

DELIVERABLE -3 - GROUP-14

Jyosthna Gandhodi- 801254449

Sree Gauthami Gundaram- 801257596

Aparna Reddy Pothula- 801203669

Manswini Ragamouni- 801217775

1) Load the tables with sufficient test data

INSERT menu_items :

```
INSERT INTO `menu_items` VALUES
```

- (1, 101, 'Chicken Nuggets', 'Hot and Crispy Nuggets with some added spices', 5.6),
- (2, 101, 'Vegetable Spring Roll', 'Roll consisting of mixed vegetables with crispy wonton wrapper', 8.7),
- (3, 106, 'Chocolate Lava Cake', 'Lava Cake consisting of Hot Chocolate', 6.8),
- (4, 101, 'French Fries', 'Hot and Crispy French Fries with Ranch', 3.6),
- (5, 102, 'Chicken Soup', 'Hot Liquid Soup with Tinge of Chicken and Spices', 7.05),
- (6, 104, 'Veggie Delight', 'A mix of boiled veggies with cheese and salad', 8.4),
- (7, 103, 'Garden Salad', 'Fresh Romaine Lettuce with added carrots and grape tomatoes', 10.2),
- (8, 106, 'Chocolate Brownie', 'Brownie with dry fruits topping', 6.8),
- (9, 105, 'Orange Chicken', 'Chicken dipped in Orange Sauce', 6.7),
- (10, 106, 'Steak Burger', 'Burger consisting of Steak and Sauce', 12.6),
- (11, 101, 'Beef Spring Roll', 'Roll consisting of beef with crispy wonton wrapper', 10.5),
- (12, 103, 'Romaine Salad', 'Fresh Romaine Lettuce with added cheese', 8.2),
- (13, 106, 'Flavoured Yoghurt', 'Yoghurt in different flavours', 4.8),
- (14, 102, 'Chicken Pizza', 'Hot and Soft Pizza with olives and chicken', 10.05),
- (15, 104, 'Veg Sub', 'A mix of boiled veggies with salad dressing', 7.6),
- (16, 102, 'Chicken Burger', 'Burger consisting of Chicken and Cheese', 7.05),

(17, 104, 'Chocolate Cookies', 'Choco Chip Cookies baked in low flame', 4.4),
(18, 103, 'SoftDrink', 'Drinks with Ice Cubes', 3.2);

INSERT order_rating:

insert into `order_rating` values(1, 3, 3.5, 4, 'Food is so good','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(2, 5, 4, 2.5, 'Food is delicious','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(3, 7, 1.5, 4,'Food can be better','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(4, 9, 3, 1.5, 'Food is delivered too late','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(6, 2, 3, 4, 'Food is so good','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(7, 2, 4, 2, 'Food is delicious','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(8, 2, 1, 4,'Food can be better','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(9, 2, 3, 2, 'Food is delivered too late','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(10, 4, 4, 4, 'Food is so good','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(11, 4, 5, 2, 'Food is delicious','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(12, 4, 2, 4,'Food can be better','https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(13, 4, 3, 1, 'Food is delivered too late', 'https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(14, 5, 4, 4, 'Food is so good', 'https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(15, 4, 4, 5, 'Food is delicious', 'https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(16, 3, 1, 3, 'Food can be better', 'https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/'),

(17, 4, 3, 1, 'Food is delivered too late', 'https://www.pexels.com/photo/flat-lay-photography-of-vegetable-salad-on-plate-1640777/');

INSERT order_status:

insert into `order_status` values (1, 'Delivered'),

(2, 'Out for Delivery'),

(3, 'In the Kitchen'),

(4, 'Order Placed');

INSERT payments:

insert into `payments` values (1, 2, 1, 21.63, 7.75),

(2, 4, 2, 18.85, 5.25),

(3, 5, 3, 17.89, 4.75),

(4, 7, 4, 12.54, 2.25),

(5, 6, 5, 4.88, 1.86),

(6, 10, 6, 25.39, 6.25),

(7, 15, 7, 14.68, 9.23),

(8, 19, 8, 19.67, 7.05);

