



EAST WEST UNIVERSITY

Project Report

Project Title: Restaurant Management System

Course Title: Advanced Database System

Course Code: CSE464

Section: 02

Submitted To

Maliha Nawshin Rahman

Lecturer of

Department of Computer Science Engineering

Submitted By

Saurov Sikder

ID: 2021-1-60-053

Submission Date: 29-05-2024

Project Title: Restaurant Management System

1. Project Description:

In our restaurant management system, we can maintain the details for a restaurant. We can maintain staff, customer, food, orders details and can update them or delete them if necessary.

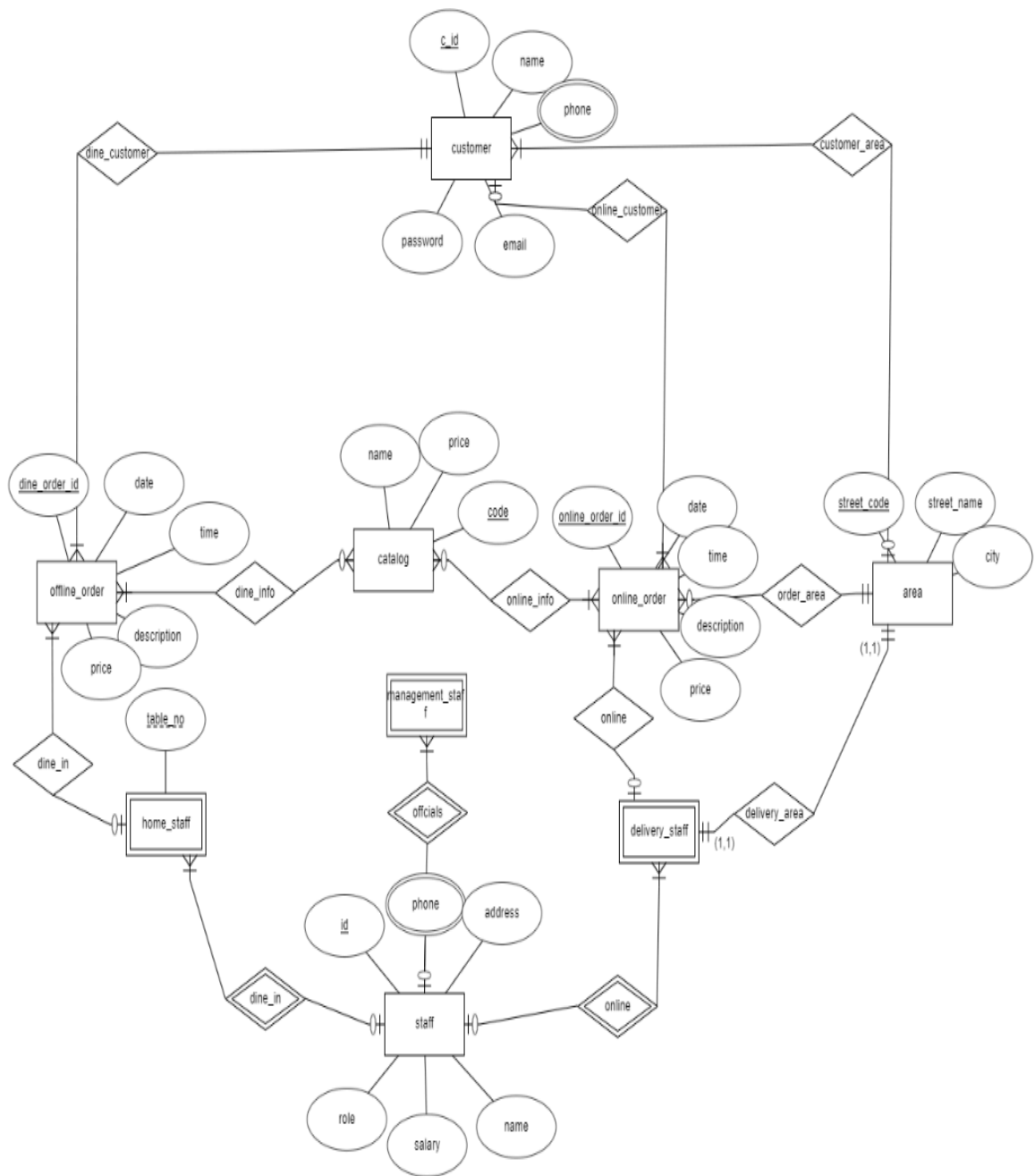
We can individually handle information for offline and online orders. We can track each customer, staff, and order details. We have a unique c_id to identify each customer, code to identify each food, order id to track each customer order. In customer we can get all the information of a customer, in catalog can get all information of food, in staff can get all information of staff, in order can get both online and offline order records. We can organize all the records of the restaurant and also can search PL/SQL queries to find out specific information regarding the management system.

2. Project Overview:

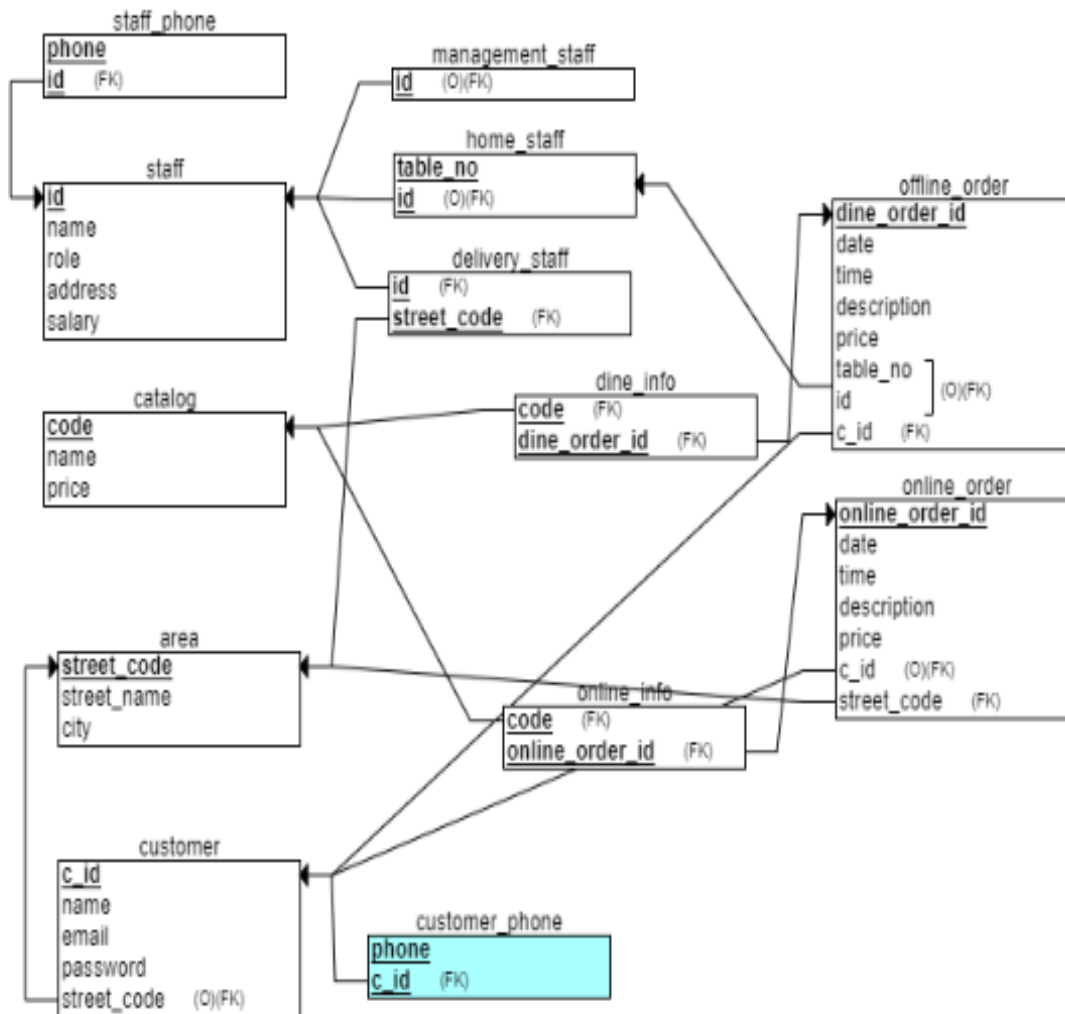
The project describes a conceptual model of the Restaurant Management System that was designed to avoid data redundancy and define functional dependencies. For instance it has entity sets such as Staff, Catalog, Customer, Offline/Online Order, and provides cardinality constraints and mandatory/non-mandatory indicators. E-R diagram and database schema in PL/SQL is included as well as an example of the data insertion. This project involves join queries for queries intended to retrieve particular data and update or delete queries as well as a view creation for the receptionist position. The conclusion affirms that mastery of database and relational schemas and appropriate PL/SQL query language is crucial when handling a restaurant system.

3. Database Schema:

E-R Diagram :



SQL Schema :



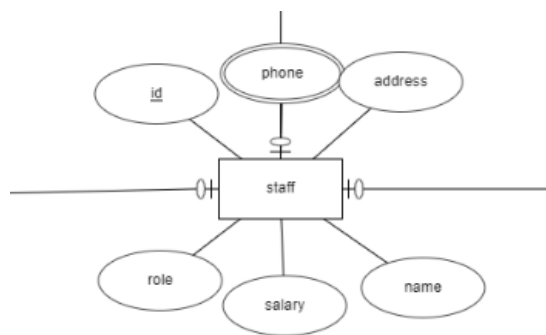
3.1 Object Types (how are the objects of the project)

Entity Sets:

All entity sets in this database are strong entity sets except “home_staff, delivery_staff and management_staff” entity sets. Because each entity set has a primary key of its own and can be identified uniquely without help of any other entity sets. “home_staff, delivery_staff and management_staff” depends on “staff”.

Staff:

Staff is a strong entity. It keeps the information about the ID ,name , phone, address, role and salary. Here Id is the primary key to uniquely identify each staff and others are normal. Phone is a multivalued attribute because an employee may have multiple phone numbers. Role defines their position in the restaurant.



