Contents

student_name	*student_id*	*contribution*
HMRS Samaranayaka	*22867*	*documentation, coding python, all sql queries, data analysis, sql query writing for data analysis*
W K Harshani	*22683*	*data analysis*
PMCS Bandara	*22834*	*data dictionary, coding python*
WSM Dilshan	*22870*	*coding python*
KDNL Kandanaarachchi	*22864*	*coding python*
MDT Alwis	*22759*	*relationships between tables*
WMYM Silva	*22689*	*documentation, drawing er and schema diagrams*
MHNP Dasun	*22710*	*test case writing*
ASS Silva	*22955*	*er diagram, normalization details, schema*
BMLCRS Gunasekara	*22721*	*er diagram, normalization details, schema*

1. Project overview

1.1 Introduction

The development of a comprehensive database system for the supermarket located in *NSBM Green University*, *Homagama* is a project of utmost significance, borne out of an extensive study and thorough analysis of the supermarket's operations. Our project team, in collaboration with the supermarket staff and supermarket management, embarked on a comprehensive journey of observation and discussion within the supermarket to gather crucial insights into its daily activities and data management requirements.

1.2 Objectives

The primary objectives of this project, informed by our observations and discussions, are as follows:

- **Data Management and Storage:** Based on our observations of the supermarket's operations, we aim to create a robust and efficient database system that can securely store and manage a wide range of data related to customers, products, orders, employees, suppliers, sales, inventory, transactions, store locations, and more. This system will serve as the linchpin for all supermarket activities.

- **Improved Customer Engagement:** Our discussions with the supermarket's manager emphasized the importance of customer satisfaction and retention. Therefore, we will focus on tracking and managing customer information, including purchase history, loyalty points, and contact details, to provide personalized services, targeted marketing, and loyalty programs.

- **Inventory Control:** The manager highlighted the challenges related to inventory management. Our database system will provide real-time insights into product availability, stock quantities, and reordering thresholds to avoid stockouts, reduce carrying costs, and optimize inventory control.

- **Employee Management:** Our discussions underscored the need for efficient employee management. Therefore, we will efficiently manage employee information, including positions, experience levels, salaries, and performance, to ensure an optimal workforce contributing to the supermarket's success.

- **Sales and Performance Analysis:** The manager expressed the need for data-driven decisions regarding product pricing, supplier relationships, and employee performance. Our project will collect and analyze sales data to provide insights that can lead to increased profitability.

- **Financial Tracking:** The manager stressed the importance of recording and categorizing various financial transactions. Our system will help maintain financial data, such as sales, refunds, expenses, and income, facilitating accounting and auditing procedures.

- **Supplier Relations:** The discussions also highlighted the importance of supplier relationships. We will maintain an updated database of supplier information to ensure timely communication and efficient procurement of goods.

- **Store Management:** Our project will provide tools for efficient management of store locations, including store names, addresses, and contact information, to ensure seamless operations.

1.3 Significance

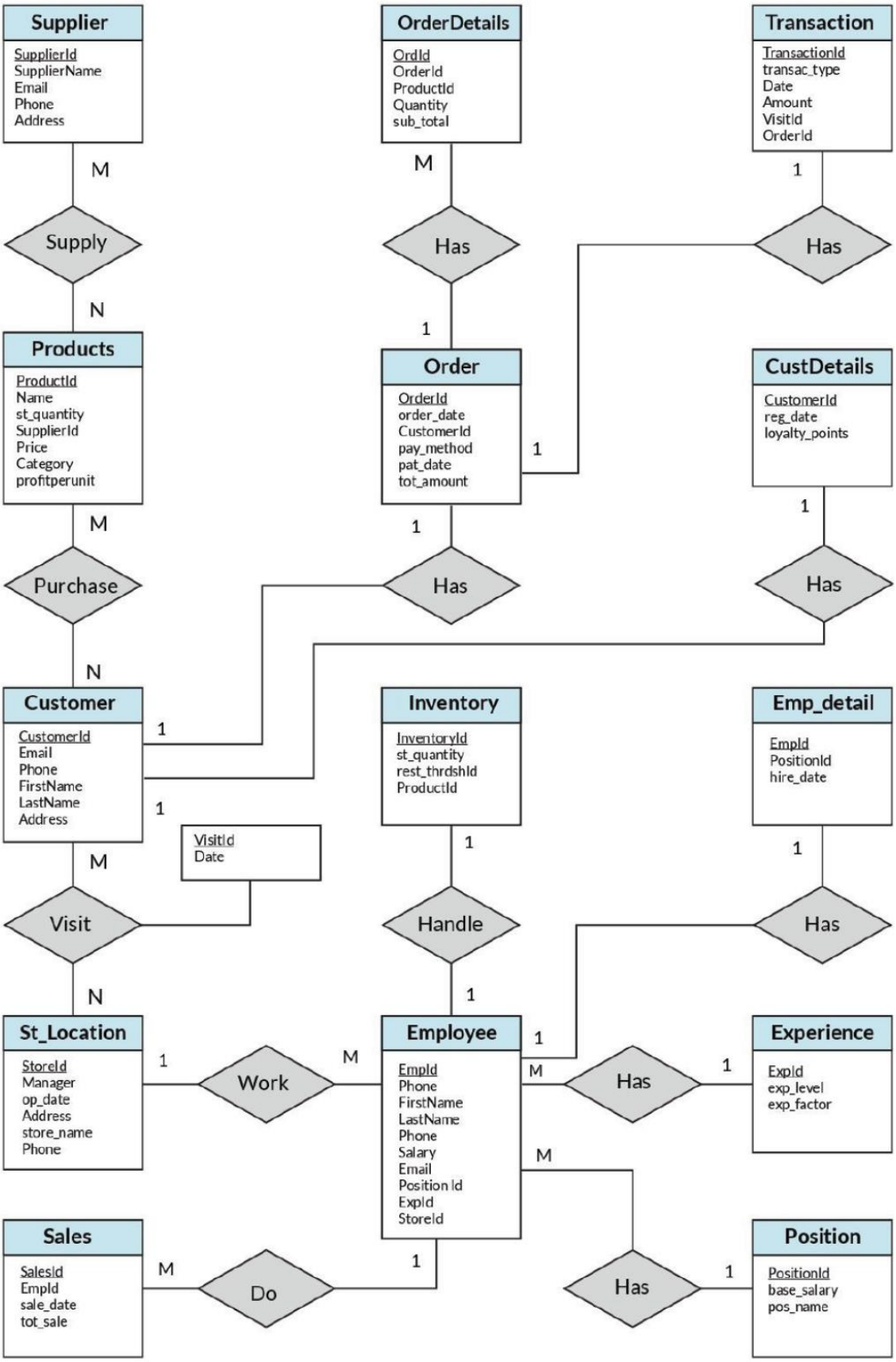
The significance of this project, informed by our observations and discussions with the supermarket manager, is profound:

- **Operational Efficiency:** The database system will address the specific operational challenges identified during our visits, reducing manual errors, and enhancing overall efficiency.
- **Enhanced Customer Experience:** Our discussions revealed the importance of customer satisfaction. The system will enable personalized services and loyalty programs to improve customer experience and retention.
- **Data-Driven Decision-Making:** The system will provide the data analytics needed to make informed decisions, addressing specific concerns raised by the manager related to inventory, pricing, and workforce management.
- **Cost Savings:** Efficient inventory and employee management will result in reduced carrying costs and labor expenses, aligning with the supermarket's financial goals.
- **Compliance and Accountability:** The system will ensure compliance with accounting and auditing standards, fostering transparency and accountability in financial transactions, a concern emphasized by the manager.
- **Competitive Advantage:** The development of this database system will position the supermarket as a technologically advanced, customer-centric, and operationally efficient organization, gaining a competitive edge in the university's retail market.