INSERT restaurant:

```
INSERT INTO `restaurant` VALUES (101, '901 University City Blvd', 'Bojangles', '7am - 11pm', 'https://www.bojangles.com/'),  
(102, '9025 University Rd, Charlotte', 'Panda Express', '8am - 9pm', 'https://www.pandaexpress.com/'),  
(103, '9201 University City Blvd', 'Wendys', '11am - 10pm', 'https://www.wendys.com/'),  
(104, '9025 University Rd, Charlotte', 'Subway', '10am - 10pm', 'https://order.subway.com/'),  
(105, '8917 Johnson Alumni Way', 'SoVi', '7am - 11pm', 'http://aux.charlotte.edu/dining/dining'),  
(106, '9025 University Rd, Charlotte', 'Panda Express', '10am - 9pm', 'https://www.pandaexpress.com/');
```

2) Create queries according to those specified in Deliverable 3 in Blackboard:

a) display the max, min and average ratings for each feature when given a restaurant ID for all orders for that restaurant

```
SELECT MAX(delivery_rating), MIN(delivery_rating), AVG(food_rating),  
MAX(food_rating), MIN(food_rating), AVG(food_rating)  
FROM order_rating  
WHERE order_id  
IN(SELECT order_id FROM `order` WHERE `order`.restaurant_id = 4);
```

Question1 x SQL File 10* order_rating order order_status order_status SQL File 11* SQL File 12* SQL File 13*

Don't Limit

```

1 • SELECT MAX(delivery_rating),MIN(delivery_rating),AVG(food_rating),
2     MAX(food_rating),MIN(food_rating),AVG(food_rating)
3     FROM order_rating
4     WHERE order_id
5     IN(SELECT order_id FROM `order` WHERE `order`.restaurant_id = 4);
6

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	MAX(delivery_rating)	MIN(delivery_rating)	AVG(food_rating)	MAX(food_rating)	MIN(food_rating)	AVG(food_rating)
▶ 5		1	3.5000	5	2	3.5000

b) display a count of the orders made by a customer for a specified date range when given a customer id

```
SELECT COUNT(order_id) from `order`
```

```
WHERE person_id = 1
```

```
and SUBSTRING(timestamp, 1, 10) BETWEEN '2021-09-05' AND '2021-10-02';
```

Question1 Question2* x SQL File 10* order_rating order order_status order_status SQL File 11* SQL File 12* SQL File 13*

Don't Limit

```

1 • SELECT COUNT(order_id) from `order`
2     WHERE person_id = 1
3     and SUBSTRING(timestamp, 1, 10) BETWEEN '2021-09-05' AND '2021-10-30';
4

