

Deliverable 3

Stored Procedures

- Stored Procedure 1:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane shows the 'campus_eats_fall2020' database with various tables listed. The main pane shows the SQL script for creating and executing the stored procedure 'get_restaurant_rating_by_restaurant'. The script includes a DROP statement, a CREATE statement with parameters, and a CALL statement. Below the script, the 'Result Grid' shows the output of the CALL statement, displaying the average rating for restaurant ID 101. The 'Output' pane shows the execution log with two messages: '1 row(s) affected' and '1 row(s) returned'.

```
1 DROP PROCEDURE IF EXISTS get_restaurant_rating_by_restaurant;
2 DELIMITER $$
3 CREATE PROCEDURE get_restaurant_rating_by_restaurant(IN restaurant_id INT, OUT res_id INT, OUT avg_rating FLOAT)
4 BEGIN
5     SELECT restaurant_id, AVG(ratingscore)
6     INTO res_id, avg_rating
7     FROM campus_eats_fall2020.restaurant_rating
8     WHERE restaurant_id = restaurant_id;
9 END$$
10
11 CALL get_restaurant_rating_by_restaurant(101,@res_id,@avg_rating);
12 SELECT @res_id,@avg_rating;
```

@res_id	@avg_rating
101	3.333332538604736

Table: feedback_question_assoc
Columns: feedback_question_id int AI PK, feedback_id int, question_id int

Stored procedure 1 returns the Average Rating of a particular restaurant.

- Stored Procedure 2:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane shows the 'campus_eats_fall2020' database with various tables listed. The main pane shows the SQL script for creating and executing the stored procedure 'get_driver_rating_by_driver'. The script includes a DROP statement, a CREATE statement with parameters, and a CALL statement. Below the script, the 'Result Grid' shows the output of the CALL statement, displaying the average rating for driver ID 1. The 'Output' pane shows the execution log with two messages: '1 row(s) affected' and '1 row(s) returned'.

```
18 DROP PROCEDURE IF EXISTS get_driver_rating_by_driver;
19 DELIMITER $$
20 CREATE PROCEDURE get_driver_rating_by_driver(IN driverid INT, OUT d_id INT, OUT avg_driver_rating INT)
21 BEGIN
22     SELECT driver_id,AVG(ratingscore)
23     INTO d_id, avg_driver_rating
24     FROM campus_eats_fall2020.driver_rating
25     WHERE driver_id = driverid;
26 END$$
27
28
29 CALL get_driver_rating_by_driver(1,@driverId,@avg_driver_rating);
30 SELECT @driverId,@avg_driver_rating;
31
```

@driverId	@avg_driver_rating
1	4

Stored procedure 2 returns the Average Rating of a driver.

Views

- View 1:

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

```
1 DROP VIEW IF EXISTS menu_availability ;
2 CREATE VIEW menu_availability AS
3 SELECT rma.restaurant_menu_id,r.restaurant_id,m.menu_id,r.restaurant_name,
4 m.menu_name FROM restaurant_menu_assoc AS rma
5 INNER JOIN restaurant r
6 ON rma.restaurant_id = r.restaurant_id
7 INNER JOIN menu m
8 ON rma.menu_id = m.menu_id WHERE is_available = 'Y';
9
10 select * from menu_availability;
```

The result grid displays the following data:

restaurant_menu_id	restaurant_id	menu_id	restaurant_name	menu_name
1	101	1	Chick-ri-A	Nuggets
3	101	3	Chick-ri-A	Fries
4	101	4	Chick-ri-A	Sandwich
5	101	5	Chick-ri-A	Brownie
6	101	6	Chick-ri-A	Milkshake
7	102	7	Bojangles	FruitCup
8	102	8	Bojangles	3 piece tender
9	102	9	Bojangles	Egg Biscuit
10	102	10	Bojangles	Botato Rounds
13	103	3	Wendys	Fries
15	103	5	Wendys	Brownie
16	104	1	Salsaritas	Nuggets
17	105	2	Star Bucks	Burger
18	106	3	Crown Commons	Fries
20	106	2	Crown Commons	Burger
22	105	1	Star Bucks	Nuogets

View 1 shows menu availability in a restaurant and returns restaurant name and menu name.