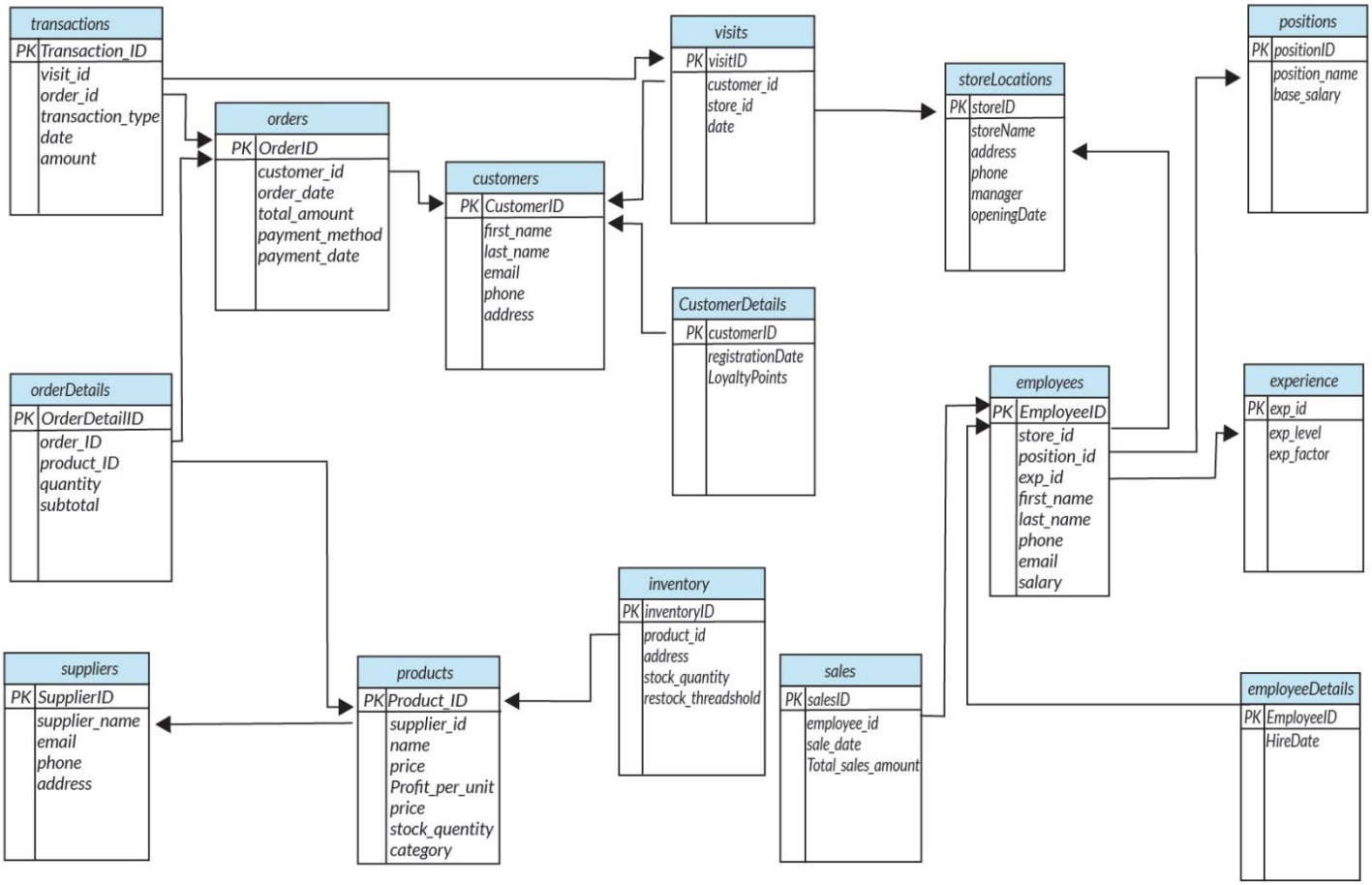
In conclusion, the insights gathered through our observations and discussions with the supermarket management have informed us of the project's objectives and significance. This project report will detail the design, development, and implementation of the database system, addressing specific needs and challenges identified during our visits and discussions, thus promoting the long-term success and competitiveness of the supermarket at *NSBM Green University, Homagama.*

2.Database design

2.1 Entity-Relationship Diagram (ERD)



2.2 Schema Diagram



2.3 Data Dictionary

1. Employee

attribute_name	data_type	description	constraint
employee_id	int	Unique identifier for each employee	primary key
first_name	varchar	First name of the employee	not null
last_name	varchar	Last name of the employee	not null
phone	varchar	Phone number of the employee	not null
email	varchar	Email address of the employee	unique, not null
salary	decimal	Salary of the employee	not null
position_id	int	Unique identifier for each position	foreign key
exp_id	int	Unique identifier for each experience entry	foreign key
hire_date	date	Hire date of the employee	not null

```
create table employee (
    employee_id int primary key,
    first_name varchar(255) not null,
    last_name varchar(255) not null,
    phone varchar(15) not null,
    email varchar(255) unique not null,
    salary decimal(10, 2) not null,
    position_id int,
    exp_id int,
    hire_date date not null,
    foreign key (position_id) references position(position_id),
    foreign key (exp_id) references experience(exp_id)
);
```

2. Experience

attribute_name	data_type	description	constraint
exp_id	int	Unique identifier for each experience entry.	primary key
exp_level	varchar	Experience level of employees	not null
exp_factor	decimal	Experience factors of employees	not null

```
create table experience (
    exp_id int primary key not null,
    exp_level varchar(255) not null,
    exp_factor decimal(10, 2) not null
);
```

3. Positions

attribute_name	data_type	description	constraint
position_id	int	Unique identifier for each position.	primary key
position_name	varchar	Title or name of the position.	not null
base_salary	decimal	Salary associated with the position.	not null

```
create table positions (
    position_id int primary key not null,
    position_name varchar(255) not null,
    base_salary decimal(10, 2) not null
);
```

4. StoreLocations

attribute_name	data_type	description	constraint
store_id	int	Unique identifier for each store location.	primary key
store_name	varchar	Name of the store or location.	not null
address	varchar	Street address of the store.	not null
phone	varchar	Phone number for the store.	not null
manager	varchar	Name of the manager responsible for the store location.	not null
opening_date	date	Date when the store was open	not null

```
create table store_locations (
    store_id int primary key not null,
    store_name varchar(255) not null,
    address varchar(255) not null,
    phone varchar(15) not null,
    manager varchar(255),
    opening_date date
);
```

5. Visits

attribute_name	data_type	description	constraint
visit_id	int	Unique identifier for each visit	primary key
date	date	Date of the visit	not null
customer_id	int	Unique identifier for each customer	foreign key
store_id	int	Unique identifier for each store location	foreign key

```
create table visits (
    visit_id int primary key,
    date date not null,
    customer_id int,
    store_id int,
    foreign key (customer_id) references customers(customer_id),
    foreign key (store_id) references storelocations(store_id)
);
```

6. Inventory

attribute_name	data_type	description	constraint
inventory_id	int	Unique identifier for each inventory item	primary key
stock_quantity	int	Quantity of the item in inventory.	not null
restock_threshold	int	Minimum amount that should have in stock	not null
product_id	int	Unique identifier for each product.	foreign key

```
create table inventory (
    inventory_id int primary key,
    stock_quantity int not null,
    restock_threshold int not null,
    product_id int,
    foreign key (product_id) references products(product_id)
);
```

7. Custormers

attribute_name	data_type	description	constraint
customer_id	int	Unique identifier for each customer.	primary key
first_name	varchar	First name of the customer.	not null
last_name	int	Last name of the customer.	not null
email	varchar	Email address of the customer.	unique,not null
phone	varchar	Phone number of the customer.	unique,not null
address	varchar	Mailing address of the customer.	not null

```
create table customers (
    customer_id int primary key,
    first_name varchar(255) not null,
    last_name varchar(255) not null,
    email varchar(255) unique not null,
    phone varchar(15) unique not null,
    address varchar(255) not null
);
```

8. Customer details

attribute_name	data_type	description	constraint
loyalty_points	int	Accumulated loyalty points for the customer	not null
registration_date	date	Customer registrated date	not null
customer_id	int	Unique identifier for each customer	foreign key

```
create table customer_details (
    customer_id int,
    registration_date date default current_date not null,
    loyalty_points int not null default 0,
    foreign key (customer_id) references customers(customer_id)
);
```

9. Transactions

attribute_name	data_type	description	constraint
transaction_id	int	Unique identifier for each transaction.	primary key
transaction_type	varchar	Transaction medium	not null
date	date time	Date and time of the transaction.	not null
amount	decimal	Amount of the transaction.	not null
order_id	int	Unique identifier for each order	foreign key
visit_id	int	Unique identifier for each visit.	foreign key

```
create table transactions (
    transaction_id int primary key,
    transaction_type varchar(255) not null,
    date datetime not null,
    amount decimal(10, 2) not null,
    order_id int,
    visit_id int,
    foreign key (order_id) references orders(order_id),
    foreign key (visit_id) references visits(visit_id)
);
```

10. Suppliers

attribute_name	data_type	description	constraint
supplier_id	int	Unique identifier for each supplier.	primary key
supplier_name	varchar	Name of the supplier	not null
email	varchar	Email address of the supplier	unique,not null
phone	varchar	Phone number of the supplier.	not null
address	varchar	Supplier's mailing address.	not null

```
create table suppliers (
    supplier_id int primary key not null,
    supplier_name varchar(255) not null,
    email varchar(255) not null,
    phone varchar(15) not null,
    address varchar(255) not null
);
```

11. Sales

attribute_name	data_type	description	constraint
sale_id	int	Unique identifier for each sale.	primary key
sale_date	date	Date and time of the sale.	not null
total_sales_amount	int	Total amount of the sale.	not null
employee_id	int	Unique identifier for each employee.	foreign key

```
create table sales (
    sale_id int primary key,
    sale_date date not null,
    total_sales_amount decimal(10, 2) not null,
    employee_id int,
    foreign key (employee_id) references employees(employee_id)
);
```

12. Orders

attribute_name	data_type	description	constraint
order_id	int	Unique identifier for each order.	primary key
order_date	date	Date when the order was placed.	not null
total_amount	decimal	Total amount of the order.	not null
payment_method	varchar	Payment medium(cash/card)	not null
payment_date	date	Date when the payment was placed.	not null
customer_id	int	Unique identifier for each customer	foreign key

```
create table orders (
    order_id int primary key,
    order_date date not null,
    total_amount decimal(10, 2) not null,
    payment_method varchar(255) not null,
    payment_date date not null,
    customer_id int,
    foreign key (customer_id) references customers(customer_id)
);
```

