SCHOOL OF COMPUTER SCIENCES
UNIVERSITI SAINS MALAYSIA

CMT221/CMM222 : Database Organization and Design
Semester II, Academic Session : 2023/2024

System Implementation:

University Canteen Food Waste Database Management System

Prepared by:
Group 2

Supervisor: *Dr Azleena Binti Mohammad Kassim*

Names	Matric No	USM Email Address	Module	Role	School
Leong Ye Xian	161491	leongyexian@student.usm.my	Analytical Module	Leader	Physics
Haleim Shah Bin Haja Mohideen	163698	haleimshah@student.usm.my	Financial Assessment Module	Member	Physics
Santtosh A/L Muniyandy	159193	santtosh3476@student.usm.my	Food Waste Data Collection Module	Member	Physics
Loh Jeng Man Angellyn	161161	angellynloh@student.usm.my	Inventory Management Module	Member	Physics

Date of submission
1 July 2024

TABLE OF CONTENTS

1.0 INTRODUCTION

Food waste is a global issue, with its effects being seen in social, economic, and environmental issues as a contributing factor. According to The World Counts, roughly a third of the world's food is wasted, which is about 1.3 billion tons a year. Evaluating its social impact reveals that the wasted food could be enough to feed over 2 billion people (2022, Food and Agriculture Organization of the United Nations).

One of the contributions to the global food waste crisis is university canteens. A study on the three canteens in Taiyuan University of Technology found the total amount of food waste by the three canteens with 22,000 students was 246.75 tons per annum, which is equivalent to carbon footprint of 539.28 tons of CO_2 (Li, Li, Wang, Jin, 2021). Our team proposes to implement a food waste Database Management System (DBMS) which is expected to raise awareness on a university canteens' food waste generation.

The university of our focus is Universiti Sains Malaysia. With its abundance of canteens in every hostel, the DBMS is expected to work well in culling excess food waste generation by bringing awareness to the canteen management teams. Policies such as sharing ingredients near expiry date can then be implemented to reduce the generation and the costs related to it.

1.1 Project Motivation

The motivation of this project revolves around the heart of the 17 Sustainable Development Goals (SDGs), especially **SDG2** (Zero Hunger) and **SDG3** (Good Health and Wellbeing) as our primary focus is reducing food waste so that excess food can be used to feed those in need. We also promote **SDG12** (Responsible Consumption and Production) for food since canteen stalls will produce as much food as they can sell instead of in excess. Reducing food waste also reduces the carbon footprint it causes, which reduces greenhouse gases and contributes to **SDG13** (Climate Action).

1.2 Benefits of The Database System

There is currently a need for a system to track the amount of food waste generated by universities. Universities, being a place where people gather, must have canteens and food stalls, and whenever these places are unable to sell their food in time, food waste is generated. Our DBMS allows food inventory and its waste data to be recorded and managed by canteen operators, then evaluated by the canteen management team. Insights from the data can assist the canteen management in making informed decisions and implementing effective policies. Reducing production and wastage also increases profit margins by reducing the related costs. The DBMS also records finance related to food production and wastes, which increases transparency.

1.3 Modules

The proposed University Canteen DBMS will have 4 modules.

- **Food Waste Collection Module** by Santtosh A/L Muniyandy.

    This module is created to track and collect the real-time data of food waste and store this data in the database. The data collected includes type of food, quantity of waste, date of waste record and reason of wastage. This module mainly revolves three entities which are FOOD, STALLS and FOOD_WASTE. FOOD entity able to catalog various types of food offered by the stalls which ease to identify from the waste. STALLS entity consists of different stalls which links to respective waste generated. FOOD_WASTE is the core of this module as it accumulates the data regarding quantity of food waste accumulated.

    By developing this module, we can detect which stall can be held accountable for an increase in food waste and implement necessary disciplinary actions. Additionally, we can help the stalls in a canteen to develop a proper tracking system as to how much food they need to produce and the number of resources they require, which leads to a more sustainable and cost-effective operation for the stalls. This module acts as the heart of this project as it links all other modules such as those for saving costs, those for discovering trends to prevent wastage and those for providing a well-planned inventory management.

- **Inventory Management System Module** by Loh Jeng Man Angellyn

This module is to track inventory levels of food items and ingredients to prevent overstocking and that food storage items are used efficiently and the accountability in managing food resources are handled responsibly. The inventory management module generates valuable data and insights that can inform decision-making processes related to purchasing, menu planning, sales promotion, and waste reduction strategies. By optimizing inventory levels and reducing waste, we can save costs associated with purchasing and disposing of excess food items.

The general idea of this module is to implement a multi-table database scheme with relationships comprising entities mainly related to FOOD STORAGE, SUPPLIER, and FOOD STORAGE ROOM. The Food Storage table will monitor expiry dates of the perishable item. The food will be categorized based on the *NOVA system* that puts food into four groups based on how processed it is. We will also maintain records of food suppliers, including contact information and delivery schedules. This is to monitor supplier performance and quality control measures to minimize waste in the supply chain. The Food Storage Room table stores important data regarding its location details, ownership details, and capacity to effectively track inventory storage locations, monitor storage conditions, and optimize inventory management and stock levels.

In addition, we are also going to work on implementing a system to track purchase orders placed with suppliers. This can help in managing inventory costs and ensuring accurate accounting, which integrates with the Financial Assessment System in our work.

- **Financial Assessment Module** by Haleim Shah Bin Haja Mohideen

The module categorizes expenses, predicts budgets, and estimates losses due to food waste. It also focuses on integrating all other modules to create a detailed record of expenditures, budgets, and avoidable losses, ensuring a high level of transparency and accountability.

The financial module categorizes all expenses into three primary categories: Collection, Processing, and Administration, with each category further detailed into subcategories to ensure transparency and precision in tracking and managing expenses. The fiscal budget for the organization will be determined by considering various revenue streams, employing a comprehensive analysis to estimate annual expenses accurately.

Collection operations require meticulous planning and execution, involving multiple employees and regular maintenance to ensure smooth operations. A dedicated storage facility is essential for processing food storage, demanding effective sorting and multiple employees to maintain efficiency. Additionally, administrative roles are critical in managing these operations, with expenses tracked and roles often overlapping among employees.

To predict the financial budget for the next fiscal year, the system will leverage data from the current annual budget, expenses, and revenue, utilizing linear regression analysis. This predictive approach will provide a data-driven estimation, aiding in strategic planning and resource management. Furthermore, the system will estimate yearly losses due to food waste, considering the type and quantity of waste and current expenses. This estimation will highlight areas for operational cost reduction and encourage efforts to minimize food waste, aligning with the overarching goal of sustainable waste management.

- **Analytical Module** by Leong Ye Xian

  This module contains tables for storing and analyzing statistics from stored data. It aims to reveal hidden trends in food waste generation for effective planning. The analytical module generates valuable insight that can enhance decision-making processes. By analyzing past records, it becomes possible to predict changes soon such that appropriate actions can be taken to minimize food waste.

  The general idea of this module is to implement a multi-table database scheme with relationships comprising entities related to Statistics, Prediction, Difference, Trend and Carbon Emissions tables. The Statistics table will process and store data from food waste records in the form of statistical descriptions. For the Prediction table, predictions are made with linear regression functions fitted using past record data. For the Difference table, the difference is shown on a month-to-month basis whereas the Trend table shows the strength of the trend. Finally, a Carbon Emissions table is used to calculate the amount of carbon dioxide released by the different types of food waste which aims to reveal the carbon food print and help in the efforts to reach zero net carbon emissions.

2.0 USER REQUIREMENTS AND BUSINESS RULES

2.1 Food Waste Collection Module

User Requirements	Business Rules
The database system must ensure that a stall owner or staffs able to list and update their food menu with various items while also ensuring that not more than two stalls can sell similar food types. E.g.: If a stall sells 'Chicken Burger', other stalls are not allowed to add or have that same food item in their menu.	A specific food type particularly can be sold by only one stall. A respective stall could have many diverse food items in its menu.
Every stall must be able to insert multiple entries of food where that waste entries must be only related to that specific stall. E.g.: A stall must have their own waste container where the food items sold respective from that stall must be disposed there to enter waste records.	Each stall can have multiple food wastage entry. Each food wastage entry can only be entry by one stall.
Database system able facilitate that a single waste links to multiple food items that had been disposed by consumers. E.g.: In a waste record it can contains multiple spoiled food such as fish, chicken, tomato, etc. While fish could also in the waste record multiple time.	A food waste entry can be referenced by multiple specific food items. A food item can reference multiple waste entries.
A stall owner or staffs must be to give valid reasons for each waste entries such as spoiled food or food is unable to be finished by consumers. E.g.: When recording wastage of 'onions', it could be for overnumbered food preparation resources used for cooking reasons.	A food waste entry can be factored by multiple reasons for food wastage. A particular reason food wastage factor multiple food waste entry.
Database system must be able to different categorize food items such as meat, fruits, vegetables, vegan, non-vegan, etc. E.g.: The staff owner must categorize	A food item can belong to only one category. Each category can include many food items.

their food items, for example 'Vegan Burger' in the category of 'Vegan' category.	
Only one owner can have ownership and manages to a single unique stall. E.g.; A stall owner named 'Adam' able to register one stall section in the cafeteria.	A stall ownership must be associated to one stall owner. A stall owner can only own one stall.
In a canteen or cafeteria, there can be many stalls in different location or that canteen can act as a single restaurant itself. E.g.: A canteen can have 'Burger Shop' and 'Western Food Shop' while another canteen can sell all types of food item as a single entity.	A canteen may have zero or more cafes. A café can be placed independently or only in one canteen.

Table 1: User Requirements and Business Rules for Food Waste Collection Module

2.2 Inventory Management System Module

User Requirements	Business Rules
All food storages will be categorized based on the *NOVA system* that puts food into four groups based on how processed it is. E.g., Unprocessed Food, Processed Culinary Ingredient, Processed Food, and Ultra-processed Food. Users can search for specific items within the inventory, based on their category.	One food storage can be under only one category. One category can have one or many food storages.
Food storage items come in a batch. E.g., a box of instant noodles, a stack of eggs and 5 bottles of soy sauce may come as one batch. Every batch can be traced to its purchase order.	One batch can have one or many food storages. One food storage comes in only one batch.
When a batch of food are ordered and purchased, the order details are recorded in a purchase order. The inventory data will be added into Food Storage once the Purchase Orders has been placed, but the Tracking_Order_Status is required as *pending*. When the Food Storage is delivered, Tracking_Order_Status is set as *fulfilled*, with its Date_of_Delivery updated.	One batch must only exist in one purchase order. One purchase order must have only one batch.
One purchase order may record a batch of food from one supplier only. Users may restock by repurchasing from the same supplier. Users can record information for suppliers early before placing an order.	One purchase order can only be received by only one supplier. One supplier can receive zero or many purchase orders.
Food storage items are stored in shelves. Each shelf is labelled with a unique series of ID number. Users must make sure they don't overstock by controlling the quantity of food storages below the shelf's capacity. There will be different types of shelves that stores the food items, such as freezers, shelf racks, fridges and cupboards.	One food storage can be stored in only one shelf. One shelf can store zero or many food storages. Food storage quantity must range between 0 to the maximum capacity of its shelf.

Food storage room contains many shelves that stores the food storage items. The layout of the shelves within the food storage room should be observed before adding them as data into the database.	One food storage room can have one or many shelves. One shelf can only be in one food storage room.
A store supervisor is assigned to manage only one food storage room. This is to ensure focused oversight and accountability for inventory control, organization, and compliance with food safety standards.	One food storage room is managed by only one supervisor. One supervisor can only manage one food storage room.
One stall/restaurant can only have one food storage room. This policy helps maintain food safety protocols and facilitate easier monitoring of storage conditions and inventory levels.	One food storage room belongs to only one stall/restaurant. One stall/restaurant can only own one food storage room.

Table 2: User Requirements and Business Rules for Inventory Management System Module

2.3 Financial Assessment Module

User Requirements	Business Rule
User must be able to add new expenses, with a unique reason, with said submitter's name. Each expense must have a date, details of said reason, and the person submitting it to ensure integrity.	Individual expenses and with said reasons are always unique.
The fiscal budget is determined based on revenue and predicted budget. Various revenues contribute to the annual budget for estimated expenses.	Multiple revenue streams must be considered to determine the annual budget for estimated expenses.
Users may view or add new revenue entries provided, said date,source,amount and reference information. Allowing all revenue to have a trail and allowable to edit said entry.	Each revenue entry must have a unique identifier, all revenue entries require a date, amount, and a reference with the source. Ensuring a detailed record of accomplishment of all revenue streams.
Users can view the losses caused due to food waste based on stalls, on an annual basis.	
The financial budget prediction considers the current annual budget, expenses, and revenue. Predicting the budget for the next fiscal year requires data on predicted mass of food waste and a few other factors	Budget prediction must use data from the current year, including expenses and revenue.
Yearly losses due to food waste being thrown out should be estimated to reduce operational costs. Loss estimation should consider the type and quantity of food waste and current expenses.	Annual losses due to food waste must be estimated based on food type, quantity, and current expenses. Efforts should be made to minimize operational costs by reducing food waste.

Table 3: User Requirements and Business Rules for Financial Assessment Module

2.4 Analytical Module

User Requirements	Business Rules
The statistics from the past records of food waste entries should be presented in a structured, tabular form to facilitate the objective understanding of the current food waste situation. The end user can see the statistical descriptions of past data as records in a table and be able to use the data directly for analytical purposes.	One statistic is related to many types of waste at different times. One type of waste at a time is related to one statistic.
Using statistics from past records, the system will automatically produce a linear regression function used to calculate the predicted values for the next month. The user can see the predicted values for the next month based on the different functions, which will provide insights to how much the values might vary.	One statistic is related to one or many predictions. One prediction is related to one or many statistics.
From the statistics, the end user can see the monthly differences and whether it is increasing or decreasing. This will give insight to whether a newly implemented policy is working or if the food waste crisis is worsening.	One statistic is related to one or many monthly differences. One monthly difference is related to one or many statistics.
The monthly difference affects the trend of food waste generated based on their type. The end user can easily determine the strength of the trend based on Relative Strength Index (RSI) and make a decision whether to implement a radical policy or not.	One monthly difference is related to one trend. One trend is related to one monthly difference.
Every type of food waste emits an approximate amount of carbon dioxide. Once the data dictionary has been set up, the carbon dioxide released by each type of food waste can be approximated. This will help monitor the monthly carbon footprint due to the monthly food waste.	One type of waste at a time is related to one carbon emission. One carbon emission is related to one or many type of waste at a time.

Table 4: User Requirements and Business Rules for Analytical Mo

3 ENTITY RELATIONSHIP MODELLING

The Crow's Foot notation is used for the entity relationship diagram (ERD). In addition, the Extended ERD (EERD) has been utilized to present the structure of the database more accurately.

3.1 Food Waste Collection Module

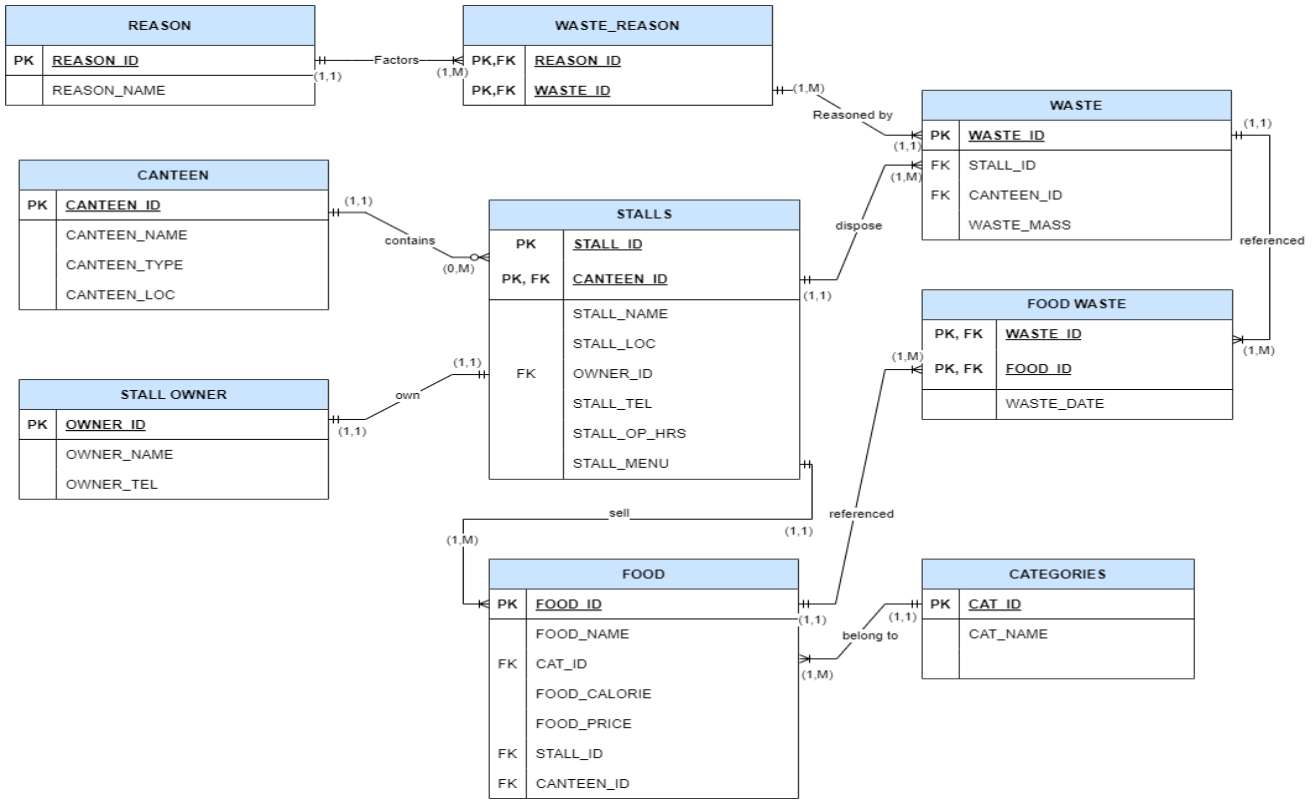


Figure 1: Food Waste Collection Module ERD

ERD Explanation

Entities:

Independent Entities	• CANTEEN • STALL OWNER • FOOD • CATEGORIES • WASTE
Dependent Entities	• STALL MENU • STALLS • FOOD WASTE
Associative Entities	• FOOD WASTE • WASTE REASON

Attributes:

1. STALLS entity has its own STALL_ID and CANTEEN_ID to represent which canteen does the stalls located at. Other attributes are STALL_NAME, STALL_LOC, OWNER_ID, STALL_TEL, STALL_OP_HOURS and STALL_MENU.
2. CANTEEN entity has its own CANTEEN_ID and other attributes are CANTEEN_NAME, CANTEEN_TYPE and CANTEEN_LOC
3. STALL OWNER entity has OWNER_ID, OWNER_NAME and OWNER_TEL.
4. FOOD entity has STALL_ID, CANTEEN_ID, FOOD_ID, FOOD_NAME, CAT_ID to represent which category does that food item falls in, FOOD_CALORIE and FOOD_PRICE.
5. CATEGORIES entity has CAT_ID and CAT_NAME.
6. FOOD WASTE act as an entity bridge between CATEGORIES and WASTE, in which this bridge entity has WASTE_ID, FOOD_ID, and WASTE_DATE.
7. WASTE has WASTE_ID, STALL_ID and CANTEEN_ID to represent which waste record lies for which stall, and WASTE_MASS in the measurement unit of kg. WASTE_DATE must be in standard date format of YY/MM/DD.
8. REASON has REASON_ID and REASON_NAME.
9. WASTE_REASON act as a bridge entity between REASON and WASTE as they have many-to-many relationship.

Relationships:

1. The relationship between CANTEEN and STALLS entity is zero-to-many where STALLS is optional.
2. The relationship between STALL OWNER and STALLS entity is one-to-one.
3. The relationship between STALLS and FOOD is one-to-many.
4. The relationship between CATEGORIES and FOOD is one-to-many.
5. The relationship between FOOD and FOOD WASTE is one-to-many.
6. The relationship between WASTE and FOOD WASTE is one-to-many.
7. The relationship between STALLS and WASTE is one-to-many.
8. The relationship between REASON and WASTE_REASON is one-to-many.
9. The relationship between WASTE and WASTE_REASON is one-to-many.

Constraints:

1. CANTEEN entity has primary key of CANTEEN_ID.
2. STALLS entity has composite primary key of STALL_ID and CANTEEN_ID where CANTEEN_ID also act as foreign key as it is taken from CANTEEN_ID. Other foreign keys are OWNER_ID from STALL OWNER.
3. STALL OWNER entity has OWNER_ID as primary key.
4. FOOD entity has FOOD_ID as its primary key, CAT_ID as foreign key from CATEGORIES entity and CANTEEN_ID with STALL_ID as foreign key from STALLS entity.
5. CATEGORIES has CAT_ID as primary key.
6. FOOD WASTE has both primary keys and foreign keys of WASTE_ID and FOOD_ID.
7. WASTE entity has WASTE_ID as primary key and STALL_ID with CANTEEN_ID as foreign key from STALLS entity.
8. REASON entity has REASON_ID as primary key.
9. WASTE_REASON entity has composite primary key of WASTE_ID and REASON_ID.