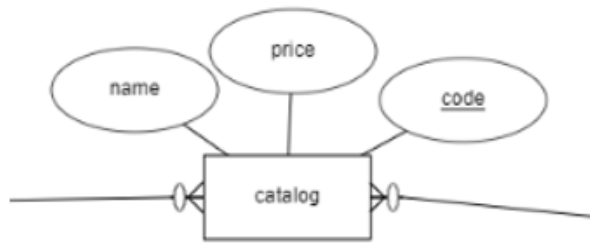
Management_staff, Home_staff & Delivery_Staff:

Management_staff, Home_staff & Delivery_staff are weak entities. They are dependent on staff. They specify the management_staff, home_staff and delivery_staff.

Management_staff	Home_staff	Delivery_staff
id (fk from staff(id))	id (fk from staff(id)) table_no	id (fk from staff(id)) street_code((fk from area(street_code)

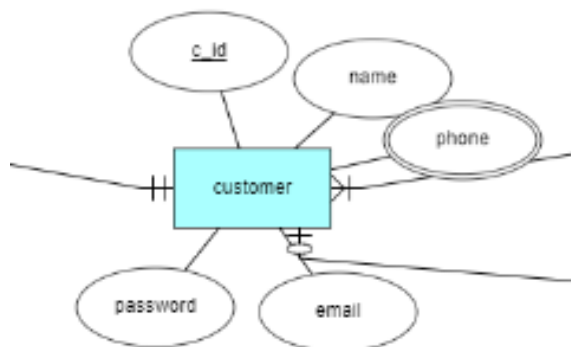
Catalog:

Catalog is a strong entity. It keeps the information name, price, code of foods. Here code is a primary key to uniquely identify each food.



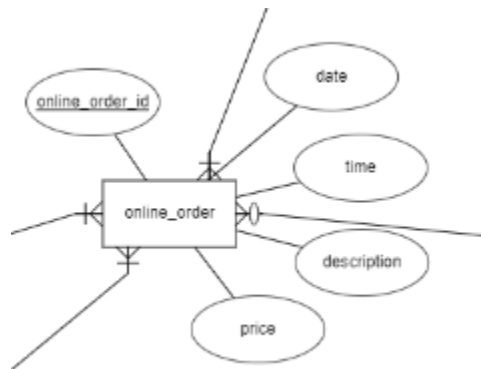
Customer:

Customer is a strong entity. It keeps the information about c_id, name, phone, email, password. Here c_id is uniquely the primary key and phone number is a multivalued attribute.



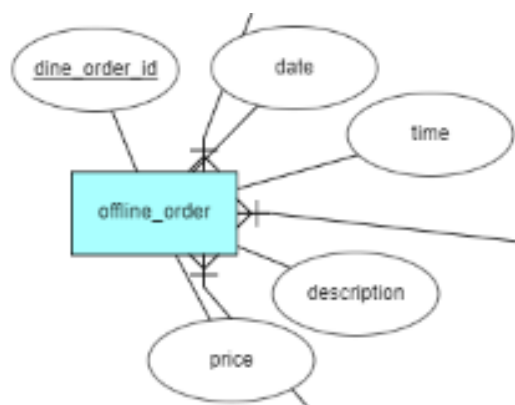
Offline_Orders:

The offline_order table holds dine_order_id which is the primary key. The date of the order is to record the date . It keeps the information about time, description, date, price, dine_order_id.



Online_orders:

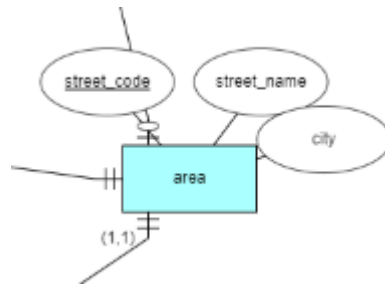
The online_order table holds online_order_id which is the primary key. The date of the order is to record the date . It keeps the information about time, description, date, price and online_order_id.



Area:

In this table street_code is strong entity set. Here street_code is uniquely define as a primary key. It keeps information about street_name and city.

3.2 Tables (how you



created the tables)

```
CREATE TABLE Catlog
(
  name VARCHAR(20) NOT NULL,
  price INT NOT NULL,
  code INT NOT NULL,
  PRIMARY KEY (code)
);
```

```
SQL> CREATE TABLE Catlog
2  (
3    name VARCHAR(20) NOT NULL,
4    price INT NOT NULL,
5    code INT NOT NULL,
6    PRIMARY KEY (code)
7  );

Table created.
```

```
CREATE TABLE area
(
  street_code INT NOT NULL,
```



```
street_name VARCHAR(20) NOT NULL,  
city VARCHAR(20) NOT NULL,  
PRIMARY KEY (street_code)  
);
```

```
SQL> CREATE TABLE area  
2 (  
3   street_code INT NOT NULL,  
4   street_name VARCHAR(20) NOT NULL,  
5   city VARCHAR(20) NOT NULL,  
6   PRIMARY KEY (street_code)  
7 );  
  
Table created.
```

```
CREATE TABLE staff  
(  
  id INT NOT NULL,  
  name VARCHAR(20) NOT NULL,  
  role VARCHAR(20) NOT NULL,  
  address VARCHAR(40) NOT NULL,  
  salary INT NOT NULL,  
  PRIMARY KEY (id)  
);
```

```
SQL> CREATE TABLE staff  
2 (  
3   id INT NOT NULL,  
4   name VARCHAR(20) NOT NULL,  
5   role VARCHAR(20) NOT NULL,  
6   address VARCHAR(40) NOT NULL,  
7   salary INT NOT NULL,  
8   PRIMARY KEY (id)  
9 );  
  
Table created.
```

```
CREATE TABLE delivery_staff  
(  
  id INT NOT NULL,
```

```
street_code INT NOT NULL,  
PRIMARY KEY (id, street_code),  
FOREIGN KEY (id) REFERENCES staff(id),  
FOREIGN KEY (street_code) REFERENCES area(street_code)  
);
```

```
SQL> CREATE TABLE delivery_staff  
2  (  
3    id INT NOT NULL,  
4    street_code INT NOT NULL,  
5    PRIMARY KEY (id, street_code),  
6    FOREIGN KEY (id) REFERENCES staff(id),  
7    FOREIGN KEY (street_code) REFERENCES area(street_code)  
8  );
```

```
CREATE TABLE home_staff  
(  
    table_no INT NOT NULL,  
    id INT,  
    PRIMARY KEY (table_no, id),  
    FOREIGN KEY (id) REFERENCES staff(id)  
);
```

```
SQL> CREATE TABLE home_staff  
2  (  
3    table_no INT NOT NULL,  
4    id INT,  
5    PRIMARY KEY (table_no, id),  
6    FOREIGN KEY (id) REFERENCES staff(id)  
7  );  
  
Table created.
```

```
CREATE TABLE management_staff  
(  
    id INT,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES staff(id)  
);
```

```
SQL> CREATE TABLE management_staff
2  (
3    id INT,
4    PRIMARY KEY (id),
5    FOREIGN KEY (id) REFERENCES staff(id)
6  );

Table created.
```

```
CREATE TABLE staff_phone
(
  phone INT NOT NULL,
  id INT NOT NULL,
  PRIMARY KEY (phone, id),
  FOREIGN KEY (id) REFERENCES staff(id)
);
```

```
SQL> CREATE TABLE staff_phone
2  (
3    phone INT NOT NULL,
4    id INT NOT NULL,
5    PRIMARY KEY (phone, id),
6    FOREIGN KEY (id) REFERENCES staff(id)
7  );

Table created.
```

```
CREATE TABLE customer
(
  c_id INT NOT NULL,
  name VARCHAR(20) NOT NULL,
  email VARCHAR(30) NOT NULL,
  password VARCHAR(20) NOT NULL,
  street_code INT,
  PRIMARY KEY (c_id),
  FOREIGN KEY (street_code) REFERENCES area(street_code)
);
```

```

SQL> CREATE TABLE customer
 2 (
 3   c_id INT NOT NULL,
 4   name VARCHAR(20) NOT NULL,
 5   email VARCHAR(30) NOT NULL,
 6   password VARCHAR(20) NOT NULL,
 7   street_code INT,
 8   PRIMARY KEY (c_id),
 9   FOREIGN KEY (street_code) REFERENCES area(street_code)
10 );

Table created.

```

```

CREATE TABLE offline_order
(
  dine_order_id INT NOT NULL,
  the_date DATE NOT NULL,
  description VARCHAR(50) NOT NULL,
  price INT NOT NULL,
  table_no INT,
  id INT,
  c_id INT NOT NULL,
  PRIMARY KEY (dine_order_id),
  FOREIGN KEY (table_no, id) REFERENCES home_staff(table_no, id),
  FOREIGN KEY (c_id) REFERENCES customer(c_id)
);

```

```

SQL> CREATE TABLE offline_order
 2 (
 3   dine_order_id INT NOT NULL,
 4   the_date DATE NOT NULL,
 5   time VARCHAR(10) NOT NULL,
 6   description VARCHAR(50) NOT NULL,
 7   price INT NOT NULL,
 8   table_no INT,
 9   id INT,
10   c_id INT NOT NULL,
11   PRIMARY KEY (dine_order_id),
12   FOREIGN KEY (table_no, id) REFERENCES home_staff(table_no, id),
13   FOREIGN KEY (c_id) REFERENCES customer(c_id)
14 );

Table created.

```

```
CREATE TABLE online_order
(
  online_order_id INT NOT NULL,
  the_date DATE NOT NULL,
  description VARCHAR(50) NOT NULL,
  price INT NOT NULL,
  c_id INT,
  street_code INT NOT NULL,
  PRIMARY KEY (online_order_id),
  FOREIGN KEY (c_id) REFERENCES customer(c_id),
  FOREIGN KEY (street_code) REFERENCES area(street_code)
);
```

```
SQL> CREATE TABLE online_order
2  (
3    online_order_id INT NOT NULL,
4    the_date DATE NOT NULL,
5    time VARCHAR(10) NOT NULL,
6    description VARCHAR(50) NOT NULL,
7    price INT NOT NULL,
8    c_id INT,
9    street_code INT NOT NULL,
10   PRIMARY KEY (online_order_id),
11   FOREIGN KEY (c_id) REFERENCES customer(c_id),
12   FOREIGN KEY (street_code) REFERENCES area(street_code)
13  );

Table created.
```

```
CREATE TABLE dine_info
(
  code INT NOT NULL,
  dine_order_id INT NOT NULL,
  PRIMARY KEY (code, dine_order_id),
  FOREIGN KEY (code) REFERENCES catlog(code),
  FOREIGN KEY (dine_order_id) REFERENCES offline_order(dine_order_id)
);
```

```

SQL> CREATE TABLE dine_info
2  (
3    code INT NOT NULL,
4    dine_order_id INT NOT NULL,
5    PRIMARY KEY (code, dine_order_id),
6    FOREIGN KEY (code) REFERENCES catlog(code),
7    FOREIGN KEY (dine_order_id) REFERENCES offline_order(dine_order_id)
8  );

Table created.

