**CS 4318**

**Post Office Database**

**By**

**Nhat Tran**

1. **Abstract:**

The purpose of this project is to create a database for a post office company like FedEx or UPS. The database should be well optimized, easy to access, and modify. The database will have 7 datasets and will have super keys and foreign keys for linking and accessing each other dataset for handling information. For this database, there will be 4 main actors: customers/guests, customer services, drivers, and managers. For the dataset, there will be 7 datasets: employees’ personal information, payment’s receipt, customers’ information, packages’ information, branches’ information, trucks’ information, and accounts’ information.

1. **Mission Statement:** Create a database for a post office company should be easily accessible, modify, and optimize.
2. **Mission Objective:** A database for a post office company should be able to:

* User-friendly views for any types of users.
* Fast access and protect integrity and privacy of each dataset depend on major users.
* The database will consist of 7 datasets in total: Customer, Receipt, Package Type, Package, Branch, Staff, and Truck.
* The database will consist of 4 major users: Customers, managers, customer services, and drivers.
* Reduce redundancy/complication of reading/modify datasets.

1. **Entity – Relation Diagram:**

A screenshot of a computer

Description automatically generated

1. **Relation Model:**

A screenshot of a computer

Description automatically generated

1. **Normalization Check:**

* **Staff:**

1. Primary Key: StaffID
2. Function Dependency: StaffID -> lastName, firstName, Sex, jobTitle, PhoneNum, Salary, BranchNo
3. Foreign key: BranchNo(parent table: Branch)
4. This table is 1NF because there is only one data in each attribute domain.
5. This table is 2NF because all domains depend on the primary key.
6. This table is not 3NF because phoneNum could be a primary key to identity staff’s ID.
7. This table is not BCNF because this table is not 3NF.

* **Branch:**

1. Primary Key: BranchNo
2. Functional Dependency: BranchNo -> Address, City, State, Profits
3. Foreign Key: None
4. This table is 1NF because there is only one data in each attribute domain.
5. This table is 2NF because all domains depend on the primary key.
6. This table is not 3NF because if the State changes, City will forcefully change as well.
7. This table is not BCNF because it is not 3NF.

* **Truck:**

1. Primary key: TruckID
2. Function Dependency: TruckID -> PlatNum, DriverID, Origin, IsAvailable
3. Foreign Key: DriverID (parent table: Staff)
4. This table is 1NF because there is only one data in each attribute domain.
5. This table is 2NF because all domains depend on the primary key.
6. This table is not 3NF because if there are any changes to IsAvailable, DriverID must be changed to either null or not null value.
7. This table is not BCNF because it is not 3NF.

* **Package Type:**

1. Primary Key: TypeName
2. Function Dependency: TypeName -> Dimension, MaxWeight, ValuePerWeight
3. Foreign Key: None
4. This table is 1NF because there is only one data in each attribute domain.
5. This table is 2NF because all domains depend on the primary key.
6. This table is 3NF because non- primary domains do not depend on each other.
7. This table is BCNF because there is only one primary key and non-primary domains fully dependent.

* **Package:**

1. Primary Key: PackageID
2. Function Dependency: PackageID -> PackageType, PackageWeight, Amount, TrackingID
3. Foreign Key: PackageType(parent table: PackageType), TrackingID(parent table: Customer)
4. This table is 1NF because there is only one data in each attribute domain.
5. This table is 2NF because all domains depend on the primary key.
6. This table is not 3NF because PackageType and PackageWeight can change the Amount of the package.
7. This table is not BCNF because it is not 3NF.

* **Receipt:**

1. Primary Key: ReceiptID
2. Function Dependency: ReceiptID -> Sender, BranchNo, CardNum, Total, TrackingID
3. Foreign Key: BranchNo(parent table: Branch), TrackingID(parent table: Customer)
4. This table is 1NF because there is only one data in each attribute domain.
5. This table is 2NF because all domains depend on the primary key.
6. This table is not 3NF because TrackingID could also be a primary key due to its unique string.
7. This table is not BCNF because it is not 3NF.