3.2 Inventory Management System Module

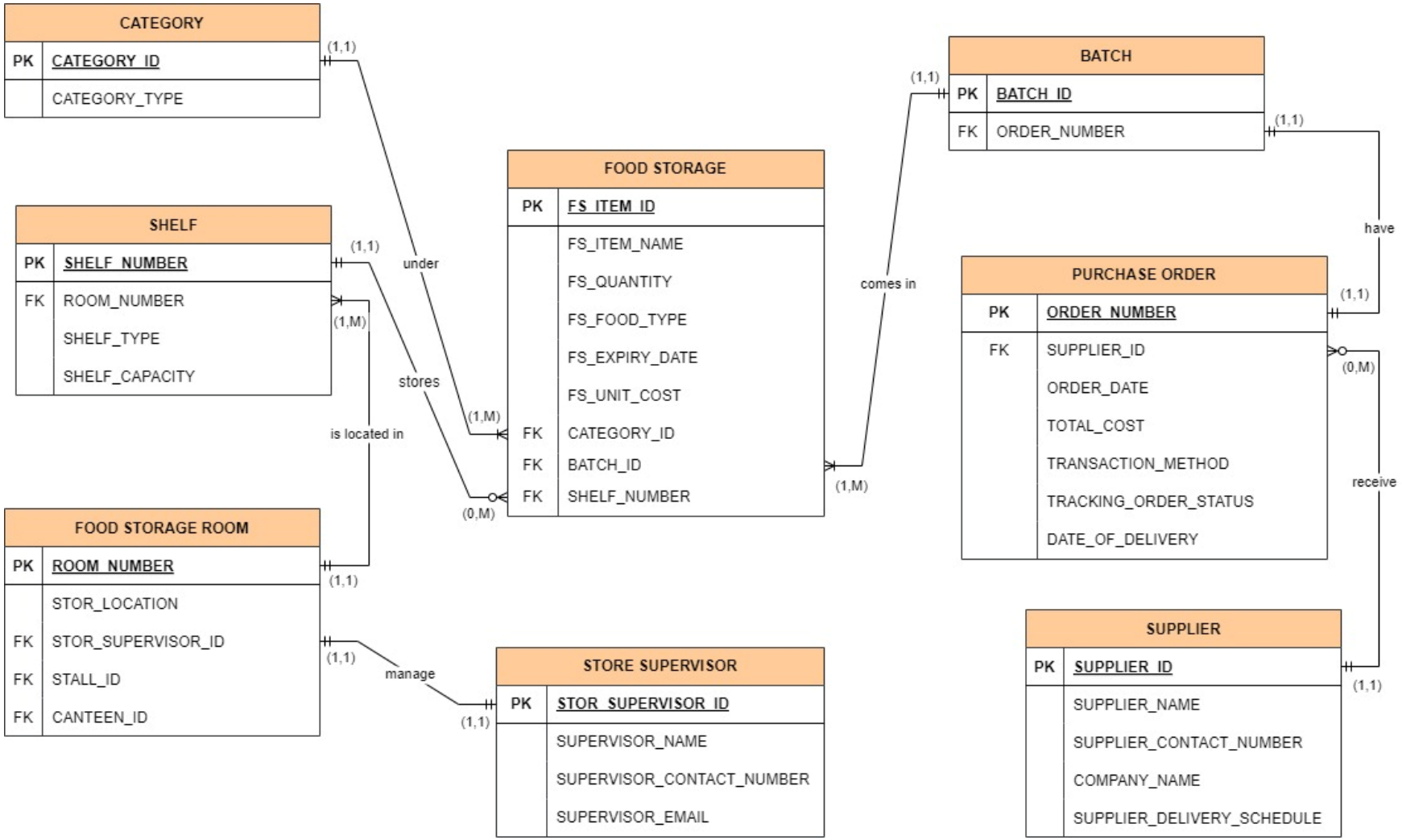


Figure 2: Inventory Management System Module ERD

ERD Explanation

Entities:

Independent Entities	• STORE SUPERVISOR • CATEGORY • SUPPLIER
Dependent Entities	• FOOD STORAGE • SHELF • FOOD STORAGE ROOM • BATCH • PURCHASE ORDER

Attributes:

1. FOOD STORAGE entity has their own unique FS_ITEM_ID. Each Food Storage has an attribute of FS_ITEM_NAME, FS_QUANTITY, FS_FOOD_TYPE, FS_EXPIRY_DATE, FS_UNIT_COST and CATEGORY_ID, BATCH_ID, and SHELF_NUMBER.
2. CATEGORY entity has a unique CATEGORY_ID. Its attribute is CATEGORY_TYPE (e.g., Unprocessed Food, Processed Culinary Ingredient, Processed Food, and Ultra-processed Food).
3. Food Storage comes in batches, and each BATCH entity has a unique BATCH_ID that tracks the PURCHASE ORDER.
4. Each FOOD STORAGE is stored in shelves that comes with a unique SHELF_NUMBER, and the SHELF_NUMBER links to the FOOD STORAGE ROOM. The attributes are SHELF_TYPE (e.g., Shelf Rack, Freezer, Fridge), and SHELF_CAPACITY.
5. Every FOOD STORAGE ROOM entity has a unique ROOM_NUMBER, that links to their respective stall or restaurant. Every FOOD STORAGE ROOM entity has an attribute of STOR_LOCATION, STOR_SUPERVISOR_ID, CANTEEN_ID and STALL_ID. CANTEEN_ID and STALL_ID is a composite foreign key that links to STALL entity from the Food Waste Collection Module.
6. Every STORE SUPERVISOR entity has a unique STOR_SUPERVISOR_ID, and has an attribute of SUPERVISOR_NAME, SUPERVISOR_CONTACT_NUMBER, and SUPERVISOR_EMAIL.
7. Each PURCHASE ORDER entity has a unique ORDER_NUMBER. The PURCHASE ORDER entity has an attribute of ORDER_DATE, TOTAL_COST, TRANSACTION_METHOD, TRACKING_ORDER_STATUS (e.g., Pending, Fulfilled), and DATE_OF_DELIVERY. SUPPLIER_ID is a foreign key that links to the SUPPLIER table.

8. Each SUPPLIER entity has a unique SUPPLIER_ID, and has an attribute of SUPPLIER_NAME, SUPPLIER_CONTACT_NUMBER, COMPANY_NAME, and SUPPLIER_DELIVERY_SCHEDULE.

Relationships:

1. The relationship between FOOD STORAGE entity and CATEGORY entity is many-to-one.
2. The relationship between FOOD STORAGE entity and BATCH entity is many-to-one.
3. The relationship between FOOD STORAGE entity and SHELF entity is many-to-one. FOOD STORAGE entity is optional to SHELF entity.
4. The relationship between SHELF entity and FOOD STORAGE ROOM entity is many-to-one.
5. The relationship between FOOD STORAGE ROOM entity and STORE SUPERVISOR entity is one-to-one.
6. The relationship between BATCH entity and PURCHASE ORDER entity is one-to-one.
7. The relationship between PURCHASE ORDER entity and SUPPLIER entity is many-to-one. PURCHASE ORDER entity is optional to the SUPPLIER entity.
8. The relationship between FOOD STORAGE ROOM entity and STALLS entity is one-to-one.

Constraints:

1. FOOD STORAGE entity has a primary key, FS_ITEM_ID. It also contains CATEGORY_ID, BATCH_ID and SHELF_NUMBER as foreign key.
2. FS_QUANTITY must range between 0 to the maximum capacity of its shelf.
3. CATEGORY entity has a primary key, CATEGORY_ID.
4. BATCH entity has a primary key, BATCH_ID and a foreign key, ORDER_NUMBER.
5. SHELF entity has a primary key, SHELF_NUMBER and a foreign key, ROOM_NUMBER.
6. FOOD STORAGE ROOM entity has a primary key, ROOM_NUMBER and contains STOR_SUPERVISOR_ID as a foreign key and STALL_ID, CANTEEN_ID as composite foreign key.
7. STORE SUPERVISOR entity has a primary key, STOR_SUPERVISOR_ID.
8. PURCHASE ORDER entity has ORDER_NUMBER as a primary key and SUPPLIER_ID as foreign key.
9. SUPPLIER entity has a primary key, SUPPLIER_ID.
10. FS_UNIT_COST, SHELF_CAPACITY, and TOTAL_COST cannot be a negative number.

3.3 Financial Assessment Module

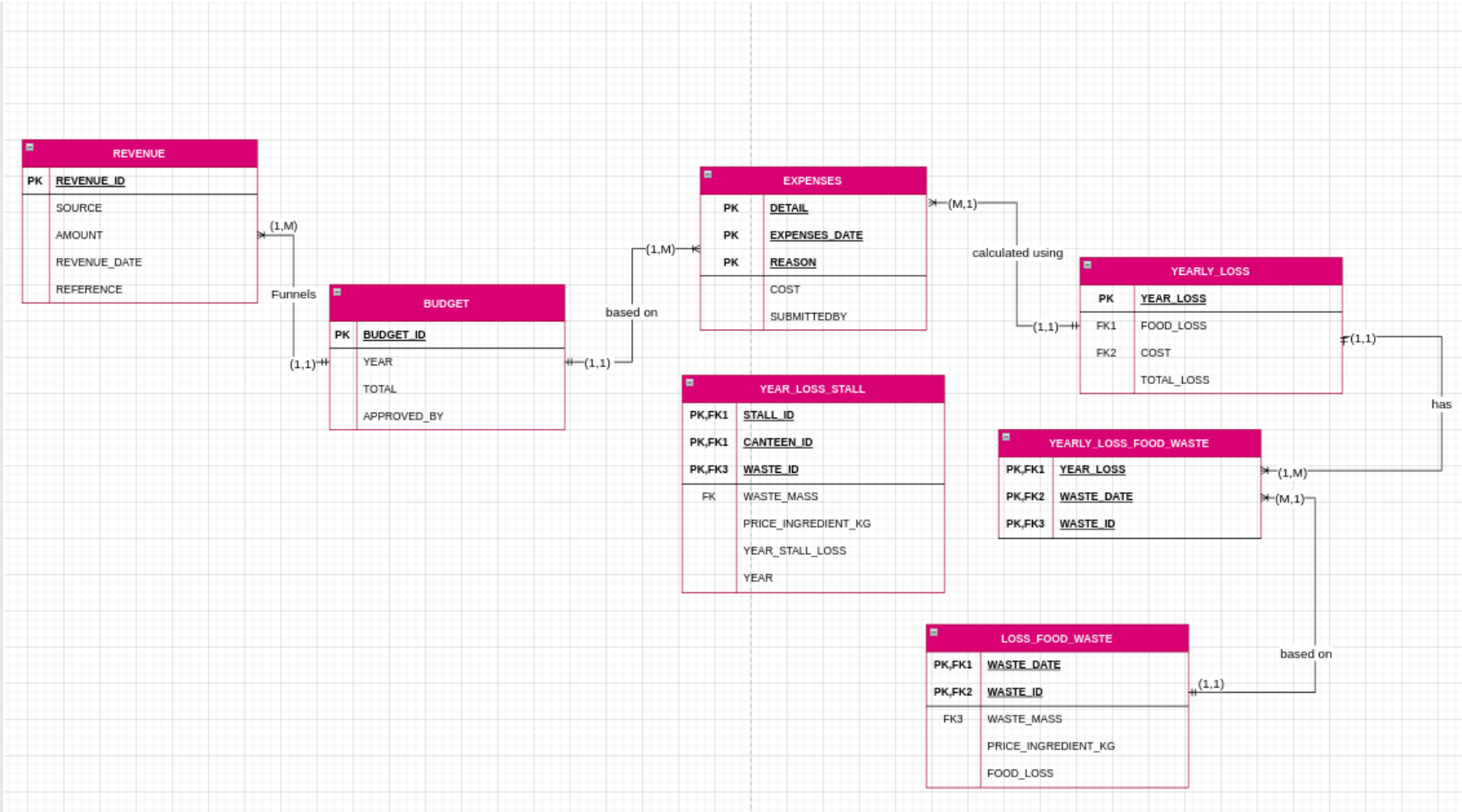


Figure 3: Financial Assessment Module ERD

ERD Explanation

Entities:

Independent Entities	• REVENUE • EXPENSES
Associative Entities	• YEARLY_LOSS_FOOD_WASTE
Supertype Entities	• BUDGET • YEARLY_LOSS • LOSS_FOOD_WASTE • YEAR_LOSS_STALL

Entities:

1. The financial module contains REVENUE has their own unique ID that is REVENUE_ID, SOURCE, AMOUNT, REVENUE_DATE and REFERENCE. This is where all income sources and their information are kept.
2. The financial module contains BUDGET, which has its own unique ID, BUDGET_ID, and includes attributes such as YEAR, TOTAL, and APPROVED_BY. This ensures each budget can be referred to who approved it, bringing transparency into the financial module.
3. The financial module contains EXPENSES, which have DETAILS, REASON and COST also requiring SUBMITTEDBY. This helps users to add their expenses with a reason and detailed reasoning ensuring more transparency for the usage of finances, with the addition of SUBMITTEDBY,to enforce integrity.
4. The financial module contains YEARLY_LOSS, which has its own unique ID, YEAR_LOSS, and includes attributes such as FOOD_LOSS, COST, and TOTAL_LOSS. This entity tracks the annual financial losses due to food waste.
5. The financial module contains YEARLY_LOSS_FOOD_WASTE, which has a composite unique ID consisting of YEAR_LOSS, WASTE_DATE, and WASTE_ID. This entity details the connection between yearly losses and specific instances of food waste.
6. The financial module contains YEAR_LOSS_STALL, which has a composite unique ID consisting of STALL_ID, WASTE_ID and CANTEEN_ID, and includes attributes such as WASTE_MASS, PRICE_INGREDIENT_KG, and YEAR_STALL_LOSS. This entity tracks the financial losses related to food waste at specific stalls within the canteen over a year.
7. The financial module contains LOSS_FOOD_WASTE, which has a composite unique ID consisting of WASTE_DATE and WASTE_ID, also includes attributes such as WASTE_MASS,

PRICE_INGREDIENT_KG, and FOOD_LOSS. This entity records the details of food waste occurrences and their associated costs.

Relationships:

1. The relationship between REVENUE entity and BUDGET is one to one
2. The relationship between BUDGET and EXPENSES is one to many.
3. The relationship between YEARLY_LOSS and YEARLY_LOSS_FOOD_WASTE is one to many
4. The relationship between YEARLY_LOSS_FOOD_WASTE and YEARLY_LOSS is many to one.
5. The relationship between LOSS_FOOD_WASTE and YEARLY_LOSS_FOOD_WASTE is one to many.
6. . The relationship between YEAR_LOSS_STALL and LOSS_FOOD_WASTE is many to one.
7. The relationship between EXPENSES and BUDGET is one to many.
8. The relationship between YEAR_LOSS_STALL and WASTE is one to many.

Constraints:

1. The REVENUE_ID is the primary key, which must be unique and not null.
2. The AMOUNT must be a positive number, and REVENUE_DATE must follow a specific date format.
3. The REVENUE_DATE must follow a specific date format (e.g., YYYY-MM-DD).
4. The SOURCE must not be null.
5. The REFERENCE must not be null.
6. The BUDGET_ID is the primary key, which must be unique and not null.
7. The YEAR must follow a valid year format (YYYY).
8. The TOTAL must be a positive number.
9. The APPROVED_BY must not be null.
10. The BUDGET_ID is a foreign key that references the BUDGET table and must not be null.
11. The TOTAL, AMOUNT, and COST must be positive numbers.
12. The DETAIL and REASON together form the primary key, which must be unique and not null.
13. The COST must be a positive number.
14. The SUBMITTEDBY must not be null.
15. he YEAR_LOSS is the primary key, which must be unique and not null.
16. The YEAR must follow a valid year format (YYYY).
17. The COST and TOTAL_LOSS must be positive numbers.
18. The YEAR_LOSS is a foreign key that references the YEARLY_LOSS table and must not be null.
19. The WASTE_DATE must follow a specific date format DD/MM/YY.
20. The WASTE_ID is a foreign key that must not be null.
21. The combination of WASTE_DATE and WASTE_ID forms the primary key, which must be unique and not null.
22. The WASTE_MASS and PRICE_INGREDIENT_KG must be positive numbers.
23. The FOOD_LOSS must not be null.
24. The combination of STALL_ID and YEAR_LOSS forms the primary key, which must be unique and not null.
25. The CANTEEN_ID and WASTE_ID are foreign keys that must not be null.
26. The WASTE_MASS and PRICE_INGREDIENT_KG must be positive numbers.
27. The YEAR_STALL_LOSS must not be null.
28. THE EXPENSES_DATE FOLLOWS DD/MM/YY

3.4 Analytical Module

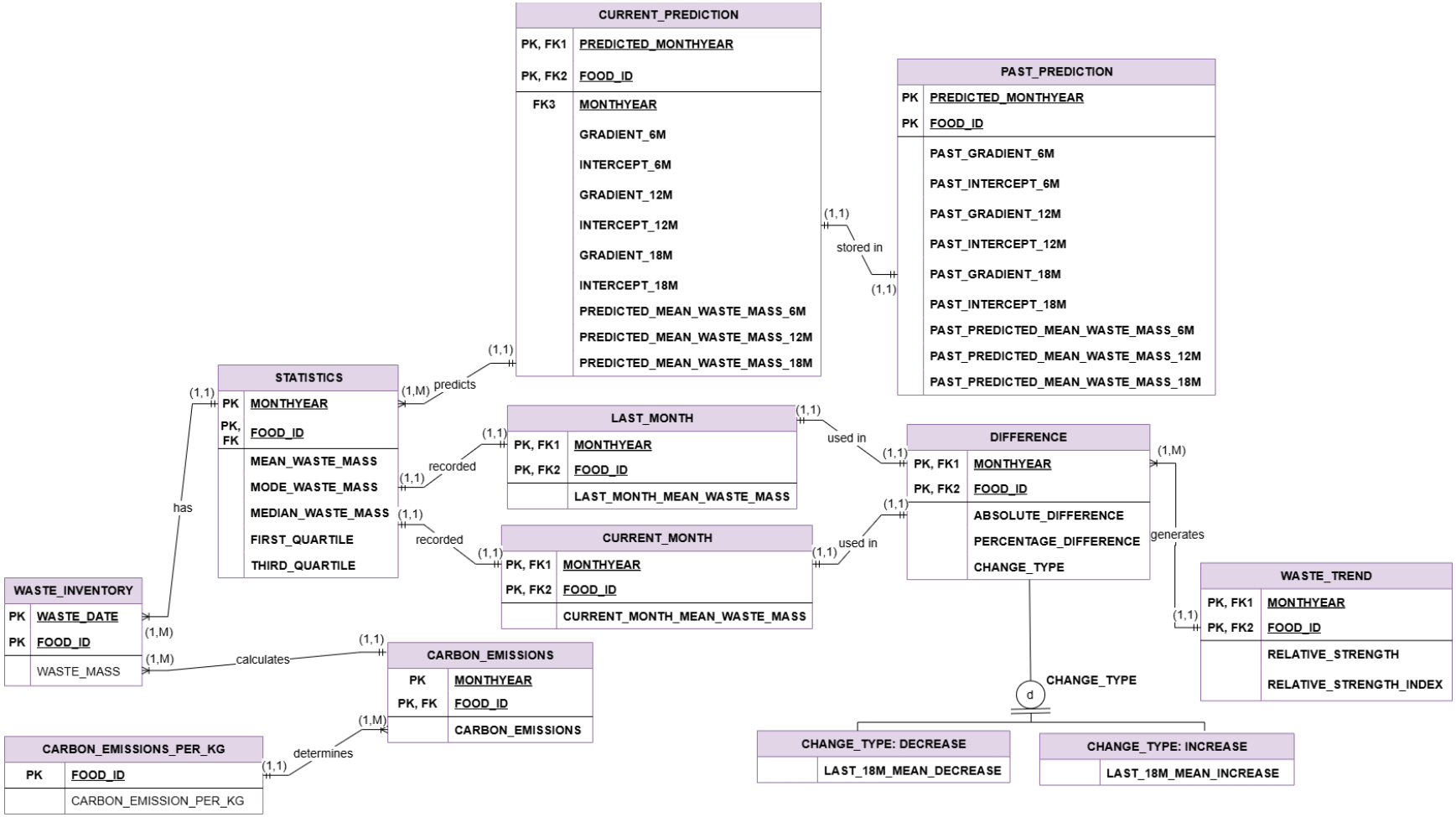


Figure 4: Analytical Module ERD

ERD Explanation

Entities:

Independent Entities	• WASTE_INVENTORY • CARBON_EMISSIONS_PER_KG
Dependent Entities	• STATISTICS • CURRENT_PREDICTION • PAST_PREDICTION • WASTE_TREND • CARBON EMISSIONS • DIFFERENCE
Associative Entities	• LAST_MONTH • CURRENT_MONTH
Subtype Entities	• CHANGE_TYPE: INCREASE • CHANGE_TYPE: DECREASE

Attributes:

1. WASTE_DATE and FOOD_ID index the type of food waste and time it was recorded.
2. MONTHYEAR attribute is formed by taking only the month and year for a group of WASTE_DATE.
3. The attributes MEAN_WASTE_MASS, MODE_WASTE_MASS, MEDIAN_WASTE_MASS, FIRST_QUARTILE and THIRD_QUARTILE represent the statistical description of the waste mass of a food type for a particular month and year.
4. The attributes GRADIENT_6M, INTERCEPT_6M, GRADIENT_12M, INTERCEPT_12M, GRADIENT_18M and INTERCEPT_18M are the parameters fitted based on 6 months (6M), 12 months (12M) and 18 months (18M) worth of data, respectively.
5. PREDICTED_MEAN_WASTE_MASS_6M, PREDICTED_MEAN_WASTE_MASS_12M and PREDICTED_MEAN_WASTE_MASS_18M are predictions from linear regression functions based on the parameters {GRADIENT_6M, INTERCEPT_6M}, {GRADIENT_12M, INTERCEPT_12M}, {GRADIENT_18M, INTERCEPT_18M} respectively.
6. All PAST_PREDICTION attributes are records of all the records that have ever existed in the CURRENT_PREDICTION table.
7. LAST_MONTH_MEAN_WASTE_MASS records the amount of every type of food waste from the last month, relative to the current month. This means all datapoints have an index of n-1, where n is the index for the latest entry. The first value by default is 0.

8. CURRENT_MONTH_MEAN_WASTE_MASS records the amount of food waste for every type from the current month. This means all datapoints have an index of n, where n is the index for the latest entry.
9. ABSOLUTE_DIFFERENCE takes the difference between the CURRENT_MONTH_MEAN_WASTE_MASS and the LAST_MONTH_MEAN_WASTE_MASS with the same composite primary key.
10. PERCENTAGE_DIFFERENCE takes the proportion of the current month's difference with respect to the current month's mean mass, which is the entry with the same composite primary key as in CURRENT_MONTH_MEAN_WASTE_MASS.
11. LAST_18M_MEAN_INCREASE represents the mean difference based on the latest 18 ABSOLUTE_DIFFERENCE datapoints that are positive and ignores the rest.
12. LAST_18_MEAN_DECREASE represents the mean difference based on the latest 18 ABSOLUTE_DIFFERENCE datapoints that are negative and ignores the rest.
13. RELATIVE_STRENGTH represents the strength of the current trend.
14. RELATIVE_STRENGTH_INDEX is a standardized RELATIVE_STRENGTH.
15. The CARBON_EMISSION_PER_KG attribute stores the approximate amount of carbon dioxide released by one kilogram of a type of food waste.
16. CARBON_EMISSIONS attribute represents the amount of carbon dioxide released by a certain type of food waste, according to its entry in the day.

Relationships:

1. The relationship between WASTE_INVENTORY and STATISTICS is many-to-one.
2. The relationship between WASTE_INVENTORY and CARBON_EMISSIONS is many-to-one.
3. The relationship between STATISTICS and CURRENT_PREDICTION is many-to-one.
4. The relationship between CURRENT_PREDICTION and PAST_PREDICTION is one-to-one.
5. The relationship between STATISTICS and LAST_MONTH is one-to-one.
6. The relationship between STATISTICS and CURRENT_MONTH is one-to-one.
7. The relationship between LAST_MONTH and DIFFERENCE is one-to-one.
8. The relationship between CURRENT_MONTH and DIFFERENCE is one-to-one.
9. DIFFERENCE is the supertype to INCREASE and DECREASE.
10. The relationship between DIFFERENCE and WASTE_TREND is many-to-one.
11. The relationship between CARBON_EMISSIONS and CARBON_EMISSIONS_PER_KG is many-to-one.

Constraints:

1. (WASTE_DATE, FOOD_ID), (MONTHYEAR, FOOD_ID) and (PREDICTED_MONTHYEAR, FOOD_ID) are composite primary keys.
2. The WASTE_MASS attribute in WASTE_INVENTORY table are non-null, numerical values from the Food Waste Collection Module.
3. The CARBON_EMISSION_PER_KG attribute from the CARBON_EMISSIONS table is user inputted, non-null, numerical values.
4. All other values in the module are computer generated values.
5. The WASTE_DATE format must be in DD/MM/YYYY.
6. The MONTHYEAR and PREDICTED_MONTHYEAR attributes format must be in MM/YYYY.
7. The composite primary key (MONTHYEAR, FOOD_ID) in all tables in this module except for STATISTICS table and CARBON_EMISSIONS table are also foreign keys.
8. STATISTICS table is null until at least one month worth of data is entered into WASTE_INVENTORY table.
9. Prediction will not function until the minimum number of months required for each attribute is inputted into the WASTE_INVENTORY table.
10. WASTE_TREND table is disabled until there is at least 1 CHANGE_TYPE: DECREASE subtype and 1 CHANGE_TYPE: INCREASE subtype for DIFFERENCE table.
11. The entries in the DIFFERENCE table must have complete disjointed subtypes discriminated using the discriminator CHANGE_TYPE, such that the record enters either CHANGE_TYPE:INCREASE or CHANGE_TYPE :DECREASE.
12. Any attributes functionally dependent on computer generated attributes that are not functional are also none functional.