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	COUNT(order_id)
▶ 2	

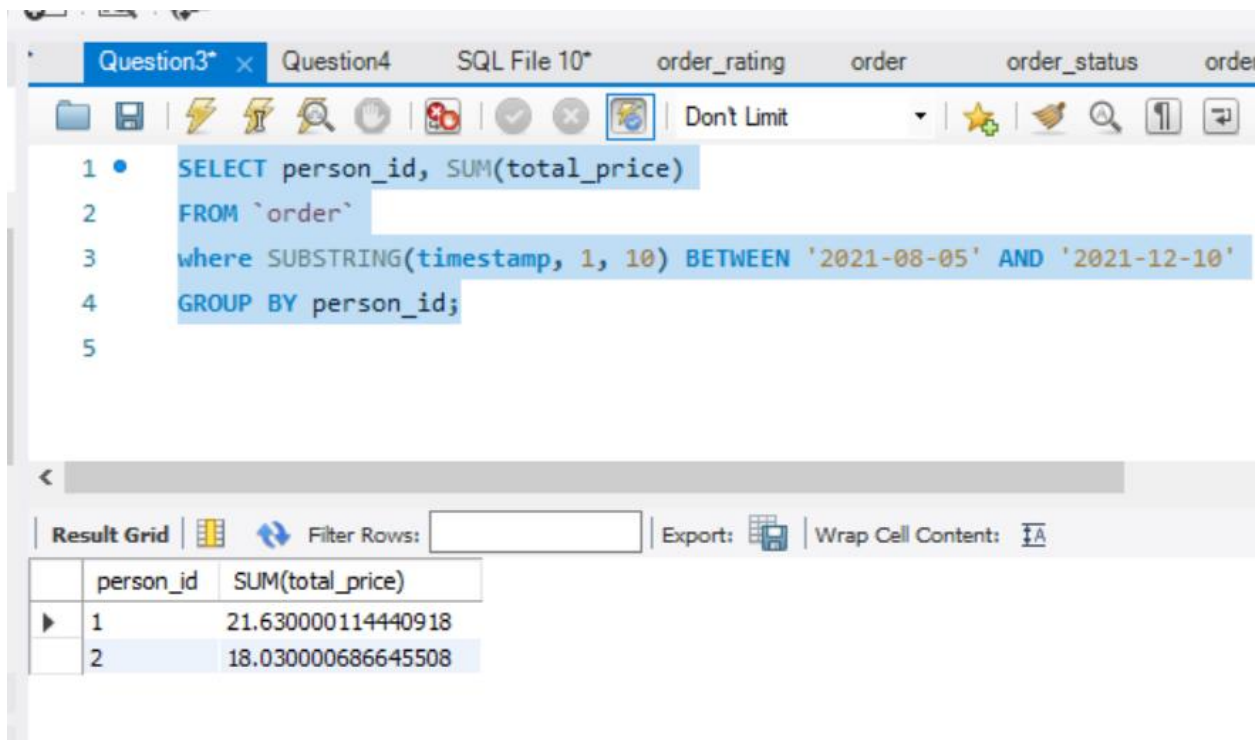
c) display total price of the orders by each customer (distinct) for a specified date range

```
SELECT person_id, SUM(total_price)
```

```
FROM `order`
```

```
where SUBSTRING(timestamp, 1, 10) BETWEEN '2021-08-05' AND '2021-12-10'
```

```
GROUP BY person_id;
```



The screenshot shows a SQL IDE interface with a query editor and a result grid. The query editor contains the following SQL code:

```
1 SELECT person_id, SUM(total_price)
2 FROM `order`
3 where SUBSTRING(timestamp, 1, 10) BETWEEN '2021-08-05' AND '2021-12-10'
4 GROUP BY person_id;
5
```

The result grid displays the following data:

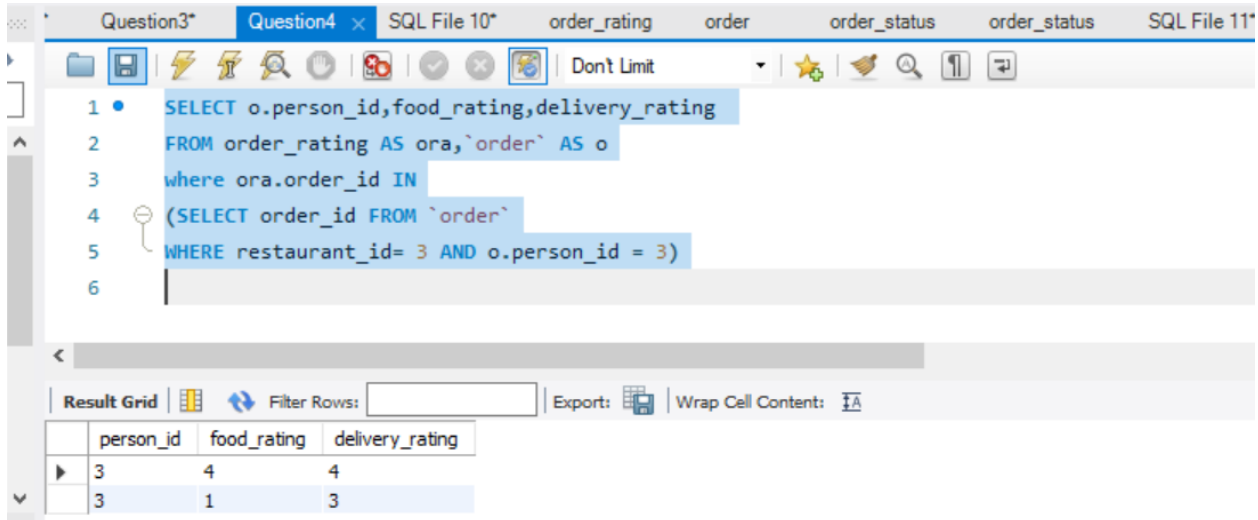
person_id	SUM(total_price)
1	21.630000114440918
2	18.030000686645508

