Group No. 8

CMPG 311: Database Documentation for the Business Operations of Kasi Lifestyle

B.Sc. Information Technology

Group Name: The Variables

Group Members:

Bonolo Zizi	43149707
Charmaine Msibi	42526566
Kopano Mere	42228336
Mohau Liphoko	42524547
Olebogeng Moremi	40123316
Phuti Malota	41370015
Tintswalo Baloyi	35481013

Table of Content

CMPG 311: Database Documentation for the 0
Business Operations of Kasi Lifestyle 0
B.Sc. Information Technology 0
Group Name 0
Group Members: 0
Meet Our Team 3
Phase 1: Database Initial Study 4
Analysis Of Kasi Lifestyle's Situation 4
1.1 Background 4
1.2 Company Objectives 4
1.3 Objectives To Reach Goal: 4
1.4 Company Operations 4
1.5 Business Rules 5
1.6 Business Structure 6
1.7 Constraints 6
1.8 Database system specifications 7
1.9 Information that the company requires from the database. 8
2. Scope 8
3. Boundaries 9
Phase 2: Database Design 9
Notes On ER Diagram 11
Logical Data Model - Mapping Logical Model 12
Phase 3: Physical Design 13
Physical Data Model in Oracle 13
1. Database Object 14
1.1 Initialisation 14
Creation of Tables 15
Employee Table 15
Cashier Table 15
2.2.3 16
2.2.4 16
2.2.6 Delivery Detail 17
2.2.7 Menu Item table 18
2.2.8 Order Item table 18

Populating The Database 19
Insert statements 19
Tables and Their Queries 19
Employee 19
  Delivery Person 22
  Online Customer 23
Queries 30

Meet Our Team

Group Leader

Mohau Liphoko

Charmaine Msibi

Phuti Malota

Olebogeng Moremi

Tintswalo Baloyi

Kopano Mere

Phase 1: Database Initial Study

Analysis Of Kasi Lifestyle's Situation

1.1 Background

Kasi Lifestyle is a fast-food shop that offers various quality dishes such as the Bunny Chow, Dagwood sandwiches, braai meat, as well as a friendly atmosphere for its customers. Situated in the township of Ikageng the shop has been making waves with news about it reaching the beyond Ikageng to the rest of Potchefstroom.

1.2 Company Objectives

Because of their newly acquired customer base, Kasi Lifestyle's way of running business is getting obsolete. Which is keeping track of business transactions and operations on paper. This has caused them a myriad of problems such as encountering data anomalies, inconsistencies, and violations of data integrity. Hence the company approached our group with the aim to offer its services to their new but distant customers, allowing online orders and deliveries to their doorstep.

1.3 Objectives To Reach Goal:

- Implement an online system where customers can order from the comfort of their homes.
- Implement a delivery system where the customer orders will be delivered with efficiency.

1.4 Company Operations

Currently, customers physically enter the restaurant and approach the cashier to place their orders from the menu. The cashier writes down the customer's order, in a physical 3-quire order book. They then accept payment from the customer, either in cash or card.

The cashier communicates the details of the order to the kitchen staff verbally or by giving them the order slip obtained from the transaction. The chef begins preparing the food according to the order.

Once the food is ready, the customer either picks it up or it is placed on the counter for them to collect.

1.5 Business Rules

Cashier – Orders (1-M):

- One cashier can have many instore orders.
- Many instore orders can be done by one cashier.

Delivery person – Orders (1-M):

- One delivery person can have many orders.
- Many orders can be done by one delivery person.

Delivery Person – Delivery details (1-M):

- One delivery person can have many delivery details.
- Many delivery details can be had by one delivery person.

Online Customer - Delivery details (1-M):

- One online customer can have many delivery details.
- Many delivery details can be had by one customer.

Online Customer – Orders (1-M):

- One online customer can have many orders.
- Many orders can be made by one online customer.

Order – Order items (1-M):

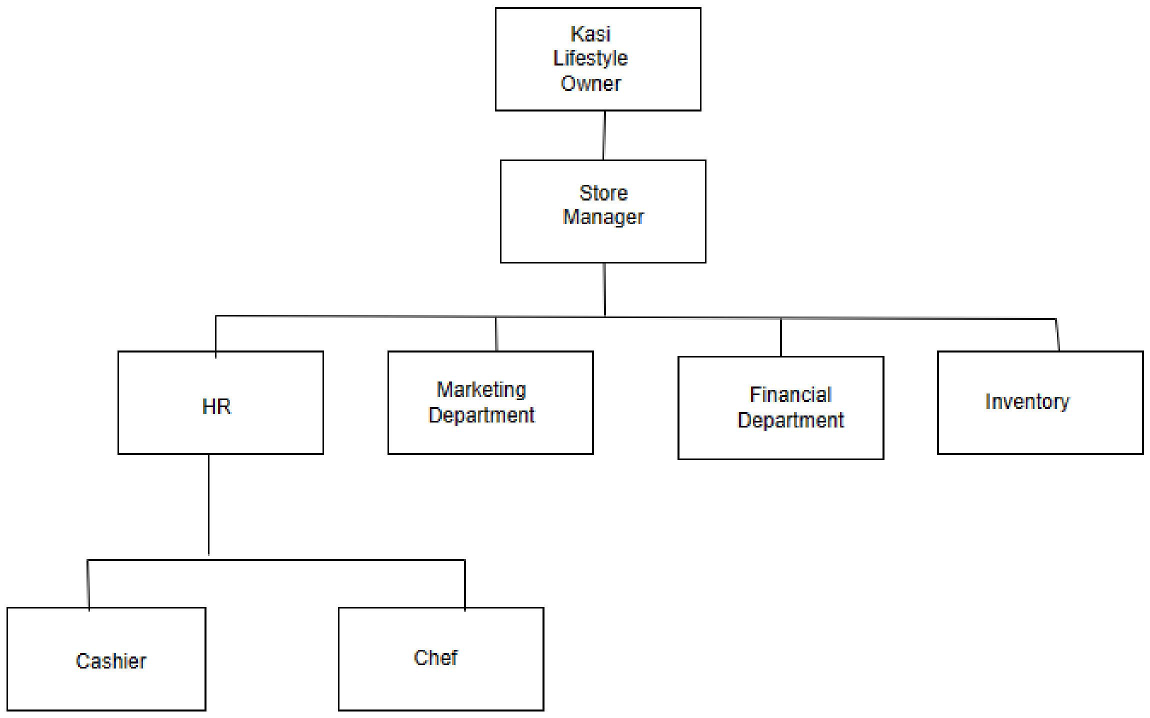
- One order can have many order items.
- Many order items can be in one order.

Menu items – Order items (1-M):

- One menu item can have many order items.

- Many orders items can have one menu item.

1.6 Business Structure



1.7 Constraints

The issues and constraints facing Kasi Lifestyle include limited customer reach due to the absence of an online ordering system, inefficient order processing methods, challenges in adopting new technology, diverse customer expectations, cost considerations, and the need for data security.

1.8 Database system specifications

Objectives to solve problems identified.

The objectives include implementing an online ordering system, customizing it for user experience, developing tailored training programs, finding cost-effective solutions for technology and staff training, and ensuring data security to prevent breaches or unauthorized access to customer information by implementing customer authorization.

The following operations must be included in the database development:

A. Insert – Add new data into the database.
   - Add new employees.
   - Add new online customers.
   - Add new orders.
   - Add menu item.

B. Delete – Remove data from the database.
   - Remove menu item no longer in stock.
   - Remove employees no longer working for Kasi Lifestyle.
   - Remove online customer no longer using Kasi Lifestyle services.

C. Update – Change existing data in the database.
   - Change employee details.
   - Change online customer details.
   - Change menu items.

D. Perform queries.

The database must store the following data:

- Orders made daily by customers.
- All employees and their details.
- Online customer details.
- Menu items available.

To ensure data security and integrity, data in the database must:

- Have no anomalies.
- Have no redundancy.
- Have no null fields.
- Be reached by correct personnel.
- Be backed up frequently.
- Be stored in correct format.

1.9 Information that the company requires from the database.

Kasi Lifestyle will need reports on the following information:

- Up to date information on current cashier employed.
- Up to date information on current delivery person employed.
- Top 10 selling items each month
- A list of all online orders made for the previous month.
- A list of all instore orders made for the previous month.
- A list of deliveries made successfully.

2. Scope

Kasi lifestyle is a restaurant in Ikageng. Looking to revamp their operations by introducing an online ordering system and delivery system. This innovative system, designed to streamline the entire ordering process, offers customers a seamless and user-friendly platform to place their orders securely online. The online ordering system will store the information in a centralised database allowing cashiers to receive and manage online orders, potentially integrating with their existing kitchen display system enhancing operational efficiency. Kasi Lifestyle will offer in-house delivery services, ensuring prompt and reliable delivery of orders directly to customers' doorsteps. The database will capture orders placed online and in-store orders, customer information and deliver information. Implementing the database will help eliminate transaction that was initially paper based. Data integrity will also be improved since there will be no risk of conflicting information existing in multiple paper records.

