CMPG311 PROJECT

Phase 1

APRIL 15, 2022

VAAL IT SOLUTIONS

Table of Contents

1.Group members

Babedi Paai (Group leader)
076 927 0747
babedipaai19@gmail.com

Omphemetse Senna
073 540 0379
omphesenna@gmail.com
www.johnsenna@gmail.com

Nokuthula Embane
073 405 4498
nokuthulagladys00@gmail.com

2. Background Information

The name of the restaurant is Pelo-Entle, and it was established in 2021 by the owner. They specialise in African cuisine. They cook their food with love. Hence, the name “Pelo-Entle”. They want to expand their business to a mobile catering business, not that they will be closing their restaurant. Meaning that they will also be catering for people in their own homes. They want to cater for weddings, funerals, graduations, parties, baby-showers, and dinners for two.
They want a system (website) that will enable their customers to book for appointments and order food online. Since well the restaurant do not have a system used to keep their records, the system (website) will enable them to keep their records safe.

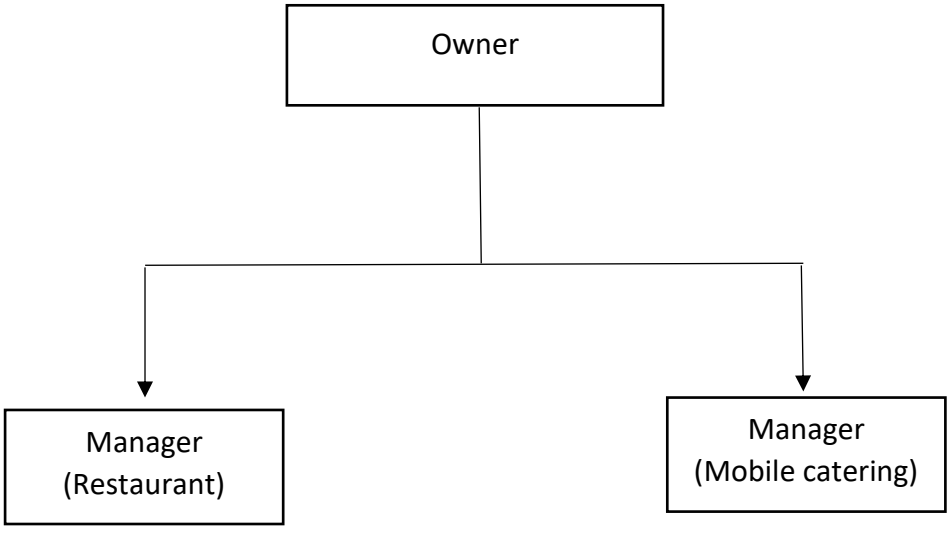
3. Company Objectives

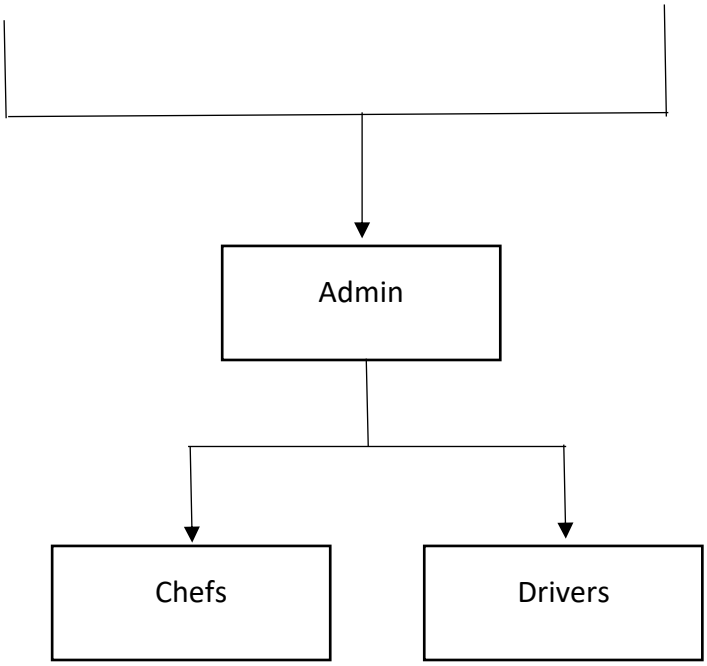
- Have a high number of bookings
- Offer quick and efficient services
- Have a high level of satisfaction from customers
- Allow customers to track down the status of their orders
- Improve communication between customers and the admin
- Increase business profit
- Hire more chefs as the business grow
- Have a friendly working environment
- Increase sales
- Have a user-friendly website

4. Company Operations

- When customers want to book for catering services or for ordering food, they should sign in or sign up into the system.
- Signing-in is for old customers. New customers must sign-up and create an account so that their information can be added to the database.
- Their details will be used to recognize them the next time they want to book or order.
- The admin staff is the one responsible for locating information of customers that have previously booked for catering services or ordered food.
- Once a customer has successfully signed-in to the system, they must choose the type of service they need. It may be ordering food or booking catering services which is a house call.
- For house calls, they must choose the type of event that we will be catering for. It may be for a wedding, birthday party, funeral, baby showers, dinner or graduation and enter their home address.
- They will need to select the type of food they want us to cook for them and the date and time they need our services.
- The customer must also specify the number of people we will cater for to determine the number of staff/chefs to be allocated.
- For ordering food, there will be a menu for food, drinks, and desserts that we offer in which they must choose from.
- Customers need to select if they will fetch their food at the restaurant or want their food to be delivered. For delivering, they will have to enter their home address where the food will be delivered.
- Customers are normally given discounts for catering services depending on the number of people attending the event.
- Then the customer will choose the payment method between online payment and cash.
- The system needs to calculate the money that will be paid, and then the amount will be calculated using the type of food the customers chose and how many people we will cater for.
- The customer will then have to receive a receipt or a confirmation email or SMS with their details that include their names, date and time of service delivery, address, the type of food to be cooked, the number of people, the amount to be paid, and the method they chose to use to payment.
- For those who ordered food they will receive a call once the delivery people are at their gate.
- The website will be user-friendly, as it will accommodate every user.

5. Company Organisational Structure





- The restaurant has one owner.
- There are two managers. One for the restaurant and one for the mobile catering.
- The manager who manages the restaurant, manages chefs who are cooking in the restaurant, and waiters and waitresses. The manager also manages the orders made through the website.
- The manager that manages the mobile catering, has an admin, and manages the chefs and the drivers.
- There are chefs that will be cooking at the restaurant, and there are others who will be cooking at the customer's preferable place (their own homes), who will be booked.
- The drivers will be delivering food that has been ordered using the website.

6. Problems and Constraints

- Customers come personally to the restaurant to book for a catering service.
- There is no effective communication between customers and the staff.
- Bookings are written in a book and it sometimes disappears.

- Bookings are mixed up.
- There is no system used to capture the details of the client.
- Customers complain about poor service.
- Chefs are sometimes overwhelmed by so many orders and makes mistakes.
- Drivers get lost as the addresses of the customers get mixed up.
- The mobile catering manager cannot get the number of people the chefs will be cooking for. The manger ends up sending more chefs for a small group of people and send few chefs for a large group of people.
- The payment system for the mobile catering and delivery has glitches.

7. Define objectives to solve problems identified

- Our historical information and current information will be utilized in future to assist with working on our administrations and our framework and furthermore to foresee what's to come.
- Not every representative or employee will be permitted to get to the database but the pertinent staff. Doing this will assist with safeguarding our clients' information.
- Our database will be observed no matter what and will be updated consistently.
- Clients' private information will be safeguarded and remained careful of aggressors.
- Our framework will have input validators to help the clients during the solicitation for our administrations.
- There will be guidelines or documentation to assist the client with the site.
- Clients will have few choices of Booking either on the web, through chats, calls, or in person.
- Every client's Bookings will be kept saved in our database and each Booking will have its unique Booking ID. Microsoft Excel will be used as a backup.
- Time usage will be incorporated to assist our representatives to stay organized so they can offer decent support to our clients.
- There will be an extra textbox part for the portrayal where the client can add routes, and any valuable data to help our drivers or courier in complex routes during deliveries.
- More chefs will be recruited to help the current ones.
- Our lines, social media and emails will work during administration times to assist the clients and our representatives with willing be quick to answer questions or queries.
- All the details (sign in, sign up, conveyance) of our clients will be kept in the database on their entities and their passwords will be encrypted before being entered into the database.