13. Order details

attribute_name	data_type	description	constraint
orderdetail_id	int	Unique identifier for each order detail	primary key
quantity	int	Quantity of the product in the order detail	not null
subtotal	int	Subtotal for the order detail	not null
order_id	int	Unique identifier for each order	foreign key
product_id	int	Unique identifier for each product	foreign key

```
create table orderdetails (
    orderdetail_id int primary key,
    quantity int not null,
    subtotal decimal(10, 2) not null,
    order_id int,
    product_id int,
    foreign key (order_id) references orders(order_id),
    foreign key (product_id) references products(product_id)
);
```

14. Products

attribute_name	data_type	description	constraint
productid	int	Unique identifier for each product.	primary key
name	varchar	Name of the product.	not null
category	varchar	Product category or type.	not null
price	decimal	Price of the product.	not null
stockquantity	int	Current stock quantity of the product.	not null
profit_per_unit	decimal	Profit gained by a unit	not null
supplierid	int	Unique identifier for each supplier	foreign key

```
create table products (
    product_id int primary key,
    name varchar(255) not null,
    category varchar(255) not null,
    price decimal(10, 2) not null,
    stock_quantity int not null,
    profit_per_unit decimal(10, 2) not null,
    supplier_id int,
    foreign key (supplier_id) references suppliers(supplier_id)
);
```

2.4 Normalization Details

Step 1: Identify the Functional Dependencies

Before we can start normalizing the tables, we need to identify the functional dependencies between attributes .Here are the functional dependencies for each table:

- Customers

customer_id	*first_name*	*last_name*	*email*	*phone*	*address*	*registration_date*	*loyalty_points*

- Products

product_id	*name*	*category*	*price*	*stock_quantity*	*supplierid*	*profit_per_unit*

- Orders

order_id	*customer_id*	*order_date*	*total_amount*	*payment_method*	*payment_date*

- OrderDetails

order_detail_id	*order_id*	*product_id*	*quantity*	*subtotal*

- Employees

employee_id	*position_id*	*exp_id*	*first_name*	*last_name*	*email*	*phone*	*hire_date*	*storeid*	*salary*

- Positions

position_id	*position_name*	*base_salary*

- Experience

exp_id	*exp_level*	*exp_factor*

- Sales

sale_id	*employee_id*	*sale_date*	*total_sales_amount*

- Inventory

inventory_id	*product_id*	*stock_quantity*	*restock_threshold*

- Suppliers

supplier_id	*supplier_name*	*email*	*phone*	*address*

- Transactions

transaction_id	*transaction_type*	*date*	*amount*	*order_id*	*visit_id*

- StoreLocations

store_id	*storename*	*address*	*phone*	*manager*	*opening_date*

- Visits

visit_id	*customer_id*	*date*	*store_id*

Step 2: Normalize the Tables

1st Normal Form (1NF)

To achieve 1NF, we need to ensure that each column contains atomic (*indivisible*) values, and each row is uniquely identifiable.

- *customers*
- *products*
- *orders*
- *order_details*
- *employees*
- *positions*
- *experience*
- *sales*
- *inventory*
- *suppliers*
- *transactions*
- *store_locations*
- *visits*

All tables are now in 1NF

2nd Normal Form (2NF)

To achieve 2NF, we need to ensure that there are no partial dependencies, meaning that non-key attributes are fully functionally dependent on the entire primary key.

- *customers*
- *products*
- *orders*
- *order_details*
- *employees*
- *positions*
- *experience*
- *sales*
- *inventory*
- *suppliers*
- *transactions*
- *store_locations*
- *visits*

All the tables are now in 2NF, meeting the requirements of 1NF and 2NF.

3rd Normal Form (3NF)

To achieve 3NF, we need to eliminate transitive dependencies, meaning that non-key attributes are not dependent on other non-key attributes.

- Customers

customerid	*firstname*	*lastname*	*email*	*phone*	*address*	*registrationdate*	*loyaltypoints*

RegistrationDate and LoyaltyPoints are transitively dependent on CustomerID. To remove this dependency, we can create a new table for CustomerDetails with CustomerID as the primary key and move RegistrationDate and LoyaltyPoints to that table.

Customers (3NF)

- **CustomerDetails** (New Table)

customerid	*registrationdate*	*loyaltypoints*

- **Customers** (Modified Table)

customerid	*firstname*	*lastname*	*email*	*phone*	*address*

In the modified Customers table, the RegistrationDate and LoyaltyPoints have been moved to a new table named CustomerDetails, which has a foreign key relationship with the Customers table.

- Products (3NF) - No transitive dependencies.

- Orders (3NF) - No transitive dependencies.

- OrderDetails (3NF) - No transitive dependencies.

- Employees (3NF)

employeeid	*position_id*	*exp_id*	*firstname*	*lastname*	*email*	*phone*	*hiredate*	*storeid*	*salary*

Employees

In the modified Employees table, the HireDate has been moved to a new table named EmployeeDetails, which also has a foreign key relationship with the Employees table.

- **EmployeeDetails** (New Table)

employeeid	*hiredate*

- **Employees** (Modified Table)

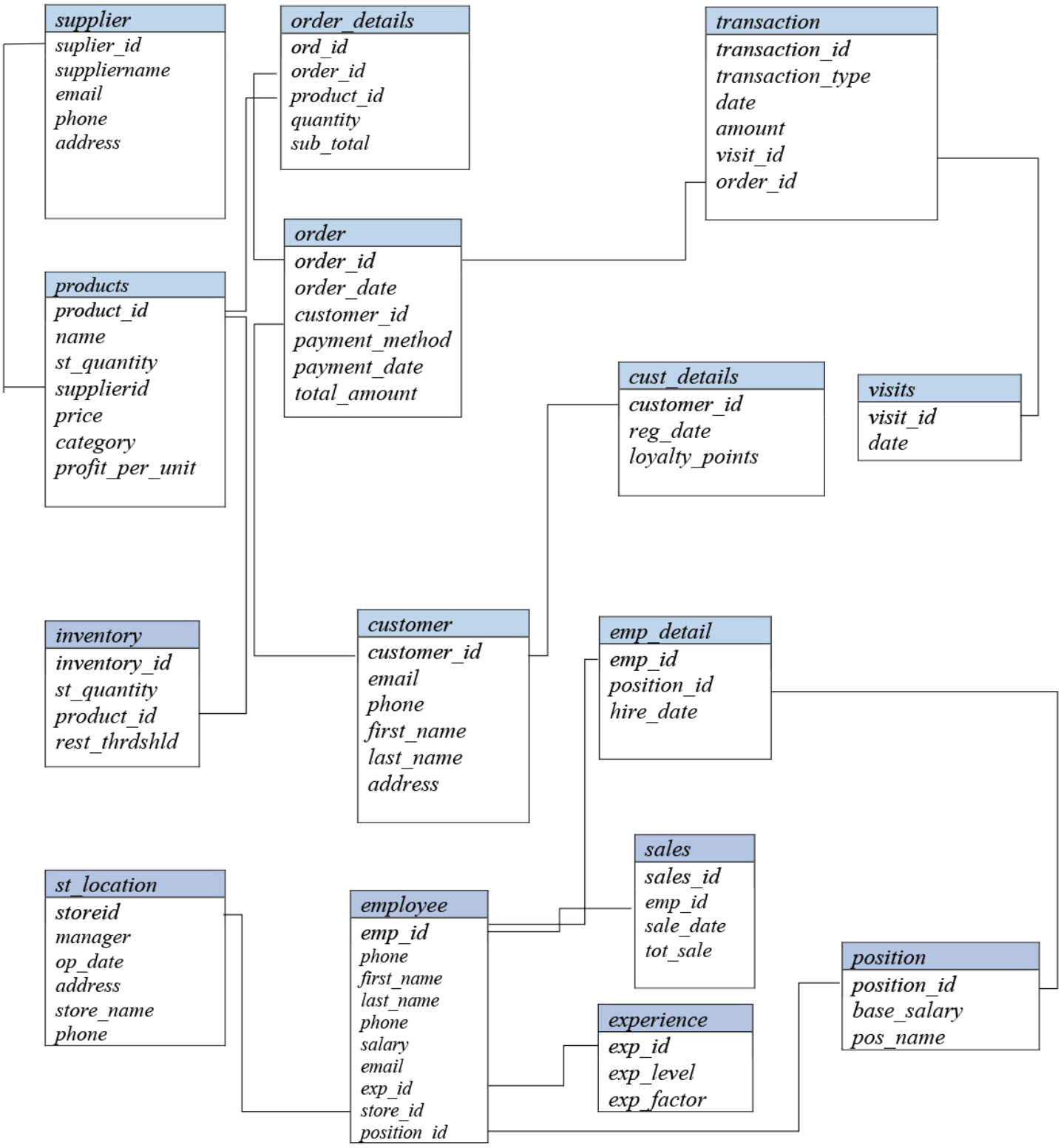
employeeid	*position_id*	*exp_id*	*firstname*	*lastname*	*email*	*phone*	*storeid*	*salary*

*** This separation of data eliminates the transitive dependencies in both the Customers and Employees tables, ensuring they are in 3NF.**

- Positions (3NF) - No transitive dependencies.
- Experience (3NF) - No transitive dependencies.
- Sales (3NF) - No transitive dependencies.
- Inventory (3NF) - No transitive dependencies.
- Suppliers (3NF) - No transitive dependencies.
- Transactions (3NF) - No transitive dependencies.
- StoreLocations (3NF) - No transitive dependencies.
- Visits (3NF) - No transitive dependencies.

