

Project Report : List and Description

Team: Jay Patel, Sharad Swaminathan, Aishwarya Babu, Ianesh Karthik Bandhuchoday

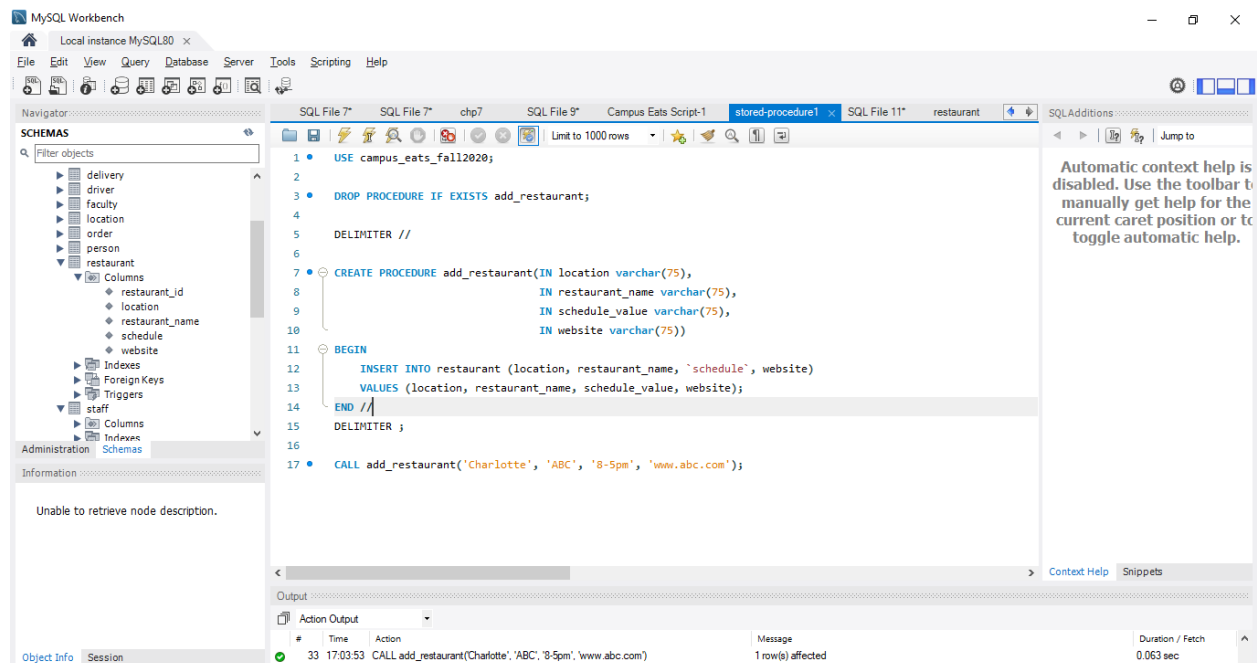
CampusEats is a food delivery app built by team Maverick, we have extended the project by adding a rating system for the restaurants and delivery drivers. We added 3 new tables to the existing 10 tables. The tables are as follows: delivery, driver, faculty, location, order, person, restaurant, staff, student, and vehicle. We added a rating, menu, and menu_order table. Menu_order table is an associative table that sits between menu and order table because of the many to many relationship. We have extended the project further by updating the eerd, adding advanced SQL statements such as stored procedures, views, and functions which are located in the queries folder in Git. We have also added a data dictionary which gives a description of the columns in the database.

Stored Procedure:

1. Name of Stored Procedure: add_restaurant

Purpose: To add new restaurants to the database

Code:



The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view of the database structure, including tables like delivery, driver, faculty, location, order, person, restaurant, staff, student, and vehicle. The 'restaurant' table is expanded, showing its columns: restaurant_id, location, restaurant_name, schedule, and website. The main editor window shows a SQL script for creating and calling a stored procedure. The script is as follows:

```
1 USE campus_eats_fall2020;
2
3 DROP PROCEDURE IF EXISTS add_restaurant;
4
5 DELIMITER //
6
7 CREATE PROCEDURE add_restaurant(IN location varchar(75),
8                                IN restaurant_name varchar(75),
9                                IN schedule_value varchar(75),
10                               IN website varchar(75))
11 BEGIN
12     INSERT INTO restaurant (location, restaurant_name, `schedule`, website)
13     VALUES (location, restaurant_name, schedule_value, website);
14 END //
15 DELIMITER ;
16
17 CALL add_restaurant('Charlotte', 'ABC', '8-5pm', 'www.abc.com');
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
33	17.03.53	CALL add_restaurant('Charlotte', 'ABC', '8-5pm', 'www.abc.com')	1 row(s) affected	0.063 sec

Result:

restaurant_id	location	restaurant_name	schedule	website
93	72392 Hahn Station Apt. 674 ...	Price-Reinger	11am - 11pm	http://stoltenbergmohr.biz/
94	9329 Vesta Harbors Suite 849 ...	Boehm, White and Kilback	9am - 10pm	http://www.hilljohnson.com/
95	30248 Eichmann Street Suite 1...	Conroy-O'Keefe	10am - 9pm	http://hermiston.org/
96	1070 Green Forks Selenaland, ...	Wiegand LLC	9am - 10pm	http://www.streich.com/
97	20073 Clyde Ways Suite 898 ...	Donnelly and Sons	10am - 9pm	http://cummings.net/
98	215 Altenwerth Mall Apt. 621 ...	McDermott, Senger and Ferry	10am - 9pm	http://www.collins.net/
99	1105 Liza Shores Apt. 158 Her...	Grimes-Lakin	9am - 10pm	http://terry.net/
100	88626 Louvenia Fork Lake Max...	Mertz Ltd	9am - 10pm	http://ortiz.com/
101	Charlotte	ABC	8-5pm	www.abc.com

2. Name of Stored Procedure: add_vehicle

Purpose: To add a new vehicle to the database.

Code:

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view with 'vehicle' selected, containing columns like vehicle_id, vehicle_plate, model, and make. The central editor window shows the following SQL code:

```
1 USE campus_eats_fall2020;
2
3 DROP PROCEDURE IF EXISTS add_vehicle;
4
5 DELIMITER //
6
7 CREATE PROCEDURE add_vehicle(IN vehicle_plate varchar(75),
8                               IN model varchar(75),
9                               IN make varchar(75))
10 BEGIN
11     INSERT INTO vehicle (vehicle_plate, model, make)
12     VALUES (vehicle_plate, model, make);
13 END //
14 DELIMITER ;
15
16 CALL add_vehicle('XYZ', 'test', 'test');
```

On the right, a message box states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the editor, the 'Output' pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
38	17:10:34	CALL add_vehicle('XYZ', 'test', 'test')	1 row(s) affected	0.047 sec

Result:

Result Grid				
Filter Rows: <input type="text"/>				
	vehide_id	vehide_plate	model	make
	15	2767	d	Bugatti
	16	4184	j	Mercedes
	17	6489	l	BMW
	18	4260	j	Audi
	19	6455	d	Mercedes
	20	7009	q	Toyota
	21	XYZ	test	test
*	NULL	NULL	NULL	NULL

vehicle 1 x

3. Name of Stored Procedure: customer_restaurant_rating

Purpose: gets the average rating of a customer from a restaurant when inputted a person_id and restaurant_id

Code:

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'campus_eats_fall2020' schema. The main editor displays the following SQL code:

```

1 DROP PROCEDURE IF EXISTS customer_restaurant_rating;
2
3 DELIMITER //
4
5 CREATE PROCEDURE customer_restaurant_rating(IN p_id INT,
6      IN r_id INT)
7 BEGIN
8     SELECT person_id, ROUND(AVG(food_rating), 2) AS avg_rating
9     FROM `order`
10    INNER JOIN rating
11    USING (order_id)
12    WHERE person_id = p_id AND restaurant_id = r_id;
13 END //
14 DELIMITER ;
15
16 CALL customer_restaurant_rating(1, 1);

```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
138	20:06:10	SELECT * FROM campus_eats_fall2020.location LIMIT 0, 1000	100 row(s) returned	0.000 sec / 0.000 sec
139	20:06:24	SELECT * FROM campus_eats_fall2020.order LIMIT 0, 1000	101 row(s) returned	0.000 sec / 0.000 sec

Result:

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
person_id	avg_rating			
1	4.00			

4. Name of Stored Procedure: total_order_cost

Purpose: given a data range, the procedure return orders placed in that timeframe

Code:

gets the total costs

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' pane shows the 'campus_eats_fall2020' schema. The main editor displays the SQL code for creating and calling the stored procedure 'total_order_cost'. The code includes a DROP statement, a CREATE PROCEDURE statement with parameters 'from_date' and 'to_date', and a CALL statement for the date range '2021-11-24' to '2021-12-03'. The procedure body uses a SELECT statement to calculate total costs from the 'order' table, grouped by 'person_id'. The output pane at the bottom shows the execution results, including the creation of the procedure and the call results.