8. Information that company requires from database

- The company will require the details of the customers such as, names, surname, cell phone number, and address when the customers sign up.
- The company will need to know if the customer is booking for an appointment or ordering food.
- If the customer is booking for an appointment, the company will need to know the event/occasion they will be rendering their services, what type of food the chefs will be cooking at the customer’s home, the number of people the chefs will be cooking for, and the date and time.
- If the customer is ordering food, the company will need to know what type of food the customer will want and whether the customer wants the food to be delivered to their homes or they will collect the food on their own.
- When the customer books for appointment or is ordering food, the company will need to know the payment method the customer will use. Between online payment and cash.
- If the customer chooses online payment method. The company will need to know the customer’s bank card, the expiry date and the CVV number to process the booking or order.

9. Scope

The scope of this project is to design and implement a database that will be accessed and managed. The organization has two business Functions or essentially offices that fall under the proprietor (owner) and they are Managers: Restaurant and Mobile catering. Different access will be granted to every representative based on their department. The database will cover both Managerial departments (Restaurant and Mobile catering).

Manager departments: Focus on managing, planning, organising, and controlling divisions that are under them, which are the admin, chef, and driver's departments.

Admin: This segment will approach all entities of the division of Mobile catering which are order, delivery, booking, payment, and the rest. The emphasis will be on evaluating the various jobs of two offices (Chef and driver) after receiving orders from the customer. The administrator will give the expected set of responsibilities to the chef department based on the orders of the customer, from that point the Chef department will offer back the outcomes to the administration office, then the administrator office will either give another role to the driver division or not in view of the solicitation of the customer.

Chef: This department focuses on receiving orders from the admin department. After receiving the orders from the admin department, this department will process that information to produce the best output and send it back to the admin department then the admin department will give it to the driver department or customer.

Driver: This department focuses on delivery and payments. If the admin department received the option for delivery and payments method, then that is when the **driver** department will have to play a role.

10. Boundaries

Boundaries can assist the representatives with adaptability while working on the project. Boundaries are there to give every representative working on the task a few points on making decisions about the initial limits that will not completely affect their life, for instance, investing all their energy in the improvement of the framework. For that reason, the representatives have consented to stick to the following Boundaries.

Financial Boundary: The project costs a lot of money, and some might take less, and some might take a lot, that is why the need of financial guidelines are needed to avoid depth time after time.

Budget Boundary: Since well there is no old system, the representatives will have to work on a flexible budget. The system needs to be implemented as soon as possible. Therefore, more money will be spent on the project.

Software Boundaries: Only windows 8,9, and 11 will be required or needed to support the framework. Windows 7 can also be supported but only if it is a professional vision. Browsers that can support the front-end(website) part are chrome, Firefox, and explorer. Other browsers are likely to not support the front-end because of versions and other unknown issues in those browsers. Update of the browsers and computers will be highly recommended over time.

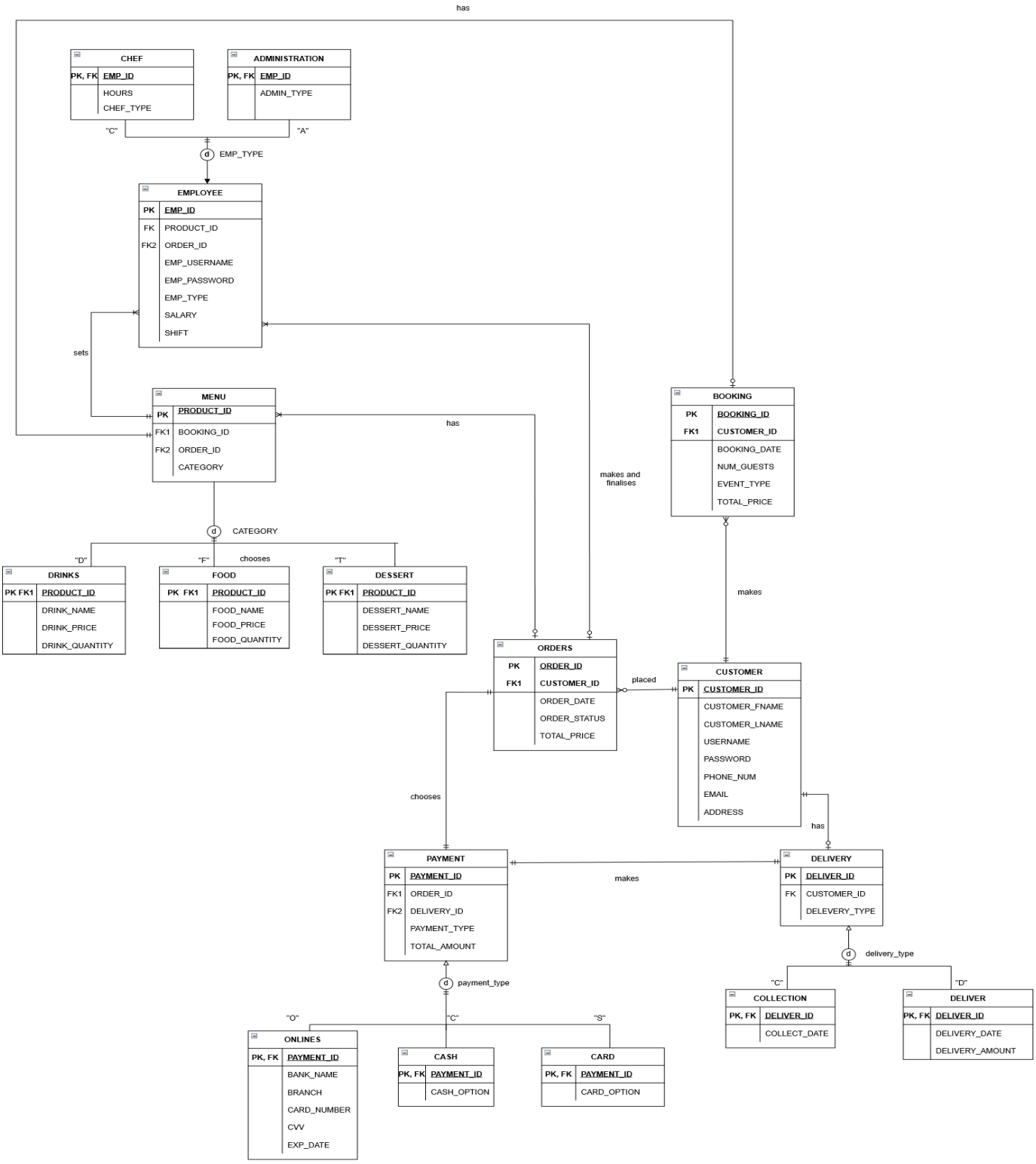
Time and Personal Boundaries: During the development of the database, there will be 4 representatives working on the design and implementation of the database. The assessed opportunity to finish the advancement of the database will generally be 5 months. Days will be 4 days per week, yet representatives can continue keeping on working whenever. For instance, working on the weekends but it will be of their free will. But every individual will be expected during those 4 days.

Intellectual Boundaries: Every representative on the project must respect each member's input because, every member has his or her own opinions to present. This will permit each representative to show their insight by being free and to add their opinions, and ideas to the project.

11. Business Rules

-Employees can set only one menu.
-A menu can be set by one or many employees.
-Orders can be finalized by zero or more employees.
-Employees can finalize many orders.
-An order can be placed by one and only one customer.
-Customers can place zero or many orders.
-An order can have one or many payment methods.
-Each payment has one order.
-Menu has many orders.
-Orders are assigned to many menus.
-A restaurant has many employees.
-Customers can make zero or more bookings.
-One booking can be made by one customer.
-A customer has one address.
-An address can have many customers.
-Customers have more than zero delivery methods.
-Delivery method can be chosen by only one customer.