3. Boundaries

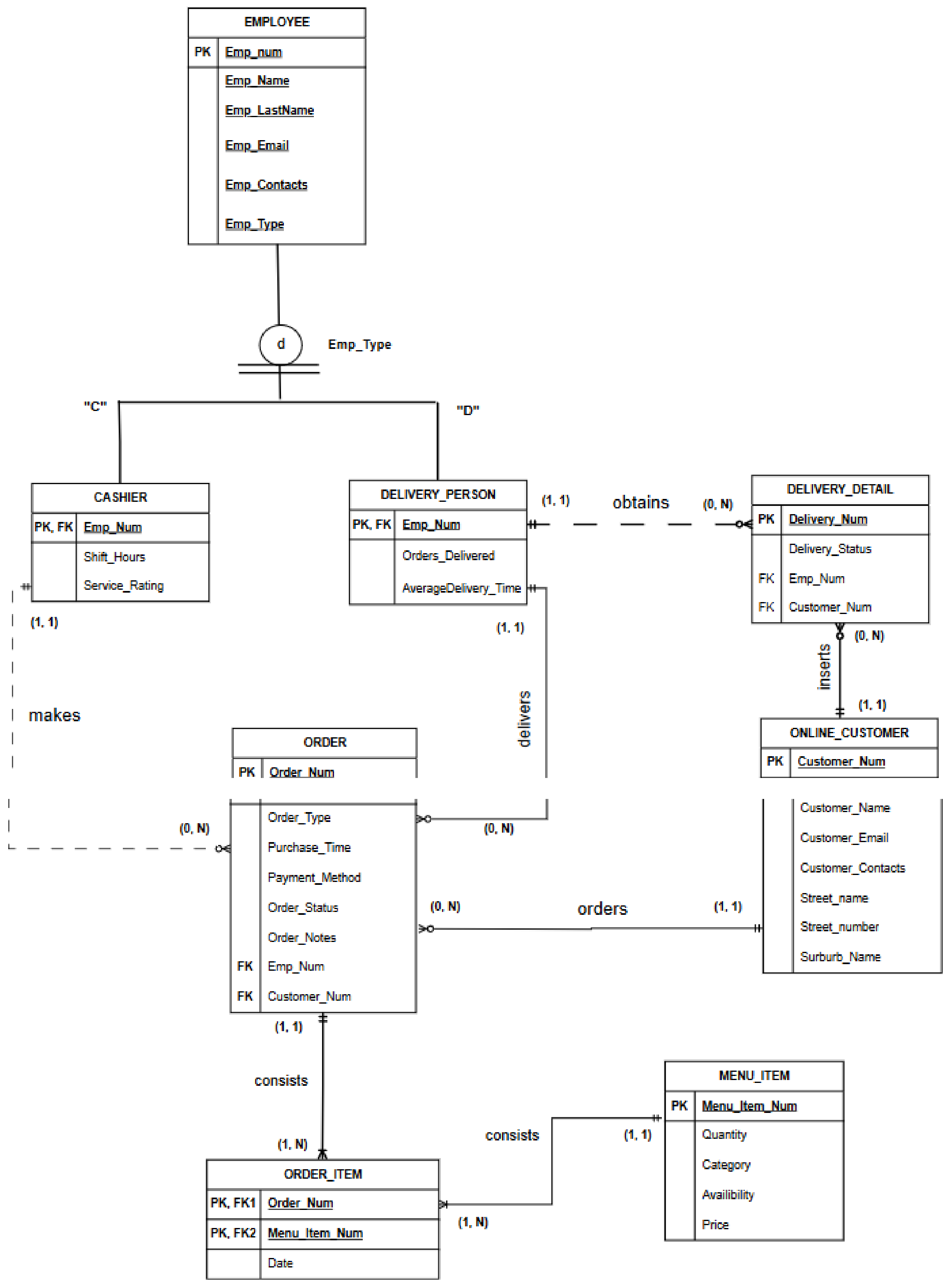
Finance: The company has allocated a total budget of R750,000 for the entire project. Out of this, R300,000 is specifically earmarked for the database design phase. The remaining funds are intended to cover employee salaries and procure all necessary resources for the project.

Time: The project is under a time constraint of two and a half months, during which a team of 7 members must deliver a system. This time limit is considered constrained because team members have other commitments and responsibilities during this period.

Human resources: Due to budget constraints, the company can only allocate 7 members for the project. This decision was made to ensure cost efficiency while still aiming to deliver the system within the allocated time frame.

Software: In response to financial limitations, the team has strategically chosen to leverage a cost-effective solution by adopting a free database software. Among the available options, Microsoft SQL Server emerged as the preferred choice due to its robust features and compatibility with the project requirements. By opting for this software, the company can allocate its resources more efficiently, ensuring that budget constraints do not compromise the quality or functionality of the database system. This decision reflects the team's commitment to delivering a high-quality solution while maximizing cost savings within the project's parameters.

Phase 2: Database Design


EMPLOYEE
PK
Emp_num
Emp_Name
Emp_LastName
Emp_Email
Emp_Contacts
Emp_Type
d
Emp_Type
"C"
"D"
CASHIER
PK, FK
Emp_Num
Shift_Hours
Service_Rating
(1, 1)
makes
DELIVERY_PERSON
PK, FK
Emp_Num
Orders_Delivered
AverageDelivery_Time
(1, 1)
obtains
(0, N)
DELIVERY_DETAIL
PK
Delivery_Num
Delivery_Status
FK
Emp_Num
FK
Customer_Num
(0, N)
inserts
(1, 1)
ONLINE_CUSTOMER
PK
Customer_Num
Customer_Name
Customer_Email
Customer_Contacts
Street_name
Street_number
Surburb_Name
(1, 1)
delivers
ORDER
PK
Order_Num
Order_Type
Purchase_Time
Payment_Method
Order_Status
Order_Notes
FK
Emp_Num
FK
Customer_Num
(0, N)
(0, N)
(0, N)
orders
(1, 1)
(1, 1)
consists
(1, N)
ORDER_ITEM
PK, FK1
Order_Num
PK, FK2
Menu_Item_Num
Date
consists
(1, 1)
(1, N)
MENU_ITEM
PK
Menu_Item_Num
Quantity
Category
Availibility
Price

Notes On ER Diagram

Composite Keys used.

- Order_Items_num (Order_num and Menu_num)

Weak Entities

- Order
- Order_Item
- Menu_Item

Strong Entities

- Employee
- Delivery_Details
- Online_Customer

Bridge Entities

- Order_Item

Strong relationships

- Employee (And all Subtypes)
- Order (And all Subtypes)

Weak relationships

- All the other relationships.

Mandatory relationships

- Employee (And all Subtypes)
- Order (And all Subtypes)
- Online_Customer
- Employee
- Online_Customer and Delivery_Details
- Order and Order_Item
- Online_Customer

Super Entities

- Employee
- Order

Subtype Entities

- Cashier
- Delivery_Person
- Orders

Logical Data Model - Mapping Logical Model

Strong Entities

- DELIVERY_DETAILS(Delivery_Details(PK), Customer_Id(FK), Delivery_Person(FK), Delivery_Status))
- ONLINE_CUSTOMER(Customer_Id(PK), Customer_Name, Customer_Address, Customer_Email, Customer_Number)

Weak Entities

- ORDER_ITEM(Order_items(PK), Menu_items(PK)(FK)) (*Also a composite entity of ORDER and Menu_ITEM)*
- MENU_ITEM(Menu_items(PK), Quantity, Category, Availability, Price)

Supertype

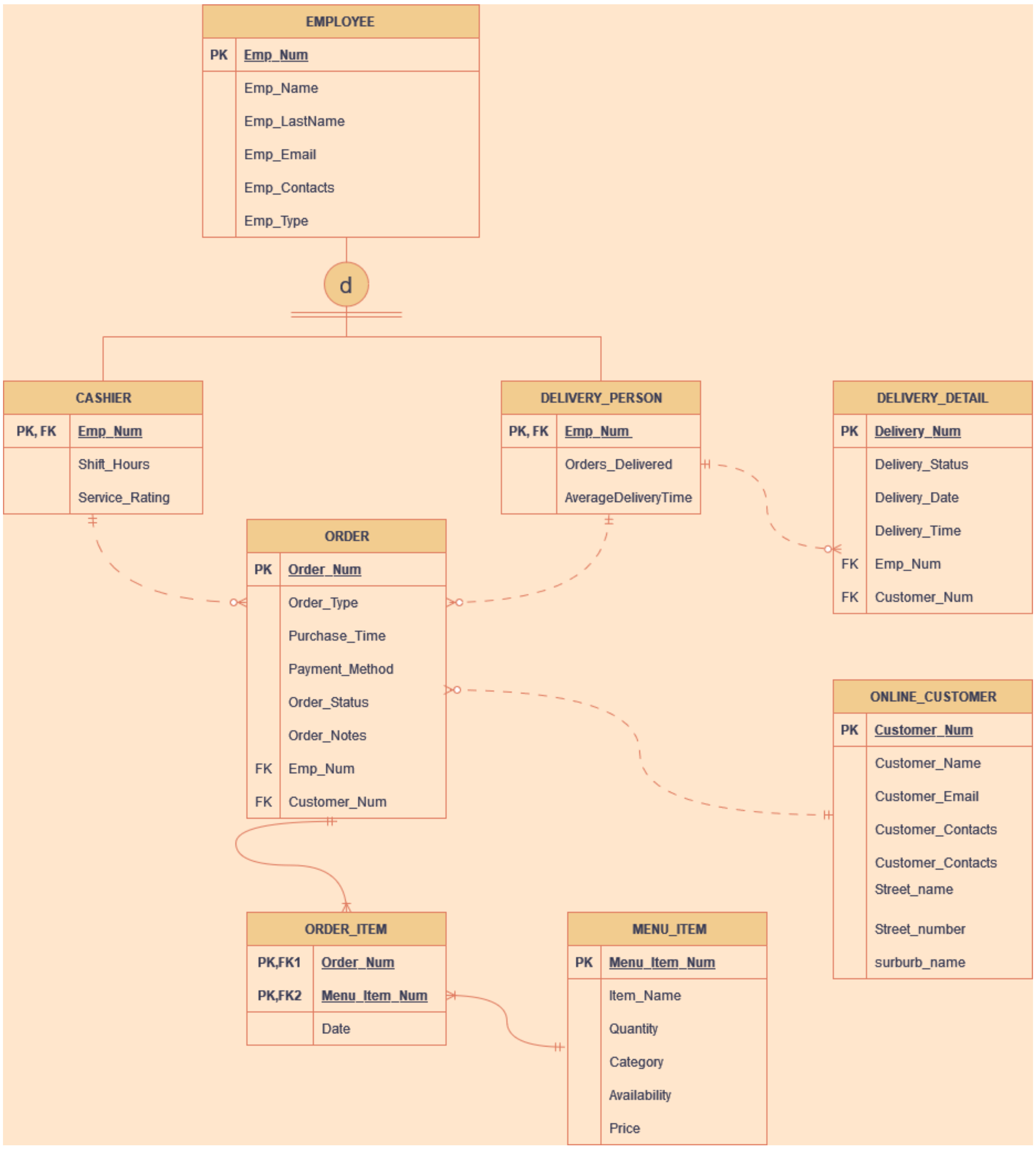
- EMPLOYEE(Emp_Num(PK), Emp_Name, Emp_LastName, Emp_Email, Emp_Type)
- ORDER(Order_ID(PK), Order_Item(FK)