```

```

CREATE TABLE online_info
(
  code INT NOT NULL,
  online_order_id INT NOT NULL,
  PRIMARY KEY (code, online_order_id),
  FOREIGN KEY (code) REFERENCES catlog(code),
  FOREIGN KEY (online_order_id) REFERENCES online_order(online_order_id)
);

```

```

SQL> CREATE TABLE online_info
2  (
3    code INT NOT NULL,
4    online_order_id INT NOT NULL,
5    PRIMARY KEY (code, online_order_id),
6    FOREIGN KEY (code) REFERENCES catlog(code),
7    FOREIGN KEY (online_order_id) REFERENCES online_order(online_order_id)
8  );

Table created.

```

```

CREATE TABLE customer_phone
(
  phone INT NOT NULL,
  c_id INT NOT NULL,
  PRIMARY KEY (phone, c_id),
  FOREIGN KEY (c_id) REFERENCES customer(c_id)
);

```

```

SQL> CREATE TABLE customer_phone
2  (
3    phone INT NOT NULL,
4    c_id INT NOT NULL,
5    PRIMARY KEY (phone, c_id),
6    FOREIGN KEY (c_id) REFERENCES customer(c_id)
7  );

Table created.

```

Insertion and Sample values:

Staff

```
INSERT INTO staff (id, name, role, address, salary) VALUES
(81, 'fardin', 'delevery_man', 'kurigram', 11000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(82, 'faruk', 'delevery_man', 'kurigram', 11000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(83, 'fardin', 'delevery_man', 'rangpur', 11000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(84, 'fidah', 'delevery_man', 'rangpur', 11000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(85, 'fahim', 'delevery_man', 'rangpur', 11000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(101, 'rezuan', 'receptionoist', 'nilfamari', 20000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(102, 'rabbi', 'receptionoist', 'nilfamari', 20000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(1001, 'mahin', 'manager', 'narayangonj', 30000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(10001, 'taskin', 'home_staff', 'sherpur', 10000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(10002, 'tawhid', 'home_staff', 'khulna', 10000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(10003, 'tahmid', 'home_staff', 'dinajpur', 10000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(10004, 'tamim', 'home_staff', 'dinajpur', 10000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(10005, 'tasin', 'home_staff', 'tangail', 10000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(10006, 'tahsan', 'home_staff', 'sherpur', 10000);
INSERT INTO staff (id, name, role, address, salary) VALUES
(10007, 'tarek', 'home_staff', 'kurigram', 10000);
```

```
SQL> select * from staff;
```

ID	NAME	ROLE	ADDRESS	SALARY
81	fardin	delevery_man	kurigram	11000
82	faruk	delevery_man	kurigram	11000
83	fardin	delevery_man	rangpur	11000
84	fidah	delevery_man	rangpur	11000
85	fahim	delevery_man	rangpur	11000
101	rezuan	receptionoist	nilfamari	20000
102	rabbi	receptionoist	nilfamari	20000
1001	mahin	manager	narayangonj	30000
10001	taskin	home_staff	sherpur	10000
10002	tawhid	home_staff	khulna	10000
10003	tahmid	home_staff	dinajpur	10000
10004	tamim	home_staff	dinajpur	10000
10005	tasin	home_staff	tangail	10000
10006	tahsan	home_staff	sherpur	10000
10007	tarek	home_staff	kurigram	10000

```
15 rows selected.
```

```
INSERT INTO staff_phone (phone, id) VALUES (1521177888, 10005);
INSERT INTO staff_phone (phone, id) VALUES (1521767888, 10007);
INSERT INTO staff_phone (phone, id) VALUES (1521773888, 10006);
INSERT INTO staff_phone (phone, id) VALUES (1521774888, 10003);
INSERT INTO staff_phone (phone, id) VALUES (1521775888, 101);
INSERT INTO staff_phone (phone, id) VALUES (1521777858, 83);
INSERT INTO staff_phone (phone, id) VALUES (1521777869, 84);
INSERT INTO staff_phone (phone, id) VALUES (1521777888, 85);
INSERT INTO staff_phone (phone, id) VALUES (1521777889, 81);
INSERT INTO staff_phone (phone, id) VALUES (1521777889, 82);
INSERT INTO staff_phone (phone, id) VALUES (1521777899, 102);
INSERT INTO staff_phone (phone, id) VALUES (1522777888, 10004);
INSERT INTO staff_phone (phone, id) VALUES (1621777888, 10001);
INSERT INTO staff_phone (phone, id) VALUES (1725836947, 1001);
INSERT INTO staff_phone (phone, id) VALUES (1821777888, 10002);
```



```
SQL> select * from staff_phone;
```

PHONE	ID
1521177888	10005
1521767888	10007
1521773888	10006
1521774888	10003
1521775888	101
1521777858	83
1521777869	84
1521777888	85
1521777889	81
1521777889	82
1521777899	102
1522777888	10004
1621777888	10001
1725836947	1001
1821777888	10002

15 rows selected.

Management staff

```
INSERT INTO management_staff (id) VALUES (1001);
INSERT INTO management_staff (id) VALUES ('101');
INSERT INTO management_staff (id) VALUES ('102');
```

```
SQL> select * from management_staff;
```

ID
1001
101
102

Home staff

```
INSERT INTO home_staff (table_no, id) VALUES (1, 10001);
INSERT INTO home_staff (table_no, id) VALUES (2, 10002);
INSERT INTO home_staff (table_no, id) VALUES (3, 10003);
INSERT INTO home_staff (table_no, id) VALUES (4, 10004);
INSERT INTO home_staff (table_no, id) VALUES (5, 10005);
INSERT INTO home_staff (table_no, id) VALUES (6, 10006);
```

INSERT INTO home_staff (table_no, id) VALUES (7, 10007);

```
SQL> select * from home_staff;
```

TABLE_NO	ID
1	10001
2	10002
3	10003
4	10004
5	10005
6	10006
7	10007

7 rows selected.

Delivery_staff

INSERT INTO delivery_staff (id, street_code) VALUES (81, 1);
INSERT INTO delivery_staff (id, street_code) VALUES (82, 2);
INSERT INTO delivery_staff (id, street_code) VALUES (83, 3);
INSERT INTO delivery_staff (id, street_code) VALUES (84, 4);
INSERT INTO delivery_staff (id, street_code) VALUES (85, 5);

```
SQL> select * from delivery_staff;
```

ID	STREET_CODE
81	1
82	2
83	3
84	4
85	5

Catalog

INSERT INTO catlog (name, price, code) VALUES ('berries', 120, 301);
INSERT INTO catlog (name, price, code) VALUES ('cold cereal', 130, 302);
INSERT INTO catlog (name, price, code) VALUES ('eggs', 50, 303);
INSERT INTO catlog (name, price, code) VALUES ('green tea', 110, 304);
INSERT INTO catlog (name, price, code) VALUES ('penaut butter', 150, 305);

```

INSERT INTO catlog (name, price, code) VALUES ('butter chicken rice', 120, 306);
INSERT INTO catlog (name, price, code) VALUES ('chicken biriyani', 130, 307);
INSERT INTO catlog (name, price, code) VALUES ('rice bowl', 80, 308);
INSERT INTO catlog (name, price, code) VALUES ('veg biriyani', 110, 309);
INSERT INTO catlog (name, price, code) VALUES ('plain rice', 70, 310);
INSERT INTO catlog (name, price, code) VALUES ('finger chicken', 120, 311);
INSERT INTO catlog (name, price, code) VALUES ('chicken grilled', 130, 312);
INSERT INTO catlog (name, price, code) VALUES ('chicken masala', 140, 313);
INSERT INTO catlog (name, price, code) VALUES ('coca-cola', 50, 331);
INSERT INTO catlog (name, price, code) VALUES ('mojo', 30, 332);
INSERT INTO catlog (name, price, code) VALUES ('sprite', 45, 333);
INSERT INTO catlog (name, price, code) VALUES ('fanta', 35, 334);
INSERT INTO catlog (name, price, code) VALUES ('7up', 30, 335);
INSERT INTO catlog (name, price, code) VALUES ('pepsi', 50, 336);

```

```

SQL> select * from catlog;

```

NAME	PRICE	CODE
berries	120	301
cold cereal	130	302
eggs	50	303
green tea	110	304
penaut butter	150	305
butter chicken rice	120	306
chicken biriyani	130	307
rice bowl	80	308
veg biriyani	110	309
plain rice	70	310
finger chicken	120	311
chicken grilled	130	312
chicken masala	140	313
coca-cola	50	331
mojo	30	332
sprite	45	333
fanta	35	334
7up	30	335
pepsi	50	336

```

19 rows selected.