12. ERD


has
CHEF
PK, FK EMP_ID
HOURS
CHEF_TYPE
ADMINISTRATION
PK, FK EMP_ID
ADMIN_TYPE
"C"
"A"
d EMP_TYPE
EMPLOYEE
PK EMP_ID
FK PRODUCT_ID
FK2 ORDER_ID
EMP_USERNAME
EMP_PASSWORD
EMP_TYPE
SALARY
SHIFT
sets
MENU
PK PRODUCT_ID
FK1 BOOKING_ID
FK2 ORDER_ID
CATEGORY
has
BOOKING
PK BOOKING_ID
FK1 CUSTOMER_ID
BOOKING_DATE
NUM_GUESTS
EVENT_TYPE
TOTAL_PRICE
makes and finalises
d CATEGORY
"D"
"F"
chooses
"T"
DRINKS
PK FK1 PRODUCT_ID
DRINK_NAME
DRINK_PRICE
DRINK_QUANTITY
FOOD
PK FK1 PRODUCT_ID
FOOD_NAME
FOOD_PRICE
FOOD_QUANTITY
DESSERT
PK FK1 PRODUCT_ID
DESSERT_NAME
DESSERT_PRICE
DESSERT_QUANTITY
makes
ORDERS
PK ORDER_ID
FK1 CUSTOMER_ID
ORDER_DATE
ORDER_STATUS
TOTAL_PRICE
placed
CUSTOMER
PK CUSTOMER_ID
CUSTOMER_FNAME
CUSTOMER_LNAME
USERNAME
PASSWORD
PHONE_NUM
EMAIL
ADDRESS
chooses
has
PAYMENT
PK PAYMENT_ID
FK1 ORDER_ID
FK2 DELIVERY_ID
PAYMENT_TYPE
TOTAL_AMOUNT
makes
DELIVERY
PK DELIVER_ID
FK CUSTOMER_ID
DELEVERY_TYPE
d delivery_type
"C"
"D"
COLLECTION
PK, FK DELIVER_ID
COLLECT_DATE
DELIVER
PK, FK DELIVER_ID
DELIVERY_DATE
DELIVERY_AMOUNT
d payment_type
"O"
"C"
"S"
ONLINES
PK, FK PAYMENT_ID
BANK_NAME
BRANCH
CARD_NUMBER
CVV
EXP_DATE
CASH
PK, FK PAYMENT_ID
CASH_OPTION
CARD
PK, FK PAYMENT_ID
CARD_OPTION

13. Notes on the ERD

Composite Keys

- EMP_ID
- PAYMENT_ID
- DELIVERY_ID
- PRODUCT_ID

Composite Attributes

- STREET_NUM
- STREET_NAME
- CITY
- POSTAL_CODE
- PROVINCE

Weak Entities

- CHEF
- ADMIN
- DRINKS
- FOOD
- DESSERT
- ONLINE
- CASH
- CARD
- COLLECTION
- DELIVERY

Super Entities

- EMPLOYEE
- MENU
- PAYMENT
- DELIVERY

Sub-type Entities

- CHEF
- ADMIN
- DRINKS
- FOOD
- DESSERT
- ONLINE
- CASH
- CARD

- COLLECTION
- DELIVER

Recursive Relationship
- EMPLOYEE entity [Relationship = manages = 0 or many to one and only one]

Strong Relationships
- EMPLOYEE and its sub-entities
- PAYMENT and its sub-entities
- DELIVERY and its sub-entities
- MENU and its sub-entities

Weak Relationships
- All the other relationships

14. Mapping Conceptual model into the Logical Model

1. We started mapping the super entities and the optional entities.
2. We mapped the weak entities.
3. Then we directed their relationships using keys to indirect that for instance, using primary keys and foreign keys.

15. Logical Data Model (Super entities)

CUSTOMER (CUSTOMER_ID (PK), CUSTOMER _FNAME, CUSTOMER_LNAME, USERNAME, PASSWORD, PHONE_NUM, EMAIL, ADDRESS)
ORDER (ORDER_ID (PK), *CUSTOMER_ID (FK)*, ORDER_DATE, ORDER_STATUS, TOTAL_PRICE)
EMPLOYEE (EMP_ID (PK), *PRODUCT_ID (FK), ORDER_ID (FK),* EMP_USERNAME, EMP_PASSWORD, EMP_TYPE, SALARY, SHIFT)
BOOKING (BOOKING_ID (PK), *CUSTOMER_ID (FK)*, BOOKING_DATE, NUM_GUESTS, EVENT_TYPE, TOTAL_PRICE)
PAYMENT (PAYMENT_ID (PK), *ORDER_ID (FK), DELIVER_ID (FK)*, PAYMENT_TYPE, TOTAL_AMOUNT)
DELIVERY (DELIVER_ID (PK), *CUSTOMER_ID (FK),* DELIVERY_TYPE)
MENU (PRODUCT_ID (PK), *BOOKING_ID (FK), ORDER_ID (FK),* CATEGORY)

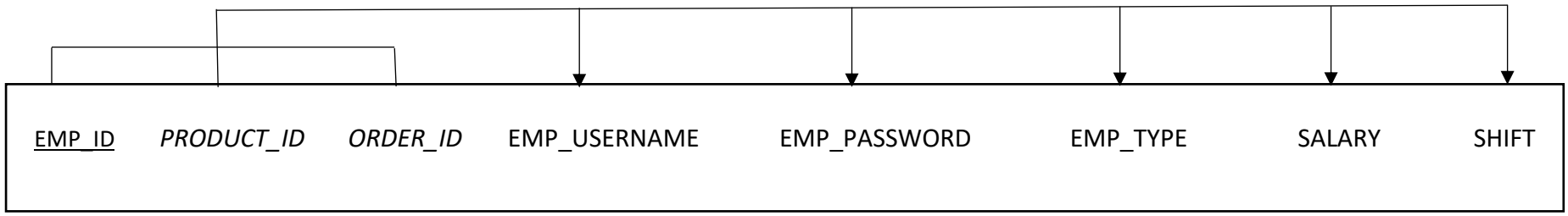
Sub-entities

COLLECTION (*DELIVER_ID (PK) (FK)*, COLLECT_DATE)
DELIVER (*DELIVER_ID (PK) (FK)*, DELIVERY_DATE, DELIVERY_AMOUNT)
ONLINE (*PAYMENT_ID (PK) (FK)*, BANK_NAME, BRANCH, CARD_NUMBER, CVV, EXP_DATE)
CASH (*PAYMENT_ID (PK) (FK)*, CASH_OPTION)
CARD (*PAYMENT_ID (PK) (FK)*, CARD_OPTION)
DRINKS (*PRODUCT_ID (PK) (FK),* DRINK_NAME, DRINK_PRICE, DRINK_QUANTITY)
FOOD (*PRODUCT_ID (PK) (FK)*, FOOD_NAME, FOOD_PRICE, FOOD_QUANTITY)

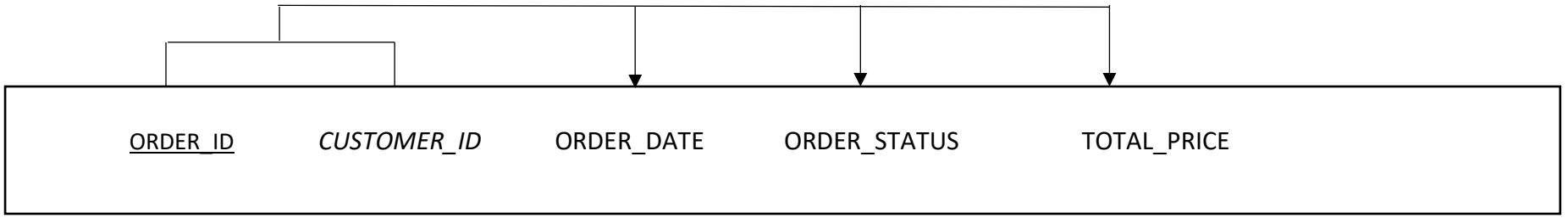
DESSERT (*PRODUCT_ID (PK) (FK)*, DESSERT_NAME, DESSERT_PRICE, DESSERT_QUANTITY)
CHEF (*EMP_ID (PK) (FK),* HOURS, CHEF_TYPE)
ADMIN (*EMP_ID (PK) (FK)*, ADMIN_TYPE)