Subtype

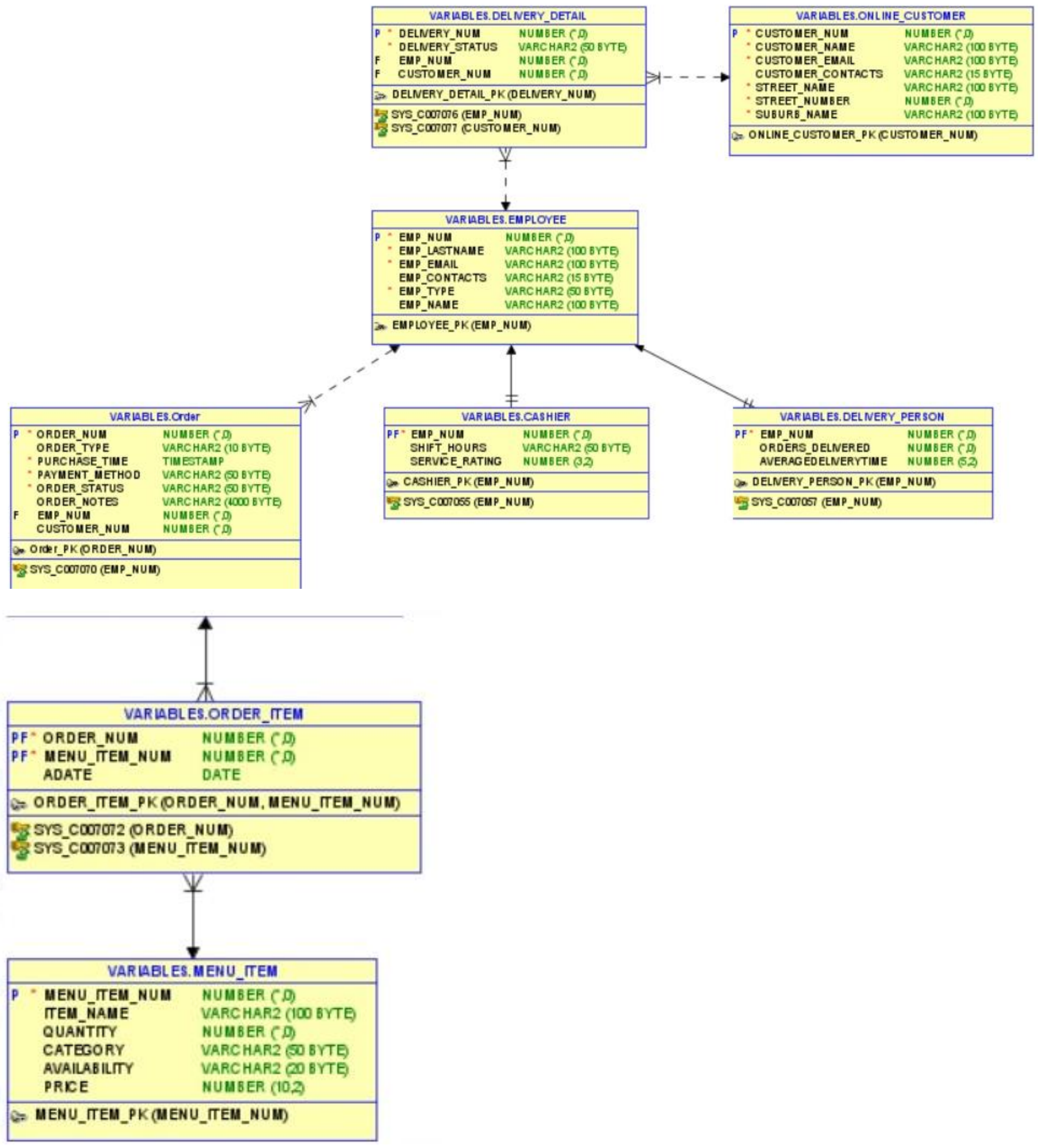
- ONLINE_ORDERS(Order_ID(PK), Customer_Id(FK))
- CASHIER(Emp_Num(PK)(FK), InStore_OrderID (FK))
- DELIVERY_PERSON(Emp_Num(PK)(FK), Online_OrderID(FK))
- ORDER(Order_ID(PK), Order_Items(FK))

Phase 3: Physical Design

Logical Data Model



Physical Data Model in Oracle

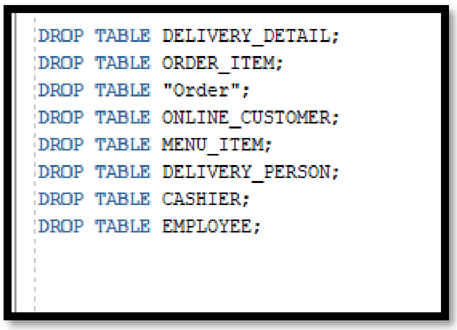

VARIABLES.DELIVERY_DETAIL
P * DELIVERY_NUM NUMBER (*,0)
* DELIVERY_STATUS VARCHAR2 (50 BYTE)
F EMP_NUM NUMBER (*,0)
F CUSTOMER_NUM NUMBER (*,0)
DELIVERY_DETAIL_PK (DELIVERY_NUM)
SYS_C007076 (EMP_NUM)
SYS_C007077 (CUSTOMER_NUM)
VARIABLES.ONLINE_CUSTOMER
P * CUSTOMER_NUM NUMBER (*,0)
* CUSTOMER_NAME VARCHAR2 (100 BYTE)
* CUSTOMER_EMAIL VARCHAR2 (100 BYTE)
CUSTOMER_CONTACTS VARCHAR2 (15 BYTE)
* STREET_NAME VARCHAR2 (100 BYTE)
* STREET_NUMBER NUMBER (*,0)
* SUBURB_NAME VARCHAR2 (100 BYTE)
ONLINE_CUSTOMER_PK (CUSTOMER_NUM)
VARIABLES.EMPLOYEE
P * EMP_NUM NUMBER (*,0)
* EMP_LASTNAME VARCHAR2 (100 BYTE)
* EMP_EMAIL VARCHAR2 (100 BYTE)
EMP_CONTACTS VARCHAR2 (15 BYTE)
* EMP_TYPE VARCHAR2 (50 BYTE)
EMP_NAME VARCHAR2 (100 BYTE)
EMPLOYEE_PK (EMP_NUM)
VARIABLES.Order
P * ORDER_NUM NUMBER (*,0)
ORDER_TYPE VARCHAR2 (10 BYTE)
* PURCHASE_TIME TIMESTAMP
* PAYMENT_METHOD VARCHAR2 (50 BYTE)
* ORDER_STATUS VARCHAR2 (50 BYTE)
ORDER_NOTES VARCHAR2 (4000 BYTE)
F EMP_NUM NUMBER (*,0)
CUSTOMER_NUM NUMBER (*,0)
Order_PK (ORDER_NUM)
SYS_C007070 (EMP_NUM)
VARIABLES.CASHIER
PF * EMP_NUM NUMBER (*,0)
SHIFT_HOURS VARCHAR2 (50 BYTE)
SERVICE_RATING NUMBER (3,2)
CASHIER_PK (EMP_NUM)
SYS_C007055 (EMP_NUM)
VARIABLES.DELIVERY_PERSON
PF * EMP_NUM NUMBER (*,0)
ORDERS_DELIVERED NUMBER (*,0)
AVERAGEDELIVERYTIME NUMBER (5,2)
DELIVERY_PERSON_PK (EMP_NUM)
SYS_C007057 (EMP_NUM)
VARIABLES.ORDER_ITEM
PF * ORDER_NUM NUMBER (*,0)
PF * MENU_ITEM_NUM NUMBER (*,0)
ADATE DATE
ORDER_ITEM_PK (ORDER_NUM, MENU_ITEM_NUM)
SYS_C007072 (ORDER_NUM)
SYS_C007073 (MENU_ITEM_NUM)
VARIABLES.MENU_ITEM
P * MENU_ITEM_NUM NUMBER (*,0)
ITEM_NAME VARCHAR2 (100 BYTE)
QUANTITY NUMBER (*,0)
CATEGORY VARCHAR2 (50 BYTE)
AVAILABILITY VARCHAR2 (20 BYTE)
PRICE NUMBER (10,2)
MENU_ITEM_PK (MENU_ITEM_NUM)

1. Database Object

In this section, we will be creating the initial database as well as creating all relationships between the different tables in the database. After the database has been created the database will be populated with default records that will be used to demonstrate our queries in section 3.

1.1 Initialisation

The following SQL statements are designed to drop or delete any existing tables that share names with the new tables we plan to create. This ensures that duplicate tables will not occur. We will remove the "child" entities first, followed by the "parent" entities for the sake of referential integrity.



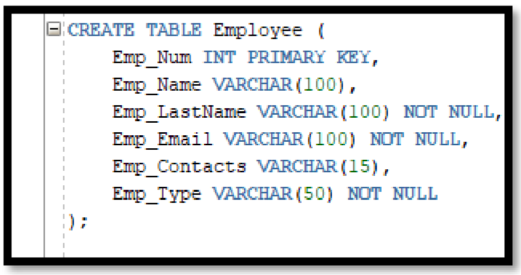
```sql
DROP TABLE DELIVERY_DETAIL;
DROP TABLE ORDER_ITEM;
DROP TABLE "Order";
DROP TABLE ONLINE_CUSTOMER;
DROP TABLE MENU_ITEM;
DROP TABLE DELIVERY_PERSON;
DROP TABLE CASHIER;
DROP TABLE EMPLOYEE;
```

Creation of Tables

The following statements were used to create the tables within our database. Constraints were included in this creation process. This includes the assigning of primary keys, foreign keys referencing their respective.

Employee Table

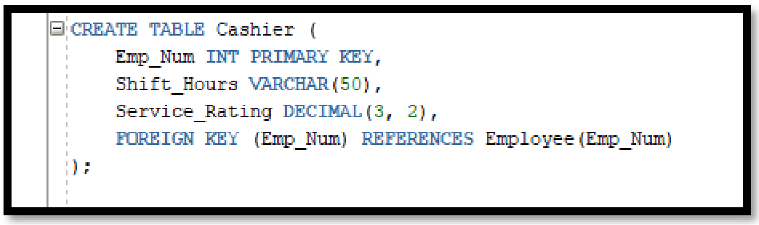
The Employee table stores information about the employees of the company. It is a supertype entity as there are two types of employees, the cashier, and delivery person which have their own tables with their unique attributes. The employee name, last name, email and employee type attributes all have a NOT NULL constraint as these are all important to be known by the company.



```sql
CREATE TABLE Employee (
     Emp_Num INT PRIMARY KEY,
     Emp_Name VARCHAR(100),
     Emp_LastName VARCHAR(100) NOT NULL,
     Emp_Email VARCHAR(100) NOT NULL,
     Emp_Contacts VARCHAR(15),
     Emp_Type VARCHAR(50) NOT NULL
);
```

Cashier Table

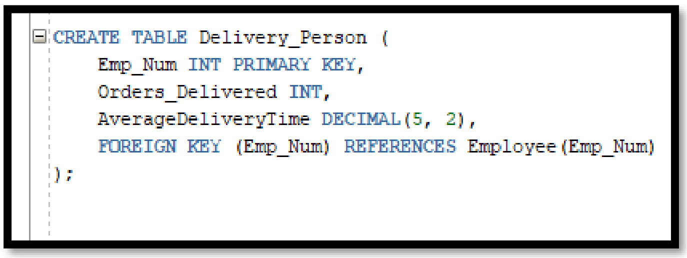
The Cashier table, which is one of two of the subtypes of Employee, stores information only specific to the employees that are cashiers. The first attribute kept is the primary key which serves as a unique identifier of each cashier. The next attribute is the shift hours the employees worked. Keeping track of shift hours can help in managing work schedules and avoiding scheduling conflicts. It also aids in calculating work hours for payroll purposes. The next stored variable is the service ratings which can help identify strengths and areas for improvement. The company can use this to award 'Employee of The Month' to the highest rated employee and provide training for the employees with lower ratings. This will overall improve customer satisfaction with Kasi Lifestyle. The foreign key establishes a relationship between this table and the employee table.



```
CREATE TABLE Cashier (
    Emp_Num INT PRIMARY KEY,
    Shift_Hours VARCHAR(50),
    Service_Rating DECIMAL(3, 2),
    FOREIGN KEY (Emp_Num) REFERENCES Employee(Emp_Num)
);
```