```

Area

```

INSERT INTO area (street_code, street_name, city) VALUES (1, 'basabo', 'dhaka');
INSERT INTO area (street_code, street_name, city) VALUES (2, 'banasree', 'dhaka');
INSERT INTO area (street_code, street_name, city) VALUES (3, 'mugda', 'dhaka');
INSERT INTO area (street_code, street_name, city) VALUES (4, 'khilgaon', 'dhaka');
INSERT INTO area (street_code, street_name, city) VALUES (5, 'malibag', 'dhaka');
INSERT INTO area (street_code, street_name, city) VALUES (6, 'farmgate', 'dhaka');

```

```
SQL> select * from area;
```

STREET_CODE	STREET_NAME	CITY
1	basabo	dhaka
2	banasree	dhaka
3	mugda	dhaka
4	khilgaon	dhaka
5	malibag	dhaka
6	farmgate	dhaka

```
6 rows selected.
```

Customer

```
INSERT INTO customer (c_id, name, email, password, street_code) VALUES
(501, 'Nuru', 'nuru@gmail.com', '2323', 2);
INSERT INTO customer (c_id, name, email, password, street_code) VALUES
(502, 'Nabil', 'nabil@gmail.com', '1221', 1);
INSERT INTO customer (c_id, name, email, password, street_code) VALUES
(503, 'Nahid', 'nahid@gmail.com', '3344', 3);
INSERT INTO customer (c_id, name, email, password, street_code) VALUES
(504, 'Nibir', 'nibir@gmail.com', '7788', 6);
INSERT INTO customer (c_id, name, email, password, street_code) VALUES
(505, 'Nikhad', 'nik@gmail.com', '7654', 4);
INSERT INTO customer (c_id, name, email, password, street_code) VALUES
(506, 'Nokib', 'nok@gmail.com', '2234', 1);
```

```
SQL> select * from customer;
```

C_ID	NAME	EMAIL	PASSWORD	STREET_CODE
501	Nuru	nuru@gmail.com	2323	2
502	Nabil	nabil@gmail.com	1221	1
503	Nahid	nahid@gmail.com	3344	3
504	Nibir	nibir@gmail.com	7788	6
505	Nikhad	nik@gmail.com	7654	4
506	Nokib	nok@gmail.com	2234	1

```
6 rows selected.
```

Customer phone

```
INSERT INTO customer_phone (phone, c_id) VALUES (1955118543, 501);
INSERT INTO customer_phone (phone, c_id) VALUES (1955118544, 506);
INSERT INTO customer_phone (phone, c_id) VALUES (1955118545, 505);
INSERT INTO customer_phone (phone, c_id) VALUES (1955118546, 504);
INSERT INTO customer_phone (phone, c_id) VALUES (1955118547, 503);
```

```
INSERT INTO customer_phone (phone, c_id) VALUES (1955118548, 502);
```

PHONE	C_ID
1955118543	501
1955118544	506
1955118545	505
1955118548	502

Offline Order

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

```
(3001, TO_DATE ('21/October/22 5:30:00 PM' , 'DD/MON/YY HH:MI:SSAM'), 'not
applicable', 500, 1, 10001, 501);
```

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

```
(3002, TO_DATE ('21/December/22 5:31:00 PM', 'DD/MON/YY HH:MI:SSAM'), 'not
applicable', 700, 4, 10004, 504);
```

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

```
(3003, TO_DATE ('21/December/22 5:31:00 PM', 'DD/MON/YY HH:MI:SSAM'), 'not
applicable', 500, 3, 10003, 502);
```

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

```
(3004, TO_DATE ('23/December/22 10:00:00 AM', 'DD/MON/YY HH:MI:SSAM'), 'not
applicable', 180, 1, 10001, 505);
```

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

```
(3005, TO_DATE ('21/December/22 11:00:00 AM', 'DD/MON/YY HH:MI:SSAM'), 'not
applicable', 190, 1, 10001, 505);
```

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

```
(3006, TO_DATE ('24/December/22 12:00:00 PM', 'DD/MON/YY HH:MI:SSAM'), 'not
applicable', 210, 2, 10002, 501);
```

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

```
(3007, TO_DATE ('25/December/22 2:00:00 PM', 'DD/MON/YY HH:MI:SSAM'), 'not
applicable', 220, 5, 10005, 504);
```

```
INSERT INTO offline_order (dine_order_id, the_date, description, price, table_no, id, c_id)
VALUES
```

(3008, TO_DATE ('26/December/22 2:00:00 PM', 'DD/MON/YY HH:MI:SSAM'), 'not applicable', 150, 7, 10007, 506);

```
SQL> select * from offline_order;
```

DINE_ORDER_ID	THE_DATE	DESCRIPTION	PRICE	TABLE_NO	ID	C_ID
3001	21-OCT-22	not applicable	500	1	10001	501
3002	21-DEC-22	not applicable	700	4	10004	504
3003	21-DEC-22	not applicable	500	3	10003	502
3004	23-DEC-22	not applicable	180	1	10001	505
3005	21-DEC-22	not applicable	190	1	10001	505
3006	24-DEC-22	not applicable	210	2	10002	501
3007	25-DEC-22	not applicable	220	5	10005	504
3008	26-DEC-22	not applicable	150	7	10007	506

8 rows selected.

Dine info

```
INSERT INTO dine_info (code, dine_order_id) VALUES (302, 3008);
INSERT INTO dine_info (code, dine_order_id) VALUES (303, 3001);
INSERT INTO dine_info (code, dine_order_id) VALUES (303, 3004);
INSERT INTO dine_info (code, dine_order_id) VALUES (304, 3001);
INSERT INTO dine_info (code, dine_order_id) VALUES (304, 3006);
INSERT INTO dine_info (code, dine_order_id) VALUES (306, 3001);
INSERT INTO dine_info (code, dine_order_id) VALUES (306, 3008);
INSERT INTO dine_info (code, dine_order_id) VALUES (310, 3001);
INSERT INTO dine_info (code, dine_order_id) VALUES (310, 3003);
INSERT INTO dine_info (code, dine_order_id) VALUES (310, 3008);
INSERT INTO dine_info (code, dine_order_id) VALUES (311, 3004);
INSERT INTO dine_info (code, dine_order_id) VALUES (312, 3007);
INSERT INTO dine_info (code, dine_order_id) VALUES (313, 3005);
INSERT INTO dine_info (code, dine_order_id) VALUES (313, 3007);
INSERT INTO dine_info (code, dine_order_id) VALUES (331, 3001);
INSERT INTO dine_info (code, dine_order_id) VALUES (331, 3003);
INSERT INTO dine_info (code, dine_order_id) VALUES (332, 3007);
INSERT INTO dine_info (code, dine_order_id) VALUES (333, 3008);
```

```
SQL> select * from dine_info;
```

CODE	DINE_ORDER_ID
302	3008
303	3001
303	3004
304	3001
304	3006
306	3001
306	3008
310	3001
310	3003
310	3008
311	3004
312	3007
313	3005
313	3007
331	3001
331	3003
332	3007
333	3008

```
18 rows selected.
```

Online Order

```
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5001, TO_DATE ('21/December/22 5:30:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 669, 505, 2);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5002, TO_DATE ('23/December/22 10:00:00 AM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 200, 501, 3);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5003, TO_DATE ('24/December/22 7:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 250, 502, 2);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5004, TO_DATE ('26/December/22 10:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 300, 502, 2);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5005, TO_DATE ('23/December/22 2:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 350, 506, 4);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5006, TO_DATE ('26/December/22 3:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 400, 504, 6);
```

```

INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5007, TO_DATE ('23/December/22 11:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 420, 503, 5);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5011, TO_DATE ('23/December/22 10:00:00 AM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 200, 501, 1);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5012, TO_DATE ('22/December/22 7:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 240, 502, 4);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5013, TO_DATE ('26/December/22 12:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 300, 502, 4);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5014, TO_DATE ('26/December/22 3:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 550, 503, 5);
INSERT INTO online_order (online_order_id, the_date, description, price, c_id, street_code)
VALUES (5015, TO_DATE ('26/December/22 3:00:00 PM', 'DD/MON/YY HH:MI:SSAM'),
'not applicable', 400, 503, 5);

```

ONLINE_ORDER_ID	THE_DATE	DESCRIPTION	PRICE	C_ID	STREET_CODE
5001	21-DEC-22	not applicable	669	505	2
5002	23-DEC-22	not applicable	200	501	3
5003	24-DEC-22	not applicable	250	502	2
5004	26-DEC-22	not applicable	300	502	2
5005	23-DEC-22	not applicable	350	506	4
5006	26-DEC-22	not applicable	400	504	6
5007	23-DEC-22	not applicable	420	503	5
5011	23-DEC-22	not applicable	200	501	1
5012	22-DEC-22	not applicable	240	502	4
5013	26-DEC-22	not applicable	300	502	4
5014	26-DEC-22	not applicable	550	503	5
5015	26-DEC-22	not applicable	400	503	5

12 rows selected.

Online info

```

INSERT INTO online_info (code, online_order_id) VALUES (301, 5001);
INSERT INTO online_info (code, online_order_id) VALUES (302, 5001);
INSERT INTO online_info (code, online_order_id) VALUES (302, 5004);
INSERT INTO online_info (code, online_order_id) VALUES (308, 5002);
INSERT INTO online_info (code, online_order_id) VALUES (309, 5002);
INSERT INTO online_info (code, online_order_id) VALUES (311, 5004);
INSERT INTO online_info (code, online_order_id) VALUES (313, 5004);

```



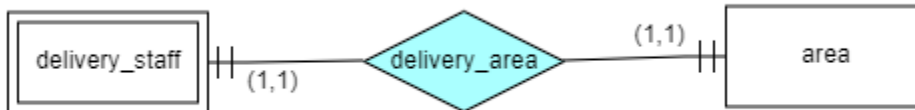
```
SQL> select * from online_info;
```

CODE	ONLINE_ORDER_ID
301	5001
302	5001
302	5004
308	5002
309	5002
311	5004
313	5004

```
7 rows selected.
```

3.3 Relationships (how the relationships are defined)

Cardinality constraints and participation:



Delivery_staff – Area:

Cardinality constraint: The cardinality constraint from the “delivery_staff” entity set to the “area” entity set is one-to-one. This means delivery_staff may have one area and area may have only one delivery_staff with specific area code. Here, the delivery staff is a weak entity set.

Participation: delivery_staff is total for area but area is total for delivery_staff.

Home_staff – staff:

Staff – delivery_staff:

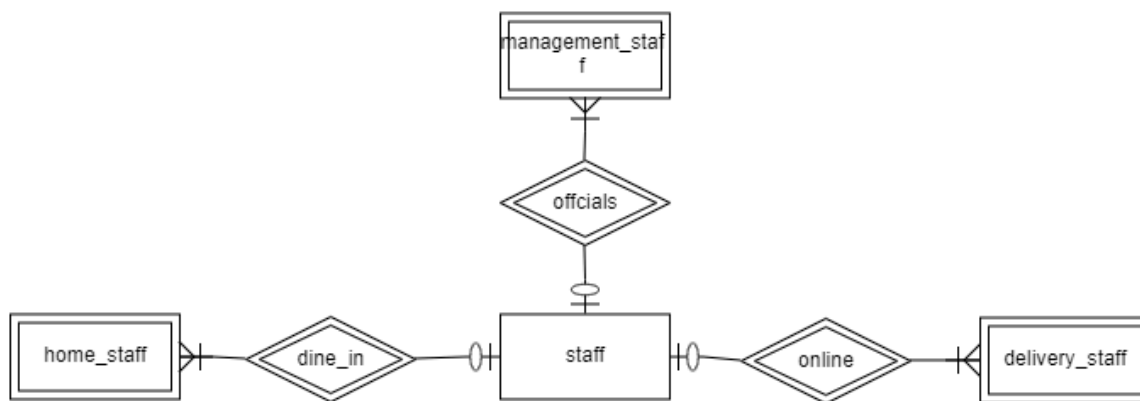
Staff – management_staff:

Cardinality constraint: Staff and managements_staff, home_staff, delivery_staff is one to many. That means staff can have many managements_staff, home_staff, delivery_staff but they must be connected with one staff individually.

