School of Computer Engineering and Technology

T.Y. B. Tech

Year : 2020-2021 Trimester: I

DBMS Mini Project Report Submission

Mini Project Report on
“Meals on Wheels - Train food delivery management system”

Submitted by

Reuben George	ERP Number: 1032190592
Pranav Patil	ERP Number: 1032190731
Ankit Firke	ERP Number: 1032190741
Aditi Govindu	ERP Number: 1032190848

Under the guidance of

Prof. Varsha Naik

At

School of Computer Science Engineering and Technology

MIT World Peace University, Kothrud,

Pune 411038, India

2021-2022

Contents

S.no	**Title**			**Page no**
1	Introduction			8
	1.1	Motivation		9
	1.2	Objectives		10
	1.3	Functional requirements		10
	1.4	System data requirements		11
2	Problem statement			11
	2.1	System features		11
	2.2	System users		12
	2.3	System architecture		13
3	Tools and technologies			14
	3.1	Frontend developer tools		14
	3.2	Backend developer tools		16
4	Database design			17
5	Database schema			18
6	Database operations and commands			19
	6.1	Data Definition Language (DDL commands)		20
	6.2	Data Manipulation Language (DML commands)		27
	6.3	Data Control Language (DCL commands)		42
	6.4	PL SQL queries		49
		6.4.1	Procedures	50
		6.4.2	Functions	55
		6.4.3	Trigger	60
7	Frontend GUI screenshots			62
8	Conclusion			72
9	Future scope and evaluation			72
10	References			73

Abstract

In India, approximately 22.15 million passengers use the railway network daily. Given the current pandemic situation, IRCTC or the Indian railways corporation has decided to discontinue the use of pantry cars. This forces passengers to bring pre-cooked meals on board or purchase meals at stations. Meals on long distance, express and night trains are difficult to purchase, especially for passengers travelling with infants or the elderly. Many people also have concerns about hygiene of food at railway stations. Thus, we have come up with the Meals on Wheels solution. Through our system - Meals on Wheels - we aim to bring passengers and restaurants closer to each other. Passengers can choose to buy meals on the go and be assured of good quality, delicious food from their favourite restaurants. This project focuses on the database operations and the frontend technology user can interact with to book a healthy meal.

List of abbreviations

Serial no	Acronym	Full form
1	IRCTC	Indian Railway Catering and Tourism Corporation
2	DDL	Data Definition Language
3	DML	Data Manipulation Language
4	DCL	Data Control Language
5	GUI	Graphic User Interface
6	CoD	Cash on Delivery
7	PIL	Python Imaging Library
8	PoC	Proof of Concept
9	MySQL	My + Structured Query Language
10	RDBMS	Relational Database Management System
11	PL SQL	Procedural Structured Language
12	ERD	Entity Relationship Diagram
13	NF	Normal form
14	PK	Primary key
15	FK	Foreign key
16	CRUD	Create Read Update and Delete operations
17	API	Application Programming Interface

List of figures

List of tables

Introduction

The Indian pantry car or a bogey reserved to cook meals on trains was introduced by the British in 1800s in India [1]. In the recent years, the drop in quality of food provided by IRCTC and accidental gas leaks on pantry cars forced the early retirement of the meals-on-wheels concept from trains. This not only symbolized a loss of tradition for all travellers fond of train meals, but posed a real challenge to passengers on long distance trains. Elderly passengers, people traveling with infants on night trains or superfast trains.

In 2015, IRCTC proposed the launch of an e-catering site to keep up the development of technology. The launch of their e-catering site[2] with over 500+ restaurants on the station was welcomed by all Indians. Fresh, hygienic food was provided to all passengers by collaborating with food industry giants, Dominos, Rajdhani and indigenous caterers, near the station. Some of their major partners have been illustrated in figure 1.1

Through this mini project called Meals on Wheels, we aim to develop a small-scale version of the IRCTC's catering website. This project aims to provide a simple GUI to help even a novice use it. Keeping covid-19 protocols in mind, we enable user to view person who cooked and delivered the meal's temperature and vaccination status.

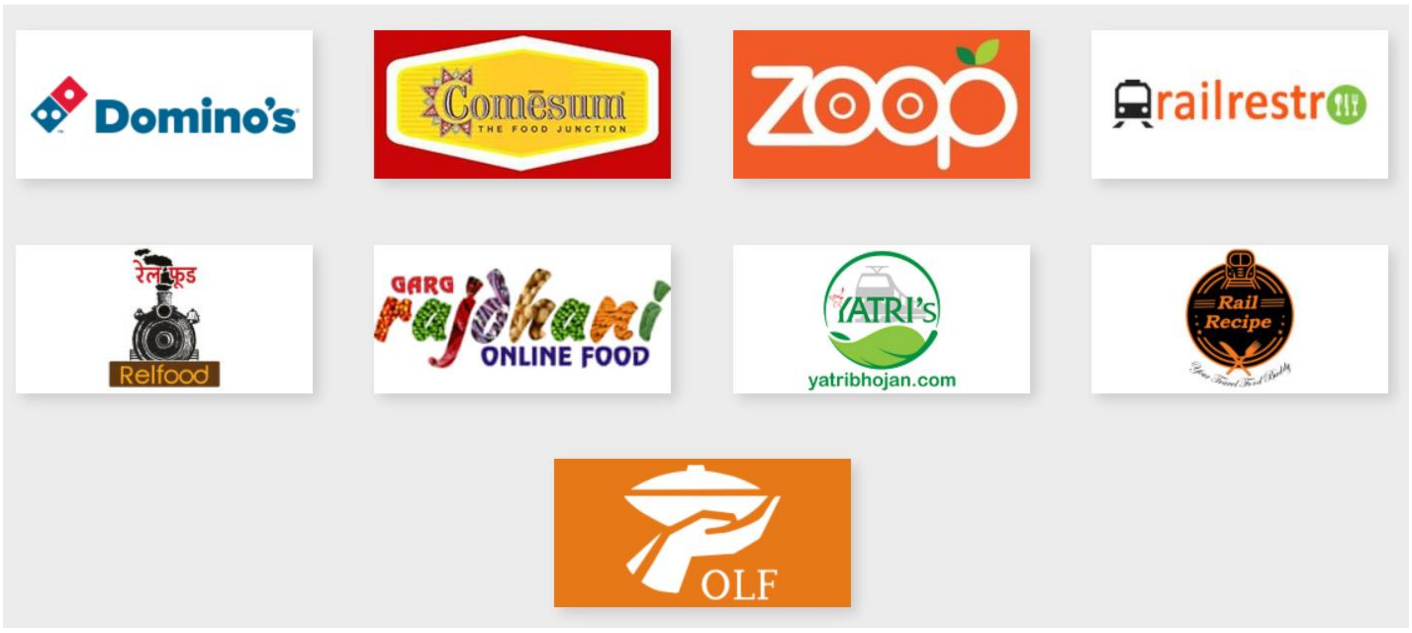


Figure 1.1: IRCTC e-catering food partners

1.1 Motivation

The aims of this project are:

- Enable users, admins and kitchens to use a single platform for all their food booking, delivery and payment operations.
- Meals on Wheels will provide a comfortable and reliable food delivery system for passengers on night trains, elderly passengers and families traveling with infants or disabled people.
- Allow customers to book healthy and hygienic meals from their favourite food partners, on the go.

1.2 Objectives

Through this mini project, we wish to:

- Provide a simple platform for booking of meals for customers
- Enable kitchens to track their income and cooks
- Admins manage the Meals on Wheels system and provide valuable customer feedback to concerned kitchens
- Learn about Python and MySQL programming for development of an end-to-end management system

1.3 Functional requirements

System allows the passenger to enter data and admin to view and modify data on system:

- Insert passenger details and add to database
- Admin can update or delete information in database

1.4 System data requirements

Table 1.1: PL SQL blocks used in system

Name of data structure used	Purpose	Requirements
Functions		
1. Generate Payment ID	Takes meal id as input and generates count of payments done for that meal id. It also displays information about cook, meal and kitchen	Input: Meal id Output: Count of payments done for that meal
2. Display delivery boy name	Takes passenger name as input and displays name of delivery boy who will deliver meal to them	Input: passenger name Output: delivery boy name
3. Track delivery	Function to enable user or admin to call delivery boy and check status of given meal id	Input: meal id Output: Delivery boy phone number
4. Count meals	Counts no. of meals ordered from a given kitchen	Input: kitchen id Output: Count of meals
Procedures		
1. Display cooks and kitchen details	Procedure is used to display name of cook and the kitchen they work at, when input parameter is kitchen id. It also retrieves count of cooks at given kitchen id	Input: kitchen id Output: No. of cooks working at that kitchen and their details
2. Display meal details	Procedure to display meal details and kitchen that cooked it	Input: Meal id Output: Meal and kitchen details
Trigger		
1. Backup of delivered meals	Trigger will move meals deleted from meals into another tables. This indicates a delivery has been done	Input: After delete on meal table Output: New record in deliveries_done

Problem statement

The Meals on Wheels system allows a user who wants to buy a meal. Passengers must provide their name, email id, phone number, 9-digit PRN number of the train and seat numbers for delivery. Meals can be chosen from a list of kitchens on the site. Individual and group bookings are allowed through 1 account. Once a meal is booked, customers will be allowed to pay online or cash on delivery (CoD) at the station. Tracking order ID will be provided to see which station meal will be delivered, estimated time of arrival and status of meal. Each kitchen will have cooks and a set of meals they can make for a day. Each cook will have a cook id, name, vaccination status and temperature check taken daily. Once food is prepared by the requested kitchen, the delivery system will be notified. Delivery person with specific id, name and contact number will travel to the station. Once the train is located, food will be delivered to the seat number specified by the passenger and CoD is collected. System has an admin to handle meal requests, allergies, preferences, special requests and quality complaints. Customers can rate or complain about the delivery or food and the system admin can see these comments (not customer name). Kitchen gets this feedback to improve their quality.

2.1 System features

Meals on Wheels system has the following unique attributes:

- Customers can book a meal using their 9-digit PRN train number and email address, without registering to the system
- Users are provided a tracking number after paying for their meal to help them track status of their meal
- Customers can view the data kitchens enter their cook health – temperature, vaccination status – when meal is delivered.
- Instead of delivery boy waiting at kitchen, kitchen provides update of meal status to system and admin can relay this information to the delivery system. Thus, when meal is at “prepared” stage, the delivery boy can come for pickup.
- System allows customers to pay by cash or card or UPI to the system. Admin will then send money to the kitchens, at the end of a week or month, as decided by contract between them.

2.2 System users

Table 2.1: Users of Meals on Wheels and their attributes

Entity	**Attributes**
Passenger / Customer	Name (VARCHAR) Phone number (INT) Email id (VARCHAR) 9-digit PRN number of the train (INT) Seat numbers for delivery (INT) Order no (INT) **Optional / Future scope:** Apply coupon (VARCHAR) Medicine request (VARCHAR)
Kitchen	Name (VARCHAR) Location (Multivalued, VARCHAR) Kitchen-ID (INT)
Meal	Passenger order no (INT) from passenger Meal-ID (VARCHAR) Type (VARCHAR) Cost (FLOAT) Quantity (FLOAT) **Optional / Future scope:** Allergies/ Special requests (VARCHAR)
Cook	Cook ID (INT) Name (VARCHAR) Kitchen working for (VARCHAR) Vaccination status (CHAR Y/N) Temperature of the day (FLOAT)
Delivery	Id (INT) Name (VARCHAR) Contact number (INT) Estimated time of delivery (DATE) Train number (INT) from passenger Seat no (INT) from passenger Station (VARCHAR)
Admin	Admin ID(INT) Admin name (VARCHAR) Contact number (INT)
Payment	Payment ID(INT) Amount (FLOAT) Mode - Cash, Card, UPI (VARCHAR) Date of payment (DATE) Kitchen name (VARCHAR) Feedback (FLOAT)

2.3 System architecture

Our proposed system will be built on Python frontend and MySQL database for backend. Figure 2.1 depicts the proposed system architecture.

Delivery partners, passengers and kitchens use system GUI to login. Based on their clearance, they can view relevant data. For instance, passenger can login and book a meal, pay for the meal, track the meal and give valuable feedback. System provides functionality to insert user input, like customer name and personal data or kitchens new cook temperature reading or change in delivery partner information to the database.

The frontend connects to webpage to system operations through requisite API calls, that is direct connection between system and operations is not allowed. All queries verified by the API will be executed. The frontend uses MySQL connector library to connect and query the MySQL Meals on Wheels database, to which the database provides a response. The admin can perform CRUD operations on database through direct connection to the backend servers.

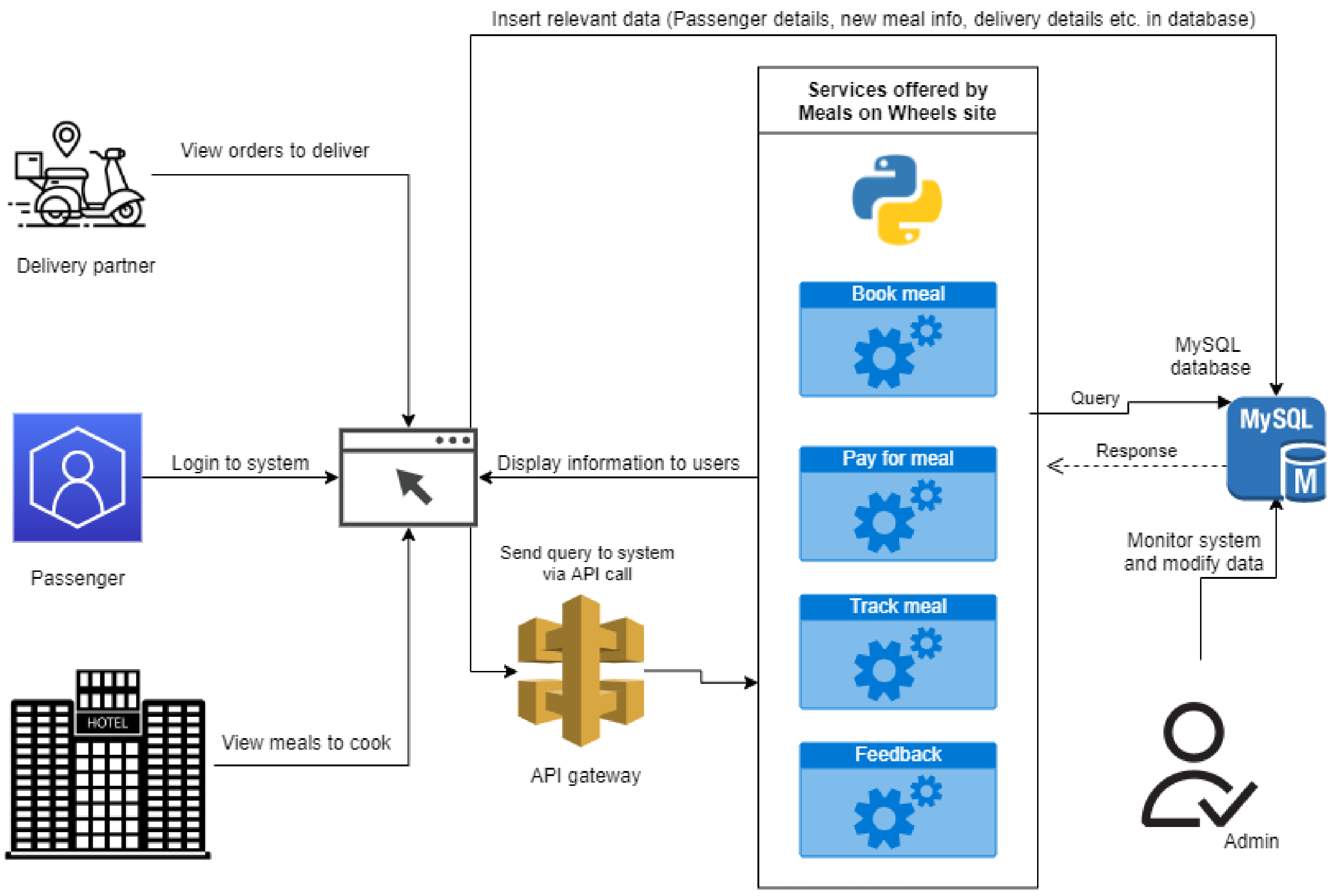

Insert relevant data (Passenger details, new meal info, delivery details etc. in database)
View orders to deliver
Delivery partner
Services offered by Meals on Wheels site
Book meal
Pay for meal
Track meal
Feedback
MySQL database
Query
Response
MySQL
Login to system
Display information to users
Passenger
Send query to system via API call
API gateway
Monitor system and modify data
HOTEL
View meals to cook
Admin

Figure 2.1: System architecture

Tools and technologies

To develop our Meals on Wheels system, we have used python and streamlit for frontend. MySQL is used for backend development and storage of data, processing etc.

3.1 Frontend development tools

Python

Python is a high level, interpreted language developed by Guido Van Rossum in 1991[3]. It offers a myriad of programming constructs, documentation and easy deployment on the web. We chose python for our frontend development [4] because:

- It is ideal for beginners to use and develop simple apps, as syntax and keywords are similar to English language
- Asynchronous coding, where every line of code is independent from the other, so debugging is easy
- Python offers a variety of libraries such as PIL, matplotlib and seaborn for eye catching visualizations
- Support for Django, Flask and streamlit for easy web app development
- Python web apps are portable and interactive. Testing is easy to perform.
- Easy integration with MySQL backend using requisite import libraries
- Quick deployment of apps compared to Java and C++, enabling users to develop start-up code and prototypes for a PoC easily.

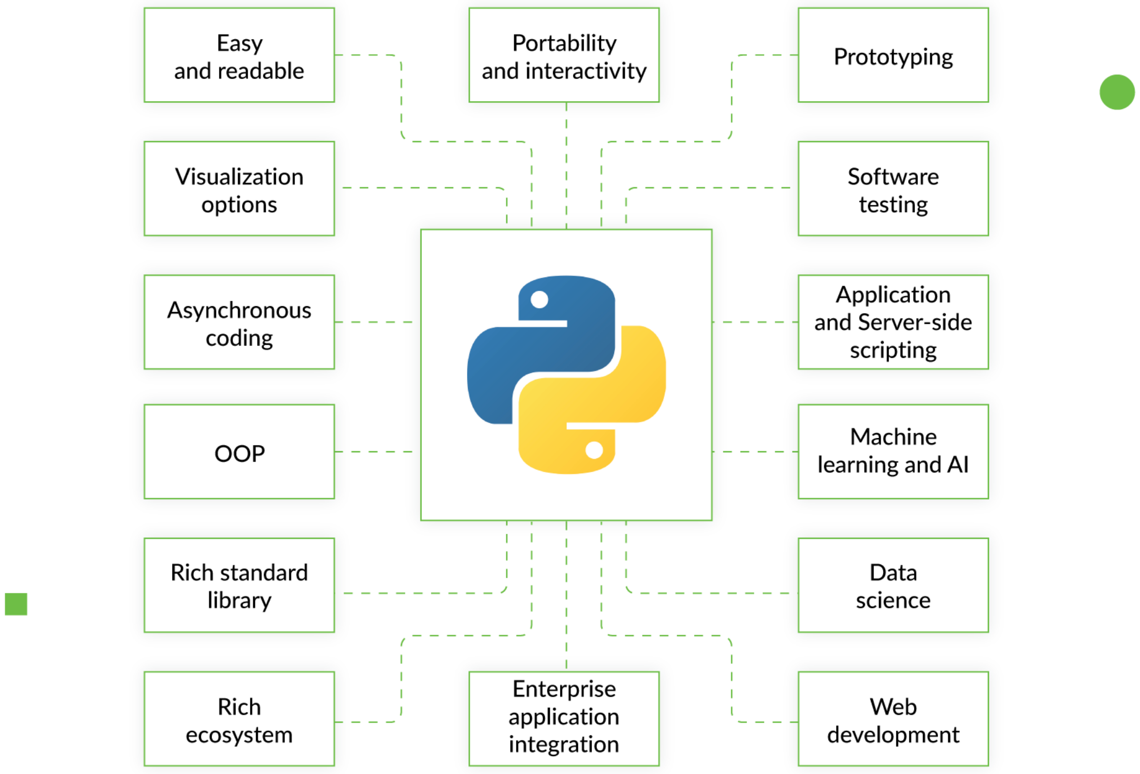


Figure 3.1: Advantages of using python for web development

Streamlit

Streamlit [5] for python is an open-source framework for building web apps quickly and efficiently. Unlike Flask and Django, streamlit allows developers to write code for web apps the same way as a python file or.py, as streamlit provides a library to run the code on the web. Any change in source code can be viewed in the website, by selecting the rerun option. Streamlit offers .cache method to cache or store databases to enable the site to run faster.

Figure 3.2: Streamlit logo

3.2 Backend development tools

MySQL

MySQL [6] is an open-source RDBMS tool developed in C/C++ by Michael Widenius in 1995. It is ideal a large range of projects from small prototypes to large data warehouses. We have used MySQL because of the following reasons:

- Effective database management and connection to python frontend
- Portable and seamless connectivity using MySQL connector library in python
- Rapid development and round the clock availability of this system. Since trains are available 24/7, our system should allow food deliveries at any time and MySQL ensures system availability
- MySQL can be deployed on the cloud with minimal modifications to the system
- Security of data, especially sensitive payment information is guaranteed by MySQL DCL commands
- Data entered by the user in Meals on Wheels system is structured and can be queried easily using PL SQL support in MySQL.

Steps involved in connecting Python frontend to MySQL database [7] have been depicted in image 3.3 below

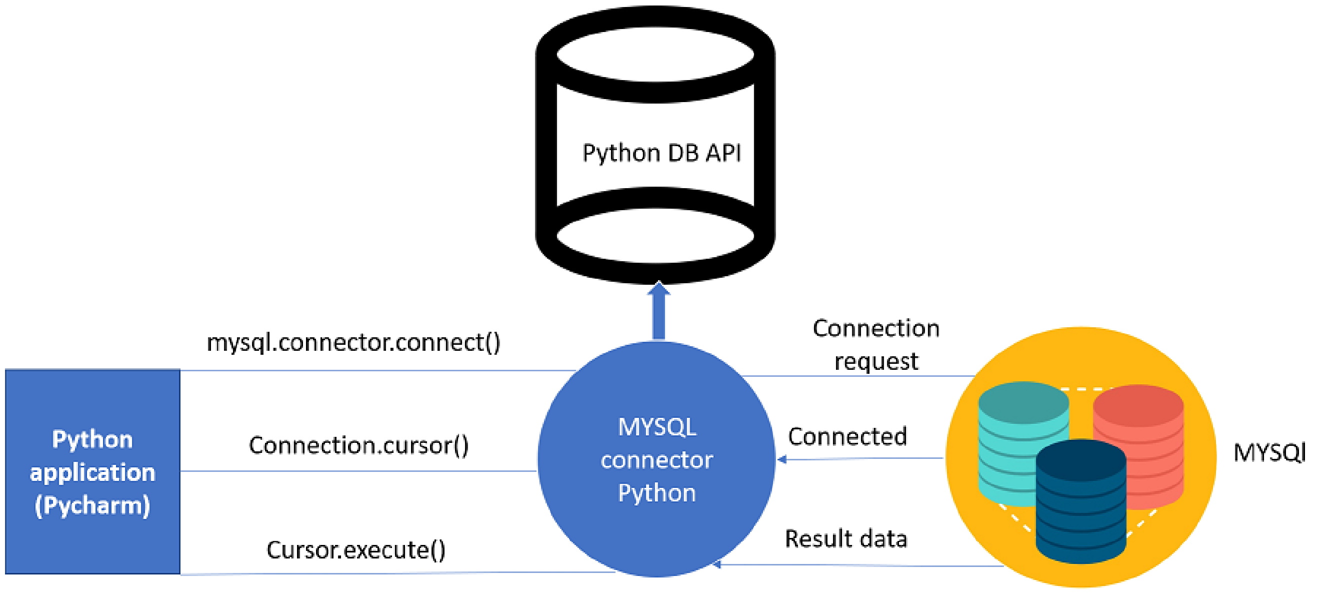


Figure 3.3: Connecting Python to MySQL using MySQL connector library

4. Database design

Figure 4.1 illustrates the ERD for Meals on Wheels system. It focuses on 6 entities – Passenger, Admin, Kitchen, Cook, Payment and Delivery. Diagram uses the Peter Chen [8] notation to represent entities and attributes.

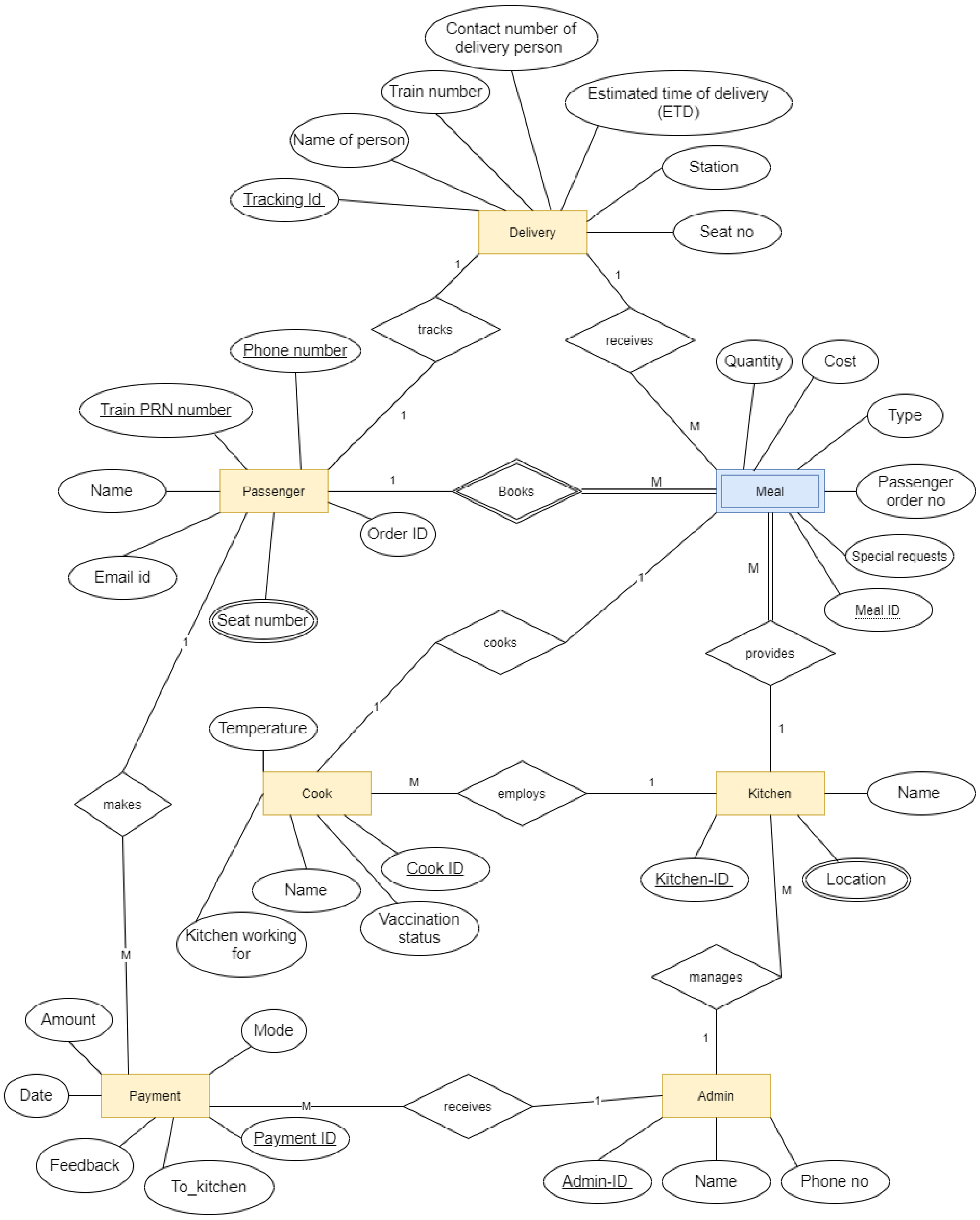


Figure 4.1: ERD for Meals on Wheels system

5. Database schema

Normalization in RDBMS is essential to reduce data redundancy between a set of tables. We have 3 levels of normalization – 1NF, 2NF, 3NF.

First normal form or 1NF

No multivalued attributes exist in the table. Intersection of 1 row and column in the table to form 1 cell, must contain single value. For example, table kitchen can have multiple values for location. We cannot insert multiple values in 1 cell, so new table called kitchen location with kitchen id and location has to be created.

Second normal form or 2NF

A relation in 1NF can be reduced to 2NF by eliminating partial dependencies. A table having attribute dependent on primary key (PK) as well as another non primary attribute (foreign key or FK) is said to be in 2NF form. For example, in payment table payment id along with passenger id determine uniqueness in table.

Third normal form or 3NF

A 2NF table with no transitive dependencies among the FK and PK attributes is said to be in 3NF. All records obey the rule of X -> Y where X is super key and Y is primary attribute in table. For example, meal id depends on passenger email id and delivery boy name depends on meal id assigned. To reduce this transitive dependency, passenger email id is sent as foreign key in meal table. Therefore, only meal id for that passenger email id is transferred to delivery in next stage. Thus, delivery table is in 3NF.

Table 5.1: Normalized form of tables

Table name	**Normal form**
Passenger	3NF
Seat no	3NF
Payment	3NF
Admin	2NF
Kitchen	3NF
Kitchen location	3NF
Delivery	3NF
Cook	3NF
Meal	2NF

Database schema refers to the process of conversion of an entity relationship diagram to a formal table layout. It ensures tables are reduced to the most appropriate normal forms. Schema represents the blueprint of data and a logical understanding of the tables in the database. The logical schema for Meals on Wheels system has been illustrated in figure 5.1, below.