```

1 DROP PROCEDURE IF EXISTS total_order_cost;
2
3 DELIMITER //
4
5 CREATE PROCEDURE total_order_cost(IN from_date DATE,
6                                   IN to_date DATE)
7 BEGIN
8     SELECT person_id, ROUND(SUM(total_price + delivery_charge), 2) AS total_order_cost,
9           ROUND(SUM(total_price), 2) AS total_food_cost,
10          ROUND(SUM(delivery_charge), 2) AS total_delivery_cost
11     FROM `order`
12    WHERE order_date BETWEEN from_date AND to_date
13    GROUP BY person_id;
14 END //
15 DELIMITER ;
16
17 CALL total_order_cost('2021-11-24', '2021-12-03');
```

Output:

#	Time	Action	Message	Duration / Fetch
139	20:06:24	SELECT * FROM campus_eats_fall2020.order LIMIT 0, 1000	101 row(s) returned	0.000 sec / 0.000 sec
140	22:18:03	CALL customer_restaurant_rating(1, 1)	1 row(s) returned	0.078 sec / 0.000 sec

Result:

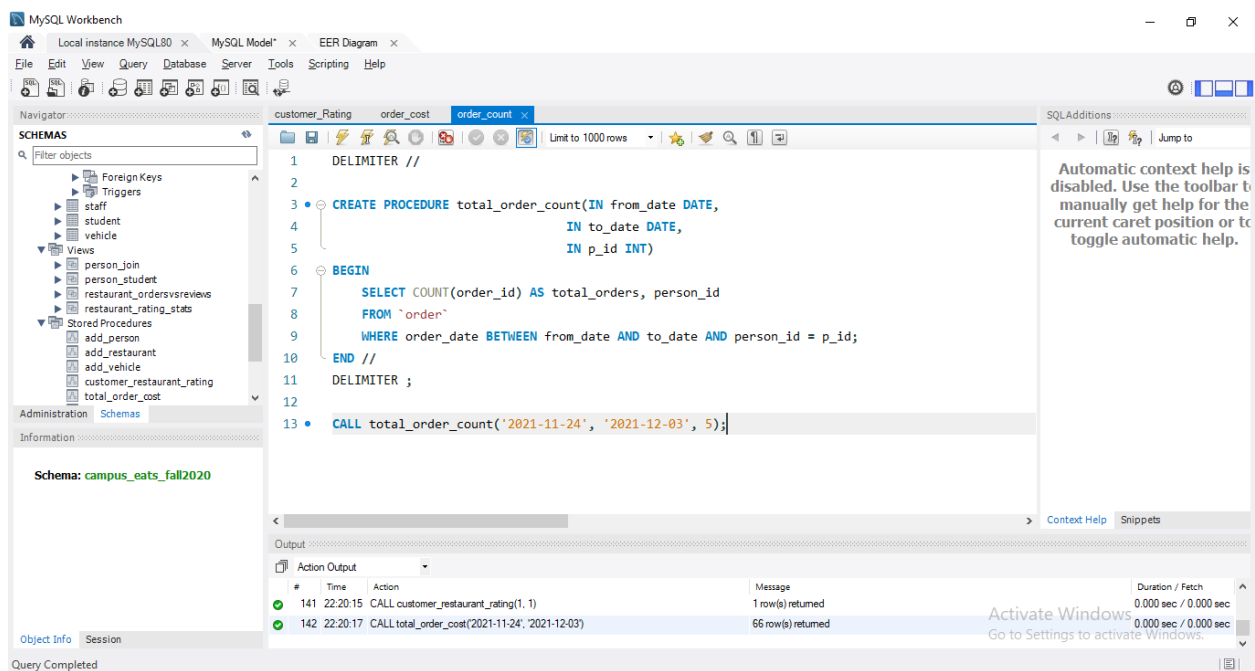
Result Grid		Filter Rows:	Export:	Wrap Cell Con
person_id	total_order_cost	total_food_cost	total_delivery_cost	