***After applying 3NF, the Customers and Employees tables would have some modifications as described above. Other tables remain in 3NF.**

2.5 Logical Data Model



2.5 Relatioships

2.5.1 Customer – Order Relationship

- Description: The *"customer"* table is related to the *"order"* table in a *One-to-Many* Relationship. Each customer can have multiple orders, but each order is associated with only one customer.
- Entities Involved: *Customer, Order*
- Relationship Type: *One-to-Many*
- Foreign Key(s): The *"customer_id"* column in the *"order"* table reference the *"customer_id"* column in the "*customer*" table.
- Purpose: This relationship allows tracking orders for each customer.

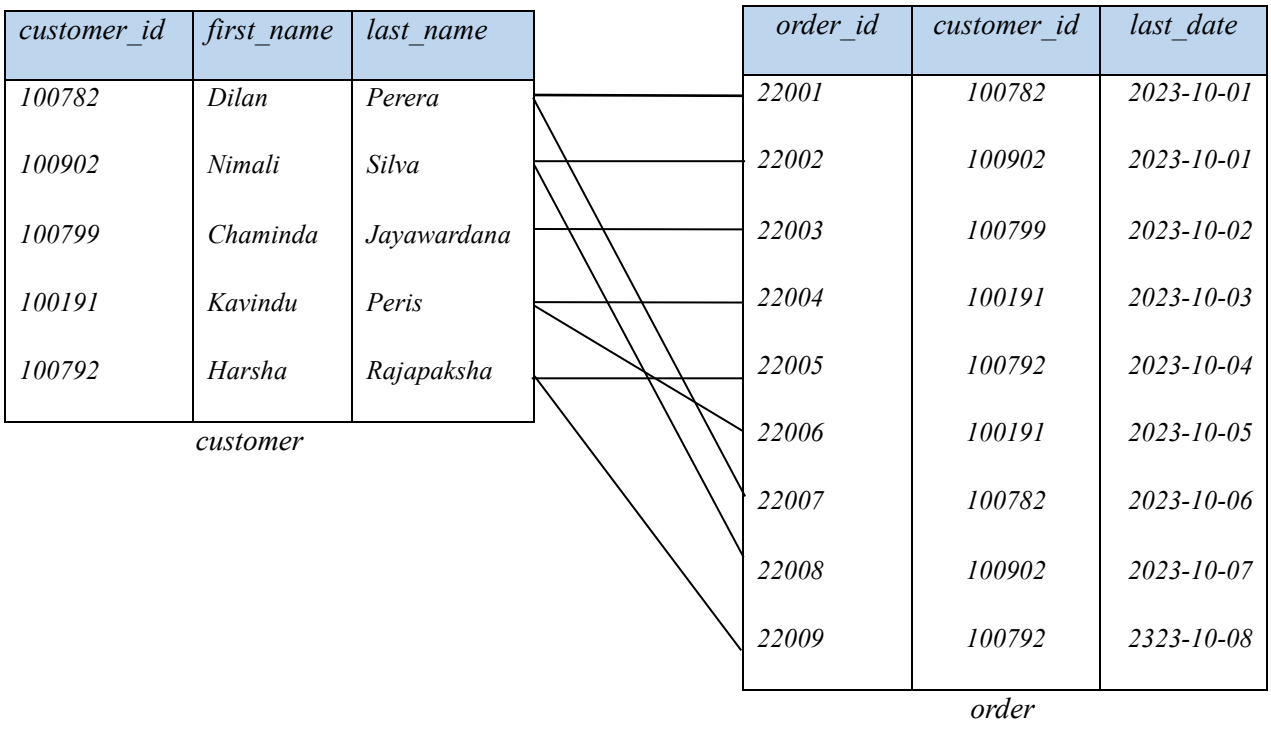


Figure 2.5.1 relationship between *customer* and *order*

2.5.2 Order – Order Details Relationship

- Description: The *"Order"* table is related to their *"Order_details"* table in *One-to-Many* relationship. Each Order has many order details but many order details are related to the only one order.

- Entities Involved: *Order, Order_Detail*

- Foreign Key(s): The *"Order_id"* column in the *"Order_detail"* table reference the *"Order_id"* column in the *"Order"* table.

- Purpose: This relationship allowed tracking order details of all the orders.

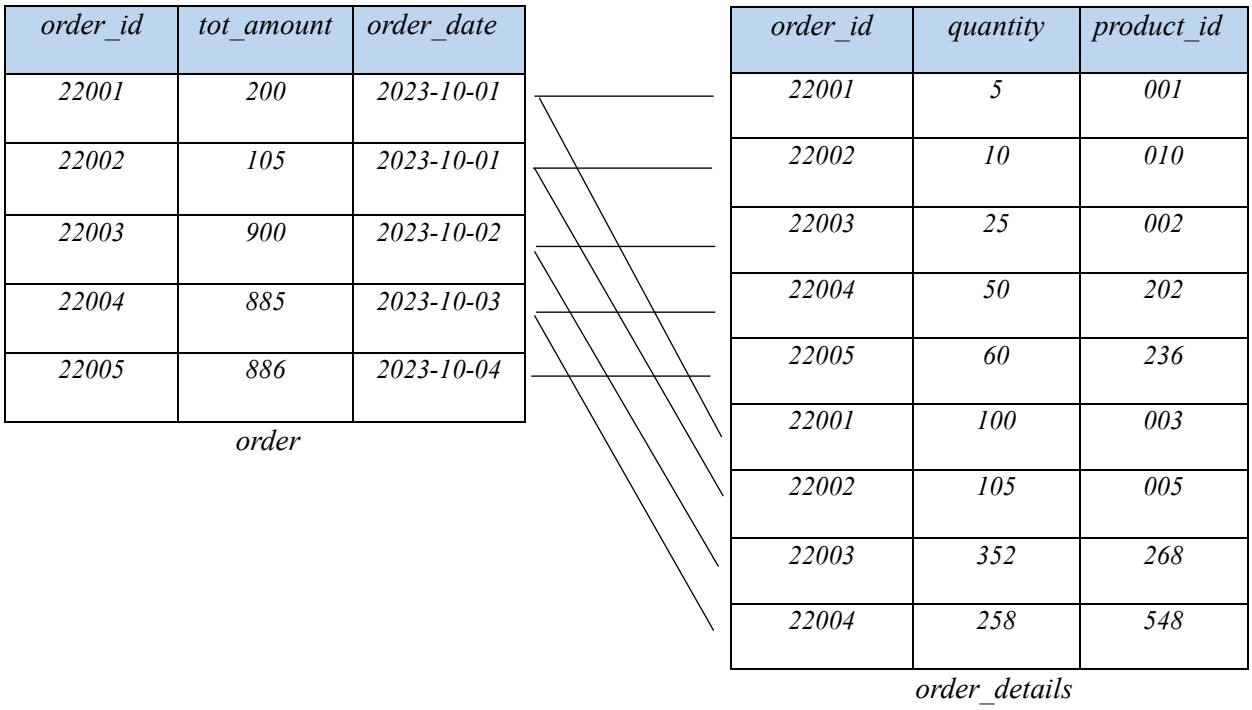


Figure 2.5.2 relationship between *order* and *order_details*

2.5.3 Order – Transaction Relationship

- Description: The *"Order"* table has transaction details related to *"transaction"* table with *One-to-One* relationship. Every one order can have only one transaction.
- Entities Involved: *Order, Transaction*
- Foreign Key(s): The *"order_id"* column in the *"Transaction"* table reference the *"order_id"* column in *"order"* table.
- Purpose: This relationship tracks every transaction details of the order.

order_id	*tot_amount*	*order_date*
22001	*200*	*2023-10-01*
22002	*105*	*2023-10-01*
22003	*900*	*2023-10-02*
22004	*885*	*2023-10-03*
22005	*886*	*2023-10-04*
22006	*100*	*2023-10-05*
22007	*790*	*2023-10-06*
22008	*91*	*2023-10-07*
22009	*792*	*2323-10-08*

order

order_id	*transaction_id*	*date*
22001	*200*	*2023-10-01*
22002	*105*	*2023-10-01*
22003	*900*	*2023-10-02*
22004	*885*	*2023-10-03*
22005	*886*	*2023-10-04*
22006	*100*	*2023-10-05*
22007	*790*	*2023-10-06*
22008	*91*	*2023-10-07*
22009	*792*	*2323-10-08*

transaction

Figure 2.5.3 relationship between *order* and *transaction*

2.5.4 Supplier – Product Relationship

- Description: The *"supplier"* table has related to the *"product"* in *Many-to-Many* Relationship. Because many suppliers can supply many products.
- Entities: *supplier, Product*
- Foreign Key(s): The *"supplier_id"* column in the *"product"* table reference the *"supplier_id"* column in the *"supplier"* table.
- Purpose: This relationship track all the details of the supplier and all the details of supplier supplied products.