3.4 Combined ERD Module

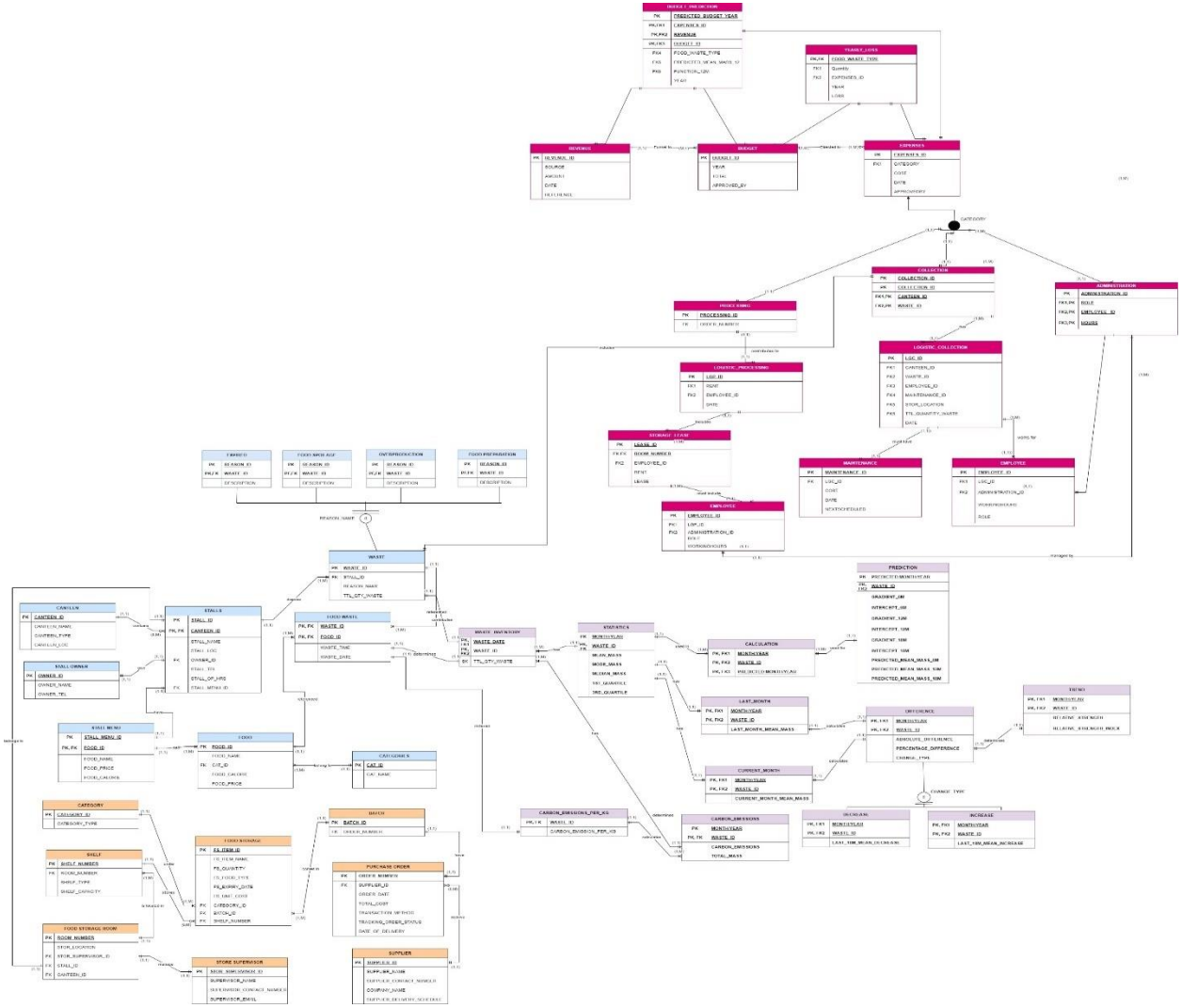


Figure 5: University Canteen Food Waste Database Management System ERD. Link: https://drive.google.com/file/d/1oQs_Hblw88AOUKCVmhoXu3Z3a_3M1tQk/view?usp=sharing

4 NORMALISATION

Normalisation is an important step in order to ensure end users are able to easily query and access data stored in the DBMS. Each table should be normalized to the third normalisation form (3NF) to eliminate partial and transitive dependencies. The occurrence of Boyce-Codd Normalisation Form (BCNF) is an additional Normalisation Form (NF) obtained as a result of the structure of some tables. This is to optimize the storage space used by the DBMS and data retrieval from the database system, which would become significant as the amount of time passes and the amount of data increases.

However, a few tables were only normalized to 2NF due to their read-heavy nature. These tables were created with the intention of only being read by the user and edited by computer generated algorithms. Since the tables are reliant on computer algorithms, performance becomes an issue as the amount of data needed to be handled increases. It is known that 2NF has a better performance than 3NF at the cost of lower data integrity, however this weakness can be avoided due to the nature of the table which is not reliant on direct human input. Coupled with frequent associations between the attributes due to their transitive dependencies, it is desirable to leave these tables in 2NF to achieve the previously stated results.

The following section contains dependency diagrams focused on each module and their normalisation form. Brief descriptions of the modules are located at the beginning of the diagrams.

4.1 Food Waste Collection Module

There are a total of nine tables in this module, which have all been normalized to 3NF.

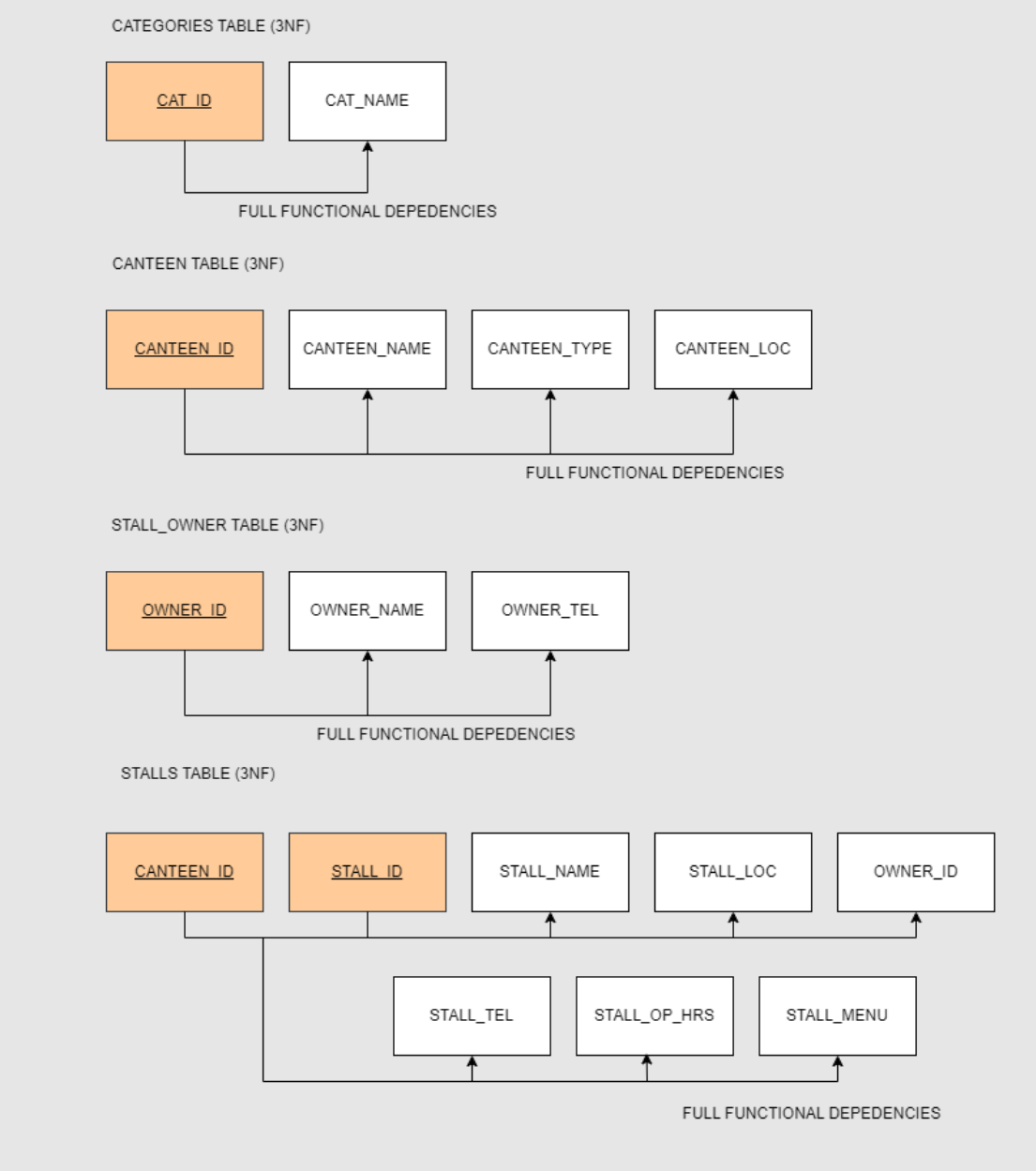


Figure 6(a): Dependency Diagram for Food Waste Collection Module

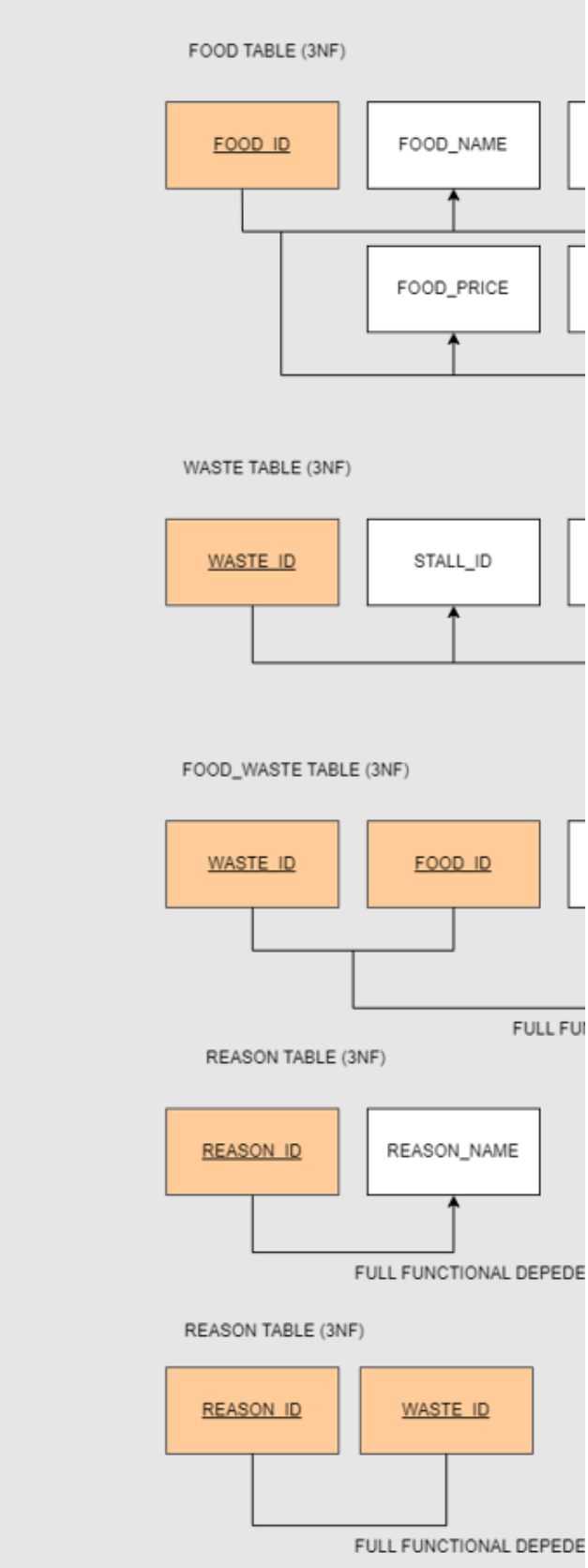

FOOD TABLE (3NF)
FOOD_ID
FOOD_NAME
FOOD_PRICE
WASTE TABLE (3NF)
WASTE_ID
STALL_ID
FOOD_WASTE TABLE (3NF)
WASTE_ID
FOOD_ID
FULL FU
REASON TABLE (3NF)
REASON_ID
REASON_NAME
FULL FUNCTIONAL DEPEDE
REASON TABLE (3NF)
REASON_ID
WASTE_ID
FULL FUNCTIONAL DEPEDE

Figure 6(b): Dependency Diagram for Food Waste Collection Module

4.2 Inventory Management System Module

All eight tables in the inventory management system module are in 3NF. The partial and transitive dependencies are eliminated through 3NF normalisation. The dependency diagram below indicates the various dependencies:

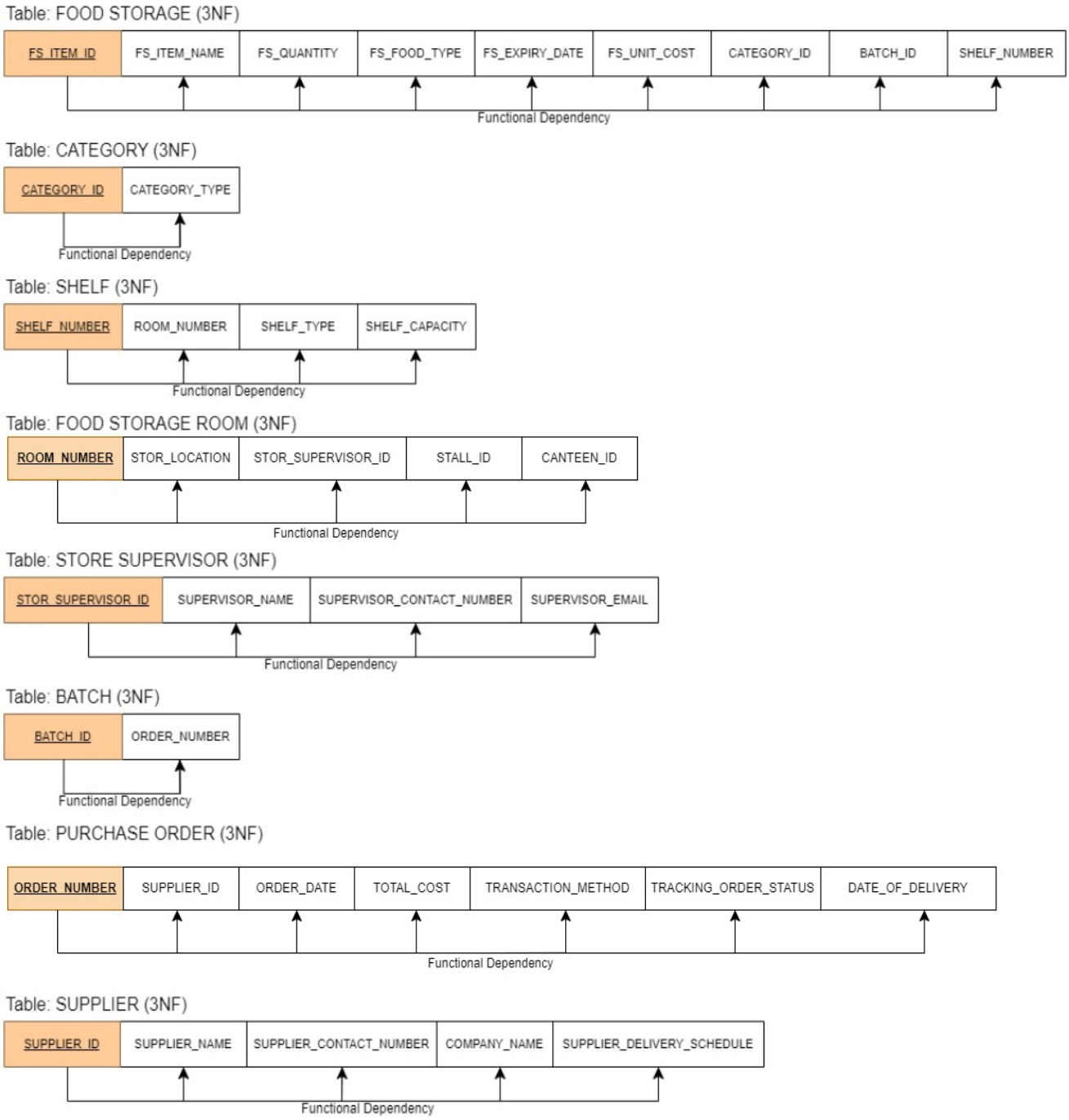
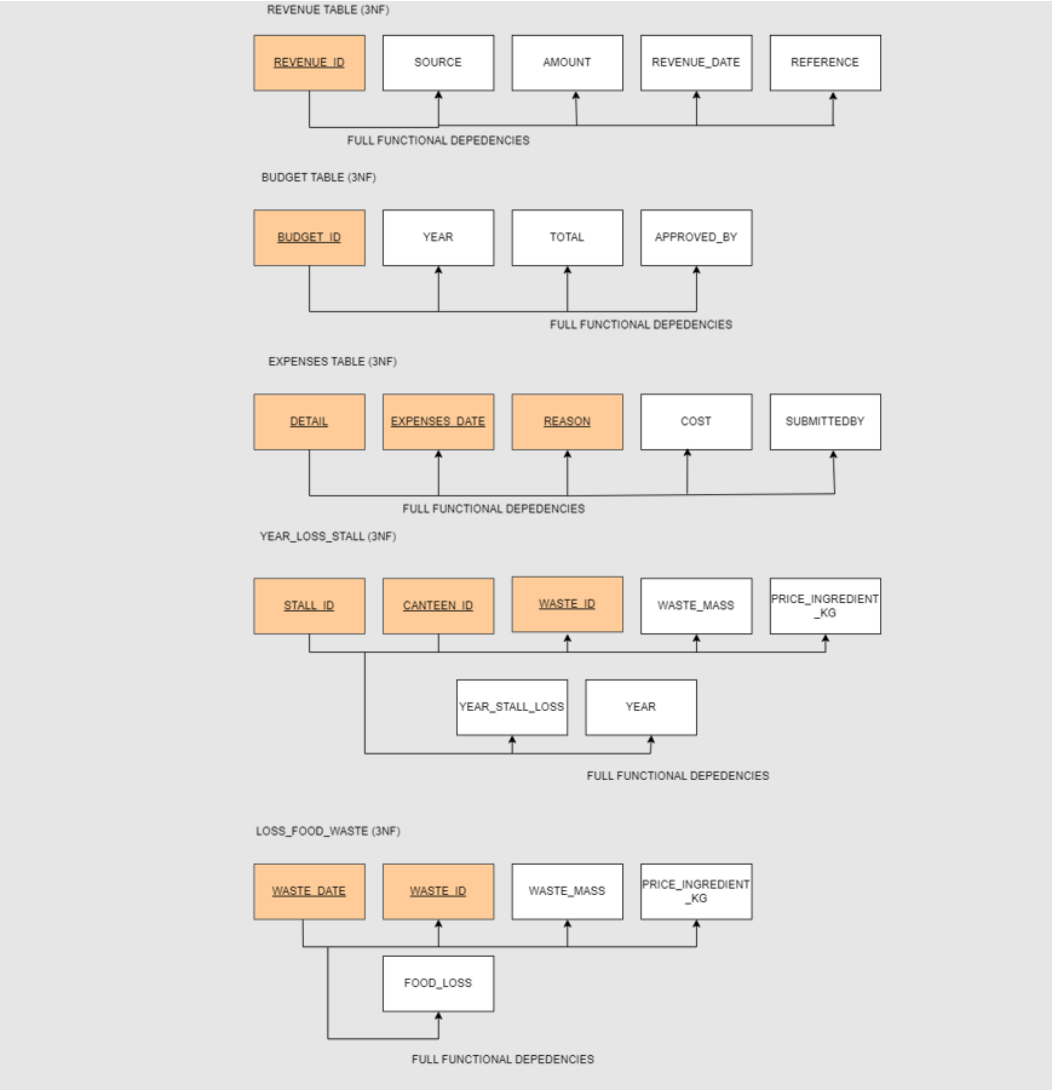
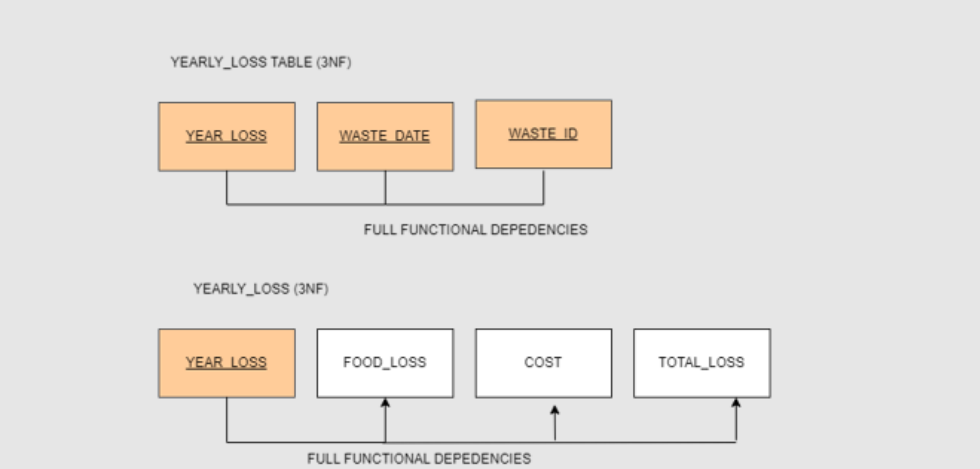


Figure 7: Dependency Diagram for Inventory Management Module

4.3 Financial

The total of seven tables in this module are all in the 3NF.





4.4 Analytical Module

There are a total of 12 tables in this module, with 3 tables in 2NF and 9 in BCNF.

Among the tables, CHANGE_TYPE:INCREASE and CHANGE_TYPE:DECREASE are subtypes of the DIFFERENCE table.

The following tables are the exceptions to the 3NF and BCNF:

1. CURRENT_PREDICTION table
2. PAST_PREDICTION table
3. DIFFERENCE table

The first two tables contain the attributes for predictions and parameters while the third table contains the differences of waste mass between the previous and current month. Since these data are often read together, attaching them to the same table can improve performance by decreasing the number of tables needed to be accessed to obtain the records.

