
- Payment method
- A list of purchased item
- Cashier staff on duty making the payment
- Timestamp
- Tax rate

1.3 Functionalities analysis

The staff members at the stores will be able to enter, view and modify data of the items available for sale at the store. Staff members will be able to create receipt for customers containing which items they bought, at which branch the purchase were made, and which staff was on duty for that purchase.

The staff members at the stores will also be able to enter data on each item when they get shipped to branches of the store, enter all purchases into a list both for data archiving and creating a receipt for the customer and create a new user account on the behalf of the customer when they request to do so.

Managers will be able to enter/modify information on the data of branches and the data staff members and will have all of the staff's permission in extension.

The customers will be able to look up what items are available for to buy, where the items are at. They will also be able to write and modify evaluations of the branches they shopped at.

With these users in mind, we imagine the database would have access points at the store entrance for public users, mostly for guest sign-ins and people with customer account sign-ins. Few more access points at the cash registers for cashier staff members sign-ins. An access point in the staff room for managers and staff members and an access point at the storage room, for staffs to log which items are shipped to the store.

2 Conceptual design

2.1 LUCIDCHART - A nice tool for conceptual design

Lucidchart is used for visual communication and cross-platform collaboration. It helps create professional flowcharts, process maps, UML models, org charts, and ER diagrams using templates or import features. Works on Mac, PC, and Linux and integrated with user apps.

LucidChart requires a premium purchase to be fully functional while our design is developed with free accounts. Thus there is a limitation on how many entries we can put into the graph (60 to be precise). However, the minimum advantages this tool can give us is that we can build beautiful diagrams easily, edit them with colleagues simultaneously and save all of your changes instantly which is well-suited for team working.

When it comes to database model design, we need to import the DBMS template and Database Entity Relationship shapes. Then after the job has been done, we can export the database into other platforms of database design or different types of file formats. If you have already created a diagram using one of the following platforms, you can easily import it into Lucidchart: Visio (VDX, VSDX, VSD) Gliffy (GXML, GLIFFY) OmniGraffle (GRAFFLE, GRAFFLE.ZIP).

Below are some screenshots of LucidCharts work environment.

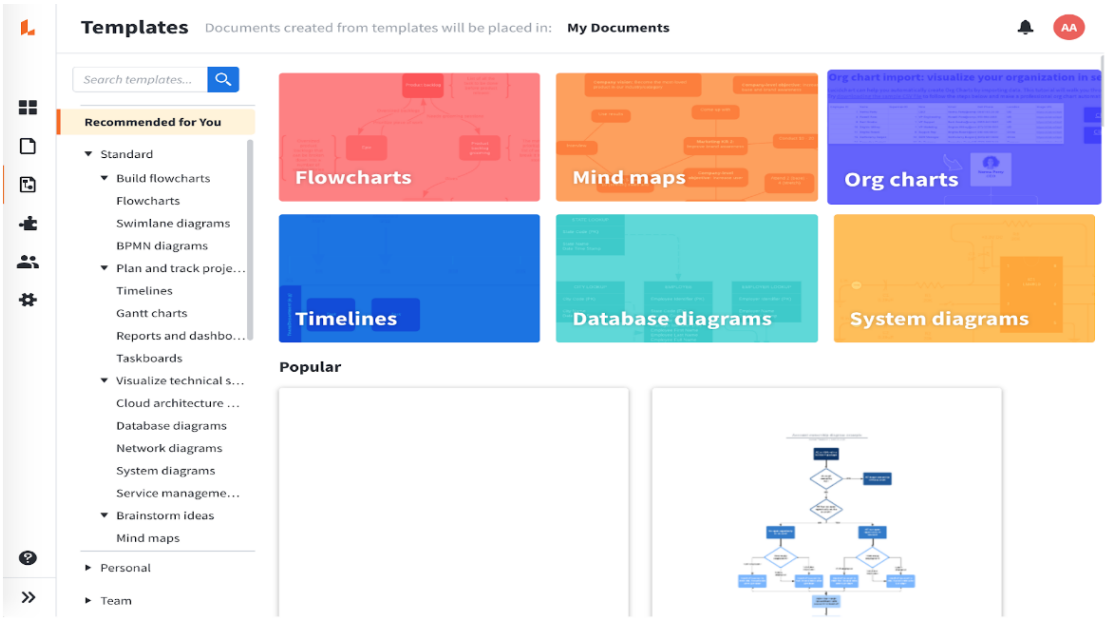


Figure 1: Getting templates for the new model.

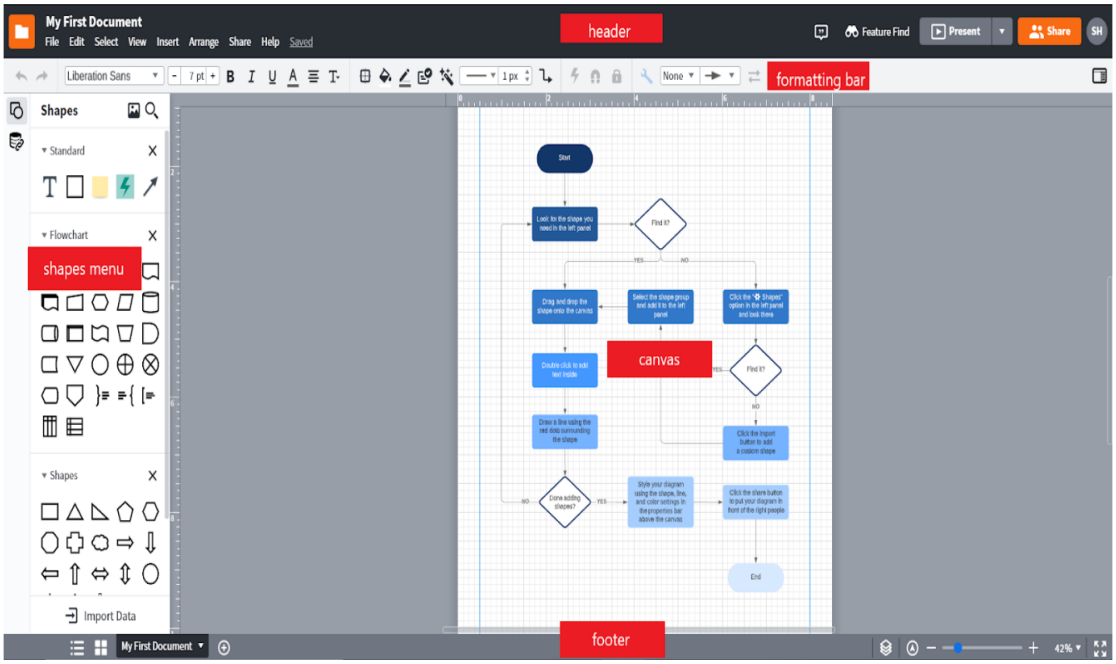


Figure 2: Lucidchart workspace - where all of the diagramming processes take place.

2.2 Conceptual entity-relationship model

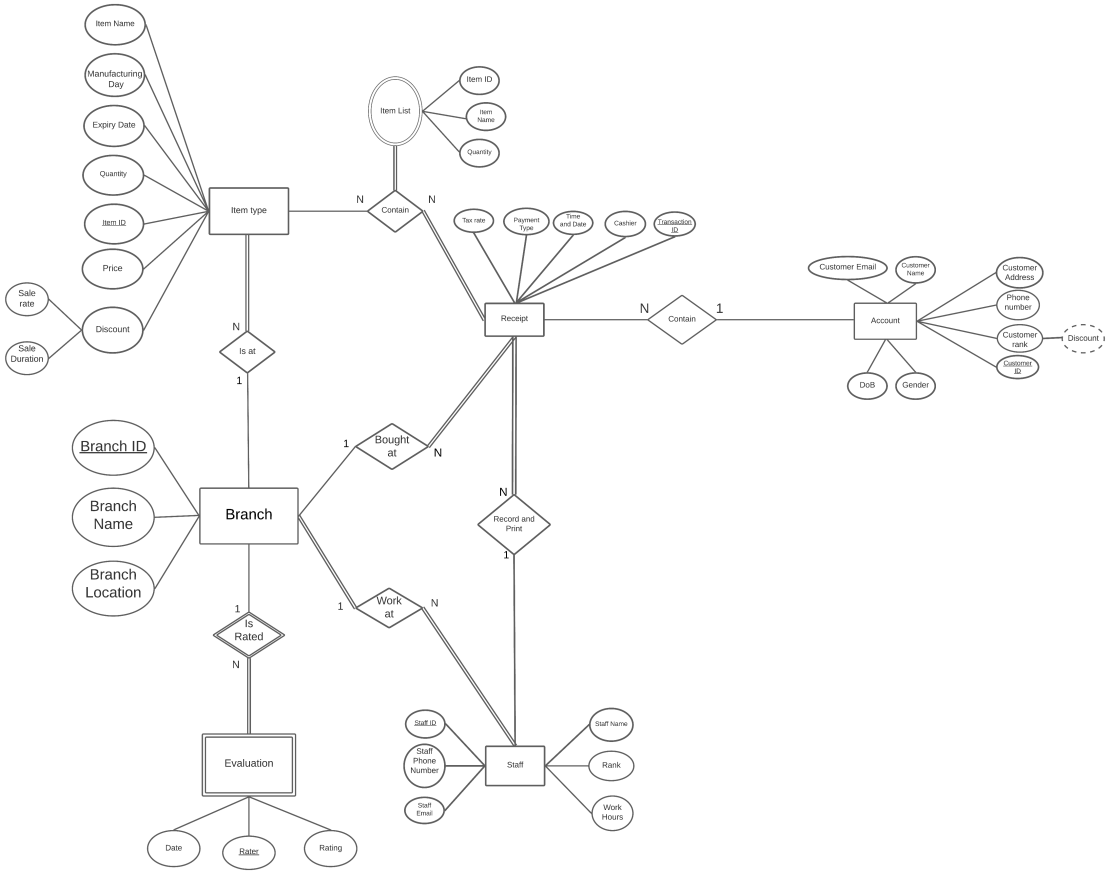


Figure 3: **The ER model for our database design**

3 Logical database design

In this step, we mostly just convert the ER diagram into a Relational Data Model and plan out ahead which is the datatype appropriate to each data field.

In the conceptual design, we had a many-to-many relationship between Item and Receipt, containing a multi valued attribute. To appropriate it into a logical design, we turn this relationship into its very own relation/table, more suitable for containing extra attributes, and more capable of maintaining that many-to-many relationship between Item and Receipt.