16. Normalisation

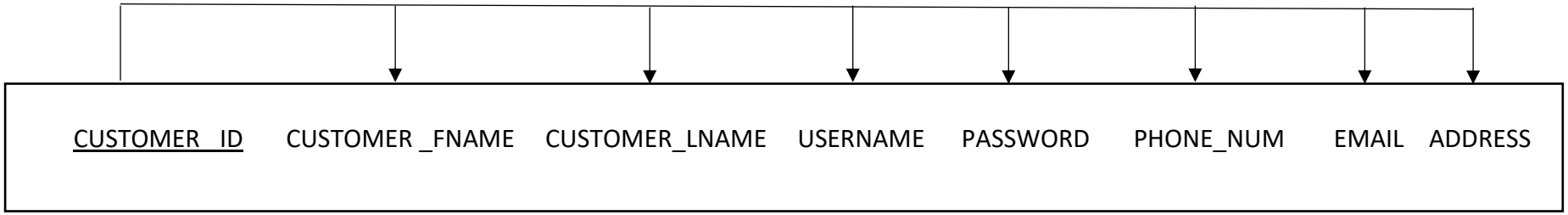
EMPLOYEE



ORDERS



CUSTOMER



17. Dropping tables

We used the following SQL statement to drop the tables that existed before in the database. We also dropped the tables that we were going to create to make sure that we do not create duplicate tables. Meaning that we deleted all tables including the data that was possibly in the tables.

```
-------------------------------------------------------DROP-------------------------------------------------------
drop table countries cascade constraints;
drop table departments cascade constraints;
drop table employees cascade constraints;
drop table jobs cascade constraints;
drop table regions cascade constraints;
drop table locations cascade constraints;
drop table customer cascade constraints;
drop table DELIVERY cascade constraints;
drop table ADMINISTRATION cascade constraints;
drop table BOOKING cascade constraints;
drop table CARD cascade constraints;
drop table CASH cascade constraints;
drop table CHEF cascade constraints;
drop table COLLECTION cascade constraints;
drop table CUSTOMER cascade constraints;
drop table DELIVER cascade constraints;
drop table DELIVERY cascade constraints;
drop table DESSERT cascade constraints;
drop table DRINKS cascade constraints;
drop table EMPLOYEE cascade constraints;
drop table FOOD cascade constraints;
drop table MENU cascade constraints;
drop table ONLINES cascade constraints;
drop table ORDERS cascade constraints;
drop table PAYMENT cascade constraints;
```

18. Dropping sequences

The following statements drop or rather delete any primary keys that might have created previously. This is to avoid duplicating primary keys.

```
-----------------------------------------DROP SEQUENCE-----------------------------------------
DROP SEQUENCE BOOKING_ID_VALUE;
DROP SEQUENCE CUSTOMER_ID_VALUE;
DROP SEQUENCE ORDER_ID_VALUE;
DROP SEQUENCE PAYMENT_ID_VALUE;
DROP SEQUENCE DELIVER_ID_VALUE;
DROP SEQUENCE PRODUCT_ID_VALUE;
DROP SEQUENCE EMP_ID_VALUE;
```

19. Create tables

We created the tables using SQL. These tables are the ones we are going to use to manipulate data. We used the appropriate data types and included all the required Primary keys and Foreign keys. We used the entities on our ERD to create these tables. The following tables are created. CUSTOMER,

ORDERS and BOOKING.

```
----------------------------------------------------------CREATE TABLES----------------------------------------------------------
CREATE TABLE CUSTOMER(
    CUSTOMER_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CUSTOMER_FNAME VARCHAR2(50) NOT NULL,
    CUSTOMER_LNAME VARCHAR2(50) NOT NULL,
    USERNAME VARCHAR2(50) NOT NULL,
    PASSWORD VARCHAR2(100) NOT NULL,
    PHONE_NUM CHAR(10) NOT NULL,
    EMAIL VARCHAR2(50),
    ADDRESS VARCHAR2(100)
);

CREATE TABLE ORDERS(
    ORDER_ID NUMBER(5) PRIMARY KEY NOT NULL,
    CUSTOMER_ID NUMBER(4),
    CONSTRAINT FK_ORDER FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID),
    ORDER_DATE DATE NOT NULL,
    ORDER_STATUS NUMBER(1),
    TOTAL_PRICE NUMBER(19,2) NOT NULL
);

CREATE TABLE BOOKING(
    BOOKING_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CUSTOMER_ID NUMBER(4),
    CONSTRAINT FK_BOOKING FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID),
    BOOKING_DATE DATE NOT NULL,
    NUM_GUESTS NUMBER (5) NOT NULL,
    EVENT_TYPE VARCHAR2(50) NOT NULL,
    TOTAL_PRICE NUMBER(19,2) NOT NULL
);

--
--CREATE TABLE MENU(
```

With the following tables are created: MENU, with its sub-entities, namely, DRINKS, FOOD, and DESSERT.

```
--
CREATE TABLE MENU(
    PRODUCT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    BOOKING_ID NUMBER(4) NULL,
    CONSTRAINT FK_MENU1 FOREIGN KEY (BOOKING_ID) REFERENCES BOOKING (BOOKING_ID),
    ORDER_ID NUMBER(4) NULL,
    CONSTRAINT FK_MENU2 FOREIGN KEY (ORDER_ID) REFERENCES ORDERS (ORDER_ID),
    CATEGORY CHAR(1)
);
--
CREATE TABLE DRINKS(
    PRODUCT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_DRINKS FOREIGN KEY (PRODUCT_ID) REFERENCES MENU (PRODUCT_ID),
    DRINK_NAME VARCHAR2(20),
    DRINK_PRICE NUMBER(19,2),
    DRINK_QUANTITY NUMBER(3,2)
);

CREATE TABLE FOOD(
    PRODUCT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_FOOD FOREIGN KEY (PRODUCT_ID) REFERENCES MENU (PRODUCT_ID),
    FOOD_NAME VARCHAR2(20),
    FOOD_PRICE NUMBER(19,2),
    FOOD_QUANTITY NUMBER(3,2)
);

CREATE TABLE DESSERT(
    PRODUCT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_DESSERET FOREIGN KEY (PRODUCT_ID) REFERENCES MENU (PRODUCT_ID),
    DESSERT_NAME VARCHAR2(20),
    DESSERT_PRICE NUMBER(19,2),
    DESSERT_QUANTITY NUMBER(3,2)
);
```

The following tables are created: PAYMENT with its sub-entities, namely, ONLINES, CASH, and CARD.

```
CREATE TABLE PAYMENT(
    PAYMENT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    ORDER_ID NUMBER(5) NOT NULL,
    CONSTRAINT FK_PAYMENT FOREIGN KEY (ORDER_ID) REFERENCES ORDERS (ORDER_ID),
    PAYMENT_TYPE CHAR(1),
    TOTAL_AMOUNT NUMBER(19,2)
);

CREATE TABLE ONLINES(
    PAYMENT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_ONLINE FOREIGN KEY (PAYMENT_ID) REFERENCES PAYMENT (PAYMENT_ID),
    BANK_NAME VARCHAR2(50),
    BRANCH VARCHAR2(50),
    CARD_NUMBER NUMBER(16),
    CVV NUMBER(3),
    EXP_DATE DATE
);

CREATE TABLE CASH(
    PAYMENT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_CASH FOREIGN KEY (PAYMENT_ID) REFERENCES PAYMENT (PAYMENT_ID),
    CASH_OPTION NUMBER(1)
);

CREATE TABLE CARD(
    PAYMENT_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_CARD FOREIGN KEY (PAYMENT_ID) REFERENCES PAYMENT (PAYMENT_ID),
    CARD_OPTION NUMBER(1)
);
--
CREATE TABLE DELIVERY(
    DELIVER_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CUSTOMER_ID NUMBER(4) NOT NULL,
    CONSTRAINT FK_DELIVERY FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID),
    DELIVERY_TYPE CHAR(1)
);
```

The following tables are created: DELIVERY with its sub-entities, which include COLLECTION and DELIVER. Table EMPLOYEE was also created.

```
CREATE TABLE DELIVERY(
    DELIVER_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CUSTOMER_ID NUMBER(4) NOT NULL,
    CONSTRAINT FK_DELIVERY FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID),
    DELIVERY_TYPE CHAR(1)
);

CREATE TABLE COLLECTION(
    DELIVER_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_COLLECTION FOREIGN KEY (DELIVER_ID) REFERENCES DELIVERY (DELIVER_ID),
    COLLECT_DATE DATE
);

CREATE TABLE DELIVER(
    DELIVER_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_DELIVER FOREIGN KEY (DELIVER_ID) REFERENCES DELIVERY (DELIVER_ID),
    DELIVERY_DATE DATE,
    DELIVERY_AMOUNT NUMBER(19,2)
);

CREATE TABLE EMPLOYEE(
    EMP_ID NUMBER(4) PRIMARY KEY NOT NULL,
    PRODUCT_ID NUMBER(4),
    CONSTRAINT FK_EMPLOYEE FOREIGN KEY (PRODUCT_ID) REFERENCES MENU (PRODUCT_ID),
    ORDER_ID NUMBER(4),
    CONSTRAINT FK_EMPLOYEE2 FOREIGN KEY (ORDER_ID) REFERENCES ORDERS (ORDER_ID),
    EMP_USERNAME VARCHAR2(50),
    EMP_PASSWORD VARCHAR2(100),
    EMP_TYPE CHAR(1),
    SALARY NUMBER(19,2),
    SHIFT DATE
);
--
--
```