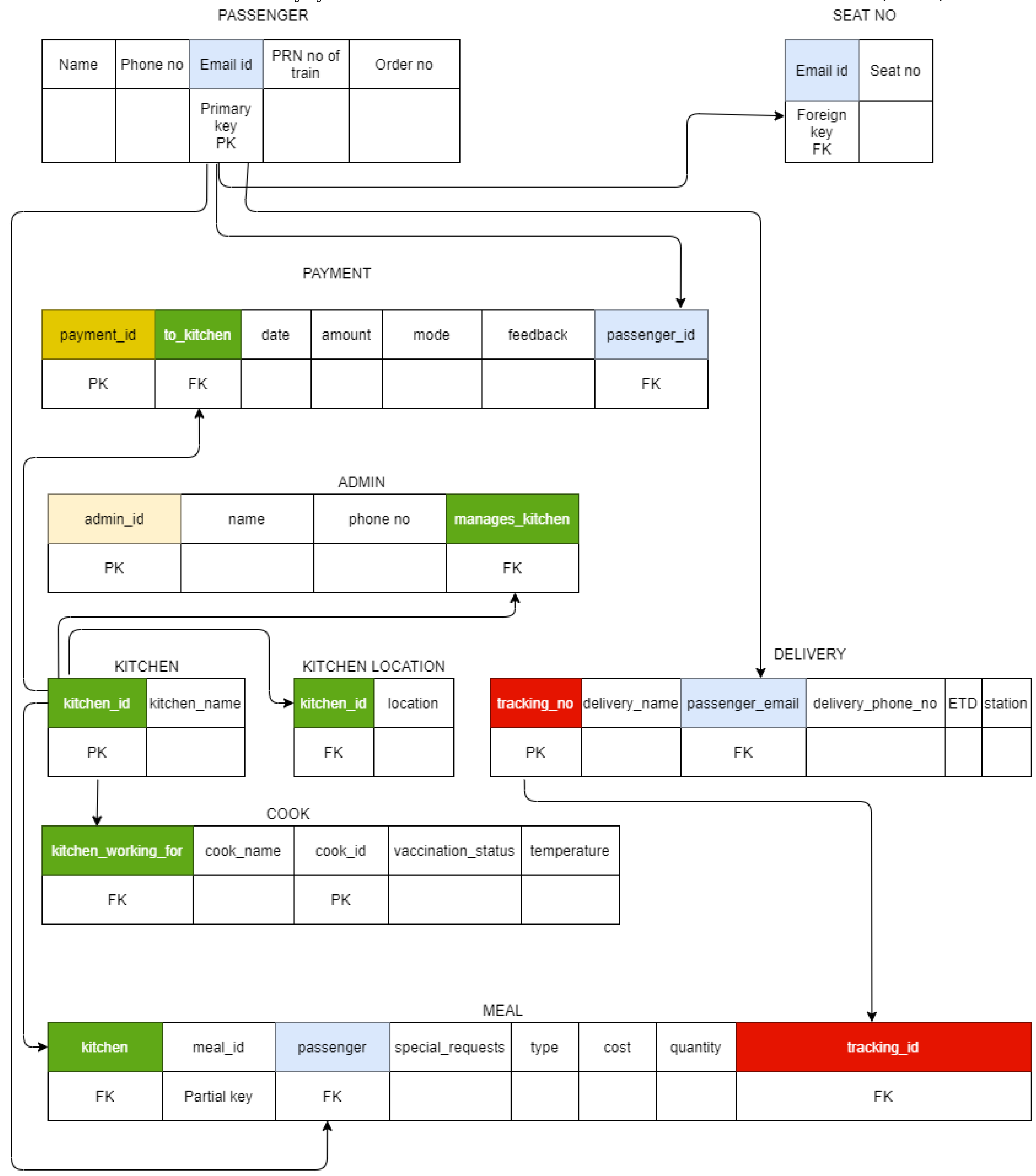


Figure 5.1: Logical schema for Meals on Wheels system

6. Database operations and commands

To realize the database depicted in figure 5.1, the following DDL, DML and DCL commands [9] have been used.

DDL or data definition commands are used in MySQL to define the attributes, constraints and features of a table in the database. Commands like CREATE, ALTER, RENAME etc. and constraints like NOT NULL, UNIQUE can be used to define the restrictions on data stored in the table.

DML or data manipulation commands are used to modify the data in a table. Commands like INSERT are used to add new records, UPDATE to change values in these records and DELETE to remove data. PL SQL blocks of code, such as trigger and procedures can be considered under this category, as they change values in the database schema. Functions defined by the user or inbuilt functions can be implemented using the CALL command.

DCL or data control language commands are used to restrict access on database created. They deal with the rights and permissions granted by database administrator to users. Privileges are granted by the GRANT and removed by the REVOKE commands.

6.1 Data Definition Language (DDL commands)

<u>Create database and tables before viewing tables</u>

```
CREATE DATABASE mealsonwheels;

USE mealsonwheels;

CREATE TABLE passenger(

Email_ID VARCHAR(30) PRIMARY KEY,

Name VARCHAR(15),

Train_PRN INT(10),

Phone INT(10),

Order_ID INT UNIQUE);

CREATE TABLE seating(

P_EmailID VARCHAR(30),

Seat_Num INT);

CREATE TABLE delivery(

Tracking_ID INT PRIMARY KEY,

D_Name VARCHAR(15),

Pass_Email VARCHAR(30),

D_Phone INT,

ETA DATETIME,

Station VARCHAR(20));
```

```
CREATE TABLE cook(

Cook_ID INT PRIMARY KEY,
Name VARCHAR(20),
Vac_Status Varchar(12),
Temp_celcius DOUBLE,
K_ID INT);
CREATE TABLE meal(
Meal_ID INT,
Special_req VARCHAR(30),
M_Type VARCHAR(10),
Cost FLOAT,
Quantity INT);
ALTER TABLE meal ADD COLUMN K_ID INT;
ALTER TABLE meal ADD COLUMN Track_ID INT;
CREATE TABLE kitchen(
Kitchen_ID INT PRIMARY KEY,
Name VARCHAR(20));
CREATE TABLE k_loc(
KI_ID INT,
K_location VARCHAR(20));
CREATE TABLE admin(
A_ID INT PRIMARY KEY,
A_name VARCHAR(20),
A_Phone INT);

ALTER TABLE admin ADD COLUMN manage_k_id INT;
CREATE TABLE payment(
P_ID INT PRIMARY KEY,
P_mode VARCHAR(15),
K_info INT,
P_date DATE,
Amount FLOAT,
Feedback VARCHAR(50));


ALTER TABLE seating
ADD FOREIGN KEY (P_EmailID)
REFERENCES passenger(Email_ID);
ALTER TABLE delivery
ADD FOREIGN KEY (Pass_Email)
REFERENCES passenger(Email_ID);
ALTER TABLE meal
ADD FOREIGN KEY (Meal_ID)
REFERENCES passenger(Order_ID);
ALTER TABLE cook
ADD FOREIGN KEY (K_ID)
REFERENCES kitchen(Kitchen_ID);
ALTER TABLE meal
ADD FOREIGN KEY (K_ID)
REFERENCES kitchen(Kitchen_ID);
ALTER TABLE k_loc
ADD FOREIGN KEY (KI_ID)
REFERENCES kitchen(Kitchen_ID);
ALTER TABLE admin
ADD FOREIGN KEY (manage_k_id)
REFERENCES kitchen(Kitchen_ID);
ALTER TABLE meal
ADD FOREIGN KEY (Track_ID)
REFERENCES delivery(Tracking_ID);
ALTER TABLE payment
ADD FOREIGN KEY (K_info)
REFERENCES kitchen(Kitchen_ID);
ALTER TABLE kitchen ADD type VARCHAR(10);
SHOW TABLES;
DESC admin;
DESC cook;
```

DESC delivery;

DESC k_loc;

DESC kitchen;

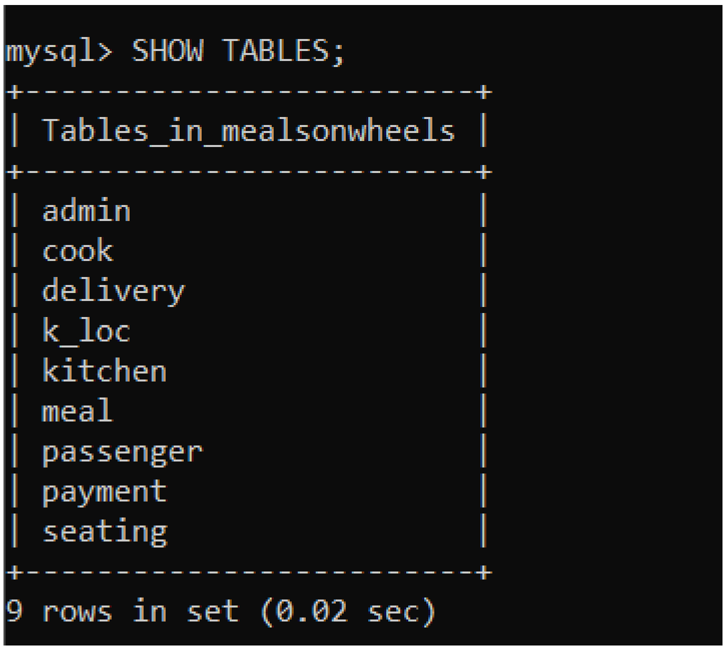
DESC meal;

DESC passenger;

DESC payment;

DESC seating;

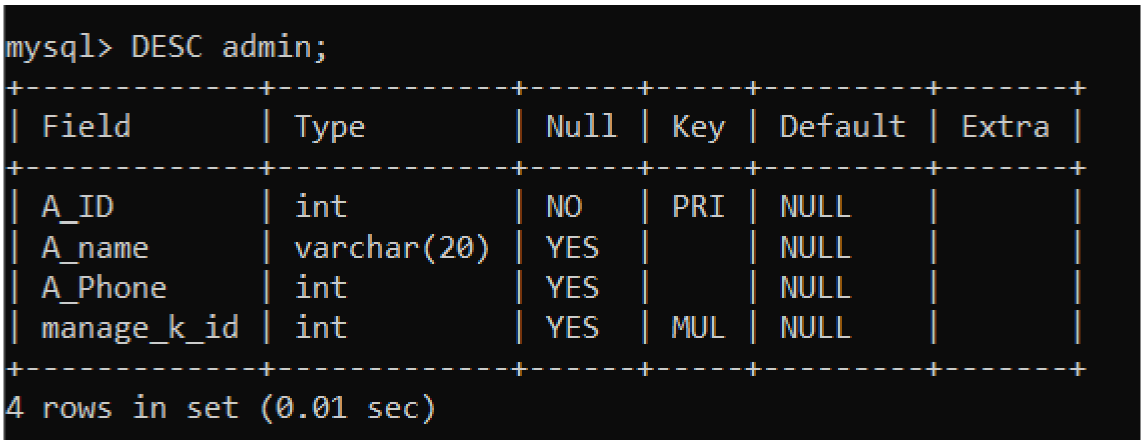
Tables created have been shown in figure 6.1



```
mysql> SHOW TABLES;
+------------------------+
| Tables_in_mealsonwheels |
+------------------------+
| admin                  |
| cook                   |
| delivery               |
| k_loc                  |
| kitchen                |
| meal                   |
| passenger              |
| payment                |
| seating                |
+------------------------+
9 rows in set (0.02 sec)
```

Figure 6.1: Tables in database - Meals on Wheels

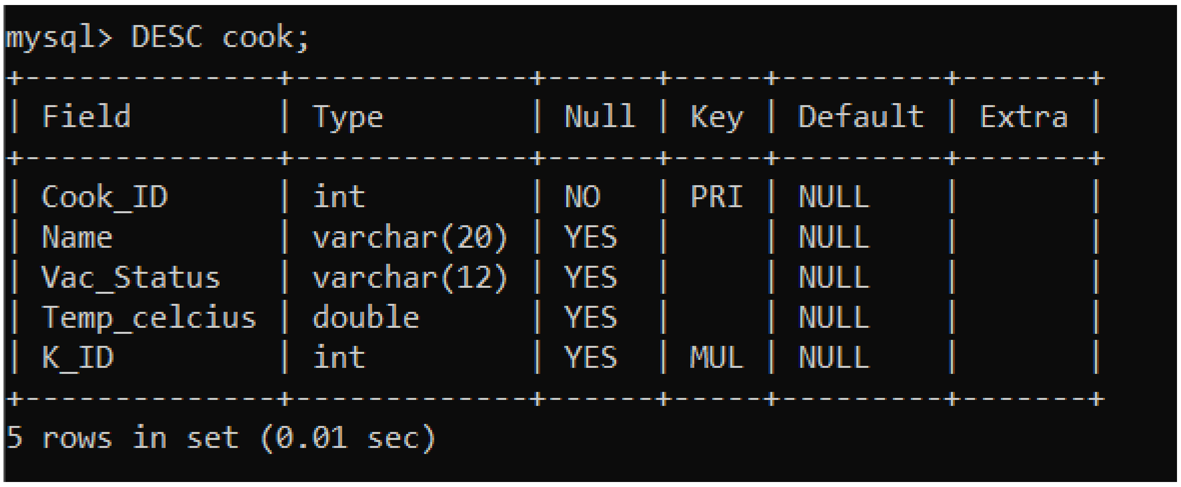
Columns in admin table – admin identification number, admin name, admin phone number and ids of kitchens they handle have been have been illustrated in figure 6.2



```
mysql> DESC admin;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| A_ID        | int         | NO   | PRI | NULL    |       |
| A_name      | varchar(20) | YES  |     | NULL    |       |
| A_Phone     | int         | YES  |     | NULL    |       |
| manage_k_id | int         | YES  | MUL | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Figure 6.2: Columns in admin table

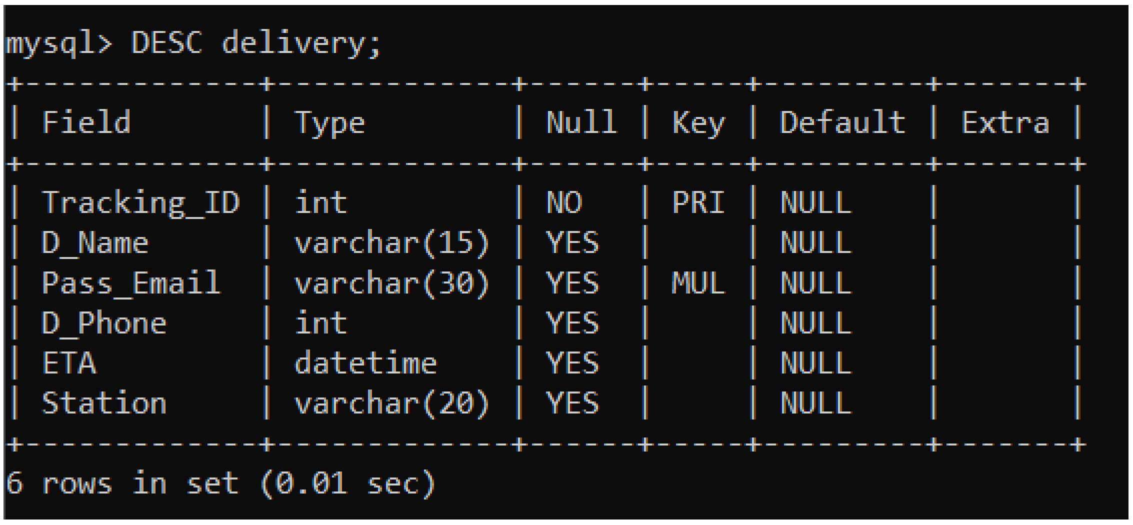
Columns in cook table – cook identification number, name of cook, vaccination status as total or partial, temperature in Celsius for that day and kitchen id they work for are depicted in figure 6.3 below



```
mysql> DESC cook;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| Cook_ID      | int         | NO   | PRI | NULL    |       |
| Name         | varchar(20) | YES  |     | NULL    |       |
| Vac_Status   | varchar(12) | YES  |     | NULL    |       |
| Temp_celcius | double      | YES  |     | NULL    |       |
| K_ID         | int         | YES  | MUL | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Figure 6.3: Columns in cook table

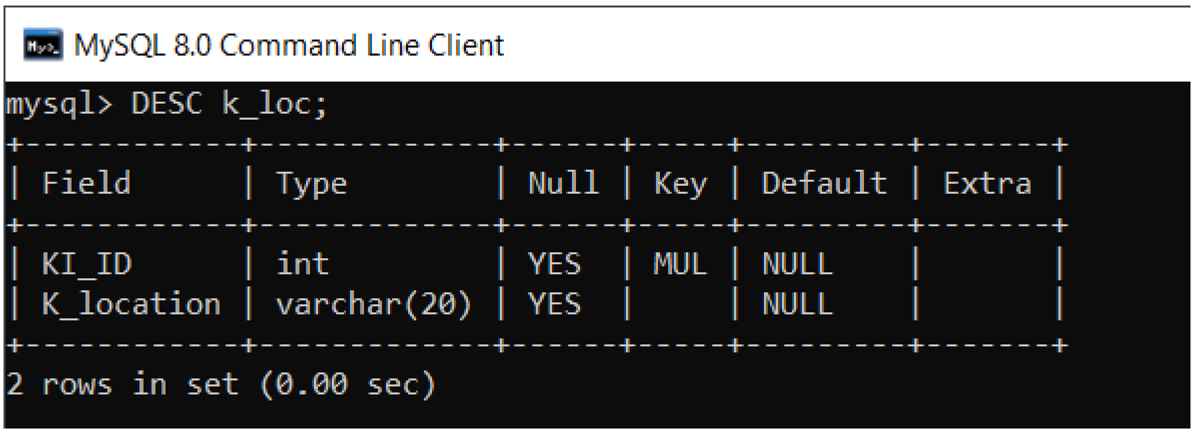
Columns in delivery table – tracking id of order, delivery boy name, passenger to deliver to email id, delivery person's phone number, estimated time of delivery and station are shown in figure 6.4



```
mysql> DESC delivery;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Tracking_ID | int         | NO   | PRI | NULL    |       |
| D_Name      | varchar(15) | YES  |     | NULL    |       |
| Pass_Email  | varchar(30) | YES  | MUL | NULL    |       |
| D_Phone     | int         | YES  |     | NULL    |       |
| ETA         | datetime    | YES  |     | NULL    |       |
| Station     | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

Figure 6.4: Columns in delivery table

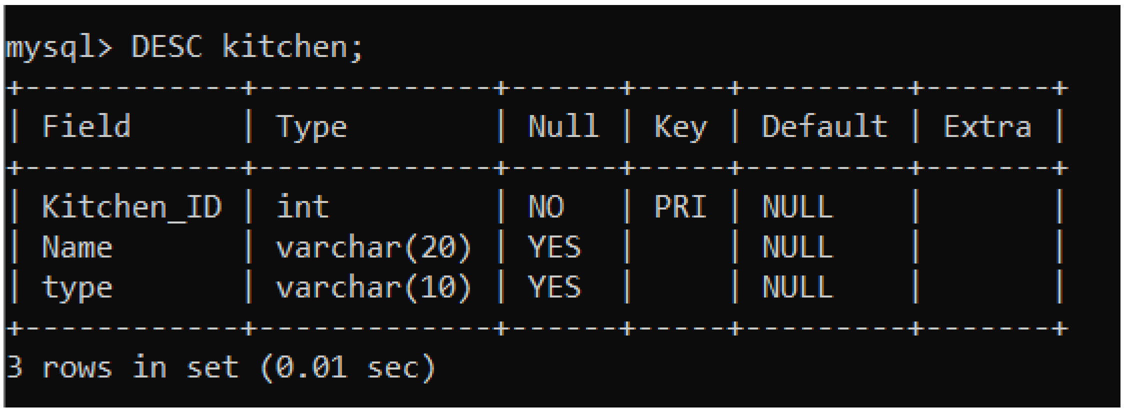
Attributes in multivalued attribute kitchen locations – kitchen id and location are represented in table k_loc under figure 6.5



```
mysql> DESC k_loc;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| KI_ID      | int         | YES  | MUL | NULL    |       |
| K_location | varchar(20) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Figure 6.5: Columns in k_loc table

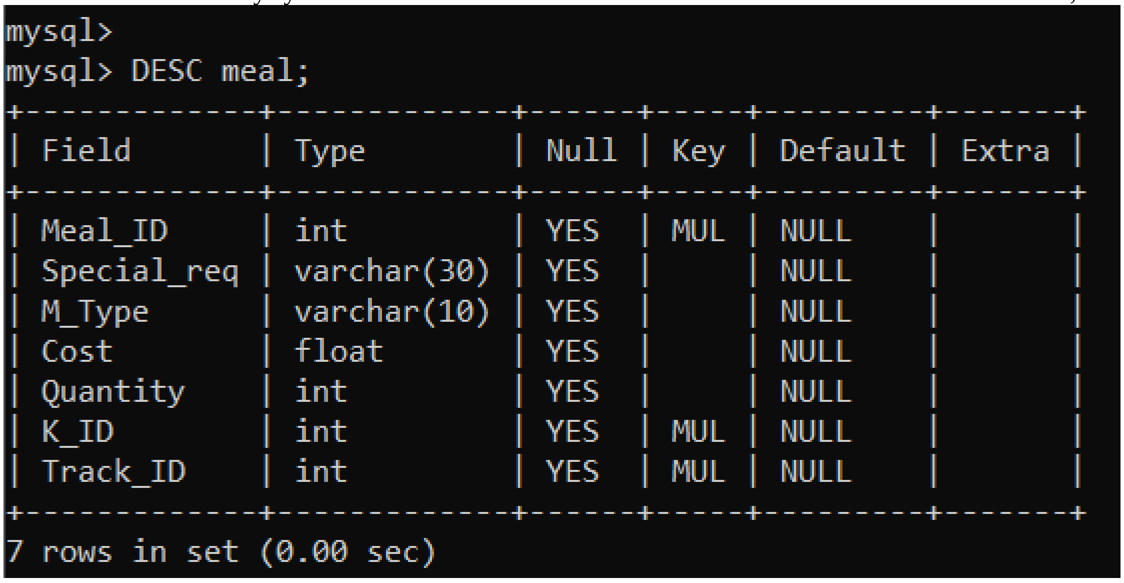
Columns in kitchen table – kitchen id as primary key, name of kitchen and type of meals they serve (veg, non veg, vegan or more than 1 indicated by mixed) are illustrated in figure 6.6



```
mysql> DESC kitchen;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Kitchen_ID | int         | NO   | PRI | NULL    |       |
| Name       | varchar(20) | YES  |     | NULL    |       |
| type       | varchar(10) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

Figure 6.6: Columns in kitchen table

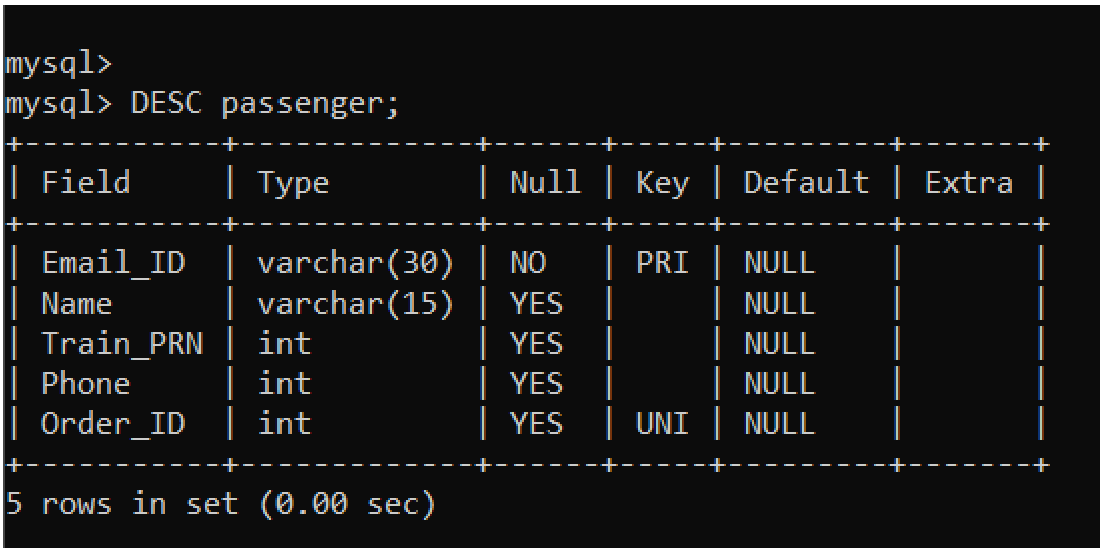
Attributes of meal table – meal id, special requests for Jain food or extra toppings are NULL by default, meal type can be veg, non veg or vegan, cost of meal is a fixed float number, quantity or no. of meals booked, kitchen id of kitchen making that meal and tracking id of order are given in figure 6.7



```
mysql>
mysql> DESC meal;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Meal_ID     | int         | YES  | MUL | NULL    |       |
| Special_req | varchar(30) | YES  |     | NULL    |       |
| M_Type      | varchar(10) | YES  |     | NULL    |       |
| Cost        | float       | YES  |     | NULL    |       |
| Quantity    | int         | YES  |     | NULL    |       |
| K_ID        | int         | YES  | MUL | NULL    |       |
| Track_ID    | int         | YES  | MUL | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Figure 6.7: Columns in meal table

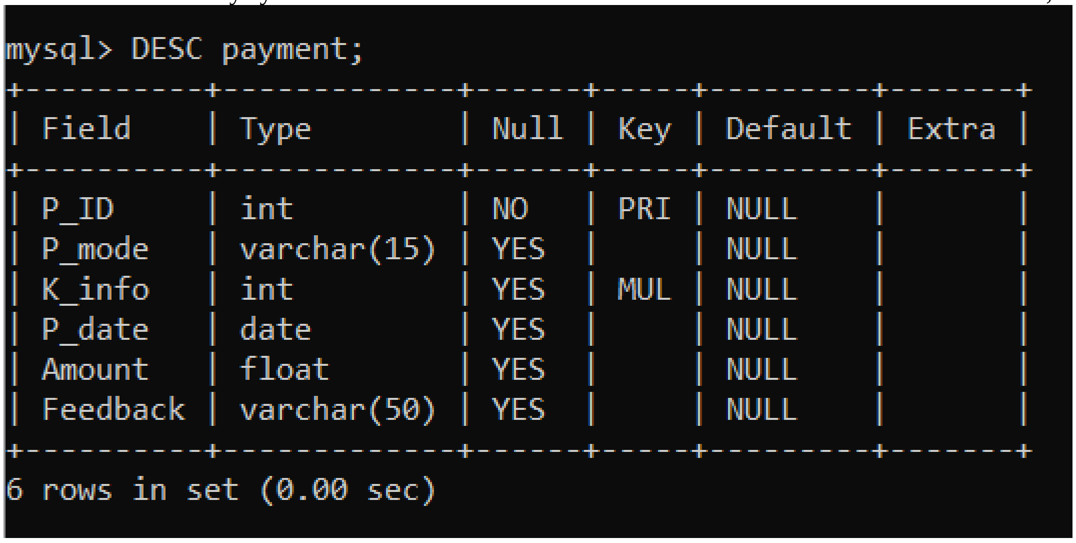
Passenger table columns – passenger email id as primary key, name of passenger train identification PRN number, phone number to call and unique order id are shown in figure 6.8



```
mysql>
mysql> DESC passenger;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Email_ID  | varchar(30) | NO   | PRI | NULL    |       |
| Name      | varchar(15) | YES  |     | NULL    |       |
| Train_PRN | int         | YES  |     | NULL    |       |
| Phone     | int         | YES  |     | NULL    |       |
| Order_ID  | int         | YES  | UNI | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Figure 6.8: Columns in passenger table

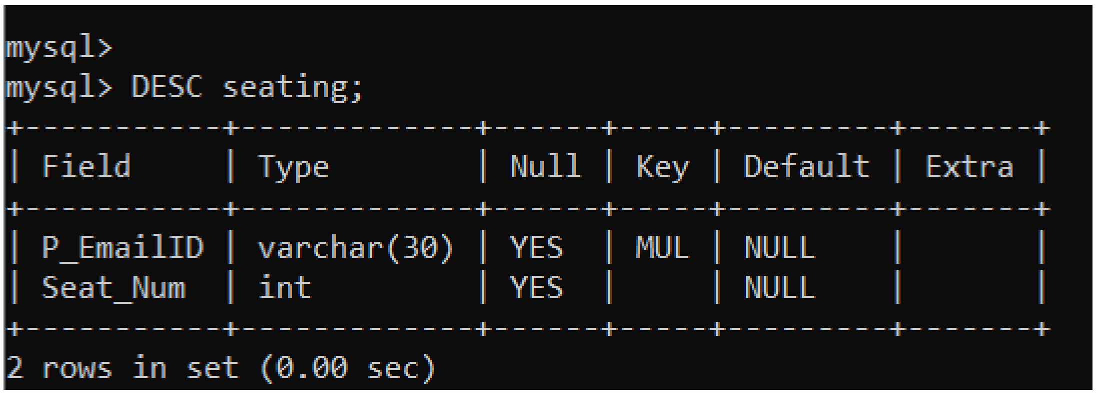
Attributes of payment table are payment id, payment mode can be cash or card or UPI, kitchen to which they paid is k_info or kitchen id, payment date, amount paid and feedback they wish to give kitchen is NULL by default. These are depicted in figure 6.9



```
mysql> DESC payment;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| P_ID     | int         | NO   | PRI | NULL    |       |
| P_mode   | varchar(15) | YES  |     | NULL    |       |
| K_info   | int         | YES  | MUL | NULL    |       |
| P_date   | date        | YES  |     | NULL    |       |
| Amount   | float       | YES  |     | NULL    |       |
| Feedback | varchar(50) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Figure 6.9: Columns in payment table

Seating table columns are shown in figure 6.10



```
mysql>
mysql> DESC seating;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| P_EmailID | varchar(30) | YES  | MUL | NULL    |       |
| Seat_Num  | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Figure 6.10: Columns in seating table

6.2 Data Manipulation Language (DML commands)

Kitchen values