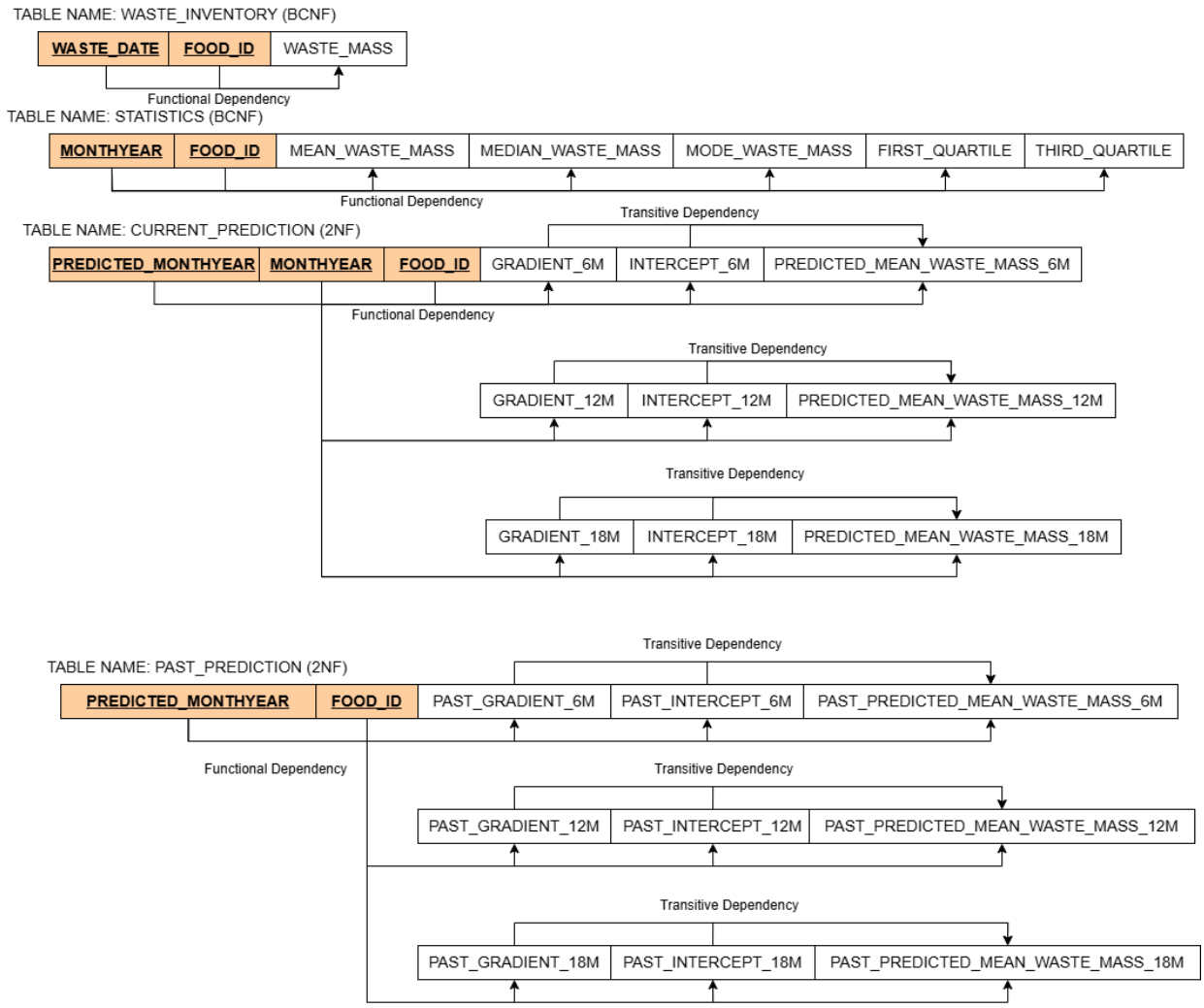


Figure 9(a): Dependency Diagram for Analytical Module

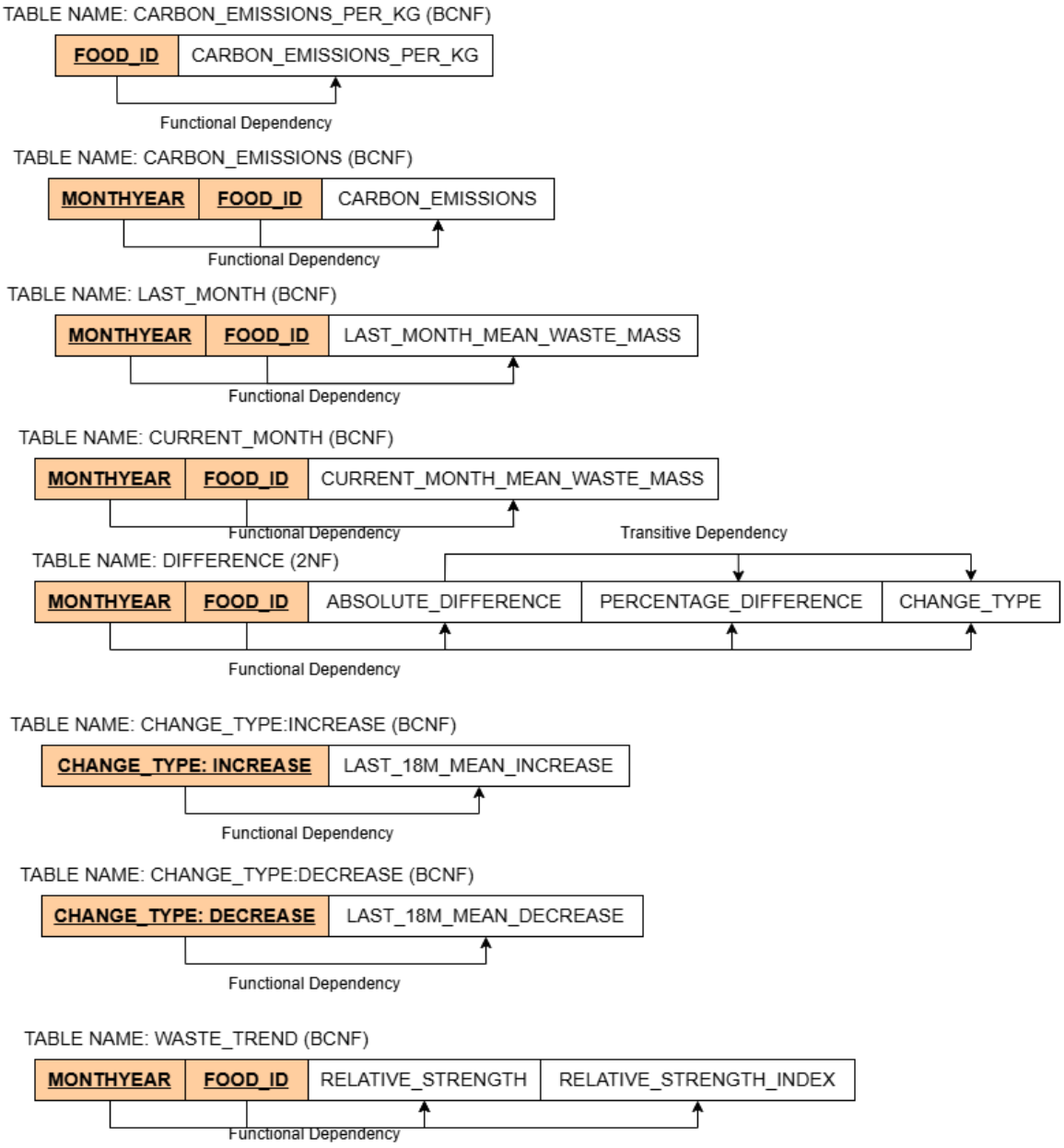


Figure 9(b): Dependency Diagram for Analytical Module

5 DATA DICTIONARY

5.2 Food Waste Collection Module

CATEGORIES TABLE

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
CATEGORIES					
CAT_ID	ID for categories	VARCHAR2(15)	'CAT1'	PRIMARY KEY	
CAT_NAME	Category names for each food types	VARCHAR2(50)	'FRUITS'	NOT NULL, CHECK (CAT_NAME IN ('FRUITS', 'VEGETABLES', 'MEAT', 'SEAFOOD', 'SNACKS', 'BEVERAGES', 'FRIED FOOD','VEGETARIAN))	
CANTEEN					
CANTEEN_ID	ID for canteens	VARCHAR2(15)	'CANT1'	PRIMARY KEY	
CANTEEN_NAME	Name of each canteen	VARCHAR2(50)	'CAFÉ FAJAR HARAPAN'	NOT NULL	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE

CANTEEN					
CANTEEN_TYPE	Type of each canteen (Independent: canteen is a restaurant itself, Dependent: canteen need many stalls)	VARCHAR2(50)	'INDEPENDENT'	NOT NULL, CHECK (CANTEEN_TYPE IN ('INDEPENDENT', 'DEPENDENT'))	
CANTEEN_LOC	Location of the canteen in campus	VARCHAR2(50)	'FAJAR'	NOT NULL	
STALL_OWNER					
OWNER_ID	ID for owner	VARCHAR2(15)	'OWN1'	PRIMARY KEY	
OWNER_NAME	Name of the owner	VARCHAR2(50)	'Santtosh'	NOT NULL	
OWNER_TEL	Telephone number of the owner	VARCHAR2(20)	'0183984119'		

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
STALLS					
CANTEEN_ID	ID for canteen	VARCHAR2(15)	'CANT1'	PRIMARY KEY	CANTEEN
STALL_ID	ID for each stall	VARCHAR2(15)	'ST1'	PRIMARY KEY	
STALL_NAME	Name of the stall	VARCHAR2(50)	‘Kedai Atuk Adam’	NOT NULL	
STALL_LOC	Location of the stall in campus	VARCHAR2(50)	‘A17’	NOT NULL	
OWNER_ID	ID for owner	VARCHAR2(15)	'OWN1'	FOREIGN KEY	STALL_OWNER
STALL_TEL	Telephone number of the stall	VARCHAR2(20)	'0487254620'	NOT NULL	
STALL_OP_HRS	Operating hours of stall	VARCHAR2(50)	'8AM-10PM'	NOT NULL	
STALL_MENU	Menu of stall	VARCHAR2(100)	'Nasi Goreng Ayam, Nasi Lemak, The Ais'	NOT NULL	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE

FOOD					
FOOD_ID	ID for food	VARCHAR2(15)	'FOOD1'	PRIMARY KEY	
FOOD_NAME	Name of food	VARCHAR2(50)	'Nasi Ayam'	NOT NULL	
CAT_ID	ID for category	VARCHAR2(15)	'CAT1'	FOREIGN KEY	CATEGORIES
FOOD_CALORIE	Calories of food in cal unit	DECIMAL(10, 2)	56.0	NOT NULL, CHECK (FOOD_CALORIE >= 0)	
FOOD_PRICE	Price of the food	DECIMAL(10, 2)	0.50	NOT NULL, CHECK (FOOD_PRICE >= 0)	
STALL_ID	ID for stall	VARCHAR2(15)	'ST1'	FOREIGN KEY	STALLS
CANTEEN_ID	ID for canteen	VARCHAR2(15)	'CANT1'	PRIMARY KEY	STALLS
WASTE					
WASTE_ID	ID for waste	VARCHAR2(15)	'W1'	PRIMARY KEY	
STALL_ID	ID for stall	VARCHAR2(15)	'ST1'	FOREIGN KEY	STALLS
CANTEEN_ID	ID for canteen	VARCHAR2(15)	'CANT1'	FOREIGN KEY	STALLS
WASTE_MASS	Total quantity of waste in kg unit	DECIMAL(10, 2)	10.00	NOT NULL, CHECK (WASTE_MASS >= 0)	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
FOOD_WASTE					
WASTE_ID	ID for waste	VARCHAR2(15)	'W1'	PRIMARY KEY FOREIGN KEY	WASTE
FOOD_ID	ID for food	VARCHAR2(15)	'FOOD1'	PRIMARY KEY FOREIGN KEY	FOOD
WASTE_DATE	Date of waste ID was entered	DATE	'2024-03-07'	NOT NULL	
REASON					
REASON_ID	ID for reason	VARCHAR2(15)	'R1'	PRIMARY KEY FOREIGN KEY	
REASON_NAME	Types of reason	VARCHAR2(15)	'Overproduction		
WASTE_REASON					
REASON_ID	ID for reason	VARCHAR2(15)	'R1'	PRIMARY KEY FOREIGN KEY	REASON
WASTE_ID	ID for waste	VARCHAR2(15)	'W1'	PRIMARY KEY FOREIGN KEY	WASTE
FOOD STORAGE					
FS_ITEM_ID	ID of food item	VARCHAR2(15)	FS10001	Primary Key	
FS_ITEM_NAME	Name of food item	VARCHAR2(30)	Maggi Sedap	Not null	
FS_QUANTITY	Quantity of food item	NUMBER (5,0)	250	Not null	
FS_FOOD_TYPE	Type of food	VARCHAR2(20)	Noodles	Not null	
FS_EXPIRY_DATE	Expiry date of food item	DATE	31/01/2025	Not null	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE

FOOD STORAGE					
FS_UNIT_COST	Unit price of food item	NUMBER (7,2)	3.50	Not null	
CATEGORY_ID	ID of food category	CHAR(5)	CAT01	Foreign Key	CATEGORY
BATCH_ID	ID of batch that food item comes in	VARCHAR2(15)	BT10002	Foreign Key	BATCH
SHELF_NUMBER	Number tag of shelf	CHAR(5,0)	A1314	Foreign Key	SHELF
CATEGORY					
CATEGORY_ID	ID of food category	CHAR(5)	CAT01	Primary Key	
CATEGORY_TYPE	Name of food category	VARCHAR2(30)	Ultra-processed food	Not null	
SHELF					
SHELF_NUMBER	Number tag of shelf	CHAR(5)	A1314	Primary Key	
ROOM_NUMBER	Storage room number	VARCHAR2(10)	SB288	Foreign Key	FOOD STORAGE ROOM
SHELF_TYPE	Type of shelf	VARCHAR2(20)	Shelf rack	Not null	
ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
SHELF					
SHELF_CAPACITY	Capacity number of the shelf	NUMBER (5,0)	2300	Not null	
FOOD STORAGE ROOM					
ROOM_NUMBER	Storage room number	VARCHAR2(10)	SB288	Primary Key	
STOR_LOCATION	Location of the storage room	VARCHAR2(30)	Subaidah USM	Not null	

STOR_SUPERVISOR_ ID	ID of store supervisor	VARCHAR2(15)	SS98737	Foreign Key	STOR SUPERVIS OR
STALL_ID	ID of owned stall	VARCHAR2(15)	ST1	Foreign Key	STALLS
CANTEEN_ID	ID for canteen	VARCHAR2(15)	CANT1	Foreign Key	STALLS
STOR SUPERVISOR					
STOR_SUPERVISOR_ID	ID of store supervisor	VARCHAR2(15)	SS98737	Primary Key	
SUPERVISOR_ CONTACT_NUMBER	Supervisor's phone number	NUMBER (20,0)	0162285693	Not null	
SUPERVISOR_EMAIL	Supervisor's email	VARCHAR2(50)	ahmadfirdz@gmail.com	Not null	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
BATCH					
BATCH_ID	ID of batch that food item comes in	VARCHAR2(15)	BT10002	Primary Key	
ORDER_NUMBER	Purchase Order Number	VARCHAR2(10)	XW3456	Foreign Key	PURCHASE ORDER
PURCHASE ORDER					
ORDER_NUMBER	Purchase Order Number	VARCHAR2(10)	XW3456	Primary Key	
SUPPLIER_ID	ID of supplier	VARCHAR2(15)	SP55978	Foreign Key	SUPPLIER
ORDER_DATE	Date of purchase order	DATE	19/06/2024	Not null	
TOTAL_COST	Total cost value of purchase	NUMBER (7,2)	508.60	Not null	

TRANSACTION_ METHOD	Method of purchase transaction	VARCHAR2(30)	Maybank Online	Not null	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
PURCHASE ORDER					
TRACKING_ORDER_STATUS	Status of tracking purchase order	VARCHAR2(10)	Pending	Not null	
DATE_OF_DELIVERY	Delivered date	DATE	28/06/2024	Nullable	
SUPPLIER					
SUPPLIER_ID	ID of supplier	VARCHAR2(15)	SP55978	Primary Key	
SUPPLIER_NAME	Name of supplier	VARCHAR2(30)	George Tan	Not null	
SUPPLIER_CONTACT_NUMBER	Supplier's phone number	NUMBER (20,0)	0195776386	Not null	
COMPANY_NAME	Name of supplier's company	VARCHAR2(100)	Gardenia Bakeries (Penang) Sdn. Bhd.	Not null	
SUPPLIER_DELIVERY_SCHEDULE	Supplier's Delivery Schedule	VARCHAR2(70)	Weekdays only	Not null	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
WASTE_INVENTORY					
FOOD_ID	Identifier for waste	INT		Primary key	
WASTE_DATE	Date of waste record	DATE	DD/MM/YYYY	Not null	
WASTE_MASS	Mass of waste	FLOAT		Not null	
STATISTICS					
FOOD_ID	Identifier for waste	INT		Foreign key	WASTE_INVENTORY
MEAN_WASTE_MASS	Mean mass of waste	FLOAT			
MODE_WASTE_MASS	Mode mass of waste	FLOAT			
MEDIAN_WASTE_MASS	Median mass of waste	FLOAT			
FIRST_QUARTILE	First quartile mass	FLOAT			
THIRD_QUARTILE	Third quartile mass	FLOAT			
MONTHYEAR	Month and year of the record	DATE	MM/YYYY	Primary key, foreign key	WASTE_INVENTORY

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
CURRENT_PREDICTION					
MONTHYEAR	Current month and year	DATE	MM/YYYY	Foreign key	STATISTICS
PREDICTED_MONTHYEAR	Predicted month and year	DATE	MM/YYYY	Primary key, foreign key	PAST_PREDICTION

FOOD_ID	Identifier for waste	INT		Primary key, foreign key	STATISTICS
CURRENT_PREDICTION					
GRADIENT_6M	Gradient for 6 months prediction	FLOAT			
INTERCEPT_6M	Intercept for 6 months prediction	FLOAT			
GRADIENT_12M	Gradient for 12 months prediction	FLOAT			
CURRENT_PREDICTION					
INTERCEPT_12M	Intercept for 12 months prediction	FLOAT			
GRADIENT_18M	Gradient for 18 months prediction	FLOAT			

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
CURRENT_PREDICTION					
INTERCEPT_18M	Intercept for 18 months prediction	FLOAT			
PREDICTED_MEAN_WASTE_MASS_6M	Predicted mean mass for 6 months	FLOAT			
PREDICTED_MEAN_WASTE_MASS_12M	Predicted mean mass for 12 months	FLOAT			
PREDICTED_MEAN_WASTE_MASS_18M	Predicted mean mass for 18 months	FLOAT			
PAST_PREDICTION					
PREDICTED_MONTHYEAR	Predicted month and year	DATE	MM/YYYY	Primary key	
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	CURRENT_PREDICTION
GRADIENT_6M	Gradient for 6 months prediction	FLOAT			

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
PAST_PREDICTION					
INTERCEPT_6M	Intercept for 6 months prediction	FLOAT			
GRADIENT_12M	Gradient for 12 months prediction	FLOAT			
INTERCEPT_12M	Intercept for 12 months prediction	FLOAT			
GRADIENT_18M	Gradient for 18 months prediction	FLOAT			
INTERCEPT_18M	Intercept for 18 months prediction	FLOAT			
PAST_PREDICTED_MEAN_WASTE_MASS_6M	Predicted mean mass for 6 months	FLOAT			
PAST_PREDICTED_MEAN_WASTE_MASS_12M	Predicted mean mass for 12 months	FLOAT			

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
PAST_PREDICTION					
PAST_PREDICTED_MEAN_WASTE_MASS_18M	Predicted mean mass for 18 months	FLOAT			
LAST_MONTH					
MONTHYEAR	Last month and year	DATE	MM/YYYY	Primary key, foreign key	STATISTICS
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	STATISTICS
LAST_MONTH_MEAN_WASTE_MASS	Mean mass for the last month	FLOAT		Not null	
CURRENT_MONTH					
MONTHYEAR	Current month and year	DATE	MM/YYYY	Primary key, foreign key	STATISTICS
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	STATISTICS
CURRENT_MONTH_MEAN_WASTE_MASS	Mean mass for the current month	FLOAT		Not null	
MONTHYEAR	Month and year	DATE	MM/YYYY	Primary key, foreign key	STATISTICS
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	STATISTICS

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
DIFFERENCE					
ABSOLUTE_DIFFERENCE	Absolute difference in waste mass	FLOAT			
PERCENTAGE_DIFFERENCE	Percentage difference in waste mass	FLOAT			

CHANGE_TYPE	Type of change (INCREASE/DECREASE)	VARCHAR2(10)		Not null	
WASTE_TRENDS					
MONTHYEAR	Month and year	DATE	MM/YYYY	Primary key, foreign key	DIFFERENCE
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	DIFFERENCE
RELATIVE_STRENGTH	Relative strength	FLOAT			
RELATIVE_STRENGTH_INDEX	Relative strength index	FLOAT			
DECREASE					
MONTHYEAR	Month and year	DATE	MM/YYYY	Primary key, foreign key	DIFFERENCE
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	DIFFERENCE
LAST_18M_MEAN_DECREASE	Mean decrease for the last 18 months	FLOAT			

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
INCREASE					
MONTHYEAR	Month and year	DATE	MM/YYYY	Primary key, fk	DIFFERENCE
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	DIFFERENCE
LAST_18M_MEAN_INCREASE	Mean increase for the last 18 months	FLOAT			
CARBON_EMISSIONS_PER_KG					
FOOD_ID	Identifier for waste	INT		Primary key	

CARBON_EMISSION_PER_KG	Carbon emissions per kg of waste	FLOAT		Not null	
CARBON_EMISSIONS					
MONTHYEAR	Month and year	DATE	MM/YYYY	Primary key	
FOOD_ID	Identifier for waste	INT		Primary key, foreign key	CARBON_EMISSIONS_PER_KG
CARBON_EMISSIONS	Total carbon emissions	FLOAT			
REVENUE					
REVENUE_ID	Identifier for revenue	INT		Primary key	
SOURCE	Source of said revenue	VARCHAR		Not Null	
AMOUNT	Amount of revenue	FLOAT		Not null, value must be above 0.0	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
REVENUE					
REVENUE_DATE	Date the revenue was recorded	DATE	DD/MM/YYYY	Not null	
REFERENCE	Reference for the revenue	VARCHAR		Not null	
BUDGET					
BUDGET_ID	Unique identifier for the budget record	INT		Primary Key	
YEAR	Year for which the budget is applicable	INT	YYYY	Not null	

TOTAL	Total budget for the year	FLOAT		Not null, value must be above 0.0	
APPROVED_BY	Person/entity who approved the budget	VARCHAR			
EXPENSES					
DETAIL	Details of the expense	VARCHAR		Primary Key	
REASON	Reason for the expenses	VARCHAR		Primary Key	
COST	Cost associated with the expense	FLOAT		Not null, value must be above 0.0	
SUBMITTED_BY	Person who submitted the expenses			Not null	
EXPENSES_DATE	Has the date expenses	DATE	DD/MM/YY		
ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
YEARLY_LOSS					
YEAR_LOSS	The year that is described, or to be calculated of said loss	INT	YYYY	Primary Key	
FOOD_LOSS	Total amount of food loss for the year	FLOAT		Not null, value must be above 0.0	LOSS_FOOD_WASTE
COST	Cost associated with the expense	FLOAT		Not null, value must be above 0.0	EXPENSES
TOTAL_LOSS	Total financial loss for the year	INT		Not null. value must be above 0.0	

ATTRIBUTE	DESCRIPTION	DATA TYPE	DATA FORMAT	CONSTRAINTS	REFERENCE TO TABLE
LOSS_FOOD_WASTE					
WASTE_DATE	Date of waste ID was entered	DATE	DD/MM/YY	Not null ,Foreign key ,Primary key	FOOD_WASTE
WASTE_ID	ID for waste	VARCHAR2(15)	'W1'	Primary key, Foreign key	WASTE
WASTE_MASS	Total quantity of waste in kg unit	FLOAT		Foreign key	WASTE
PRICE_INGREDIENT_KG	Price per kilogram of wasted ingredient	FLOAT		Not null, value must be above 0.0	
FOOD_LOSS	Amount of food loss	FLOAT		Not null, value must be above 0.0	
YEAR_LOSS_STALL					
STALL_ID	Unique identifier for each stall	VARCHAR2(15)		Not null, Primary key, Foreign key	STALLS

CANTEEN_ID	Unique identifier for each canteen	VARCHAR2(15)		Not null, Primary key, Foreign key	STALLS
WASTE_ID	ID for waste	VARCHAR2(15)	'W1'	Primary key, Foreign key	WASTE
WASTE_MASS	Total quantity of waste in kg unit	FLOAT		Foreign key	WASTE
PRICE_INGREDIENT_KG	Price per kilogram of wasted ingredient	FLOAT		Not null, value must be above 0.0	
YEAR_STALL_LOSS	Yearly loss for the stall	FLOAT		Not null, value must be above 0.0	
YEAR	YEAR for the stall	INT	YYYY		

6 DATABASE IMPLEMENTATION

For the creation of tables, sequences and triggers for the system, SQL scripts with Data Definition Language(DDL) and Data Manipulation Language(DML) were written locally and uploaded into the shared workspace in Oracle APEX. Triggers used to update tables between modules were added after the modules were hooked up properly. A preview of the scripts for each module is shown in the table below.