Participation: home_staff is partial for staff but staff is total for home_staff.

Staff is total for delivery staff but delivery staff is partial for staff.

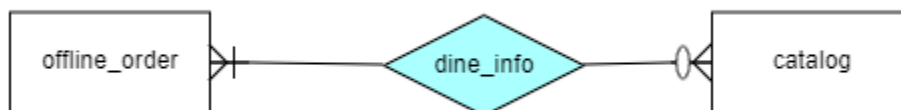
Staff is total for management_staff but management_staff is partial for staff.



Offline_order – Catalog:

Cardinality constraint: The cardinality constraint from the “offline_order” entity set to the “catalog” entity set is many-to-many. That means offline_order can have more than one catalog and catalog can have more than one offline_order.

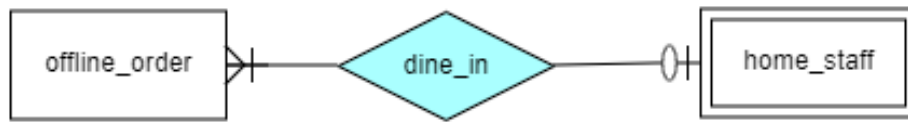
Participation: offline_order is partial for catalog but catalog is total for offline_order.



Offline_order –Home_staff:

Cardinality constraint: The cardinality constraint from the “offline_order” entity set to the “home_staff” entity set is one-to-many. This means offline_orders may have many home_staff and home_staff may have only offline_order.

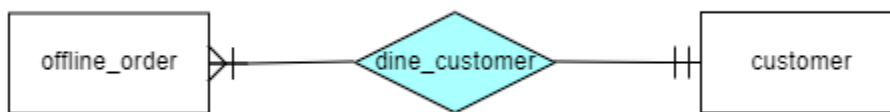
Participation: offline_order is partial for home_staff but home_staff is total for offline_order.



Offline_order – Customer:

Cardinality constraint: The “Offline_orders” entity set to the “customer” entity set is one-to-many. This means an offline_order may have more than one customer, as well as a customer may have more than one offline_orders.

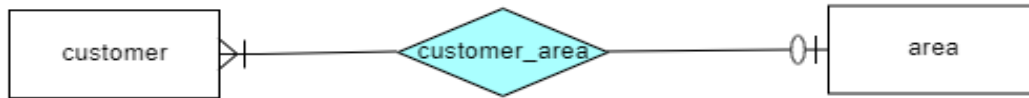
Participation: offline_order is total for customer and customer is also total for offline_order.



Customer – Area:

Cardinality constraint: The “customer” entity set to the “area” entity set is one-to-many. This means a customer can have only one area but areas have more than one customer. It means many customers may live in the same area

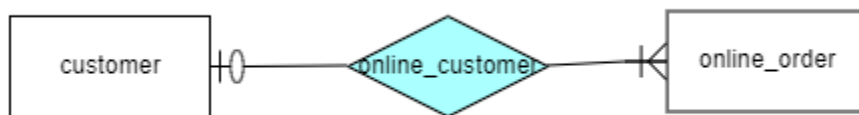
Participation: customer is partial for area but area is total for customer.



Customer – Online_order:

Cardinality constraint: The “customer” entity set to the “online_order” entity set is many-to-one. This customer entity may have many online_order but online_orders entity set has one customer.

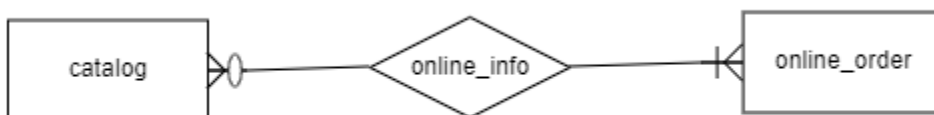
Participation: customer is total for online_order but online_order is partial for customer.



Catalog – Online_order:

Cardinality constraint: The “catalog” entity set to the “online_order” entity set is many-to-many. This means a catalog can have done more than one online_orders of food and a online_orders are also be included in catalog entity set.

Participation: catalog is total for online_order but online_order is partial for catalog.



Online_order – Delivery_staff:

Cardinality constraint: The cardinality constraint from the “online_order” entity set to the “delivery_staff” entity set is one-to-many. That means online orders can only be delivered by delivery staff and one delivery staff have many online orders.

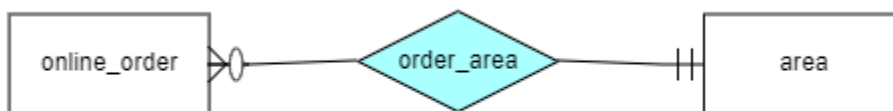
Participation: online_order is partial for delivery_staff but delivery_staff is total for online_order.



Online_order – Area:

Cardinality constraint: The “online_orders” entity set to the “area” entity set is one-to-many. This online order entity may have only one area with street code but area entity has many online orders with different customer.

Participation: online_order is total for area but area is partial for online_order.



Implementation details

Database schema creation:

The schema includes tables for catalog, area, staff, delivery staff, home staff, management staff, customer, offline order, online order, dine info, online info, staff phone, and customer phone and those are created with appropriate data types and constraints, such as primary keys and foreign keys, to ensure data integrity.

Data management and stored procedures:

The data is inserted into tables using SQL INSERT INTO statements and diversified to cover various scenarios, such as different staff roles, customer orders, and delivery locations. CRUD operations are implemented through stored procedures for the Catlog and area tables. Procedures are created for adding, retrieving, updating, and deleting records in these tables, enhancing data manipulation capabilities.

Deployment, optimization and error handling:

For our project, error handling is implemented in stored procedures and PL/SQL blocks for reliability. Testing validates correctness and performance, with various scenarios covered. Documentation comprehensively describes schema, procedures, queries, and implementation for clarity. Optimization ensures efficiency via query performance, indexing, and resource use. Deployment includes schema, procedures, and data, while maintenance tasks ensure ongoing reliability and security.

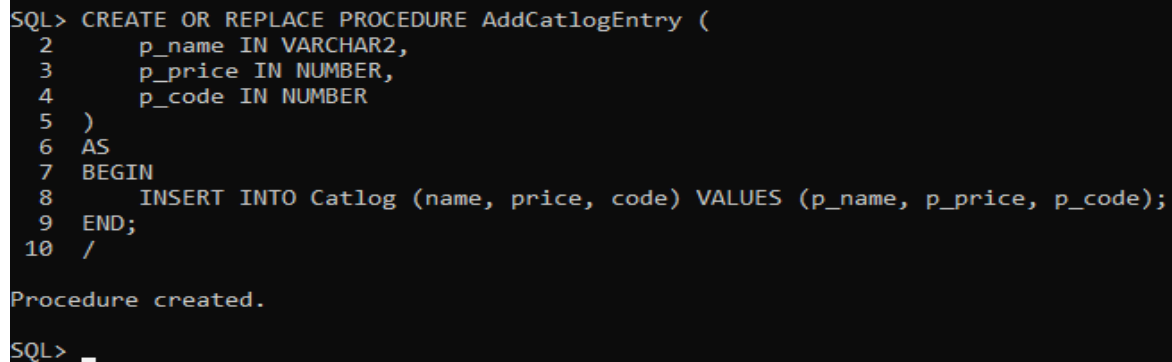
Procedures for crud operation(Catlog and area table):

In this project, PL/SQL CRUD operations are essential for managing data effectively in SQL plus. For creation, read, updating, and deletion of records within tables, ensuring comprehensive data manipulation capabilities. By implementing these operations through PL/SQL procedures, we can maintain data integrity and enhance code maintainability. The procedures also provide a clear framework for handling these tasks, making database interactions more efficient and reliable. This approach simplifies database management and promotes better application performance. Here we give two examples.

1. Catlog

Create:

```
CREATE OR REPLACE PROCEDURE AddCatlogEntry (  
    p_name IN VARCHAR2,  
    p_price IN NUMBER,  
    p_code IN NUMBER  
)  
AS  
BEGIN  
    INSERT INTO Catlog (name, price, code) VALUES (p_name, p_price, p_code);  
END;  
/
```



```
SQL> CREATE OR REPLACE PROCEDURE AddCatlogEntry (  
2     p_name IN VARCHAR2,  
3     p_price IN NUMBER,  
4     p_code IN NUMBER  
5 )  
6 AS  
7 BEGIN  
8     INSERT INTO Catlog (name, price, code) VALUES (p_name, p_price, p_code);  
9 END;  
10 /  
  
Procedure created.  
  
SQL> _
```

Read:

```
SET SERVEROUTPUT ON;
```

```
CREATE OR REPLACE PROCEDURE GetCatlogEntries
```

```
AS
```

```
BEGIN
```

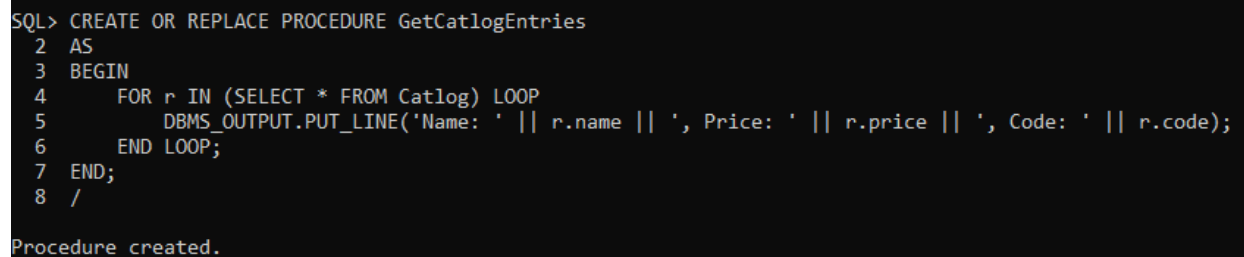
```
    FOR r IN (SELECT * FROM Catlog) LOOP
```

```
        DBMS_OUTPUT.PUT_LINE('Name: ' || r.name || ', Price: ' || r.price || ', Code: ' || r.code);
```

```
    END LOOP;
```

```
END;
```

```
/
```



```
SQL> CREATE OR REPLACE PROCEDURE GetCatlogEntries
 2  AS
 3  BEGIN
 4      FOR r IN (SELECT * FROM Catlog) LOOP
 5          DBMS_OUTPUT.PUT_LINE('Name: ' || r.name || ', Price: ' || r.price || ', Code: ' || r.code);
 6      END LOOP;
 7  END;
 8  /

Procedure created.
```

Update:

```
CREATE OR REPLACE PROCEDURE UpdateCatlogEntry (
```

```
    p_code IN NUMBER,
```

```
    p_name IN VARCHAR2,
```

```
    p_price IN NUMBER
```

```
)
```

```
AS
```

```
BEGIN
```

```
    UPDATE Catlog
```

```
    SET name = p_name, price = p_price
```

```
    WHERE code = p_code;
```

```
END;
```

```
/
```



```

SQL> CREATE OR REPLACE PROCEDURE UpdateCatlogEntry (
  2     p_code IN NUMBER,
  3     p_name IN VARCHAR2,
  4     p_price IN NUMBER
  5 )
  6 AS
  7 BEGIN
  8     UPDATE Catlog
  9     SET name = p_name, price = p_price
 10     WHERE code = p_code;
 11 END;
 12 /

```

Procedure created.