- View 2:

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

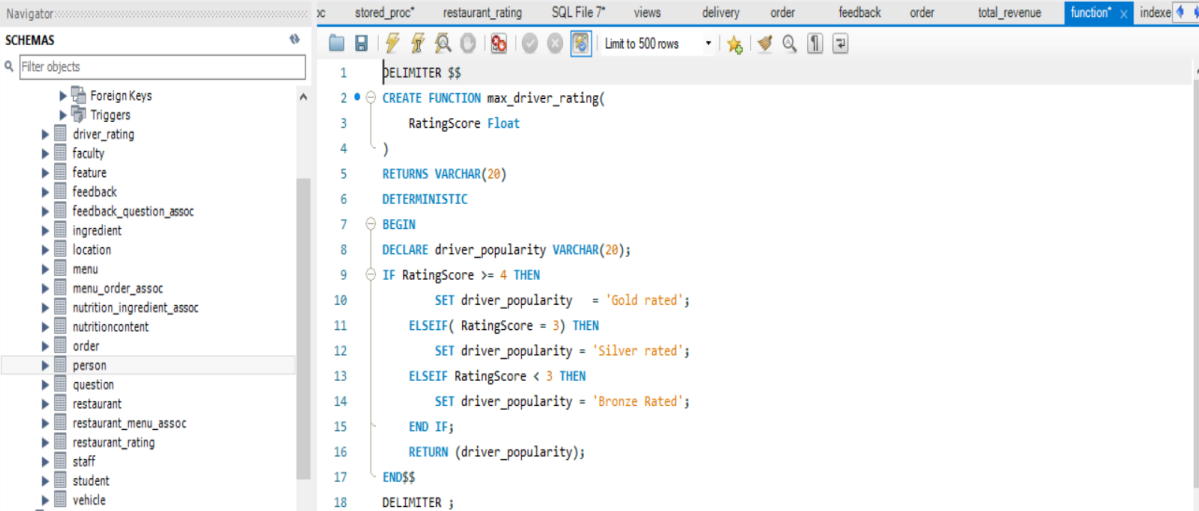
```
17 DROP VIEW IF EXISTS total_revenue;
18 CREATE VIEW total_revenue AS
19 SELECT res.restaurant_id,restaurant_name, SUM(total_price) AS total
20 FROM campus_eats_fall2020.order AS ord INNER JOIN restaurant AS res
21 ON ord.restaurant_id=res.restaurant_id
22 WHERE DATE(ord_date) = current_date() GROUP BY restaurant_id;
23
24 select * FROM total_revenue;
```

The result grid displays the following data:

restaurant_id	restaurant_name	total
101	Chick-ri-A	36.47999954223633
102	Bojangles	12.59000015258789

View 2 shows total revenue of a restaurant and returns the restaurant name and it's revenue.