supplier_id	*supplier_name*
1	*Lanka Food Distributors*
2	*Ceylon Grocery Stores*
3	*Sri Lanka Educational Supplier*
4	*Island Fresh Foods*
5	*Cinnamon Spice Garden*

supplier

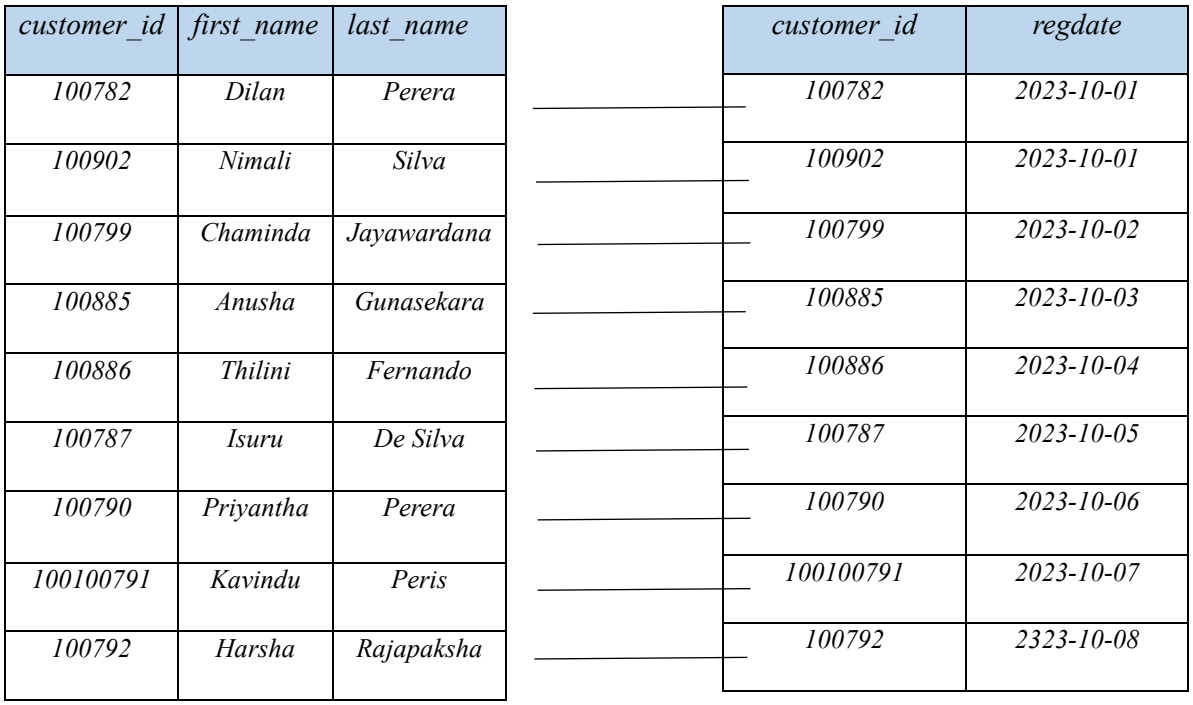
order_id	*tot_amount*	*supplier_id*
22001	*200*	*1*
22002	*100*	*2*
22003	*900*	*3*
22004	*800*	*4*
22005	*800*	*5*
22006	*100*	*1*
22007	*800*	*2*
22008	*100*	*3*
22009	*600*	*4*

product

Figure 2.5.4 relationship between *supplier* and *product*

2.5.5 Customer – Customer Details Relationship

- Description: The *"customer"* table has related to the *"customerdetails"* in *One-to-One* Relationship. One customer has one customer details.
- Entities: *Customer, CustomerDetails*
- Foreign Key(s): The *"CustomerId"* column in the *"CustomerDetails"* table reference The *"Cust_id"* column in the *"customer"* table.
- Purpose: In this relationship allows tracking details of the customer.



customer_id	first_name	last_name
100782	Dilan	Perera
100902	Nimali	Silva
100799	Chaminda	Jayawardana
100885	Anusha	Gunasekara
100886	Thilini	Fernando
100787	Isuru	De Silva
100790	Priyantha	Perera
100100791	Kavindu	Peris
100792	Harsha	Rajapaksha

customer

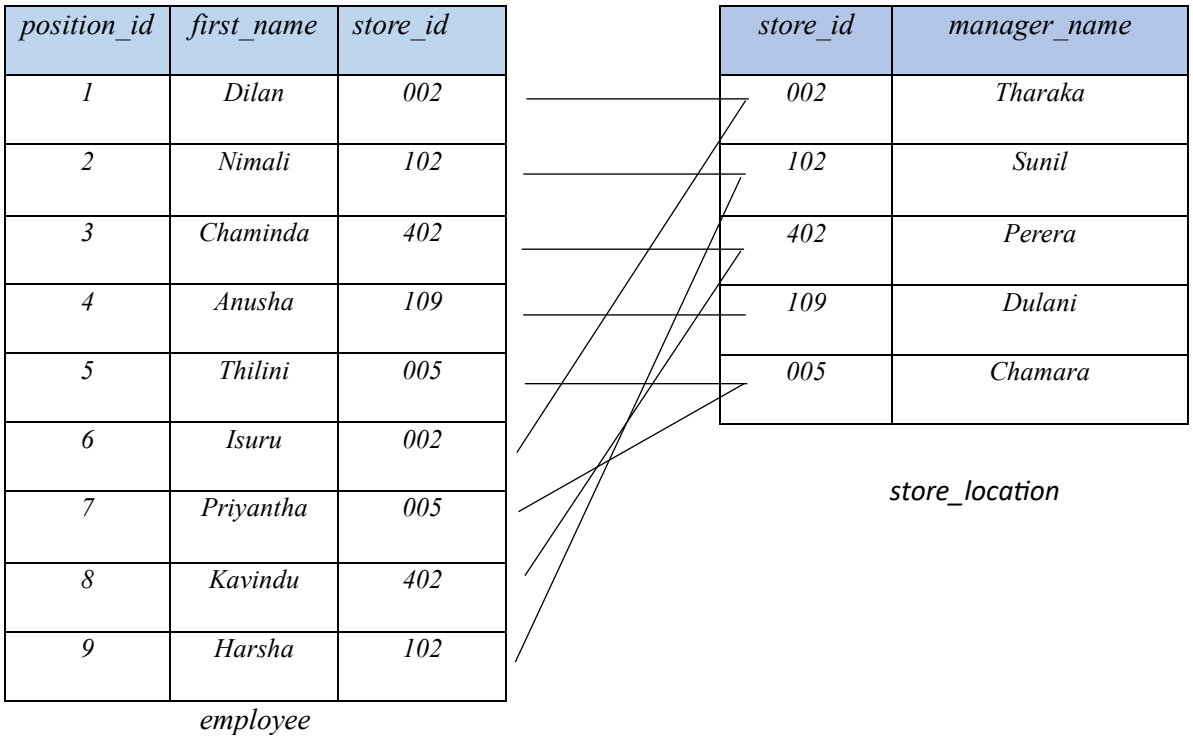
customer_id	regdate
100782	2023-10-01
100902	2023-10-01
100799	2023-10-02
100885	2023-10-03
100886	2023-10-04
100787	2023-10-05
100790	2023-10-06
100100791	2023-10-07
100792	2323-10-08

customer_details

Figure 2.5.5 relationship between *customer* and *customer-detail*

2.5.6 Employee – Store Location Relationship

- Description: The *"Employee"* table is related to *"St_Location"* table in *Many-to-One* Relationship. Many employees can work in one stores.
- Entities: *Employee, St_Location*
- Foreign Key(s): The *"Store_id"* column in the *"Employee"* table reference the *"StoreId"* Column in *"St_Location"* table.
- Purpose: This relationship tracks Employee details and the Store Locations as well as it Define the employee's working location.



position_id	first_name	store_id
1	Dilan	002
2	Nimali	102
3	Chaminda	402
4	Anusha	109
5	Thilini	005
6	Isuru	002
7	Priyantha	005
8	Kavindu	402
9	Harsha	102