```
INSERT INTO kitchen VALUES(1,'Kranti Kitchen','VEG');
INSERT INTO kitchen VALUES(2,'Mumbai Kitchen','NON VEG');
INSERT INTO kitchen VALUES(3,'Punjabi Kitchen','VEGAN');
INSERT INTO kitchen VALUES(4,'Kranti Kitchen','VEG');
INSERT INTO kitchen VALUES(5,'MC Donalds','Mixed');
INSERT INTO kitchen VALUES(6,'Yalla Yalla','VEG');
INSERT INTO kitchen VALUES(7,'Manor Kitchen','NON VEG');
INSERT INTO kitchen VALUES(8,'Madhuban','VEGAN');
INSERT INTO kitchen VALUES(9,'Kailash ','MIXED');
INSERT INTO kitchen VALUES(10,'Udupi','VEG');
INSERT INTO kitchen VALUES(11,'Meghana','NON VEG');
INSERT INTO kitchen VALUES(12,'Khana Khazana','VEGAN');
INSERT INTO kitchen VALUES(13,'Lakshmi','MIXED');
INSERT INTO kitchen VALUES(14,'Godavari','VEG');
INSERT INTO kitchen VALUES(15,'Swad Punjab','NON VEG');
INSERT INTO kitchen VALUES(16,'Rolls Mania','VEGAN');
INSERT INTO kitchen VALUES(17,'Rolls on Wheels','MIXED');
INSERT INTO kitchen VALUES(18,'Burger King','VEGAN');
INSERT INTO kitchen VALUES(19,'Eassy Bites','NON VEG');
INSERT INTO kitchen VALUES(20,'Balaji','MIXED');
SELECT * FROM kitchen;
```

Passenger values

```
INSERT INTO passenger VALUES
('adams@gmail.com', 'VARSHA', 1032190848, 827522484, 101);
INSERT INTO passenger VALUES
('allen@gmail.com', 'MANGESH', 788332740, 982240354, 102);
INSERT INTO passenger VALUES
('blake@gmail.com', 'AMRUTA', 730910718, 77380972, 103);
```

```
INSERT INTO passenger VALUES
('clark@gmail.com', 'CHUDAMAN', 652134081, 961919876, 105);
INSERT INTO passenger VALUES
('ford@gmail.com', 'SAMPADA', 351101145, 978177303, 107);
INSERT INTO passenger VALUES
('james@gmail.com', 'SEEMA', 275132251, 830846028, 104);
INSERT INTO passenger VALUES
('jones@gmail.com', 'AMIT', 853545750, 750645619, 109);
INSERT INTO passenger VALUES
('king@gmail.com', 'ANEESH', 666085069, 740021758, 110);
INSERT INTO passenger VALUES
('martin@gmail.com', 'VRUSHALI', 30574633, 770475608, 106);
INSERT INTO passenger VALUES
('scott@gmail.com', 'AROHI', 775609140, 955286477, 111);
INSERT INTO passenger VALUES
('singh@gmail.com', 'VIMAL', 964077614, 882498302, 108);
INSERT INTO passenger VALUES
('smith@gmail.com', 'PARAS', 737390518, 888882570, 112);
INSERT INTO passenger VALUES
('ward@gmail.com', 'PAVAN', 855134483, 992345423, 115);
INSERT INTO passenger VALUES
('kingsley@gmail.com', 'DINA', 865645499, 914600578, 113);
INSERT INTO passenger VALUES
('farooq@gmail.com','ARAB', 919680931, 800589956, 114);
SELECT * FROM passenger;
```

Seating values

```
INSERT INTO seating VALUES
('adams@gmail.com', 18);
INSERT INTO seating VALUES
('allen@gmail.com', 2);
INSERT INTO seating VALUES
('blake@gmail.com', 5);
```

```
INSERT INTO seating VALUES
('clark@gmail.com', 17);
INSERT INTO seating VALUES
('ford@gmail.com', 33);
INSERT INTO seating VALUES
('james@gmail.com', 10);
INSERT INTO seating VALUES
('jones@gmail.com', 19);
INSERT INTO seating VALUES
('king@gmail.com', 24);
INSERT INTO seating VALUES
('martin@gmail.com', 11);
INSERT INTO seating VALUES
('scott@gmail.com', 20);
INSERT INTO seating VALUES
('singh@gmail.com', 45);
INSERT INTO seating VALUES
('smith@gmail.com', 30);
INSERT INTO seating VALUES
('ward@gmail.com', 63);
INSERT INTO seating VALUES
('kingsley@gmail.com', 50);
INSERT INTO seating VALUES
('farooq@gmail.com', 44);
SELECT * FROM seating;
```

<u>Kitchen location values</u>

```
INSERT INTO k_loc (KI_ID,K_location)
Values
(1,'pune'),
(2,'nashik'),
(3,'pune'),
(4,'mumbai'),
(5,'pune'),
(6,'mumbai'),
(7,'nashik'),
(8,'chennai'),
(9,'mumbai'),
(10,'nashik'),
(11,'pune'),
(12,'chennai'),
(13,'nashik'),
(14,'pune'),
(15,'mumbai'),
(16,'pune'),
(17,'chennai'),
(18,'nashik'),
(19,'pune'),
(20,'mumbai');
SELECT * FROM k_loc;
```

Cook values

```
INSERT INTO cook VALUES(11,'chris','Yes',36.0,NULL);
INSERT INTO cook VALUES(12,'Rohit','Partial',36.4,NULL);
INSERT INTO cook VALUES(13,'Rohan','No',34.0,NULL);
INSERT INTO cook VALUES(14,'Rahul','Partial',35.5,NULL);
INSERT INTO cook VALUES(15,'Ben','Yes',36.3,NULL);
INSERT INTO cook VALUES(16,'Robert','Partial',34.0,NULL);
INSERT INTO cook VALUES(17,'Mark','No',34.8,NULL);
INSERT INTO cook VALUES(18,'Sam','Yes',33.6,NULL);
INSERT INTO cook VALUES(19,'Tim','Partial',35.4,NULL);
INSERT INTO cook VALUES(21,'Kartik','Yes',35.7,NULL);
SELECT * FROM cook;
```

Payment values

```
INSERT INTO payment VALUES(101,'UPI',1,'2021-10-01',240,'6');
INSERT INTO payment VALUES(102,'COD',2,'2021-10-01',600,'7');
INSERT INTO payment VALUES(103,'CARD',8,'2021-10-01',800,'2');
INSERT INTO payment VALUES(104,'UPI',4,'2021-10-01',840,'6');

INSERT INTO payment VALUES(105,'Card',7,'2021-10-01',1200,'9');

INSERT INTO payment VALUES(106,'UPI',3,'2021-10-01',160,'8');

INSERT INTO payment VALUES(201,'UPI',5,'2021-10-01',600,'4');

INSERT INTO payment VALUES(202,'COD',2,'2021-10-01',150,'3');

INSERT INTO payment VALUES(205,'CARD',3,'2021-10-01',640,'10');

INSERT INTO payment VALUES(301,'COD',10,'2021-10-01',120,'1');

INSERT INTO payment VALUES(107,'UPI',7,'2021-10-01',450,'8');

INSERT INTO payment VALUES(208,'UPI',3,'2021-10-01',160,'4');

INSERT INTO payment VALUES(209,'COD',1,'2021-10-01',120,'3');

INSERT INTO payment VALUES(203,'CARD',9,'2021-10-01',450,'10');

INSERT INTO payment VALUES(309,'COD',3,'2021-10-01',160,'1');

SELECT * FROM payment;
```

Admin values

```
INSERT INTO Admin (A_ID,A_name,A_Phone)

VALUES

(11,'reuben',985635241),

(18,'pranav',988190020),

(20,'ankit',985568625),

(22,'aditi',985698571);

SELECT * FROM admin;
```

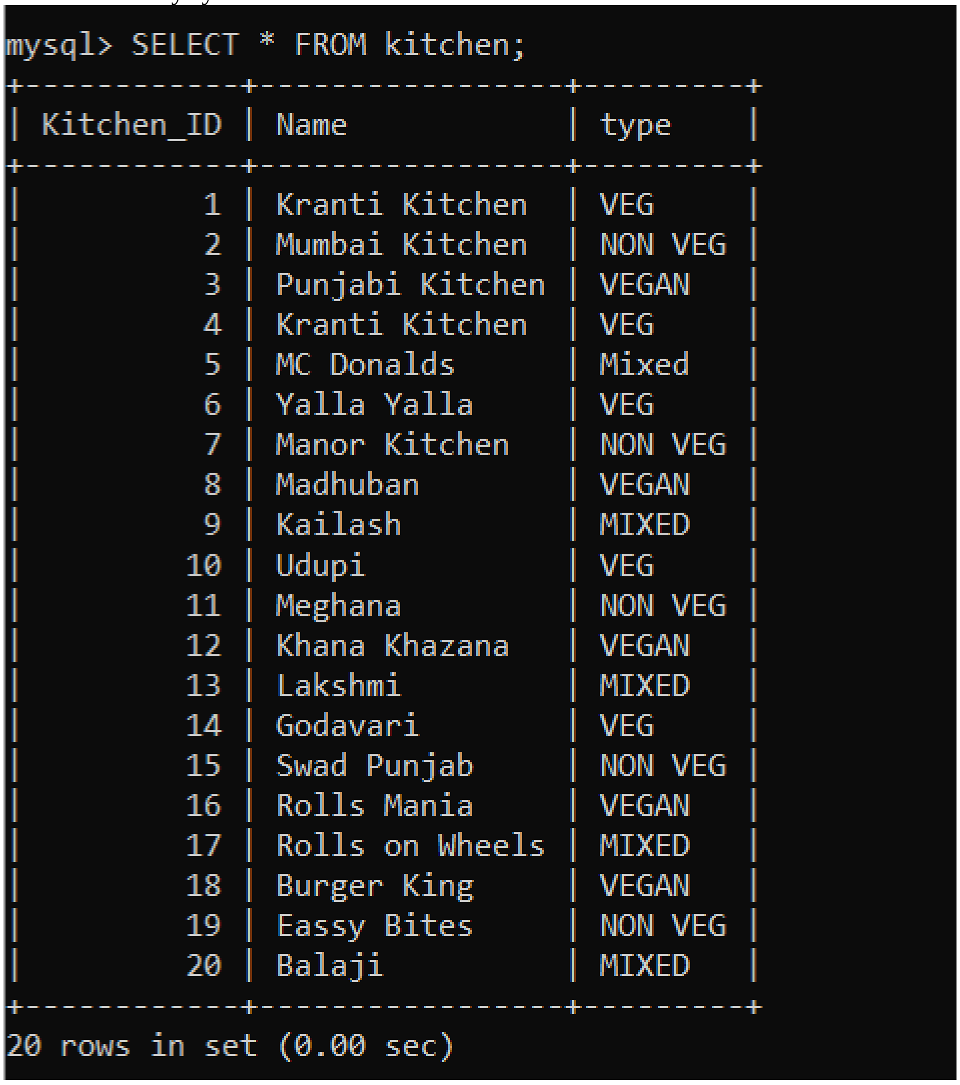
Delivery values

```
INSERT INTO delivery (Tracking_ID,D_Name,D_Phone,ETA,Station,Pass_Email)

VALUES

(24,'aryan',881569201,'2021-10-01 13:23:14','pune', 'scott@gmail.com' );

INSERT INTO delivery (Tracking_ID,D_Name,D_Phone,ETA,Station,Pass_Email)

VALUES

(16,'manish',958384649,'2021-10-01 13:26:16','nashik','martin@gmail.com' ),

(36,'mukesh',923456701,'2021-10-01 13:35:07','mumbai','adams@gmail.com' ),

(72,'vijay',881558692,'2021-10-01 13:47:29', 'pune', 'allen@gmail.com'),

(58,'danish',952369201,'2021-10-01 13:53:55','chennai','blake@gmail.com' ),

(64,'raj',987438201,'2021-10-01 14:05:04','nashik', 'clark@gmail.com'),

(51,'vasu',988116521,'2021-10-01 14:19:45','pune','ford@gmail.com' ),
(39,'alex',98336681,'2021-10-01 14:23:24', 'mumbai', 'james@gmail.com'),
(86,'sudhir',988998821,'2021-10-01 14:33:33','chennai','jones@gmail.com' ),
(63,'mahesh',978563201,'2021-10-01 14:45:40','mumbai','king@gmail.com' ),
(84,'mamoj',977856201,'2021-10-01 14:45:40','mumbai','singh@gmail.com' ),
(61,'om',977853201,'2021-10-01 14:45:40','mumbai','smith@gmail.com' ),
(49,'sai',978563201,'2021-10-01 14:45:40','mumbai','ward@gmail.com' ),
(97,'vikas',977863201,'2021-10-01 14:45:40','mumbai','kingsley@gmail.com'),
(21,'vinod',977863201,'2021-10-01 14:45:40','mumbai','farooq@gmail.com' );
SELECT * FROM delivery;
```

<u>Meal values</u>

```
INSERT INTO meal VALUES(101,NULL,'VEG','120','2','1','16');
INSERT INTO meal VALUES(102,NULL,'NON VEG','150','4','2','21');
INSERT INTO meal VALUES(103,NULL,'VEGAN','160','5','8','24');
INSERT INTO meal VALUES(105,NULL,'VEG','120','7','4','36');
INSERT INTO meal VALUES(114,NULL,'NON VEG','150','8','7','39');
INSERT INTO meal VALUES(107,NULL,'VEGAN','160','1','3','49');
INSERT INTO meal VALUES(104,NULL,'VEG','120','5','5','51');
INSERT INTO meal VALUES(109,NULL,'NON VEG','150','1','2','58');
INSERT INTO meal VALUES(110,NULL,'VEGAN','160','4','3','61');
INSERT INTO meal VALUES(113,NULL,'VEG','120','1','10','64');
INSERT INTO meal VALUES(106,NULL,'NON VEG','150','3','7','63');
INSERT INTO meal VALUES(111,NULL,'VEGAN','160',1,'3','72');
INSERT INTO meal VALUES(108,NULL,'VEG','120','1','1','84');
INSERT INTO meal VALUES(112,NULL,'NON VEG','150','3','9','86');
INSERT INTO meal VALUES(115,NULL,'VEGAN','160',1,'3','97');
SELECT * FROM meal;
UPDATE meal SET Special_req = 'Jain' WHERE meal_id = 110;
UPDATE meal SET Special_req = 'Jain' WHERE meal_id = 115;
UPDATE meal SET Special_req = 'Jain' WHERE meal_id = 101;
UPDATE meal SET Special_req = 'Pepperoni' WHERE meal_id = 102;
UPDATE meal SET Special_req = 'Extra cheese' WHERE meal_id = 111;
```

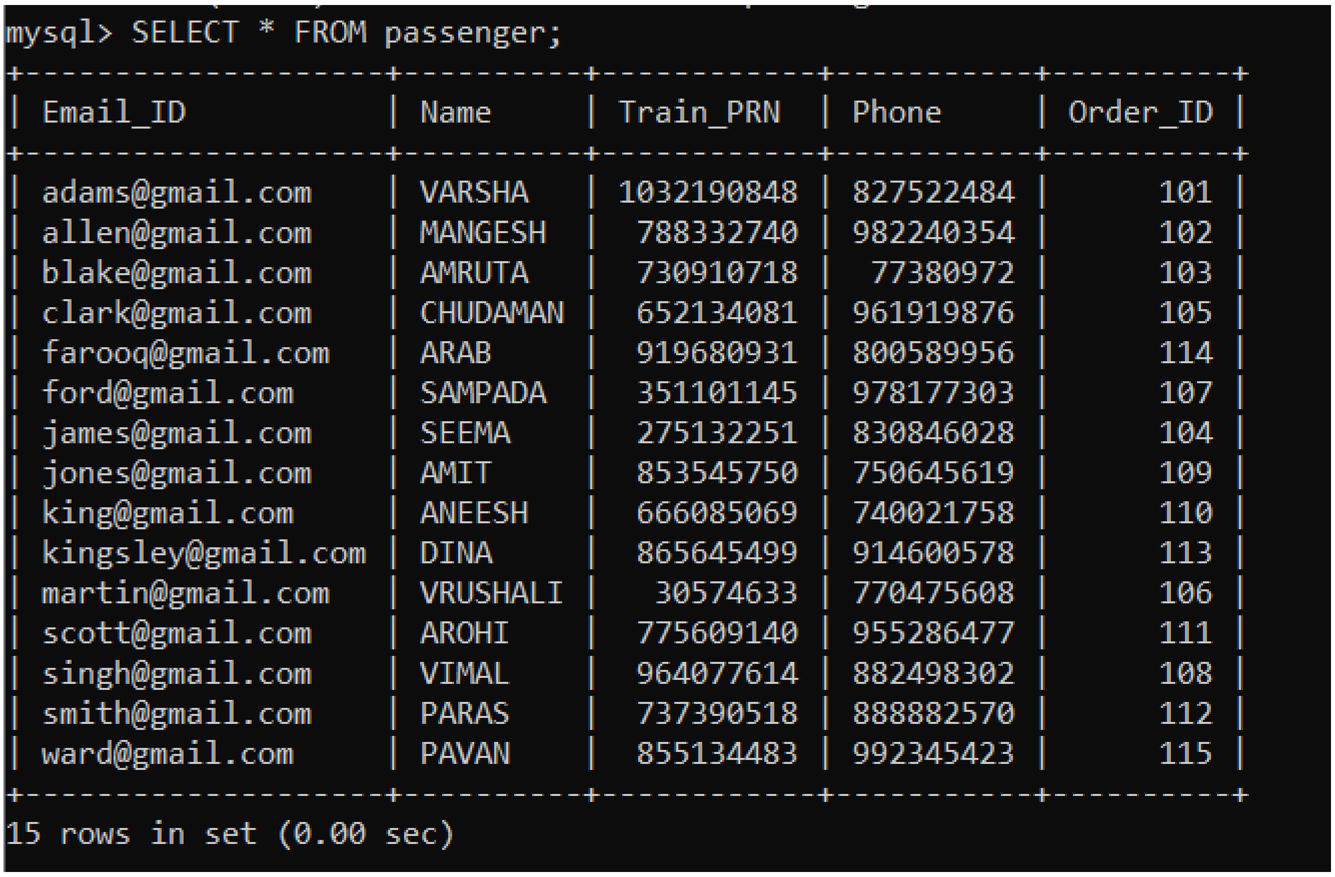
Records in kitchen table have been depicted in figure 6.11



```
mysql> SELECT * FROM kitchen;
+------------+-----------------+---------+
| Kitchen_ID | Name            | type    |
+------------+-----------------+---------+
|          1 | Kranti Kitchen  | VEG     |
|          2 | Mumbai Kitchen  | NON VEG |
|          3 | Punjabi Kitchen | VEGAN   |
|          4 | Kranti Kitchen  | VEG     |
|          5 | MC Donalds      | Mixed   |
|          6 | Yalla Yalla     | VEG     |
|          7 | Manor Kitchen   | NON VEG |
|          8 | Madhuban        | VEGAN   |
|          9 | Kailash         | MIXED   |
|         10 | Udupi           | VEG     |
|         11 | Meghana         | NON VEG |
|         12 | Khana Khazana   | VEGAN   |
|         13 | Lakshmi         | MIXED   |
|         14 | Godavari        | VEG     |
|         15 | Swad Punjab     | NON VEG |
|         16 | Rolls Mania     | VEGAN   |
|         17 | Rolls on Wheels | MIXED   |
|         18 | Burger King     | VEGAN   |
|         19 | Eassy Bites     | NON VEG |
|         20 | Balaji          | MIXED   |
+------------+-----------------+---------+
20 rows in set (0.00 sec)
```

Figure 6.11: Records in kitchen table

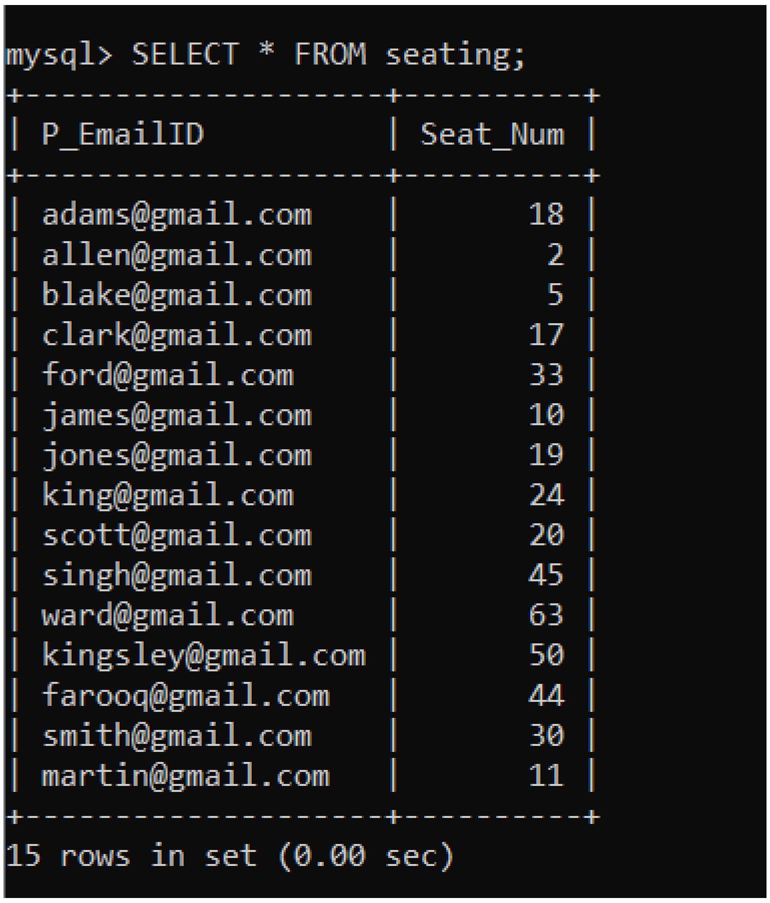
Records from passenger table have been illustrated in table 6.12 below



```
mysql> SELECT * FROM passenger;
+---------------------+----------+------------+-----------+----------+
| Email_ID            | Name     | Train_PRN  | Phone     | Order_ID |
+---------------------+----------+------------+-----------+----------+
| adams@gmail.com     | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com     | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com     | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com     | CHUDAMAN |  652134081 | 961919876 |      105 |
| farooq@gmail.com    | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com      | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com     | SEEMA    |  275132251 | 830846028 |      104 |
| jones@gmail.com     | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com      | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com  | DINA     |  865645499 | 914600578 |      113 |
| martin@gmail.com    | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com     | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com     | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com     | PARAS    |  737390518 | 888882570 |      112 |
| ward@gmail.com      | PAVAN    |  855134483 | 992345423 |      115 |
+---------------------+----------+------------+-----------+----------+
15 rows in set (0.00 sec)
```

Figure 6.12: Records in passenger table

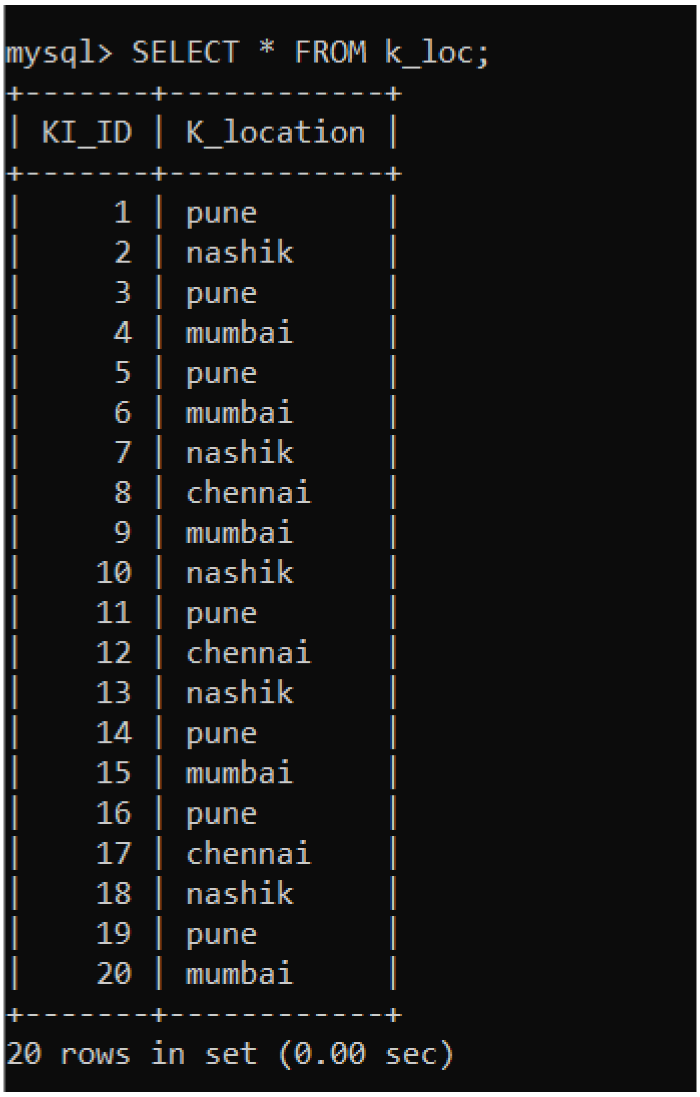
Records in seating table are shown in figure 6.13



```
mysql> SELECT * FROM seating;
+--------------------+----------+
| P_EmailID          | Seat_Num |
+--------------------+----------+
| adams@gmail.com    |       18 |
| allen@gmail.com    |        2 |
| blake@gmail.com    |        5 |
| clark@gmail.com    |       17 |
| ford@gmail.com     |       33 |
| james@gmail.com    |       10 |
| jones@gmail.com    |       19 |
| king@gmail.com     |       24 |
| scott@gmail.com    |       20 |
| singh@gmail.com    |       45 |
| ward@gmail.com     |       63 |
| kingsley@gmail.com |       50 |
| farooq@gmail.com   |       44 |
| smith@gmail.com    |       30 |
| martin@gmail.com   |       11 |
+--------------------+----------+
15 rows in set (0.00 sec)
```

Figure 6.13: Records in seating table

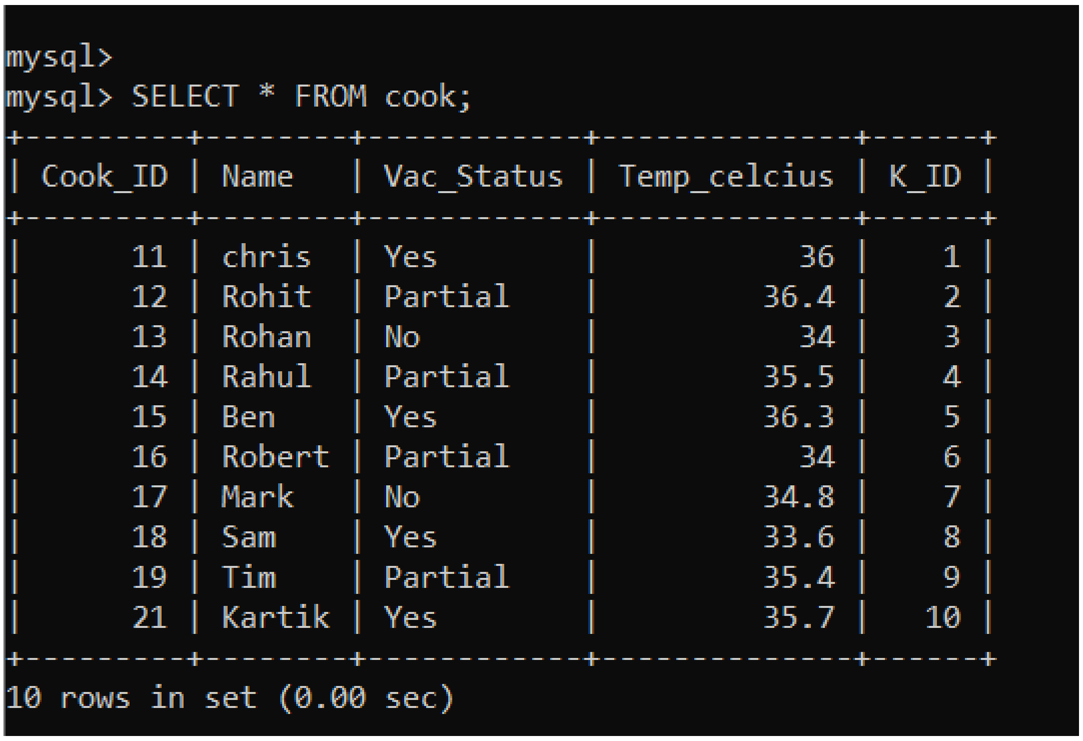
Records in k_loc table are depicted in figure 6.13