CHEF and ADMINISTRATION tables were created, which are the sub-entities of EMPLOYEE.

```
);
--

CREATE TABLE CHEF(
    EMP_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_CHEF FOREIGN KEY (EMP_ID) REFERENCES EMPLOYEE (EMP_ID),
    HOURS NUMBER(2),
    CHEF_TYPE NUMBER(1)
);

CREATE TABLE ADMINISTRATION(
    EMP_ID NUMBER(4) PRIMARY KEY NOT NULL,
    CONSTRAINT FK_ADMIN FOREIGN KEY (EMP_ID) REFERENCES EMPLOYEE (EMP_ID),
    ADMIN_TYPE NUMBER(1)
);
```

20. Create sequences

We created sequences to generate primary keys automatically. These methods will make it easier for the client as they will not have to create their own primary keys, as these sequences will do the work.

```
----------------------------------------------------------SEQUENCENCES----------------------------------------------------------
CREATE SEQUENCE CUSTOMER_ID_VALUE
    start with 1
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 1000
    CYCLE;
--
--
CREATE SEQUENCE ORDER_ID_VALUE
    start with 50
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 10000
    CYCLE;

CREATE SEQUENCE BOOKING_ID_VALUE
    start with 10
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 1000
    CYCLE;

CREATE SEQUENCE PRODUCT_ID_VALUE
    start with 100
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 1000000
    CYCLE;

CREATE SEQUENCE PAYMENT_ID_VALUE
    start with 75
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 10000
    CYCLE;
```

```
CREATE SEQUENCE DELIVER_ID_VALUE
    start with 150
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 100000
    CYCLE;

CREATE SEQUENCE EMP_ID_VALUE
    start with 1
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 1000
    CYCLE;
```

21. Create views

We created views for the tables that will be used more often by the client. The client will be able to execute the required actions faster using views. This will be an added advantage to the restaurant

business because they client mentioned that they work many customers in daily basis.

```
----------------------------------------------DROPPED VIEW-----------------------------------------------------------
DROP VIEW CUSTOMER_COUNTINAMES;

-----------------------------------------------------------VIEWS-----------------------------------------------------

CREATE VIEW CUSTOMER_DETAILS AS
SELECT CUSTOMER_ID,CUSTOMER_FNAME ,CUSTOMER_LNAME,USERNAME ,PHONE_NUM ,EMAIL ,ADDRESS  FROM CUSTOMER WHERE customer_fname = 'NTHABISENG' AND customer_lname = 'TLOU';

CREATE VIEW CUSTOMER_COUNTINAMES AS
SELECT COUNT(customer_fname) AS "NUMBER_OF_CUSTOMER" FROM CUSTOMER WHERE customer_fname = 'NEO';


CREATE VIEW ACTIVE_ORDERS AS
SELECT c.CUSTOMER_ID AS "CUSTOMER ID", c.CUSTOMER_FNAME AS "CUSTOMER NAME", c.CUSTOMER_LNAME AS "CUSTOMER SURNAME", o.ORDER_STATUS AS "ORDER STATUS"
FROM CUSTOMER c, ORDERS o WHERE ORDER_STATUS = 1 AND c.CUSTOMER_ID = o.CUSTOMER_ID;

CREATE VIEW NUMBER_GUESTS AS
SELECT c.CUSTOMER_FNAME AS "CUSTOMER NAME", c.CUSTOMER_LNAME AS "CUSTOMER SURNAME", c.ADDRESS AS "CUSTOMER ADDRESS", b. NUM_GUESTS AS "NUMBER OF GUESTS", b.EVENT_TYPE AS "EVENT TYPE"
FROM CUSTOMER c, BOOKING b
WHERE NUM_GUESTS >= 10 AND c.CUSTOMER_ID = b.CUSTOMER_ID;
```

22. Create indexes

We created indexes to increase the speed in which the client will retrieve information from the database using queries. This will enable the client to attend her customers' details without any hassle or delay. For the increase in production, the client needs to access the customer's details to process the customer's booking appointment or order.

```
-----------------------------------------------------INDEXES-----------------------------------------------------------
CREATE INDEX idx_CustomerNames
ON CUSTOMER (CUSTOMER_FNAME, CUSTOMER_LNAME, ADDRESS);

CREATE INDEX idx_EmpDetails
ON EMPLOYEE (EMP_USERNAME, EMP_TYPE, SHIFT);

CREATE INDEX idx_DRINKS
ON DRINKS (DRINK_NAME, DRINK_PRICE);

CREATE INDEX idx_FOOD
ON FOOD (FOOD_NAME, FOOD_PRICE);

CREATE INDEX idx_DESSERT
ON DESSERT (DESSERT_NAME, DESSERT_PRICE);
```

22. Check constraints

For extra functionality, we checked constraints for all the tables that has CHAR as a data type. Tables such as MENU which has CATEGORY for DRINKS as 'D', FOOD as 'F', and DESSERT as 'T'. EMPLOYEE which has EMP_TYPE for CHEF as 'C' and ADMINISTRATION as 'A'. DELIVERY table which has DELIVERY_TYPE for DELIVER as 'D' and COLLECTION as 'C'. And lastly, we have PAYMENT table which has PAYMENT_TYPE for ONLINES as 'O', CASH as 'C' and CARD as 'S'.

```
-----------------------------------------------------CHECK CONSTRAINTS-------------------------------------------------

ALTER TABLE MENU ADD CONSTRAINT "CATEGORY_CHECK_F_OR_D_T" CHECK (CATEGORY IN('F','D','T')) ENABLE;
ALTER TABLE EMPLOYEE ADD CONSTRAINT "EMPLOY_CHECK_C_OR_A" CHECK (EMP_TYPE IN('C','A')) ENABLE;
ALTER TABLE DELIVERY ADD CONSTRAINT "DELIVERY_CHECK_D_OR_C" CHECK (DELIVERY_TYPE IN('D','C')) ENABLE;
ALTER TABLE PAYMENT ADD CONSTRAINT "PAYMENT_CHECK_O_OR_C_S" CHECK (PAYMENT_TYPE IN('O','C','S')) ENABLE;
```

23. Insert data in tables

We inserted data for CUSTOMER table

```
-----------------------------------------------------------ADD DATA-----------------------------------------------------------
-----------------------------------------------------ADD INTO CUSTOMER--------------------------------------------------------
INSERT INTO EMPLOYEE (EMP_ID,CUSTOMER_ID, EMP_USERNAME, EMP_PASSWORD, EMP_TYPE, SALARY, SHIFT)
VALUES (ORDER_ID_VALUE.nextval, CUSTOMER_ID, EMP_ID, TO_DATE('24/05/2022 12:00:00', 'dd/mm/yyyy hh:mi:ss'),1,1500.00);

INSERT INTO EMPLOYEE (EMP_ID,product_id, EMP_USERNAME,EMP_PASSWORD, EMP_TYPE, SALARY, SHIFT)
VALUES (EMP_ID_VALUE.nextval,PRODUCT_ID_VALUE.nextval,"John","#######",1,10000.56, TO_DATE('24/05/2022 12:00:00', 'dd/mm/yyyy hh:mi:ss'));

INSERT INTO CUSTOMER (CUSTOMER_ID, CUSTOMER_FNAME, CUSTOMER_LNAME,USERNAME,PASSWORD,PHONE_NUM,EMAIL,ADDRESS)
VALUES (CUSTOMER_ID_VALUE.nextval,'Thando','Wenn','THANN','#########','0785401615','THANDO@GMAIL.COM','21st Street Pretoria');

INSERT INTO CUSTOMER (CUSTOMER_ID, CUSTOMER_FNAME, CUSTOMER_LNAME,USERNAME,PASSWORD,PHONE_NUM,EMAIL,ADDRESS)
VALUES (CUSTOMER_ID_VALUE.nextval,'Lindi','Nkosi','LN','#########','0624875852','LUNDI@GMAIL.COM','1st Street Mangau');

INSERT INTO CUSTOMER (CUSTOMER_ID, CUSTOMER_FNAME, CUSTOMER_LNAME,USERNAME,PASSWORD,PHONE_NUM,EMAIL,ADDRESS)
VALUES (CUSTOMER_ID_VALUE.nextval,'Neo','Babedi','NeoP','#########','0785401644','Neo@GMAIL.COM','21st Street Taung');

INSERT INTO CUSTOMER (CUSTOMER_ID, CUSTOMER_FNAME, CUSTOMER_LNAME,USERNAME,PASSWORD,PHONE_NUM,EMAIL,ADDRESS)
VALUES (CUSTOMER_ID_VALUE.nextval,'Lebo','Paai','LPaai','#########','0844701615','Paai@GMAIL.COM','nwu Street 1999 Pretoria');

INSERT INTO CUSTOMER (CUSTOMER_ID, CUSTOMER_FNAME, CUSTOMER_LNAME,USERNAME,PASSWORD,PHONE_NUM,EMAIL,ADDRESS)
VALUES (CUSTOMER_ID_VALUE.nextval,'JOHN','MOORE','JMOORE','##########','0844987615','MOORE@GMAIL.COM','KZN Street 1999 Pretoria');
```