d) display a particular customer's rating for a restaurant

```

SELECT o.person_id,food_rating,delivery_rating
FROM order_rating AS ora,`order` AS o
where ora.order_id IN
(SELECT order_id FROM `order`
WHERE restaurant_id= 3 AND o.person_id = 3)

```



e) Have one of the above requirements represented in a View

view_for_customer_orders

```

-- view for customer orders

USE `campus_eats_fall2020`;

CREATE OR REPLACE VIEW `customer_orders` AS

SELECT DISTINCT person_id as customer_id,

ROUND(total_price + delivery_charge) AS order_total

FROM campus_eats_fall2020.order

```

```
GROUP BY person_id;  
select * from customer_orders
```

The screenshot shows a database IDE with a SQL editor and a result grid. The SQL editor contains the following code:

```
-- view for customer orders  
USE `campus_eats_fall2020`;  
CREATE OR REPLACE VIEW `customer_orders` AS  
SELECT DISTINCT person_id AS customer_id,  
       ROUND(total_price + delivery_charge) AS order_total  
FROM campus_eats_fall2020.order  
GROUP BY person_id;  
select * from customer_orders
```

The result grid displays the following data:

	customer_id	order_total
1	1	22
2	2	27
3	3	19
4	4	25
5	5	20
6	6	10
7	7	23
8	8	6
9	9	24
10	10	15
11	11	11
12	12	14
13	13	12
14	14	13
15	15	9
16	16	15

The IDE interface includes a toolbar with icons for file operations, a "Limit to 1000 rows" dropdown, and a sidebar with options for "Result Grid", "Form Editor", "Field Types", and "Query Stats". The status bar at the bottom indicates "Read Only".

f) Have one of the above requirements represented in a Stored Procedure

stored_procedure_for_count_of_orders

```
USE campus_eats_fall2020;
DROP PROCEDURE IF EXISTS count_of_orders;
DELIMITER //
CREATE PROCEDURE count_of_orders(IN start_year INT,IN end_year INT, OUT output_str varchar(100))
BEGIN
    DECLARE order_count Varchar(20);
    SELECT count(*) into order_count
    FROM `order`
    WHERE person_id in (
        select person_id from student where graduation_year
        between start_year and end_year
    );
    IF order_count < 0 THEN
        SET output_str = CONCAT("The number of orders are
0");
    ELSE
        SET output_str = CONCAT("The number of orders are ",
order_count);
    END IF;
END //
DELIMITER ;
```

-- Gets number of orders from 2010 to 2013

CALL count_of_orders(2009,2013,@output_str);