* **Customer:**

1. Primary Key: TrackingID
2. Function Dependency: TrackingID-> Sender, Recepient, Origin, Destination
3. Foreign Key: None
4. This table is 1NF because there is only one data in each attribute domain.
5. This table is 2NF because all domains depend on the primary key.
6. This table is 3NF because non-primary domains do not depend on each other.
7. This table is BCNF because there is only one primary key and non-primary domains are fully dependent.
8. **Use cases:**

**All Major Users: Managers, Customer Services, Drivers, and Customers**

**Employees’ personal information:**

1. **Use Case 1: Add Employee**
2. **Actor/User: Manager**
3. **Steps:**
4. User clicks on “Add Staff” button.
5. The dataset will generate id.
6. Wait for user to enter name, DOB, gender, job title, branch information, and phone number.
7. All information is displayed and wait for confirmation.
8. User clicks on “Confirm” button.
9. **SQL Statement:**

INSERT INTO Staff (StaffID, LastName, FirstName, Sex, JobTitle, PhoneNum, Salary, BranchNo)

VALUES (1, 'Smith', 'John', ‘Male’, 'Manager', ‘111-111-1111’, 25.00, ‘Bn01’);

1. **Use Case 2: Delete Employee**
2. **Actor/User: Manager**
3. **Steps:**
4. User clicks on “Delete Staff” button.
5. The dataset will be displayed and ask who the user wants to delete.
6. User enters the id of the employee.
7. The information will be displayed and confirmed with the employee.
8. User clicks “Confirm” button.
9. **SQL Statement**

DELETE FROM Staff WHERE StaffID =1;

1. **Use Case 3: Edit Employee’s First Name**
2. **Actor/User: Manager**
3. **Steps:**
4. User clicks on “Edit Staff” button.
5. The dataset will be displayed and ask who the user wants to edit.
6. User enters the id of the employee.
7. The information will be displayed and ask which information the user wants to edit.
8. User will select the option and enter the new information.
9. The dataset will confirm the change and user clicks “Confirm” button.
10. **SQL Statement:**

UPDATE Staff

SET FirstName = ‘Jane’

WHERE StaffID = 1;

1. **Use Case 4: View Employee’s List**
2. **Actor/User: Manager**
3. **Steps:**
4. User accesses to the staff menu.
5. User clicks on “View Employee’s List” button.
6. Dataset will show the list of employees (only the Staff ID, names, Sex, PhoneNum, and Job Title).
7. **SQL Statement:**

SELECT StaffID, LastName, FirstName, Sex, PhoneNum, JobTitle

FROM Staff;

1. **Use Case 5: Get Amount of Employees**
2. **Actor/User: Manager**
3. **Steps:**
4. User accesses to the staff menu.
5. User clicks on “View Employee’s List” button.
6. Dataset will show the list of employees.
7. At the bottom will show how many employees are in a branch (depend on manager’s location).
8. **SQL Statement:**

SELECT BranchNo, COUNT(StaffID) AS Total

FROM Staff

WHERE BranchNo = ‘Bn01’;

1. **Use Case 6: View list of employees and cities they work at**
2. **Actor/User: Manager**
3. **Steps:**
4. User accesses to the staff menu.
5. User clicks on “View All Employee’s List” button.
6. Dataset will show the list of employees.
7. At the bottom will show how many employees are in a branch
8. **SQL Statement:**

SELECT Staff.firstName, Staff.lastName, staff.jobTitle, branch.BranchNo, branch.City

FROM StafF

JOIN Branch ON Staff.BranchNo = Branch.BranchNo

ORDER BY Branch.City;

**Payment’s Receipt:**

1. **Use Case 1: Add Receipt**
2. **User/Actor: Customer Service, Manager**
3. **Steps:**
4. User clicks on “Add payment” button.
5. The dataset will generate ID of the payment and ask for payment information such as card numbers, totals, and tracking ID of a package.
6. Dataset will automatically save the new information.
7. **SQL Statement:**

INSERT INTO Receipt (ReceiptID, Sender, Branch, CardNum, Total, TrackingID)

VALUES (1, ‘Janna James’, ‘Bn01’, ‘1113-3333-3333-3333’, 10.00, ‘AAA1’);

1. **Use Case 2: Delete Receipt**
2. **User/Actor: Manager**
3. **Steps:**
4. User clicks on “Delete payment” button.
5. The dataset will ask which payment receipt user wants to delete.
6. User clicks on "Confirm” to process the deletion.
7. **SQL Statement:**