We inserted data for Booking table, which consisted of BOOKING_ID, CUSTOMER_ID, BOOKING_DATE, NUM_GUESTS, EVENT_TYPE, and TOTAL_PRICE.

```
-------------------------------------------------INSERT BOOKING-------------------------------------------------------------
INSERT INTO BOOKING(BOOKING_ID, CUSTOMER_ID, BOOKING_DATE, NUM_GUESTS, EVENT_TYPE, TOTAL_PRICE)
VALUES (BOOKING_ID_VALUE.nextval,1,TO_DATE('24-05-2022 12:00:00', 'dd/mm/yyyy hh:mi:ss' ), 50, 'WEDDING', 15000.46);

INSERT INTO BOOKING(BOOKING_ID, CUSTOMER_ID, BOOKING_DATE, NUM_GUESTS, EVENT_TYPE, TOTAL_PRICE)
VALUES (BOOKING_ID_VALUE.nextval,2, TO_DATE('30-07-2022 14:45:00', 'dd/mm/yyyy hh24:mi:ss' ), 100, 'FUNERAL', 150);

INSERT INTO BOOKING(BOOKING_ID, CUSTOMER_ID, BOOKING_DATE, NUM_GUESTS, EVENT_TYPE, TOTAL_PRICE)
VALUES (BOOKING_ID_VALUE.nextval,3, TO_DATE('30-08-2022 18:45:00', 'dd/mm/yyyy hh24:mi:ss' ), 5, 'BABY SHOWER', 5000);

INSERT INTO BOOKING(BOOKING_ID, CUSTOMER_ID, BOOKING_DATE, NUM_GUESTS, EVENT_TYPE, TOTAL_PRICE)
VALUES (BOOKING_ID_VALUE.nextval,1, TO_DATE('15-08-2022 18:45:00', 'dd/mm/yyyy hh24:mi:ss' ), 2, 'DINNER FOR 2', 500);
```

We inserted the following data into ORDERS table

```
-------------------------------------------------INSERT ORDER-------------------------------------------------------------
INSERT INTO ORDERS(ORDER_ID,CUSTOMER_ID,ORDER_DATE,ORDER_STATUS,TOTAL_PRICE)
VALUES (ORDER_ID_VALUE.nextval ,4,TO_DATE('24-08-2022 18:00:00', 'dd/mm/yyyy hh24:mi:ss'),1,35.00);

INSERT INTO ORDERS(ORDER_ID,CUSTOMER_ID,ORDER_DATE,ORDER_STATUS,TOTAL_PRICE)
VALUES (ORDER_ID_VALUE.nextval ,5,TO_DATE('15-06-2022 12:00:00', 'dd/mm/yyyy hh24:mi:ss'),0,3500.10);
```

We inserted relevant data into DELIVERY table.

```
-----------------------------------------------INSERT DELIVERY-----------------------------------------------------------

INSERT INTO DELIVERY (deliver_id,customer_id,delivery_type)
VALUES (DELIVER_ID_VALUE.nextval,4,'C');

INSERT INTO DELIVERY (deliver_id,customer_id,delivery_type)
VALUES (DELIVER_ID_VALUE.nextval,5,'D');


INSERT INTO deliver(deliver_id,DELIVERY_DATE,DELIVERY_AMOUNT)
VALUES (151,TO_DATE('24-08-2022 18:00:00', 'dd/mm/yyyy hh24:mi:ss'),49.99);

INSERT INTO COLLECTION(deliver_id,collect_date)
VALUES (150,TO_DATE('24-08-2022 13:30:00', 'dd/mm/yyyy hh24:mi:ss'));
```

We inserted data into EMPLOYEE table

```
----------------------------------------------INSERT EMPLOYEE-----------------------------------------------------------

INSERT INTO EMPLOYEE (EMP_ID,order_id,PRODUCT_ID,EMP_USERNAME,EMP_PASSWORD,EMP_TYPE,SALARY,SHIFT)
VALUES (EMP_ID_VALUE.nextval,51,100,'KARABO','######','A',14999.999,TO_DATE('24-08-2022 13:30:00', 'dd/mm/yyyy hh24:mi:ss'));

INSERT INTO administration (EMP_ID,admin_type)
VALUES (1,1);

INSERT INTO EMPLOYEE (EMP_ID,order_id,PRODUCT_ID,EMP_USERNAME,EMP_PASSWORD,EMP_TYPE,SALARY,SHIFT)
VALUES (EMP_ID_VALUE.nextval,52,100,'KGOMOTSO','####','C',15999.999,TO_DATE('01-08-2022 07:30:00', 'dd/mm/yyyy hh24:mi:ss'));

INSERT INTO CHEF (EMP_ID,chef_type)
VALUES (2,1);
```

We inserted data into MENU table.

```
-------------------------------------------------INSERT MENU-----------------------------------------------------------
INSERT INTO MENU(PRODUCT_ID,BOOKING_ID,ORDER_ID, CATEGORY)
VALUES (PRODUCT_ID_VALUE.nextval ,NULL, 52, 'F');


INSERT INTO MENU(PRODUCT_ID,BOOKING_ID,ORDER_ID,CATEGORY)
VALUES (PRODUCT_ID_VALUE.nextval ,NULL,51, 'F');

INSERT INTO MENU(PRODUCT_ID,BOOKING_ID,ORDER_ID,CATEGORY)
VALUES (PRODUCT_ID_VALUE.nextval ,10,NULL, 'D');

INSERT INTO MENU(PRODUCT_ID,BOOKING_ID,ORDER_ID,CATEGORY)
VALUES (PRODUCT_ID_VALUE.nextval ,11,NULL, 'F');
```

We inserted data into DRINKS and FOOD tables.

```
-------------------------------------INSERT DATA DRINKS-----------------------------------------------------------
INSERT INTO DRINKS (PRODUCT_ID, DRINK_NAME, DRINK_PRICE, DRINK_QUANTITY)
VALUES (101, 'GINGER BEER', 25.00, 2);
-------------------------------------INSERT FOOD-----------------------------------------------------------

INSERT INTO FOOD (PRODUCT_ID, FOOD_NAME, FOOD_PRICE, FOOD_QUANTITY)
VALUES (100, 'MOGODU', 300.00, 8);

INSERT INTO FOOD (PRODUCT_ID, FOOD_NAME, FOOD_PRICE, FOOD_QUANTITY)
VALUES (102, 'MOGODU', 550.00, 9);

INSERT INTO DESSERT (PRODUCT_ID, DESSERT_NAME, FOOD_PRICE, FOOD_QUANTITY)
VALUES (102, 'MOGODU', 550.00, 9);
```

Queries

We used DELETE statements to delete data in the tables, this method will make sure that all the data is deleted from the tables. We also used SELECT to show the data that has been inserted into the tables. The asterisk * means that we select all the columns of the tables, instead of specifying which

column to show.

```
DELETE FROM CUSTOMER;
DELETE FROM BOOKING;
DELETE FROM ORDERS;
SELECT * FROM CUSTOMER;
SELECT * FROM BOOKING ORDER BY BOOKING_ID;
SELECT * FROM MENU;

SELECT * FROM DRINKS;
SELECT * FROM DELIVERY;
SELECT * FROM DELIVER;
SELECT * FROM collection;
SELECT * FROM EMPLOYEE;
SELECT * FROM orders;

DESC EMPLOYEE;
DESC ADMINISTRATION;
```

Like, AND, OR

With the LIKE statement, we used it to retrieve data for customers whose first name start with L and we also we used COUNT to count the customers.