```
mysql> SELECT * FROM k_loc;
+-------+------------+
| KI_ID | K_location |
+-------+------------+
|     1 | pune       |
|     2 | nashik     |
|     3 | pune       |
|     4 | mumbai     |
|     5 | pune       |
|     6 | mumbai     |
|     7 | nashik     |
|     8 | chennai    |
|     9 | mumbai     |
|    10 | nashik     |
|    11 | pune       |
|    12 | chennai    |
|    13 | nashik     |
|    14 | pune       |
|    15 | mumbai     |
|    16 | pune       |
|    17 | chennai    |
|    18 | nashik     |
|    19 | pune       |
|    20 | mumbai     |
+-------+------------+
20 rows in set (0.00 sec)
```

Figure 6.14: Records in k_loc table

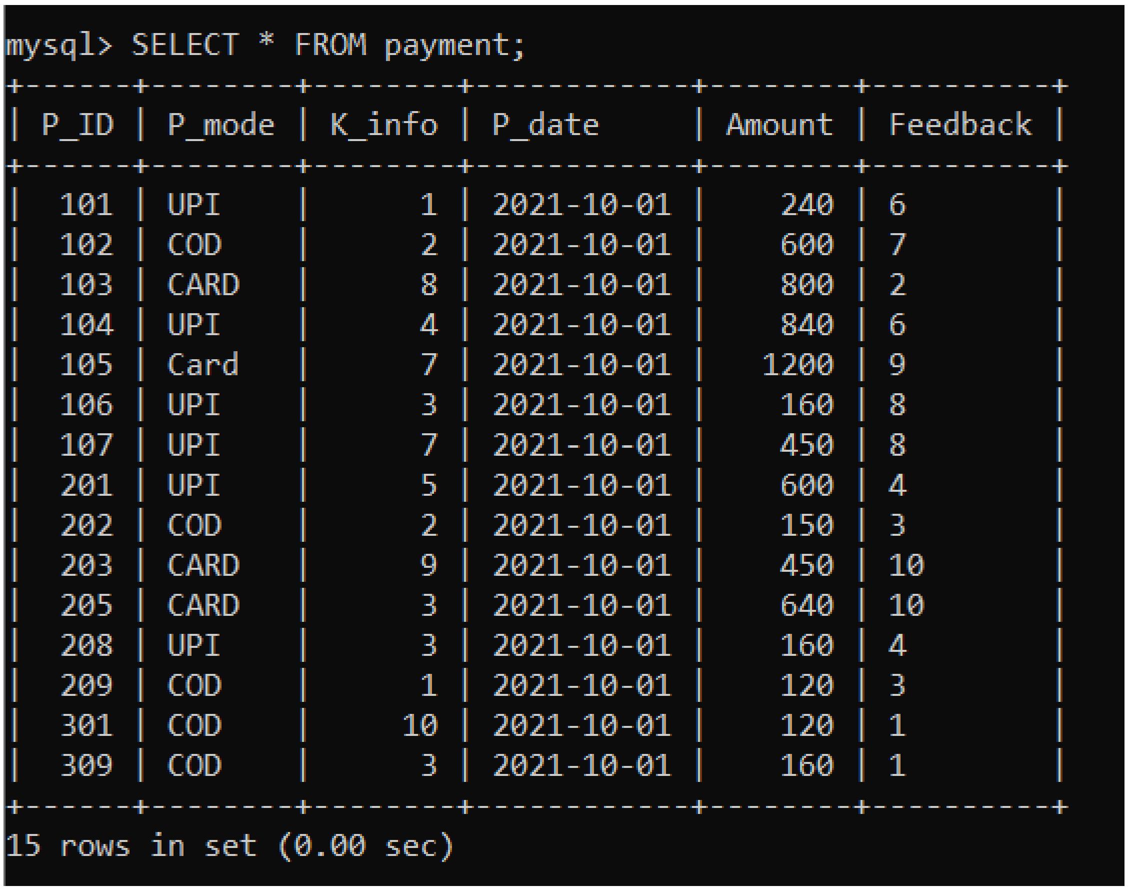
Records in cook table have been shown in figure 6.15



```
mysql>
mysql> SELECT * FROM cook;
+---------+--------+-----------+-------------+------+
| Cook_ID | Name   | Vac_Status | Temp_celcius | K_ID |
+---------+--------+-----------+-------------+------+
|      11 | chris  | Yes        |           36 |    1 |
|      12 | Rohit  | Partial    |         36.4 |    2 |
|      13 | Rohan  | No         |           34 |    3 |
|      14 | Rahul  | Partial    |         35.5 |    4 |
|      15 | Ben    | Yes        |         36.3 |    5 |
|      16 | Robert | Partial    |           34 |    6 |
|      17 | Mark   | No         |         34.8 |    7 |
|      18 | Sam    | Yes        |         33.6 |    8 |
|      19 | Tim    | Partial    |         35.4 |    9 |
|      21 | Kartik | Yes        |         35.7 |   10 |
+---------+--------+-----------+-------------+------+
10 rows in set (0.00 sec)
```

Figure 6.15: Records in cook table

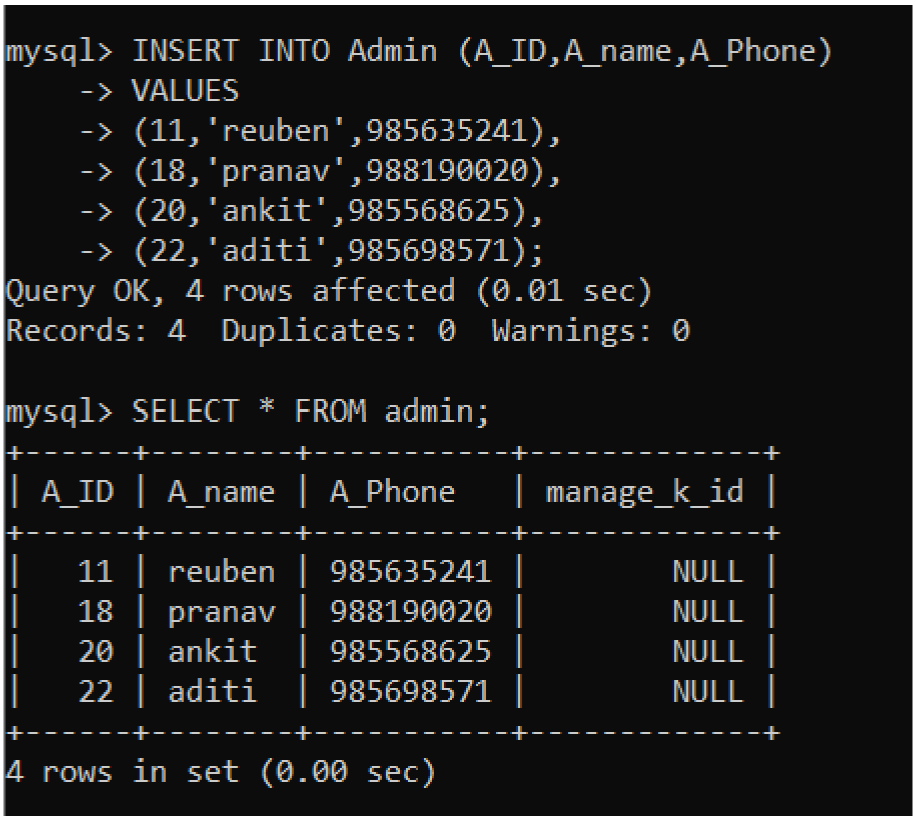
Records in payment table have been illustrated in figure 6.16



```
mysql> SELECT * FROM payment;
+------+--------+--------+------------+--------+----------+
| P_ID | P_mode | K_info | P_date     | Amount | Feedback |
+------+--------+--------+------------+--------+----------+
|  101 | UPI    |      1 | 2021-10-01 |    240 | 6        |
|  102 | COD    |      2 | 2021-10-01 |    600 | 7        |
|  103 | CARD   |      8 | 2021-10-01 |    800 | 2        |
|  104 | UPI    |      4 | 2021-10-01 |    840 | 6        |
|  105 | Card   |      7 | 2021-10-01 |   1200 | 9        |
|  106 | UPI    |      3 | 2021-10-01 |    160 | 8        |
|  107 | UPI    |      7 | 2021-10-01 |    450 | 8        |
|  201 | UPI    |      5 | 2021-10-01 |    600 | 4        |
|  202 | COD    |      2 | 2021-10-01 |    150 | 3        |
|  203 | CARD   |      9 | 2021-10-01 |    450 | 10       |
|  205 | CARD   |      3 | 2021-10-01 |    640 | 10       |
|  208 | UPI    |      3 | 2021-10-01 |    160 | 4        |
|  209 | COD    |      1 | 2021-10-01 |    120 | 3        |
|  301 | COD    |     10 | 2021-10-01 |    120 | 1        |
|  309 | COD    |      3 | 2021-10-01 |    160 | 1        |
+------+--------+--------+------------+--------+----------+
15 rows in set (0.00 sec)
```

Figure 6.16: Records in payment table

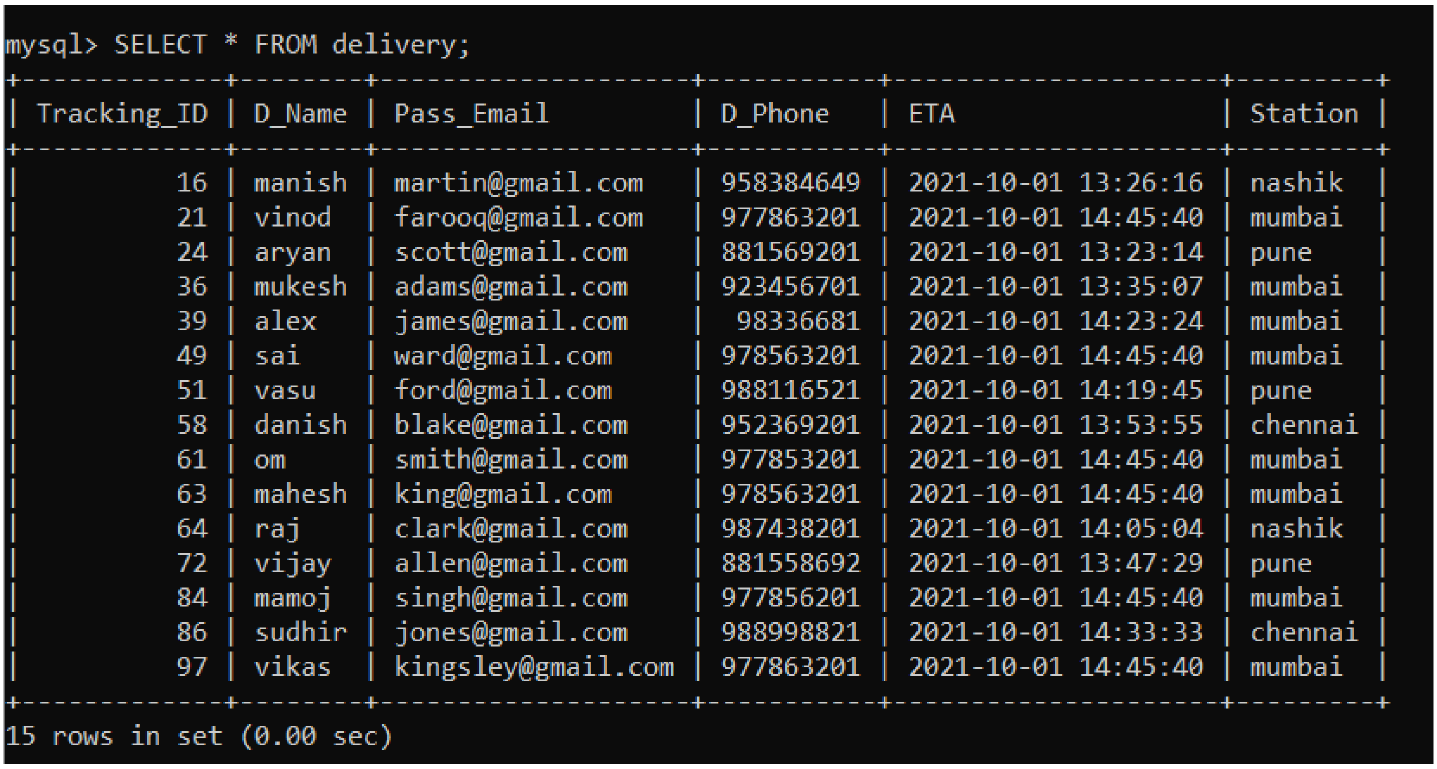
Records in admin table are names of developers, as they have control over access and manipulations in database. This has been depicted in figure 6.17



```
mysql> INSERT INTO Admin (A_ID,A_name,A_Phone)
    -> VALUES
    -> (11,'reuben',985635241),
    -> (18,'pranav',988190020),
    -> (20,'ankit',985568625),
    -> (22,'aditi',985698571);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM admin;
+------+--------+-----------+-------------+
| A_ID | A_name | A_Phone   | manage_k_id |
+------+--------+-----------+-------------+
|   11 | reuben | 985635241 |        NULL |
|   18 | pranav | 988190020 |        NULL |
|   20 | ankit  | 985568625 |        NULL |
|   22 | aditi  | 985698571 |        NULL |
+------+--------+-----------+-------------+
4 rows in set (0.00 sec)
```

Figure 6.17: Records in payment table

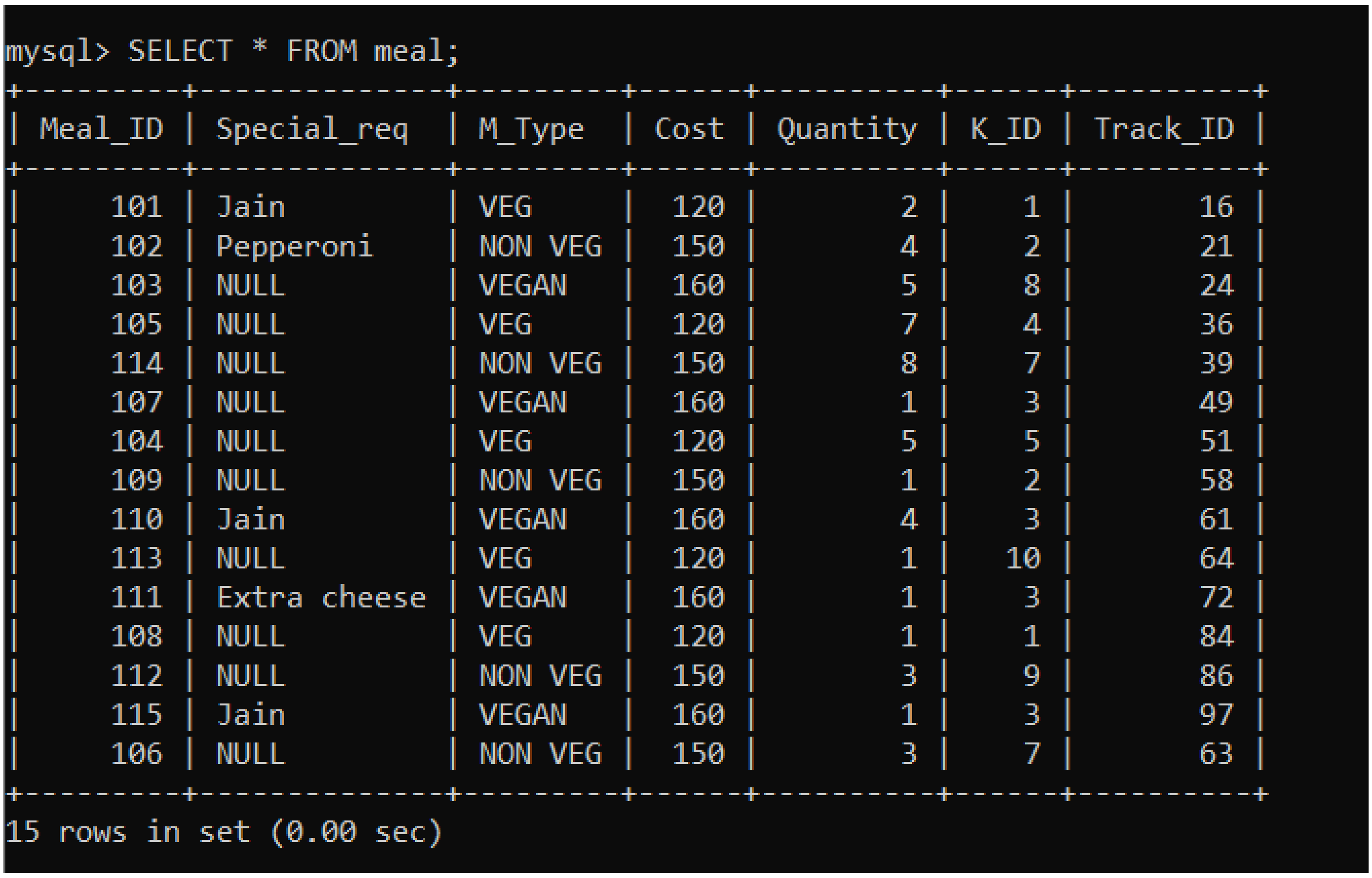
Records in delivery table have been shown in figure 6.18



```
mysql> SELECT * FROM delivery;
+-------------+--------+-------------------+-----------+---------------------+---------+
| Tracking_ID | D_Name | Pass_Email        | D_Phone   | ETA                 | Station |
+-------------+--------+-------------------+-----------+---------------------+---------+
|          16 | manish | martin@gmail.com  | 958384649 | 2021-10-01 13:26:16 | nashik  |
|          21 | vinod  | farooq@gmail.com  | 977863201 | 2021-10-01 14:45:40 | mumbai  |
|          24 | aryan  | scott@gmail.com   | 881569201 | 2021-10-01 13:23:14 | pune    |
|          36 | mukesh | adams@gmail.com   | 923456701 | 2021-10-01 13:35:07 | mumbai  |
|          39 | alex   | james@gmail.com   |  98336681 | 2021-10-01 14:23:24 | mumbai  |
|          49 | sai    | ward@gmail.com    | 978563201 | 2021-10-01 14:45:40 | mumbai  |
|          51 | vasu   | ford@gmail.com    | 988116521 | 2021-10-01 14:19:45 | pune    |
|          58 | danish | blake@gmail.com   | 952369201 | 2021-10-01 13:53:55 | chennai |
|          61 | om     | smith@gmail.com   | 977853201 | 2021-10-01 14:45:40 | mumbai  |
|          63 | mahesh | king@gmail.com    | 978563201 | 2021-10-01 14:45:40 | mumbai  |
|          64 | raj    | clark@gmail.com   | 987438201 | 2021-10-01 14:05:04 | nashik  |
|          72 | vijay  | allen@gmail.com   | 881558692 | 2021-10-01 13:47:29 | pune    |
|          84 | mamoj  | singh@gmail.com   | 977856201 | 2021-10-01 14:45:40 | mumbai  |
|          86 | sudhir | jones@gmail.com   | 988998821 | 2021-10-01 14:33:33 | chennai |
|          97 | vikas  | kingsley@gmail.com | 977863201 | 2021-10-01 14:45:40 | mumbai  |
+-------------+--------+-------------------+-----------+---------------------+---------+
15 rows in set (0.00 sec)
```

Figure 6.18: Records in delivery table

Records in meal table have been portrayed in figure 6.19

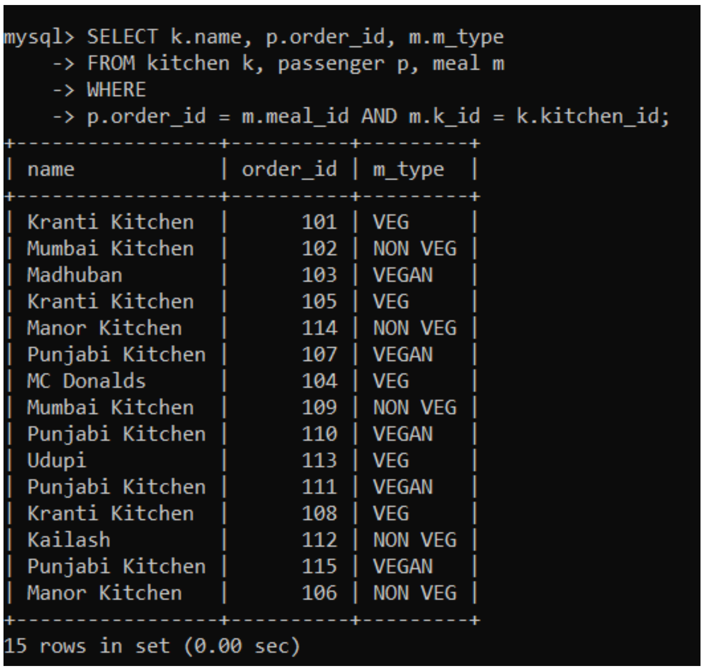
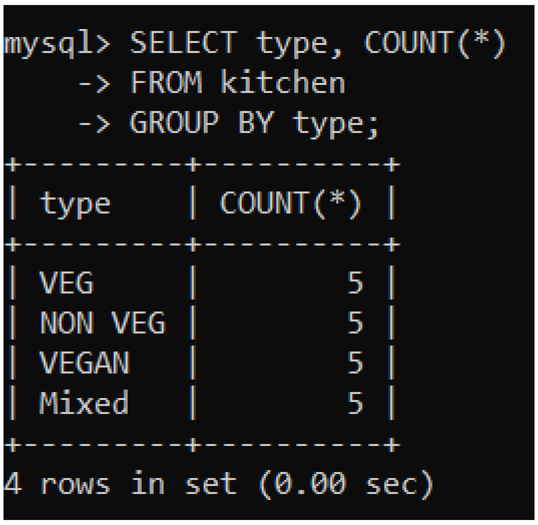


```
mysql> SELECT * FROM meal;
+---------+--------------+---------+------+----------+------+----------+
| Meal_ID | Special_req  | M_Type  | Cost | Quantity | K_ID | Track_ID |
+---------+--------------+---------+------+----------+------+----------+
|     101 | Jain         | VEG     |  120 |        2 |    1 |       16 |
|     102 | Pepperoni    | NON VEG |  150 |        4 |    2 |       21 |
|     103 | NULL         | VEGAN   |  160 |        5 |    8 |       24 |
|     105 | NULL         | VEG     |  120 |        7 |    4 |       36 |
|     114 | NULL         | NON VEG |  150 |        8 |    7 |       39 |
|     107 | NULL         | VEGAN   |  160 |        1 |    3 |       49 |
|     104 | NULL         | VEG     |  120 |        5 |    5 |       51 |
|     109 | NULL         | NON VEG |  150 |        1 |    2 |       58 |
|     110 | Jain         | VEGAN   |  160 |        4 |    3 |       61 |
|     113 | NULL         | VEG     |  120 |        1 |   10 |       64 |
|     111 | Extra cheese | VEGAN   |  160 |        1 |    3 |       72 |
|     108 | NULL         | VEG     |  120 |        1 |    1 |       84 |
|     112 | NULL         | NON VEG |  150 |        3 |    9 |       86 |
|     115 | Jain         | VEGAN   |  160 |        1 |    3 |       97 |
|     106 | NULL         | NON VEG |  150 |        3 |    7 |       63 |
+---------+--------------+---------+------+----------+------+----------+
15 rows in set (0.00 sec)
```

Figure 6.19: Records in payment table

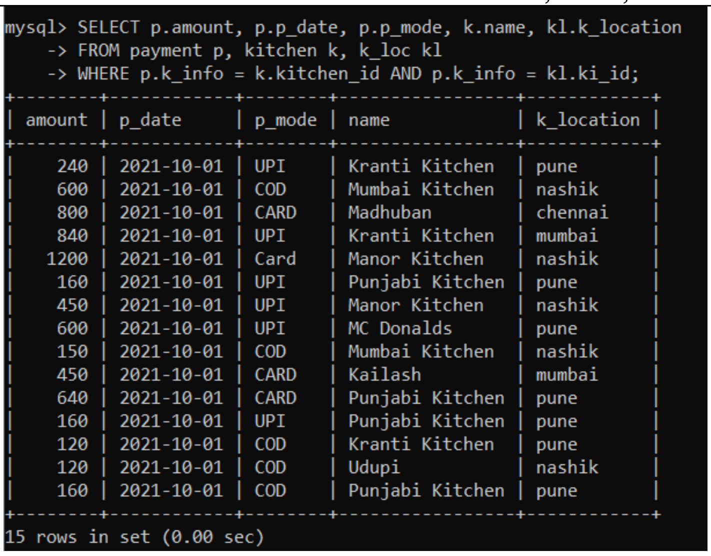
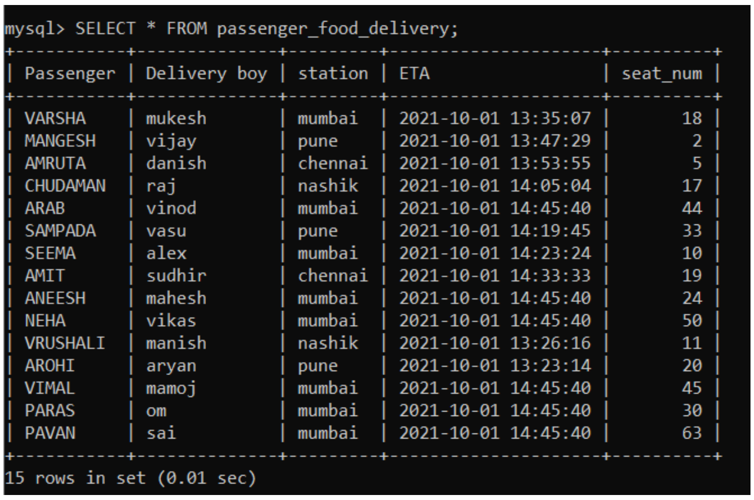
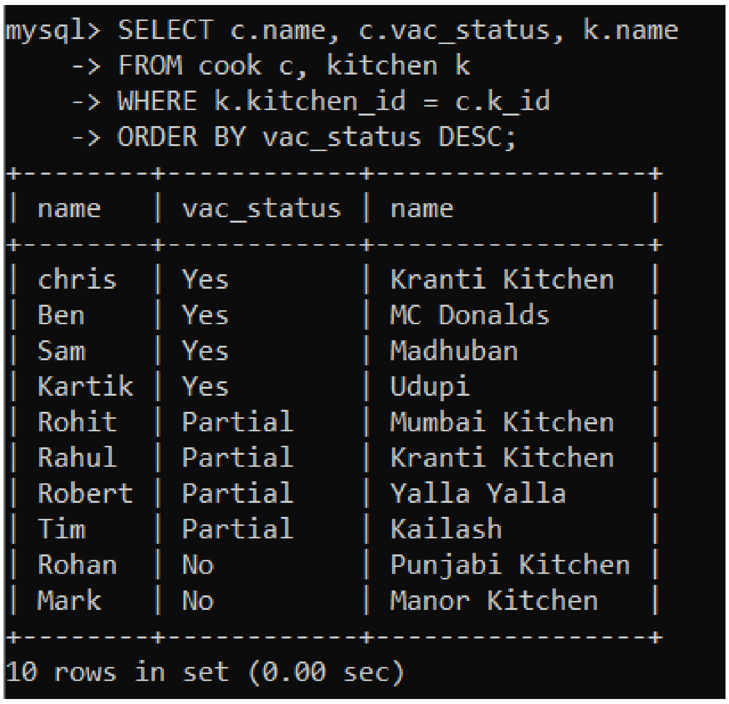
Select queries to view certain records have been shown in table 6.1 below:

Table 6.2: DML queries on database

Required data	**Query in MySQL**	**Screenshot of results**
Display kitchen name, order numbers received and type of meal the kitchens serve	SELECT k.name, p.order_id, m.m_type FROM kitchen k, passenger p, meal m WHERE p.order_id = m.meal_id AND m.k_id = k.kitchen_id;	
No. of available kitchens and the type of food they serve	SELECT type, COUNT(*) FROM kitchen GROUP BY type;	
Display money earned, date and mode of payment for a kitchen, in a given location	SELECT p.amount, p.p_date, p.p_mode, k.name, kl.k_location FROM payment p, kitchen k, k_loc kl WHERE p.k_info = k.kitchen_id AND p.k_info = kl.ki_id;	

```
mysql> SELECT k.name, p.order_id, m.m_type
    -> FROM kitchen k, passenger p, meal m
    -> WHERE
    -> p.order_id = m.meal_id AND m.k_id = k.kitchen_id;
+----------------+----------+---------+
| name           | order_id | m_type  |
+----------------+----------+---------+
| Kranti Kitchen |      101 | VEG     |
| Mumbai Kitchen |      102 | NON VEG |
| Madhuban       |      103 | VEGAN   |
| Kranti Kitchen |      105 | VEG     |
| Manor Kitchen  |      114 | NON VEG |
| Punjabi Kitchen|      107 | VEGAN   |
| MC Donalds     |      104 | VEG     |
| Mumbai Kitchen |      109 | NON VEG |
| Punjabi Kitchen|      110 | VEGAN   |
| Udupi          |      113 | VEG     |
| Punjabi Kitchen|      111 | VEGAN   |
| Kranti Kitchen |      108 | VEG     |
| Kailash        |      112 | NON VEG |
| Punjabi Kitchen|      115 | VEGAN   |
| Manor Kitchen  |      106 | NON VEG |
+----------------+----------+---------+
15 rows in set (0.00 sec)
```

```
mysql> SELECT type, COUNT(*)
    -> FROM kitchen
    -> GROUP BY type;
+---------+----------+
| type    | COUNT(*) |
+---------+----------+
| VEG     |        5 |
| NON VEG |        5 |
| VEGAN   |        5 |
| Mixed   |        5 |
+---------+----------+
4 rows in set (0.00 sec)
```

		
Create a view of passenger details and delivery boy details	CREATE VIEW passenger_food_delivery AS SELECT p.name AS 'Passenger', d.d_name AS 'Delivery boy', d.station, d.ETA, s.seat_num FROM passenger p, delivery d, seating s WHERE p.email_id = d.pass_email AND p.email_id = s.P_EmailID;	
Display cooks details and vaccination status of them working in a specific kitchen	SELECT c.name, c.vac_status, k.name FROM cook c, kitchen k WHERE k.kitchen_id = c.k_id ORDER BY vac_status DESC;	

```
mysql> SELECT p.amount, p.p_date, p.p_mode, k.name, kl.k_location
    -> FROM payment p, kitchen k, k_loc kl
    -> WHERE p.k_info = k.kitchen_id AND p.k_info = kl.ki_id;
+--------+------------+--------+-----------------+------------+
| amount | p_date     | p_mode | name            | k_location |
+--------+------------+--------+-----------------+------------+
|    240 | 2021-10-01 | UPI    | Kranti Kitchen  | pune       |
|    600 | 2021-10-01 | COD    | Mumbai Kitchen  | nashik     |
|    800 | 2021-10-01 | CARD   | Madhuban        | chennai    |
|    840 | 2021-10-01 | UPI    | Kranti Kitchen  | mumbai     |
|   1200 | 2021-10-01 | Card   | Manor Kitchen   | nashik     |
|    160 | 2021-10-01 | UPI    | Punjabi Kitchen | pune       |
|    450 | 2021-10-01 | UPI    | Manor Kitchen   | nashik     |
|    600 | 2021-10-01 | UPI    | MC Donalds      | pune       |
|    150 | 2021-10-01 | COD    | Mumbai Kitchen  | nashik     |
|    450 | 2021-10-01 | CARD   | Kailash         | mumbai     |
|    640 | 2021-10-01 | CARD   | Punjabi Kitchen | pune       |
|    160 | 2021-10-01 | UPI    | Punjabi Kitchen | pune       |
|    120 | 2021-10-01 | COD    | Kranti Kitchen  | pune       |
|    120 | 2021-10-01 | COD    | Udupi           | nashik     |
|    160 | 2021-10-01 | COD    | Punjabi Kitchen | pune       |
+--------+------------+--------+-----------------+------------+
15 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM passenger_food_delivery;
+-----------+--------------+---------+---------------------+----------+
| Passenger | Delivery boy | station | ETA                 | seat_num |
+-----------+--------------+---------+---------------------+----------+
| VARSHA    | mukesh       | mumbai  | 2021-10-01 13:35:07 |       18 |
| MANGESH   | vijay        | pune    | 2021-10-01 13:47:29 |        2 |
| AMRUTA    | danish       | chennai | 2021-10-01 13:53:55 |        5 |
| CHUDAMAN  | raj          | nashik  | 2021-10-01 14:05:04 |       17 |
| ARAB      | vinod        | mumbai  | 2021-10-01 14:45:40 |       44 |
| SAMPADA   | vasu         | pune    | 2021-10-01 14:19:45 |       33 |
| SEEMA     | alex         | mumbai  | 2021-10-01 14:23:24 |       10 |
| AMIT      | sudhir       | chennai | 2021-10-01 14:33:33 |       19 |
| ANEESH    | mahesh       | mumbai  | 2021-10-01 14:45:40 |       24 |
| NEHA      | vikas        | mumbai  | 2021-10-01 14:45:40 |       50 |
| VRUSHALI  | manish       | nashik  | 2021-10-01 13:26:16 |       11 |
| AROHI     | aryan        | pune    | 2021-10-01 13:23:14 |       20 |
| VIMAL     | mamoj        | mumbai  | 2021-10-01 14:45:40 |       45 |
| PARAS     | om           | mumbai  | 2021-10-01 14:45:40 |       30 |
| PAVAN     | sai          | mumbai  | 2021-10-01 14:45:40 |       63 |
+-----------+--------------+---------+---------------------+----------+
15 rows in set (0.01 sec)
```