DELETE FROM Receipt

WHERE ReceiptID = 1;

1. **Use Case 3: Edit Receipt’s Total**
2. **Actor/User: Customer Service, Manager**
3. **Steps:**
4. User clicks on “Edit Receipt” button.
5. Dataset will be displayed and ask which id the user wants to select.
6. User enters the id and make the change to the receipt.
7. Dataset will be displayed and ask for confirmation.
8. User clicks “Confirm” button.
9. **SQL Statement:**

UPDATE Receipt

SET Total = 15.00

WHERE ReceiptID = 1;

1. **Use Case 4: View Receipt**
2. **Actor/User: Customer Service, Manager**
3. **Steps:**
4. User enters the receipt menu.
5. User clicks on “View Receipts”.
6. Dataset will show the list of receipts.
7. **SQL Statement:**

SELECT \*   
FROM Receipt;

1. **Use Case 5: View Receipt**
2. **Actor/User: Customer**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their receipt.
6. Dataset will show the Sender, Total, and TrackingID of the receipt.
7. **SQL Statement:**

SELECT TrackingID, Sender, Total

FROM Receipt;

1. **Use Case 6: View Receipt and Customer’s Information**
2. **Actor/User: Customer Service, Manager**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their receipt.
6. Dataset will show the Sender, Recepient, Origin, Destination, and CardNum.
7. **SQL Statement:**

SELECT customer.Sender, customer.Recepient, customer.Origin, customer.Destination, receipt.CardNum

FROM customer

JOIN Receipt ON Customer.TrackingID = Receipt.TrackingID;

**Customers’ Information:**

1. **Use Case 1: Add Customer**
2. **User/Actor: Customer Service, Manager**
3. **Steps:**
4. User clicks on “Add Customer” button.
5. The id of the new customer will be generated and displayed.
6. User enters first and last name, phone number, and address.
7. Information will be displayed and waiting for confirmation.
8. User clicks “Confirm” button.
9. **SQL Statement:**

INSERT INTO Customer (TrackingID, Sender, Recepient, Origin, Destination)

VALUES (‘AAA1’, ‘Janna James’, ‘Ryan James’, ‘Houston’, ‘Dallas’);

1. **Use Case 2: Delete Customer**
2. **User/Actor: Customer Service, Manager**
3. **Steps:**
4. User clicks “Delete Customer” button.
5. Dataset will be displayed and ask who user wants to delete.
6. User enters the id and shows the information of the id.
7. Dataset waits for the confirmation of the user.
8. User clicks “Confirm” button.
9. **SQL Statement:**

DELETE FROM Customer

WHERE TrackingID = ‘AAA1’;

1. **Use Case 3: Edit Customer’s Recipient**
2. **User/Actor: Customer Service, Manager**
3. **Steps:**
4. User clicks “Edit Customer” button.
5. Dataset will be displayed and ask who user wants to edit.
6. User selects the id and ask what information user wants to edit.
7. User enters new information and asks for confirmation.
8. User clicks “Confirm” button.
9. **SQL Statement:**

UPDATE Customer

SET Recepient = ‘Lisa James’

WHERE TrackingID = ‘AAA1’;

1. **Use Case 4: View List**
2. **Actor/User: Customer**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their order.
6. Dataset will show the all the information of the order.
7. **SQL Statement:**

SELECT \*

FROM Customer;

1. **Use Case 5: View List**
2. **Actor/User: Customer Service, Manager**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their order.
6. Dataset will show all the information of the order.
7. **SQL Statement:**

SELECT \*

FROM Customer;

**Packages’ Information:**

1. **Use Case 1: Add Package**
2. **User/Actor: Customer Service, Manager**
3. **Steps:**
4. User clicks “Add Package” button.
5. Dataset will generate new id and displayed.
6. User enters package sender and recipient, address, package priority, and weight.
7. The information will be displayed and wait for confirmation.
8. User clicks “Confirm” button.
9. **SQL Statement:**

INSERT INTO Package (PackageID, Type, Weight, Amount, TrackingID)

VALUES (1, ‘Normal’, 5.00, 3.00, ‘AAA2’);