6.1 Table Creation

Module	Table Creation Commands
Food Waste Collection	CREATE TABLE CATEGORIES (CAT_ID VARCHAR2(15) PRIMARY KEY, CAT_NAME VARCHAR2(50) NOT NULL CHECK (CAT_NAME IN ('FRUITS', 'VEGETABLES', 'MEAT', 'SEAFOOD', 'SNACKS', 'BEVERAGES', 'FRIED FOOD', 'VEGETARIAN')));
Inventory Management System	CREATE TABLE CATEGORY (CATEGORY_ID CHAR(5), CONSTRAINT PK_CATEGORY_ID PRIMARY KEY(CATEGORY_ID), CATEGORY_TYPE VARCHAR2(30) NOT NULL);
Financial	CREATE TABLE REVENUE (REVENUE_ID INT PRIMARY KEY, SOURCE VARCHAR NOT NULL, AMOUNT FLOAT NOT NULL CHECK (AMOUNT > 0.0), REVENUE_DATE DATE NOT NULL, REFERENCE VARCHAR NOT NULL);
Analytical	CREATE TABLE WASTE_INVENTORY (FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL, WASTE_DATE DATE NOT NULL, MASS FLOAT NOT NULL, PRIMARY KEY (FOOD_WASTE_TYPE), CONSTRAINT CHK_WASTE_DATE_FORMAT CHECK (WASTE_DATE = TO_DATE(TO_CHAR(WASTE_DATE, 'DD/MM/YYYY'), 'DD/MM/YYYY'))

	);

Table 5: Preview of table creation commands extracted from SQL Scripts for each module

6.2 Trigger Creation

Module	Trigger Creation Commands

Food Waste Collection	CREATE OR REPLACE TRIGGER categories_id_trigger
	BEFORE INSERT ON CATEGORIES
	FOR EACH ROW
	BEGIN
	:NEW.CAT_ID := 'CAT' || categories_seq.NEXTVAL;
	END;

Financial Assessment	DELIMITER // CREATE TRIGGER calculate_yearly_loss AFTER INSERT OR UPDATE ON EXPENSES FOR EACH ROW BEGIN DECLARE total_cost FLOAT; DECLARE total_food_loss FLOAT; DECLARE year INT; SET year = YEAR(NEW.EXPENSES_DATE); -- Calculate total cost from EXPENSES table SELECT SUM(COST) INTO total_cost FROM EXPENSES WHERE YEAR(EXPENSES_DATE) = year; -- Calculate total food loss from LOSS_FOOD_WASTE table SELECT SUM(FOOD_LOSS) INTO total_food_loss FROM LOSS_FOOD_WASTE WHERE YEAR(WASTE_DATE) = year; -- Insert or update the YEARLY_LOSS table INSERT INTO YEARLY_LOSS (YEAR_LOSS, FOOD_LOSS, COST, TOTAL_LOSS) VALUES (year, total_food_loss, total_cost, total_food_loss + total_cost)

	ON DUPLICATE KEY UPDATE FOOD_LOSS = total_food_loss, COST = total_cost, TOTAL_LOSS = total_food_loss + total_cost; END// DELIMITER ;

Analytical	CREATE OR REPLACE TRIGGER UPDATE_STATISTICS
	AFTER INSERT OR UPDATE OF MASS ON WASTE_INVENTORY
	FOR EACH ROW
	DECLARE
	vMonthYear DATE;
	vMassCount INT;
	vLastDay INT;
	BEGIN
	vMonthYear := TRUNC(:NEW.WASTE_DATE, 'MM');
	
	SELECT COUNT(DISTINCT TO_CHAR(WASTE_DATE, 'DD'))
	INTO vMassCount
	FROM WASTE_INVENTORY
	WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear;

Table 6: Preview of trigger commands extracted from SQL Scripts for Food Waste Collection Module, Financial Assessment Module and Analytical Module

6.3 Sequence Creation

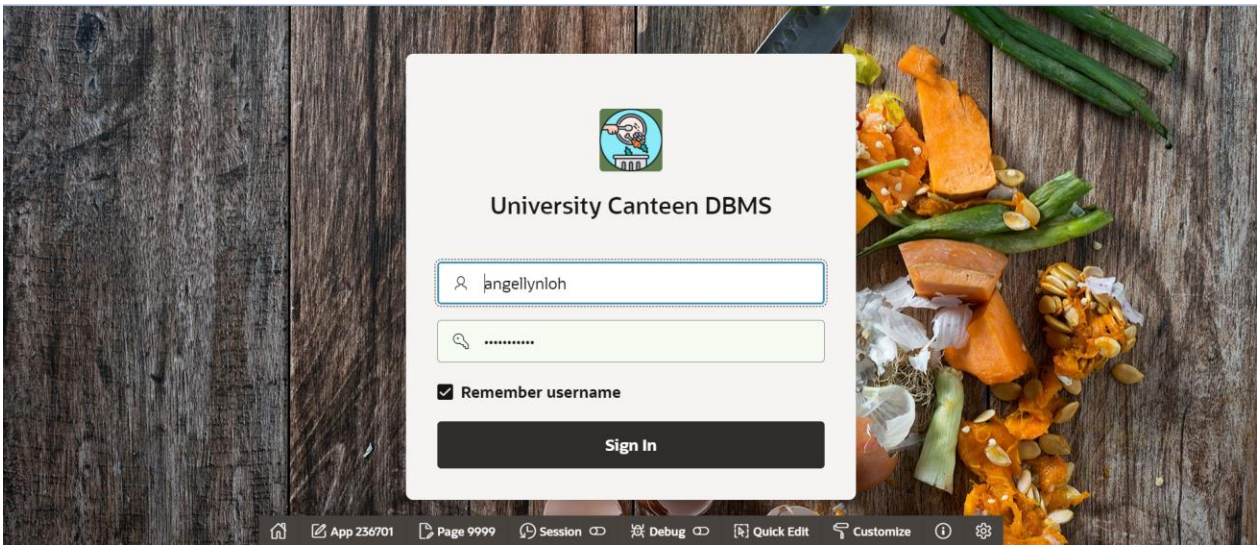
Module	Sequence Creation Commands
Food Waste Collection	CREATE SEQUENCE categories_seq START WITH 1 INCREMENT BY 1 NOCACHE;

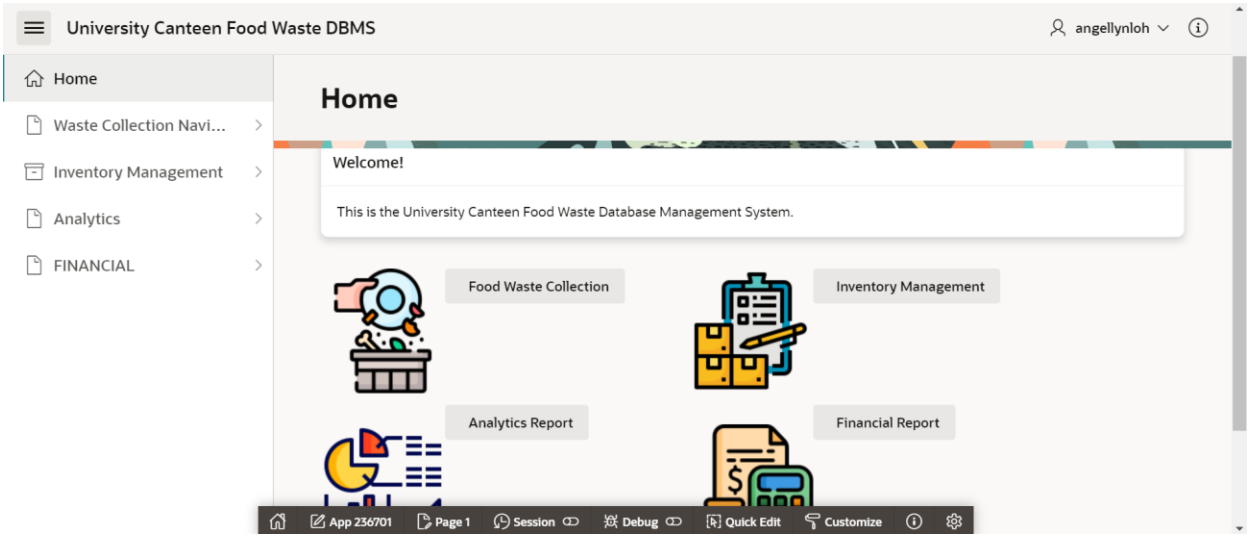
Table 7: Preview of sequence commands extracted from SQL Scripts for Food Waste Collection Module

*For the full SQL Script, see **Appendix *1***

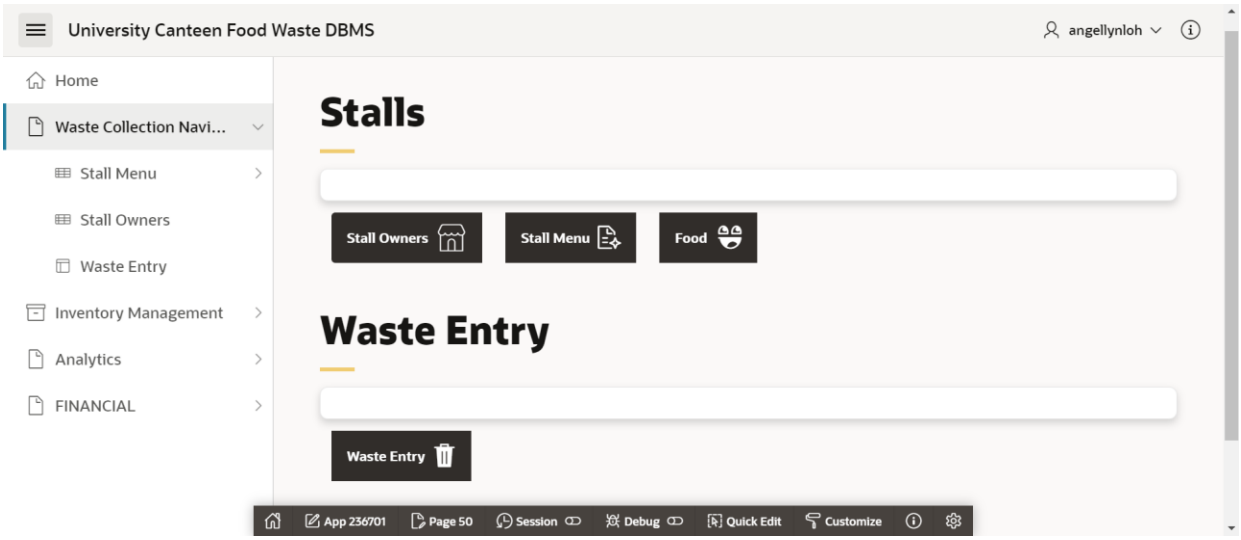
7 FRONT-END SYSTEM DESIGN AND IMPLEMENTATION