```
mysql> SELECT c.name, c.vac_status, k.name
    -> FROM cook c, kitchen k
    -> WHERE k.kitchen_id = c.k_id
    -> ORDER BY vac_status DESC;
+--------+------------+-----------------+
| name   | vac_status | name            |
+--------+------------+-----------------+
| chris  | Yes        | Kranti Kitchen  |
| Ben    | Yes        | MC Donalds      |
| Sam    | Yes        | Madhuban        |
| Kartik | Yes        | Udupi           |
| Rohit  | Partial    | Mumbai Kitchen  |
| Rahul  | Partial    | Kranti Kitchen  |
| Robert | Partial    | Yalla Yalla     |
| Tim    | Partial    | Kailash         |
| Rohan  | No         | Punjabi Kitchen |
| Mark   | No         | Manor Kitchen   |
+--------+------------+-----------------+
10 rows in set (0.00 sec)
```

6.3 Data Control Language (DCL commands)

All 4 admins – Aditi, Ankit, Pranav and Reuben – can access all tables in meals on wheels database, but operations they can perform are limited. For instance, access to select, update and delete from tables have been granted to 3 admins, while admin Aditi can only use select from the tables.

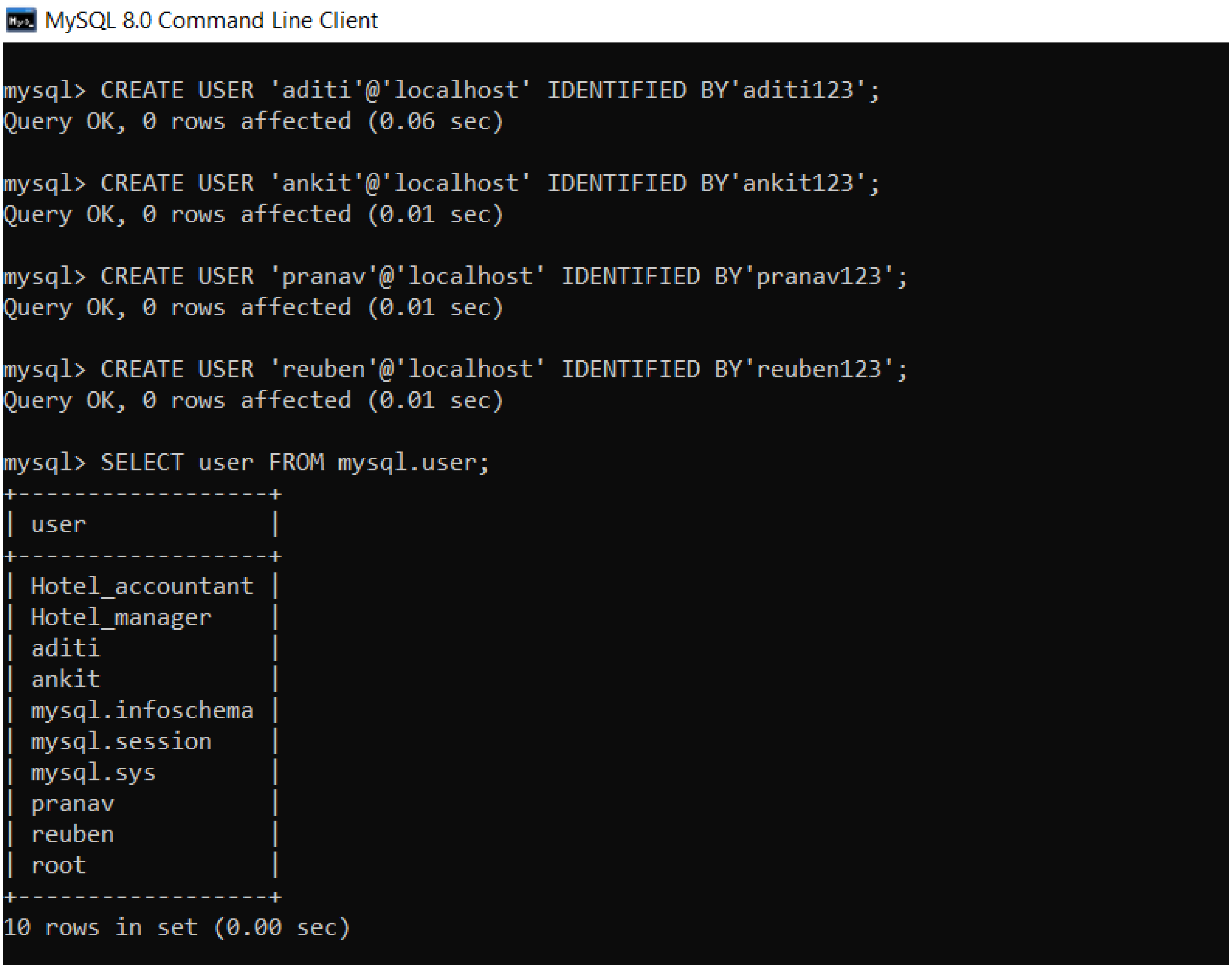
<u>Create admin logins</u>

```
CREATE USER 'aditi'@'localhost' IDENTIFIED BY'aditi123';

CREATE USER 'ankit'@'localhost' IDENTIFIED BY'ankit123';

CREATE USER 'pranav'@'localhost' IDENTIFIED BY'pranav123';

CREATE USER 'reuben'@'localhost' IDENTIFIED BY'reuben123';

SELECT user FROM mysql.user;
```

<u>Grant and view privileges</u>

```
GRANT SELECT ON mealsonwheels.* TO 'aditi'@'localhost';

FLUSH PRIVILEGES;

SHOW GRANTS FOR 'aditi'@'localhost';

GRANT ALL ON mealsonwheels.* TO 'ankit'@'localhost';

FLUSH PRIVILEGES;

SHOW GRANTS FOR 'ankit'@'localhost';

GRANT ALL ON mealsonwheels.* TO 'pranav'@'localhost';

FLUSH PRIVILEGES;

SHOW GRANTS FOR 'pranav'@'localhost';

GRANT SELECT ON mealsonwheels.* TO 'reuben'@'localhost';

FLUSH PRIVILEGES;

SHOW GRANTS FOR 'reuben'@'localhost';

GRANT UPDATE ON mealsonwheels.* TO 'reuben'@'localhost';

GRANT DELETE ON mealsonwheels.* TO 'reuben'@'localhost';

FLUSH PRIVILEGES;

SHOW GRANTS FOR 'reuben'@'localhost';
```

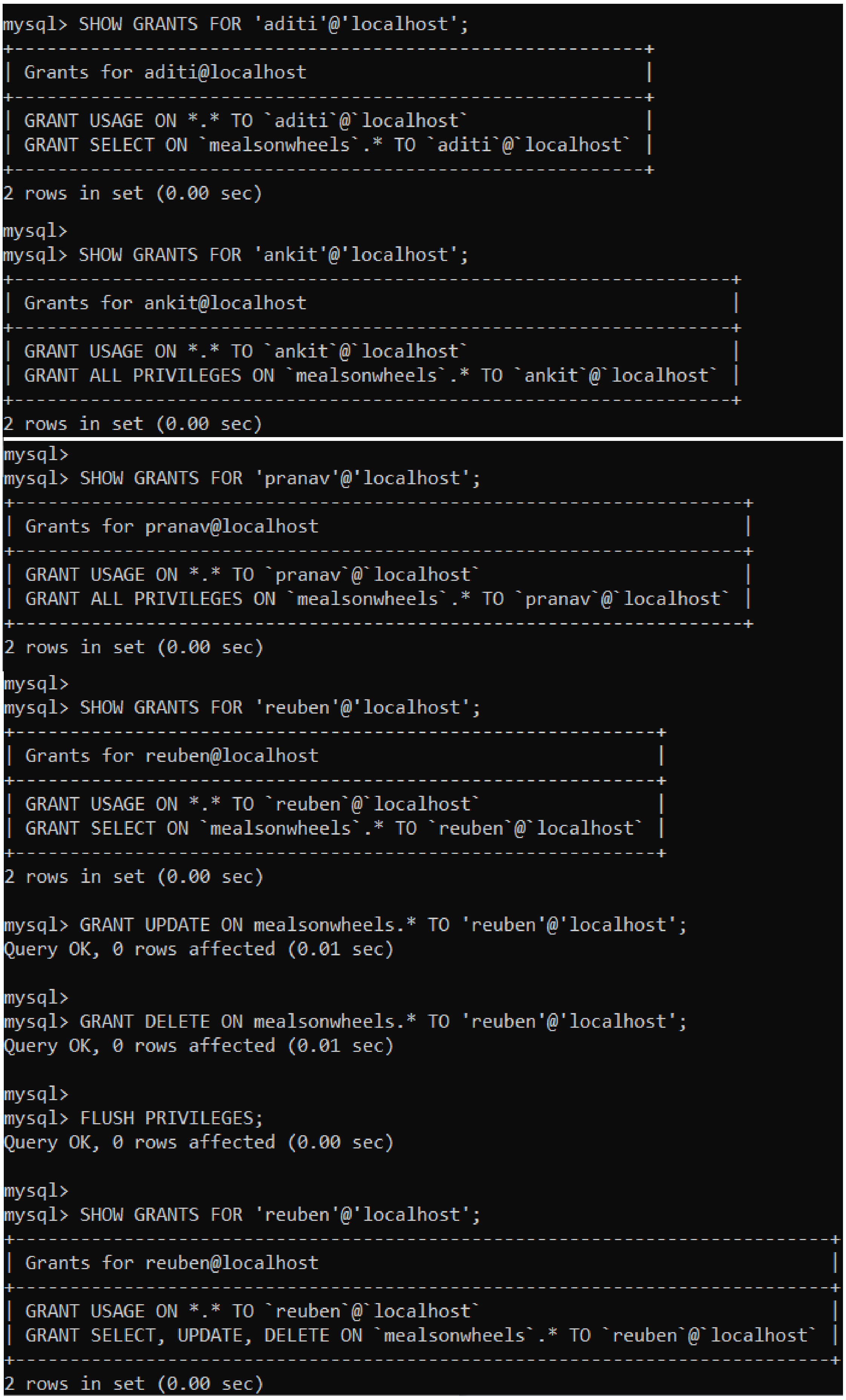
The usernames and passwords of 4 admins have been declared using following commands. Users created have been shown in figure 6.20. The results of access level commands have been demonstrated as given in figure 6.21



MySQL 8.0 Command Line Client

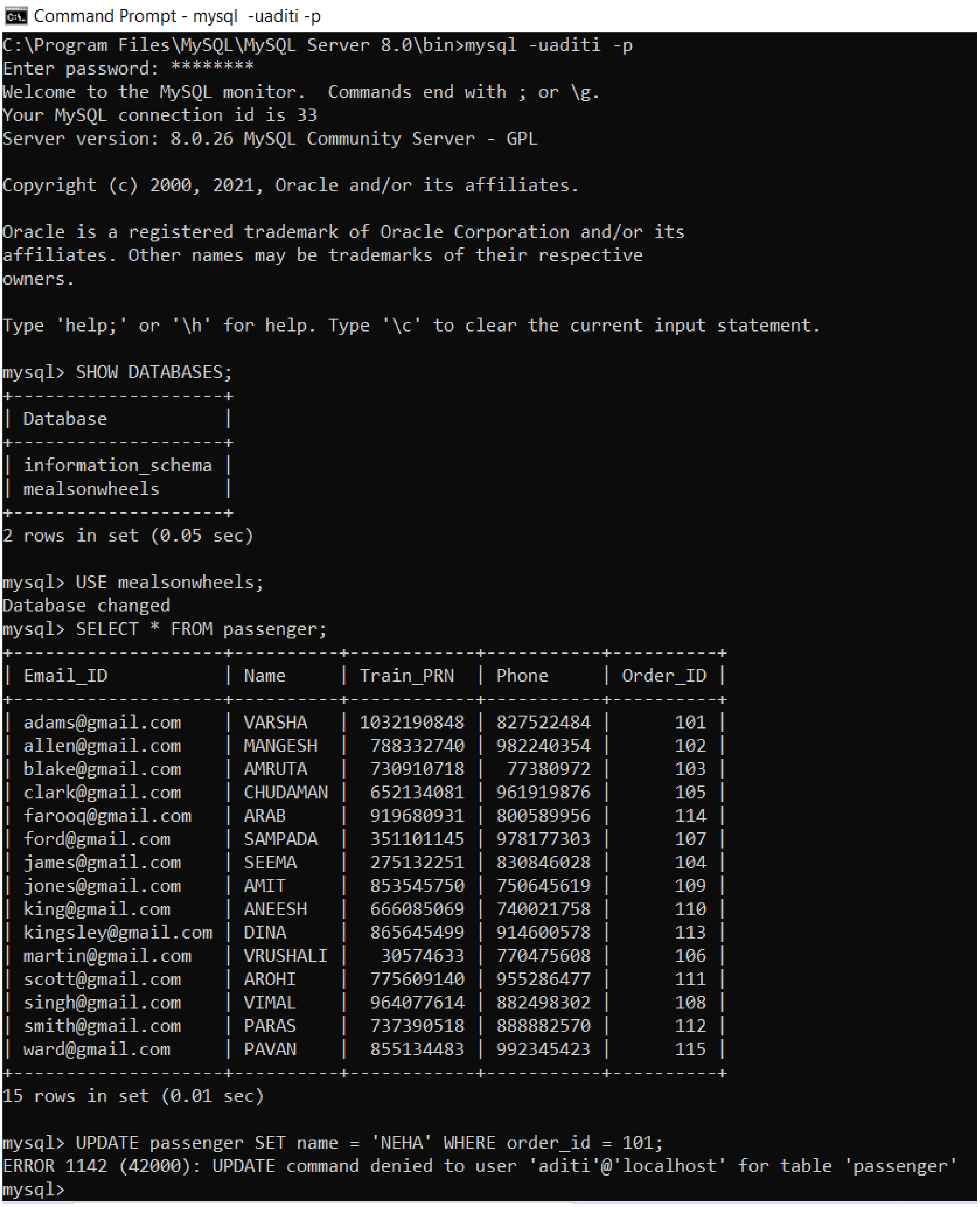
```
mysql> CREATE USER 'aditi'@'localhost' IDENTIFIED BY'aditi123';
Query OK, 0 rows affected (0.06 sec)

mysql> CREATE USER 'ankit'@'localhost' IDENTIFIED BY'ankit123';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER 'pranav'@'localhost' IDENTIFIED BY'pranav123';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER 'reuben'@'localhost' IDENTIFIED BY'reuben123';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT user FROM mysql.user;
+------------------+
| user             |
+------------------+
| Hotel_accountant |
| Hotel_manager    |
| aditi            |
| ankit            |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| pranav           |
| reuben           |
| root             |
+------------------+
10 rows in set (0.00 sec)
```

Figure 6.20: Admins created in system



```
mysql> SHOW GRANTS FOR 'aditi'@'localhost';
+-----------------------------------------------------------+
| Grants for aditi@localhost                                |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO `aditi`@`localhost`                 |
| GRANT SELECT ON `mealsonwheels`.* TO `aditi`@`localhost`  |
+-----------------------------------------------------------+
2 rows in set (0.00 sec)

mysql>
mysql> SHOW GRANTS FOR 'ankit'@'localhost';
+-------------------------------------------------------------------+
| Grants for ankit@localhost                                        |
+-------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `ankit`@`localhost`                         |
| GRANT ALL PRIVILEGES ON `mealsonwheels`.* TO `ankit`@`localhost`  |
+-------------------------------------------------------------------+
2 rows in set (0.00 sec)
mysql>
mysql> SHOW GRANTS FOR 'pranav'@'localhost';
+--------------------------------------------------------------------+
| Grants for pranav@localhost                                        |
+--------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `pranav`@`localhost`                         |
| GRANT ALL PRIVILEGES ON `mealsonwheels`.* TO `pranav`@`localhost`  |
+--------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql>
mysql> SHOW GRANTS FOR 'reuben'@'localhost';
+------------------------------------------------------------+
| Grants for reuben@localhost                                |
+------------------------------------------------------------+
| GRANT USAGE ON *.* TO `reuben`@`localhost`                 |
| GRANT SELECT ON `mealsonwheels`.* TO `reuben`@`localhost`  |
+------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> GRANT UPDATE ON mealsonwheels.* TO 'reuben'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> GRANT DELETE ON mealsonwheels.* TO 'reuben'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> SHOW GRANTS FOR 'reuben'@'localhost';
+----------------------------------------------------------------------------+
| Grants for reuben@localhost                                                |
+----------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `reuben`@`localhost`                                 |
| GRANT SELECT, UPDATE, DELETE ON `mealsonwheels`.* TO `reuben`@`localhost`  |
+----------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

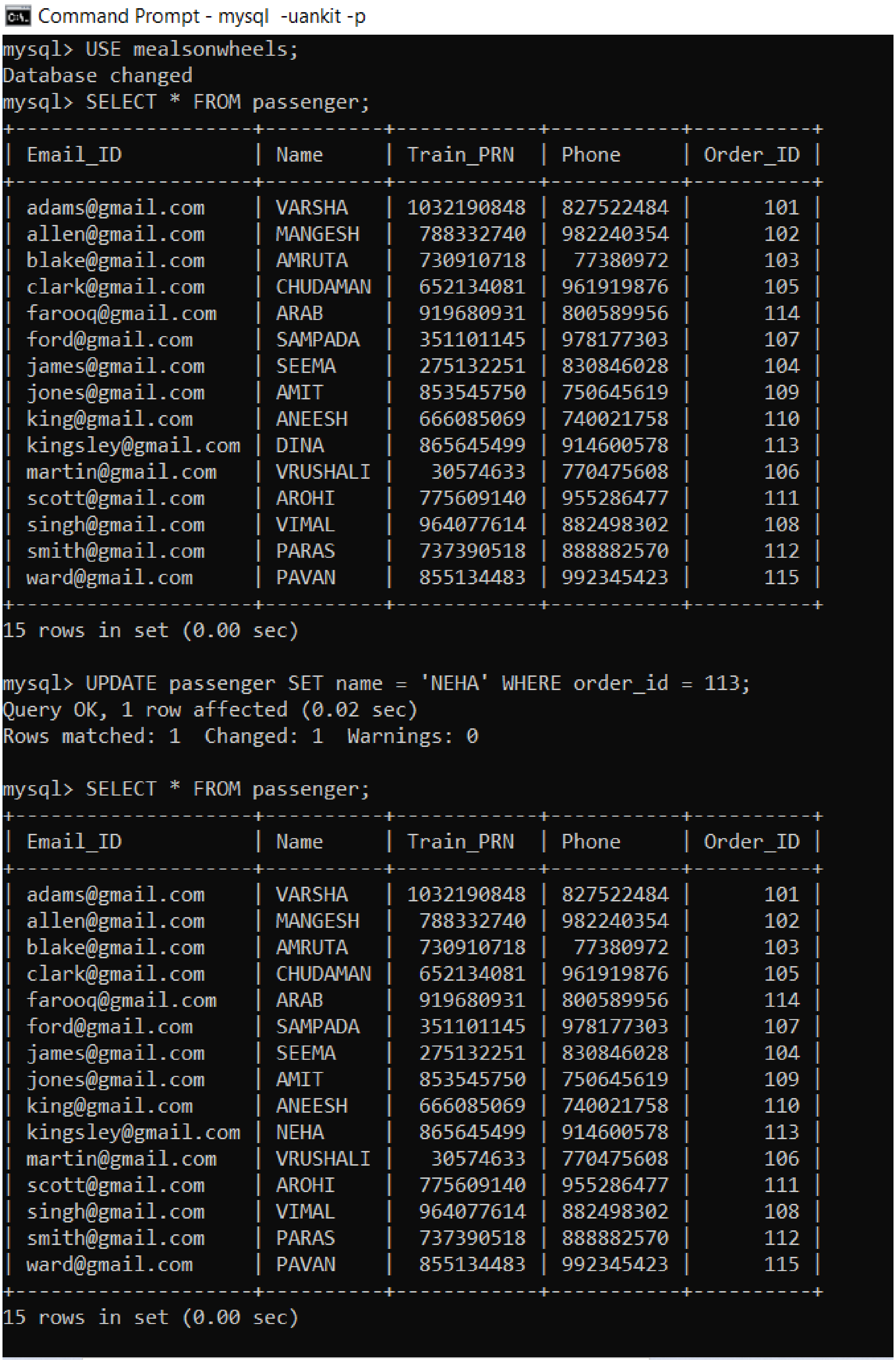
Figure 6.21: Privileges granted to each user



Command Prompt - mysql -uaditi -p

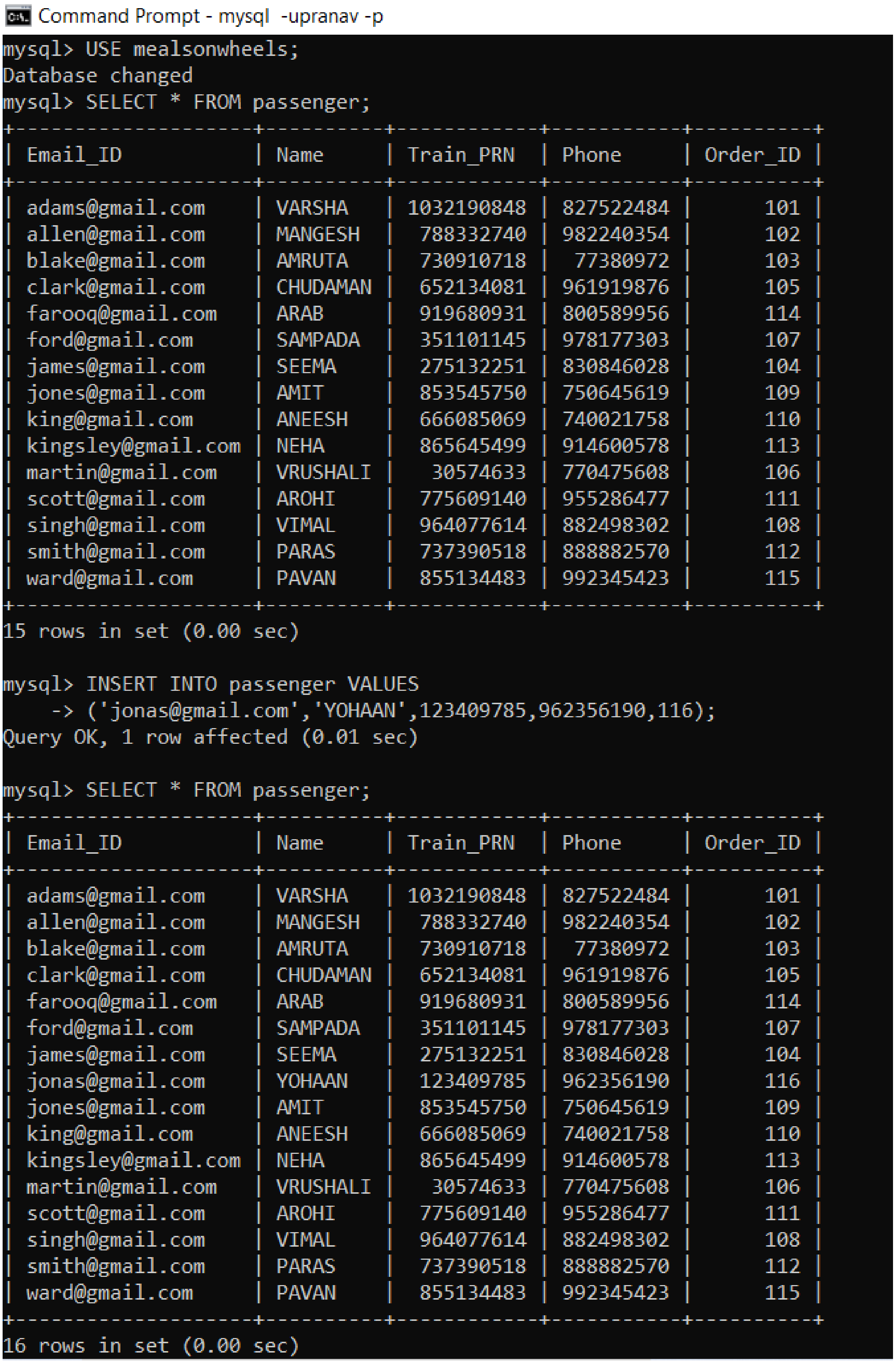
```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -uaditi -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 33
Server version: 8.0.26 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mealsonwheels      |
+--------------------+
2 rows in set (0.05 sec)

mysql> USE mealsonwheels;
Database changed
mysql> SELECT * FROM passenger;
+--------------------+----------+------------+-----------+----------+
| Email_ID           | Name     | Train_PRN  | Phone     | Order_ID |
+--------------------+----------+------------+-----------+----------+
| adams@gmail.com    | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com    | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com    | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com    | CHUDAMAN |  652134081 | 961919876 |      105 |
| farooq@gmail.com   | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com     | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com    | SEEMA    |  275132251 | 830846028 |      104 |
| jones@gmail.com    | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com     | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com | DINA     |  865645499 | 914600578 |      113 |
| martin@gmail.com   | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com    | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com    | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com    | PARAS    |  737390518 | 888882570 |      112 |
| ward@gmail.com     | PAVAN    |  855134483 | 992345423 |      115 |
+--------------------+----------+------------+-----------+----------+
15 rows in set (0.01 sec)

mysql> UPDATE passenger SET name = 'NEHA' WHERE order_id = 101;
ERROR 1142 (42000): UPDATE command denied to user 'aditi'@'localhost' for table 'passenger'
mysql>
```

Figure 6.22: Admin Aditi cannot update tables


Command Prompt - mysql -uankit -p

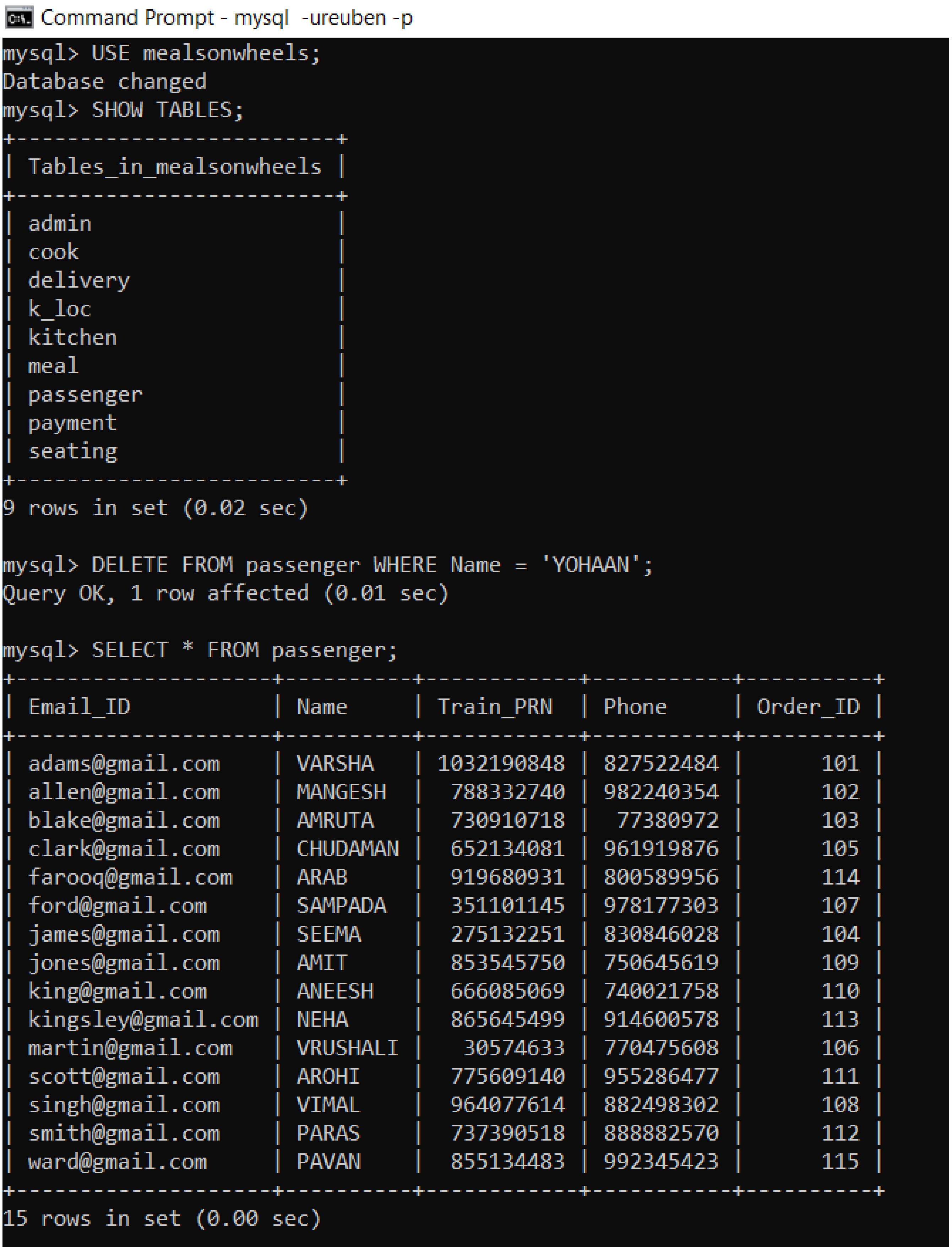
```
mysql> USE mealsonwheels;
Database changed
mysql> SELECT * FROM passenger;
+--------------------+----------+------------+-----------+----------+
| Email_ID           | Name     | Train_PRN  | Phone     | Order_ID |
+--------------------+----------+------------+-----------+----------+
| adams@gmail.com    | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com    | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com    | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com    | CHUDAMAN |  652134081 | 961919876 |      105 |
| farooq@gmail.com   | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com     | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com    | SEEMA    |  275132251 | 830846028 |      104 |
| jones@gmail.com    | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com     | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com | DINA     |  865645499 | 914600578 |      113 |
| martin@gmail.com   | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com    | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com    | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com    | PARAS    |  737390518 | 888882570 |      112 |
| ward@gmail.com     | PAVAN    |  855134483 | 992345423 |      115 |
+--------------------+----------+------------+-----------+----------+
15 rows in set (0.00 sec)

mysql> UPDATE passenger SET name = 'NEHA' WHERE order_id = 113;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM passenger;
+--------------------+----------+------------+-----------+----------+
| Email_ID           | Name     | Train_PRN  | Phone     | Order_ID |
+--------------------+----------+------------+-----------+----------+
| adams@gmail.com    | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com    | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com    | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com    | CHUDAMAN |  652134081 | 961919876 |      105 |
| farooq@gmail.com   | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com     | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com    | SEEMA    |  275132251 | 830846028 |      104 |
| jones@gmail.com    | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com     | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com | NEHA     |  865645499 | 914600578 |      113 |
| martin@gmail.com   | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com    | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com    | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com    | PARAS    |  737390518 | 888882570 |      112 |
| ward@gmail.com     | PAVAN    |  855134483 | 992345423 |      115 |
+--------------------+----------+------------+-----------+----------+
15 rows in set (0.00 sec)
```