Select @output_str

The screenshot shows a SQL IDE window with a tab titled "stored_procedure_for_count_of...". The main editor displays a SQL script with the following content:

```
1 • USE campus_eats_fall2020;
2 • DROP PROCEDURE IF EXISTS count_of_orders;
3   DELIMITER //
4 • CREATE PROCEDURE count_of_orders(IN start_year INT,IN end_year INT, OUT output_str varchar(100))
5   BEGIN
6       DECLARE order_count Varchar(20);
7       SELECT count(*) into order_count
8       FROM `order`
9       WHERE person_id in (
10          select person_id from student where graduation_year between start_year and end_year
11      );
12      IF order_count < 0 THEN
13          SET output_str = CONCAT("The number of orders are 0");
14      ELSE
15          SET output_str = CONCAT("The number of orders are ", order_count);
16      END IF;
17  END //
18
19  DELIMITER ;
20 • -- Gets number of orders from 2010 to 2013
21  CALL count_of_orders(2009,2013,@output_str);
```

Below the script, the "Result Grid" is visible, showing the output of the stored procedure call:

@output_str
The number of orders are 4

The IDE interface includes a toolbar at the top with icons for file operations, a "Limit to 1000 rows" dropdown, and a "Read Only" status indicator at the bottom right.

stored_procedure_for_min_max_avg

```
USE campus_eats_fall2020;

DROP PROCEDURE IF EXISTS get_min_max_avg_rating_for_restaurant;

DELIMITER //

CREATE PROCEDURE get_min_max_avg_rating_for_restaurant (IN restaurant_id INT(50), OUT max_food
INT, OUT min_food INT, OUT avg_food INT, OUT max_del INT, OUT min_del INT, OUT avg_del INT)

BEGIN

    SET max_food = 0;

                                SET min_food = 0;

    SET avg_food = 0;

    SET max_del = 0;

    SET min_del = 0;

    SET avg_del = 0;


    SELECT MAX(food_rating)

    INTO max_food

    FROM order_rating

    LEFT JOIN campus_eats_fall2020.order

    ON order_rating.order_id = campus_eats_fall2020.order.order_id

    where campus_eats_fall2020.order.restaurant_id = restaurant_id;
```

```
SELECT MIN(food_rating)
INTO min_food
FROM order_rating
LEFT JOIN campus_eats_fall2020.order
ON order_rating.order_id = campus_eats_fall2020.order.order_id
where campus_eats_fall2020.order.restaurant_id = restaurant_id;
```

```
SELECT AVG(food_rating)
INTO avg_food
FROM order_rating
LEFT JOIN campus_eats_fall2020.order
ON order_rating.order_id = campus_eats_fall2020.order.order_id
where campus_eats_fall2020.order.restaurant_id = restaurant_id;
```

```
SELECT MAX(delivery_rating)
INTO max_del
FROM order_rating
LEFT JOIN campus_eats_fall2020.order
ON order_rating.order_id = campus_eats_fall2020.order.order_id
where campus_eats_fall2020.order.restaurant_id = restaurant_id;
```

```
SELECT MIN(delivery_rating)
INTO min_del
FROM order_rating
LEFT JOIN campus_eats_fall2020.order
ON order_rating.order_id = campus_eats_fall2020.order.order_id
where campus_eats_fall2020.order.restaurant_id = restaurant_id;
```

```

SELECT AVG(delivery_rating)

INTO avg_del

FROM order_rating

LEFT JOIN campus_eats_fall2020.order

ON order_rating.order_id = campus_eats_fall2020.order.order_id

where campus_eats_fall2020.order.restaurant_id = restaurant_id;

END //

DELIMITER ;

CALL get_min_max_avg_rating_for_restaurant(2,@Max_Food_Rating,@Min_Food_Rating,
@Avg_Food_Rating, @Max_Del_Rating, @Min_Del_Rating, @Avg_Del_Rating);

SELECT @Max_Food_Rating, @Min_Food_Rating, @Avg_Food_Rating, @Max_Del_Rating,
@Min_Del_Rating, @Avg_Del_Rating ;