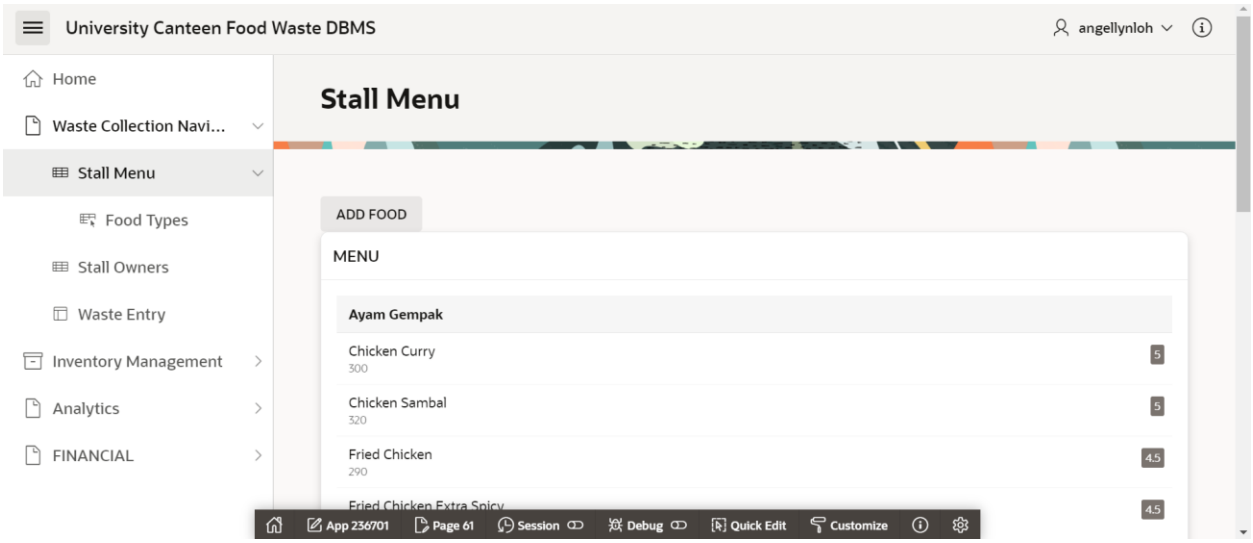
7.1 Application Page and Home Page

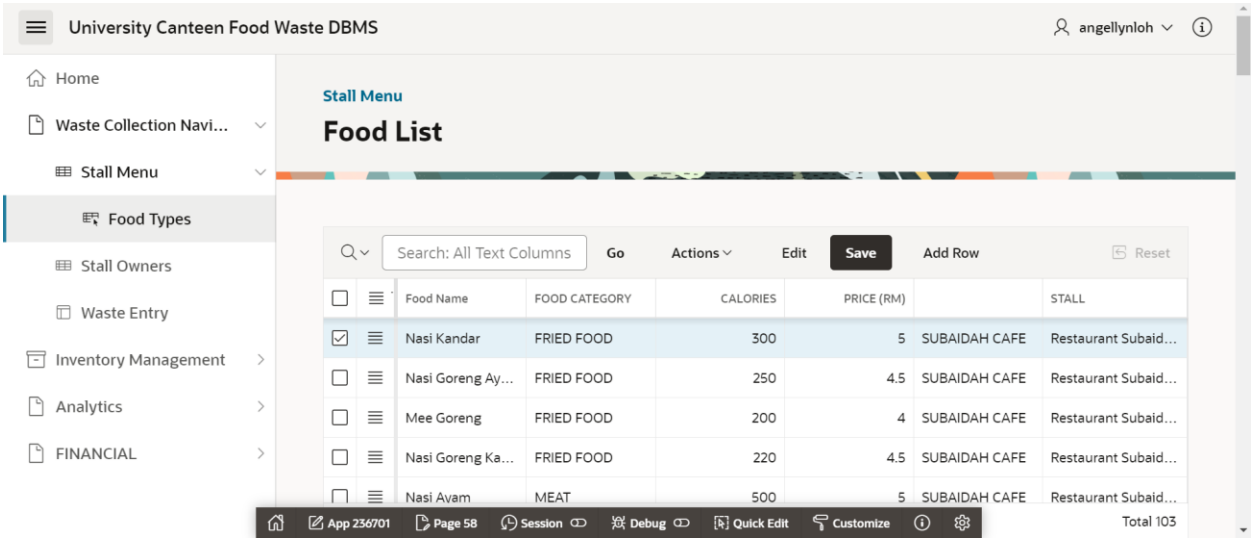


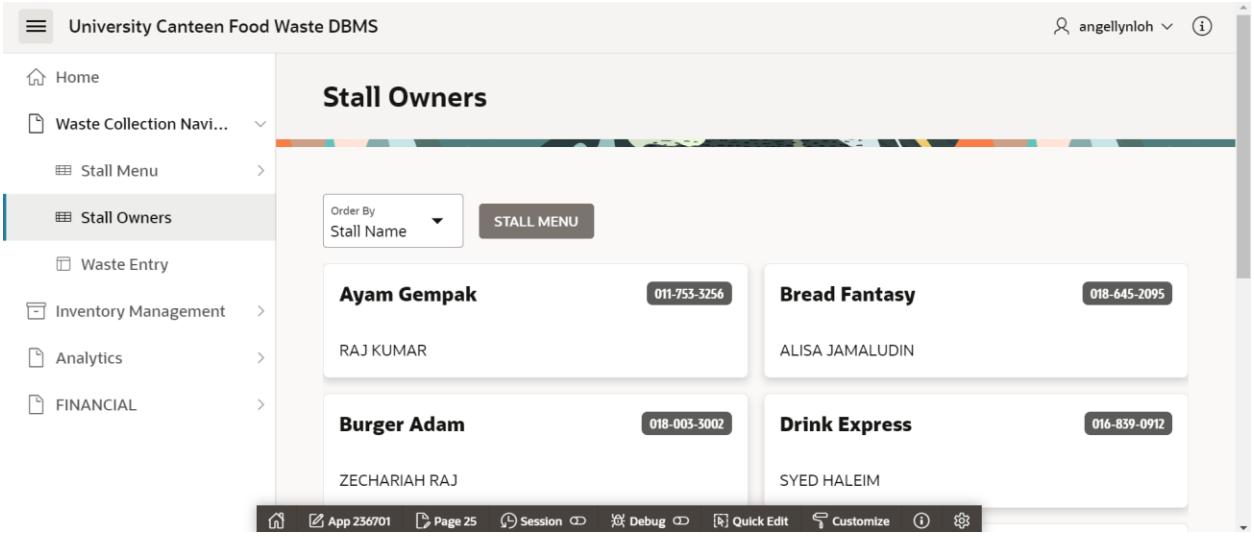


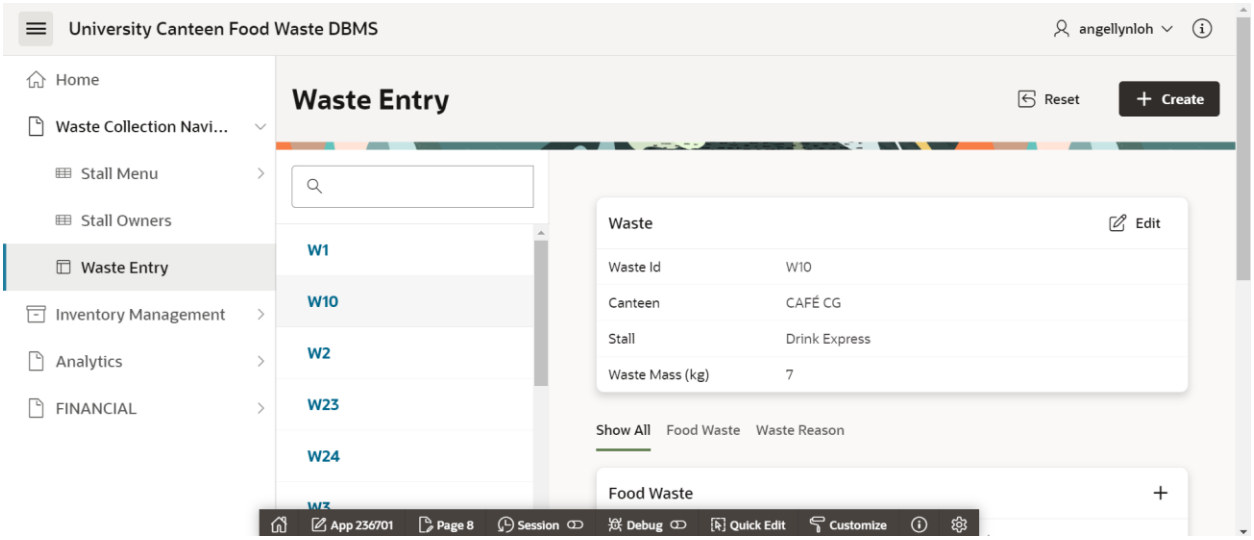
7.1 Food Waste Collection Page



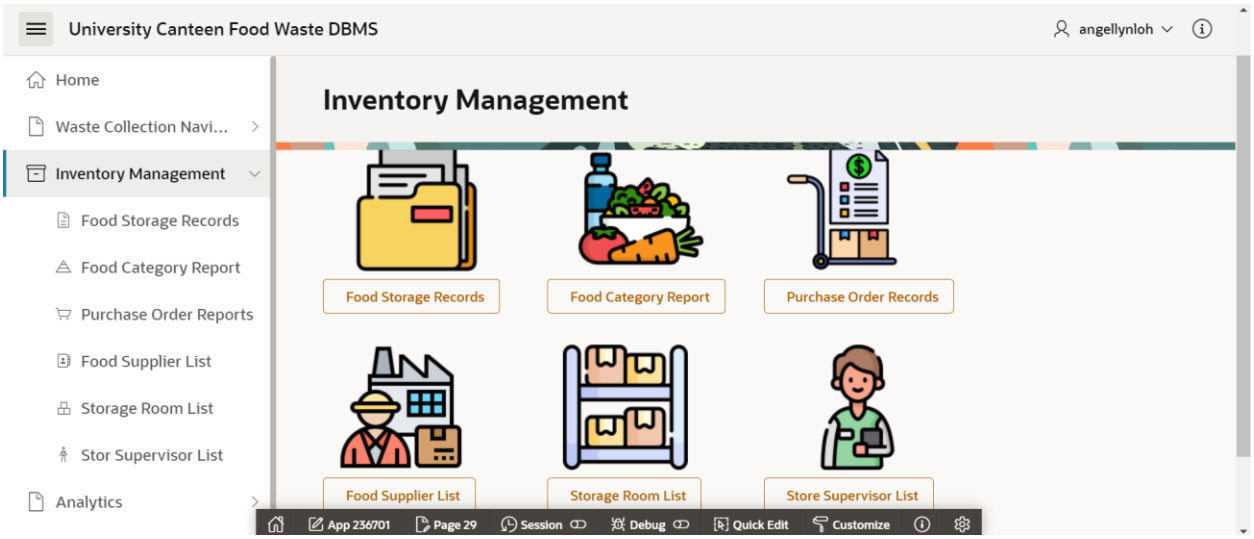


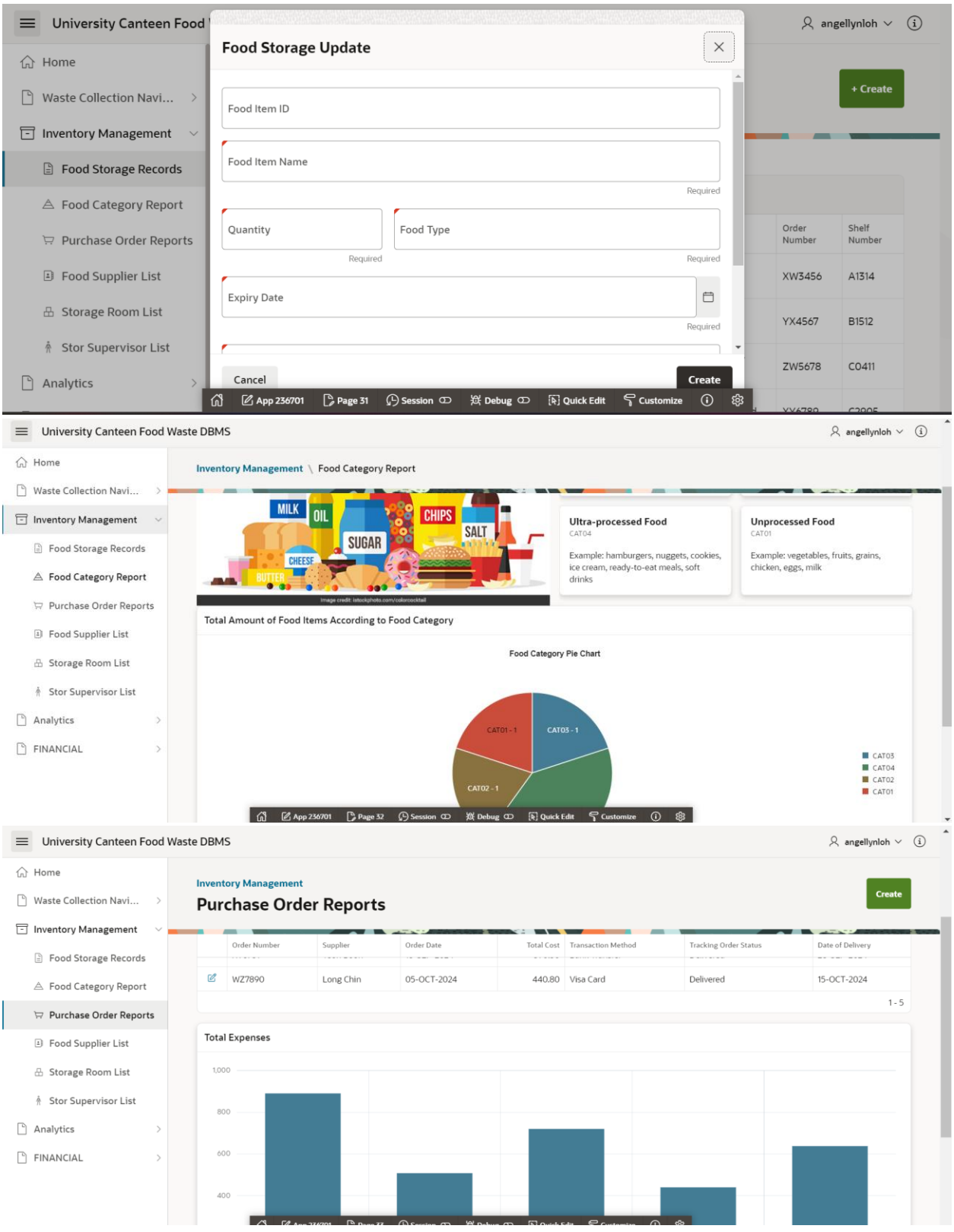


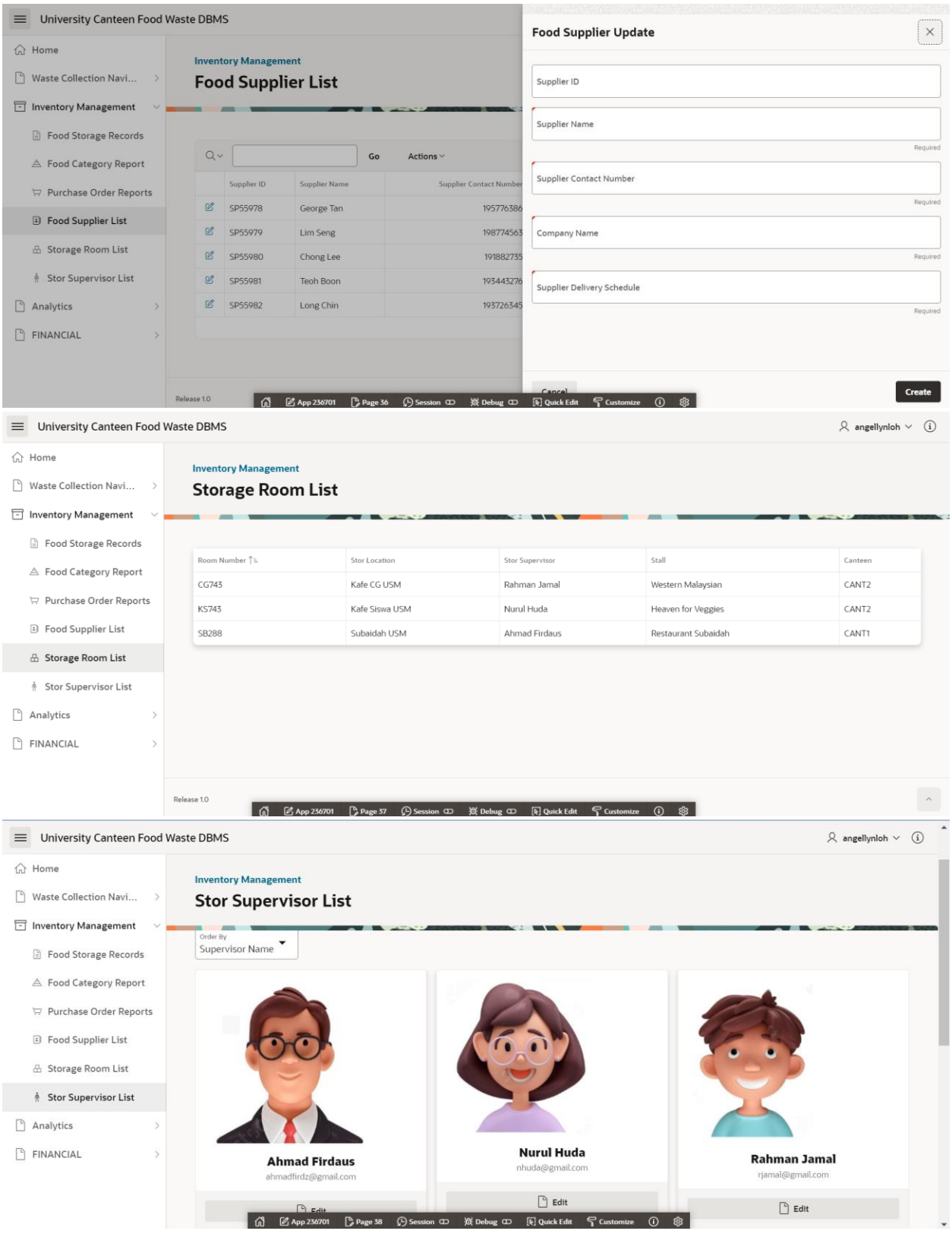




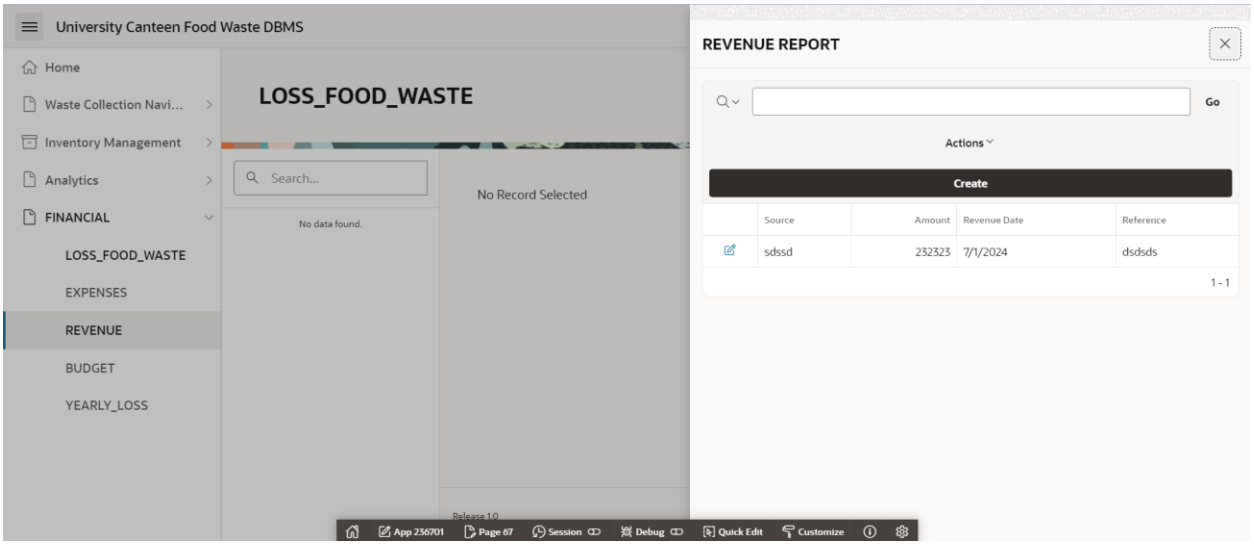
7.2 Inventory Management Page



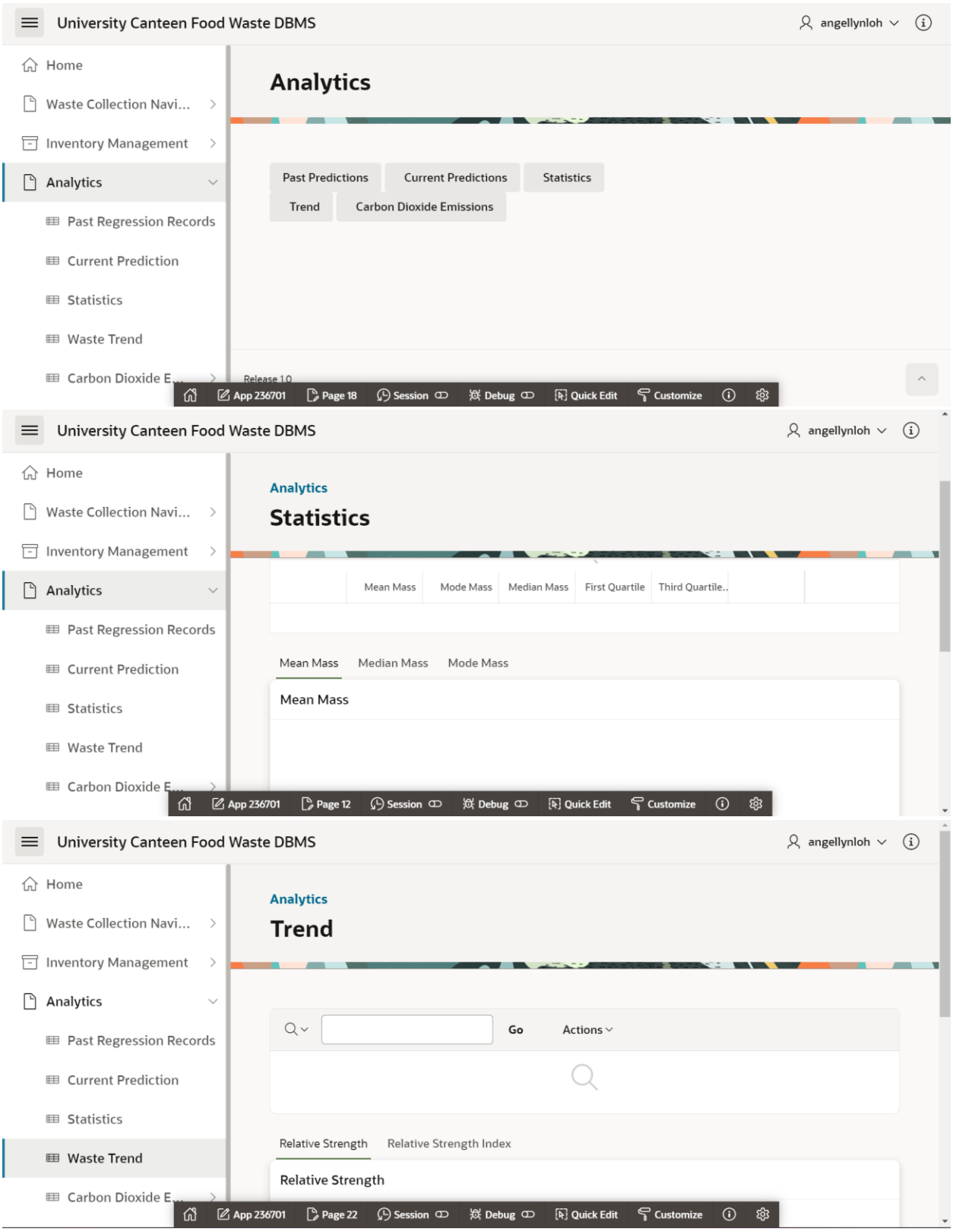

University Canteen Food
angellynloh
Home
Waste Collection Navi...
Inventory Management
Food Storage Records
Food Category Report
Purchase Order Reports
Food Supplier List
Storage Room List
Stor Supervisor List
Analytics
+ Create
Order Number
Shelf Number
XW3456
A1314
YX4567
B1512
ZW5678
C0411
Food Storage Update
Food Item ID
Food Item Name
Required
Quantity
Food Type
Required
Required
Expiry Date
Required
Cancel
Create
App 236701
Page 31
Session
Debug
Quick Edit
Customize
University Canteen Food Waste DBMS
angellynloh
Home
Waste Collection Navi...
Inventory Management
Food Storage Records
Food Category Report
Purchase Order Reports
Food Supplier List
Storage Room List
Stor Supervisor List
Analytics
FINANCIAL
Inventory Management \ Food Category Report
MILK
OIL
SUGAR
CHIPS
SALT
CHEESE
BUTTER
Ultra-processed Food
CAT04
Example: hamburgers, nuggets, cookies, ice cream, ready-to-eat meals, soft drinks
Unprocessed Food
CAT01
Example: vegetables, fruits, grains, chicken, eggs, milk
Total Amount of Food Items According to Food Category
Food Category Pie Chart
CAT01 - 1
CAT03 - 1
CAT02 - 1
CAT03
CAT04
CAT02
CAT01
App 236701
Page 32
Session
Debug
Quick Edit
Customize
University Canteen Food Waste DBMS
angellynloh
Home
Waste Collection Navi...
Inventory Management
Food Storage Records
Food Category Report
Purchase Order Reports
Food Supplier List
Storage Room List
Stor Supervisor List
Analytics
FINANCIAL
Inventory Management
Purchase Order Reports
Create
Order Number
Supplier
Order Date
Total Cost
Transaction Method
Tracking Order Status
Date of Delivery
WZ7890
Long Chin
05-OCT-2024
440.80
Visa Card
Delivered
15-OCT-2024
1 - 5
Total Expenses
1,000
800
600
400


University Canteen Food Waste DBMS
Home
Waste Collection Navi...
Inventory Management
Food Storage Records
Food Category Report
Purchase Order Reports
Food Supplier List
Storage Room List
Stor Supervisor List
Analytics
FINANCIAL
Inventory Management
Food Supplier List
Go
Actions
Supplier ID
Supplier Name
Supplier Contact Number
SP55978
George Tan
SP55979
Lim Seng
SP55980
Chong Lee
SP55981
Teoh Boon
SP55982
Long Chin
Food Supplier Update
Supplier ID
Supplier Name
Required
Supplier Contact Number
Required
Company Name
Required
Supplier Delivery Schedule
Required
Cancel
Create
Release 1.0
App 236701
Page 36
Session
Debug
Quick Edit
Customize
University Canteen Food Waste DBMS
angellynloh
Inventory Management
Storage Room List
Room Number
Stor Location
Stor Supervisor
Stall
Canteen
CG743
Kafe CG USM
Rahman Jamal
Western Malaysian
CANT2
KS743
Kafe Siswa USM
Nurul Huda
Heaven for Veggies
CANT2
SB288
Subaidah USM
Ahmad Firdaus
Restaurant Subaidah
CANT1
Release 1.0
App 236701
Page 37
Session
Debug
Quick Edit
Customize
University Canteen Food Waste DBMS
angellynloh
Inventory Management
Stor Supervisor List
Order By
Supervisor Name
Ahmad Firdaus
ahmadfirdz@gmail.com
Nurul Huda
nhuda@gmail.com
Rahman Jamal
rjamal@gmail.com
Edit
Edit
Edit
App 236701
Page 38
Session
Debug
Quick Edit
Customize

7.3 Financial Assesment Page



7.4 Analytics Page



8 PROJECT PROBLEMS AND PITFALLS

We used Oracle Application Express (APEX) for structuring the user interface for our database. Throughout the process, we faced plenty of challenges as the SQL language and Oracle APEX application were new to us. Problems such as familiarising ourselves with the functions in SQL and understanding the limits of Oracle APEX were among the biggest challenges for us. Additionally, our lack of experience has caused doubts on whether we can achieve our project goals. Fortunately, we had an excellent tutor and two amazing lecturers guiding us on the theories and applications of SQL and Oracle APEX, allowing us to quickly grasp the foundations and implement said theories in the project.

Besides that, we were able to experience the to-and-fro cycle of drawing and planning the Entity Relationship Diagram (ERD) and implementing the user interface in Oracle APEX. This made us realise the tediousness of the entire implementation process, and this is a continuous refinement process to optimise how we achieve our goals.

9 CONCLUSION

In conclusion, to construct a good database management system is a challenging ordeal. Various expertise are required in every step of the development phase, from brainstorming tools required to plan out a comprehensive DBMS, such as the Crow's foot notation used in the ERD development phase, to technical skills used in implementing the DBMS, such as a sufficient understanding of the Oracle APEX environment. We have created a university canteen food waste database management system which enables the university's canteen management to monitor the food waste situation in order to work towards the four SDGs involved in this project, which are **SDG2** (Zero Hunger), **SDG3** (Good Health and Wellbeing), **SDG12** (Responsible Consumption and Production) and **SDG13** (Climate action). Stall owners can keep track of the amount and quality of their ingredients, while the canteen management can set up new policies to work towards reducing food waste generation. Additionally, built-in analytical and financial assessment modules in the DBMS allows the canteen management to avoid having to involve Data Analysts for low-level analysis, such as simply predicting the amount of waste in the near future, improving accessibility of data analytics to the general public while automating simple processes.

Through the Oracle APEX web application, a dashboard allows users to easily switch between the different modules in the system. In addition, the navigation menu allows users to go into pages parked under the main pages representing the different modules. Benefits of such a design is a clean navigation menu and dashboard so that the user does not feel overwhelmed when using the app, while at the same time allowing users who want to know more about the data records to access and manipulate data through interactive forms and grids. Constraints have been implemented to prevent users from accidentally or maliciously changing some of the computer generated data, as the algorithms are in place to automate the prediction processes.

As a result, stall owners can now plan their next grocery run more meticulously as they will have better visibility and clarity to make decisions on their next purchase, allowing them to save ingredient costs while reducing food waste generated from expired food. Stall owners can also produce less food if they know the amount they usually produce for a certain amount of customers are not consumed completely. This decision is aided by the DBMS in revealing trends in customer consumption which may not vary proportionally to the quantity of food produced.

Overall, the DBMS empowers stall owners with more information about their customers in order to make better decisions.

For the university's canteen management, the DBMS can be used to make simple analysis on the statistics of daily, monthly or even yearly food waste statistics, allowing them to implement and enforce cost-effective policies on stall owners or even the customers. If a lot of food waste is generated by customers, it could mean the stalls in the canteen need to be reviewed for their food quality, which can in turn lead to internal audits and improvement in the quality of life of customers. However, it could also mean the customers have bad consumption behaviour, in which case other solutions, such as raising awareness on the amount of food waste generated, may be considered by the management. In all cases, the DBMS provides information which enables the canteen management to make accurate, valid and timely decisions.

Despite the comprehensiveness of the DBMS, there are still recommendations for improvement. For one, integration with Internet of Things (IoT) can create an immensely powerful, automated system to further minimize the food waste generation. The DBMS can take input from sensors with the ability to differentiate different food wastes and measure their mass, then process the data and send recommendations based on some machine learning model to the management team for them to take action. Purchase orders can also be done if storage spaces for food ingredients have sensors to detect the amount of each type of ingredient in the storage space such that if an ingredient falls below a certain threshold, a purchase order will be sent to the grocery store to order more ingredients. As it can be seen, integration with IoT creates more possibilities for automating manual work while working towards the SDGs.

Other recommendations include increasing its security and countermeasures against unauthorized users. As the current project focuses on making the DBMS a practical marvel, other issues such as data security, backup and integrity checks as the system is accessed by more than one user at a time were not given enough attention. Should the system be used in a large canteen, the number of end users will scale proportionally, requiring either scaling up or out of the system. Decentralisation and turning the DBMS into a distributed DBMS then becomes a viable solution with new issues such as concurrency control and access control. It is recommended that the DO-UNDO-REDO protocol be implemented when the number of transaction processors increases.

Future works for the DBMS include allowing users to sign up through the login page, increasing distribution, failure and performance transparency through augmenting the system and

increasing the number of built-in analytical functions. These features can increase the security and utility value of the application and DBMS, which will help university canteens all across Malaysia to adopt the system in order to work towards the SDGs. A simple first step would be to update the authorization schema of the login page first, which will allow new users to register themselves and only view parts of the database that are open to the public, such as the amount of food waste and carbon emissions per month. It is our hope that the DBMS can be adopted to large scale usage in order to eliminate hunger and encourage people to be more responsible with their production and consumption.

REFERENCES

1. Li, J.; Li, W.; Wang, L.; Jin, B. Environmental and Cost Impacts of Food Waste in University Canteen from a Life Cycle Perspective. Energies 2021, 14, 5907. https://doi.org/10.3390/en14185907
2. Oracle. (2024). *Database Express Edition 2 Day Developer's Guide*. Docs.oracle.com. https://docs.oracle.com/cd/E17781_01/appdev.112/e18147/tdddg_triggers.htm#TDDDG52200

APPENDIX

Appendix 1: SQL Scripts for Database Implementation

Appendix 1.1: Food Waste Module

```
CREATE TABLE CATEGORIES (
   CAT_ID VARCHAR2(15) PRIMARY KEY,
   CAT_NAME VARCHAR2(50) NOT NULL CHECK ( CAT_NAME IN ('FRUITS',
'VEGETABLES', 'MEAT', 'SEAFOOD', 'SNACKS', 'BEVERAGES', 'FRIED FOOD',
'VEGETARIAN'))
);

CREATE TABLE CANTEEN (
   CANTEEN_ID VARCHAR2(15) PRIMARY KEY,
   CANTEEN_NAME VARCHAR2(50) NOT NULL,
   CANTEEN_TYPE VARCHAR2(50) NOT NULL CHECK (CANTEEN_TYPE IN
('INDEPENDENT', 'DEPENDENT')),
   CANTEEN_LOC VARCHAR2(50) NOT NULL
);

CREATE TABLE STALL_OWNER (
   OWNER_ID VARCHAR2(15) PRIMARY KEY,
   OWNER_NAME VARCHAR2(50) NOT NULL,
   OWNER_TEL VARCHAR2(20) NOT NULL
);

CREATE TABLE STALLS (
   CANTEEN_ID VARCHAR2(15),
   STALL_ID VARCHAR2(15),
   STALL_NAME VARCHAR2(50) NOT NULL,
   STALL_LOC VARCHAR2(50) NOT NULL,
   OWNER_ID VARCHAR2(15),
   STALL_TEL VARCHAR2(20) NOT NULL,
   STALL_OP_HRS VARCHAR2(50) NOT NULL,
   STALL_MENU VARCHAR2(100) NOT NULL,
```

```
   PRIMARY KEY (CANTEEN_ID, STALL_ID),
   FOREIGN KEY (CANTEEN_ID) REFERENCES CANTEEN(CANTEEN_ID),
   FOREIGN KEY (OWNER_ID) REFERENCES STALL_OWNER(OWNER_ID)
);

CREATE TABLE FOOD (
   FOOD_ID VARCHAR2(15) PRIMARY KEY,
   FOOD_NAME VARCHAR2(50) NOT NULL,
   CAT_ID VARCHAR2(15),
   FOOD_CALORIE NUMBER(10, 2) NOT NULL CHECK (FOOD_CALORIE >= 0),
   FOOD_PRICE NUMBER(10, 2) NOT NULL CHECK (FOOD_PRICE >= 0),
   CANTEEN_ID VARCHAR2(15),
   STALL_ID VARCHAR2(15),
   FOREIGN KEY (CAT_ID) REFERENCES CATEGORIES(CAT_ID),
   FOREIGN KEY (CANTEEN_ID, STALL_ID) REFERENCES STALLS (CANTEEN_ID,
STALL_ID)
);--need to put canteenid as well bcs its composite pk from stalls as we wanted stall id

CREATE TABLE WASTE (
   WASTE_ID VARCHAR2(15) PRIMARY KEY,
   CANTEEN_ID VARCHAR2(15),
   STALL_ID VARCHAR2(15),
   TTL_QTY_WASTE NUMBER(10, 2) NOT NULL CHECK (TTL_QTY_WASTE >= 0),
   FOREIGN KEY (CANTEEN_ID, STALL_ID) REFERENCES STALLS(CANTEEN_ID,
STALL_ID)
);

CREATE TABLE FOOD_WASTE (
   WASTE_ID VARCHAR2(15),
   FOOD_ID VARCHAR2(15),
   WASTE_DATE DATE NOT NULL,
   PRIMARY KEY (WASTE_ID, FOOD_ID),
   FOREIGN KEY (WASTE_ID) REFERENCES WASTE(WASTE_ID),
   FOREIGN KEY (FOOD_ID) REFERENCES FOOD(FOOD_ID)
```

```
);

CREATE TABLE REASON (
   REASON_ID VARCHAR2(15) PRIMARY KEY,
   REASON_NAME VARCHAR2(255) NOT NULL
);

CREATE TABLE WASTE_REASON (
   REASON_ID VARCHAR2(15),
   WASTE_ID VARCHAR2(15),
   PRIMARY KEY (REASON_ID, WASTE_ID),
   FOREIGN KEY (REASON_ID) REFERENCES REASON(REASON_ID),
   FOREIGN KEY (WASTE_ID) REFERENCES WASTE(WASTE_ID)
);

-- Sequence Definitions
CREATE SEQUENCE categories_seq START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE canteen_seq START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE stall_owner_seq START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE stalls_seq START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE food_seq START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE waste_seq START WITH 1 INCREMENT BY 1 NOCACHE;
CREATE SEQUENCE reason_seq START WITH 1 INCREMENT BY 1 NOCACHE;

-- Trigger Definitions
CREATE OR REPLACE TRIGGER categories_id_trigger
BEFORE INSERT ON CATEGORIES
FOR EACH ROW
BEGIN
  :NEW.CAT_ID := 'CAT' || categories_seq.NEXTVAL;
END;



CREATE OR REPLACE TRIGGER canteen_id_trigger
```

```
BEFORE INSERT ON CANTEEN
FOR EACH ROW
BEGIN
  :NEW.CANTEEN_ID := 'CANT' || canteen_seq.NEXTVAL;
END;
/

CREATE OR REPLACE TRIGGER stall_owner_id_trigger
BEFORE INSERT ON STALL_OWNER
FOR EACH ROW
BEGIN
  :NEW.OWNER_ID := 'OWN' || stall_owner_seq.NEXTVAL;
END;
/

CREATE OR REPLACE TRIGGER stalls_id_trigger
BEFORE INSERT ON STALLS
FOR EACH ROW
BEGIN
  :NEW.STALL_ID := 'ST' || stalls_seq.NEXTVAL;
END;
/

CREATE OR REPLACE TRIGGER food_id_trigger
BEFORE INSERT ON FOOD
FOR EACH ROW
BEGIN
  :NEW.FOOD_ID := 'FOOD' || food_seq.NEXTVAL;
END;
/

CREATE OR REPLACE TRIGGER waste_id_trigger
BEFORE INSERT ON WASTE
FOR EACH ROW
```

```
BEGIN
  :NEW.WASTE_ID := 'W' || waste_seq.NEXTVAL;
END;
/

CREATE OR REPLACE TRIGGER reason_id_trigger
BEFORE INSERT ON REASON
FOR EACH ROW
BEGIN
  :NEW.REASON_ID := 'R' || reason_seq.NEXTVAL;
END;
/
```