Figure 6.23: Access provided to admin Ankit



```
Command Prompt - mysql  -upranav -p
mysql> USE mealsonwheels;
Database changed
mysql> SELECT * FROM passenger;
+--------------------+----------+------------+-----------+----------+
| Email_ID           | Name     | Train_PRN  | Phone     | Order_ID |
+--------------------+----------+------------+-----------+----------+
| adams@gmail.com    | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com    | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com    | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com    | CHUDAMAN |  652134081 | 961919876 |      105 |
| farooq@gmail.com   | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com     | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com    | SEEMA    |  275132251 | 830846028 |      104 |
| jones@gmail.com    | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com     | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com | NEHA     |  865645499 | 914600578 |      113 |
| martin@gmail.com   | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com    | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com    | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com    | PARAS    |  737390518 | 888882570 |      112 |
| ward@gmail.com     | PAVAN    |  855134483 | 992345423 |      115 |
+--------------------+----------+------------+-----------+----------+
15 rows in set (0.00 sec)

mysql> INSERT INTO passenger VALUES
    -> ('jonas@gmail.com','YOHAAN',123409785,962356190,116);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM passenger;
+--------------------+----------+------------+-----------+----------+
| Email_ID           | Name     | Train_PRN  | Phone     | Order_ID |
+--------------------+----------+------------+-----------+----------+
| adams@gmail.com    | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com    | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com    | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com    | CHUDAMAN |  652134081 | 961919876 |      105 |
| farooq@gmail.com   | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com     | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com    | SEEMA    |  275132251 | 830846028 |      104 |
| jonas@gmail.com    | YOHAAN   |  123409785 | 962356190 |      116 |
| jones@gmail.com    | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com     | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com | NEHA     |  865645499 | 914600578 |      113 |
| martin@gmail.com   | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com    | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com    | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com    | PARAS    |  737390518 | 888882570 |      112 |
| ward@gmail.com     | PAVAN    |  855134483 | 992345423 |      115 |
+--------------------+----------+------------+-----------+----------+
16 rows in set (0.00 sec)
```

Figure 6.24: Access provided to admin Pranav


Command Prompt - mysql -ureuben -p

```
mysql> USE mealsonwheels;
Database changed
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_mealsonwheels |
+-------------------------+
| admin                   |
| cook                    |
| delivery                |
| k_loc                   |
| kitchen                 |
| meal                    |
| passenger               |
| payment                 |
| seating                 |
+-------------------------+
9 rows in set (0.02 sec)

mysql> DELETE FROM passenger WHERE Name = 'YOHAAN';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM passenger;
+---------------------+----------+------------+-----------+----------+
| Email_ID            | Name     | Train_PRN  | Phone     | Order_ID |
+---------------------+----------+------------+-----------+----------+
| adams@gmail.com     | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com     | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com     | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com     | CHUDAMAN |  652134081 | 961919876 |      105 |
| farooq@gmail.com    | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com      | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com     | SEEMA    |  275132251 | 830846028 |      104 |
| jones@gmail.com     | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com      | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com  | NEHA     |  865645499 | 914600578 |      113 |
| martin@gmail.com    | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com     | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com     | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com     | PARAS    |  737390518 | 888882570 |      112 |
| ward@gmail.com      | PAVAN    |  855134483 | 992345423 |      115 |
+---------------------+----------+------------+-----------+----------+
15 rows in set (0.00 sec)
```

Figure 6.25: Access provided to admin Reuben

6.4 PL SQL

SQL supports the use of procedural programming data structures, such as procedures, functions, triggers and cursors, this is known as PL SQL [10]. Every PL SQL block has a mandatory BEGIN and END section, but other parts are optional. PL SQL blocks enable users to perform a series of operations, simultaneously instead of single query execution. PL SQL triggers offer automation of database updates, while procedures and functions ensure referential integrity in database. Some important PL SQL blocks in our system are illustrated below.

Subprograms - Procedures and functions

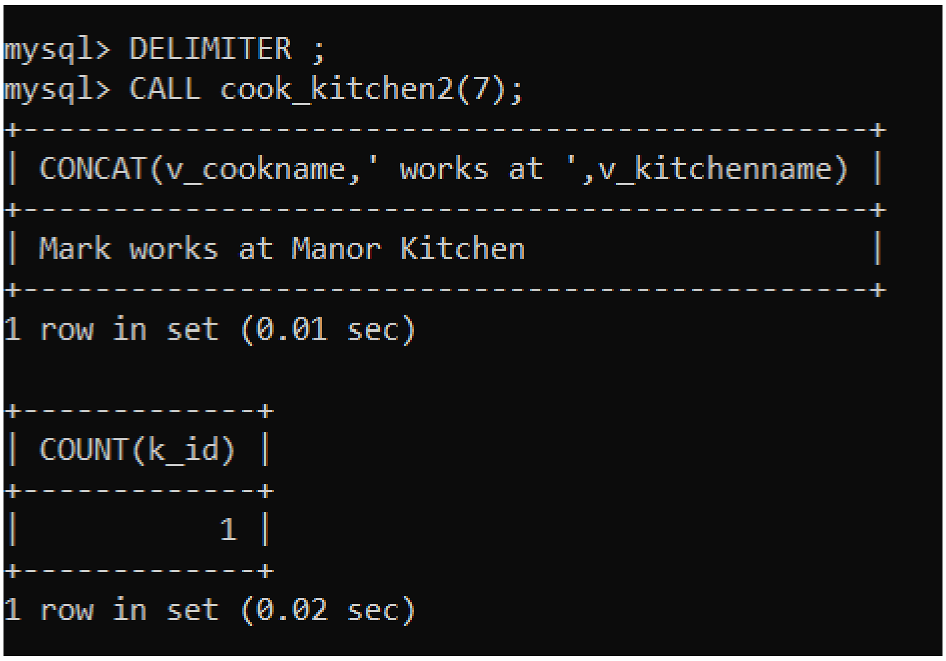
PL SQL subprograms we have used are:

6.4.1 Procedures

Procedures [11] are subprograms in MySQL that do not have a return value. They have 3 types of modes for parameters – IN for input variable, OUT for output variable and INOUT for same parameter as input and output. In our program, we have defined the following procedures using cursors. Cursors [12] are similar to pointers, referring to a context area or region from which records are retrieved.

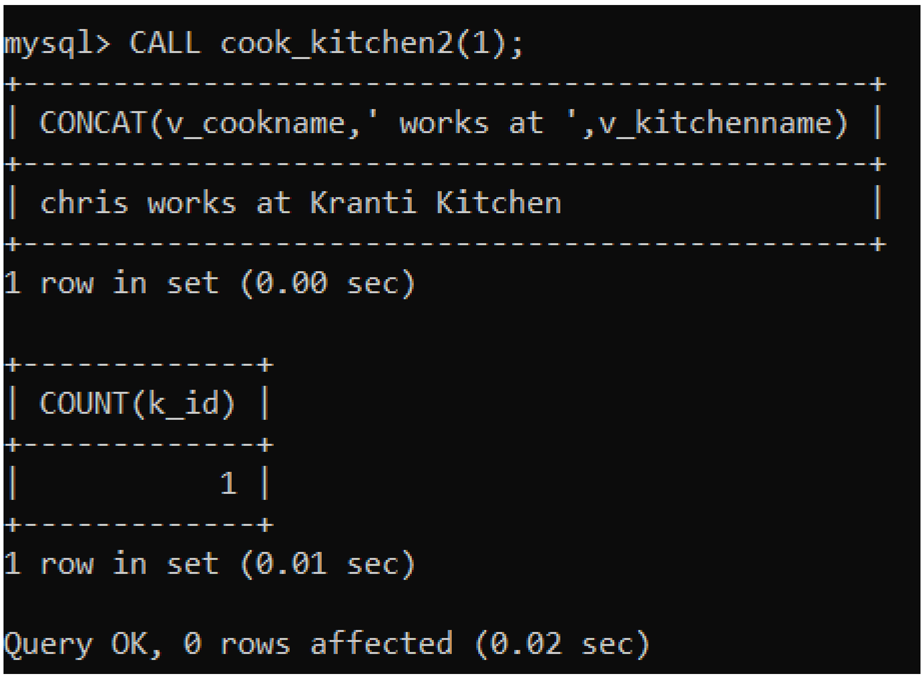
Procedure cook_kitchen2 is used to display name of cook and the kitchen they work at, when input parameter is kitchen id. It also retrieves count of cooks at given kitchen id, as depicted in figures 6.25 and 6.26.

```
DELIMITER #
CREATE procedure cook_kitchen2(IN kid INT)
BEGIN
/*Local variables*/
DECLARE v_finished INT;
DECLARE v_kitchenid INT;
DECLARE v_cookid INT;
DECLARE v_cookname VARCHAR(20);
DECLARE v_kitchenname VARCHAR(20);
/*Cursor 1 gets cook name while cursor 2 gets kitchen name*/
DECLARE c1 CURSOR FOR SELECT cook_id, name, k_id FROM cook;
DECLARE c2 CURSOR FOR SELECT name FROM kitchen;
DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET v_finished = 1;
OPEN c1;
OPEN c2;
cookid: LOOP
FETCH c1 INTO v_cookid, v_cookname, v_kitchenid;
FETCH c2 INTO v_kitchenname;


/*Exit loop condition*/

IF v_finished = 1 THEN

LEAVE cookid;

END IF;

IF v_kitchenid = kid THEN

SELECT CONCAT(v_cookname,' works at ',v_kitchenname);

END IF;

END LOOP;

SELECT COUNT(k_id) FROM cook WHERE k_id = kid;

CLOSE c1;

CLOSE c2;

END#

DELIMITER ;

CALL cook_kitchen2(7);

CALL cook_kitchen2(1);
```



```
mysql> DELIMITER ;
mysql> CALL cook_kitchen2(7);
+-------------------------------------------+
| CONCAT(v_cookname,' works at ',v_kitchenname) |
+-------------------------------------------+
| Mark works at Manor Kitchen               |
+-------------------------------------------+
1 row in set (0.01 sec)

+-------------+
| COUNT(k_id) |
+-------------+
|           1 |
+-------------+
1 row in set (0.02 sec)
```

Figure 6.25: Procedure to retrieve cook name and kitchen name for kitchen id = 7



```
mysql> CALL cook_kitchen2(1);
+-------------------------------------------------+
| CONCAT(v_cookname,' works at ',v_kitchenname) |
+-------------------------------------------------+
| chris works at Kranti Kitchen                   |
+-------------------------------------------------+
1 row in set (0.00 sec)

+-------------+
| COUNT(k_id) |
+-------------+
|           1 |
+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

Figure 6.26: Procedure to retrieve cook name and kitchen name for kitchen id = 1

Procedure meal_details retrieve meal id; type of meal and kitchen meal is cooked by. The results have been depicted in figure 6.27.

DROP PROCEDURE IF EXISTS meal_details;

DELIMITER $

CREATE PROCEDURE meal_details(IN mid INT)

BEGIN

/*Local variables*/

DECLARE v_finished INT;

DECLARE v_mealid INT;

DECLARE v_req VARCHAR(30);

DECLARE v_type VARCHAR(10);

DECLARE v_cost FLOAT;

DECLARE v_quantity INT;

DECLARE v_kid INT;

DECLARE v_kname VARCHAR(20);

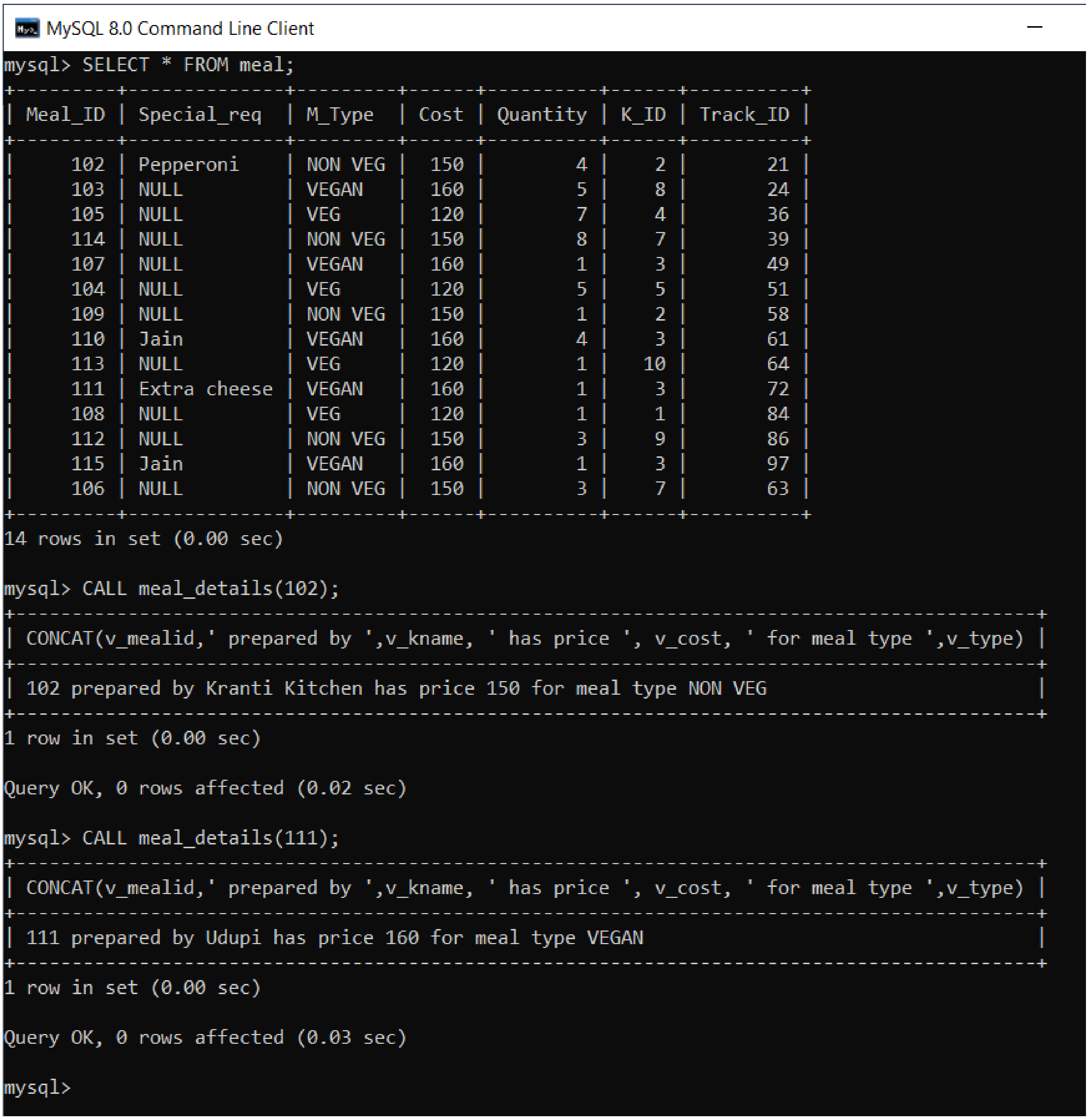
DECLARE v_track INT;

/*Cursor 1 gets cook name while cursor 2 gets kitchen name*/

DECLARE c1 CURSOR FOR SELECT Meal_ID, Special_req, M_Type, Cost, Quantity, track_id FROM meal;

DECLARE c2 CURSOR FOR SELECT kitchen_id, name FROM kitchen;

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
                    SET v_finished = 1;
OPEN c1;
OPEN c2;
mealinfo: LOOP
FETCH c1 INTO v_mealid, v_req, v_type, v_cost, v_quantity, v_track;
FETCH c2 INTO v_kid, v_kname;
/*Exit loop condition*/
 IF v_finished = 1 THEN
LEAVE mealinfo;
END IF;
IF v_mealid = mid THEN
SELECT CONCAT(v_mealid,' prepared by ',v_kname, ' has price ', v_cost, ' for meal type ',v_type);
END IF;
END LOOP;
CLOSE c1;
CLOSE c2;
END $
DELIMITER ;
CALL meal_details(102);
CALL meal_details(111);
```


MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM meal;
+---------+---------------+---------+------+----------+------+----------+
| Meal_ID | Special_req   | M_Type  | Cost | Quantity | K_ID | Track_ID |
+---------+---------------+---------+------+----------+------+----------+
|     102 | Pepperoni     | NON VEG |  150 |        4 |    2 |       21 |
|     103 | NULL          | VEGAN   |  160 |        5 |    8 |       24 |
|     105 | NULL          | VEG     |  120 |        7 |    4 |       36 |
|     114 | NULL          | NON VEG |  150 |        8 |    7 |       39 |
|     107 | NULL          | VEGAN   |  160 |        1 |    3 |       49 |
|     104 | NULL          | VEG     |  120 |        5 |    5 |       51 |
|     109 | NULL          | NON VEG |  150 |        1 |    2 |       58 |
|     110 | Jain          | VEGAN   |  160 |        4 |    3 |       61 |
|     113 | NULL          | VEG     |  120 |        1 |   10 |       64 |
|     111 | Extra cheese  | VEGAN   |  160 |        1 |    3 |       72 |
|     108 | NULL          | VEG     |  120 |        1 |    1 |       84 |
|     112 | NULL          | NON VEG |  150 |        3 |    9 |       86 |
|     115 | Jain          | VEGAN   |  160 |        1 |    3 |       97 |
|     106 | NULL          | NON VEG |  150 |        3 |    7 |       63 |
+---------+---------------+---------+------+----------+------+----------+
14 rows in set (0.00 sec)

mysql> CALL meal_details(102);
+-------------------------------------------------------------------------------------------+
| CONCAT(v_mealid,' prepared by ',v_kname, ' has price ', v_cost, ' for meal type ',v_type) |
+-------------------------------------------------------------------------------------------+
| 102 prepared by Kranti Kitchen has price 150 for meal type NON VEG                        |
+-------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.02 sec)

mysql> CALL meal_details(111);
+-------------------------------------------------------------------------------------------+
| CONCAT(v_mealid,' prepared by ',v_kname, ' has price ', v_cost, ' for meal type ',v_type) |
+-------------------------------------------------------------------------------------------+
| 111 prepared by Udupi has price 160 for meal type VEGAN                                   |
+-------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.03 sec)

mysql>
```

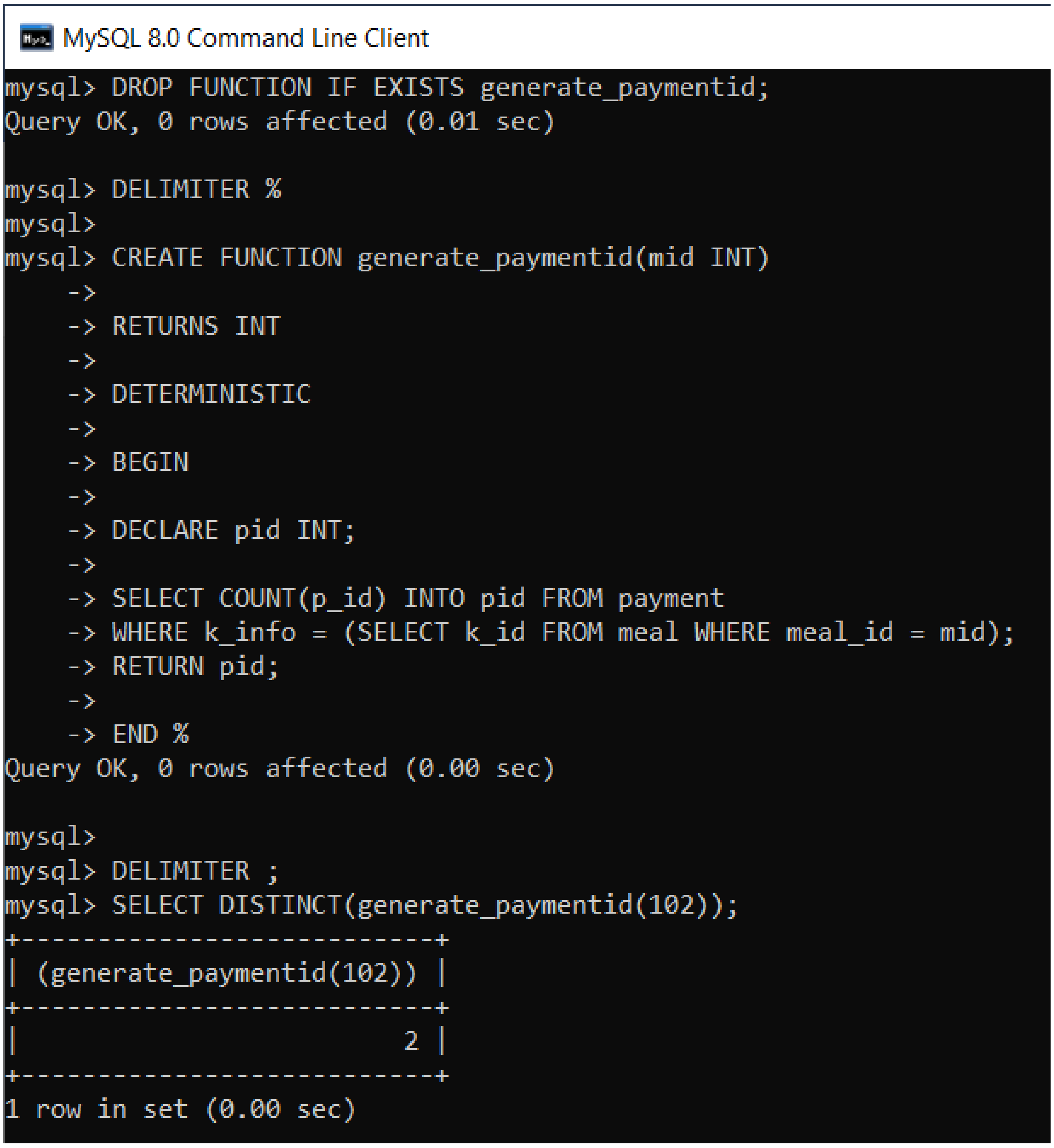
Figure 6.27: Procedure to get data for meal ids = 102 and 111

6.4.2 Functions

Functions [13] in PL SQL are stored lines of code that perform certain operations and return a value. In MySQL, there are 2 types of functions – inbuilt and user defined. Inbuilt functions exist in the system, such as MAX, AVG, SUM etc. and they are known as aggregate functions. User defined functions can be defined by the user to perform specific operations.

Function generate_paymentid takes meal id as input and displays count of meals ordered. The results have been shown in figure 6.28.

```
DROP FUNCTION IF EXISTS generate_paymentid;

DELIMITER %

CREATE FUNCTION generate_paymentid(mid INT)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE pid INT;

SELECT COUNT(p_id) INTO pid FROM payment

WHERE k_info = (SELECT k_id FROM meal WHERE meal_id = mid);

RETURN pid;

END %

DELIMITER ;

SELECT DISTINCT(generate_paymentid(102));

FROM payment p;

SELECT DISTINCT(generate_paymentid(111)), p.p_mode, p.p_date FROM payment p;
```

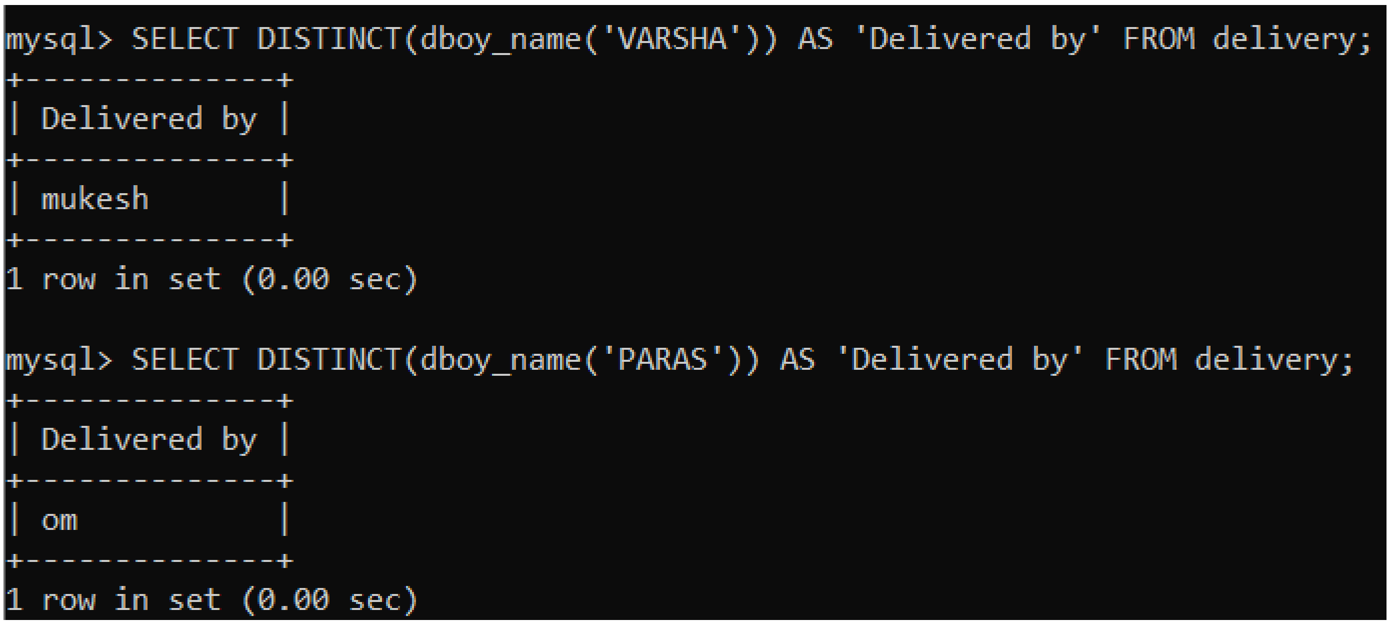

MySQL 8.0 Command Line Client

```
mysql> DROP FUNCTION IF EXISTS generate_paymentid;
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER %
mysql>
mysql> CREATE FUNCTION generate_paymentid(mid INT)
    ->
    -> RETURNS INT
    ->
    -> DETERMINISTIC
    ->
    -> BEGIN
    ->
    -> DECLARE pid INT;
    ->
    -> SELECT COUNT(p_id) INTO pid FROM payment
    -> WHERE k_info = (SELECT k_id FROM meal WHERE meal_id = mid);
    -> RETURN pid;
    ->
    -> END %
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT DISTINCT(generate_paymentid(102));
+----------------------------+
| (generate_paymentid(102)) |
+----------------------------+
|                          2 |
+----------------------------+
1 row in set (0.00 sec)
```

Figure 6.28: Function to generate count of payments made for given meal id

Function dboy_name is used to display name of delivery boy responsible for delivery of food to name of passenger given as input to the function. Output of function has been depicted in figure 6.29.