2.2.3 Delivery Person Table

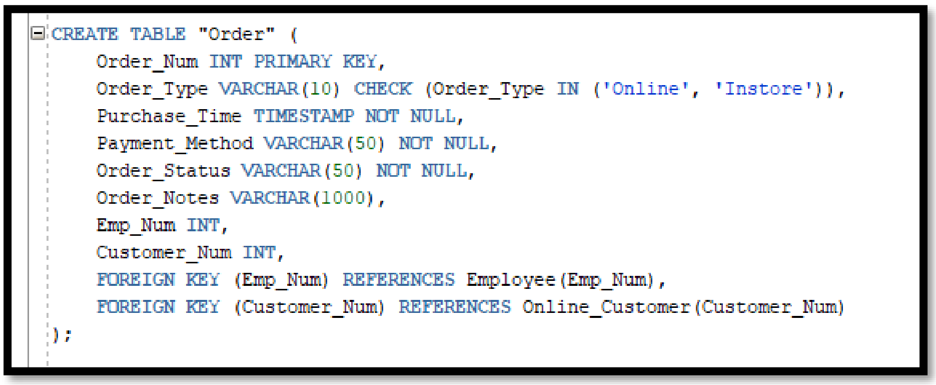
The Delivery person table is the other subtype of the employee table, and it stores information specific to the employees that will be delivering the order by the online customer. This is one of the operations that is new to the business and is crucial to why the database was created. It stores a unique identifier for each of the delivery people. Orders delivered, which helps measure performance, identifying high performers, and providing incentives or rewards based on the number of deliveries made. Average Delivery Time which helps identify potential issues with routes taken which can be used to identify alternative routes to use. All this will reduce delivery times.



```
CREATE TABLE Delivery_Person (
    Emp_Num INT PRIMARY KEY,
    Orders_Delivered INT,
    AverageDeliveryTime DECIMAL(5, 2),
    FOREIGN KEY (Emp_Num) REFERENCES Employee(Emp_Num)
);
```

2.2.4 Order Table

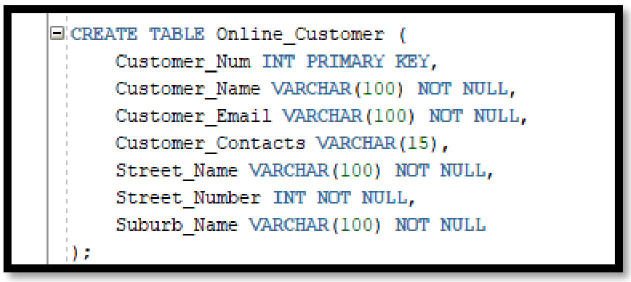
The order table stores information on what the customer orders whether online or instore. A primary key is a unique identifier for each order. Keeping track of the order type is crucial in knowing whether the online system is being utilised by majority of the customer base or not. keeping track of purchase times helps in identifying peak ordering periods, which can be used to assign highly rated customers to those specific peak hours in terms of shift hours. The type of payment used will help identify which payment method is most popular with the customers. If the consensus of customers pays in cash the business can use this in investing in more cash registers and if card, the business can use this to invest in POS machines. Order status helps ensure that orders are processed efficiently, and any issues are addressed. Capturing customer's order notes or instructions helps in maintaining customer satisfaction. Customer_Num to keep track of which customer made which order this value however can be null. It is null in the case that the order made is instore because the cashier logs in the order based on what the customer says they want verbally so Emp_Num will have the value of whichever cashier is logging the customer's order. Customer_Num will not be null when the order type is online because then the customer is directly logging the order into the database and the cashier is not involved. This means the Emp_Num in this case is null.



```sql
CREATE TABLE "Order" (
    Order_Num INT PRIMARY KEY,
    Order_Type VARCHAR(10) CHECK (Order_Type IN ('Online', 'Instore')),
    Purchase_Time TIMESTAMP NOT NULL,
    Payment_Method VARCHAR(50) NOT NULL,
    Order_Status VARCHAR(50) NOT NULL,
    Order_Notes VARCHAR(1000),
    Emp_Num INT,
    Customer_Num INT,
    FOREIGN KEY (Emp_Num) REFERENCES Employee(Emp_Num),
    FOREIGN KEY (Customer_Num) REFERENCES Online_Customer(Customer_Num)
);
```

2.2.5 Online Customer

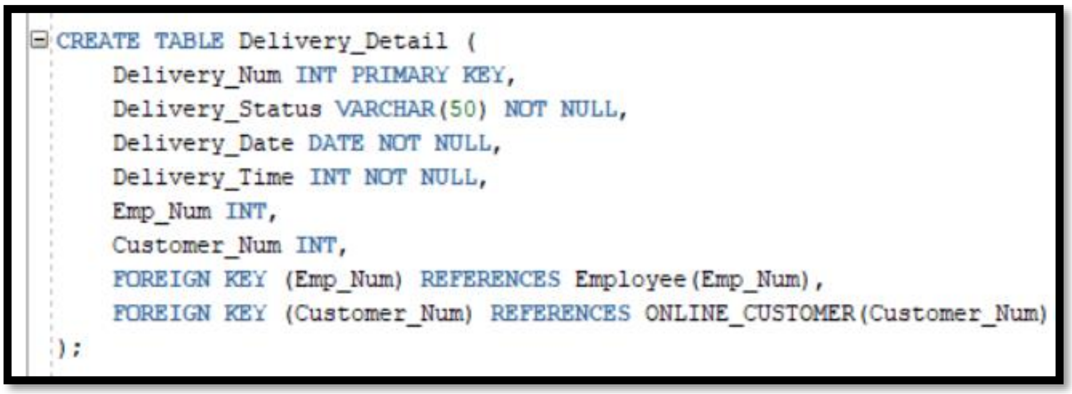
This table stores information about the customer such as their address, name, and contact details. This for the delivery person to be able to identify which order to take to whom. It also contains a primary key which uniquely identifies the customers.



```sql
CREATE TABLE Online_Customer (
    Customer_Num INT PRIMARY KEY,
    Customer_Name VARCHAR(100) NOT NULL,
    Customer_Email VARCHAR(100) NOT NULL,
    Customer_Contacts VARCHAR(15),
    Street_Name VARCHAR(100) NOT NULL,
    Street_Number INT NOT NULL,
    Suburb_Name VARCHAR(100) NOT NULL
);
```

2.2.6 Delivery Detail

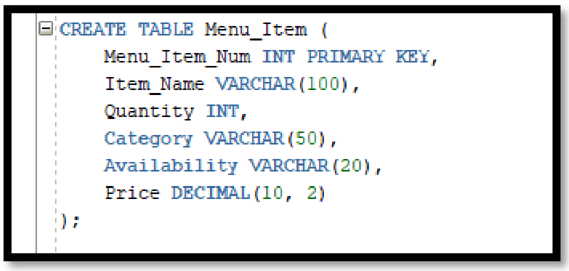
The Delivery Detail table is designed to store detailed information about each delivery, including a unique identifier, the status of the delivery, the date, and the time of the delivery. It also includes foreign keys to reference the Employee responsible for the delivery and the who placed the order. This design ensures tracking and management of the newly delivery operations of the business, linking each delivery to the respective employee and customer.



```
CREATE TABLE Delivery_Detail (
    Delivery_Num INT PRIMARY KEY,
    Delivery_Status VARCHAR(50) NOT NULL,
    Delivery_Date DATE NOT NULL,
    Delivery_Time INT NOT NULL,
    Emp_Num INT,
    Customer_Num INT,
    FOREIGN KEY (Emp_Num) REFERENCES Employee(Emp_Num),
    FOREIGN KEY (Customer_Num) REFERENCES ONLINE_CUSTOMER(Customer_Num)
);
```

2.2.7 Menu Item table

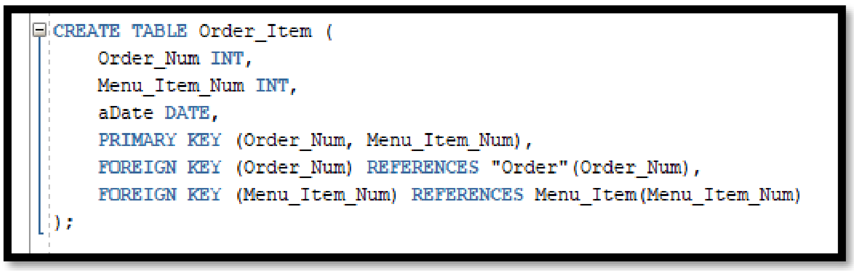
The Menu Item table stores the details of menu items available at the restaurant. It includes a unique identifier for each item, the name of the item, the available quantity, the category of the item which could be, food, drinks, or dessert. its availability status, and the price of the item. This design allows the restaurant to track inventory, categorize menu offerings, manage item availability, and set increase or decrease price based on how often an item is bought for example.



```
CREATE TABLE Menu_Item (
    Menu_Item_Num INT PRIMARY KEY,
    Item_Name VARCHAR(100),
    Quantity INT,
    Category VARCHAR(50),
    Availability VARCHAR(20),
    Price DECIMAL(10, 2)
);
```