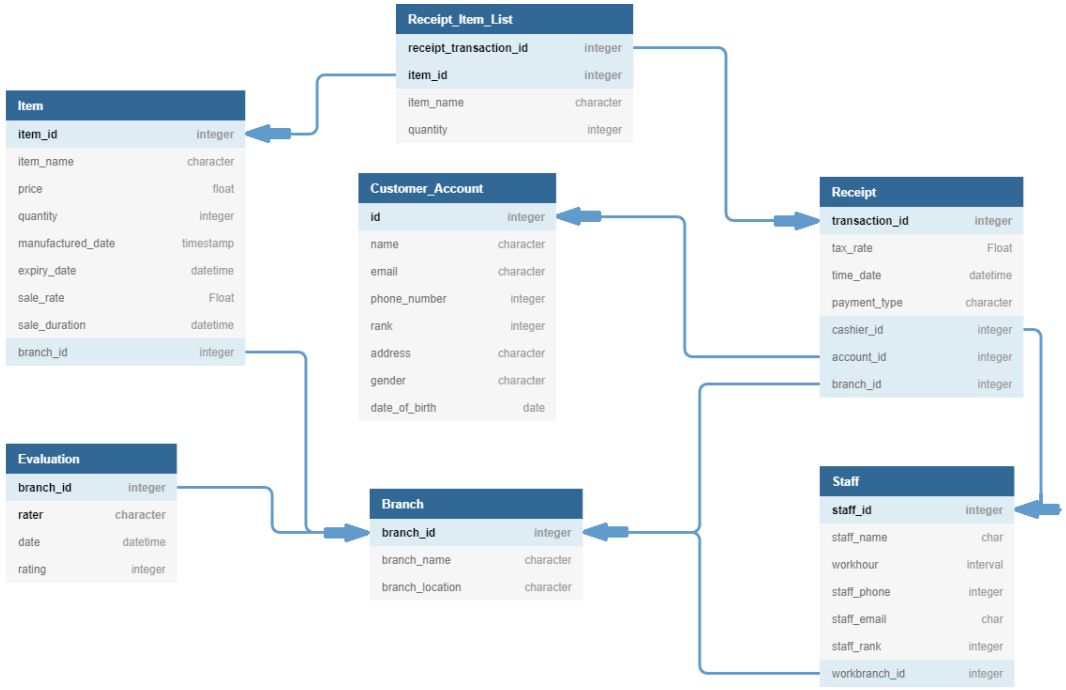


Figure 4: **The relational data model representing our logical design**

4 Implementation onto a Database Management System

4.1 Discussion on DBMS

4.1.1 What is Oracle APEX

When it comes to selecting a Database Management System for our project, we choose Oracle as our management system, based on how wide used it is, the system are design for data warehousing and the material provided in classes was most suitable with this.

More specifically, we implement the database via the Oracle Application Express platform (APEX), really helpful for setting up a database, simulating multiple root user so our team can work on the same database concurrently, and saving a history of SQL queries.

Though we must admit, the Oracle APEX platform offered to automation for a lot of things, namely the interface app builder, leading to us having a bit more of a learning curve to first understand the SQL queries, database systems and afterward understand how APEX automatically does those things to utilize it fully. In hindsight, it was not a good platform for learning about fundamentals of database systems.

4.1.2 How can Oracle APEX help us:

In the Oracle APEX Express Workshop, we discover how to develop database-enteric web applications quickly using Oracle Application Express. There is wide variety of option which we can use through interactive learning, with hands-on trials and errors result.

We Learn To:

- Create a database application for desktop web interfaces.

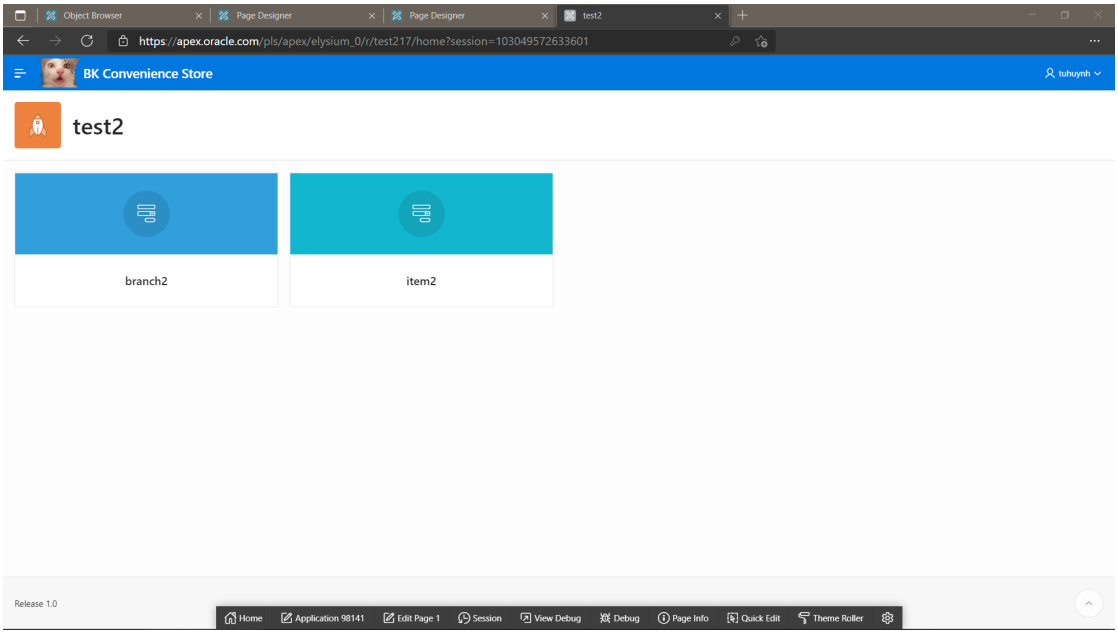


Figure 5: A screenshot of desktop web interfaces.(testing version)

- Add various components like new pages, reports regions, items and other components required to enhance an application.

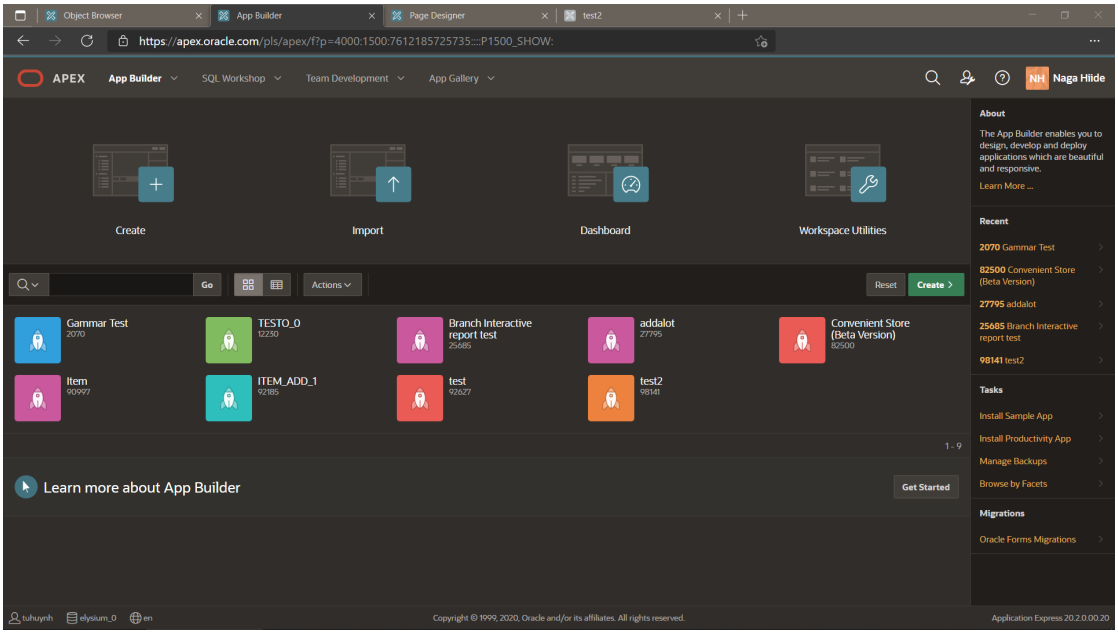


Figure 6: Behind the interface are pages with their attribute.

- Create processes and validations within an application.

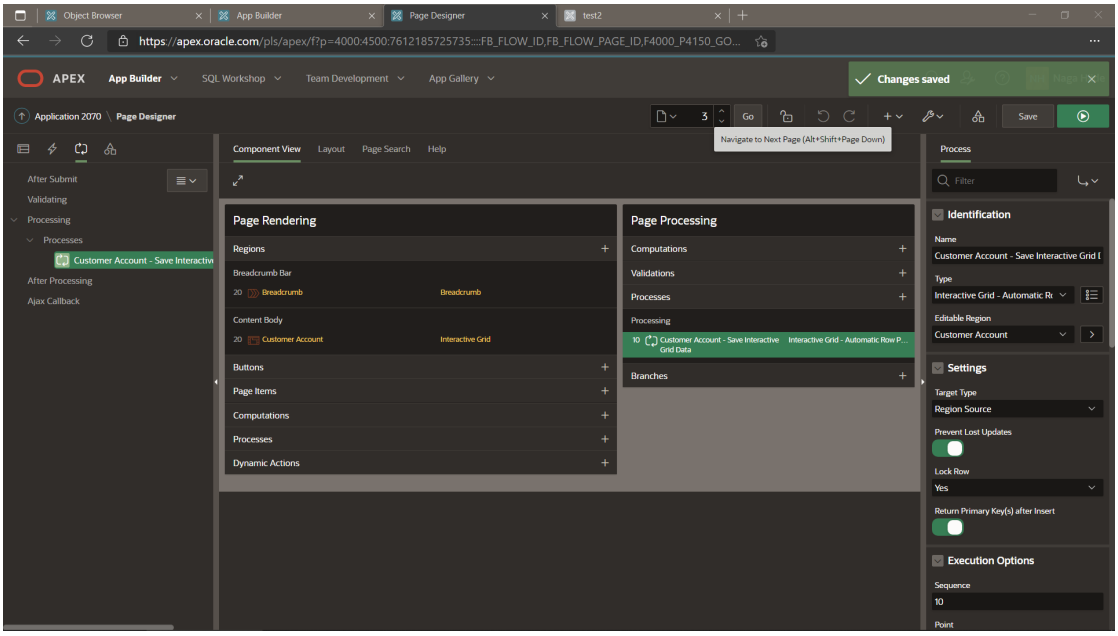


Figure 7: We can edit many things in back-end perspective in Page design.

- Implement security in an application.