Function

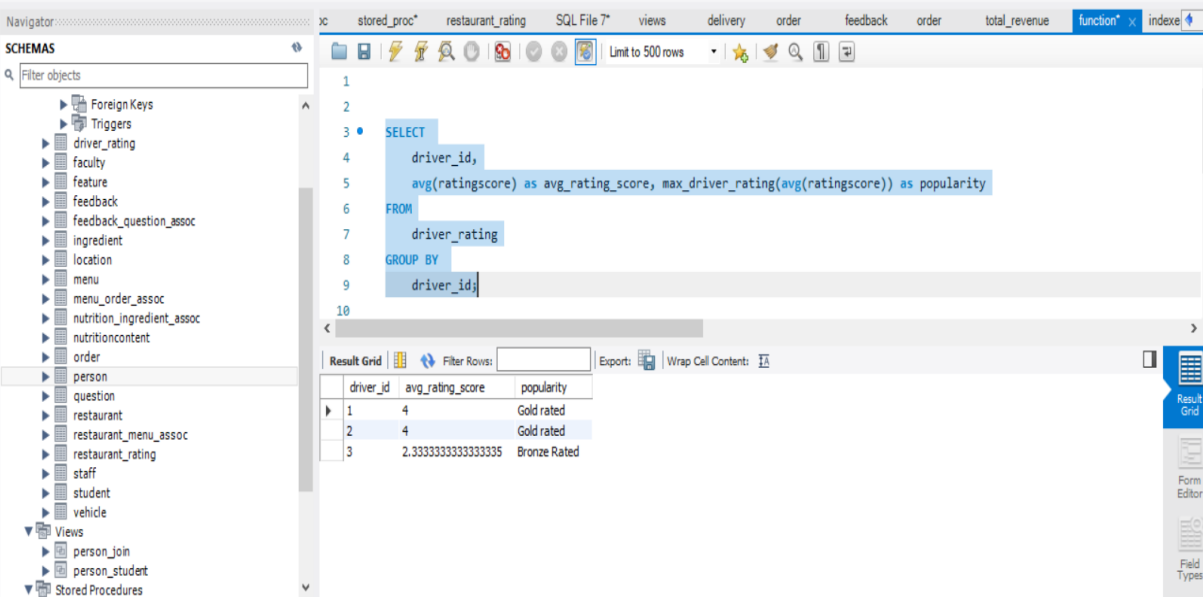


The screenshot shows a database IDE with a 'Schemas' pane on the left and a 'SQL File 7*' editor on the right. The editor contains the following SQL code:

```
1 DELIMITER $$
2 CREATE FUNCTION max_driver_rating(
3     RatingScore Float
4 )
5 RETURNS VARCHAR(20)
6 DETERMINISTIC
7 BEGIN
8     DECLARE driver_popularity VARCHAR(20);
9     IF RatingScore >= 4 THEN
10         SET driver_popularity = 'Gold rated';
11     ELSEIF (RatingScore = 3) THEN
12         SET driver_popularity = 'Silver rated';
13     ELSEIF RatingScore < 3 THEN
14         SET driver_popularity = 'Bronze Rated';
15     END IF;
16     RETURN (driver_popularity);
17 END$$
18 DELIMITER ;
```

Created a function named max_driver_rating which returns a driver's rating.

Function Call



The screenshot shows the same database IDE with a new SQL query in the editor:

```
1
2
3 SELECT
4     driver_id,
5     avg(ratingscore) as avg_rating_score, max_driver_rating(avg(ratingscore)) as popularity
6 FROM
7     driver_rating
8 GROUP BY
9     driver_id;
```

Below the query, the 'Result Grid' is displayed with the following data:

driver_id	avg_rating_score	popularity
1	4	Gold rated
2	4	Gold rated
3	2.3333333333333335	Bronze Rated

Index

The screenshot shows a database management tool interface. On the left is a 'SCHEMAS' tree with various database objects. The main window displays SQL commands in a text editor:

```
1 CREATE INDEX restaurant_index ON restaurant(restaurant_name, location, schedule);
2
3 #To show the indexes of a table
4
5 SHOW INDEXES FROM restaurant;
6
```

Below the editor is a 'Result Grid' showing the output of the SQL command. The grid has columns: Table, Non_unique, Key_name, Seq_in_index, Column_name, Collation, Cardinality, Sub_part, Packed, Null, Index_type, Comment, and Index. The data shows three indexes for the 'restaurant' table:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index
restaurant	0	PRIMARY	1	restaurant_id	A	6	NULL	NULL		BTREE		
restaurant	1	restaurant_index	1	restaurant_name	A	6	NULL	NULL	YES	BTREE		
restaurant	1	restaurant_index	2	location	A	6	NULL	NULL	YES	BTREE		
restaurant	1	restaurant_index	3	schedule	A	6	NULL	NULL	YES	BTREE		

The index we have created returns three indexed columns which are restaurant_name, location and schedule.