2.2.8 Order Item table

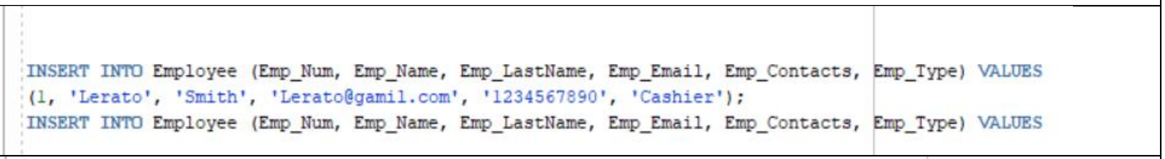
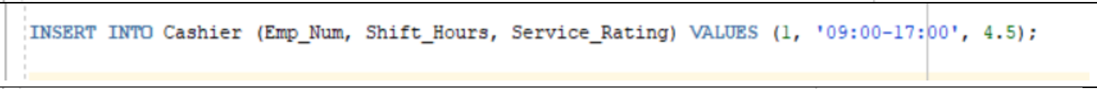
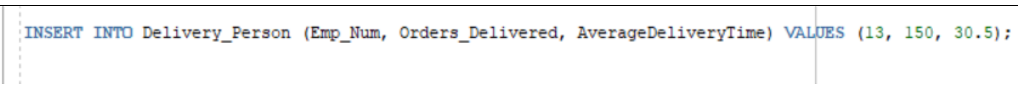
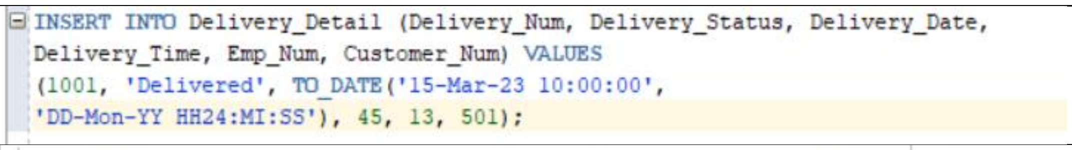
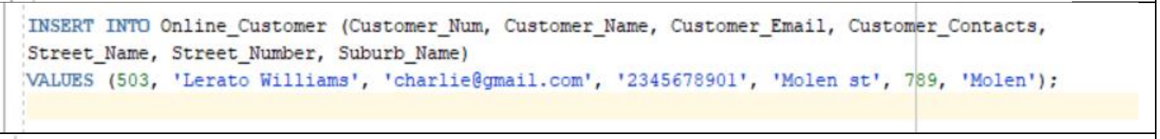
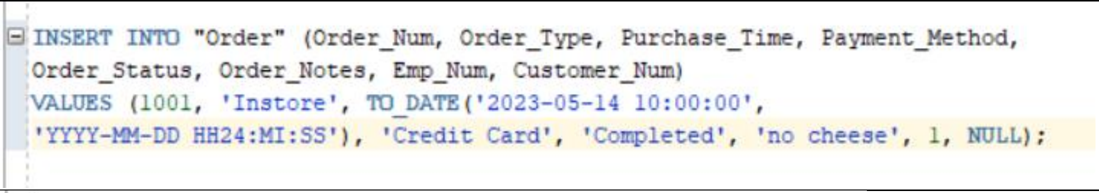
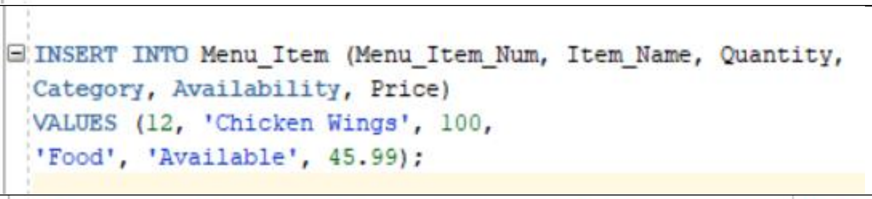
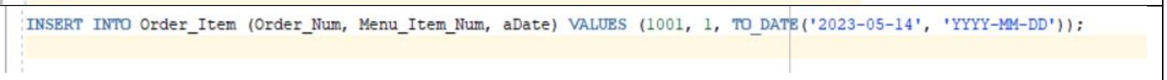
The Order Item table represents the relationship between orders and menu items in the restaurant's database. It includes the Order Num and `Menu Item Num columns to associate each order with the items it contains. The Date (named aDate because date is a reserved keyword) attribute stores the date when the order was placed. The primary key consists of both Order Num and Menu Item Num which makes this table a composite entity.



```
CREATE TABLE Order_Item (
    Order_Num INT,
    Menu_Item_Num INT,
    aDate DATE,
    PRIMARY KEY (Order_Num, Menu_Item_Num),
    FOREIGN KEY (Order_Num) REFERENCES "Order"(Order_Num),
    FOREIGN KEY (Menu_Item_Num) REFERENCES Menu_Item(Menu_Item_Num)
);
```

Populating The Database

Insert statements

Table	Insert Statement
Employee	 `INSERT INTO Employee (Emp_Num, Emp_Name, Emp_LastName, Emp_Email, Emp_Contacts, Emp_Type) VALUES (1, 'Lerato', 'Smith', 'Lerato@gamil.com', '1234567890', 'Cashier');` `INSERT INTO Employee (Emp_Num, Emp_Name, Emp_LastName, Emp_Email, Emp_Contacts, Emp_Type) VALUES`
Cashier	 `INSERT INTO Cashier (Emp_Num, Shift_Hours, Service_Rating) VALUES (1, '09:00-17:00', 4.5);`
Delivery Person	 `INSERT INTO Delivery_Person (Emp_Num, Orders_Delivered, AverageDeliveryTime) VALUES (13, 150, 30.5);`
Delivery Detail	 `INSERT INTO Delivery_Detail (Delivery_Num, Delivery_Status, Delivery_Date, Delivery_Time, Emp_Num, Customer_Num) VALUES (1001, 'Delivered', TO_DATE('15-Mar-23 10:00:00', 'DD-Mon-YY HH24:MI:SS'), 45, 13, 501);`
Online Customer	 `INSERT INTO Online_Customer (Customer_Num, Customer_Name, Customer_Email, Customer_Contacts, Street_Name, Street_Number, Suburb_Name) VALUES (503, 'Lerato Williams', 'charlie@gmail.com', '2345678901', 'Molen st', 789, 'Molen');`
Order	 `INSERT INTO "Order" (Order_Num, Order_Type, Purchase_Time, Payment_Method, Order_Status, Order_Notes, Emp_Num, Customer_Num) VALUES (1001, 'Instore', TO_DATE('2023-05-14 10:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Credit Card', 'Completed', 'no cheese', 1, NULL);`
Menu Item	 `INSERT INTO Menu_Item (Menu_Item_Num, Item_Name, Quantity, Category, Availability, Price) VALUES (12, 'Chicken Wings', 100, 'Food', 'Available', 45.99);`
Order Item	 `INSERT INTO Order_Item (Order_Num, Menu_Item_Num, aDate) VALUES (1001, 1, TO_DATE('2023-05-14', 'YYYY-MM-DD'));`

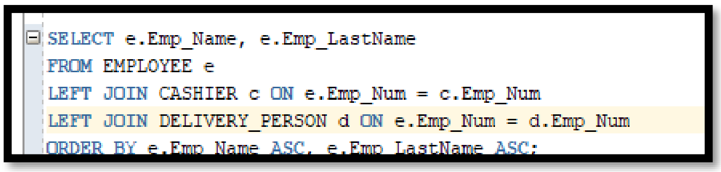
Tables and Their Queries

Employee

a) Employee Select

This acts as a basic employee directory, allowing for quick retrieval of employee information. This would allow sorting and filtering employees by type of employees they are along with their names. Retrieve a sorted list of employees, sorted on first names and then last names.(Sort)

1

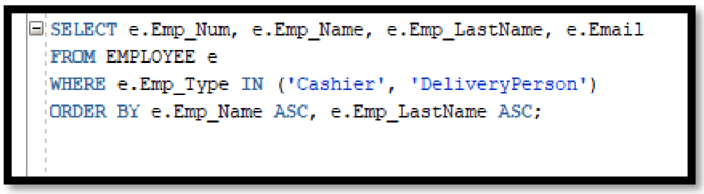


```sql
SELECT e.Emp_Name, e.Emp_LastName
FROM EMPLOYEE e
LEFT JOIN CASHIER c ON e.Emp_Num = c.Emp_Num
LEFT JOIN DELIVERY_PERSON d ON e.Emp_Num = d.Emp_Num
ORDER BY e.Emp_Name ASC, e.Emp_LastName ASC;
```

b) Employee Filter

This enhanced query effectively filters and sorts employee data based on their type, providing valuable information for various business purposes. It offers better efficiency by filtering unnecessary data (employees who are not cashiers or deliver person).

2



```sql
SELECT e.Emp_Num, e.Emp_Name, e.Emp_LastName, e.Email
FROM EMPLOYEE e
WHERE e.Emp_Type IN ('Cashier', 'DeliveryPerson')
ORDER BY e.Emp_Name ASC, e.Emp_LastName ASC;
```

CASHIER

Cashiers is a subtype of EMPLOYEE and contain attributes specific to each cashier but not to a delivery person.

Constraint

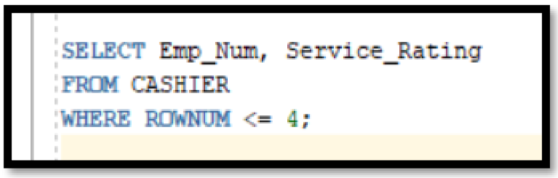
Constraints for table CASHIER

Limitation of rows and columns

Display only the first four record of cashier.

The ROWNUM <= 4 condition limits the output to the first 4 rows of the table.

3

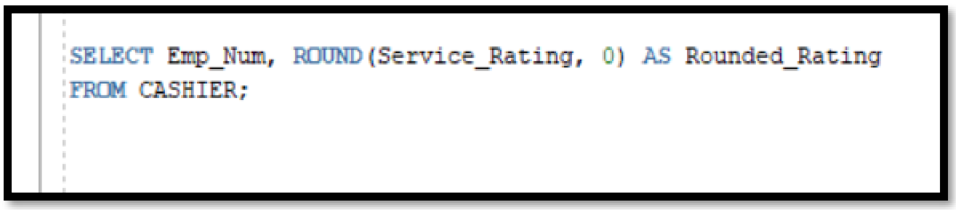


```
SELECT Emp_Num, Service_Rating
FROM CASHIER
WHERE ROWNUM <= 4;
```

Round

This query retrieves the service rating for each cashier, rounded to the nearest whole number.

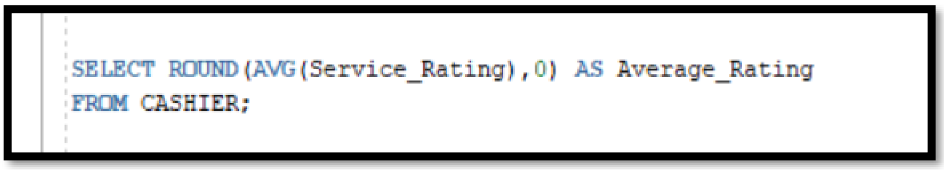
4



```
SELECT Emp_Num, ROUND(Service_Rating, 0) AS Rounded_Rating
FROM CASHIER;
```

Aggregate functions
Calculates the average service rating (Average_Rating) for all cashiers from the CASHIER table and rounds the resulting average to the nearest whole number.

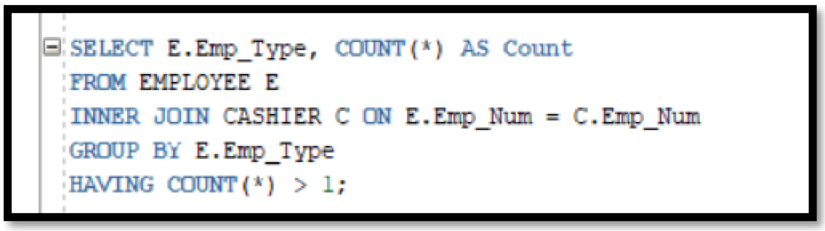
5



```
SELECT ROUND(AVG(Service_Rating),0) AS Average_Rating
FROM CASHIER;
```

Group by and Having
This query lies in its ability to provide a count of cashiers in the organization, grouped by their employee type, but only where there is more than one cashier of that type. This information could be useful for management to understand the distribution of cashier roles among different types of employees.

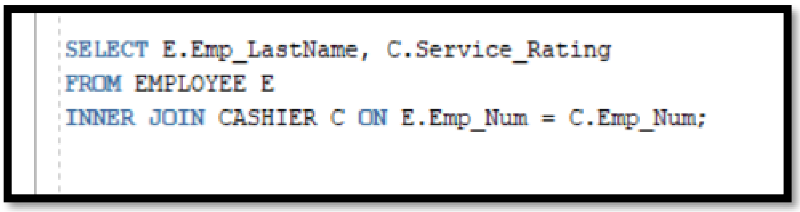
6



```
SELECT E.Emp_Type, COUNT(*) AS Count
FROM EMPLOYEE E
INNER JOIN CASHIER C ON E.Emp_Num = C.Emp_Num
GROUP BY E.Emp_Type
HAVING COUNT(*) > 1;
```

Join
Fetches last names and service ratings of all cashiers.