With the AND statement, we used it to determine Bookings that has a number of guests that is greater or equals to 10 AND event type is FUNERAL. Both conditions will have to be true for the statement to be applied.

With the OR statement, we used it to determine Bookings that has a number of guests that is greater or equals to 10 OR event type is FUNERAL. Either of conditions will have to be true for the statement to be applied.

JOIN

First join statement - We used JOIN to join two tables which are CUSTOMER table and BOOKING table using CUSTOMER_ID on CUSTOMER table as PK and CUSTOMER_ID on BOOKING table as FK to return the number of guests which are equals to 100.

Second join statement - We joined CUSTOMER and BOOKING table using CUSTOMER_ID on CUSTOMER table as PK and CUSTOMER_ID on BOOKING table as FK with MENU table using BOOKING_ID on BOOKING table as PK and on MENU as FK with CUSTOMER_ID on CUSTOMER table as PK and CUSTOMER_ID on BOOKING table as FK to return the number of guests which are equals to 100.

```
----------------------------------------------QUERIES-----------------------------------------------------------------------


-----------------------------------------STATEMENT1------LIKE--AND---OR-----------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
SELECT COUNT(customer_fname) AS "NUMBER_OF_CUSTOMER" FROM CUSTOMER WHERE customer_fname LIKE 'L%';
SELECT customer_fname AS "CUSTOMER" FROM CUSTOMER WHERE customer_fname LIKE 'L%';
SELECT * FROM BOOKING WHERE num_guests >=10 AND event_type = 'FUNERAL';
SELECT * FROM BOOKING WHERE num_guests >=10 OR event_type = 'FUNERAL';

-----------------------------------------------SORTING-----------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
                                ---Retrieve
SELECT * FROM CUSTOMER ORDER BY customer_lname ASC;
SELECT * FROM CUSTOMER ORDER BY customer_fname DESC;
SELECT * FROM CUSTOMER ORDER BY phone_num ASC;
SELECT CATEGORY FROM MENU ORDER BY 1;

-----------------------------------------------JOIN--------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------

SELECT c.CUSTOMER_FNAME, c.CUSTOMER_LNAME, c.ADDRESS, b. NUM_GUESTS, b.EVENT_TYPE, b.booking_date
FROM CUSTOMER c JOIN BOOKING b ON c.CUSTOMER_ID = b.CUSTOMER_ID
WHERE NUM_GUESTS = 100;


SELECT c.CUSTOMER_FNAME, c.CUSTOMER_LNAME, c.ADDRESS, b. NUM_GUESTS, b.EVENT_TYPE, b.booking_date, m.CATEGORY
FROM CUSTOMER c JOIN BOOKING b
ON c.CUSTOMER_ID = b.CUSTOMER_ID JOIN
MENU m
ON b.booking_id = m.booking_id AND c.customer_id = b.customer_id
WHERE NUM_GUESTS = 100;
```

Limitation

First limitation statement – To check customers with active orders we used CUSTOMER and ORDERS tables where 1 means active linking the CUSTOMER_ID on CUSTOMER table as PK and on ORDERS table as FK.

Second limitation statement - We joined CUSTOMER and BOOKING table using CUSTOMER_ID on CUSTOMER table as PK and CUSTOMER_ID on BOOKING table as FK with MENU table using BOOKING_ID on BOOKING table as PK and on MENU as FK with CUSTOMER_ID on CUSTOMER table as PK and CUSTOMER_ID on BOOKING table as FK to return the number of guests which are grater than 5.

```
---------------------------------------limitation-----------------------------------------------
-------------------------------------------------------------------------------------------------


----CUSTOMERS WITH ACTIVE ORDERS--------
SELECT c.CUSTOMER_ID, c.CUSTOMER_FNAME, c.CUSTOMER_LNAME, o.ORDER_STATUS
FROM CUSTOMER c, ORDERS o WHERE ORDER_STATUS = 1 AND c.CUSTOMER_ID = o.CUSTOMER_ID;


SELECT c.CUSTOMER_FNAME, c.CUSTOMER_LNAME, c.ADDRESS, b. NUM_GUESTS, b.EVENT_TYPE, b.booking_date, m.CATEGORY
FROM CUSTOMER c JOIN BOOKING b
ON c.CUSTOMER_ID = b.CUSTOMER_ID JOIN
MENU m
ON b.booking_id = m.booking_id AND c.customer_id = b.customer_id
WHERE NUM_GUESTS > 5 ;




                        ------------SHOW 1 ROWS ONLY--------------------------
select * from (select * from CUSTOMER order by PHONE_NUM) where rownum = 1;
                         ------------SHOW 3 ROWS ONLY--------------------------
select * from (select * from CUSTOMER order by PHONE_NUM) where rownum <= 3;
```

Having and Group by

First statement – This statement will retrieve category and count renamed “Count” from Menu table and group values to be returned by Category.

Second statement – The inner statement will select category attribute and use it to be displayed and to count and will group them by category. This statement will read from menu table. The outer statement will use the output of statement to retrieve all attributes from inner statement and apply the condition also where count is less than or equals to 3.

Third statement – This statement will retrieve category and count renamed “Count” from Menu table and group values that are having a greater than 2 value by category.

Fourth statement – This statement functions the same as the second statement. The only difference is the condition which is equals to 3 in this statement. It will retrieve category and count that is equals to 3.

Aggregate functions

First statement – This statement will select food price and food quantity from the Food table and multiply them together after they will be summed up. And the total sum will be returned.

Second statement – This statement will retrieve 4 attributes from food table, and it will use the inner statement as the same as the first statement and add it as an additional column of summed up values. It will return same values.

Third statement – This statement will select Drink price and Drink quantity from the Drink table and multiply them together after they will be summed up. And the total sum will be returned.

Fourth statement – This statement will retrieve 4 attributes from Drink table, and it will use the inner statement to find the minimum value after getting the product of Drink price and Drink quantity and display it as a new attribute named as “TOTAL_MIN”.

Firth statement – This statement will retrieve the product value of food price and food quantity from Food table and name it as “Food total price”.

Sixth statement – This statement will retrieve the maximum product value of food price and food quantity from Food table and name it as “FOOD MAX PRICE”.

SEVENTH statement – This statement will retrieve 4 attributes from Food table, and it will use the inner statement to find the maximum value after getting the product of Food price and Food quantity and display it as a new attribute named as “MAX VAU”.

```
----------------------------------------------HAVING AND GROUB BY----------------------------------------------
SELECT CATEGORY, COUNT(CATEGORY)AS COUNT FROM MENU GROUP BY CATEGORY;

SELECT * FROM (SELECT CATEGORY, COUNT(CATEGORY)AS COUNT FROM MENU GROUP BY CATEGORY) WHERE COUNT <=3;

SELECT CATEGORY, COUNT(CATEGORY)AS COUNT FROM MENU GROUP BY CATEGORY HAVING COUNT(CATEGORY) >2;

SELECT * FROM (SELECT CATEGORY, COUNT(CATEGORY)AS COUNT FROM MENU GROUP BY CATEGORY) WHERE COUNT =3;




----------------------------------------------AGGREGATE FUNCTION----------------------------------------------
--------------------SUM---------------------------------
SELECT SUM(FOOD_PRICE * FOOD_QUANTITY) FROM FOOD;
SELECT PRODUCT_ID,FOOD_NAME,FOOD_PRICE,FOOD_QUANTITY,(SELECT SUM(FOOD_PRICE * FOOD_QUANTITY) FROM FOOD) AS "TOTAL_SUM" FROM FOOD;

--------------------MIN---------------------------------
SELECT SUM(DRINK_PRICE * DRINK_QUANTITY)AS "DRINK MIN PRICE" FROM DRINKS;
SELECT PRODUCT_ID,DRINK_NAME,DRINK_PRICE,DRINK_QUANTITY,(SELECT MIN(DRINK_PRICE * DRINK_QUANTITY) FROM DRINKS) AS "TOTAL_MIN" FROM DRINKS;

--------------------MAX---------------------------------
SELECT FOOD_PRICE * FOOD_QUANTITY AS "FOOD TOTAL PRICES" FROM FOOD;
SELECT MAX(FOOD_PRICE * FOOD_QUANTITY)AS "FOOD MAX PRICE" FROM FOOD;
SELECT PRODUCT_ID,FOOD_NAME,FOOD_PRICE,FOOD_QUANTITY,(SELECT MAX(FOOD_PRICE * FOOD_QUANTITY) FROM FOOD) AS "MAX VALU" FROM FOOD;
```

Date functions

Second statement – Will cast the booking as a timestamp and return both the date and the time from the Booking table and name it as time. Reason behind cast and timestamp is because statement on its own will return only the date but using the two will give the date and time.