employee

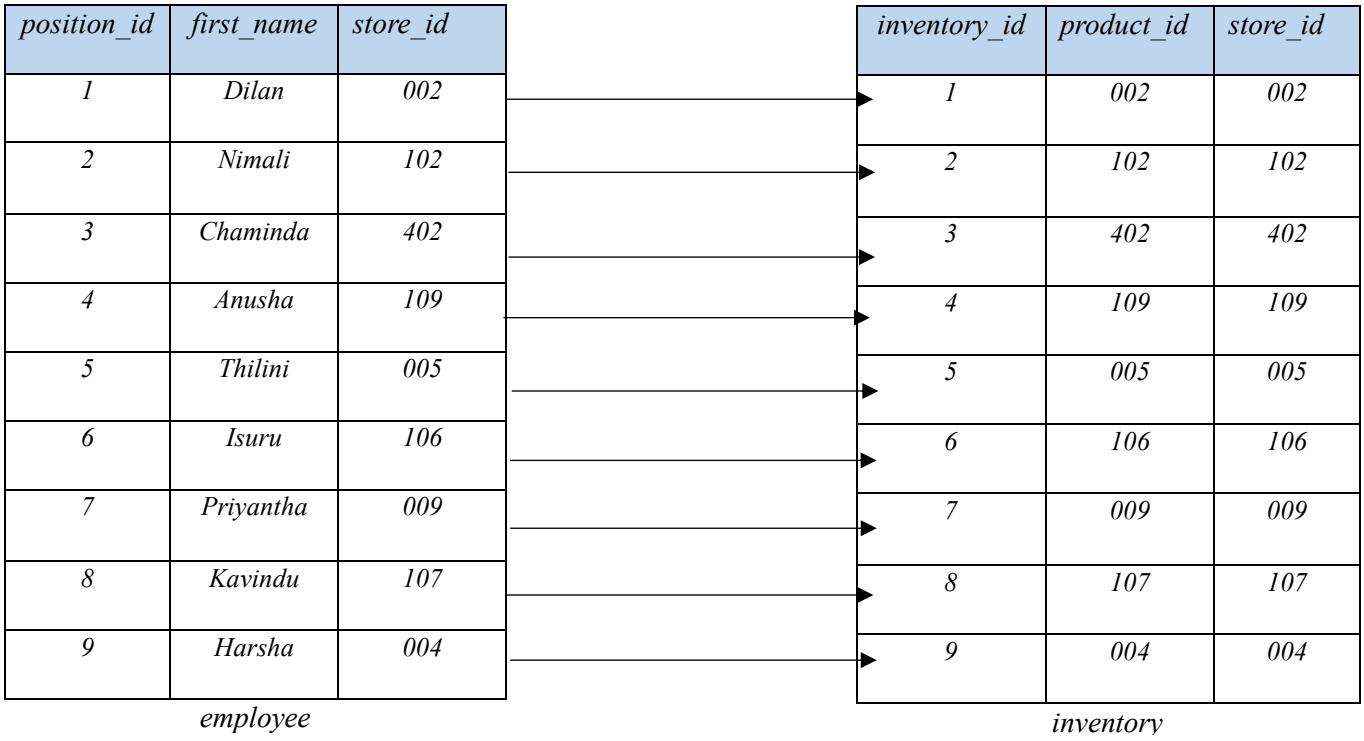
store_id	manager_name
002	Tharaka
102	Sunil
402	Perera
109	Dulani
005	Chamara

store_location

Figure 2.5.6 relationship between *customer* and *order*

2.5.7 Employee – Inventory Relationship

- Description: The *"Employee"* table is related to the *"Inventory"* table in *One-to-One* Relationship. One employee can handle one inventory.
- Entities: *Employee, Inventory*
- Purpose: In this relationship tracks Employee handling of inventory.



position_id	*first_name*	*store_id*
1	*Dilan*	*002*
2	*Nimali*	*102*
3	*Chaminda*	*402*
4	*Anusha*	*109*
5	*Thilini*	*005*
6	*Isuru*	*106*
7	*Priyantha*	*009*
8	*Kavindu*	*107*
9	*Harsha*	*004*

employee

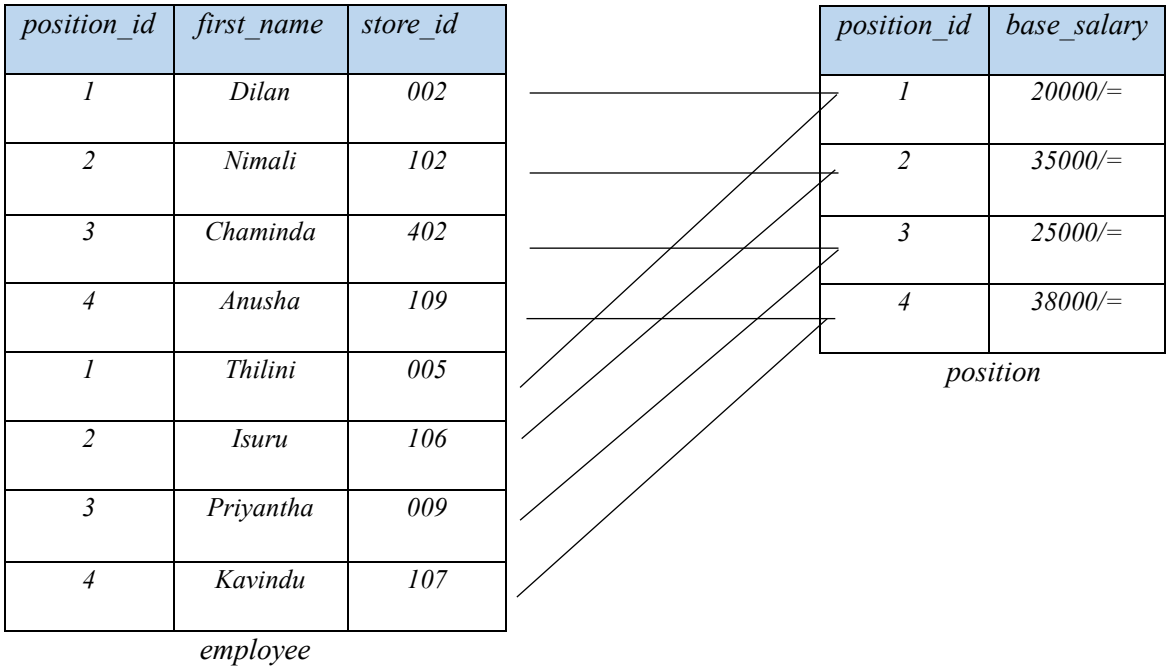
inventory_id	*product_id*	*store_id*
1	*002*	*002*
2	*102*	*102*
3	*402*	*402*
4	*109*	*109*
5	*005*	*005*
6	*106*	*106*
7	*009*	*009*
8	*107*	*107*
9	*004*	*004*

inventory

Figure 2.5.7 relationship between *employee* and *inventory*

2.5.8 Employee – Position Relationship

- Description: The *"Employee"* table is related to the *"position"* table in *Many-to-One* relationship. Many employees can have one position.
- Entities: *Employee, Position*
- Foreign Key(s): The *"PositionId"* column in the *"Employee"* table references the *"positionId"* column in the *"Position"* table.
- Purpose: This relationship tracks all the positions of the employees.



position_id	first_name	store_id
1	Dilan	002
2	Nimali	102
3	Chaminda	402
4	Anusha	109
1	Thilini	005
2	Isuru	106
3	Priyantha	009
4	Kavindu	107

employee

position_id	base_salary
1	20000/=
2	35000/=
3	25000/=
4	38000/=

position

Figure 2.5.8 relationship between *employee* and *position*

2.5.9 Employee – Experience Relationship

- Description: The *"Employee"* table is related to the *"Experience"* table in *One-to-One* relationship. One employee has one experience at work.
- Entities: *Employee, Experience*
- Purpose: This relationship tracks all the experiences of the employee.

position_id	*first_name*	*store_id*	*exp_id*
1	*Dilan*	*002*	*101*
2	*Nimali*	*102*	*202*
3	*Chaminda*	*402*	*303*
4	*Anusha*	*109*	*404*
5	*Thilini*	*005*	*505*
6	*Isuru*	*106*	*606*
7	*Priyantha*	*009*	*707*
8	*Kavindu*	*107*	*808*
9	*Harsha*	*004*	*909*

employee

exp_id	*exp_level*
101	*1*
202	*2*
303	*3*
404	*4*
505	*5*
606	*6*
707	*7*
808	*8*
909	*9*

experience

Figure 2.5.9 relationship between *employee* and *experience*

2.5.10 Employee – Employee Details Relationship

- Description: The *"Employee"* table is related to the *"Emp_detail"* table in *One-to-One* Relationship. Many employees can have one employee detail.
- Entities: *Employee, Emp_detail*
- Foreign Key(s): The *"EmpId"* column in the *"employee"* table reference the *"EmpId"* Column in the *"Emp_detail"* table.
- Purpose: This relationship tracks all the employee details of the employee.

emp_id	*first_name*	*store_id*
1	*Dilan*	*002*
2	*Nimali*	*102*
3	*Chaminda*	*402*
4	*Anusha*	*109*
5	*Thilini*	*005*
6	*Isuru*	*106*
7	*Priyantha*	*009*
8	*Kavindu*	*107*
9	*Harsha*	*004*