1. **Use Case 2: Delete Package**
2. **User/Actor: Customer Service, Manager**
3. **Steps:**
4. User clicks “Delete Package” button.
5. Dataset will be displayed and ask which package user wants to delete.
6. User enters the id and the dataset displayed and waits for confirmation.
7. User clicks “Confirm” button.
8. **SQL Statement:**

DELETE FROM Package

WHERE PackageID = 1;

1. **Use Case 3: Edit Package’s Price**
2. **User/Actor: Customer Service, Manager**
3. **Steps:**
4. User clicks “Edit Package” button.
5. Dataset will be displayed and ask which package user wants to edit.
6. User enters the id and dataset asks which information user wants to change.
7. User enters new information and wait for confirmation.
8. User clicks “Confirm” button.
9. **SQL Statement:**

UPDATE Package

SET Amount = 5.00

WHERE PackageID = 1;

1. **Use Case 4: View Package**
2. **Actor/User: Customer Service, Manager, Driver**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their packages.
6. Dataset will show all the information of the packages.
7. **SQL Statement:**

SELECT \*

FROM Package;

1. **Use Case 5: View Package**
2. **Actor/User: Customer**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their packages.
6. Dataset will show the Package Type, Weight, and Amount(price) of the packages.
7. **SQL Statement:**

SELECT Type, Weight, Amount

FROM Package;

1. **Use Case 6: View Package with Customer Information**
2. **Actor/User: Customer Service, Manager**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their packages.
6. Dataset will show the PackageType, Weight, Sender, Recipient, and TrackingID of the packages.
7. **SQL Statement:**

SELECT package.PackageType, package.PackageWeight, customer.Sender, customer.Recepient, package.TrackingID

FROM Package

INNER JOIN customer ON customer.TrackingID = package.TrackingID;

1. **Use Case 7: View Package with Weight and ValuePerWeight**
2. **Actor/User: Customer Service, Manager**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their packages.
6. Dataset will show the TrackingID, PackageType, Weight, and ValuePerWeight of the packages.
7. **SQL Statement:**

SELECT package.TrackingID, package.PackageType, package.PackageWeight, packagetype.ValuePerWeight

FROM Package

JOIN packagetype ON package.TrackingID = 'AAA1' AND package.PackageType = packagetype.TypeName

ORDER BY package.PackageType;

1. **Use Case 8: View Package with Receipt Information**
2. **Actor/User: Customer Service, Manager**
3. **Steps:**
4. User enters the search TrackingID menu.
5. User enters the TrackingID to search their packages.
6. Dataset will show the TrackingID, Sender, PackageType, and Weight of the packages.
7. **SQL Statement:**

SELECT receipt.TrackingID, receipt.Sender, package.PackageType, package.PackageWeight

FROM receipt

INNER JOIN package ON receipt.TrackingID = 'AAA1' AND receipt.TrackingID = package.TrackingID

ORDER BY package.PackageType;

**Branches’ Information:**

1. **Use Case 1: Add Branch**
2. **User/Actor: Manager**
3. **Steps:**
4. User clicks “Add Branch” button.
5. Dataset will generate new id and displayed.
6. User enters address, city, and state.
7. The information will be displayed and wait for confirmation.
8. User clicks “Confirm” button.
9. **SQL Statement:**

INSERT INTO Branch (BranchNo, Address, City, State, Profits)

VALUES (‘Bn01’, ‘123 Main St’, ‘Houston’, 0.00);

1. **Use Case 2: Delete Branch**
2. **User/Actor: Manager**
3. **Steps:**
4. User clicks “Delete Branch” button.
5. Dataset will be displayed and asks which branch user wants to delete.
6. User choose the branch and dataset ask for confirmation.
7. User clicks “Confirm” button.
8. **SQL Statement:**

DELETE FROM Branch

WHERE BranchNo = ‘BN01’;

1. **Use Case 3: Edit Branch’s Profits**
2. **User/Actor: Manager**
3. **Steps:**
4. User clicks “Edit Package” button.
5. Dataset will be displayed and ask which branch user wants to edit.
6. User chooses the branch and dataset asks which information user wants to edit.
7. User enters new information and wait for confirmation button.
8. User clicks “Confirm” button.
9. **SQL Statement:**

UPDATE Branch

SET Profits = 100.00

WHERE City = ‘Houston’;