Delete:

```

CREATE OR REPLACE PROCEDURE DeleteCatlogEntry (
  p_code IN NUMBER
)
AS
BEGIN
  DELETE FROM Catlog WHERE code = p_code;
END;
/

```

```

SQL> CREATE OR REPLACE PROCEDURE DeleteCatlogEntry (
  2     p_code IN NUMBER
  3 )
  4 AS
  5 BEGIN
  6     DELETE FROM Catlog WHERE code = p_code;
  7 END;
  8 /

```

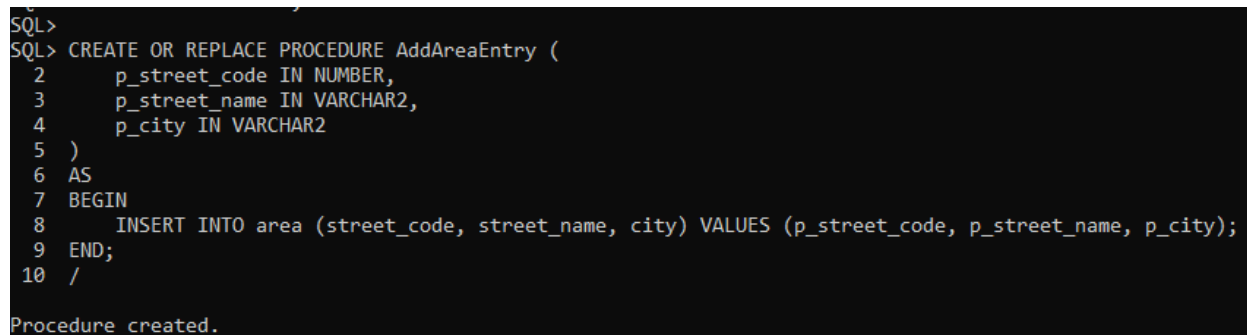
Procedure created.

1. Catlog

Create:

```
SET SERVEROUTPUT ON;
```

```
CREATE OR REPLACE PROCEDURE AddAreaEntry (  
    p_street_code IN NUMBER,  
    p_street_name IN VARCHAR2,  
    p_city IN VARCHAR2  
)  
AS  
BEGIN  
    INSERT INTO area (street_code, street_name, city) VALUES (p_street_code, p_street_name,  
p_city);  
END;  
/
```



```
SQL>  
SQL> CREATE OR REPLACE PROCEDURE AddAreaEntry (  
2     p_street_code IN NUMBER,  
3     p_street_name IN VARCHAR2,  
4     p_city IN VARCHAR2  
5 )  
6 AS  
7 BEGIN  
8     INSERT INTO area (street_code, street_name, city) VALUES (p_street_code, p_street_name, p_city);  
9 END;  
10 /  
  
Procedure created.
```

Read:

```
SET SERVEROUTPUT ON;
```

```
CREATE OR REPLACE PROCEDURE GetAreaEntries  
AS  
BEGIN  
    FOR r IN (SELECT * FROM area) LOOP  
        DBMS_OUTPUT.PUT_LINE('Street Code: ' || r.street_code || ', Street Name: ' ||  
r.street_name || ', City: ' || r.city);  
    END LOOP;  
END;
```

```

/
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> CREATE OR REPLACE PROCEDURE GetAreaEntries
2 AS
3 BEGIN
4     FOR r IN (SELECT * FROM area) LOOP
5         DBMS_OUTPUT.PUT_LINE('Street Code: ' || r.street_code || ', Street Name: ' || r.street_name || ', City: ' || r.city);
6     END LOOP;
7 END;
8 /

procedure created.

```

Update:

```
SET SERVEROUTPUT ON;
```

```

CREATE OR REPLACE PROCEDURE UpdateAreaEntry (
    p_street_code IN NUMBER,
    p_street_name IN VARCHAR2,
    p_city IN VARCHAR2
)
AS
BEGIN
    UPDATE area
    SET street_name = p_street_name, city = p_city
    WHERE street_code = p_street_code;
END;
/

```

```

SQL>
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> CREATE OR REPLACE PROCEDURE UpdateAreaEntry (
2     p_street_code IN NUMBER,
3     p_street_name IN VARCHAR2,
4     p_city IN VARCHAR2
5 )
6 AS
7 BEGIN
8     UPDATE area
9     SET street_name = p_street_name, city = p_city
10    WHERE street_code = p_street_code;
11 END;
12 /

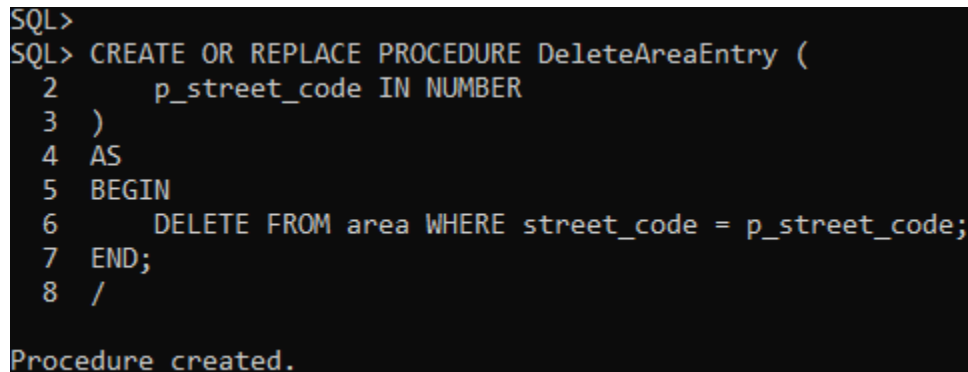
Procedure created.

```

Delete:

```
SET SERVEROUTPUT ON;
```

```
CREATE OR REPLACE PROCEDURE DeleteAreaEntry (  
    p_street_code IN NUMBER  
)  
AS  
BEGIN  
    DELETE FROM area WHERE street_code = p_street_code;  
END;  
/
```

A screenshot of a terminal window with a black background and light blue/grey text. It shows the execution of an SQL command to create or replace a procedure named 'DeleteAreaEntry'. The command is entered line by line, and the terminal responds with 'Procedure created.' at the bottom.

```
SQL>  
SQL> CREATE OR REPLACE PROCEDURE DeleteAreaEntry (  
2      p_street_code IN NUMBER  
3  )  
4  AS  
5  BEGIN  
6      DELETE FROM area WHERE street_code = p_street_code;  
7  END;  
8  /  
  
Procedure created.
```

Sample queries:

Sample queries based on this database schema allow for efficient extraction of meaningful insights and information from the underlying data. These queries make the structured relationships between various entities such as customers, staff members, orders, and items. By combining tables through JOIN operations and applying functions like update, delete and others which queries retrieve diverse sets of data. For instance, the total sales amount for each item, the count of orders made by each customer, or even details such as staff names along with their respective roles and contact information. Moreover, by incorporating filtering conditions based on dates or specific attributes, these queries can offer targeted analyses, such as identifying customers who ordered online on a particular date or computing sales figures for a specific time period. Overall, these sample queries showcase the versatility and power of SQL in querying relational databases to support decision-making and analysis in various business contexts.

Join Queries:

Query 1: find all info of the customer whose customer id = 502

SQL:

```
SQL> select * from customer NATURAL JOIN customer_phone WHERE c_id = 502;
```

C_ID	NAME	EMAIL	PASSWORD	STREET_CODE	PHONE
502	Nabil	nabil@gmail.com	1221	1	1955118548

Query 2: find the number of offline orders of customer whose id = 504

SQL:

SET SERVEROUTPUT ON

DECLARE

 v_customer_id NUMBER := 504;

 v_order_count NUMBER;

BEGIN

 -- Query to count the number of offline orders for the customer

 SELECT COUNT(*) INTO v_order_count

 FROM offline_order

 WHERE c_id = v_customer_id;

 -- Output the result

 DBMS_OUTPUT.PUT_LINE('Number of offline orders for customer ' || v_customer_id || ': ' ||
v_order_count);

END;

/

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     v_customer_id NUMBER := 504;
  3     v_order_count NUMBER;
  4 BEGIN
  5     -- Query to count the number of offline orders for the customer
  6     SELECT COUNT(*) INTO v_order_count
  7     FROM offline_order
  8     WHERE c_id = v_customer_id;
  9
 10     -- Output the result
 11     DBMS_OUTPUT.PUT_LINE('Number of offline orders for customer ' || v_customer_id || ': ' || v_order_count);
 12 END;
 13
 14 /
Number of offline orders for customer 504: 2

PL/SQL procedure successfully completed.
```

Query 3: find the delivery staff whose salary is more than 11k

SQL:

```
SET SERVEROUTPUT ON;
```

```
DECLARE
```

```
    v_salary_threshold NUMBER := 11000;
```

```
BEGIN
```

```
    FOR staff_rec IN (
```

```
        SELECT s.id, s.name, s.salary
```

```
        FROM staff s
```

```
        JOIN delivery_staff ds ON s.id = ds.id
```

```
        WHERE s.salary > v_salary_threshold
```

```
    ) LOOP
```

```
        DBMS_OUTPUT.PUT_LINE('ID: ' || staff_rec.id || ', Name: ' || staff_rec.name || ', Salary: ' || staff_rec.salary);
```

```
    END LOOP;
```

```
END;
```

```
/
```

```
SELECT staff.id,name,role,salary FROM staff JOIN delivery_staff WHERE staff.id =  
delivery_staff.id AND salary >11000;
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Number of rows: 25 ▼ Filter rows: Sort by

Extra options

id	name	role	salary
82	faruk	delevery_man	15000
85	fahim	delevery_man	13000

Query 4: find the online orders and name of customer id = 502 .