Triggers were referenced from [2]

Appendix 1.2: SQL Script for Inventory Module

```
CREATE TABLE CATEGORY (

   CATEGORY_ID CHAR(5),
   CONSTRAINT PK_CATEGORY_ID PRIMARY KEY(CATEGORY_ID),

   CATEGORY_TYPE VARCHAR2(30) NOT NULL

);

CREATE TABLE STOR_SUPERVISOR (

   STOR_SUPERVISOR_ID VARCHAR2(15),
   CONSTRAINT PK_STOR_SUPERVISOR_ID PRIMARY KEY(STOR_SUPERVISOR_ID),

   SUPERVISOR_NAME VARCHAR2(30) NOT NULL,

   SUPERVISOR_CONTACT_NUMBER NUMBER(20,0) NOT NULL,

   SUPERVISOR_EMAIL VARCHAR2(50) NOT NULL

);

CREATE TABLE SUPPLIER (

   SUPPLIER_ID VARCHAR2(15),
   CONSTRAINT PK_SUPPLIER_ID PRIMARY KEY(SUPPLIER_ID),

   SUPPLIER_NAME VARCHAR2(30) NOT NULL,

   SUPPLIER_CONTACT_NUMBER NUMBER(20,0) NOT NULL,

   COMPANY_NAME VARCHAR2(100) NOT NULL,

   SUPPLIER_DELIVERY_SCHEDULE VARCHAR2(70) NOT NULL

);

CREATE TABLE PURCHASE_ORDER (

   ORDER_NUMBER VARCHAR2(10),
   CONSTRAINT PK_ORDER_NUMBER PRIMARY KEY(ORDER_NUMBER),

   SUPPLIER_ID VARCHAR2(15),
```

```
    CONSTRAINT FK_SUPPLIER_ID FOREIGN KEY(SUPPLIER_ID) REFERENCES SUPPLIER(SUPPLIER_ID),

    ORDER_DATE DATE NOT NULL,

    TOTAL_COST NUMBER(7,2) NOT NULL,
    CONSTRAINT CK_TOTAL_COST CHECK(TOTAL_COST >0),

    TRANSACTION_METHOD VARCHAR2(30) NOT NULL,

    TRACKING_ORDER_STATUS VARCHAR2(10) NOT NULL,

    DATE_OF_DELIVERY DATE

);

CREATE TABLE BATCH (

    BATCH_ID VARCHAR2(15),
    CONSTRAINT PK_BATCH_ID PRIMARY KEY(BATCH_ID),

    ORDER_NUMBER VARCHAR2(10),
    CONSTRAINT FK_ORDER_NUMBER FOREIGN KEY(ORDER_NUMBER) REFERENCES PURCHASE_ORDER (ORDER_NUMBER)

);

CREATE TABLE FOOD_STORAGE_ROOM (

    ROOM_NUMBER VARCHAR2(10),
    CONSTRAINT PK_ROOM_NUMBER PRIMARY KEY(ROOM_NUMBER),

    STOR_LOCATION VARCHAR2(30) NOT NULL,

    STOR_SUPERVISOR_ID VARCHAR2(15),
    CONSTRAINT FK_SUPERVISOR_ID FOREIGN KEY(STOR_SUPERVISOR_ID) REFERENCES STOR_SUPERVISOR(STOR_SUPERVISOR_ID),

    STALL_ID VARCHAR2(15),
    CANTEEN_ID VARCHAR2(15),
    CONSTRAINT FK_STALL FOREIGN KEY(STALL_ID, CANTEEN_ID) REFERENCES STALLS(STALL_ID, CANTEEN_ID)
```

```
);

CREATE TABLE SHELF (

   SHELF_NUMBER CHAR(5),
   CONSTRAINT PK_SHELF_NUMBER PRIMARY KEY(SHELF_NUMBER),

   ROOM_NUMBER VARCHAR2(10),
   CONSTRAINT FK_ROOM_NUMBER FOREIGN KEY(ROOM_NUMBER) REFERENCES
FOOD_STORAGE_ROOM(ROOM_NUMBER),

   SHELF_TYPE VARCHAR2(20) NOT NULL,

   SHELF_CAPACITY NUMBER(5,0) NOT NULL,
   CONSTRAINT CK_SHELF_CAPACITY CHECK(SHELF_CAPACITY >0)

);

CREATE TABLE FOOD_STORAGE (

   FS_ITEM_ID VARCHAR2(15),
   CONSTRAINT PK_FS_ITEM_ID PRIMARY KEY(FS_ITEM_ID),

   FS_ITEM_NAME VARCHAR2(30) NOT NULL,

   FS_QUANTITY NUMBER(5,0) NOT NULL,
   CONSTRAINT CK_FS_QUANTITY CHECK(FS_QUANTITY>0),

   FS_FOOD_TYPE VARCHAR2(20) NOT NULL,

   FS_EXPIRY_DATE DATE NOT NULL,

   FS_UNIT_COST NUMBER(7,2) NOT NULL,
   CONSTRAINT CK_FS_UNIT_COST CHECK(FS_UNIT_COST >0),

   CATEGORY_ID CHAR(5),
   CONSTRAINT FK_CATEGORY_ID FOREIGN KEY(CATEGORY_ID) REFERENCES
CATEGORY(CATEGORY_ID),

   BATCH_ID VARCHAR2(15),
   CONSTRAINT FK_BATCH_ID FOREIGN KEY(BATCH_ID) REFERENCES BATCH(BATCH_ID),

   SHELF_NUMBER CHAR(5),
```

```
    CONSTRAINT FK_SHELF_ID FOREIGN KEY(SHELF_NUMBER) REFERENCES
SHELF(SHELF_NUMBER)

);
```