```
DROP FUNCTION IF EXISTS dboy_name;

DELIMITER $

CREATE FUNCTION dboy_name(pname VARCHAR(15))

RETURNS VARCHAR(15)

DETERMINISTIC

BEGIN

DECLARE dname VARCHAR(15);

SELECT delivery.D_Name

INTO dname

FROM delivery JOIN passenger

ON delivery.Pass_Email = passenger.Email_ID

WHERE Pass_Email=(SELECT Email_ID FROM passenger

            WHERE name=pname);

RETURN dname;

END $

DELIMITER ;

SELECT DISTINCT(dboy_name('VARSHA')) AS 'Delivered by' FROM delivery;

SELECT DISTINCT(dboy_name('PARAS')) AS 'Delivered by' FROM delivery;
```

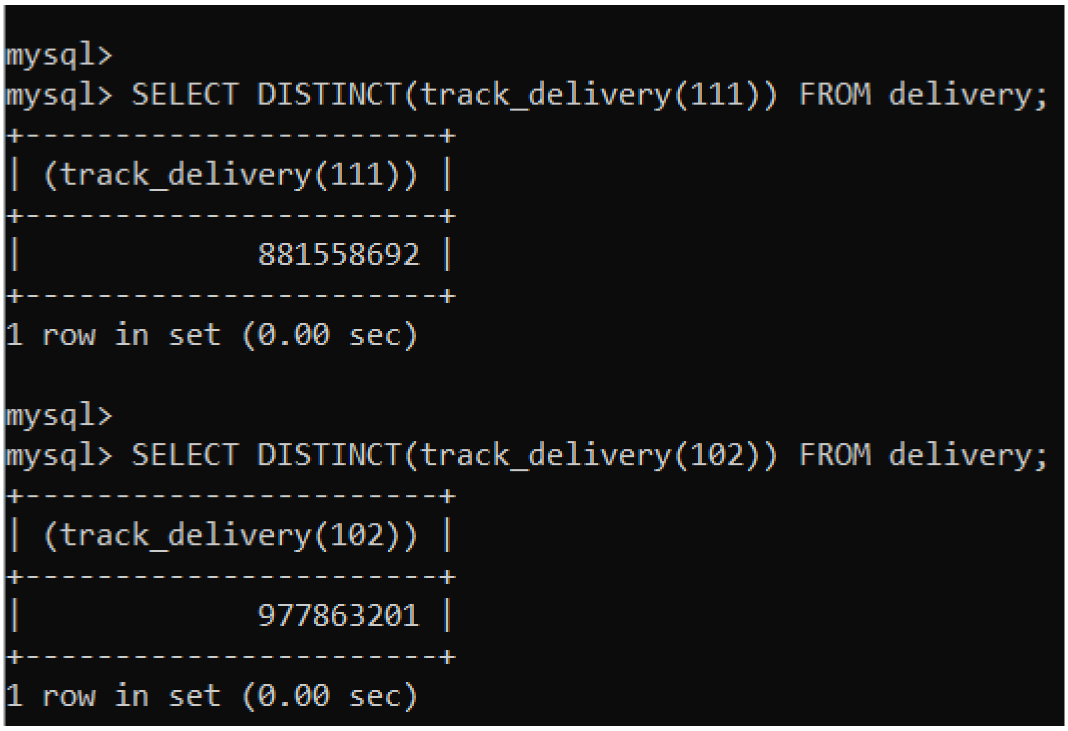


```
mysql> SELECT DISTINCT(dboy_name('VARSHA')) AS 'Delivered by' FROM delivery;
+--------------+
| Delivered by |
+--------------+
| mukesh       |
+--------------+
1 row in set (0.00 sec)

mysql> SELECT DISTINCT(dboy_name('PARAS')) AS 'Delivered by' FROM delivery;
+--------------+
| Delivered by |
+--------------+
| om           |
+--------------+
1 row in set (0.00 sec)
```

Figure 6.29: Function to display name of delivery boy for a given passenger name

Function track_delivery is a deterministic function used to retrieve contact number of delivery boy, when meal id is given as input to the function. Results have been illustrated in figure 6.30.

```
DROP FUNCTION IF EXISTS track_delivery;

DELIMITER ?

CREATE FUNCTION track_delivery(mid INT)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE dphone INT;

SELECT delivery.D_Phone into dphone

FROM delivery JOIN meal ON delivery.Tracking_ID=meal.Track_ID

WHERE delivery.Tracking_ID= (SELECT Track_ID FROM meal

                        WHERE Meal_ID= mid);

RETURN dphone;

END ?

DELIMITER ;

SELECT DISTINCT(track_delivery(111)) FROM delivery;

SELECT DISTINCT(track_delivery(102)) FROM delivery;
```

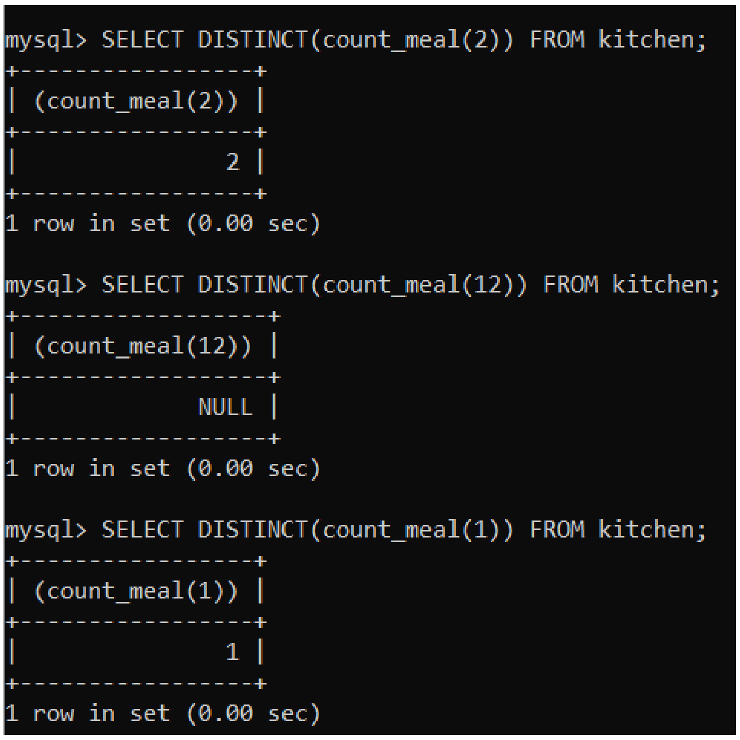


```
mysql>
mysql> SELECT DISTINCT(track_delivery(111)) FROM delivery;
+-----------------------+
| (track_delivery(111)) |
+-----------------------+
|             881558692 |
+-----------------------+
1 row in set (0.00 sec)

mysql>
mysql> SELECT DISTINCT(track_delivery(102)) FROM delivery;
+-----------------------+
| (track_delivery(102)) |
+-----------------------+
|             977863201 |
+-----------------------+
1 row in set (0.00 sec)
```

Figure 6.30: Function to display name of delivery contact number for a given meal id

Function count_meal is used to return the no of meals booked for a given meal id. Output of the function in figure 6.31 shows that there is 1 meal each booked for meal id 2 and 1, but none for meal id 12.

```
DROP FUNCTION IF EXISTS count_meal;

DELIMITER //

CREATE FUNCTION count_meal(kid INT)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE abc INT;

SELECT COUNT(meal.k_id) INTO abc FROM meal

WHERE meal.k_id = kid

GROUP BY meal.k_id;

RETURN abc;

END //

DELIMITER ;

SELECT DISTINCT(count_meal(10)) FROM kitchen k;

SELECT DISTINCT(count_meal(2)) FROM kitchen;

SELECT DISTINCT(count_meal(12)) FROM kitchen;
```



```
mysql> SELECT DISTINCT(count_meal(2)) FROM kitchen;
+-----------------+
| (count_meal(2)) |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DISTINCT(count_meal(12)) FROM kitchen;
+------------------+
| (count_meal(12)) |
+------------------+
|             NULL |
+------------------+
1 row in set (0.00 sec)

mysql> SELECT DISTINCT(count_meal(1)) FROM kitchen;
+-----------------+
| (count_meal(1)) |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)
```

Figure 6.31: Function to display count of meals for given meal id

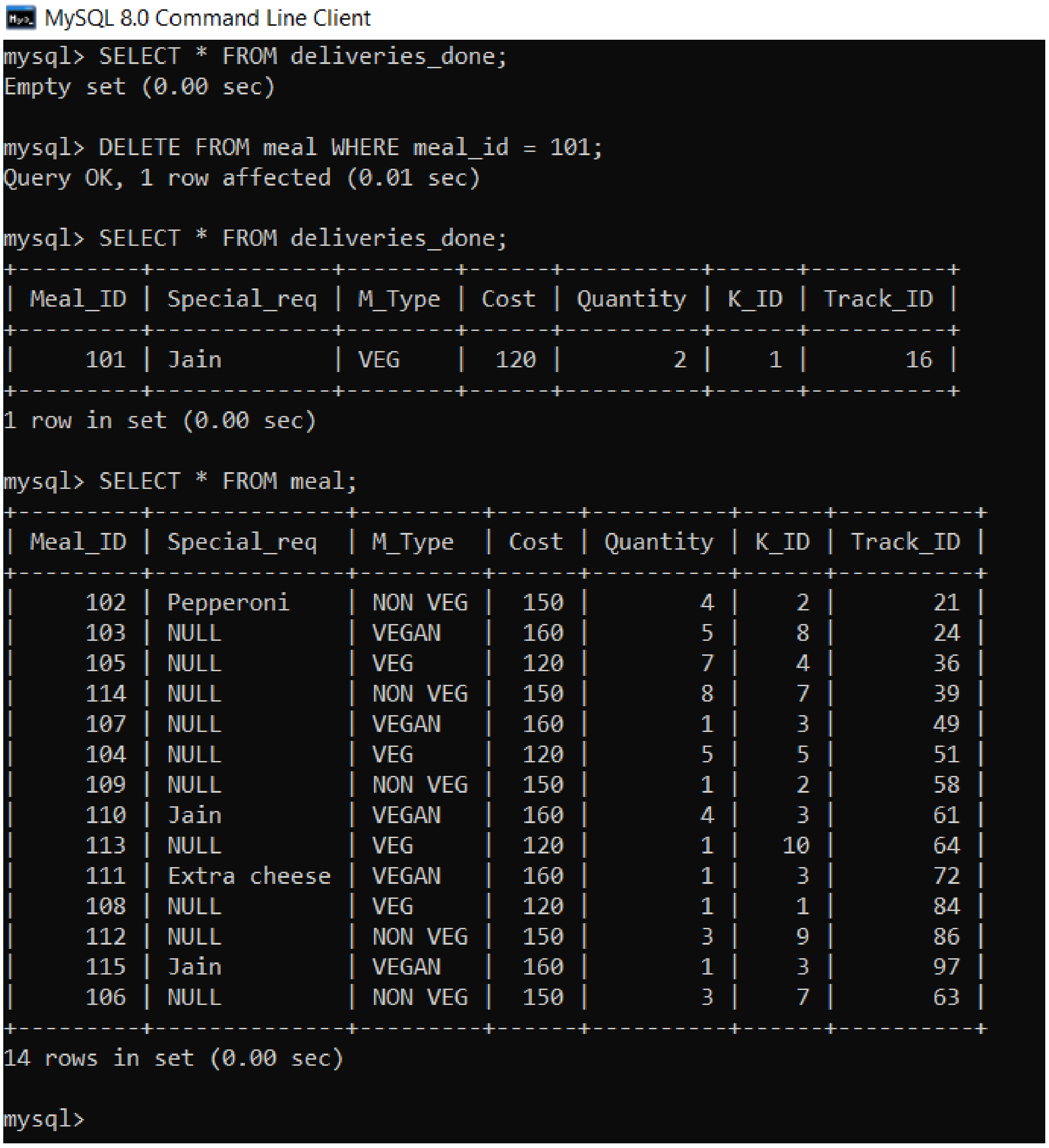
6.4.3 Triggers

A trigger [14] is an automated function that calls itself when an update is done on a table. It has 2 options – BEFORE and AFTER for 3 commands – INSERT, DELETE and UPDATE. Triggers thus have 6 modes of operation. They are useful in reducing ambiguity in database and allowing backups from time to time.

Here we have created a table deliveres_done to store details of all meals that have been sent to the passengers. We move food sent into deliveries_done after deleting from meal table.

```
CREATE TABLE deliveries_done AS

SELECT * FROM meal;

DELETE FROM deliveries_done;

DELIMITER $

CREATE TRIGGER del_over

AFTER DELETE

ON meal

FOR EACH ROW

BEGIN

INSERT INTO deliveries_done VALUES(

OLD.Meal_ID,

OLD.Special_req,

OLD.M_Type,

OLD.Cost,

OLD.Quantity,

OLD.K_ID,

OLD.Track_ID);

END$

DELIMITER ;

SELECT * FROM meal;

SELECT * FROM deliveries_done;
```

The result of trigger has been displayed in figure 6.32.


MySQL 8.0 Command Line Client

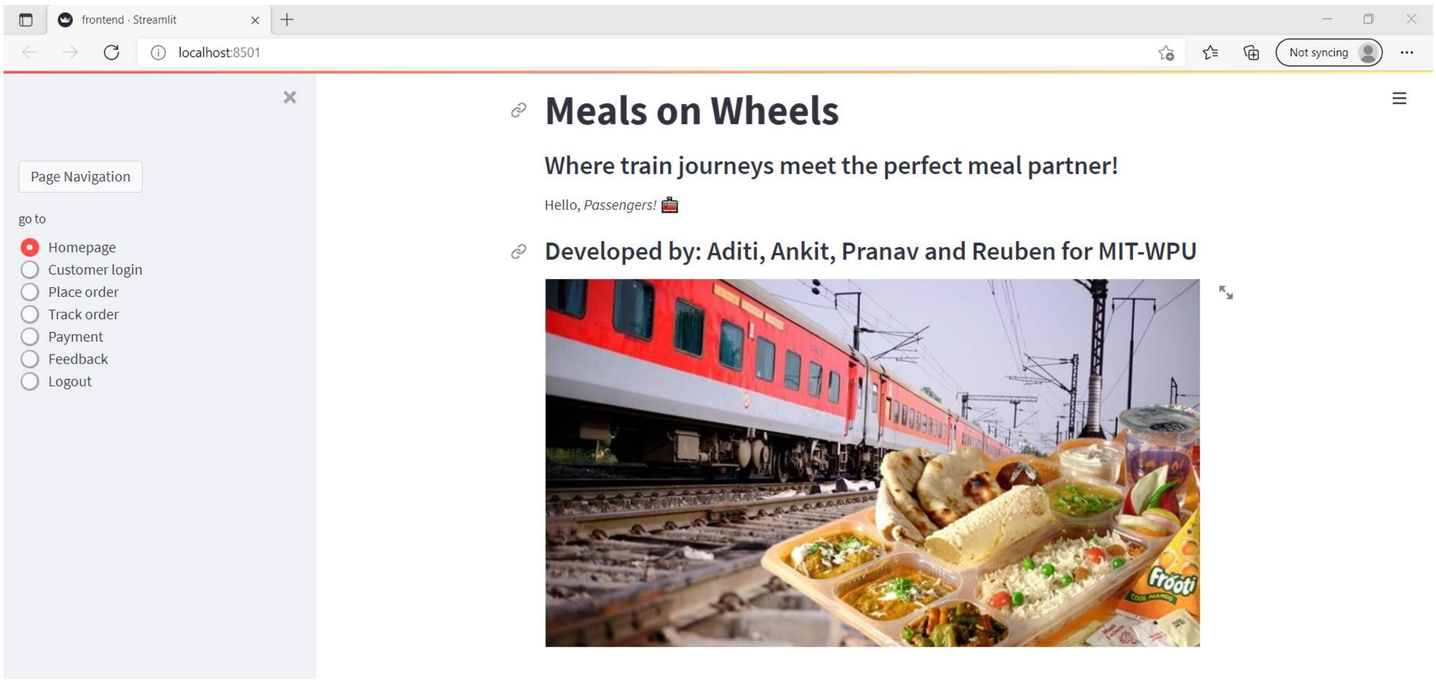
```
mysql> SELECT * FROM deliveries_done;
Empty set (0.00 sec)

mysql> DELETE FROM meal WHERE meal_id = 101;
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM deliveries_done;
+---------+-------------+--------+------+----------+------+----------+
| Meal_ID | Special_req | M_Type | Cost | Quantity | K_ID | Track_ID |
+---------+-------------+--------+------+----------+------+----------+
|     101 | Jain        | VEG    |  120 |        2 |    1 |       16 |
+---------+-------------+--------+------+----------+------+----------+
1 row in set (0.00 sec)

mysql> SELECT * FROM meal;
+---------+--------------+---------+------+----------+------+----------+
| Meal_ID | Special_req  | M_Type  | Cost | Quantity | K_ID | Track_ID |
+---------+--------------+---------+------+----------+------+----------+
|     102 | Pepperoni    | NON VEG |  150 |        4 |    2 |       21 |
|     103 | NULL         | VEGAN   |  160 |        5 |    8 |       24 |
|     105 | NULL         | VEG     |  120 |        7 |    4 |       36 |
|     114 | NULL         | NON VEG |  150 |        8 |    7 |       39 |
|     107 | NULL         | VEGAN   |  160 |        1 |    3 |       49 |
|     104 | NULL         | VEG     |  120 |        5 |    5 |       51 |
|     109 | NULL         | NON VEG |  150 |        1 |    2 |       58 |
|     110 | Jain         | VEGAN   |  160 |        4 |    3 |       61 |
|     113 | NULL         | VEG     |  120 |        1 |   10 |       64 |
|     111 | Extra cheese | VEGAN   |  160 |        1 |    3 |       72 |
|     108 | NULL         | VEG     |  120 |        1 |    1 |       84 |
|     112 | NULL         | NON VEG |  150 |        3 |    9 |       86 |
|     115 | Jain         | VEGAN   |  160 |        1 |    3 |       97 |
|     106 | NULL         | NON VEG |  150 |        3 |    7 |       63 |
+---------+--------------+---------+------+----------+------+----------+
14 rows in set (0.00 sec)

mysql>
```

Figure 6.32: Trigger to update delivered meals

7. Frontend GUI screenshots

To connect our MySQL database to python frontend, we used MySQL connector [15] library. The pages – Homepage, customer login, place order, make payment, track order and give feedback have been illustrated in figures 7.1 onwards.

Homepage is the welcome page to our site. It displays names of members and purpose of developing such a system.



Meals on Wheels

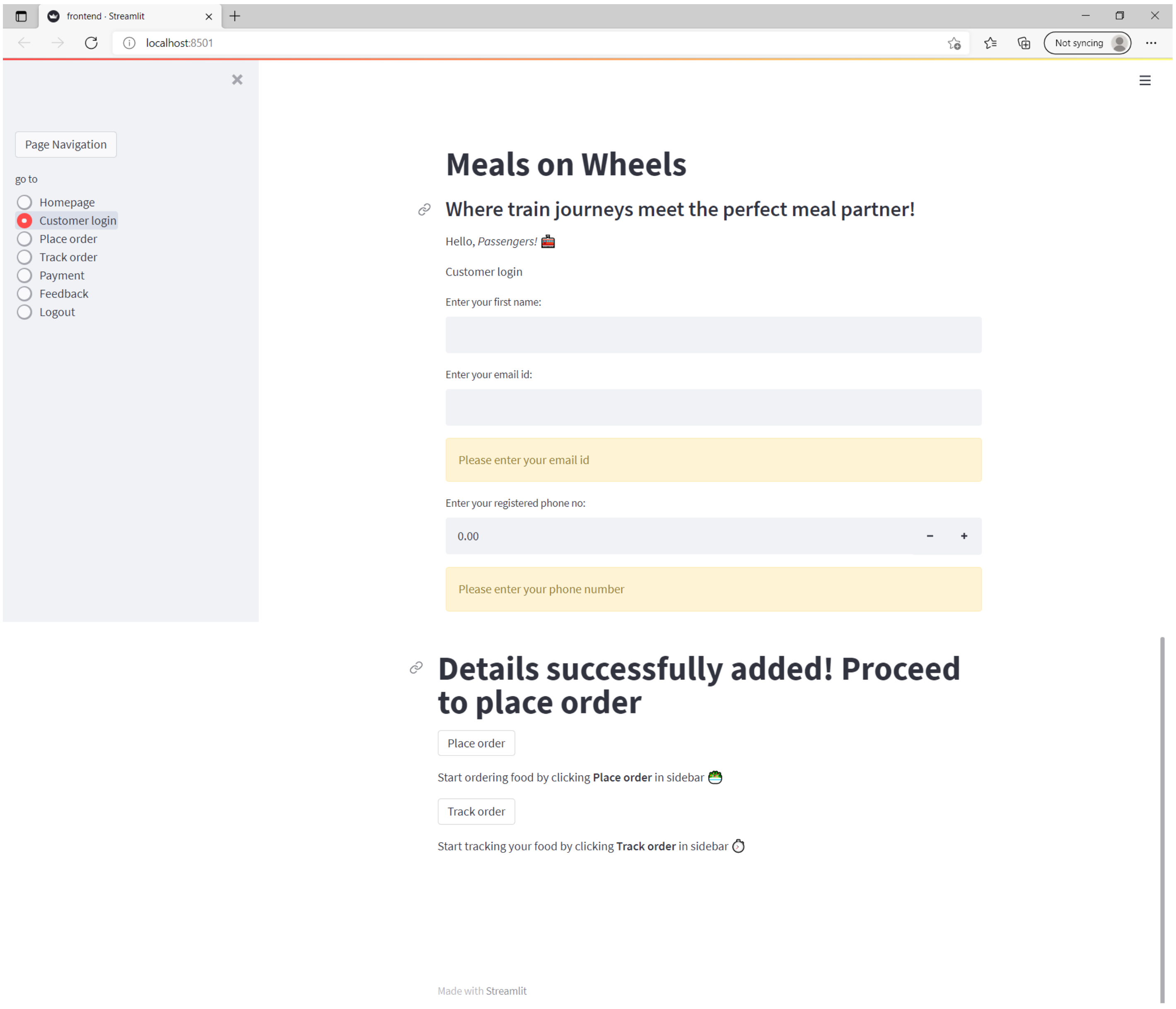
Where train journeys meet the perfect meal partner!

Hello, *Passengers!*

Developed by: Aditi, Ankit, Pranav and Reuben for MIT-WPU

Figure 7.1: Homepage

Customer must enter their name, email id, train PRN number and name to place meal order. Customer login has been displayed in figure 7.2. Amit Sorenson's records have been added, as depicted in figure 7.3 and inserted record can be viewed in figure 7.4



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!*

Customer login

Enter your first name:

Enter your email id:

Please enter your email id

Enter your registered phone no:

0.00

Please enter your phone number

Details successfully added! Proceed to place order

Place order

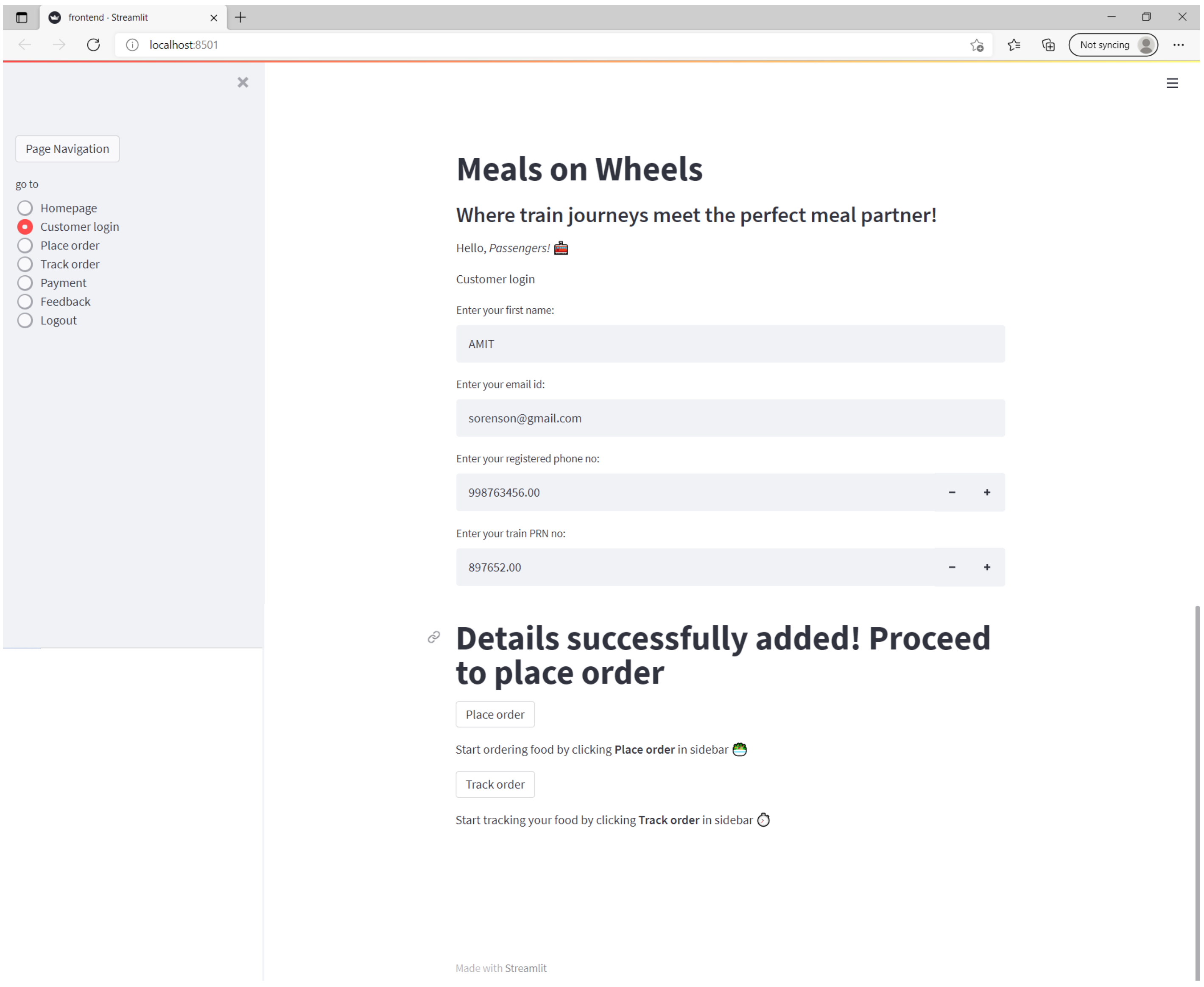
Start ordering food by clicking **Place order** in sidebar

Track order

Start tracking your food by clicking **Track order** in sidebar

Made with Streamlit

Figure 7.2: Default customer login page



frontend · Streamlit

localhost:8501

Page Navigation

go to

- Homepage
- [✓ Customer login]
- Place order
- Track order
- Payment
- Feedback
- Logout

Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!*

Customer login

Enter your first name:	AMIT
Enter your email id:	sorenson@gmail.com
Enter your registered phone no:	998763456.00
Enter your train PRN no:	897652.00

Details successfully added! Proceed to place order

Place order

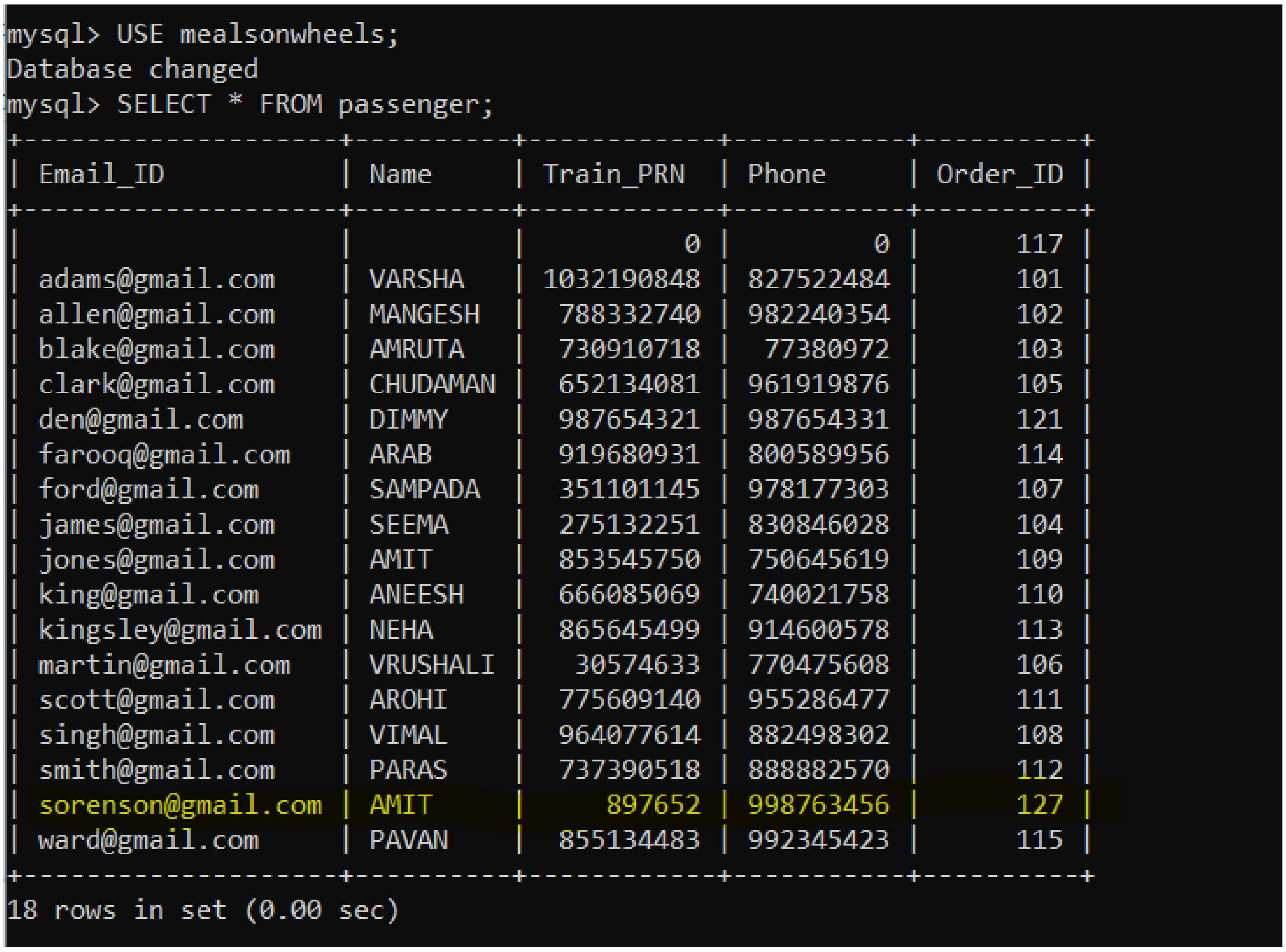
Start ordering food by clicking **Place order** in sidebar

Track order

Start tracking your food by clicking **Track order** in sidebar

Made with Streamlit

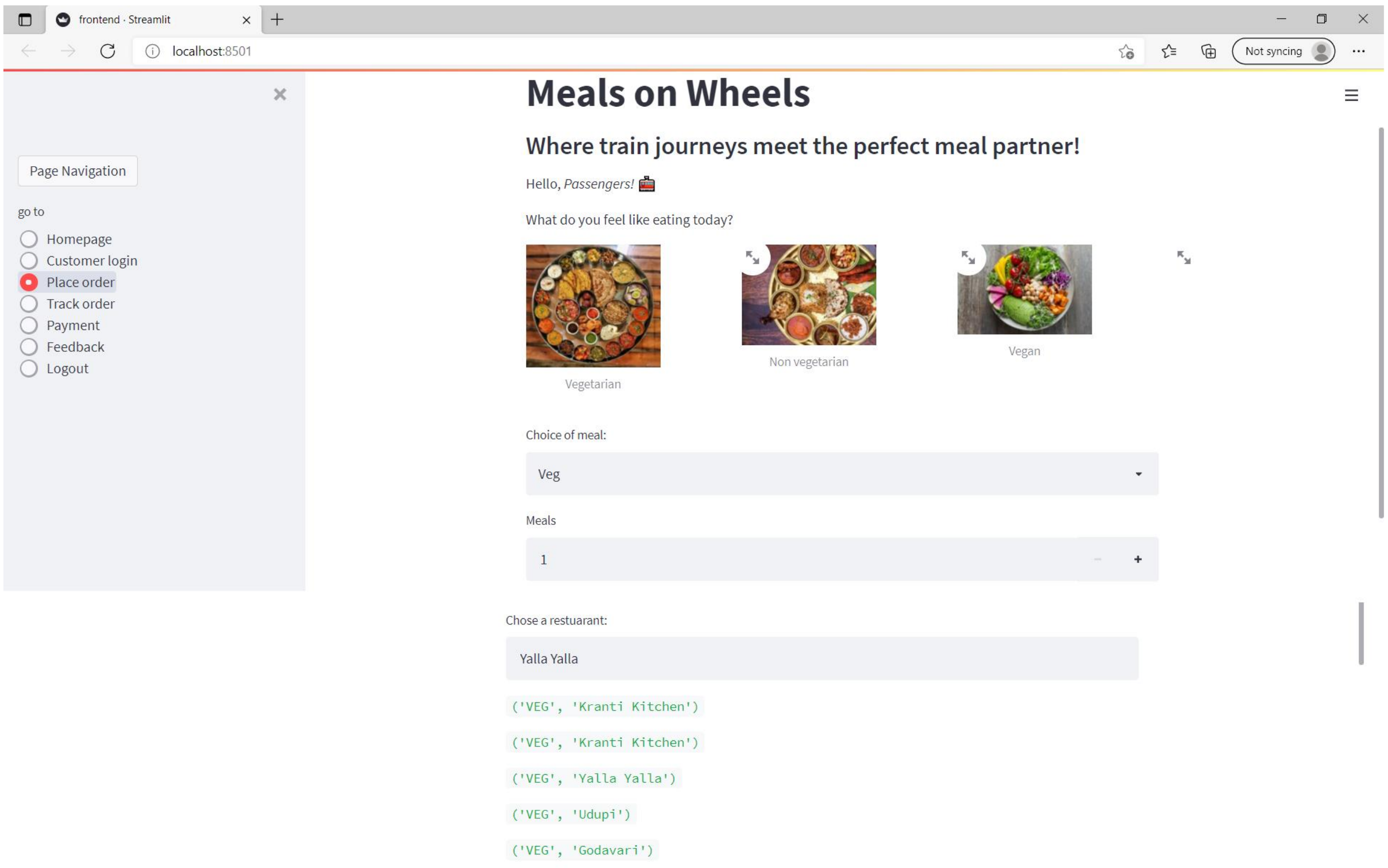
Figure 7.3: Registration of new passenger