SQL:

```

SET SERVEROUTPUT ON;
DECLARE
    v_customer_id NUMBER := 502;
BEGIN
    FOR order_rec IN (
        SELECT o.online_order_id, o.the_date, o.description, o.price, c.name AS
customer_name
        FROM online_order o
        JOIN customer c ON o.c_id = c.c_id
        WHERE o.c_id = v_customer_id
    ) LOOP
        DBMS_OUTPUT.PUT_LINE('Online Order ID: ' || order_rec.online_order_id || ', Date: '
|| order_rec.the_date || ', Description: ' || order_rec.description || ', Price: ' || order_rec.price || ',
Customer Name: ' || order_rec.customer_name);
    END LOOP;
END;
/

```

```

SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
2   v_customer_id NUMBER := 502;
3 BEGIN
4   -- Query to fetch online orders and customer names for the specified customer ID
5   FOR order_rec IN (
6     SELECT o.online_order_id, o.the_date, o.description, o.price, c.name AS customer_name
7     FROM online_order o
8     JOIN customer c ON o.c_id = c.c_id
9     WHERE o.c_id = v_customer_id
10  ) LOOP
11    -- Output the details of online orders and customer names
12    DBMS_OUTPUT.PUT_LINE('Online Order ID: ' || order_rec.online_order_id || ', Date: ' || order_rec.the_date || ', Description: ' ||
order_rec.description || ', Price: ' || order_rec.price || ', Customer Name: ' || order_rec.customer_name);
13  END LOOP;
14 END;
15 /
Online Order ID: 5003, Date: 24-DEC-22, Description: not applicable, Price: 250, Customer Name: Nabil
Online Order ID: 5004, Date: 26-DEC-22, Description: not applicable, Price: 300, Customer Name: Nabil
Online Order ID: 5012, Date: 22-DEC-22, Description: not applicable, Price: 240, Customer Name: Nabil
Online Order ID: 5013, Date: 26-DEC-22, Description: not applicable, Price: 300, Customer Name: Nabil
PL/SQL procedure successfully completed.

```

Query 5: find the orders delivered by fardin

SQL:

```

SET SERVEROUTPUT ON
DECLARE
    v_delivery_staff_name VARCHAR2(20) := 'fardin';
BEGIN
    FOR order_rec IN (
        SELECT o.dine_order_id, o.the_date, o.description, o.price, c.name AS customer_name

```

```

FROM offline_order o
JOIN customer c ON o.c_id = c.c_id
JOIN delivery_staff ds ON o.id = ds.id
JOIN staff s ON ds.id = s.id
WHERE s.name = v_delivery_staff_name
) LOOP
    DBMS_OUTPUT.PUT_LINE('Dine Order ID: ' || order_rec.dine_order_id || ', Date: ' ||
order_rec.the_date || ', Description: ' || order_rec.description || ', Price: ' || order_rec.price || ',
Customer Name: ' || order_rec.customer_name);
    END LOOP;
END;
/

```

[SELECT](#) name , online_order_id, date,time FROM staff join delivery_staff JOIN
online_order WHERE staff.id = delivery_staff.id AND delivery_staff.street_code =
online_order.street_code AND name = 'fardin';

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Number of rows: 25 ▼ Filter rows: Sort by

Extra options

name	online_order_id	date	time
fardin	5011	2022-12-23	10:00am
fardin	5002	2022-12-23	10:00pm

Query 6: Find the all the order of customer nuru.

SQL:

SET SERVEROUTPUT ON

DECLARE

v_customer_name VARCHAR2(20) := 'Nuru';

BEGIN

FOR order_rec IN (

SELECT o.dine_order_id, o.the_date, o.description, o.price

FROM offline_order o

JOIN customer c ON o.c_id = c.c_id

WHERE c.name = v_customer_name


```

        UNION ALL
        SELECT oo.online_order_id, oo.the_date, oo.description, oo.price
        FROM online_order oo
        JOIN customer c ON oo.c_id = c.c_id
        WHERE c.name = v_customer_name
    ) LOOP
        DBMS_OUTPUT.PUT_LINE('Order ID: ' || order_rec.dine_order_id || ', Date: ' ||
order_rec.the_date || ', Description: ' || order_rec.description || ', Price: ' || order_rec.price);
    END LOOP;
END;
/

```

```

SQL> DECLARE
2     v_customer_name VARCHAR2(20) := 'Nuru';
3 BEGIN
4     -- Query to fetch all orders of the specified customer
5     FOR order_rec IN (
6         SELECT o.dine_order_id, o.the_date, o.description, o.price
7         FROM offline_order o
8         JOIN customer c ON o.c_id = c.c_id
9         WHERE c.name = v_customer_name
10        UNION ALL
11        SELECT oo.online_order_id, oo.the_date, oo.description, oo.price
12        FROM online_order oo
13        JOIN customer c ON oo.c_id = c.c_id
14        WHERE c.name = v_customer_name
15    ) LOOP
16        -- Output the details of each order
17        DBMS_OUTPUT.PUT_LINE('Order ID: ' || order_rec.dine_order_id || ', Date: ' ||
order_rec.the_date || ', Description: ' || order_rec.description || ', Price: ' || order_
rec.price);
18    END LOOP;
19 END;
20 /
Order ID: 3001, Date: 21-OCT-22, Description: not applicable, Price: 500
Order ID: 3006, Date: 24-DEC-22, Description: not applicable, Price: 210
Order ID: 5002, Date: 23-DEC-22, Description: not applicable, Price: 200
Order ID: 5011, Date: 23-DEC-22, Description: not applicable, Price: 200

PL/SQL procedure successfully completed.

```

Query 7: Find the online orders whose order value is more than 500 and area is basabo

SQL:

```

SET SERVEROUTPUT ON;
DECLARE
    v_order_value_threshold NUMBER := 500;

```

```

v_area_name VARCHAR2(20) := 'basabo';
BEGIN
  FOR order_rec IN (
    SELECT oo.online_order_id, oo.the_date, oo.description, oo.price, c.name AS
customer_name, a.street_name AS area_name
    FROM online_order oo
    JOIN customer c ON oo.c_id = c.c_id
    JOIN area a ON oo.street_code = a.street_code
    WHERE oo.price > v_order_value_threshold
    AND a.street_name = v_area_name
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Online Order ID: ' || order_rec.online_order_id || ', Date: '
|| order_rec.the_date || ', Description: ' || order_rec.description || ', Price: ' || order_rec.price || ',
Customer Name: ' || order_rec.customer_name || ', Area Name: ' || order_rec.area_name);
  END LOOP;
END;
/

```

✓ Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)

```

SELECT street_name, online_order.online_order_id,price FROM online_order JOIN area
WHERE online_order.street_code = area.street_code AND street_name = 'basabo' AND
price >100;

```

☐ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

☐ Show all | Number of rows: 25 ▼ | Filter rows:

Extra options

street_name	online_order_id	price
basabo	5011	200

Query 8: find all online orders of nibir.

SQL:

```

SET SERVEROUTPUT ON;
DECLARE
  v_customer_name VARCHAR2(20) := 'Nibir';

```

```

BEGIN
  FOR order_rec IN (
    SELECT oo.online_order_id, oo.the_date, oo.description, oo.price
    FROM online_order oo
    JOIN customer c ON oo.c_id = c.c_id
    WHERE c.name = v_customer_name
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Online Order ID: ' || order_rec.online_order_id || ', Date: '
|| order_rec.the_date || ', Description: ' || order_rec.description || ', Price: ' || order_rec.price);
  END LOOP;
END;
/

```

```

SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2   v_customer_name VARCHAR2(20) := 'Nibir';
  3   BEGIN
  4     -- Query to fetch all online orders of the specified customer
  5     FOR order_rec IN (
  6       SELECT oo.online_order_id, oo.the_date, oo.description, oo.price
  7       FROM online_order oo
  8       JOIN customer c ON oo.c_id = c.c_id
  9       WHERE c.name = v_customer_name
 10     ) LOOP
 11       -- Output the details of each online order
 12       DBMS_OUTPUT.PUT_LINE('Online Order ID: ' || order_rec.online_order_id || ', D
ate: ' || order_rec.the_date || ', Description: ' || order_rec.description || ', Price: '
|| order_rec.price);
 13     END LOOP;
 14   END;
 15   /
Online Order ID: 5006, Date: 26-DEC-22, Description: not applicable, Price: 400

PL/SQL procedure successfully completed.

```

Update and delete:

Query 1: Increase the salary of all home_staff by 2%.

SQL:

```

SET SERVEROUTPUT ON;
DECLARE
  v_increase_percentage NUMBER := 0.02; -- 2% increase
BEGIN
  UPDATE staff

```

```

SET salary = salary * (1 + v_increase_percentage)
WHERE id IN (
    SELECT id
    FROM home_staff
);

DBMS_OUTPUT.PUT_LINE('Salary of home staff increased by 2%.');

FOR staff_rec IN (
    SELECT id, name, salary
    FROM staff
    WHERE id IN (
        SELECT id
        FROM home_staff
    )
) LOOP
    DBMS_OUTPUT.PUT_LINE('ID: ' || staff_rec.id || ', Name: ' || staff_rec.name || ', New
Salary: ' || staff_rec.salary);
END LOOP;
END;
/

```

```

SQL> set serveroutput on;
SQL> DECLARE
2   v_increase_percentage NUMBER := 0.02; -- 2% increase
3 BEGIN
4   -- Update the salary of home staff by increasing it by 2%
5   UPDATE staff
6   SET salary = salary * (1 + v_increase_percentage)
7   WHERE id IN (
8     SELECT id
9     FROM home_staff
10  );
11
12   -- Output a message indicating the update was successful
13   DBMS_OUTPUT.PUT_LINE('Salary of home staff increased by 2%.');
14
15   -- Output the updated salary of each home staff member
16   FOR staff_rec IN (
17     SELECT id, name, salary
18     FROM staff
19     WHERE id IN (
20       SELECT id
21       FROM home_staff
22     )
23   ) LOOP
24     DBMS_OUTPUT.PUT_LINE('ID: ' || staff_rec.id || ', Name: ' || staff_rec.name |
| ', New Salary: ' || staff_rec.salary);
25   END LOOP;
26 END;
27 /
Salary of home staff increased by 2%.
ID: 10001, Name: taskin, New Salary: 10200
ID: 10002, Name: tawhid, New Salary: 10200
ID: 10003, Name: tahmid, New Salary: 10200
ID: 10004, Name: tamim, New Salary: 10200
ID: 10005, Name: tasin, New Salary: 10200
ID: 10006, Name: tahsan, New Salary: 10200
ID: 10007, Name: tarek, New Salary: 10200
PL/SQL procedure successfully completed.