1. **Use Case 4: View Branch List**
2. **Actor/User: Manager**
3. **Steps:**
4. User enters the staff menu.
5. User clicks on “View Branch” button.
6. Dataset will show the list of branches available.
7. **SQL Statement:**

SELECT \*

FROM Branch;

1. **Use Case 5: Calculate Total Profits**
2. **Actor/User: Manager**
3. **Steps:**
4. User enters the staff menu.
5. User clicks on “View Branch List” button.
6. Dataset will show all the information of available branches.
7. User clicks on “Total Profits” button.
8. Dataset will show the total profits of the day.
9. **SQL Statement:**

SELECT SUM(Profits) AS Totals

FROM Branch;

1. **Use Case 6: Calculate Total Order of each branch**
2. **Actor/User: Manager**
3. **Steps:**
4. User enters the staff menu.
5. User clicks on “View Branch List” button.
6. Dataset will show all the information of available branches.
7. User clicks on “Total Order” button.
8. Dataset will show the total order of the day.
9. **SQL Statement:**

SELECT branch.BranchNo, branch.City, COUNT(ReceiptID) AS totalOrder

FROM branch

JOIN receipt ON branch.BranchNo = receipt.BranchNo

GROUP BY BranchNo

ORDER BY BranchNo;

**Truck’s Information:**

1. **Use Case 1: Add Truck**
2. **User/Actor: Manager**
3. **Steps:**
4. User clicks “Add Truck” button.
5. Dataset will generate new id and displayed.
6. User enters plate numbers, start location, and destination.
7. User clicks “Confirm” button.
8. **SQL Statement:**

INSERT INTO Truck (TruckID, PlatNum, DriverID, Origin, HasGoneDeliver)

VALUES (1, ‘WTT-3082’, 2, ‘Houston’, False);

1. **Use Case 2: Delete Truck**
2. **User/Actor: Manager**
3. **Steps:**
4. User clicks “Delete Truck” button.
5. Dataset will be displayed and ask which truck user wants to delete.
6. User chooses the truck and shows information about the truck.
7. User clicks “Confirm” button.
8. **SQL Statement:**

DELETE FROM Truck

WHERE TruckID = 1;

1. **Use Case 3: Edit Truck’s Driver ID**
2. **User/Actor: Manager**
3. **Steps:**
4. User clicks “Edit Truck” button.
5. Dataset will be displayed and ask which truck user wants to edit.
6. User chooses the truck and dataset asks which information user wants to edit.
7. User chooses the option and enters new information.
8. User clicks “Confirm” button.
9. **SQL Statement:**

UPDATE Truck

SET DriverID = 3

WHERE TruckID = 1;

1. **Use Case 4: View Truck List**
2. **User/Actor: Manager, Driver**
3. **Step:**
4. User enters the staff menu.
5. User clicks on “View Truck List” button.
6. Dataset will show all information of truck table.
7. **SQL Statement:**

SELECT \*

FROM Truck;

1. **Use Case 5: Calculate Truck Availability**
2. **Actor/User: Manager**
3. **Steps:**
4. User enters the staff menu.
5. User clicks on “View Truck List” button.
6. Dataset will show the list of trucks available.
7. At the bottom of the page, dataset will show how many trucks are available in the station.
8. **SQL Statement:**

SELECT COUNT(Available) AS Truck Available

FROM Truck

WHERE Origin = ‘Houston’;

1. **Use Case 6: List of Truck Availability and drivers’ name**
2. **Actor/User: Manager**
3. **Steps:**
4. User enters the staff menu.
5. User clicks on “View Truck List” button.
6. Dataset will show the list of trucks, driver’s name, and availability.
7. **SQL Statement:**

SELECT truck.PlatNum, staff.lastName, staff.firstName, truck.Origin, truck.IsAvailable

FROM Truck

LEFT JOIN Staff ON truck.DriverID = staff.StaffID;

1. **Use Case 7: List of Truck Availability and drivers’ name**
2. **Actor/User: Manager**
3. **Steps:**
4. User enters the staff menu.
5. User clicks on “View Truck List” button.
6. Dataset will show the list of trucks, driver’s name, and availability.
7. **SQL Statement:**

SELECT truck.PlatNum, staff.lastName, staff.firstName, truck.Origin, truck.IsAvailable