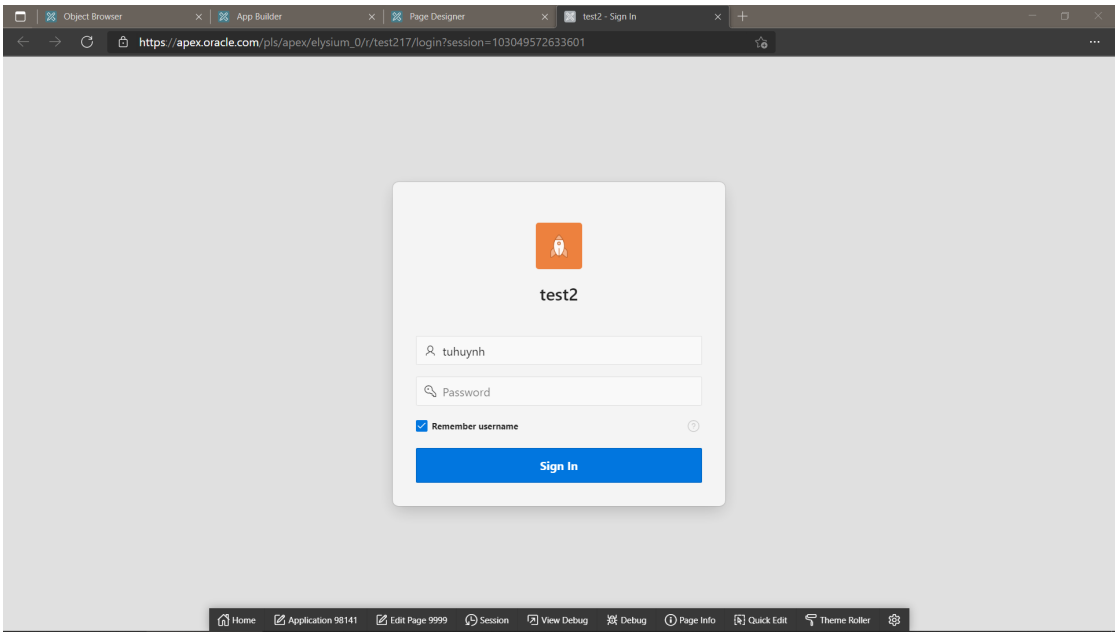


Figure 8: A screenshot of login's interface

- Manage and extend application by adding more components using some built-in wizards.

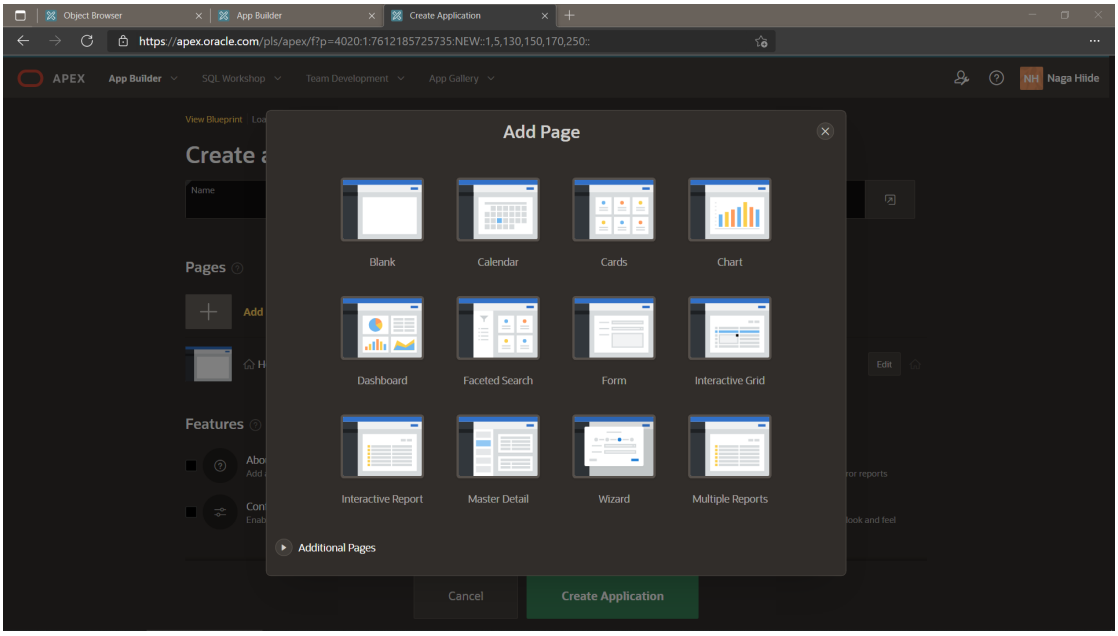


Figure 9: The wizard for creating new pages.

4.2 Physical design - the visual paradigm

On this step, we further expand the design on the logical step, assigning data types, nullability and making sure the foreign keys are referencing correctly.

4.3 Database implementation

We use commands of CREATE TABLE for creating each table in the physical database design, with appropriate data types and foreign keys references, here is a query to create the tables of RECEIPT, you can click here to view the full SQL script containing all queries for creating:

```
...
CREATE TABLE  "RECEIPT"
   ( "TRANSACTION_ID" NUMBER(10,0),
"TAX_RATE" FLOAT(10) NOT NULL ,
"TIME_DATE" TIMESTAMP (0) NOT NULL ,
"PAYMENT_TYPE" VARCHAR2(255) NOT NULL ,
"BRANCHBRANCH_ID" NUMBER(10,0) NOT NULL ,
"STAFFSTAFF_ID" NUMBER(10,0) NOT NULL ,
"CUSTOMER_ACCOUNTID" NUMBER(10,0),
 PRIMARY KEY ("TRANSACTION_ID")
   )
```

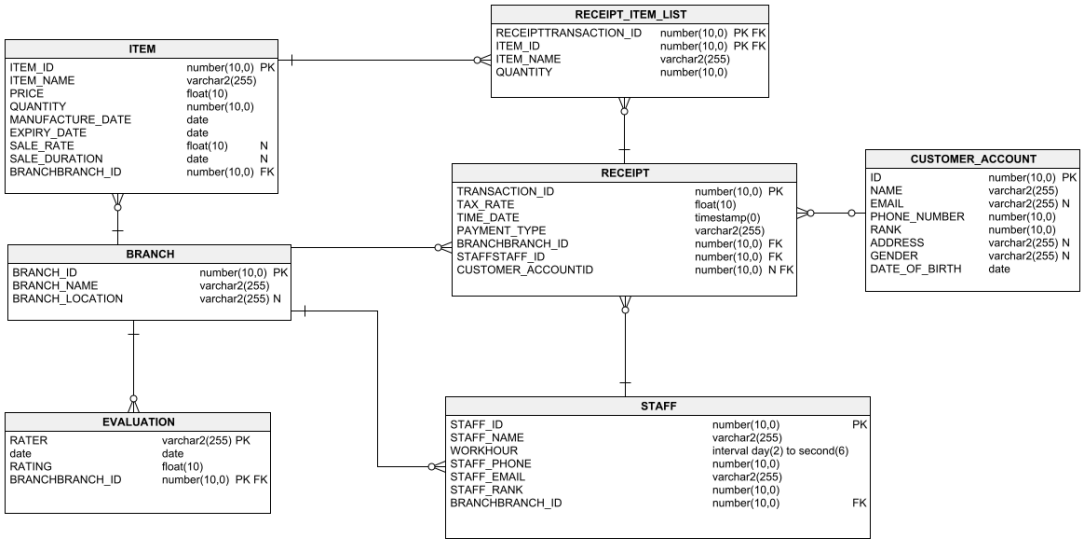


Figure 10: Our database physical design

```
ALTER TABLE  "RECEIPT" ADD CONSTRAINT "FKRECEIPT183360"
    FOREIGN KEY ("CUSTOMER_ACCOUNTID")
REFERENCES  "CUSTOMER_ACCOUNT" ("ID")

ALTER TABLE  "RECEIPT" ADD CONSTRAINT "FK_receipt_staff"
    FOREIGN KEY ("STAFFSTAFF_ID")
REFERENCES  "STAFF" ("STAFF_ID")

ALTER TABLE  "RECEIPT" ADD CONSTRAINT "FKRECEIPT620466"
    FOREIGN KEY ("BRANCHBRANCH_ID")
REFERENCES  "BRANCH" ("BRANCH_ID")
...
```

In result, we achieve the following tables in the database

Column Name	Data Type	Nullable	Default	Primary Key
BRANCH_ID	NUMBER(10,0)	No		1
BRANCH_NAME	VARCHAR2(255)	No		
BRANCH_LOCATION	VARCHAR2(255)	No		

Figure 11: "Branch" Table

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No		1
NAME	VARCHAR2(255)	No		
EMAIL	VARCHAR2(255)	Yes		
PHONE_NUMBER	NUMBER(10,0)	No		
RANK	NUMBER(10,0)	No		
ADDRESS	VARCHAR2(255)	Yes		
GENDER	VARCHAR2(255)	Yes		
DATE_OF_BIRTH	DATE	No		

Figure 12: "Customer_Account" Table

Column Name	Data Type	Nullable	Default	Primary Key
RATER	VARCHAR2(255)	No		1
BRANCHBRANCH_ID	NUMBER(10,0)	No		2
date	DATE	No		
RATING	FLOAT	No		

Figure 13: "Evaluation" Table

Column Name	Data Type	Nullable	Default	Primary Key
ITEM_ID	NUMBER(10,0)	No		1
ITEM_NAME	VARCHAR2(255)	No		
PRICE	FLOAT	No		
QUANTITY	NUMBER(10,0)	No		
MANUFACTURE_DATE	DATE	No		
EXPIRY_DATE	DATE	No		
SALE_RATE	FLOAT	Yes		
SALE_DURATION	DATE	Yes		
BRANCHBRANCH_ID	NUMBER(10,0)	No		

Figure 14: "Item" Table

Column Name	Data Type	Nullable	Default	Primary Key
TRANSACTION_ID	NUMBER(10,0)	No		1
TAX_RATE	FLOAT	No		
TIME_DATE	TIMESTAMP(0)	No		
PAYMENT_TYPE	VARCHAR2(255)	No		
BRANCHBRANCH_ID	NUMBER(10,0)	No		
STAFFSTAFF_ID	NUMBER(10,0)	No		
CUSTOMER_ACCOUNTID	NUMBER(10,0)	Yes		

Figure 15: "Receipt" Table

Column Name	Data Type	Nullable	Default	Primary Key
RECEIPTTRANSACTION_ID	NUMBER(10,0)	No		1
ITEM_ID	NUMBER(10,0)	No		2
ITEM_NAME	VARCHAR2(255)	No		
QUANTITY	NUMBER(10,0)	No		

Figure 16: "Receipt Item List" Table

Column Name	Data Type	Nullable	Default	Primary Key
STAFF_ID	NUMBER(10,0)	No		1
STAFF_NAME	VARCHAR2(255)	No		
WORKHOUR	INTERVAL DAY(2) TO SECOND(6)	No		
STAFF_PHONE	NUMBER(10,0)	No		
STAFF_EMAIL	VARCHAR2(255)	No		
STAFF_RANK	NUMBER(10,0)	No		
BRANCHBRANCH_ID	NUMBER(10,0)	No		

Figure 17: "Staff" Table

5 Database Normalization

5.1 Analyzing the normalization of the current database

Regarding 1NF: The requirement for achieving first normal form (1NF) is to have atomic columns(attributes) on all table.

Our design did have a small problem regarding the *list of items* bought, which is stored in a receipt, which we design as a multi-value attribute in the first conceptual design. However fortunately we changed it into a singular column(attribute) afterward in the logical design.

So we achieved first normal form - atomic attributes.