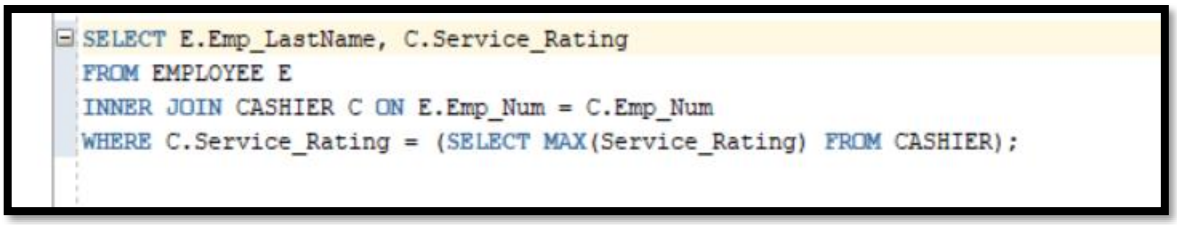
7



```
SELECT E.Emp_LastName, C.Service_Rating
FROM EMPLOYEE E
INNER JOIN CASHIER C ON E.Emp_Num = C.Emp_Num;
```

Sub-queries
Fetches last names and service ratings of cashiers with the highest service rating.

8

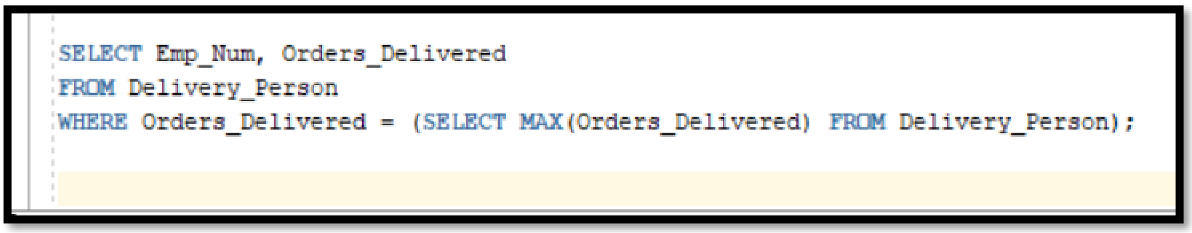


```
SELECT E.Emp_LastName, C.Service_Rating
FROM EMPLOYEE E
INNER JOIN CASHIER C ON E.Emp_Num = C.Emp_Num
WHERE C.Service_Rating = (SELECT MAX(Service_Rating) FROM CASHIER);
```

Delivery Person

Delivery Person with the Highest Number of Orders Delivered.

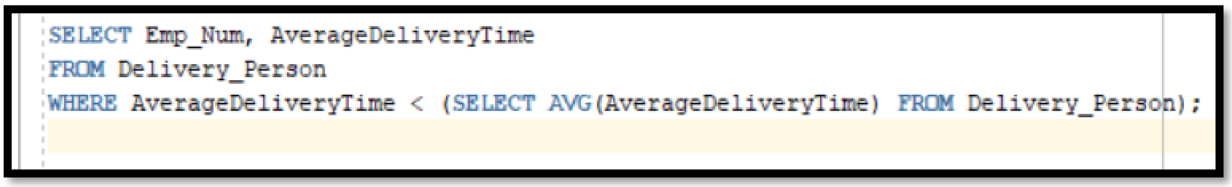
13



```
SELECT Emp_Num, Orders_Delivered
FROM Delivery_Person
WHERE Orders_Delivered = (SELECT MAX(Orders_Delivered) FROM Delivery_Person);
```

Delivery Persons with Below Average Delivery Times,

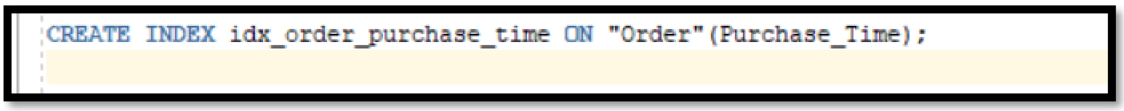
14



```
SELECT Emp_Num, AverageDeliveryTime
FROM Delivery_Person
WHERE AverageDeliveryTime < (SELECT AVG(AverageDeliveryTime) FROM Delivery_Person);
```

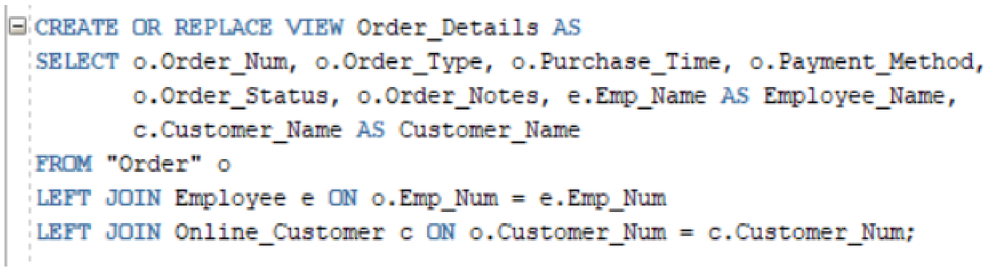
Order

Index



```
CREATE INDEX idx_order_purchase_time ON "Order"(Purchase_Time);
```

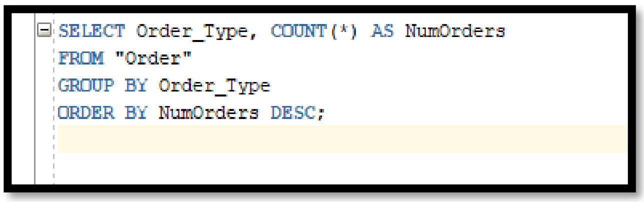
Views



```sql
CREATE OR REPLACE VIEW Order_Details AS
SELECT o.Order_Num, o.Order_Type, o.Purchase_Time, o.Payment_Method,
       o.Order_Status, o.Order_Notes, e.Emp_Name AS Employee_Name,
       c.Customer_Name AS Customer_Name
FROM "Order" o
LEFT JOIN Employee e ON o.Emp_Num = e.Emp_Num
LEFT JOIN Online_Customer c ON o.Customer_Num = c.Customer_Num;
```

which order type is made most

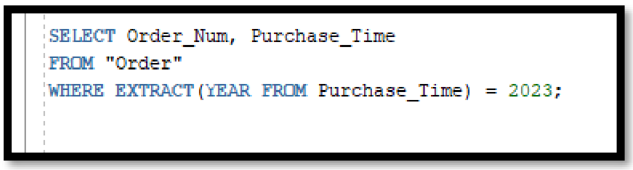
15



```sql
SELECT Order_Type, COUNT(*) AS NumOrders
FROM "Order"
GROUP BY Order_Type
ORDER BY NumOrders DESC;
```

---orders made for a specific year------

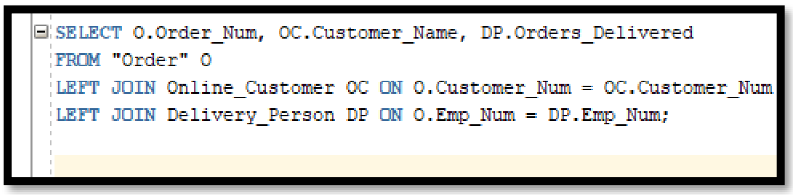
16



```sql
SELECT Order_Num, Purchase_Time
FROM "Order"
WHERE EXTRACT(YEAR FROM Purchase_Time) = 2023;
```

-----This query retrieves order numbers, customer names, and the number of orders delivered by the respective delivery person---------

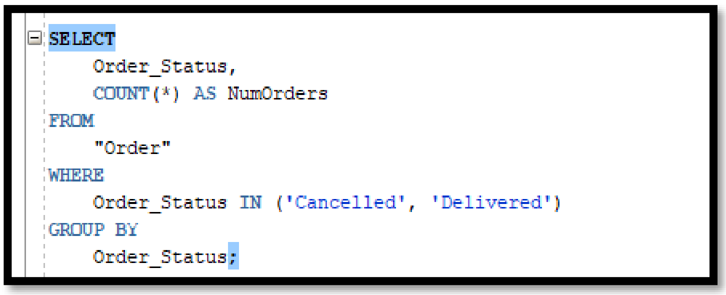
18



```sql
SELECT O.Order_Num, OC.Customer_Name, DP.Orders_Delivered
FROM "Order" O
LEFT JOIN Online_Customer OC ON O.Customer_Num = OC.Customer_Num
LEFT JOIN Delivery_Person DP ON O.Emp_Num = DP.Emp_Num;
```

----orders delivered vs orders cancelled----

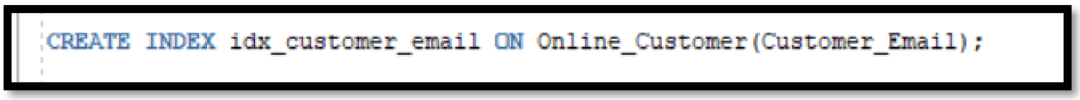
19



```sql
SELECT
    Order_Status,
    COUNT(*) AS NumOrders
FROM
    "Order"
WHERE
    Order_Status IN ('Cancelled', 'Delivered')
GROUP BY
    Order_Status;
```

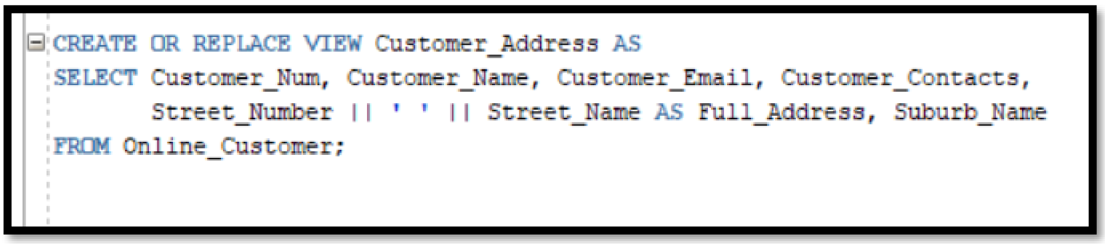
Online Customer

Index



```sql
CREATE INDEX idx_customer_email ON Online_Customer(Customer_Email);
```

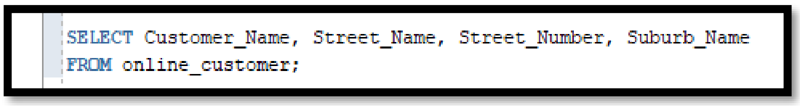
Views



```sql
CREATE OR REPLACE VIEW Customer_Address AS
SELECT Customer_Num, Customer_Name, Customer_Email, Customer_Contacts,
       Street_Number || ' ' || Street_Name AS Full_Address, Suburb_Name
FROM Online_Customer;
```