FROM Truck

LEFT JOIN Staff ON truck.DriverID = staff.StaffID;

1. **Use Case 8: List of Truck Availability and Branch**
2. **Actor/User: Manager**
3. **Steps:**
4. User enters the staff menu.
5. User clicks on “View Truck List” button.
6. Dataset will show the list of trucks, availability and branchNo.
7. **SQL Statement:**

SELECT truck.PlatNum, branch.BranchNo, truck.Origin, truck.IsAvailable

FROM Truck

JOIN branch ON branch.City = truck.Origin

ORDER BY branch.BranchNo;

**Package Type’s Information:**

1. **Use Case 1: Add Package Type**
2. **User/Actor: Manager**
3. **Steps:**
4. Manager enters the package type’s menu.
5. Dataset will ask for manager to prompt new package type.
6. User enters the package type information.
7. User clicks “Confirm” button.
8. **SQL Statement:**

INSERT INTO PackageType (PackTypeID, TypeName, Dimension, MaxWeight, ValuePerWeight)

VALUES

(1, 'normal', '20x20x20',30,5),

1. **Use Case 2: Delete Package Type**
2. **User/Actor: Manager**
3. **Steps:**
4. Manager enters the Package Type’s menu.
5. Dataset asks which package type user wants to delete.
6. User chooses the package type and confirm by presses the “Confirm” button.
7. **SQL Statement:**

DELETE FROM PackageType

WHERE TypeName = ‘Normal’;

1. **Use Case 3: Edit PackageType**
2. **User/Actor: Manager**
3. **Steps:**
4. Manager enters the Package Type’s menu.
5. Dataset will be displayed, and user can choose which information they want to edit.
6. User enters new information and wait for confirmation.
7. User clicks “Confirm” button.
8. **SQL Statement:**

UPDATE PackageType

SET ValuePerWeight = 10

WHERE TypeName = ‘Express’;

1. **Use Case 4: View PackageType**
2. **Actor/User: Manager, Customer Service**
3. **Steps:**
4. User enters the Package Type’s menu.
5. User clicks on “View Package Type” button.
6. Dataset will show the list of package types available.
7. **SQL Statement:**

SELECT \*

FROM PackageType;

1. **Test Plan:**

**CREATE TABLE:**

**A computer screen shot of a black background

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A computer screen shot of text

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screenshot of a computer code

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**Staff Table:**

**A computer screen with text and numbers

Description automatically generated**

****

**A black background with white text

Description automatically generated**

**A black background with white text

Description automatically generated**

**A black background with white text

Description automatically generated**

**Branch Table:**

**A black background with white text

Description automatically generated**

****

**A black background with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**Truck Table:**

**A black background with white text

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**A black screen with white text

Description automatically generated**

****

**A black screen with white text

Description automatically generated**

**PackageType Table:**

**A screen shot of a computer program

Description automatically generated**

****

**A black background with white text

Description automatically generated**

****

**Package Table:**

**A black background with white text and numbers

Description automatically generated**

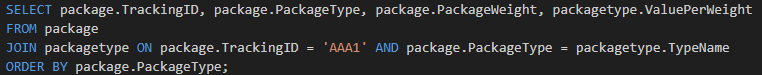
****

**A black background with white text

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

****

**Receipt Table:**

**A screenshot of a computer

Description automatically generated**

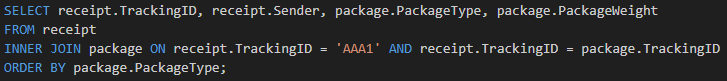
****

**A black background with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

****

**Customer Table:**

**A black background with white text

Description automatically generated**

****

**A black background with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

****

1. **Conclusion:**

With the basic structure of the database completed, it is known that the database has many Seconds Normalize Form and could go to Third Normalize Form with more optimize datasets. Foreign keys are important to link datasets together for database to work, although it is not encouraged to have more than two foreign keys. What I learned from this project is every domain should take consideration when making and finalize a dataset because it could increase redundancy and integrity if one data makes a change, a whole row of data could change with it.

1. **Reference:**

“W3schools.Com.” *W3Schools Online Web Tutorials*, www.w3schools.com/sql/default.asp. Accessed 10 Aug. 2024.