Regarding 2NF: For satisfying second normal form (2NF), we must first have the database on 1NF, and in addition have every non-prime attribute of the relation is dependent on the whole of every candidate key. On examination of our RECEIPT_ITEM_LIST table:

Column Name	Data Type	Nullable	Default	Primary Key
RECEIPTTRANSACTION_ID	NUMBER(10,0)	No		1
ITEM_ID	NUMBER(10,0)	No		2
ITEM_NAME	VARCHAR2(255)	No		
QUANTITY	NUMBER(10,0)	No		

Figure 18: Original receipt_item_list relation

We can see that the attribute of ITEM_NAME is dependent on only ITEM_ID and not RECEIPTTRANSACTION_ID, violating the rule of second normal form.

5.2 Solution for improving the Form Normalization:

Regarding 2NF: On examination of the ITEM table, we can see that we already saved the ITEM_NAME data over at that table, so the ITEM_NAME under RECEIPT_ITEM_LIST is completely redundant and can be removed entirely.

Resulting in this table for RECEIPT_ITEM_LIST, and satisfying the requirement for 2NF:

Column Name	Data Type	Nullable	Default	Primary Key
RECEIPTTRANSACTION_ID	NUMBER(10,0)	No		1
ITEM_ID	NUMBER(10,0)	No		2
QUANTITY	NUMBER(10,0)	No		

Figure 19: Improved receipt_item_list relation

Further improvements:

With the design our current ITEM table, when a type of merchandise is expired or sold out, and a new shipment of that same type arrive, there would be a redundancy in the old entry, it is not for sale anymore and have no use.

Furthermore the field of ITEM_NAME would also have to retyped in, repeatedly every time we restock a certain type of merchandise many times.

In addition, in reality, a certain type of item would have a fixed duration until its expiration, so

we could have the database calculate the expiration date on every shipment of every item type, instead of having to log a separate expiration date in the system every time we restock that type of item.

Using these ideas, we could change the ITEM relation design in the database into these 2(two) separate table, achieving better sufficiency:

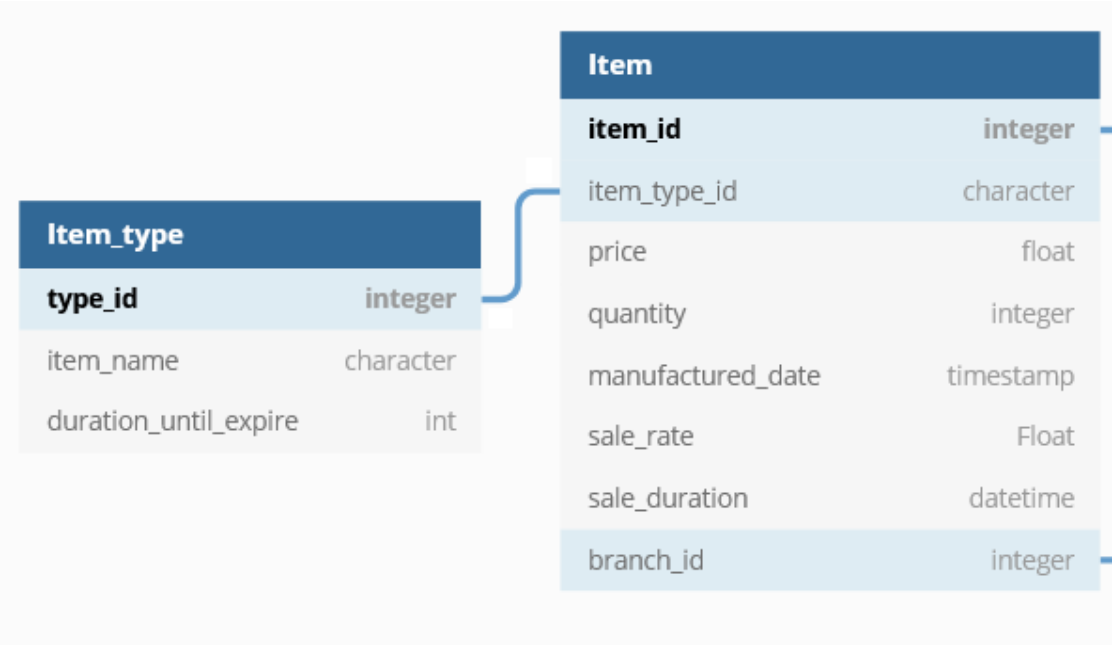


Figure 20: Improved Item Table

6 Database security

With the database in this current state, there is virtually no security. But we designed a security system as following:

Based on the original requirement, we took notice that the database will be:

- Database can be accessed at multiple branches
- The managers can create, modify and view all tables
- The staff members can create, modify and view all tables **except** BRANCHES and STAFF
- Customers can view ITEMS and ITEM_TYPES for browsing, and create/modify their EVALUATION entry.

According to these requirement, we can implement a Role-Based access control (RBAC) mechanism, creating specific roles with specific privileges. The design can be seen through the following RBAC matrix

	BRANCH	STAFF	RECEIPT	RECEIPT_ITEM_LIST	ITEM	CUSTOMER_ACCOUNT	EVALUATION
Manager	rw	rw	rw	rw	rw	rw	rw
Staff			rw	rw	rw	rw	rw
Customer account			r	r	r	rw	rw*
Guest account					r		

Figure 21: RBAC matrix

r : privilege to read, meaning being able to SELECT on data tables

w : privilege to write data, including the SQL commands of INSERT, MODIFY, UPDATE. DROP COLUMN, ALTER;

* : Each user(customer account) should only be able to create 1 evaluation, and only being able to change their very own evaluation. Meaning they would only modify 1(one) row in the evaluation table, not the entire table.

Afterward, for every person that uses the database would have their own user account with password secured

6.1 Creating and granting privileges to roles:

Following the designs in the previous section, we come up with a few SQL commands that helps implementing the RBAC security system. Firstly we would grant the necessary privileges to every role in the system, including *manager, staff, customer account* and *guest account*

For example:

```
CREATE ROLE manager;
GRANT SELECT ON BRANCH TO manager;
GRANT CREATE ON BRANCH TO manager;
...
(repeat for every necessary privileges on corresponding tables to every roles)
```

Though we do notice that every users should be able to read from the *ITEM* tables regardless of roles, we can run this line to speed up the process:

```
GRANT SELECT ON ITEM TO PUBLIC
```

Afterward, we would assign the users to their corresponding roles:

```
GRANT manager TO [manager_user_name_goes_here];
GRANT staff TO [staff_user_name_goes_here];
...
```

6.2 Limitation of this security system:

On designing the role-based system, we are not entirely sure on how to implement the privilege for managers to create user account for their staff members, or how would staff members create an account for customers. We did think about utilizing the [*WITH GRANT OPTION*] command when granting privileges. But that would be a quite serious vulnerability if a certain person hack the users with [*WITH GRANT OPTION*] privileges, namely if a manager forgot to log out of their session. So for now, only the root user can create new user accounts.

Furthermore there is also the requirement to make sure customers can only create 1 review and only be able to change their review, needing a system of security on column-level in the EVALUATION table. To which we must admit we are not quite experienced in.

7 Utilizing the database functions with SQL queries

7.1 Basic queries

7.1.1 CREATE SEQUENCE

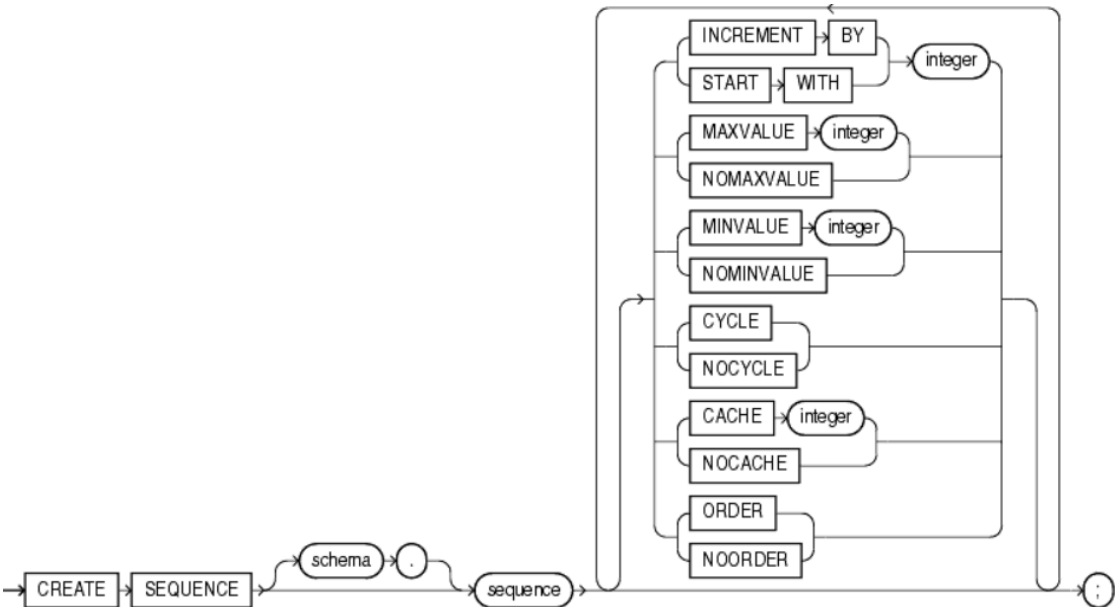
7.1.1.a The need for sequences

We intend to create a trigger for automatically generate an ID for each time a new data is inserted into the database. To do that, what we need is:

- A sequence for each table
- A coordinate trigger for each table

7.1.1.b Overview of sequences creation in SQL:

The structure for creating a sequence is :



Important Semantics which we will use:

- INCREMENT BY

Specify the interval between sequence numbers. This integer value can be any positive or negative integer, but it cannot be 0. This value can have 28 or fewer digits. The absolute of this value must be less than the difference of MAXVALUE and MINVALUE. If this value is negative, then the sequence descends. If the value is positive, then the sequence ascends. If you omit this clause, then the interval defaults to 1.

- START WITH

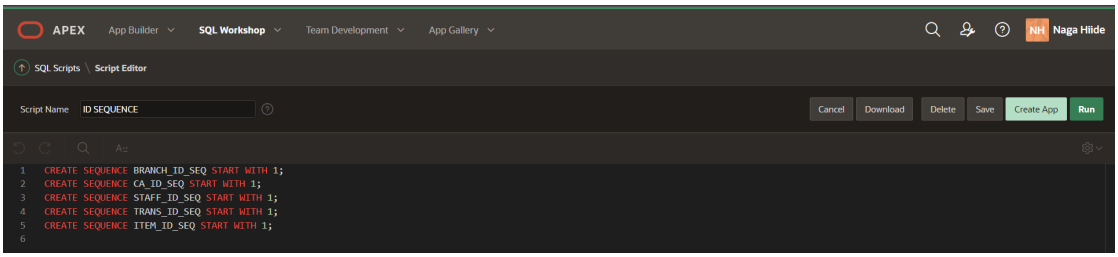
Specify the first sequence number to be generated. Use this clause to start an ascending sequence at a value greater than its minimum or to start a descending sequence at a value less than its maximum. For ascending sequences, the default value is the minimum value of the sequence. For descending sequences, the default value is the maximum value of the sequence. This integer value can have 28 or fewer digits.