```
mysql> USE mealsonwheels;
Database changed
mysql> SELECT * FROM passenger;
+--------------------+----------+------------+-----------+----------+
| Email_ID           | Name     | Train_PRN  | Phone     | Order_ID |
+--------------------+----------+------------+-----------+----------+
|                    |          |          0 |         0 |      117 |
| adams@gmail.com    | VARSHA   | 1032190848 | 827522484 |      101 |
| allen@gmail.com    | MANGESH  |  788332740 | 982240354 |      102 |
| blake@gmail.com    | AMRUTA   |  730910718 |  77380972 |      103 |
| clark@gmail.com    | CHUDAMAN |  652134081 | 961919876 |      105 |
| den@gmail.com      | DIMMY    |  987654321 | 987654331 |      121 |
| farooq@gmail.com   | ARAB     |  919680931 | 800589956 |      114 |
| ford@gmail.com     | SAMPADA  |  351101145 | 978177303 |      107 |
| james@gmail.com    | SEEMA    |  275132251 | 830846028 |      104 |
| jones@gmail.com    | AMIT     |  853545750 | 750645619 |      109 |
| king@gmail.com     | ANEESH   |  666085069 | 740021758 |      110 |
| kingsley@gmail.com | NEHA     |  865645499 | 914600578 |      113 |
| martin@gmail.com   | VRUSHALI |   30574633 | 770475608 |      106 |
| scott@gmail.com    | AROHI    |  775609140 | 955286477 |      111 |
| singh@gmail.com    | VIMAL    |  964077614 | 882498302 |      108 |
| smith@gmail.com    | PARAS    |  737390518 | 888882570 |      112 |
| sorenson@gmail.com | AMIT     |     897652 | 998763456 |      127 |
| ward@gmail.com     | PAVAN    |  855134483 | 992345423 |      115 |
+--------------------+----------+------------+-----------+----------+
18 rows in set (0.00 sec)
```

Figure 7.4: New records inserted in passenger table

Ask user to chose a thali that is veg, non veg or vegan. User can chose any number of meals from a list of hotels depicted in green.



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!*

What do you feel like eating today?

Choice of meal: Veg

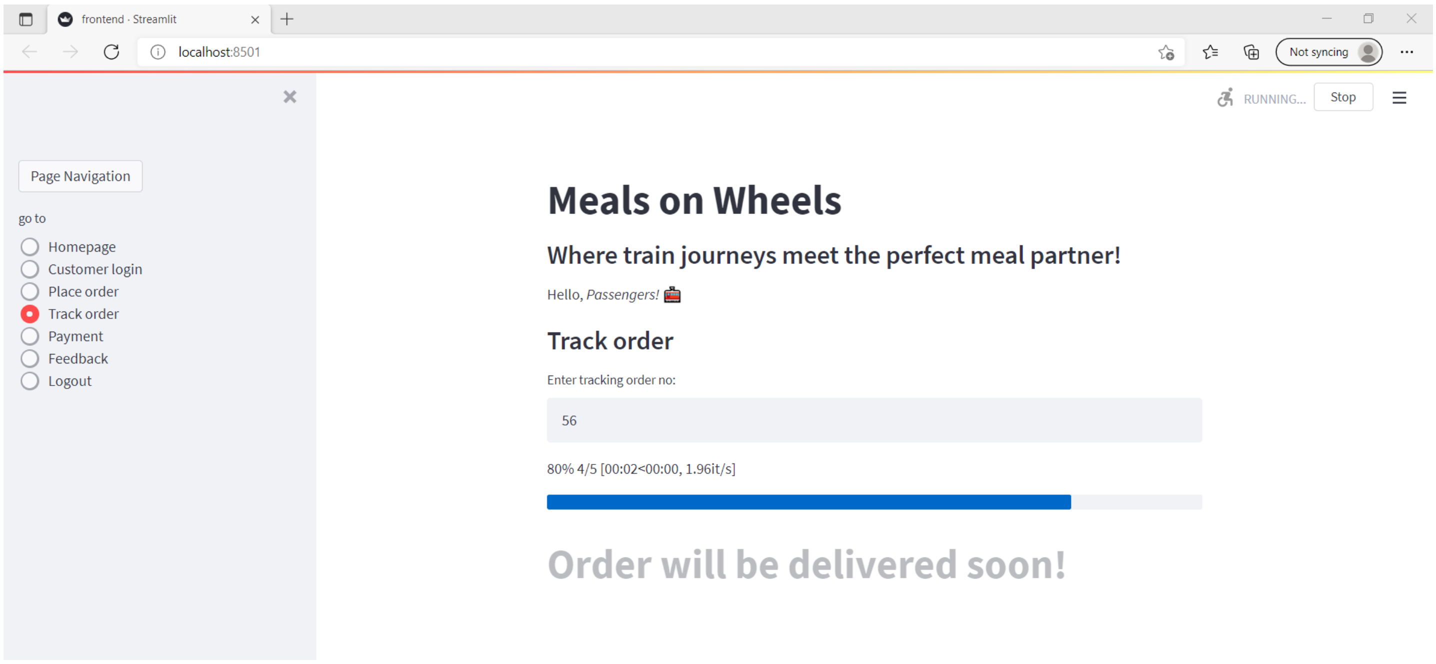
Meals: 1

Chose a restuarant: Yalla Yalla

```
('VEG', 'Kranti Kitchen')
('VEG', 'Kranti Kitchen')
('VEG', 'Yalla Yalla')
('VEG', 'Udupi')
('VEG', 'Godavari')
```

Figure 7.5: Book meal

Track order displays progress bar of meal preparation, when user enters tracking order provided at time of meal booking



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!*

Track order

Enter tracking order no:

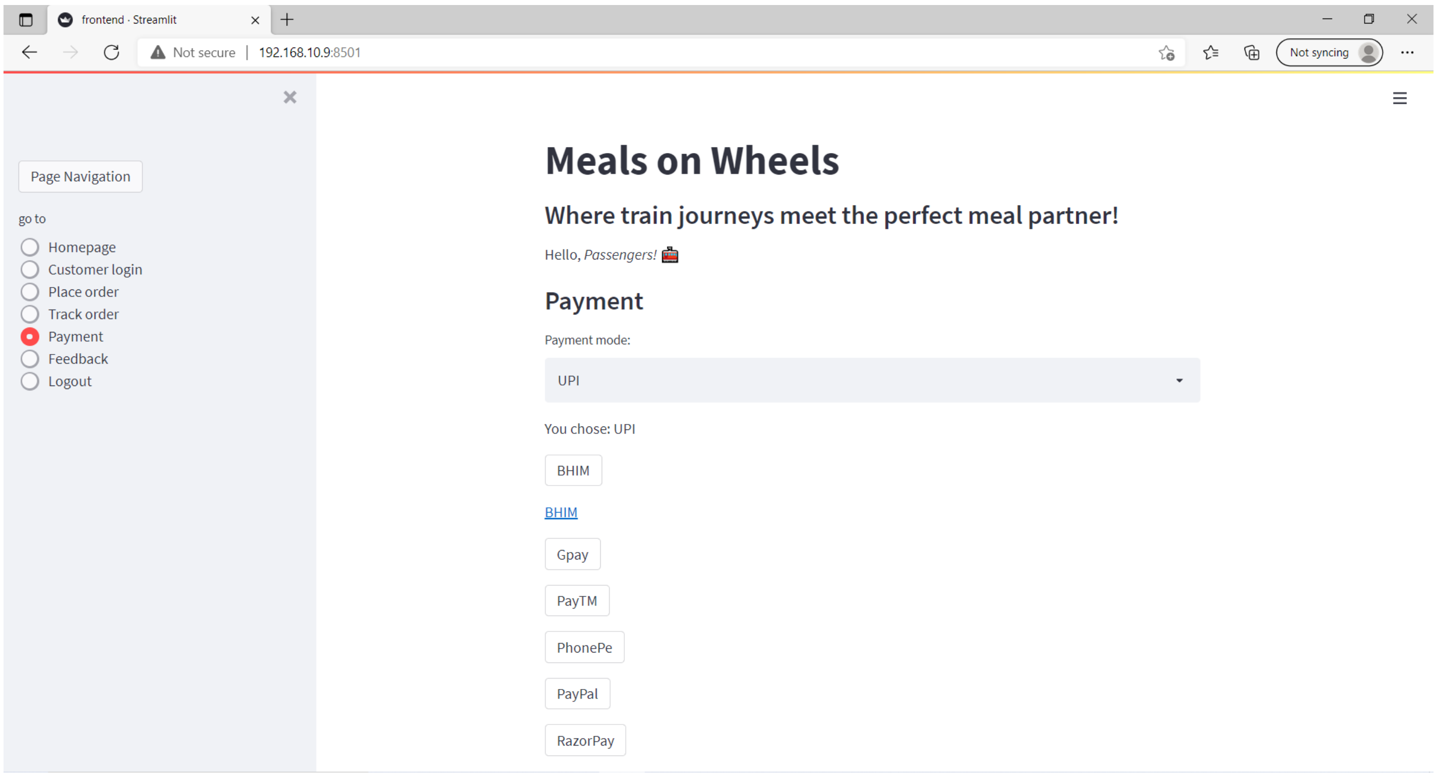
56

80% 4/5 [00:02<00:00, 1.96it/s]

Order will be delivered soon!

Figure 7.6: Progress of order

Passenger can be paid using UPI, cash or card. If user choses UPI, user will be redirected to appropriate page, as depicted in figure 7.7



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!* 🚋

Payment

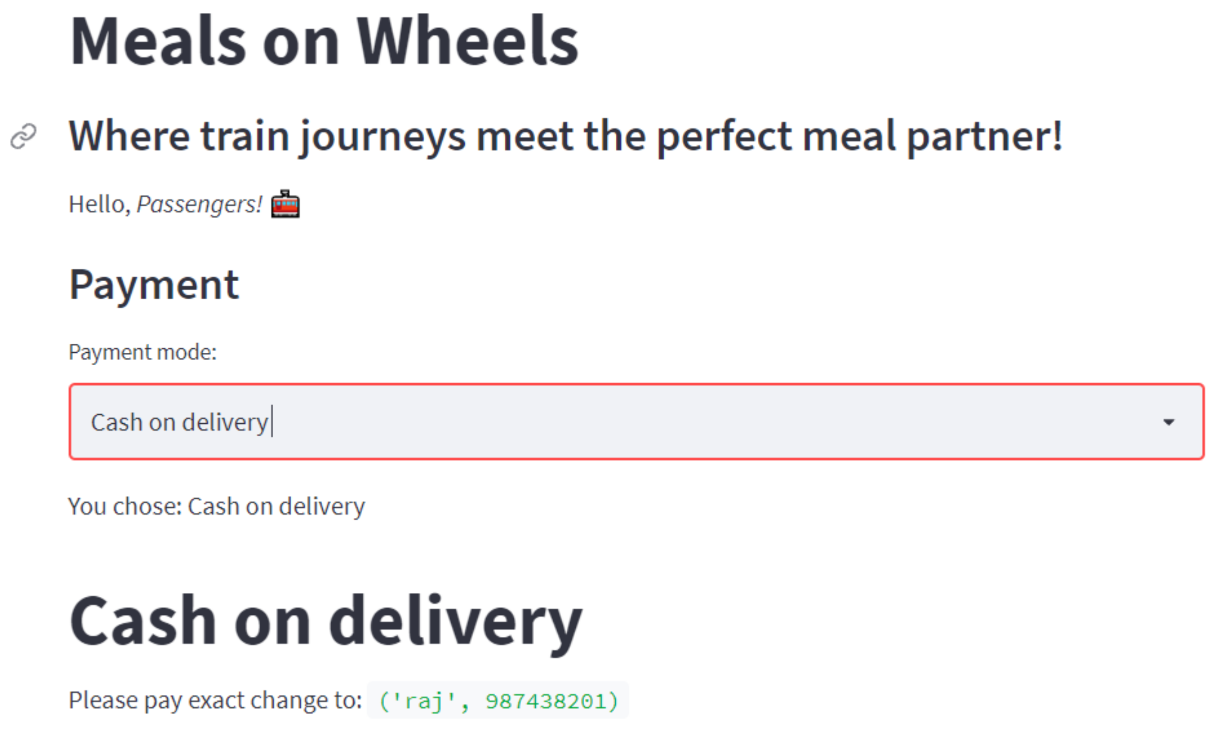
Payment mode:

UPI

You chose: UPI

Figure 7.7: Payment by UPI

User can pay by cash to delivery boy who has been appointed for that station. Name and contact details retrieved from database will be displayed as given in figure 7.8



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!* 🚋

Payment

Payment mode:

Cash on delivery

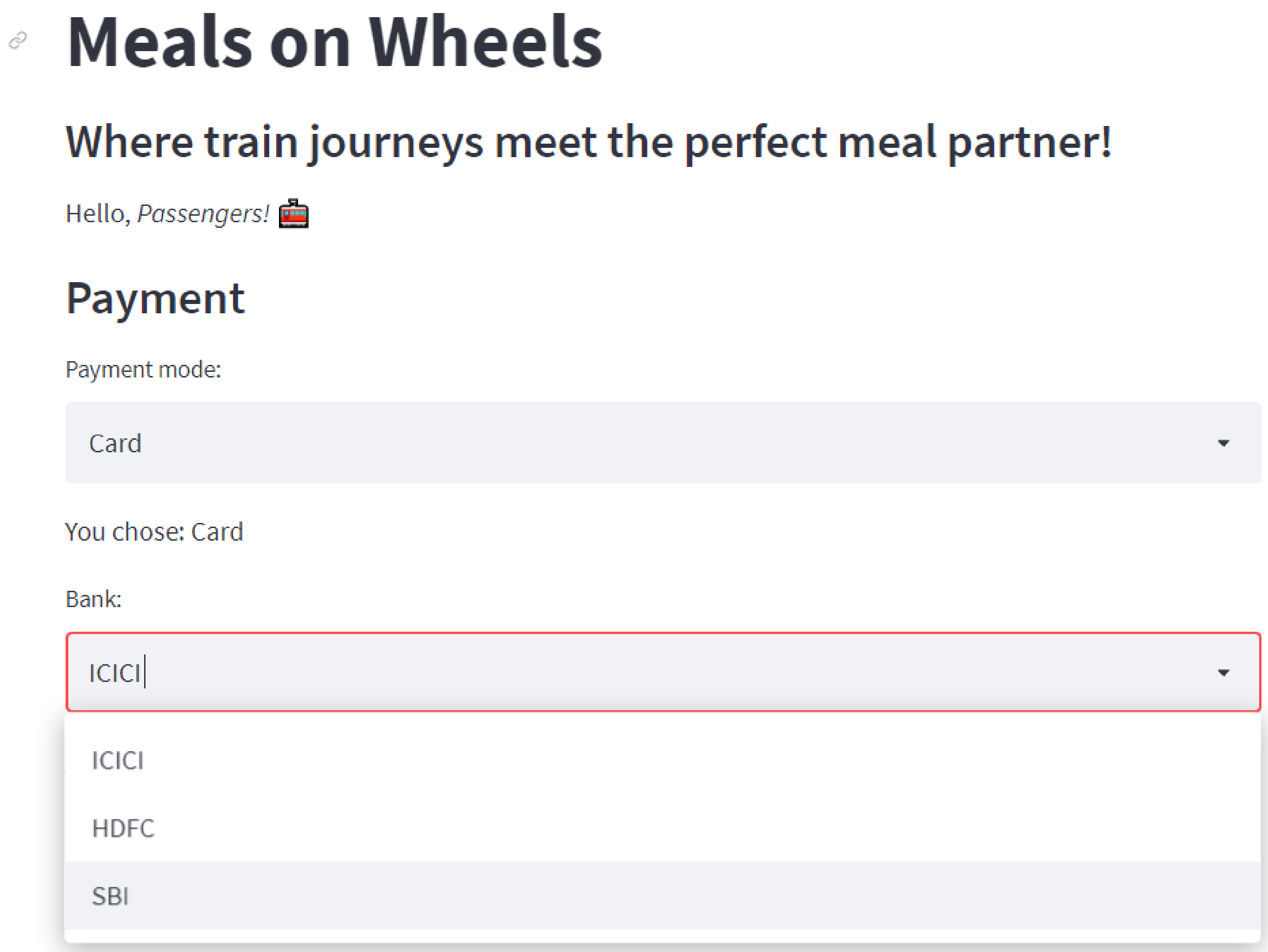
You chose: Cash on delivery

Cash on delivery

Please pay exact change to: `('raj', 987438201)`

Figure 7.8: Payment by cash

User can pay by any card provided in list on screen. This has been depicted in figure 7.9



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!* 🚋

Payment

Payment mode:

Card

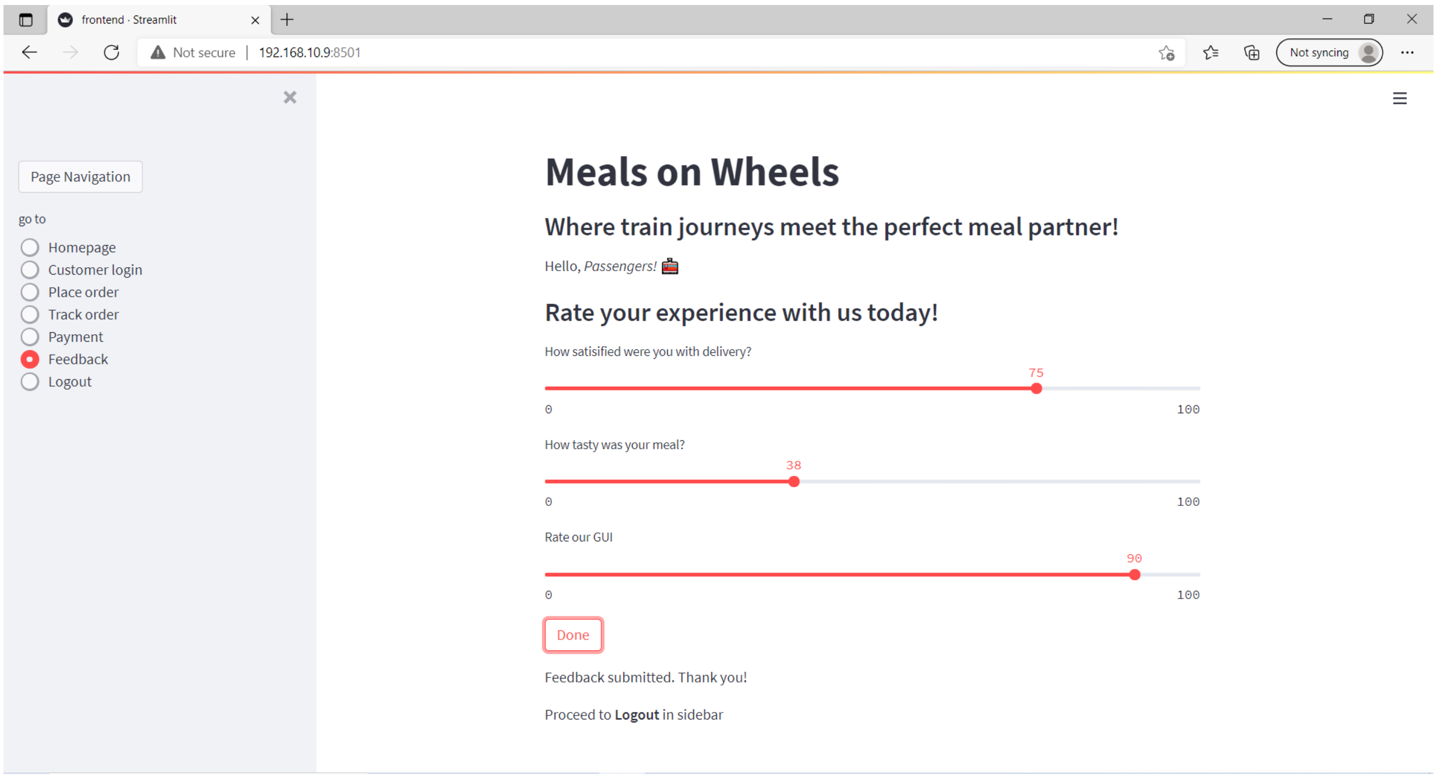
You chose: Card

Bank:

ICICI

Figure 7.9: Payment by card

User can provide feedback using the 3 sliders, as depicted in figure 7.10. Average of 3 results is provided as feedback to kitchen in payment table.



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!*

Rate your experience with us today!

How satisified were you with delivery?
75
0 100

How tasty was your meal?
38
0 100

Rate our GUI
90
0 100

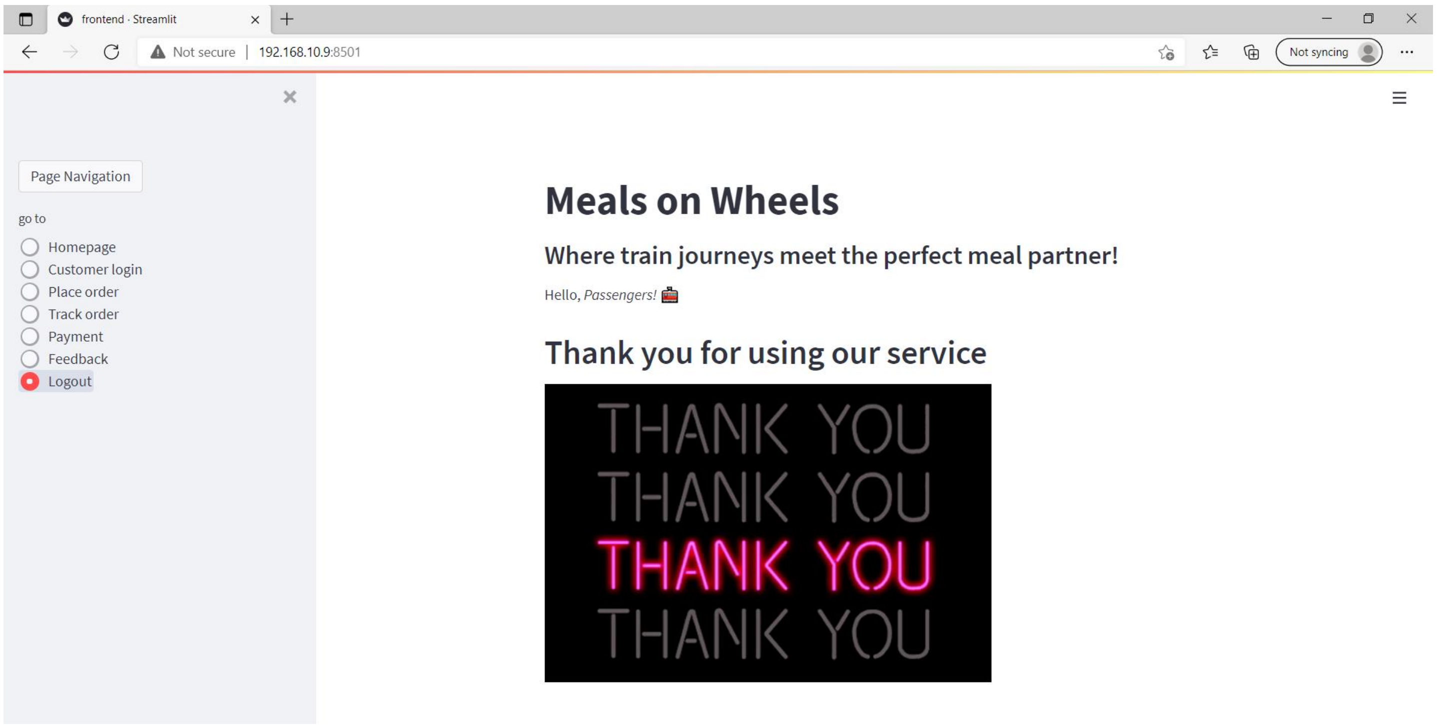
Done

Feedback submitted. Thank you!

Proceed to **Logout** in sidebar

Figure 7.10: Feedback page

User can select logout radio button in sidebar to exit the system. This has been illustrated in figure 7.11



Meals on Wheels

Where train journeys meet the perfect meal partner!

Hello, *Passengers!*

Thank you for using our service

Figure 7.11: Logout of system

8. Conclusion

In conclusion, we have successfully created a Python MySQL end-to-end project for Meals on Wheels booking system. This project aims to cover the main requirements of a food delivery system – book, pay and track the meal.

9. Future scope and evaluation

Following are some aspects, that can be implemented for future development of this rudimentary system for train meal booking:

- Customer can track order using a static progress bar. Future scope involves real time tracking of the meal and delivery boy using GPS.
- If meal is not delivered at station specified in payment table, delivery boy should inform admin of this delay and next station enroute of train will be requested to cover missed meal.
- Updates can be sent to the passenger via provided phone number or email id.
- Development on workbench or Oracle will be easier than backend development on MySQL command line.

Following students contributed in making this project a success. Table 9.1 depicts their self-determined evaluation of contribution to this Meals on Wheels development project.

Table 9.1: Evaluation of contribution of individual members

Contribution to project	Student name			
	PC-11 Reuben	**PC-18 Pranav**	**PC-20 Ankit**	**PC-22 Aditi**
Research	5	5	5	5
Understanding of project and development of prototype	4	4	4	4
Deployment and coding	4	3	3	5
Documentation	3	3	4	4
Total score out of 20 for each student	16	15	16	18

10. References

[1] Ghosh, Mohul (6 November, 2015). Death Of a Tradition – Indian Railways to Phase Out the Legendary Pantry Car; Pushes For E-Catering & Takeaways. Travel, https://trak.in/tags/business/2015/11/06/indian-railways-pantry-car-phase-out-ecatering-takeaway/

[2] IRCTC e-catering site https://www.ecatering.irctc.co.in/

[3] Python https://www.python.org/

[4] Pros and cons of using python for web development https://djangostars.com/blog/python-web-development/

[5] Streamlit documentation https://streamlit.io/

[6] MySQL documentation https://www.mysql.com/

[7] Python and MySQL usage https://data-flair.training/blogs/python-database-access/

[8] Peter Chen notation in ERD https://www.conceptdraw.com/examples/what-is-chen-notation

[9] DDL, DML and DCL command's introduction in MySQL https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/

[10] Oracle documentation for PL SQL https://www.oracle.com/in/database/technologies/appdev/plsql.html

[11] Procedure in MySQL https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

[12] Cursor in MySQL https://dev.mysql.com/doc/refman/8.0/en/cursors.html

[13] Functions in PL SQL https://www.javatpoint.com/mysql-stored-function

[14]Trigger syntax and documentation on MySQL https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

[15] MySQL connector with python tutorial https://www.w3schools.com/python/python_mysql_getstarted.asp