```

Query 2: Increase all the food price by 5%

SQL:

SET SERVEROUTPUT ON;

DECLARE

 v_increase_percentage NUMBER := 0.05; -- 5% increase

BEGIN

 UPDATE catlog

 SET price = price * (1 + v_increase_percentage);

 DBMS_OUTPUT.PUT_LINE('All food prices increased by 5%.');

 -- Output the updated price of each food item

 FOR food_rec IN (SELECT name, price FROM catlog) LOOP

 DBMS_OUTPUT.PUT_LINE('Food: ' || food_rec.name || ', New Price: ' ||
food_rec.price);

 END LOOP;

END;

/

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
2   v_increase_percentage NUMBER := 0.05; -- 5% increase
3 BEGIN
4   -- Update the price of all food items by increasing it by 5%
5   UPDATE catlog
6   SET price = price * (1 + v_increase_percentage);
7
8   -- Output a message indicating the update was successful
9   DBMS_OUTPUT.PUT_LINE('All food prices increased by 5%.');
10
11  -- Output the updated price of each food item
12  FOR food_rec IN (SELECT name, price FROM catlog) LOOP
13    DBMS_OUTPUT.PUT_LINE('Food: ' || food_rec.name || ', New Price: ' || food_rec
14    .price);
15  END LOOP;
16 /
All food prices increased by 5%.
Food: berries, New Price: 126
Food: cold cereal, New Price: 137
Food: eggs, New Price: 53
Food: green tea, New Price: 116
Food: peanut butter, New Price: 158
Food: butter chicken rice, New Price: 126
Food: chicken biriyani, New Price: 137
Food: rice bowl, New Price: 84
Food: veg biriyani, New Price: 116
Food: plain rice, New Price: 74
Food: finger chicken, New Price: 126
Food: chicken grilled, New Price: 137
Food: chicken masala, New Price: 147
Food: coca-cola, New Price: 53
Food: mojo, New Price: 32
Food: sprite, New Price: 47
Food: fanta, New Price: 37
Food: 7up, New Price: 32
Food: pepsi, New Price: 53

PL/SQL procedure successfully completed.
```

Query 3:Deleting Customer Who Don't Have A Phone Number

SQL:

SET SERVEROUTPUT ON;

DECLARE

 v_count NUMBER;

BEGIN

 SELECT COUNT(*) INTO v_count

 FROM customer c

 WHERE NOT EXISTS (

 SELECT 1

 FROM customer_phone cp

 WHERE cp.c_id = c.c_id

);

IF v_count > 0 THEN

 DELETE FROM customer

```

WHERE NOT EXISTS (
SELECT 1
FROM customer_phone cp
WHERE cp.c_id = customer.c_id
);
DBMS_OUTPUT.PUT_LINE(v_count || ' customer(s) deleted.');
```

ELSE

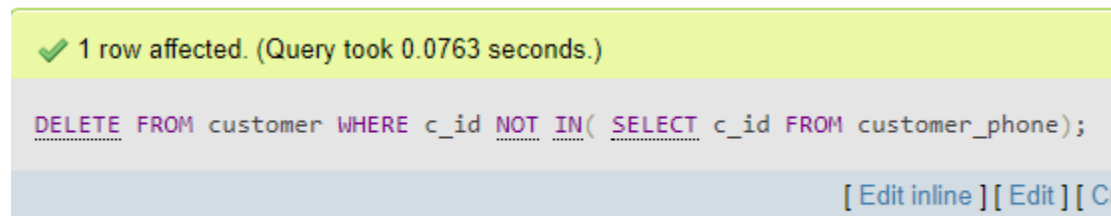
```

    DBMS_OUTPUT.PUT_LINE('No customers found without a phone number.');
```

END IF;

```

END;
/
```



✓ 1 row affected. (Query took 0.0763 seconds.)

```
DELETE FROM customer WHERE c_id NOT IN( SELECT c_id FROM customer_phone);
```

[\[Edit inline \]](#) [\[Edit \]](#) [\[Create \]](#)

View :

Here, the view is created for the receptionist. A receptionist must have access to the customer information such as name , c_id, order information such as offline order id, online order id , price of the foods.

SQL:

```

SET SERVEROUTPUT ON;
DECLARE
v_sql_statement VARCHAR2(1000);
BEGIN
v_sql_statement := 'CREATE VIEW receptionist1 AS
SELECT c.c_id, c.name, o.online_order_id, d.dine_order_id
FROM customer c
LEFT JOIN online_order o ON c.c_id = o.c_id
LEFT JOIN offline_order d ON c.c_id = d.c_id';
```

```
EXECUTE IMMEDIATE v_sql_statement;
```

```
DBMS_OUTPUT.PUT_LINE('View "receptionist1" created successfully.');
```

```
EXCEPTION
```

```
WHEN OTHERS THEN
```

```
DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
```

```
END;
```

```
/
```

```
Select * from receptionist1;
```



```

SQL> set serverout on
SQL> DECLARE
  2     v_sql_statement VARCHAR2(1000);
  3 BEGIN
  4     v_sql_statement := 'CREATE VIEW receptionist1 AS
  5                         SELECT c.c_id, c.name, o.online_order_id, d.dine_order_id
  6                         FROM customer c
  7                         LEFT JOIN online_order o ON c.c_id = o.c_id
  8                         LEFT JOIN offline_order d ON c.c_id = d.c_id';
  9
 10     EXECUTE IMMEDIATE v_sql_statement;
 11
 12     DBMS_OUTPUT.PUT_LINE('View "receptionist1" created successfully.');
```

```

13 EXCEPTION
14     WHEN OTHERS THEN
15         DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
16 END;
17 /
```

View "receptionist1" created successfully.

PL/SQL procedure successfully completed.

```
SQL> select * from receptionist1;
```

C_ID	NAME	ONLINE_ORDER_ID	DINE_ORDER_ID
505	Nikhad	5001	3004
505	Nikhad	5001	3005
501	Nuru	5002	3001
501	Nuru	5002	3006
502	Nabil	5003	3003
502	Nabil	5004	3003
506	Nokib	5005	3008
504	Nibir	5006	3002
504	Nibir	5006	3007
503	Nahid	5007	
501	Nuru	5011	3001
501	Nuru	5011	3006
502	Nabil	5012	3003
502	Nabil	5013	3003
503	Nahid	5014	
503	Nahid	5015	

16 rows selected.

1. Introduction

2. Project Overview

3. Database Schema

3.1 Object Types (how are the objects of the project)

3.2 Tables (how you created the tables)

3.3 Relationships (how the relationships are defined)

4. Implementation Details (you may add anything more/less here)

4.1 Procedures for CRUD Operations

4.2 Sample Queries

5. Codes (Attach PL/SQL codes and screenshots in each of the following section)

5.1 Object Types

5.2 Tables

5.3 Procedures

5.4 Queries to show all information

5.5 Queries to add information

(In the end Reset database, Drop Procedures)

6. Conclusion (summarize shortly what you learned)

This project embarked on a journey to unlock the potential of SQL for efficient restaurant management. We delved into the language, crafting a series of queries that addressed real-world restaurant scenarios. Through this exploration, we not only achieved our goals but also gained valuable insights into the power and versatility of SQL. Our code encompassed a diverse range of functionalities, including:

- **Views:** We constructed a custom view named `receptionist1`, granting receptionists access to crucial customer and order data tailored to their needs. This enhanced user experience and promoted efficient task management.
- **Targeted Queries:** We crafted a series of queries to address specific restaurant management needs. We retrieved customer information, calculated offline order counts for individual customers, identified high-earning delivery staff, and pinpointed online orders within a specific area exceeding a certain value threshold. These queries provided managers with actionable data to optimize operations and customer service.

- **Data Manipulation:** We delved into data manipulation techniques, leveraging the UPDATE statement to increase the salaries of home staff by a designated percentage. Furthermore, we employed the DELETE statement to eliminate customer records that lacked phone numbers, ensuring data integrity and facilitating accurate contact management.
- **Cursor Utilization:** We explored the power of cursors, iterating through query results and processing data effectively. This technique proved particularly valuable for handling large datasets.
- **Subquery Mastery:** We mastered the art of subqueries, incorporating them within SELECT and DELETE statements. Subqueries enabled us to define intricate data filtering criteria, resulting in more precise and efficient queries.

This project not only yielded tangible solutions for restaurant management but also equipped us with invaluable SQL expertise. We:

- **Sharpened Data Retrieval Skills:** We honed our ability to craft effective SELECT statements with joins, enabling us to extract relevant data from multiple tables seamlessly. This skill proves essential for generating comprehensive reports and analyzing trends.
- **Empowered Data Manipulation:** We gained proficiency in manipulating data through UPDATE and DELETE statements. This knowledge empowers us to maintain data accuracy, adjust values as needed, and keep the database clean.
- **Unlocked Cursor Potential:** We explored the practical applications of cursors, allowing us to manage and process large datasets efficiently.
- **Conquered Subqueries:** We mastered the art of subqueries, empowering us to construct complex filtering criteria within our queries, resulting in more targeted and efficient data retrieval.

By delving into these functionalities, we gained a profound understanding of how SQL interacts with relational databases. This project has not only enhanced our ability to manage restaurant data effectively but also equipped us with a versatile and powerful toolset for various database management tasks. Moving forward, we can confidently leverage this newfound expertise to tackle complex data challenges and empower efficient information retrieval and manipulation across diverse applications.

7. Responsibility collaboration

Project Work Responsibility (Sample Table for the last section, all team members' name and section that s/he worked in should be mentioned here)

Team Member	Task
Muhammad Sadman Tajwer	1. Database Design
Saurov Sikder	2. Report Writing
Asif Amir Noor	3. Implementation
Rahma Mahbub	4. ..
Hossain Sheikh	Implementation, crud operations and sample queries