- Order

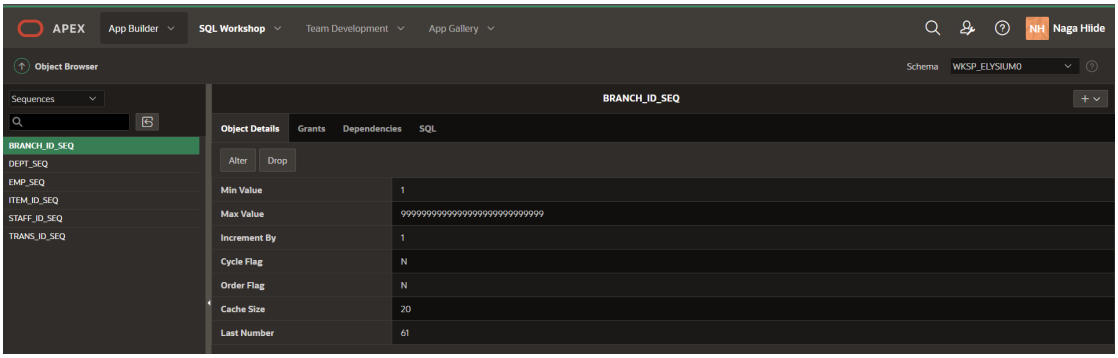
Specify ORDER to guarantee that sequence numbers are generated in order of request. This clause is useful if you are using the sequence numbers as timestamps. Guaranteeing order is usually not important for sequences used to generate primary keys. That's the reason why our schema won't acquire this.

7.1.1.c Sequences implementation:

We access to our schema via Oracle's Apex SQL Workshop and generate those aforementioned sequences:



After running the SQL commands, we now have the newly created sequences ready to go.



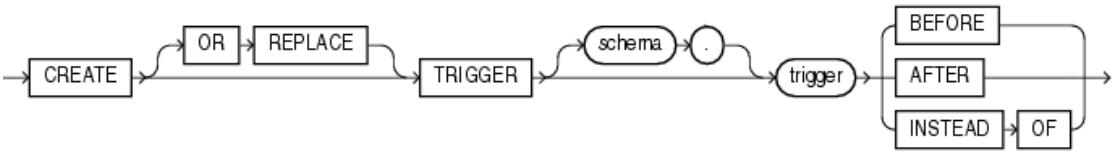
7.2 Triggers

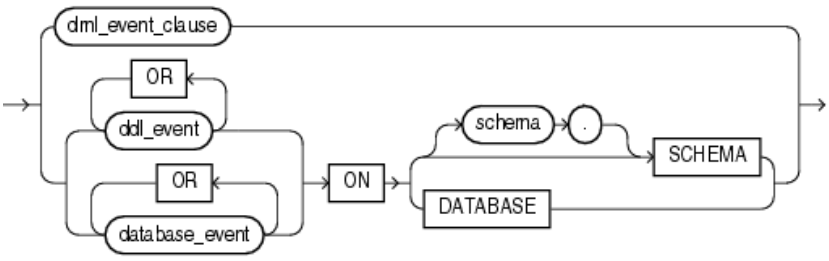
The next step in auto-ID-generation is to create a trigger so that whenever we insert values into the table, we don't have to set the ID ourselves.

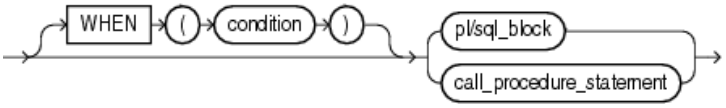
7.2.1 Recap about Trigger syntax in SQL:

Below are the syntax structure of trigger:

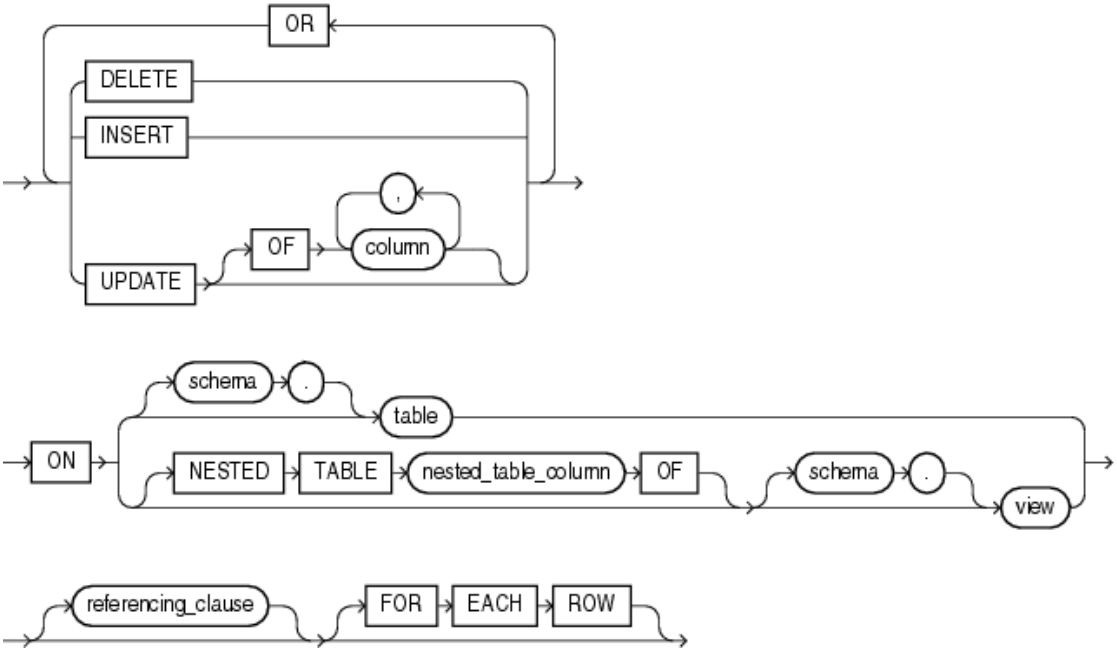
create_trigger::=



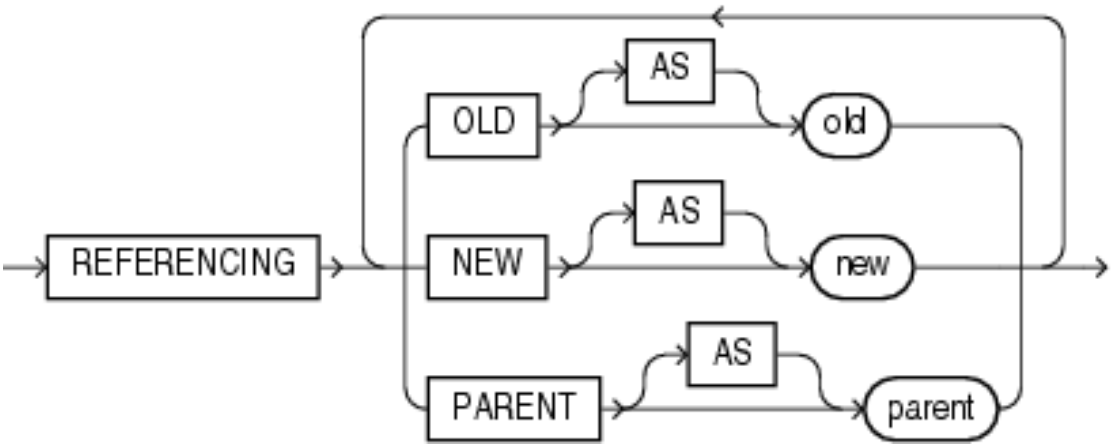




DML_event_clause ::=



referencing_clause::=



Important sematics which we'll use:

- **DML_event_clause**

The DML_event_clause lets you specify one of three DML statements(DELETE, INSERT, UPDATE that can cause the trigger to fire. Oracle Database fires the trigger in the existing user transaction.

You cannot specify the MERGE keyword in the DML_event_clause. If you want a trigger to fire in relation to a MERGE operation, you must create triggers on the INSERT and UPDATE operations to which the MERGE operation decomposes.

- **referencing_clause**

The referencing_clause lets you specify correlation names. You can use correlation names in the PL/SQL block and WHEN condition of a row trigger to refer specifically to old and new values of the current row. The default correlation names are OLD and NEW. If your row trigger is associated with a table named OLD or NEW, use this clause to specify different correlation names to avoid confusion between the table name and the correlation name.

If the trigger is defined on a nested table, then OLD and NEW refer to the row of the nested table, and PARENT refers to the current row of the parent table.

If the trigger is defined on an object table or view, then OLD and NEW refer to object instances.

- OR REPLACE

Specify OR REPLACE to re-create the trigger if it already exists. Use this clause to change the definition of an existing trigger without first dropping it.

- BEFORE

Specify BEFORE to cause the database to fire the trigger before executing the triggering event. For row triggers, the trigger is fired before each affected row is changed.

Restrictions on BEFORE Triggers BEFORE triggers are subject to the following restrictions:

You cannot specify a BEFORE trigger on a view or an object view.

You can write to the :NEW value but not to the :OLD value.

- INSERT

Specify INSERT if you want the database to fire the trigger whenever an INSERT statement adds a row to a table or adds an element to a nested table.

- FOR EACH ROW

Specify FOR EACH ROW to designate the trigger as a row trigger. Oracle Database fires a row trigger once for each row that is affected by the triggering statement and meets the optional trigger constraint defined in the WHEN (OR BEFORE) condition.

7.2.2 Trigger Implementation:

As we have discussed, we will create trigger for Branch, Staff, Customer Account, Item and Receipt. First we go to SQL workshop within APEX to create Trigger's SQL script, below is an example of trigger for Transaction (Receipt):

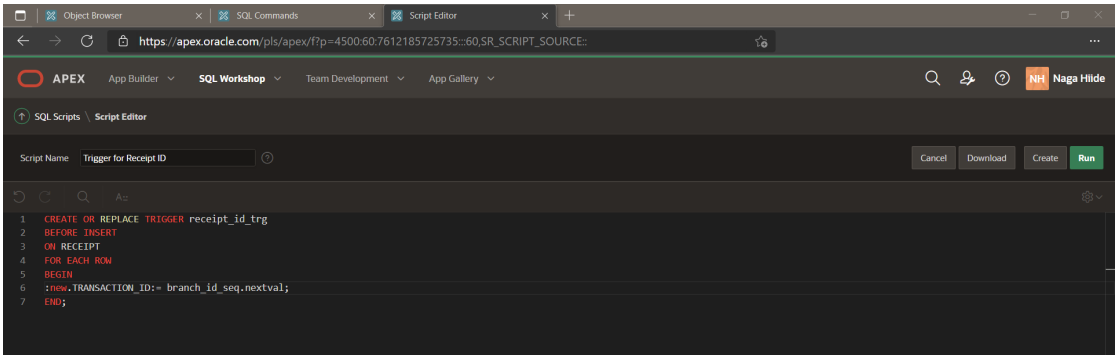


Figure 22: Trigger for Receipt ID Auto-generating.