employee

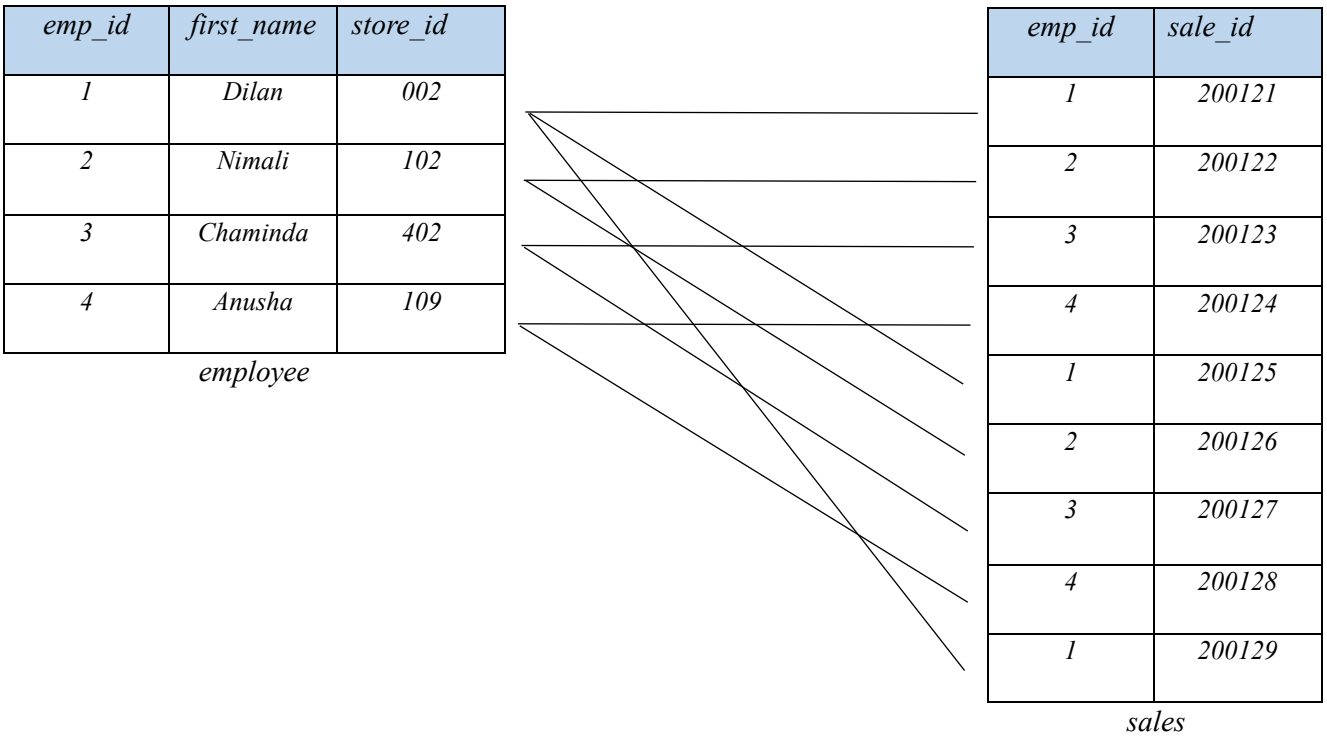
emp_id	*position_id*
1	*002*
2	*102*
3	*402*
4	*109*
5	*005*
6	*106*
7	*009*
8	*107*
9	*004*

employee detail

Figure 2.5.10 relationship between *employee* and *employee detail*

2.5.11 Employee – Sales Relationship

- Description: The *"Employee"* table is related to the *"Sales"* table in *One-to-Many* relationship. One employee does many sales.
- Entities: *Employee, Sales*
- Foreign Key(s): The *"EmpId"* column in the *"Employee"* table reference the *"EmpId"* Column in the *"Sales"* table.
- Purpose: This relationship tracks all the details of the sales that employees do.



emp_id	*first_name*	*store_id*
1	*Dilan*	*002*
2	*Nimali*	*102*
3	*Chaminda*	*402*
4	*Anusha*	*109*

employee

emp_id	*sale_id*
1	*200121*
2	*200122*
3	*200123*
4	*200124*
1	*200125*
2	*200126*
3	*200127*
4	*200128*
1	*200129*

sales

Figure 2.5.11 relationship between *employee* and *employee sales*

3. Advanced Data Analysis Techniques

3.1 Sales Analysis for Store Managers

Tables: Sales, Employees

This analysis focuses on evaluating the sales performance of store managers within the supermarket. By identifying and assessing the total sales generated by each store manager, this analysis provides insights into the effectiveness of managerial staff in driving sales. It helps pinpoint top-performing store managers and areas where sales strategies may be improved or optimized for better results.

Objectives:

- Identify Top-Performing Store Managers: To pinpoint and acknowledge store managers who excel in driving sales and revenue within their respective stores.
- Performance Evaluation: To evaluate the effectiveness of managerial staff in achieving and exceeding sales targets, providing insights into the strengths and areas for improvement in their sales strategies.
- Optimization of Sales Strategies: To identify opportunities for optimizing and enhancing sales strategies employed by store managers, ultimately leading to improved results and increased revenue.
- Informed Decision-Making: To provide data-driven insights that support decision-making related to staffing, performance recognition, and resource allocation within the supermarket.

```
SELECT e.employee_id, SUM(s.total_sales_amount) AS TotalSales
FROM Sales s
INNER JOIN Employees e ON s.employee_id = e.employee_id
WHERE e.position_id = (SELECT position_id FROM Positions WHERE position_name = 'manager')
GROUP BY e.employee_id
ORDER BY TotalSales DESC;
```

3.2 Customer Analysis

Tables: customers, order_details

This analysis is focused on identifying and acknowledging the top 50 customers who have made the most significant contributions to the supermarket's revenue. It achieves this by evaluating the total purchase amounts (total sales) of each customer, providing insights into their importance in terms of revenue generation. By limiting the results to the top 50 customers, this analysis offers valuable information about the most substantial contributors to the supermarket's sales revenue.

Objective:

The primary objective of this analysis is to:

- Recognize High-Value Customers: By evaluating and ranking customers based on their total purchase amounts recorded in the "Sales" table, the analysis aims to identify and recognize those customers who have made the most substantial contributions to the supermarket's revenue.

```
SELECT c.customer_id, SUM(od.subtotal) AS TotalPurchaseAmount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
GROUP BY c.customer_id
ORDER BY TotalPurchaseAmount DESC
LIMIT 50;
```

3.3 Inventory Management

Tables: Inventory, Products

Inventory management involves monitoring stock levels and ensuring that products are available when customers need them. It helps optimize inventory and prevent stockouts.

Objective:

- Prevent Stockouts: Identify and address products that are currently below their restock threshold, ensuring that the supermarket maintains a sufficient supply of products to meet customer demand without experiencing stockouts.
- Optimize Inventory Levels: Maintain a balanced inventory level that minimizes carrying costs while ensuring products are consistently available, ultimately enhancing operational efficiency.
- Resource Allocation: Allocate resources efficiently by focusing on products that require restocking, thereby avoiding unnecessary expenditures on overstocked items and preventing revenue losses due to understocked items.
- Supplier Management: Evaluate product suppliers based on restocking needs, ensuring timely deliveries and effective supplier relationships, reducing the risk of supply chain disruptions.
- Enhance Customer Satisfaction: Offer a superior shopping experience by guaranteeing that products are accessible, thereby increasing customer satisfaction and loyalty.

```
SELECT p.product_id, p.name AS ProductName, i.stock_quantity, i.restock_threshold
FROM Products p
JOIN Inventory i ON p.product_id = i.product_id
WHERE i.stock_quantity < i.restock_threshold;
```

3.4 Employee Performance

Tables; Employees, Sales

The "Employee Performance" analysis assesses individual employee performance in achieving sales targets and customer satisfaction within the supermarket.

Objectives:

- Identify Top-Performing Employees: To pinpoint and acknowledge employees who excel in driving sales and revenue within the supermarket.
- Performance Evaluation: To evaluate the effectiveness of employees in achieving and exceeding sales targets, providing insights into their strengths and areas for improvement in their sales strategies.
- Recognition and Incentives: To recognize and incentivize top performers, fostering motivation and loyalty among the workforce.
- Data-Driven Decision-Making: To provide data-driven insights that support decision-making related to staffing, training, and sales strategies.
- Customer Satisfaction: To ensure that employees contribute to a positive customer experience, resulting in higher customer satisfaction and loyalty.

```
SELECT e.employee_id, e.first_name, e.last_name, SUM(s.total_sales_amount) AS TotalSales,
COUNT(s.sale_id) AS TotalTransactions, SUM(s.total_sales_amount) / COUNT(s.sale_id) AS
AvgSalesPerTransaction
FROM Sales s
INNER JOIN Employees e ON s.employee_id = e.employee_id
GROUP BY e.employee_id, e.first_name, e.last_name;
```

3.5 Store Analysis

Tables: Store Locations, Visits, Transactions

Description: Store analysis involves assessing the performance of different store locations. It includes analyzing foot traffic, conversion rates, and sales per store.

Objective:

- Evaluate Foot Traffic: To understand the level of foot traffic and customer visits at each store location, helping in assessing store popularity.
- Analyze Unique Visitors: To determine the number of unique visitors to each store, providing insights into customer diversity and market reach.
- Measure Sales Activities: To assess the total number of transactions and the cumulative sales amounts at each store, aiding in evaluating revenue generation.
- Identify Top-Performing Stores: To pinpoint and recognize the top-performing store locations in terms of visitor engagement and sales performance.
- Data-Driven Decision-Making: To provide data-driven insights for making informed decisions related to store management, resource allocation, and marketing strategies.

```
SELECT
    sl.store_id,
    sl.store_name,
    COUNT(v.visit_id) AS TotalVisits,
    COUNT(DISTINCT v.customer_id) AS UniqueVisitors,
    COUNT(DISTINCT t.transaction_id) AS TotalTransactions,
    SUM(t.amount) AS TotalSales
FROM
    store_locations sl
LEFT JOIN
    visits v ON sl.store_id = v.store_id
LEFT JOIN
    transactions t ON v.visit_id = t.visit_id
GROUP BY
    sl.store_id, sl.store_name
ORDER BY
    sl.store_id;
```

3.6 Financial Analysis

Tables: Sales, Transactions

Financial analysis evaluates the financial health of the business. It includes assessing profitability, revenue, and the impact of transactions on the business's financial status.