```

The screenshot shows a SQL IDE with a script editor and a results grid. The script editor contains the following SQL code:

```

1 USE campus_eats_fall2020;
2 DROP PROCEDURE IF EXISTS get_min_max_avg_rating_for_restaurant;
3 DELIMITER //
4 CREATE PROCEDURE get_min_max_avg_rating_for_restaurant (IN restaurant_id INT(50), OUT max_food INT, OUT min_food INT, OUT avg_food INT, OUT max_del INT, OUT min_del INT, OUT avg_del INT)
5 BEGIN
6     SET max_food = 0;
7     SET min_food = 0;
8     SET avg_food = 0;
9     SET max_del = 0;
10    SET min_del = 0;
11    SET avg_del = 0;
12
13    SELECT MAX(food_rating)
14    INTO max_food
15    FROM order_rating
16    LEFT JOIN campus_eats_fall2020.order
17    ON order_rating.order_id = campus_eats_fall2020.order.order_id
18    where campus_eats_fall2020.order.restaurant_id = restaurant_id;
19
20    SELECT MIN(food_rating)
21    INTO min_food

```

The results grid shows the output of the stored procedure call for restaurant_id 2:

@Max_Food_Rating	@Min_Food_Rating	@Avg_Food_Rating	@Max_Del_Rating	@Min_Del_Rating	@Avg_Del_Rating
8	8	8	8	8	8

Function :

```
DROP FUNCTION IF EXISTS funct_driver_rating;

DELIMITER //

CREATE FUNCTION funct_driver_rating
(
    rating INT
)
RETURNS varchar(30)
deterministic
BEGIN
    DECLARE rating_comment varchar(30);
    IF rating = 1 THEN
        SET rating_comment = "Worst driver";
    ELSEIF rating = 2 THEN
        SET rating_comment = "Bad driver";
    ELSEIF rating = 3 THEN
        SET rating_comment = "Average driver";
    ELSEIF rating = 4 THEN
        SET rating_comment = "Good driver";
    ELSEIF rating = 5 THEN
        SET rating_comment = "Excellent driver";
    END IF;

    RETURN rating_comment;
END//

DELIMITER ;

select driver_id, funct_driver_rating(rating) from driver;
```

SQL File 6" x order_rating driver delivery order driver

Limit to 1000 rows

```

1 • DROP FUNCTION IF EXISTS func_driver_rating;
2 DELIMITER //
3 • CREATE FUNCTION func_driver_rating
4 (
5     rating INT
6 )
7 RETURNS varchar(30)
8 deterministic
9 BEGIN
10     DECLARE rating_comment varchar(30);
11     IF rating = 1 THEN
12         SET rating_comment = "Worst driver";
13     ELSEIF rating = 2 THEN
14         SET rating_comment = "Bad driver";
15     ELSEIF rating = 3 THEN
16         SET rating_comment = "Average driver";
17     ELSEIF rating = 4 THEN

```

Result Grid

driver_id	func_driver_rating(rating)
1	Good driver
2	Average driver
3	Average driver
4	Average driver
5	Good driver
6	Average driver
7	Average driver

Result 2 x

Output

Action Output

#	Time	Action	Message
42	19:37:59	CREATE FUNCTION func_driver_rating (rating INT) RETURNS varchar(30) determinist...	0 row(s) affected
43	19:37:59	select driver_id, func_driver_rating(rating) from driver LIMIT 0, 1000	8 row(s) returned

INDEX :

DROP TABLE IF EXISTS `order_rating`;

CREATE TABLE IF NOT EXISTS `campus_eats_fall2020`.`order_rating` (

`id` INT NOT NULL,

`order_id` INT NOT NULL,

`food_rating` INT NULL,

`delivery_rating` INT NULL,

```
`comments` VARCHAR(200) NULL,  
`picture` VARCHAR(100) NULL,  
PRIMARY KEY (`id`),  
INDEX `order_id_idx` (`order_id` ASC),  
CONSTRAINT `order_id`  
FOREIGN KEY (`order_id`)  
REFERENCES `campus_eats_fall2020`.`order` (`order_id`))  
ENGINE = InnoDB
```