Appendix 1.3: Financial Module

L

Appendix 1.4: Analytical Module

```
CREATE TABLE WASTE_INVENTORY (
   FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
   WASTE_DATE DATE NOT NULL,
   MASS FLOAT NOT NULL,
   PRIMARY KEY (FOOD_WASTE_TYPE),
   CONSTRAINT CHK_WASTE_DATE_FORMAT CHECK (WASTE_DATE = TO_DATE(TO_CHAR(WASTE_DATE, 'DD/MM/YYYY'), 'DD/MM/YYYY'))
);

CREATE TABLE STATISTICS (
   FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
   MEAN_MASS FLOAT,
   MODE_MASS FLOAT,
   MEDIAN_MASS FLOAT,
   FIRST_QUARTILE FLOAT,
   THIRD_QUARTILE FLOAT,
   MONTHYEAR DATE NOT NULL,
   PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
   CONSTRAINT FOOD_WASTE_TYPE_FK FOREIGN KEY (FOOD_WASTE_TYPE) REFERENCES WASTE_INVENTORY(FOOD_WASTE_TYPE)
);

CREATE TABLE CURRENT_PREDICTION (
   MONTHYEAR DATE NOT NULL,
   PREDICTED_MONTHYEAR DATE NOT NULL,
   FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
   GRADIENT_6M FLOAT,
   INTERCEPT_6M FLOAT,
```

```
    GRADIENT_12M FLOAT,
    INTERCEPT_12M FLOAT,
    GRADIENT_18M FLOAT,
    INTERCEPT_18M FLOAT,
    PREDICTED_MEAN_MASS_6M FLOAT,
    PREDICTED_MEAN_MASS_12M FLOAT,
    PREDICTED_MEAN_MASS_18M FLOAT,
    PRIMARY KEY (PREDICTED_MONTHYEAR, FOOD_WASTE_TYPE),
    CONSTRAINT MONTHYEAR_FOOD_WASTE_TYPE_FK FOREIGN KEY (MONTHYEAR, FOOD_WASTE_TYPE) REFERENCES STATISTICS (MONTHYEAR, FOOD_WASTE_TYPE)
);

CREATE TABLE PAST_PREDICTION (
    PREDICTED_MONTHYEAR DATE NOT NULL,
    FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
    GRADIENT_6M FLOAT,
    INTERCEPT_6M FLOAT,
    GRADIENT_12M FLOAT,
    INTERCEPT_12M FLOAT,
    GRADIENT_18M FLOAT,
    INTERCEPT_18M FLOAT,
    PREDICTED_MEAN_MASS_6M FLOAT,
    PREDICTED_MEAN_MASS_12M FLOAT,
    PREDICTED_MEAN_MASS_18M FLOAT,
    PRIMARY KEY (PREDICTED_MONTHYEAR, FOOD_WASTE_TYPE)
);

CREATE TABLE LAST_MONTH (
    MONTHYEAR DATE NOT NULL,  -- MM/YYYY format
```

```
    FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
    LAST_MONTH_MEAN_MASS FLOAT NOT NULL,
    PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
    CONSTRAINT LAST_MONTH_FK FOREIGN KEY (MONTHYEAR, FOOD_WASTE_TYPE) REFERENCES STATISTICS(MONTHYEAR, FOOD_WASTE_TYPE)
);

CREATE TABLE CURRENT_MONTH (
    MONTHYEAR DATE NOT NULL,
    FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
    CURRENT_MONTH_MEAN_MASS FLOAT NOT NULL,
    PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
    CONSTRAINT CURRENT_MONTH_FK FOREIGN KEY (MONTHYEAR, FOOD_WASTE_TYPE) REFERENCES STATISTICS(MONTHYEAR, FOOD_WASTE_TYPE)
);

CREATE TABLE DIFFERENCE (
    MONTHYEAR DATE NOT NULL,
    FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
    ABSOLUTE_DIFFERENCE FLOAT,
    PERCENTAGE_DIFFERENCE FLOAT,
    CHANGE_TYPE VARCHAR2(10) NOT NULL,
    CONSTRAINT CHANGE_TYPE_CK CHECK (CHANGE_TYPE IN ('INCREASE', 'DECREASE')),
-- Ensure only 'INCREASE' or 'DECREASE'
    PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
    CONSTRAINT DIFFERENCE_MONTHYEAR_FOOD_WASTE_TYPE_FK FOREIGN KEY (MONTHYEAR, FOOD_WASTE_TYPE) REFERENCES STATISTICS (MONTHYEAR, FOOD_WASTE_TYPE)
);
```

```
CREATE TABLE WASTE_TREND (
   MONTHYEAR DATE NOT NULL,
   FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
   RELATIVE_STRENGTH FLOAT,
   RELATIVE_STRENGTH_INDEX FLOAT,
   PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
   CONSTRAINT WASTE_TREND_FK FOREIGN KEY (MONTHYEAR, FOOD_WASTE_TYPE) REFERENCES DIFFERENCE(MONTHYEAR, FOOD_WASTE_TYPE)
);

CREATE TABLE DECREASE (
   MONTHYEAR DATE NOT NULL,  -- MM/YYYY format
   FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
   LAST_18M_MEAN_DECREASE FLOAT,
   PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
   CONSTRAINT DECREASE_FK FOREIGN KEY (MONTHYEAR, FOOD_WASTE_TYPE) REFERENCES DIFFERENCE(MONTHYEAR, FOOD_WASTE_TYPE)
);

CREATE TABLE INCREASE (
   MONTHYEAR DATE NOT NULL,
   FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
   LAST_18M_MEAN_INCREASE FLOAT,
   PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
   CONSTRAINT INCREASE_FK FOREIGN KEY (MONTHYEAR, FOOD_WASTE_TYPE) REFERENCES DIFFERENCE(MONTHYEAR, FOOD_WASTE_TYPE)
);

CREATE TABLE CARBON_EMISSIONS_PER_KG (
   FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
```

```
    CARBON_EMISSION_PER_KG FLOAT NOT NULL,
    PRIMARY KEY (FOOD_WASTE_TYPE)
);

CREATE TABLE CARBON_EMISSIONS (
    MONTHYEAR DATE NOT NULL,
    FOOD_WASTE_TYPE VARCHAR2(20) NOT NULL,
    CARBON_EMISSIONS FLOAT,
    PRIMARY KEY (MONTHYEAR, FOOD_WASTE_TYPE),
    CONSTRAINT CARBON_EMISSIONS_FK FOREIGN KEY (FOOD_WASTE_TYPE)
REFERENCES CARBON_EMISSIONS_PER_KG(FOOD_WASTE_TYPE)
);

CREATE OR REPLACE TRIGGER UPDATE_STATISTICS
AFTER INSERT OR UPDATE OF MASS ON WASTE_INVENTORY
FOR EACH ROW
DECLARE
    vMonthYear DATE;
    vMassCount INT;
    vLastDay INT;
BEGIN
    vMonthYear := TRUNC(:NEW.WASTE_DATE, 'MM');

    SELECT COUNT(DISTINCT TO_CHAR(WASTE_DATE, 'DD'))
    INTO vMassCount
    FROM WASTE_INVENTORY
    WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear;

    SELECT EXTRACT(DAY FROM LAST_DAY(:NEW.WASTE_DATE))
```

```
    INTO vLastDay
    FROM DUAL;

    IF vMassCount = vLastDay THEN
      INSERT INTO STATISTICS (FOOD_WASTE_TYPE, MONTHYEAR, MEAN_MASS, MODE_MASS, MEDIAN_MASS, FIRST_QUARTILE, THIRD_QUARTILE)
      VALUES (
        :NEW.FOOD_WASTE_TYPE,
        vMonthYear,
        (SELECT AVG(MASS) FROM WASTE_INVENTORY WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear),
        (SELECT MASS FROM (
          SELECT MASS, COUNT(MASS) AS FREQUENCY, RANK() OVER (ORDER BY COUNT(MASS) DESC) AS RANK
          FROM WASTE_INVENTORY WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear
          GROUP BY MASS
        ) WHERE RANK = 1),
        (SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY MASS) FROM WASTE_INVENTORY WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear),
        (SELECT PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY MASS) FROM WASTE_INVENTORY WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear),
        (SELECT PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY MASS) FROM WASTE_INVENTORY WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear)
      );
    END IF;

END;

CREATE OR REPLACE TRIGGER UPDATE_PREDICTION_PARAMETERS
AFTER INSERT OR UPDATE ON STATISTICS
FOR EACH ROW
```

```
DECLARE
    vCurrentMonth DATE;
    vStartDate DATE;
    vEndDate DATE;
    vMeanMass SYS_REFCURSOR;
    vMonthsCount INT;
    vMeanMassList SYS.ODCINUMBERLIST := SYS.ODCINUMBERLIST();
    vMonthList SYS.ODCINUMBERLIST := SYS.ODCINUMBERLIST();
    vIndex INT := 1;
    vGradient_6M FLOAT;
    vIntercept_6M FLOAT;
    vGradient_12M FLOAT;
    vIntercept_12M FLOAT;
    vGradient_18M FLOAT;
    vIntercept_18M FLOAT;
    vSumX FLOAT := 0;
    vSumY FLOAT := 0;
    vSumXY FLOAT := 0;
    vSumXX FLOAT := 0;
    vN FLOAT;
    vCount NUMBER;
    vPredictedMeanMass_6M FLOAT;
    vPredictedMeanMass_12M FLOAT;
    vPredictedMeanMass_18M FLOAT;
BEGIN
    vCurrentMonth := TRUNC(:NEW.MONTHYEAR, 'MM');

    OPEN vMeanMass FOR
```

```
    SELECT MEAN_MASS, MONTHYEAR
    FROM STATISTICS
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE AND MONTHYEAR >= ADD_MONTHS(vCurrentMonth, -5) AND MONTHYEAR <= vCurrentMonth
    ORDER BY MONTHYEAR;

    LOOP
        FETCH vMeanMass INTO vMeanMassList(vIndex), vMonthList(vIndex);
        EXIT WHEN vMeanMass%NOTFOUND;
        vIndex := vIndex + 1;
    END LOOP;
    CLOSE vMeanMass;

    vN := vIndex - 1;
    FOR i IN 1..vN LOOP
        vSumX := vSumX + i;
        vSumY := vSumY + vMeanMassList(i);
        vSumXY := vSumXY + (i * vMeanMassList(i));
        vSumXX := vSumXX + (i * i);
    END LOOP;

    vGradient_6M := (vN * vSumXY - vSumX * vSumY) / (vN * vSumXX - vSumX * vSumX);
    vIntercept_6M := (vSumY - vGradient_6M * vSumX) / vN;

    vIndex := 1;
    vSumX := 0;
    vSumY := 0;
    vSumXY := 0;
    vSumXX := 0;
```

```
    OPEN vMeanMass FOR
    SELECT MEAN_MASS, MONTHYEAR
    FROM STATISTICS
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE AND MONTHYEAR >= ADD_MONTHS(vCurrentMonth, -11) AND MONTHYEAR <= vCurrentMonth
    ORDER BY MONTHYEAR;

    LOOP
        FETCH vMeanMass INTO vMeanMassList(vIndex), vMonthList(vIndex);
        EXIT WHEN vMeanMass%NOTFOUND;
        vIndex := vIndex + 1;
    END LOOP;
    CLOSE vMeanMass;

    vN := vIndex - 1;
    FOR i IN 1..vN LOOP
        vSumX := vSumX + i;
        vSumY := vSumY + vMeanMassList(i);
        vSumXY := vSumXY + (i * vMeanMassList(i));
        vSumXX := vSumXX + (i * i);
    END LOOP;

    vGradient_12M := (vN * vSumXY - vSumX * vSumY) / (vN * vSumXX - vSumX * vSumX);
    vIntercept_12M := (vSumY - vGradient_12M * vSumX) / vN;

    vIndex := 1;
    vSumX := 0;
    vSumY := 0;
```

```
    vSumXY := 0;
    vSumXX := 0;

    OPEN vMeanMass FOR
    SELECT MEAN_MASS, MONTHYEAR
    FROM STATISTICS
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE AND MONTHYEAR >= ADD_MONTHS(vCurrentMonth, -17) AND MONTHYEAR <= vCurrentMonth
    ORDER BY MONTHYEAR;

    LOOP
        FETCH vMeanMass INTO vMeanMassList(vIndex), vMonthList(vIndex);
        EXIT WHEN vMeanMass%NOTFOUND;
        vIndex := vIndex + 1;
    END LOOP;
    CLOSE vMeanMass;

    vN := vIndex - 1;
    FOR i IN 1..vN LOOP
        vSumX := vSumX + i;
        vSumY := vSumY + vMeanMassList(i);
        vSumXY := vSumXY + (i * vMeanMassList(i));
        vSumXX := vSumXX + (i * i);
    END LOOP;

    vGradient_18M := (vN * vSumXY - vSumX * vSumY) / (vN * vSumXX - vSumX * vSumX);
    vIntercept_18M := (vSumY - vGradient_18M * vSumX) / vN;

    vPredictedMeanMass_6M := vGradient_6M * (vN + 1) + vIntercept_6M;
```

```
vPredictedMeanMass_12M := vGradient_12M * (vN + 1) + vIntercept_12M;
vPredictedMeanMass_18M := vGradient_18M * (vN + 1) + vIntercept_18M;

SELECT COUNT(*)
INTO vCount
FROM CURRENT_PREDICTION;

IF vCount > 0 THEN
   DELETE FROM CURRENT_PREDICTION;
END IF;

INSERT INTO CURRENT_PREDICTION (
   MONTHYEAR,
   PREDICTED_MONTHYEAR,
   FOOD_WASTE_TYPE,
   GRADIENT_6M,
   INTERCEPT_6M,
   GRADIENT_12M,
   INTERCEPT_12M,
   GRADIENT_18M,
   INTERCEPT_18M,
   PREDICTED_MEAN_MASS_6M,
   PREDICTED_MEAN_MASS_12M,
   PREDICTED_MEAN_MASS_18M
)
VALUES (
   vCurrentMonth,
   ADD_MONTHS(vCurrentMonth, 1),
```

```
        :NEW.FOOD_WASTE_TYPE,
        vGradient_6M,
        vIntercept_6M,
        vGradient_12M,
        vIntercept_12M,
        vGradient_18M,
        vIntercept_18M,
        vPredictedMeanMass_6M,
        vPredictedMeanMass_12M,
        vPredictedMeanMass_18M
    );

END;

CREATE OR REPLACE TRIGGER UPDATE_PAST_PREDICTION
AFTER INSERT OR UPDATE ON CURRENT_PREDICTION
FOR EACH ROW
BEGIN
    INSERT INTO PAST_PREDICTION (
        PREDICTED_MONTHYEAR,
        FOOD_WASTE_TYPE,
        GRADIENT_6M,
        INTERCEPT_6M,
        GRADIENT_12M,
        INTERCEPT_12M,
        GRADIENT_18M,
        INTERCEPT_18M,
        PREDICTED_MEAN_MASS_6M,
```

```
    PREDICTED_MEAN_MASS_12M,
    PREDICTED_MEAN_MASS_18M
  )
  VALUES (
    :NEW.PREDICTED_MONTHYEAR,
    :NEW.FOOD_WASTE_TYPE,
    :NEW.GRADIENT_6M,
    :NEW.INTERCEPT_6M,
    :NEW.GRADIENT_12M,
    :NEW.INTERCEPT_12M,
    :NEW.GRADIENT_18M,
    :NEW.INTERCEPT_18M,
    :NEW.PREDICTED_MEAN_MASS_6M,
    :NEW.PREDICTED_MEAN_MASS_12M,
    :NEW.PREDICTED_MEAN_MASS_18M
  );
END;

CREATE OR REPLACE TRIGGER UPDATE_LAST_MONTH_MEAN_MASS
AFTER INSERT OR UPDATE ON STATISTICS
FOR EACH ROW
DECLARE
  vLastMonth DATE;
  vLastMonthMeanMass FLOAT;
BEGIN
  SELECT ADD_MONTHS(TRUNC(MAX(MONTHYEAR), 'MM'), -1)
  INTO vLastMonth
  FROM STATISTICS
```

```
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;

    SELECT AVG(MEAN_MASS)
    INTO vLastMonthMeanMass
    FROM STATISTICS
    WHERE MONTHYEAR = vLastMonth
      AND FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;

    UPDATE LAST_MONTH
    SET LAST_MONTH_MEAN_MASS = vLastMonthMeanMass
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;
END;

CREATE OR REPLACE TRIGGER UPDATE_CURRENT_MONTH_MEAN_MASS
AFTER INSERT OR UPDATE ON STATISTICS
FOR EACH ROW
DECLARE
    vCurrentMonth DATE;
    vCurrentMonthMeanMass FLOAT;
BEGIN
    SELECT TRUNC(MAX(MONTHYEAR), 'MM')
    INTO vCurrentMonth
    FROM STATISTICS
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;

    SELECT AVG(MEAN_MASS)
    INTO vCurrentMonthMeanMass
    FROM STATISTICS
```

```
    WHERE MONTHYEAR = vCurrentMonth
      AND FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;

    UPDATE CURRENT_MONTH
    SET CURRENT_MONTH_MEAN_MASS = vCurrentMonthMeanMass
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;
END;

CREATE OR REPLACE TRIGGER UPDATE_DIFFERENCE
AFTER INSERT OR UPDATE ON CURRENT_MONTH
FOR EACH ROW
DECLARE
    vLastMonthMeanMass FLOAT;
    vCurrentMonthMeanMass FLOAT;
    vAbsoluteDifference FLOAT;
    vPercentageDifference FLOAT;
    vChangeType VARCHAR2(8);
BEGIN
    SELECT LAST_MONTH_MEAN_MASS
    INTO vLastMonthMeanMass
    FROM LAST_MONTH
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;

    SELECT CURRENT_MONTH_MEAN_MASS
    INTO vCurrentMonthMeanMass
    FROM CURRENT_MONTH
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;
```

```
    vAbsoluteDifference := vCurrentMonthMeanMass - vLastMonthMeanMass;
    vPercentageDifference := (vAbsoluteDifference / vLastMonthMeanMass) * 100;

    IF vAbsoluteDifference > 0 THEN
        vChangeType := 'INCREASE';
    ELSIF vAbsoluteDifference < 0 THEN
        vChangeType := 'DECREASE';
    ELSE
        vChangeType := 'NO CHANGE';
    END IF;

    UPDATE DIFFERENCE
    SET ABSOLUTE_DIFFERENCE = vAbsoluteDifference,
        PERCENTAGE_DIFFERENCE = vPercentageDifference,
        CHANGE_TYPE = vChangeType
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;
END;

CREATE OR REPLACE TRIGGER UPDATE_DECREASE
AFTER INSERT OR UPDATE ON DIFFERENCE
FOR EACH ROW
DECLARE
    vMeanDecrease FLOAT;
BEGIN
    SELECT AVG(ABSOLUTE_DIFFERENCE)
    INTO vMeanDecrease
    FROM (SELECT ABSOLUTE_DIFFERENCE
          FROM DIFFERENCE
```

```
        WHERE CHANGE_TYPE = 'DECREASE'
          AND FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE
        ORDER BY MONTHYEAR DESC)
    WHERE ROWNUM <= 18;

    UPDATE DECREASE
    SET LAST_18M_MEAN_DECREASE = vMeanDecrease
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;
END;

CREATE OR REPLACE TRIGGER UPDATE_INCREASE
AFTER INSERT OR UPDATE ON DIFFERENCE
FOR EACH ROW
DECLARE
    vMeanIncrease FLOAT;
BEGIN
    SELECT AVG(ABSOLUTE_DIFFERENCE)
    INTO vMeanIncrease
    FROM (SELECT ABSOLUTE_DIFFERENCE
        FROM DIFFERENCE
        WHERE CHANGE_TYPE = 'INCREASE'
          AND FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE
        ORDER BY MONTHYEAR DESC)
    WHERE ROWNUM <= 18;

    UPDATE INCREASE
    SET LAST_18M_MEAN_INCREASE = vMeanIncrease
    WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;
```

```
END;

CREATE OR REPLACE TRIGGER UPDATE_WASTE_TREND
AFTER INSERT OR UPDATE ON DIFFERENCE
FOR EACH ROW
DECLARE
   vGain FLOAT;
   vLoss FLOAT;
   vRS FLOAT;
   vRSI FLOAT;

BEGIN
   SELECT (SELECT LAST_18M_MEAN_INCREASE
         FROM INCREASE
         WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE
          AND MONTHYEAR = (SELECT MAX(MONTHYEAR) FROM INCREASE WHERE
FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE)),
        (SELECT LAST_18M_MEAN_DECREASE
         FROM DECREASE
         WHERE FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE
          AND MONTHYEAR = (SELECT MAX(MONTHYEAR) FROM DECREASE WHERE
FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE))
   INTO vGain, vLoss
   FROM DUAL;

   IF vGain IS NULL OR vLoss IS NULL THEN
      vRS := NULL;
      vRSI := NULL;
   ELSE
```

```
        vRS := vGain / vLoss;
        vRSI := 100 - (100 / (1 + vRS));
    END IF;

    UPDATE WASTE_TREND
    SET RELATIVE_STRENGTH = vRS,
        RELATIVE_STRENGTH_INDEX = vRSI
    WHERE MONTHYEAR = :NEW.MONTHYEAR
      AND FOOD_WASTE_TYPE = :NEW.FOOD_WASTE_TYPE;
END;

CREATE OR REPLACE TRIGGER UPDATE_CARBON_EMISSIONS
AFTER INSERT OR UPDATE ON WASTE_INVENTORY
FOR EACH ROW
DECLARE
    vMonthYear DATE;
    vTotalMass FLOAT;
    vCarbonEmissions FLOAT;
BEGIN
    vMonthYear := TRUNC(:NEW.WASTE_DATE, 'MM');

    SELECT NVL(SUM(MASS), 0)
    INTO vTotalMass
    FROM WASTE_INVENTORY
    WHERE TRUNC(WASTE_DATE, 'MM') = vMonthYear;

    SELECT NVL(SUM(WI.MASS * CE.CARBON_EMISSION_PER_KG), 0)
    INTO vCarbonEmissions
```

```
    FROM WASTE_INVENTORY WI
    JOIN CARBON_EMISSIONS_PER_KG CE
    ON WI.FOOD_WASTE_TYPE = CE.FOOD_WASTE_TYPE
    WHERE TRUNC(WI.WASTE_DATE, 'MM') = vMonthYear;

    MERGE INTO CARBON_EMISSIONS CE
    USING DUAL
    ON (CE.MONTHYEAR = vMonthYear)
    WHEN MATCHED THEN
        UPDATE SET
            CE.CARBON_EMISSIONS = vCarbonEmissions
    WHEN NOT MATCHED THEN
        INSERT (MONTHYEAR, FOOD_WASTE_TYPE, CARBON_EMISSIONS)
        VALUES (vMonthYear, :NEW.FOOD_WASTE_TYPE, vCarbonEmissions);

    COMMIT; -- Ensure changes are committed
END;
```

```
CREATE TABLE REVENUE ( REVENUE_ID INT PRIMARY KEY, SOURCE VARCHAR(255) NOT NULL, AMOUNT FLOAT NOT NULL CHECK (AMOUNT > 0.0), REVENUE_DATE DATE NOT NULL, REFERENCE VARCHAR(255) NOT NULL );

CREATE TABLE YEAR_LOSS_STALL (
    STALL_ID VARCHAR2(15),
    CANTEEN_ID VARCHAR2(15),
    WASTE_ID VARCHAR(15),
    WASTE_MASS DECIMAL(10, 2),
    PRICE_INGREDIENT_KG DECIMAL(10, 2),
```

```
  YEAR_STALL_LOSS DECIMAL(10, 2),
  YEAR INT NOT NULL,
  PRIMARY KEY (STALL_ID, CANTEEN_ID, WASTE_ID),
  FOREIGN KEY (STALL_ID,CANTEEN_ID) REFERENCES STALLS(STALL_ID,CANTEEN_ID),
  FOREIGN KEY (WASTE_ID) REFERENCES WASTE(WASTE_ID)
);
```

```
CREATE TABLE BUDGET ( BUDGET_ID INT PRIMARY KEY, YEAR INT NOT NULL, TOTAL FLOAT NOT NULL CHECK (TOTAL > 0.0), APPROVED_BY VARCHAR(255) );
```

```
CREATE TABLE EXPENSES (
  DETAIL VARCHAR(255),
  REASON VARCHAR(255),
  COST FLOAT NOT NULL CHECK (COST > 0.0),
  SUBMITTED_BY VARCHAR(255) NOT NULL,
  EXPENSES_DATE DATE NOT NULL,
  PRIMARY KEY (DETAIL, REASON, EXPENSES_DATE)
);
```

```
CREATE TABLE YEARLY_LOSS ( YEAR_LOSS INT PRIMARY KEY, FOOD_LOSS FLOAT NOT NULL CHECK (FOOD_LOSS > 0.0), COST FLOAT NOT NULL CHECK (COST > 0.0), TOTAL_LOSS INT NOT NULL CHECK (TOTAL_LOSS > 0.0) );
```

```
CREATE TABLE LOSS_FOOD_WASTE ( WASTE_DATE DATE NOT NULL, WASTE_ID VARCHAR2(15) PRIMARY KEY, WASTE_MASS FLOAT, PRICE_INGREDIENT_KG FLOAT NOT
```

```
NULL CHECK (PRICE_INGREDIENT_KG > 0.0), FOOD_LOSS FLOAT NOT NULL CHECK
(FOOD_LOSS > 0.0) );

CREATE TABLE LOSS_FOOD_WASTE (
  WASTE_DATE DATE NOT NULL,
  WASTE_ID VARCHAR2(15) NOT NULL,
  WASTE_MASS FLOAT NOT NULL CHECK(WASTE_MASS > 0.0),
  PRICE_INGREDIENT_KG FLOAT NOT NULL CHECK (PRICE_INGREDIENT_KG > 0.0),
  FOOD_LOSS FLOAT NOT NULL CHECK (FOOD_LOSS > 0.0),
  PRIMARY KEY (WASTE_DATE, WASTE_ID),
  FOREIGN KEY (WASTE_ID) REFERENCES WASTE(WASTE_ID),
  FOREIGN KEY (WASTE_MASS)  REFERENCES WASTE(WASTE_MASS)
);

CREATE TABLE YEAR_LOSS_STALL (
  STALL_ID VARCHAR2(15) NOT NULL,
  CANTEEN_ID VARCHAR2(15) NOT NULL,
  WASTE_ID VARCHAR2(15) NOT NULL,
  WASTE_MASS FLOAT NOT NULL CHECK (WASTE_MASS > 0.0),
  PRICE_INGREDIENT_KG FLOAT NOT NULL CHECK (PRICE_INGREDIENT_KG > 0.0),
  YEAR_STALL_LOSS FLOAT NOT NULL CHECK (YEAR_STALL_LOSS > 0.0),
  PRIMARY KEY (STALL_ID, CANTEEN_ID, WASTE_ID),
  FOREIGN KEY (WASTE_ID) REFERENCES WASTE(WASTE_ID),
  FOREIGN KEY (STALL_ID,CANTEEN_ID) REFERENCES STALLS(STALL_ID,CANTEEN_ID)
);
```

```
CREATE TABLE BUDGET_PREDICTION (
  PREDICTED_BUDGET_YEAR INT PRIMARY KEY,
  PREDICTED_MEAN_WASTE_MASS_12M DECIMAL(10, 2),
  TOTAL DECIMAL(10, 2),
  AMOUNT DECIMAL(10, 2),
  BUDGET_ID INT,
  COST DECIMAL(10, 2),
  PRED_BUDGET DECIMAL(10, 2),
  FOREIGN KEY(BUDGET_ID) REFERENCES BUDGET(BUDGET_ID)
);

-- Create a trigger to handle insert into LOSS_FOOD_WASTE
CREATE OR REPLACE TRIGGER trg_before_insert_loss_food_waste
BEFORE INSERT ON LOSS_FOOD_WASTE
FOR EACH ROW
DECLARE
  v_waste_mass FLOAT;
BEGIN
  -- Cursor to loop through all WASTE_IDs in FOOD_WASTE for the given WASTE_DATE
  FOR waste_rec IN (SELECT WASTE_ID
                    FROM FOOD_WASTE
                    WHERE WASTE_DATE = :NEW.WASTE_DATE) LOOP
    -- Check if WASTE_ID exists in WASTE table
    BEGIN
```

```
        SELECT WASTE_MASS INTO v_waste_mass
        FROM WASTE
        WHERE WASTE_ID = waste_rec.WASTE_ID;
      EXCEPTION
        WHEN NO_DATA_FOUND THEN
          -- Raise an error if no WASTE_ID or WASTE_MASS is found
          RAISE_APPLICATION_ERROR(-20001, 'No values found for WASTE_ID: ' ||
waste_rec.WASTE_ID || ' or WASTE_MASS.');
      END;

      -- Insert WASTE_ID into LOSS_FOOD_WASTE table
      INSERT INTO LOSS_FOOD_WASTE (WASTE_DATE, WASTE_ID, WASTE_MASS)
      VALUES (:NEW.WASTE_DATE, waste_rec.WASTE_ID, v_waste_mass);
    END LOOP;
END;
/




-- Create YEARLY_LOSS_FOOD_WASTE table
CREATE TABLE YEARLY_LOSS_FOOD_WASTE (
    YEAR_LOSS INT NOT NULL,
    WASTE_DATE DATE NOT NULL,
    WASTE_ID VARCHAR2(15) NOT NULL,
    FOREIGN KEY (YEAR_LOSS) REFERENCES YEARLY_LOSS(YEAR_LOSS),
```

```
  FOREIGN KEY (WASTE_DATE, WASTE_ID) REFERENCES LOSS_FOOD_WASTE(WASTE_DATE, WASTE_ID)
);

FOR SUMMATION FOR THE YEARLY_LOSS
CREATE OR REPLACE TRIGGER before_insert_yearly_loss
BEFORE INSERT ON YEARLY_LOSS
FOR EACH ROW
DECLARE
  totalCost DECIMAL(10, 2);
  totalFoodLoss DECIMAL(10, 2);
  startDate DATE;
  endDate DATE;
BEGIN
  totalCost := 0;
  totalFoodLoss := 0;
  startDate := TO_DATE(:NEW.YEAR_LOSS || '-01-01', 'YYYY-MM-DD');
  endDate := TO_DATE(:NEW.YEAR_LOSS || '-12-31', 'YYYY-MM-DD');

  -- Calculate total cost for the year
  SELECT SUM(COST) INTO totalCost
  FROM EXPENSES
  WHERE EXPENSES_DATE BETWEEN startDate AND endDate;

  -- Calculate total food loss for the year
  SELECT SUM(FOOD_LOSS) INTO totalFoodLoss
```

```
  FROM LOSS_FOOD_WASTE
  WHERE WASTE_DATE BETWEEN startDate AND endDate;

  :NEW.COST := totalCost;
  :NEW.FOOD_LOSS := totalFoodLoss;
  :NEW.TOTAL_LOSS := totalCost + totalFoodLoss;
END;
/
FOR THE LOSS_FOOD_WASTE
CREATE OR REPLACE TRIGGER trg_insert_waste_id
AFTER INSERT ON LOSS_FOOD_WASTE
FOR EACH ROW
BEGIN
  -- Insert all waste_ids from FOOD_WASTE where the waste_date matches the inserted waste_date
  FOR waste_rec IN (
    SELECT WASTE_ID
    FROM FOOD_WASTE
    WHERE WASTE_DATE = :NEW.WASTE_DATE
  ) LOOP
    INSERT INTO LOSS_FOOD_WASTE (WASTE_DATE, WASTE_ID)
    VALUES (:NEW.WASTE_DATE, waste_rec.WASTE_ID);
  END LOOP;
END;
/
```

```
CREATE OR REPLACE TRIGGER trg_update_waste_mass
AFTER INSERT OR UPDATE ON LOSS_FOOD_WASTE
FOR EACH ROW
BEGIN
  -- Update the waste_mass in LOSS_FOOD_WASTE for the given waste_id
  UPDATE LOSS_FOOD_WASTE
  SET WASTE_MASS = (
    SELECT WASTE_MASS
    FROM FOOD_WASTE
    WHERE WASTE_ID = :NEW.WASTE_ID
  )
  WHERE WASTE_ID = :NEW.WASTE_ID;
END;
/
CREATE OR REPLACE TRIGGER trg_calculate_food_loss
AFTER INSERT OR UPDATE ON LOSS_FOOD_WASTE
FOR EACH ROW
DECLARE
  food_loss_val FLOAT;
BEGIN
  -- Calculate food_loss
  food_loss_val := :NEW.PRICE_INGREDIENT_KG * :NEW.WASTE_MASS;

  -- Update the FOOD_LOSS in the LOSS_FOOD_WASTE table with the calculated value
  UPDATE LOSS_FOOD_WASTE
```

```
  SET FOOD_LOSS = food_loss_val
  WHERE WASTE_ID = :NEW.WASTE_ID;
END;
/

CREATE OR REPLACE TRIGGER trg_update_food_loss_on_price_change
AFTER UPDATE OF PRICE_INGREDIENT_KG ON LOSS_FOOD_WASTE
FOR EACH ROW
DECLARE
  food_loss_val FLOAT;
BEGIN
  -- Calculate food_loss
  food_loss_val := :NEW.PRICE_INGREDIENT_KG * :NEW.WASTE_MASS;

  -- Update the FOOD_LOSS in the LOSS_FOOD_WASTE table with the calculated value
  UPDATE LOSS_FOOD_WASTE
  SET FOOD_LOSS = food_loss_val
  WHERE WASTE_ID = :NEW.WASTE_ID
  AND WASTE_DATE = :NEW.WASTE_DATE;
END;

TRIGGER FOR THE BUDET PREDICON
CREATE OR REPLACE TRIGGER trg_update_budget_prediction
AFTER INSERT ON BUDGET_PREDICTION
FOR EACH ROW
DECLARE
```

```
  total_cost DECIMAL(10, 2);
  total_amount DECIMAL(10, 2);
BEGIN
  -- Calculate total cost from EXPENSES table for the given year
  SELECT COALESCE(SUM(COST), 0)
  INTO total_cost
  FROM EXPENSES
  WHERE EXTRACT(YEAR FROM EXPENSES_DATE) = :NEW.PREDICTED_BUDGET_YEAR;

  -- Calculate total amount from REVENUE table for the given year
  SELECT COALESCE(SUM(AMOUNT), 0)
  INTO total_amount
  FROM REVENUE
  WHERE EXTRACT(YEAR FROM REVENUE_DATE) = :NEW.PREDICTED_BUDGET_YEAR;

  -- Update the COST and AMOUNT fields in BUDGET_PREDICTION table
  UPDATE BUDGET_PREDICTION
  SET COST = total_cost,
    AMOUNT = total_amount
  WHERE PREDICTED_BUDGET_YEAR = :NEW.PREDICTED_BUDGET_YEAR;
END;
/
CREATE OR REPLACE TRIGGER trg_update_total_budget_prediction
AFTER INSERT ON BUDGET_PREDICTION
FOR EACH ROW
DECLARE
```

```
  total_value DECIMAL(10, 2);
BEGIN
  -- Fetch the TOTAL from BUDGET table for the given BUDGET_ID
  SELECT TOTAL
  INTO total_value
  FROM BUDGET
  WHERE BUDGET_ID = :NEW.BUDGET_ID;

  -- Update the TOTAL field in BUDGET_PREDICTION table
  UPDATE BUDGET_PREDICTION
  SET TOTAL = total_value
  WHERE PREDICTED_BUDGET_YEAR = :NEW.PREDICTED_BUDGET_YEAR
    AND BUDGET_ID = :NEW.BUDGET_ID;
END;
/
CREATE OR REPLACE TRIGGER trg_update_predicted_mean_waste_mass
AFTER INSERT ON BUDGET_PREDICTION
FOR EACH ROW
DECLARE
  max_predicted_mass DECIMAL(10, 2);
BEGIN
  -- Fetch the highest predicted_mean_waste_mass_12m from CURRENT_PREDICTION table for the given year
  SELECT MAX(PREDICTED_MEAN_WASTE_MASS_12M)
  INTO max_predicted_mass
  FROM CURRENT_PREDICTION
```

```
  WHERE EXTRACT(YEAR FROM MONTHYEAR) = :NEW.PREDICTED_BUDGET_YEAR;

  -- Update the PREDICTED_MEAN_WASTE_MASS_12M field in BUDGET_PREDICTION table
  UPDATE BUDGET_PREDICTION
  SET PREDICTED_MEAN_WASTE_MASS_12M = max_predicted_mass
  WHERE PREDICTED_BUDGET_YEAR = :NEW.PREDICTED_BUDGET_YEAR;
END;
/
CREATE OR REPLACE TRIGGER trg_calculate_pred_budget
AFTER INSERT OR UPDATE ON BUDGET_PREDICTION
FOR EACH ROW
DECLARE
  calculated_pred_budget DECIMAL(10, 2);
BEGIN
  -- Calculate the PRED_BUDGET using the specified algorithm
  calculated_pred_budget := (:NEW.PREDICTED_MEAN_WASTE_MASS_12M / 1000) *
(:NEW.AMOUNT - :NEW.COST + :NEW.TOTAL);

  -- Update the PRED_BUDGET field in the BUDGET_PREDICTION table
  UPDATE BUDGET_PREDICTION
  SET PRED_BUDGET = calculated_pred_budget
  WHERE PREDICTED_BUDGET_YEAR = :NEW.PREDICTED_BUDGET_YEAR
    AND BUDGET_ID = :NEW.BUDGET_ID;
END;
/
```

Triigger FOR YEAR_LOSS_STALL

```
-- Create the trigger
CREATE OR REPLACE TRIGGER trg_insert_waste_ids
AFTER INSERT ON YEAR_LOSS_STALL
FOR EACH ROW
DECLARE
  waste_id_val VARCHAR2(15);
BEGIN
  -- Cursor to fetch all WASTE_IDs
  FOR waste_rec IN (
    SELECT WASTE_ID
    FROM WASTE
  ) LOOP
    BEGIN
      -- Insert each WASTE_ID for the given YEAR, STALL_ID, and CANTEEN_ID
      INSERT INTO YEAR_LOSS_STALL (YEAR, STALL_ID, CANTEEN_ID, WASTE_ID)
      VALUES (:NEW.YEAR, :NEW.STALL_ID, :NEW.CANTEEN_ID, waste_rec.WASTE_ID);
    EXCEPTION
      WHEN DUP_VAL_ON_INDEX THEN
        -- Handle duplicate values, if necessary
        NULL;
    END;
  END LOOP;
END;
```

```
/
CREATE OR REPLACE TRIGGER trg_insert_waste_ids
AFTER INSERT ON YEAR_LOSS_STALL
FOR EACH ROW
DECLARE
  waste_mass_val DECIMAL(10, 2);
BEGIN
  -- Cursor to fetch all WASTE_IDs and corresponding WASTE_MASS
  FOR waste_rec IN (
    SELECT WASTE_ID, WASTE_MASS
    FROM WASTE
    WHERE CANTEEN_ID = :NEW.CANTEEN_ID AND STALL_ID = :NEW.STALL_ID
  ) LOOP
    BEGIN
      -- Insert each WASTE_ID for the given YEAR, STALL_ID, and CANTEEN_ID
      INSERT INTO YEAR_LOSS_STALL (YEAR, STALL_ID, CANTEEN_ID, WASTE_ID, WASTE_MASS)
      VALUES (:NEW.YEAR, :NEW.STALL_ID, :NEW.CANTEEN_ID, waste_rec.WASTE_ID, waste_rec.WASTE_MASS);
    EXCEPTION
      WHEN DUP_VAL_ON_INDEX THEN
        -- Handle duplicate values, if necessary
        NULL;
    END;
  END LOOP;
END;
```

```
/
CREATE OR REPLACE TRIGGER trg_update_year_stall_loss
BEFORE INSERT OR UPDATE ON YEAR_LOSS_STALL
FOR EACH ROW
DECLARE
  prev_year_stall_loss DECIMAL(10, 2);
BEGIN
  -- Initialize the previous year stall loss
  prev_year_stall_loss := 0;

  -- Fetch the previous year stall loss if exists
  SELECT NVL(SUM(YEAR_STALL_LOSS), 0)
  INTO prev_year_stall_loss
  FROM YEAR_LOSS_STALL
  WHERE STALL_ID = :NEW.STALL_ID
    AND CANTEEN_ID = :NEW.CANTEEN_ID
    AND WASTE_ID = :NEW.WASTE_ID
    AND YEAR < :NEW.YEAR;

  -- Calculate the new year stall loss
  :NEW.YEAR_STALL_LOSS := (:NEW.PRICE_INGREDIENT_KG * :NEW.WASTE_MASS) +
prev_year_stall_loss;
END;
/
CREATE OR REPLACE TRIGGER trg_revenue_id
BEFORE INSERT ON REVENUE
```

```
FOR EACH ROW
BEGIN
  -- Use the sequence to generate a new REVENUE_ID
  :NEW.REVENUE_ID := REVENUE_SEQ.NEXTVAL;
END;
/
```