---Get customers and thier address for delivery person----

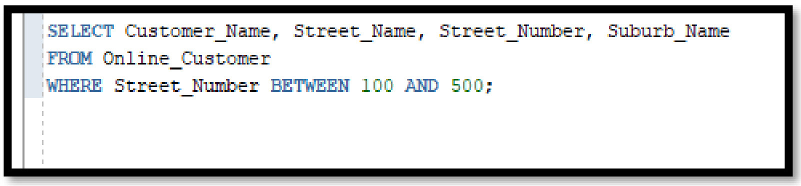
20



```sql
SELECT Customer_Name, Street_Name, Street_Number, Suburb_Name
FROM online_customer;
```

------House numbers in specific range--------------

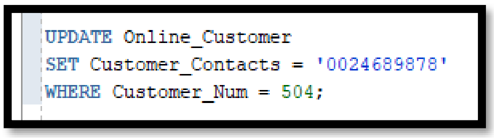
21



```sql
SELECT Customer_Name, Street_Name, Street_Number, Suburb_Name
FROM Online_Customer
WHERE Street_Number BETWEEN 100 AND 500;
```

------CUSTOMERS WANT TO CHANGE THIER CONTATCS---------------

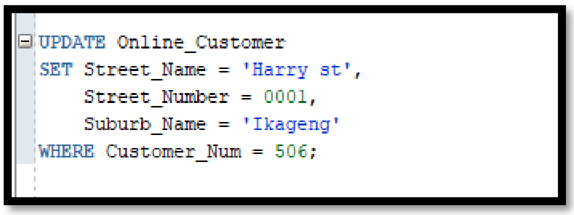
22



```sql
UPDATE Online_Customer
SET Customer_Contacts = '0024689878'
WHERE Customer_Num = 504;
```

-------Customers moved geographical location------------

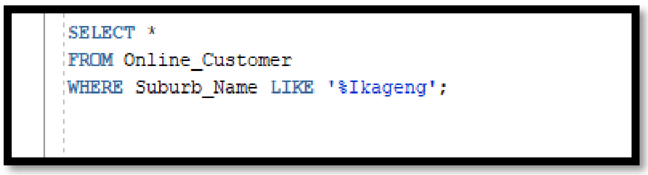
23



```sql
UPDATE Online_Customer
SET Street_Name = 'Harry st',
    Street_Number = 0001,
    Suburb_Name = 'Ikageng'
WHERE Customer_Num = 506;
```

--------Find Customers who live in the same suburb--------

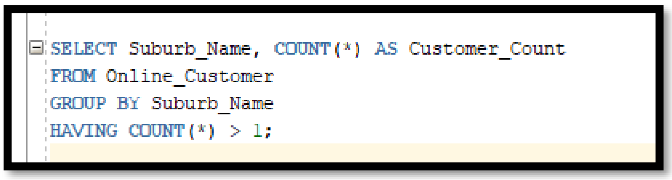
24



```sql
SELECT *
FROM Online_Customer
WHERE Suburb_Name LIKE '%Ikageng';
```

-------calculating number of customers in each suburb---------

25



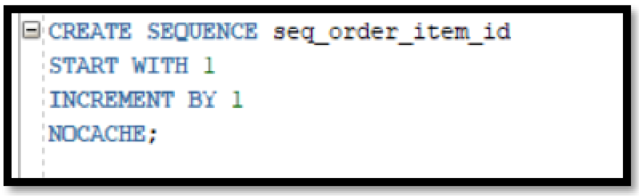
```sql
SELECT Suburb_Name, COUNT(*) AS Customer_Count
FROM Online_Customer
GROUP BY Suburb_Name
HAVING COUNT(*) > 1;
```

Order item

Order items placed online, or in-store can all be grouped together by the system thanks to the Order_Num column, which associates each order item with a unique order in the ORDER table. With its relationship to the MENU_ITEM table, Menu_Item_Num uniquely identifies the menu item that is being requested, guaranteeing precise identification of every item in an order. To help with order tracking and sales report generation, the Date column keeps track of the order item's addition date. The quantity of each menu item requested is specified, and this information is essential for stock level changes and inventory management. In order to guarantee that the customer is invoiced the exact amount, the Price field captures the whole cost for the desired quantity. Order_Num and Menu_Item_Num make up the composite primary key, which guarantees that every combination is distinct and avoids duplicate entries.

SEQUENCE:

Sequences for the Order_Item table is used to automatically produce unique identifiers, albeit they are not expressly necessary. This is helpful in case more unique fields are required later on. Every new number created will be one greater than the last by beginning the sequence at 1 (START WITH 1) and increasing by 1 (INCREMENT BY 1). In order to prevent gaps in sequence numbers and maintain numbering continuity, the NOCACHE option makes sure that sequence values are not cached.

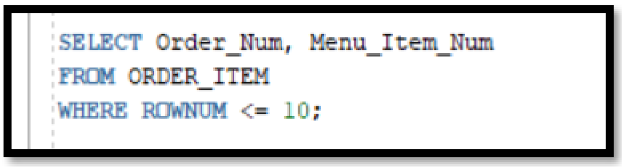


```sql
CREATE SEQUENCE seq_order_item_id
START WITH 1
INCREMENT BY 1
NOCACHE;
```

QUERIES

--------------------Limitation of Rows and Columns----------------------

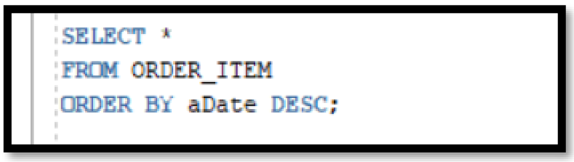
27



```
SELECT Order_Num, Menu_Item_Num
FROM ORDER_ITEM
WHERE ROWNUM <= 10;
```

-------------------Sorting----------------------

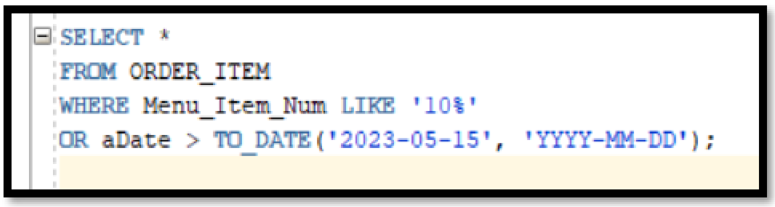
28



```
SELECT *
FROM ORDER_ITEM
ORDER BY aDate DESC;
```

-------------------LIKE, AND, OR----------------------

29

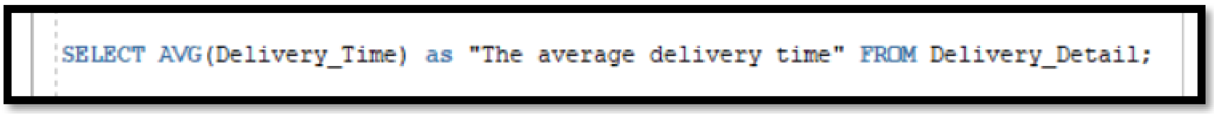


```
SELECT *
FROM ORDER_ITEM
WHERE Menu_Item_Num LIKE '10%'
OR aDate > TO_DATE('2023-05-15', 'YYYY-MM-DD');
```

Delivery Detail

Display the average time that a delivery takes

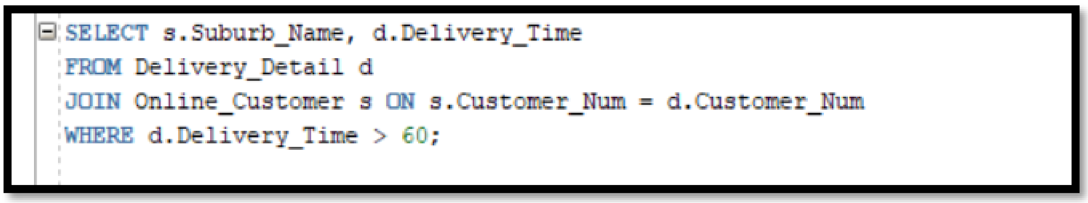
30



```
SELECT AVG(Delivery_Time) as "The average delivery time" FROM Delivery_Detail;
```

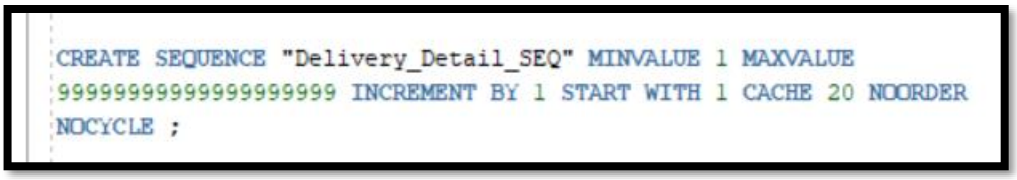
JOIN -Retrival of all the suburb that that more than 60 mins to deliver to

31



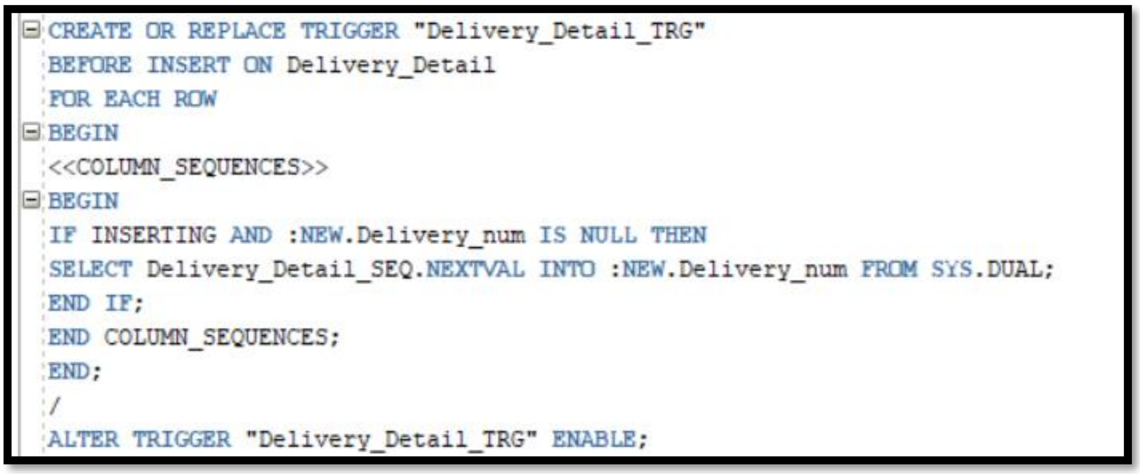
```
SELECT s.Suburb_Name, d.Delivery_Time
FROM Delivery_Detail d
JOIN Online_Customer s ON s.Customer_Num = d.Customer_Num
WHERE d.Delivery_Time > 60;
```

Sequence



```
CREATE SEQUENCE "Delivery_Detail_SEQ" MINVALUE 1 MAXVALUE
99999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER
NOCYCLE ;
```

Trigger

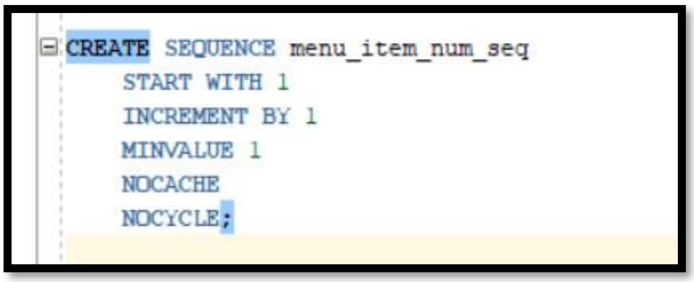


```
CREATE OR REPLACE TRIGGER "Delivery_Detail_TRG"
BEFORE INSERT ON Delivery_Detail
FOR EACH ROW
BEGIN
<<COLUMN_SEQUENCES>>
BEGIN
IF INSERTING AND :NEW.Delivery_num IS NULL THEN
SELECT Delivery_Detail_SEQ.NEXTVAL INTO :NEW.Delivery_num FROM SYS.DUAL;
END IF;
END COLUMN_SEQUENCES;
END;
/
ALTER TRIGGER "Delivery_Detail_TRG" ENABLE;
```

Menu Item

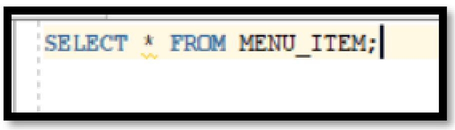
This is the sequence query for the MENU_ITEM table:

This sequence statement would be necessary to maintain data integrity, ensure uniqueness, and improve performance and consistency in the database operations within a company.