When we finish edit the script, we run the it by clicking in the green "run" button in the top right of the editor.

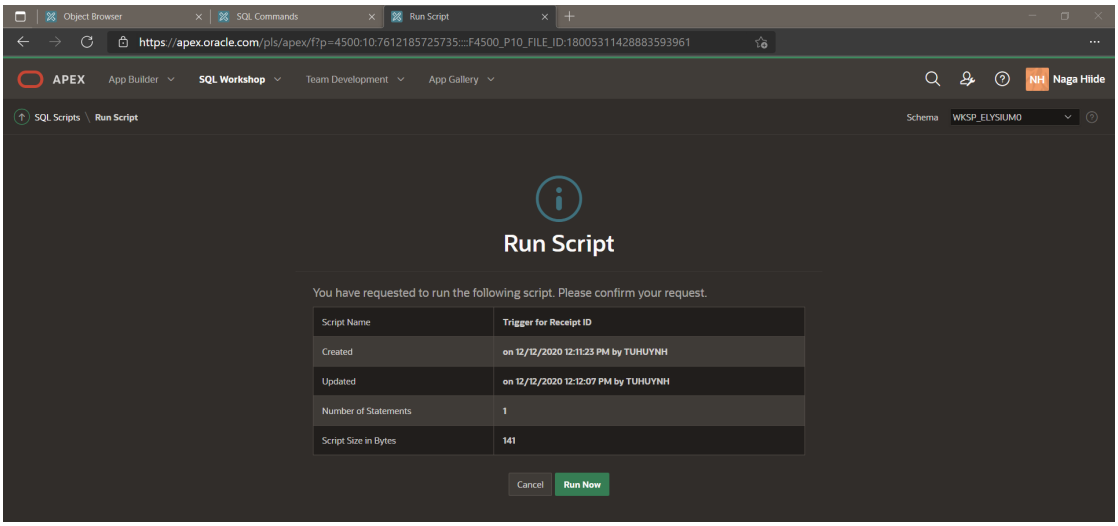


Figure 23: We will have to confirm before commit.

Even though we have included the REPLACE semantic into the SQL code, we still need to drop the trigger everytime we fail to create a new one, the debug result will be presented soon after our confirmation.

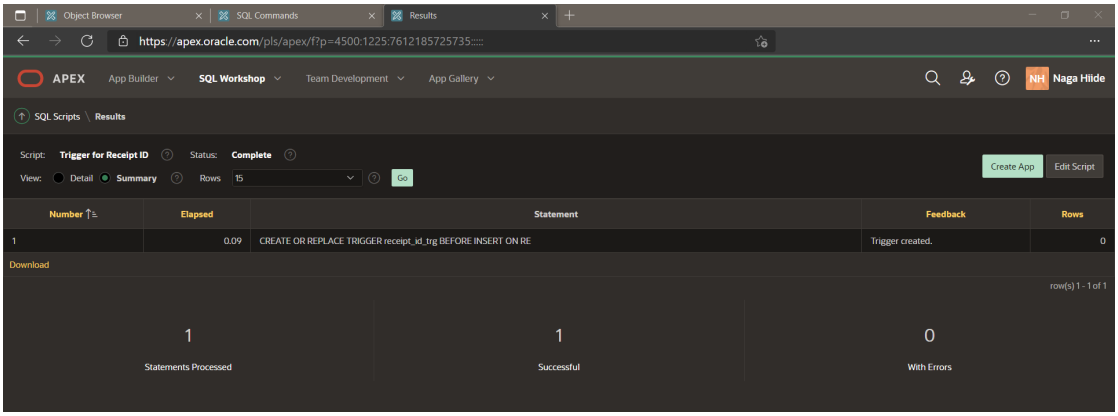


Figure 24: Our Trigger code generated successfully.

8 Basic interface

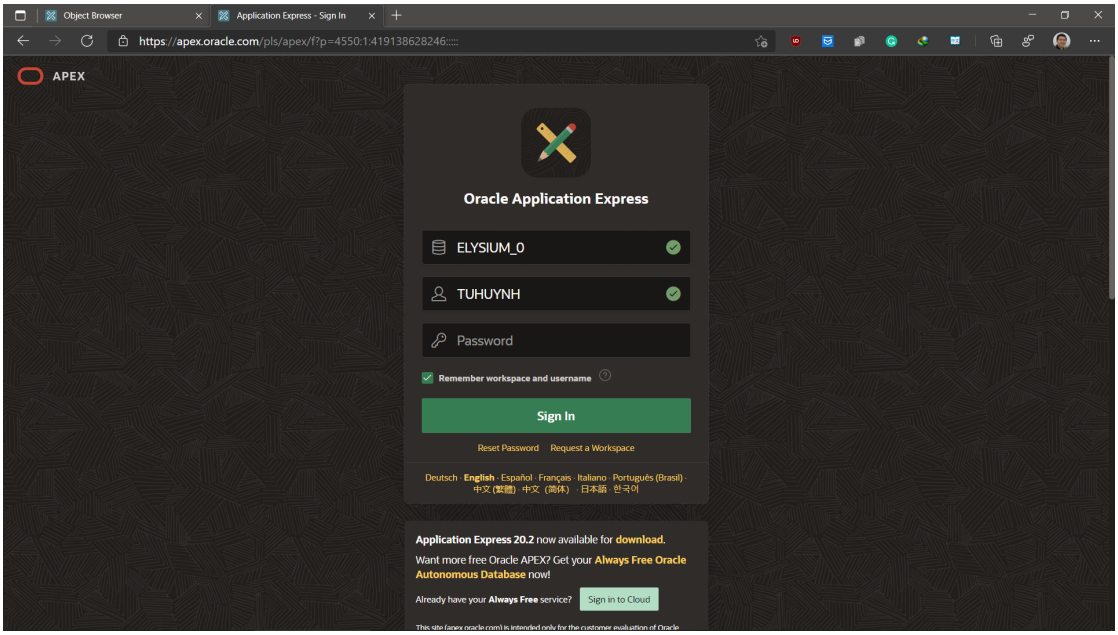


Figure 25: Workspace Login Screen

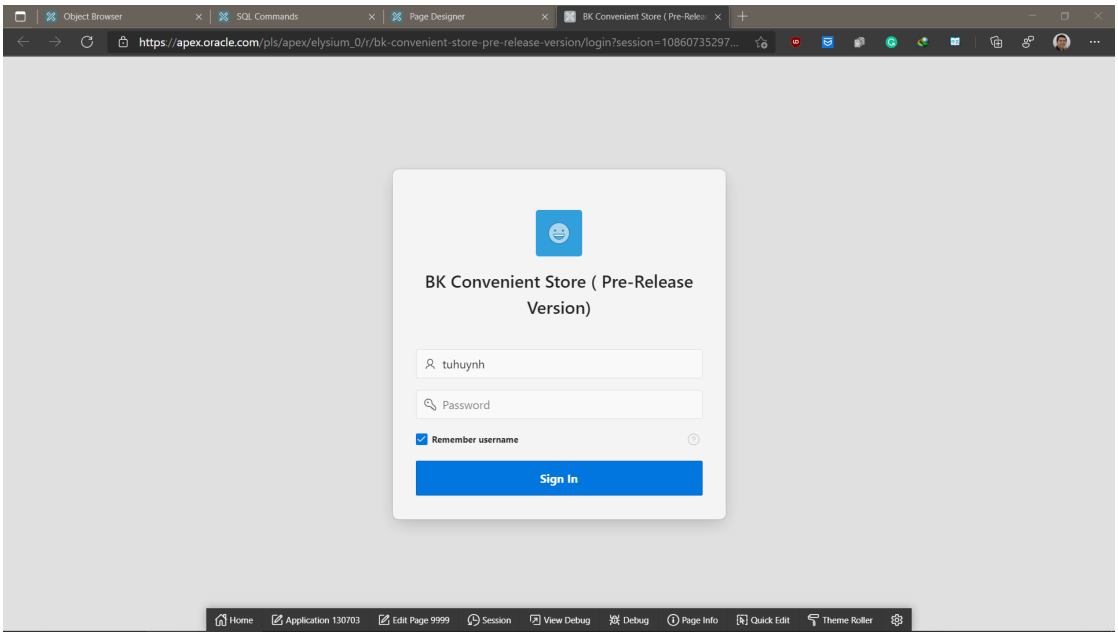


Figure 26: App Login Screen (We login as a Administrator)