Third statement – Functions the same as the second statement but the only difference is that the third statement will have to apply the condition which is used to return the values of the row 1 only.

Fourth statement – Functions the same as the second and the third statements but the only difference is that the Fourth statement will have to choose rows greater or equals to 1 and the number of guest greater or equals to 100. It will Retrieve few columns from Booking table.

Character functions, Variables

First statement – We used INTCAP to capitalize the first letter of CUSTOMER_FNAME and CUSTOMER_LNAME on the CUSTOMER table.

Second statement – We used UPPER to capitalize all the letters of CUSTOMER_FNAME and CUSTOMER_LNAME on the CUSTOMER table.

Third statement – We used LOWER to display all the letters of CUSTOMER_FNAME and CUSTOMER_LNAME in lower case on the CUSTOMER table.

Fourth statement – This statement will call two tables and select some few attributes from each that are linked by the Customer ID that is acting as primary key and foreign key. The condition will also be applied to return values. The customer first names will be set to Upper case whilst the customer Last name will only be changes in the first letter because of INTCAP.

Fifth and Sixth statement – These statements will return the connected Customer_Fname and Customer_Lname from Customer table.

```
-----------------------------------------------DATE FUNCTIONS-----------------------------------------------
-------------------------------------------------------------------------------------------------------------
SELECT BOOKING_ID, booking_date, event_type, num_guests, total_price FROM BOOKING WHERE TO_CHAR(booking_date) = '3';
SELECT CAST(booking_date AS TIMESTAMP)"TIME" FROM BOOKING;
SELECT CAST(booking_date AS TIMESTAMP)"TIME" FROM BOOKING WHERE rownum = 1;

SELECT booking_id,customer_id,num_guests,CAST(booking_date AS TIMESTAMP)"TIME" FROM BOOKING WHERE rownum >= 1 and num_guests>=100;




-----------------------------------------------VARIABLES AND CHARACTER FUNCTION -----------------------------------------------------------
------FIRST LETTER AS CAPITAL---------
SELECT INITCAP(customer_fname),INITCAP(customer_lname),address FROM CUSTOMER;
------CAPITAL LETTERS---------
SELECT UPPER(customer_fname),UPPER(customer_lname),address FROM CUSTOMER;
------LOWER LETTERS---------
SELECT LOWER(customer_fname),LOWER(customer_lname),address FROM CUSTOMER;

SELECT c.CUSTOMER_ID, UPPER(c.CUSTOMER_FNAME), INITCAP(c.CUSTOMER_LNAME), o.ORDER_STATUS
FROM CUSTOMER c, ORDERS o WHERE ORDER_STATUS = 1 AND c.CUSTOMER_ID = o.CUSTOMER_ID;

------CONCAT FNAME AND LMANE---------
SELECT customer_id,(CUSTOMER_FNAME||CUSTOMER_LNAME) FROM CUSTOMER;
SELECT customer_id,CONCAT(CUSTOMER_FNAME,CUSTOMER_LNAME) FROM CUSTOMER;
```

Round

First ROUND statement - We used ROUND to round off the TOTAL_PRICE on BOOKING table using the specified columns to 1 decimal place.

Second ROUND statement – We used ROUND to round off the TOTAL_PRICE to 1 decimal place on CUSTOMER table joining it with BOOKING table. WE then used CUSTOMER_ID on CUSTOMER table as PK and BOOKING table as FK where the number of guests is equals to 100.

Third ROUND statement - We used ROUND to round off SALARY on EMPLOYEE table which is joined with ORDERS table with ORDER_ID on ORDERS as PK and on EMPLOYEE as FK.

Fourth ROUND statement – We used ROUND to round off the TOTAL_PRICE to 2 decimal places on CUSTOMER table joining it with BOOKING table. WE then used CUSTOMER_ID on CUSTOMER table as PK and BOOKING table as FK where the number of guests is greater than 100.

Sub-queries

First Query statement – The inner statement will select every attribute from the Customer table and order them by phone numbers. After ordering has happened, the outer statement will select everything from the output of the inner statement and show only the first row.

Second Query statement – The inner statement will compute the average value of the total price from orders table. The outer statement will then select few attributes to be displayed with the output from the orders table also and give it will give the new average column as "AVARAGE_PRICE". The results to be displayed will be Those few attributes joined by the new attribute.

Third Query statement – The inner statement will calculate the average using total price from orders table and keep it for a temporary time. The outer statement will call the inner statement and

select every attribute from orders table and compare the results of inner statement with the total price from the orders table. Then the results will be retrieved ones the results meet the condition.

FOUTH Query statement – The inner statement will retrieve the Customer IDs from the Orders table after retrieving, the outer statement will take the output of the inner statement and use it to retrieve every attribute from Customer table which equals it or if the Customer IDs match each other from both sides.

Fifth Query statement – The inner statement will retrieve Booking IDs from the Booking table where number of Guest is less than 200. After that the outer statement will retrieve every attribute in the Menu table where Booking ID is found in the Output of the Inner statement.

```
-----------------------------------------------ROUND ------------------------------------------------------------------
SELECT booking_id,customer_id,num_guests,event_type,booking_date,ROUND(TOTAL_PRICE,1) FROM BOOKING;


SELECT c.CUSTOMER_FNAME, c.CUSTOMER_LNAME, c.ADDRESS, b. NUM_GUESTS, b.EVENT_TYPE, b.booking_date, Round(b.total_price,1)
FROM CUSTOMER c JOIN BOOKING b ON c.CUSTOMER_ID = b.CUSTOMER_ID
WHERE NUM_GUESTS = 100;

SELECT c.emp_username, c.emp_type, c.shift, ROUND(c.salary), b.order_date, b.order_status, b.customer_id
FROM EMPLOYEE c JOIN ORDERS b ON c.order_id = b.order_id;

SELECT c.CUSTOMER_FNAME, c.CUSTOMER_LNAME, c.ADDRESS, b.order_date, b.order_status, Round(b.total_price,2)
FROM CUSTOMER c JOIN ORDERS b ON c.CUSTOMER_ID = b.CUSTOMER_ID
WHERE total_price > 100;


-----------------------------------------------SUB QUERIES-----------------------------------------
-----------------------------------------------------------------------------------------------------------------
--------SELECT 1ROW AFTER ORDER---------------
select * from (select * from CUSTOMER order by PHONE_NUM) where rownum = 1;

----------------------------SHOW AVARAGE---------------
SELECT ORDER_ID,CUSTOMER_ID,ORDER_DATE,ORDER_STATUS,TOTAL_PRICE, (SELECT AVG(TOTAL_PRICE) FROM ORDERS) AS AVARAGE_PRICE FROM ORDERS;
SELECT * FROM orders WHERE total_price < (select AVG(TOTAL_PRICE) FROM ORDERS);


-----------------------SELECT ALL FROM CUSTOMER TABLE THAT CROSSPOND TO THE CUSTOMER ID IN THE ORDER TABLE----------------------------
SELECT * FROM CUSTOMER WHERE CUSTOMER_ID IN (SELECT CUSTOMER_ID FROM ORDERS);

-----------------------RETRIEVE MENUES THAT CROSPONDE TO THE BOOKING TABLE WHERE NUMBER OF GUESTS IS LESS THAN 200--------------------
SELECT * FROM MENU WHERE booking_id IN (SELECT booking_id FROM BOOKING WHERE NUM_GUESTS < 200);
```