Objectives:

- Assess Profitability: To determine the overall profitability of the supermarket by analyzing the relationship between sales and transactions.
- Evaluate Revenue: To assess the total revenue generated by the supermarket, considering sales amounts and transaction values.
- Impact of Transactions: To understand the financial impact of different transaction types on the business's financial status.
- Identify Trends: To identify financial trends and patterns that can inform decision-making and financial planning.
- Support Financial Decision-Making: To provide financial data for making informed decisions related to budgeting, pricing, and resource allocation.

```
SELECT
    DATE_FORMAT(s.sale_date, '%Y-%m') AS Month,
    SUM(s.total_sales_amount) AS TotalSales,
    COUNT(t.transaction_id) AS TotalTransactions,
    SUM(t.amount) AS TotalTransactionAmount
FROM
    Sales s
LEFT JOIN
    Transactions t ON s.sale_date = t.date
GROUP BY
    Month
ORDER BY
    Month;
```

3.7 Visitation Analysis

Tables: Visits, Transactions

Visitation analysis focuses on customer visits to the store without making transactions. It helps identify potential reasons for non-purchase, such as product availability or pricing issues.

Objectives:

- Analyze Non-Purchase Visits: To assess and understand the frequency and patterns of customer visits to the store without making transactions.
- Identify Potential Issues: To identify potential reasons for non-purchase, such as product availability, pricing, or other factors affecting customer engagement.
- Improve Customer Experience: To use the insights from visitation analysis to enhance the overall customer experience and encourage conversion.
- Optimize Store Layout: To consider optimizing the store layout, product placement, or pricing strategies based on visitation analysis findings.

```
SELECT
    v.date AS VisitDate,
    COUNT(v.visit_id) AS TotalVisits,
    COUNT(t.transaction_id) AS TotalTransactions
FROM
    Visits v
LEFT JOIN
    Transactions t ON v.visit_id = t.visit_id
GROUP BY
    VisitDate
ORDER BY
    VisitDate;
```

3.8 Transaction Analysis

Tables: Transactions, Orders, Products

Transaction analysis involves understanding the details of customer transactions, including the products purchased, payment methods, and order amounts.

Objective:

- The objective of this analysis is to delve into the intricacies of customer transactions. It aims to gain a comprehensive understanding of the products purchased, payment methods employed, and order amounts, thereby enabling data-driven insights for better decision-making.

```
SELECT
    t.transaction_id,
    t.transaction_type,
    t.date AS transaction_date,
    o.order_id,
    o.customer_id,
    o.order_date,
    o.total_amount AS order_total_amount,
    o.payment_method,
    o.payment_date AS payment_date,
    p.product_id,
    p.name AS product_name,
    p.category AS product_category,
    p.price AS product_price
FROM transactions t
INNER JOIN orders o ON t.order_id = o.order_id
INNER JOIN order_details od ON o.order_id = od.order_id
INNER JOIN products p ON od.product_id = p.product_id;
```

4. Testing and Validation

The testing and validation process in the Food City organization should be designed to ensure that the organization's software systems meet the needs of its customers and that they are of high quality. The testing process should include all of the basic testing processes described above, as well as any advanced testing processes that are relevant to the organization's systems.

The Food City organization should also have a process in place to manage the testing and validation process. This process should include the following steps:

1. *Plan: The first step is to develop a testing plan that identifies the systems to be tested, the types of testing to be performed, and the resources required.*
2. *Execute: The next step is to execute the testing plan. This involves running the test cases and documenting the results.*
3. *Analyze: Once the testing is complete, the results should be analyzed to identify any defects in the system.*
4. *Report: The final step is to report the testing results to the appropriate stakeholders. This report should include a summary of the testing activities, the defects found, and the recommendations for fixing those defects.*

4.1 Unit Testing

Unit testing is the process of testing individual units of code to ensure that they are working as expected. This can be done by writing test cases that simulate different inputs and outputs to the unit of code. The test cases should be designed to cover all possible scenarios that the unit of code could encounter in production.

- **Unit tests for stored procedures in Food City database:**
    - *Test that stored procedures return the correct results for a given set of input parameters.*
    - *Test that stored procedures handle invalid input parameters correctly.*
    - *Test that stored procedures do not modify data unexpectedly.*
- **Unit tests for functions in Food City database:**
    - *Test that functions return the correct results for a given set of input parameters.*
    - *Test that functions handle invalid input parameters correctly.*
    - *Test that functions do not modify data unexpectedly.*
- **Unit tests for views in Food City database:**
    - *Test that views return the correct results for a given set of input parameters.*
    - *Test that views are updated correctly when the underlying data is changed.*
- **Unit tests for triggers in Food City database:**
    - *Test that triggers fire correctly when the specified events occur.*
    - *Test that triggers do not modify data unexpectedly.*

4.2 Integration Testing

Integration testing is the process of testing how different units of code work together,order calculation function might be integrated with a function that retrieves the prices of items from a database. The integration test would check that the two functions work together correctly to calculate the total price of an order.

- **Integration tests for Food City database:**
    - *Test that the database components interact correctly with each other.*
    - *Test that the database can handle concurrent transactions.*
    - *Test that the database can recover from failures.*
- **Test data generator for Food City database:**
    - *Generate realistic test data that can be used for integration testing.*
    - *Generate test data that covers a wide range of scenarios.*

4.3 User Acceptance Testing

User acceptance testing (UAT) is the process of testing a system with real users to ensure that it meets their needs.

- **User acceptance tests for Food City database:**
    - *Test that the database meets the business requirements.*
    - *Test that the database is easy to use.*
    - *Test that the database is performant.*
- **User involvement in user acceptance testing for Food City database:**
    - *Involve users in the development of user acceptance tests.*
    - *Obtain feedback from users on the usability of the database.*

test id	***title***	***steps***	***expected results***	***state***
101	*Verify product availability*	*1. Search for a product in the Food City inventory system.* *2. Check if the product is listed as "In Stock" or "Out of Stock".*	*The product is listed as "In Stock".*	*pass*
102	*Confirm product pricing*	*1. Select a product from the Food City inventory system* *2. Verify that the displayed price matches the current price tag on the shelf.*	*The displayed price matches the current price tag on the shelf.*	*pass*
103	*Ensure product freshness*	*1. Inspect the product for signs of spoilage or damage.* *2. Check the expiration date on the product packaging.*	*The product is fresh and has not expired.*	*pass*

104	Test product barcode scanning	1. Scan the product barcode at a Food City checkout counter. 2. Verify that the correct product name and price are displayed on the checkout screen.	The correct product name and price are displayed on the checkout screen.	pass
105	Validate product discounts and promotions	1. Apply a discount or promotion to a product at checkout. 2. Verify that the correct discount amount is applied to the final bill.	The correct discount amount is applied to the final bill.	pass
106	Confirm accurate product weights	1. Weigh a product on a Food City scale. 2. Compare the weight to the weight listed on the product packaging.	The weight matches the weight listed on the product packaging.	pass
107	Ensure product labeling accuracy	1. Inspect the product labeling for accuracy. 2. Verify that all required information is present, such as the ingredients list, nutritional information, and manufacturer's contact information.	The product labeling is accurate and complete.	pass
108	Test product returns and exchanges	1. Attempt to return or exchange a product at a Food City customer service desk. 2. Verify that the return or exchange is processed correctly.	The return or exchange is processed correctly.	pass
109	Evaluate product customer satisfaction	1. Conduct a survey of Food City customers to gather feedback on their satisfaction with product quality. 2. Analyze the survey results to identify areas for improvement.	The survey results indicate that customers are generally satisfied with Food City product quality.	pass

110	*Verify employee login and access*	*1. Attempt to log in to the Food City employee portal using a valid username and password.* *2. Verify that the employee has access to the appropriate modules and data.*	*The employee is able to log in successfully and has access to the appropriate modules and data.*	*pass*
111	*Test employee time and attendance tracking*	*1. Clock in and out of work using the Food City time and attendance system.* *2. Verify that the employee's hours are accurately recorded.*	*The employee's hours are accurately recorded.*	*pass*
112	*Confirm employee pay calculation*	*1. Review the employee's pay stub to verify that their pay is correct.* *2. Calculate the employee's pay using the Food City payroll system and compare the results to the pay stub.*	*The employee's pay is correct.*	*pass*
113	*Test employee performance reviews*	*1. Complete a performance review for an employee.* *2. Verify that the review is submitted and stored correctly in the Food City employee records system.*	*The performance review is submitted and stored correctly.*	*pass*
114	*Evaluate employee training effectiveness*	*1. Conduct a survey of Food City employees to gather feedback on the effectiveness of the company's training programs* *.* *2. Analyze the survey results to identify areas for improvement.*	*The survey results indicate that employees are generally satisfied with the effectiveness of the company's training programs.*	*pass*

115	*Verify customer loyalty program functionality*	*1. Enroll a customer in the Food City loyalty program.* *2. Earn loyalty points by making purchases at Food City. 3. Redeem loyalty points for discounts on future purchases.*	*The customer is able to enroll in the loyalty program, earn loyalty points, and redeem loyalty points for discounts.*	*pass*
116	*Test customer communication channels*	*1. Send an email to Food City customer service* *. 2. Call the Food City customer service hotline.* *3. Submit a feedback form on the Food City website.*	*The customer receives a timely and helpful response from Food City customer service.*	*pass*

Figure 4.1 *Some test cases*

-The end of the document-