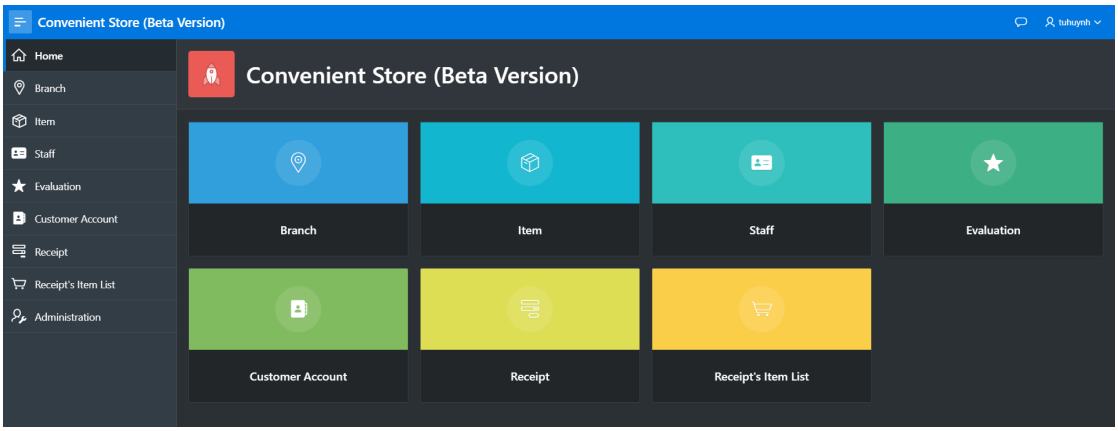


Figure 27: A screenshot of Application's interface

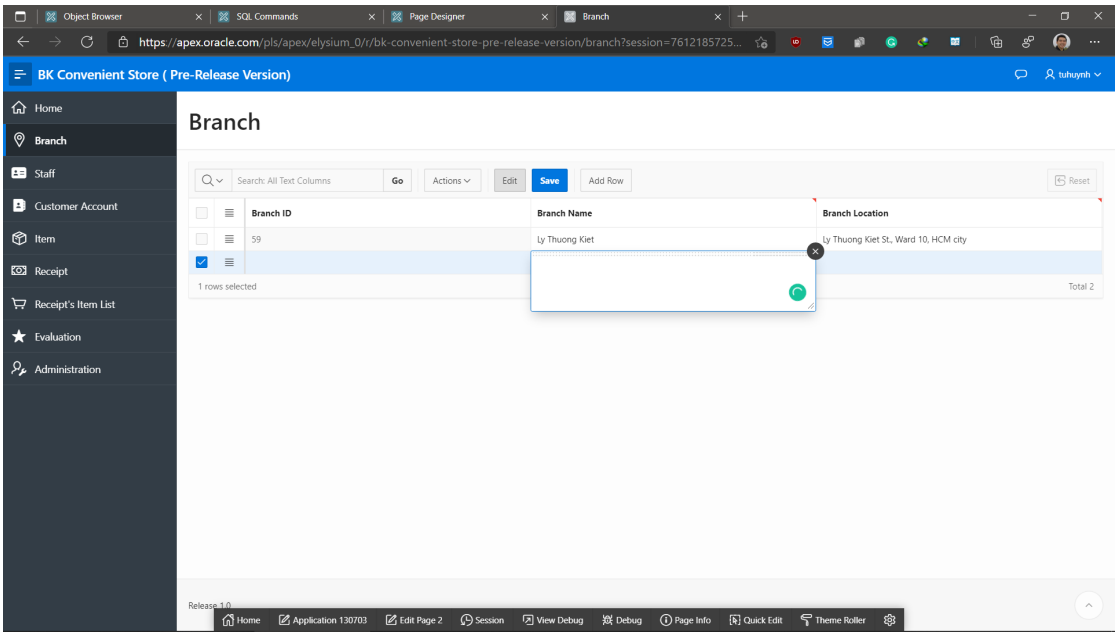


Figure 28: Branch Interface

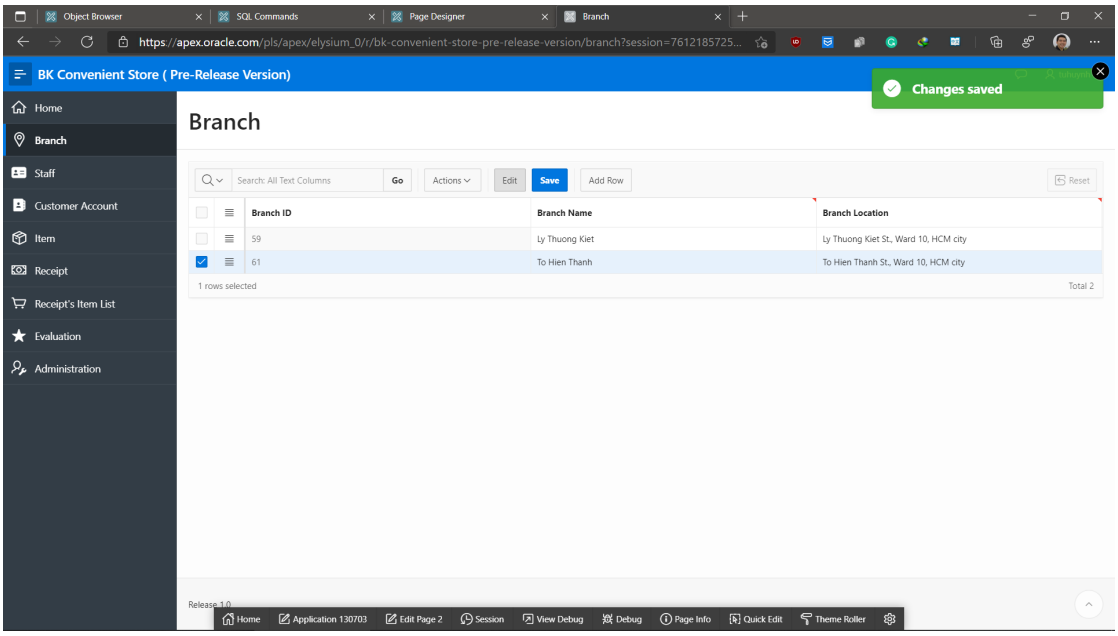


Figure 29: We create some data for branch

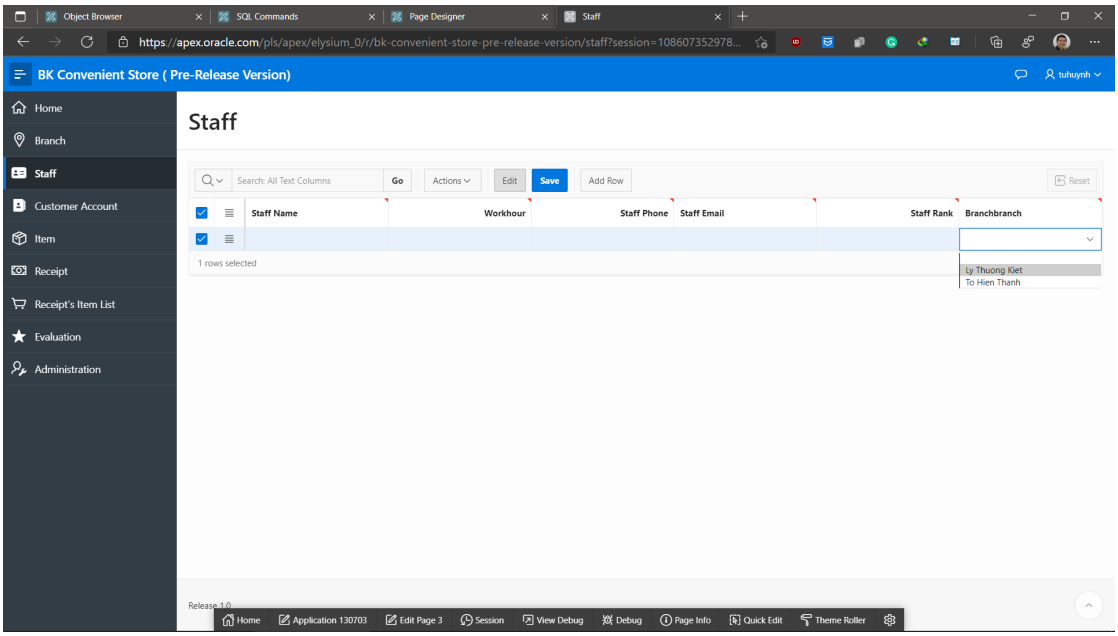


Figure 30: Then for Staff

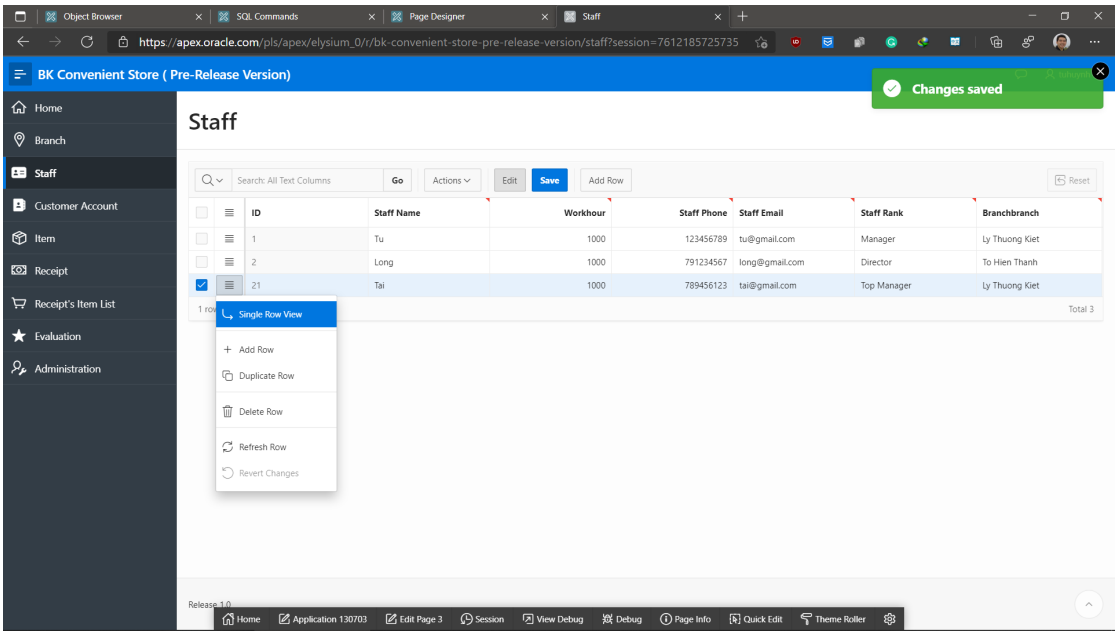


Figure 31: Staff data interface

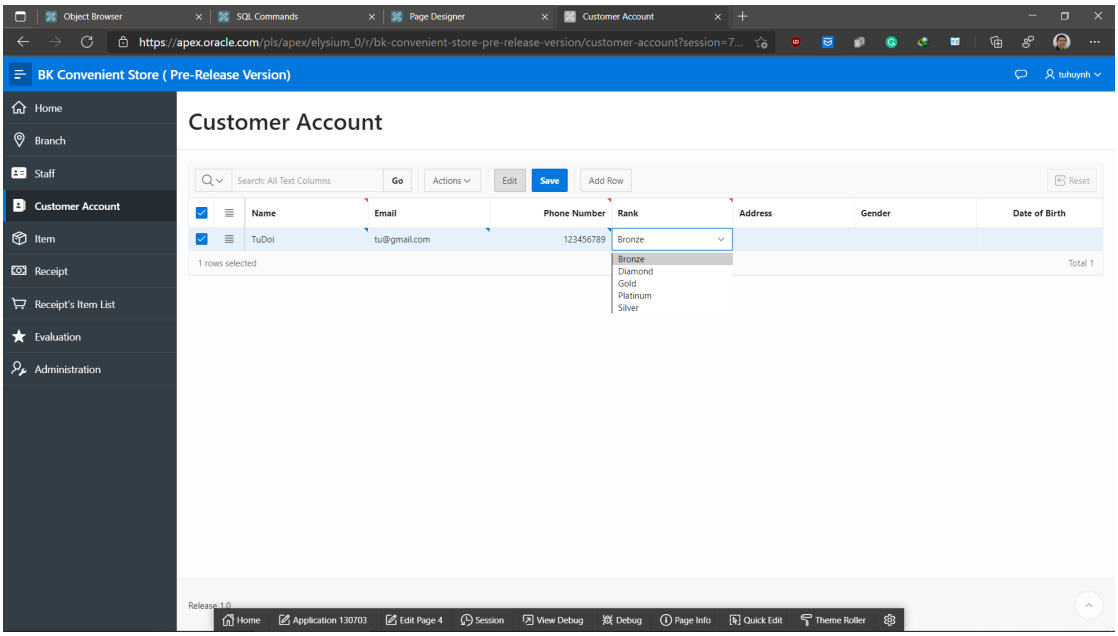