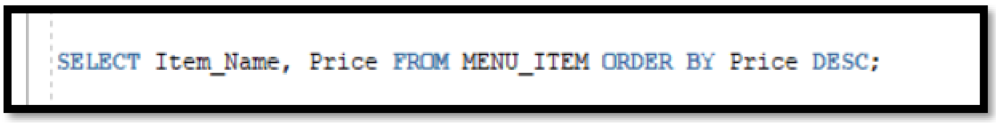
```
CREATE SEQUENCE menu_item_num_seq
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NOCACHE
    NOCYCLE;
```

Retrieves all data from the MENU_ITEM table. This is useful for getting a complete overview of the menu items, which can be important for inventory checks or menu planning.



```sql
SELECT * FROM MENU_ITEM;
```

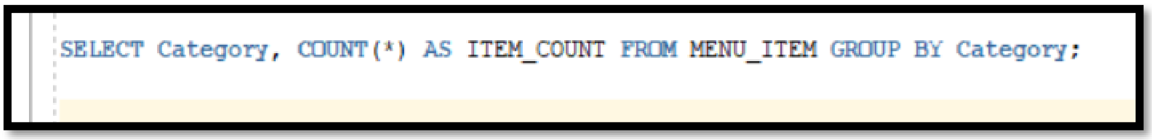
Lists menu items and their prices in descending order. This helps in analyzing the pricing structure and identifying the most expensive items, which can inform pricing strategies and menu design.



```sql
SELECT Item_Name, Price FROM MENU_ITEM ORDER BY Price DESC;
```

Counts the number of items in each category. This information is important for understanding the distribution of menu items and can help in category management and promotional planning.

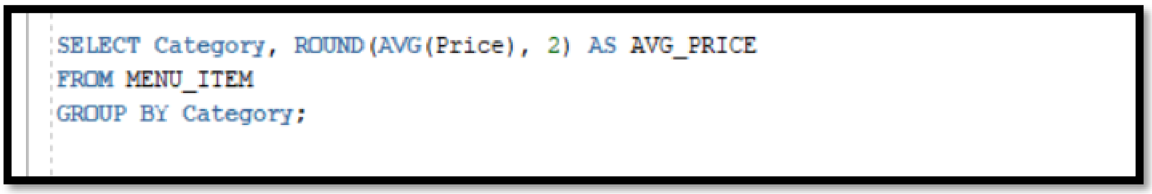
32



```sql
SELECT Category, COUNT(*) AS ITEM_COUNT FROM MENU_ITEM GROUP BY Category;
```

Calculates the average price of items within each category. This aids in pricing analysis and can help ensure that each category is priced appropriately according to business strategy.

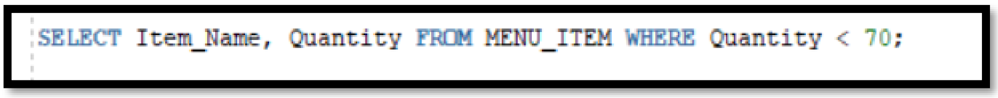
33



```sql
SELECT Category, ROUND(AVG(Price), 2) AS AVG_PRICE
FROM MENU_ITEM
GROUP BY Category;
```

Finds items with low stock levels. This is critical for inventory control and can trigger restocking orders to prevent stockouts.

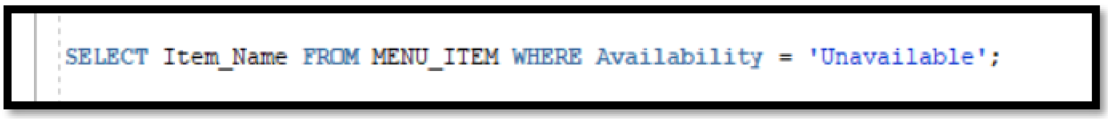
34



```sql
SELECT Item_Name, Quantity FROM MENU_ITEM WHERE Quantity < 70;
```

Identifies items that are currently unavailable. This is important for customer service and menu updates, ensuring that unavailable items are not promised to customers.

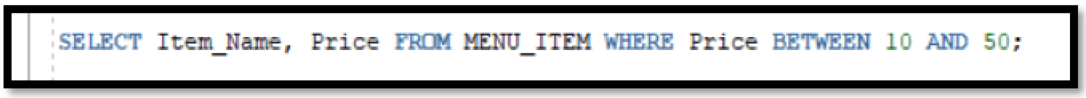
35



```
SELECT Item_Name FROM MENU_ITEM WHERE Availability = 'Unavailable';
```

Lists items within a specific price range. This can be used for creating promotions or special offers targeting a certain price point.

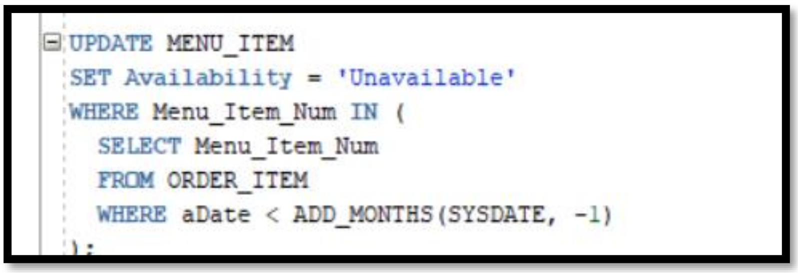
36



```
SELECT Item_Name, Price FROM MENU_ITEM WHERE Price BETWEEN 10 AND 50;
```

Marks items as unavailable if they haven't been ordered in over a month. This helps in menu optimization and reducing waste by phasing out unpopular items.

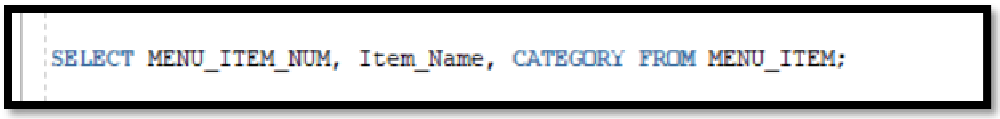
37



```
UPDATE MENU_ITEM
SET Availability = 'Unavailable'
WHERE Menu_Item_Num IN (
  SELECT Menu_Item_Num
  FROM ORDER_ITEM
  WHERE aDate < ADD_MONTHS(SYSDATE, -1)
);
```

Retrieves specific columns from the MENU_ITEM table. This limits the data to only what is necessary, which can improve performance and focus analysis on key attributes.(limitation)

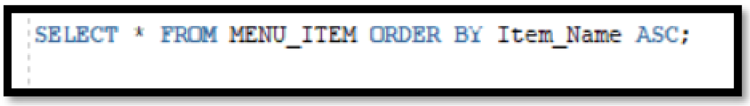
38



```
SELECT MENU_ITEM_NUM, Item_Name, CATEGORY FROM MENU_ITEM;
```

Sorts menu items alphabetically. This is useful for reports and displays where items need to be listed in a readable and organized manner.(sorting)

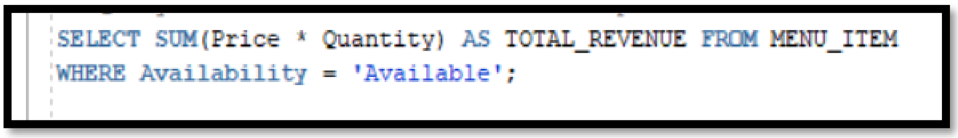
39



```
SELECT * FROM MENU_ITEM ORDER BY Item_Name ASC;
```

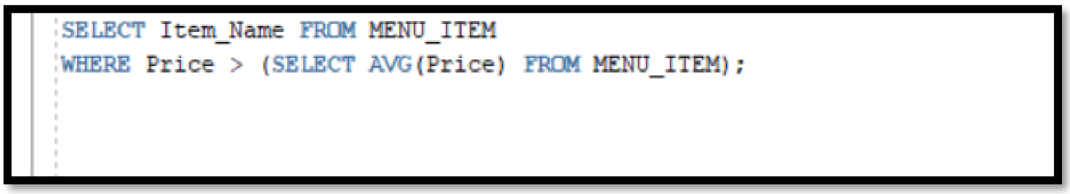
Calculates the total potential revenue from available items. This is crucial for financial forecasting and assessing the revenue impact of the current inventory.(aggregate)

40

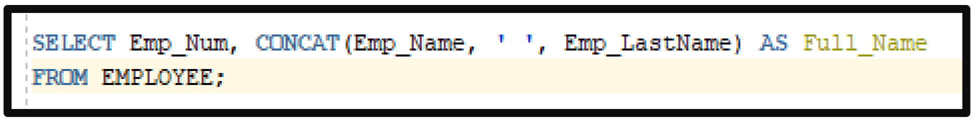

```sql
SELECT SUM(Price * Quantity) AS TOTAL_REVENUE FROM MENU_ITEM
WHERE Availability = 'Available';
```

Identifies items priced above the average. This can be used to evaluate the competitiveness of pricing and identify premium offerings.(subquery)

41


```sql
SELECT Item_Name FROM MENU_ITEM
WHERE Price > (SELECT AVG(Price) FROM MENU_ITEM);
```

42


```sql
SELECT Emp_Num, CONCAT(Emp_Name, ' ', Emp_LastName) AS Full_Name
FROM EMPLOYEE;
```

Queries

Limitation of rows and columns	7, 27, 38
Sorting	1, 2, 28, 39
LIKE, AND and OR	12, 29, 13, 21, 36
Variables and character functions	42
Round or trunc	8
Date functions	29
Aggregate functions	9, 30, 33, 40
Group by and having	17
Joins	6, 10, 11, 12, 18
Sub-queries	12, 13, 14