Figure 32: Customer Account interface

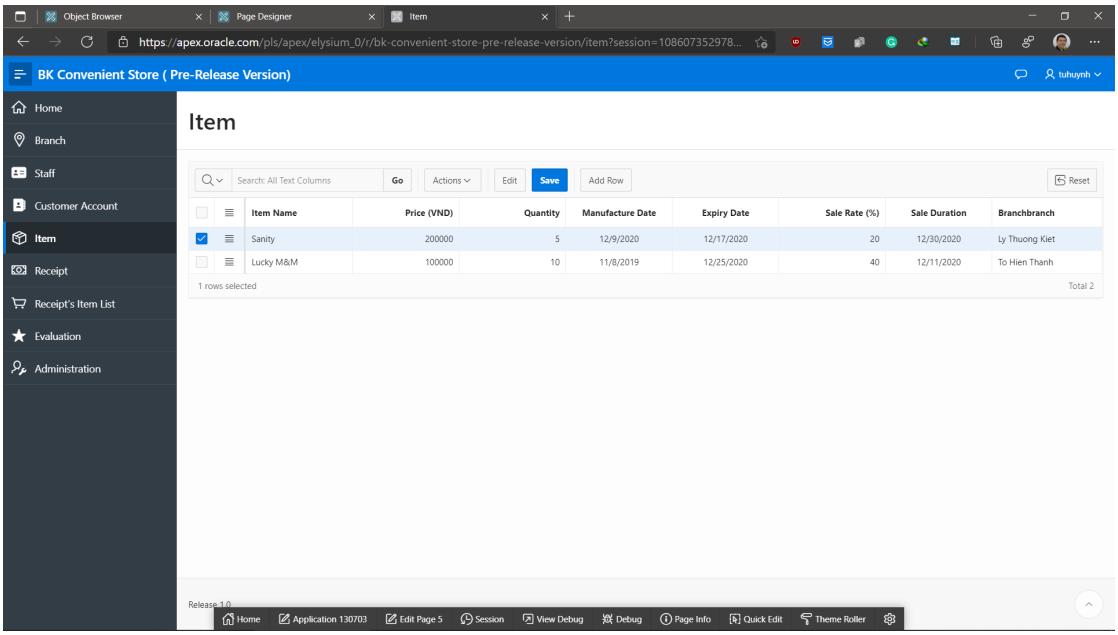


Figure 33: Item data

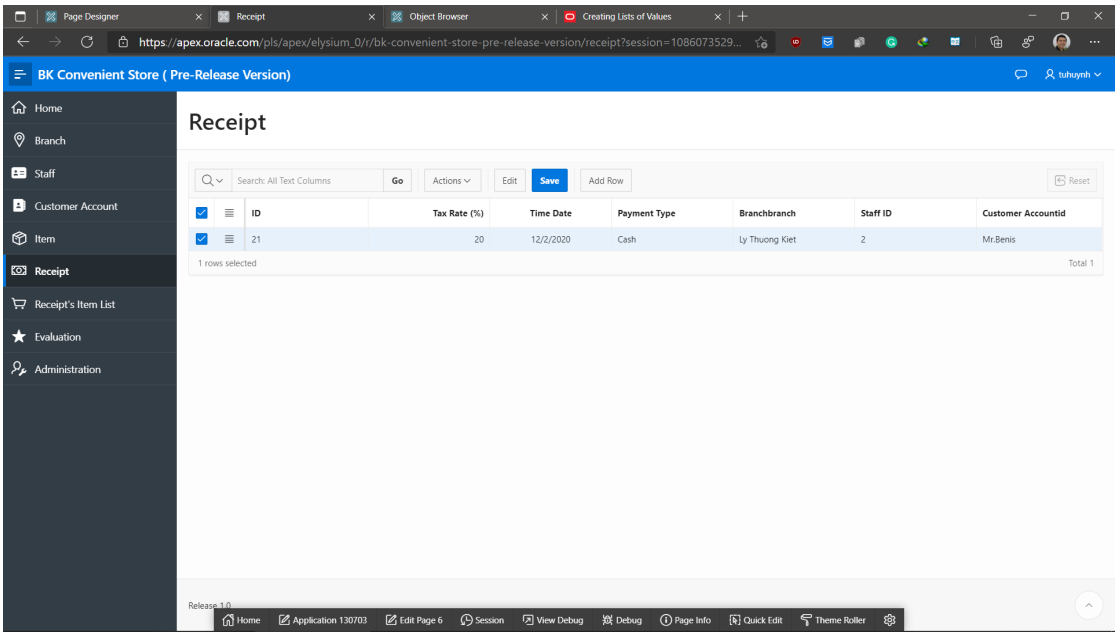


Figure 34: Receipt data

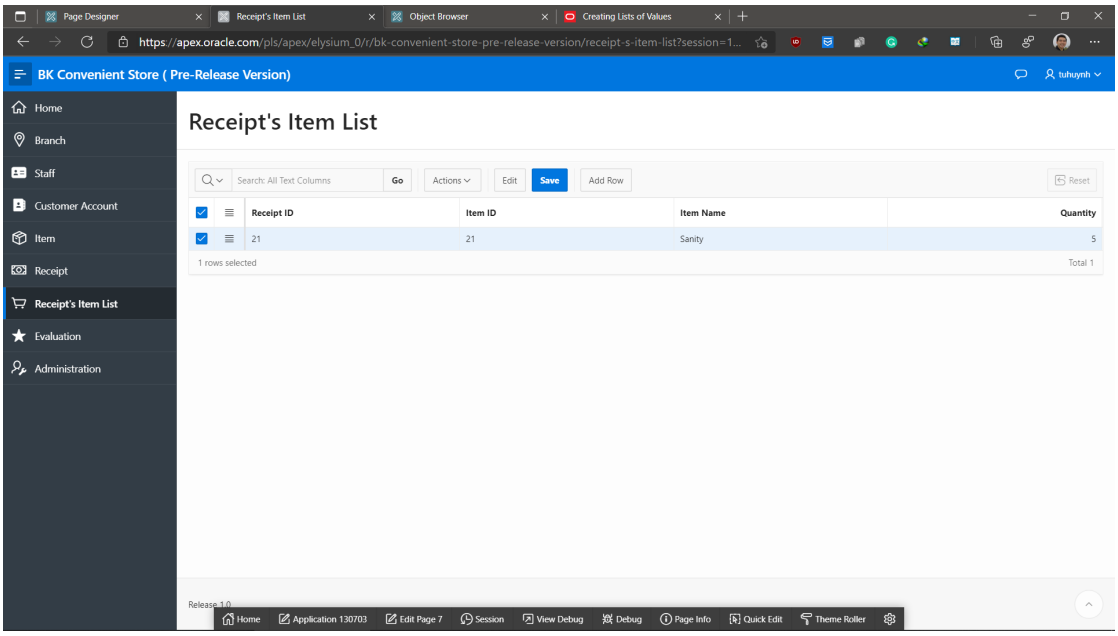


Figure 35: Receipt's Item list simple interface

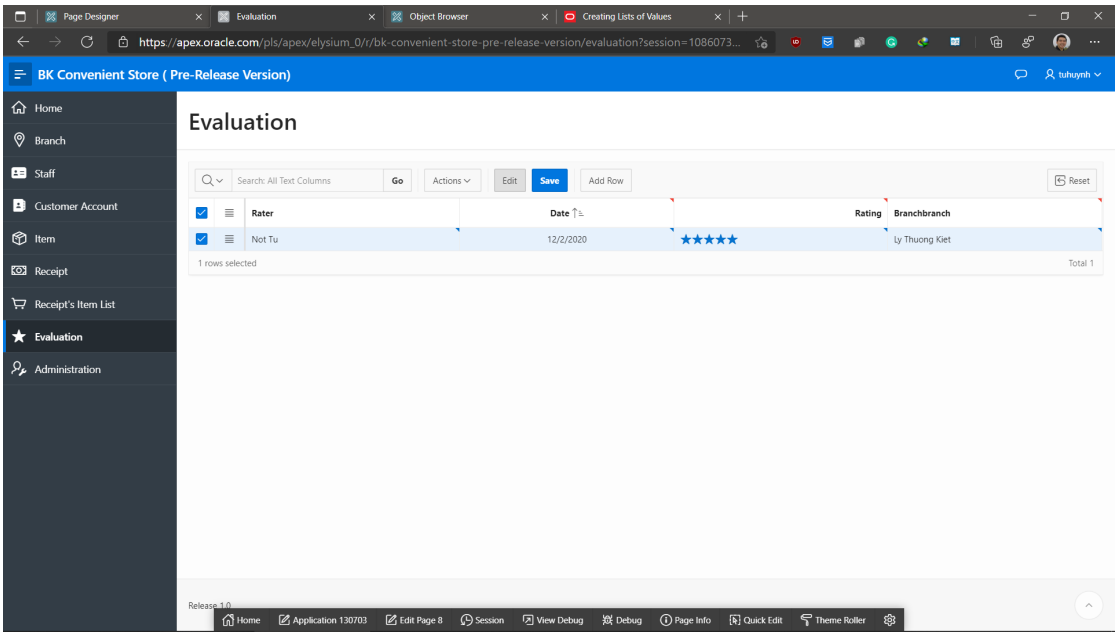


Figure 36: Evaluation tab

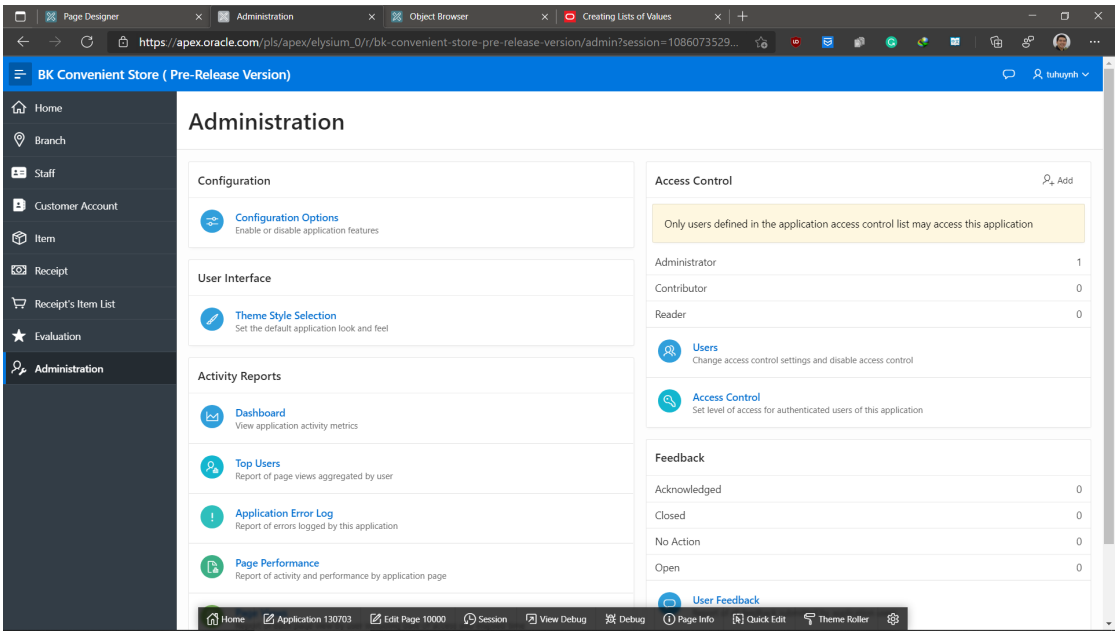


Figure 37: Administration setting