3	19.33	11.91	7.42	
5	20	13.76	6.24	
6	10.23	5.4	4.83	
7	22.62	14.05	8.57	
9	23.82	17.1	6.72	
10	14.53	12.71	1.82	
11	11.16	3.9	7.26	
14	13.29	12.08	1.21	
17	11.46	6.61	4.85	

Result 1 x

5. Name of Stored Procedure: total_order_count

Purpose: Given a date range and person_id, it will return the total number of orders placed by that customer in that time frame.

Code:



Result:

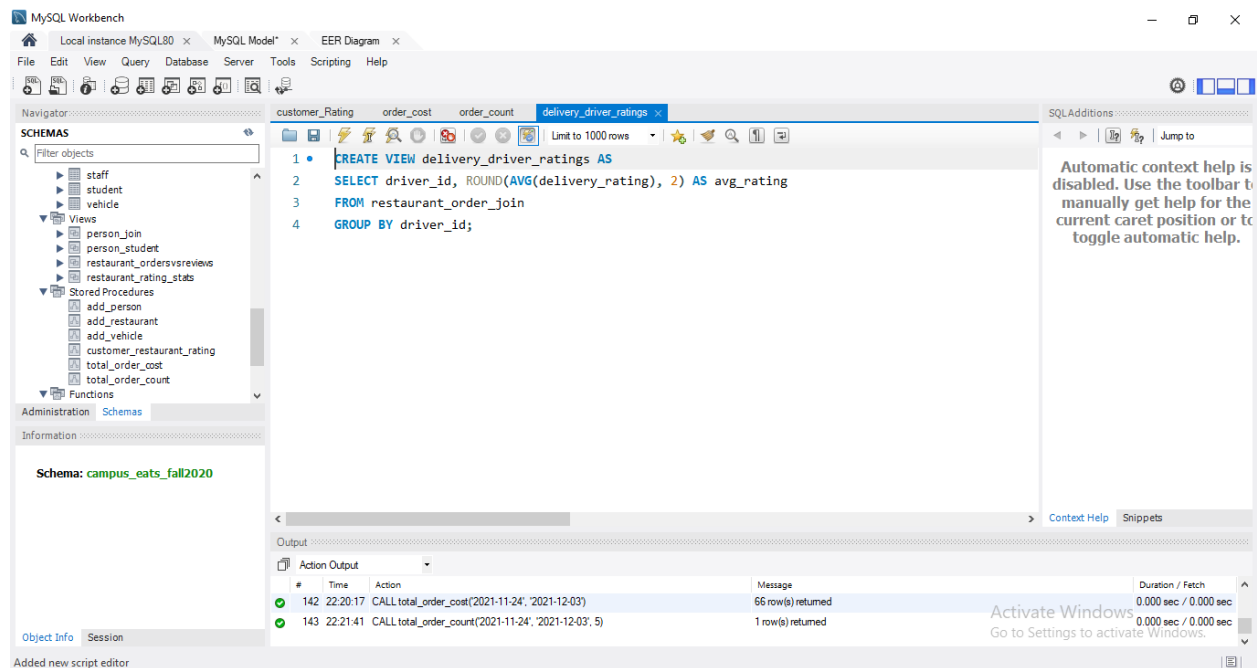
Result Grid			Filter Rows:
	total_orders	person_id	
▶	1	5	

Views:

1. Name: delivery_driver_ratings

Purpose: It shows the average performance of the driver according to the ratings given by the customer.

Code:



Result:

Result Grid			Filter Rows	
	driver_id	avg_rating		
▶	1	2.50		
	2	4.00		
	3	3.00		
	4	2.25		
	5	3.25		
	6	4.00		
	7	4.00		
	8	3.33		

2. Name: restaurant_rating_stats

Purpose: For all the restaurants, it will show the avg, min and max for the delivery rating and restaurant rating.

Code:

```
1 CREATE VIEW restaurant_rating_stats AS
2 SELECT restaurant_id, ROUND(AVG(food_rating), 2) AS avg_food_rating,
3 MIN(food_rating) AS min_food_rating, MAX(food_rating) AS max_food_rating,
4 ROUND(AVG(delivery_rating), 2) AS avg_delivery_rating,
5 MIN(delivery_rating) AS min_delivery_rating, MAX(delivery_rating) AS max_delivery_rating
6 FROM restaurant_order_join
7 GROUP BY restaurant_id;
```

View: restaurant_ordersvsreviews

Columns:

- restaurant_id int
- total_orders bigint
- total_reviews bigint

Output

#	Time	Action	Message	Duration / Fetch
147	22:28:47	CREATE VIEW restaurant_order_join AS SELECT o.order_id, rs.restaurant_id, food_r...	Error Code: 1050. Table 'restaurant_order_join' already exists.	0.015 sec
148	22:28:54	SELECT * FROM campus_eats_fal2020.restaurant_order_join LIMIT 0, 1000	30 row(s) returned	0.000 sec / 0.000 sec

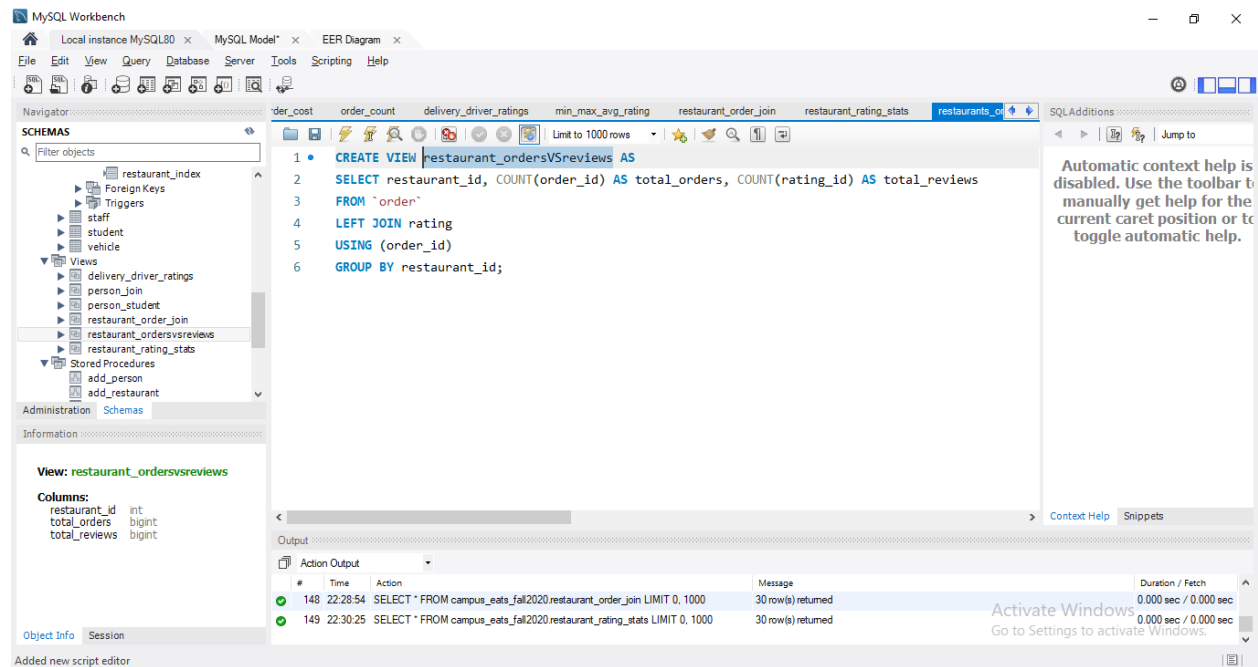
Result:

	restaurant_id	avg_food_rating	min_food_rating	max_food_rating	avg_delivery_rating	min_delivery_rating	max_delivery_rating
1	1	4.00	4	4	2.00	2	2
2	2	3.00	3	3	5.00	5	5
3	3	5.00	5	5	1.00	1	1
4	4	3.00	3	3	2.00	2	2
5	5	2.00	2	2	3.00	3	3
6	6	1.00	1	1	4.00	4	4
7	7	5.00	5	5	5.00	5	5
8	8	4.00	4	4	3.00	3	3
9	9	3.00	3	3	2.00	2	2

3. Name: restaurant_ordersVSreviews

Purpose: It will give a statistic of total number of orders and total number of views for the restaurant.

Code:



Result:

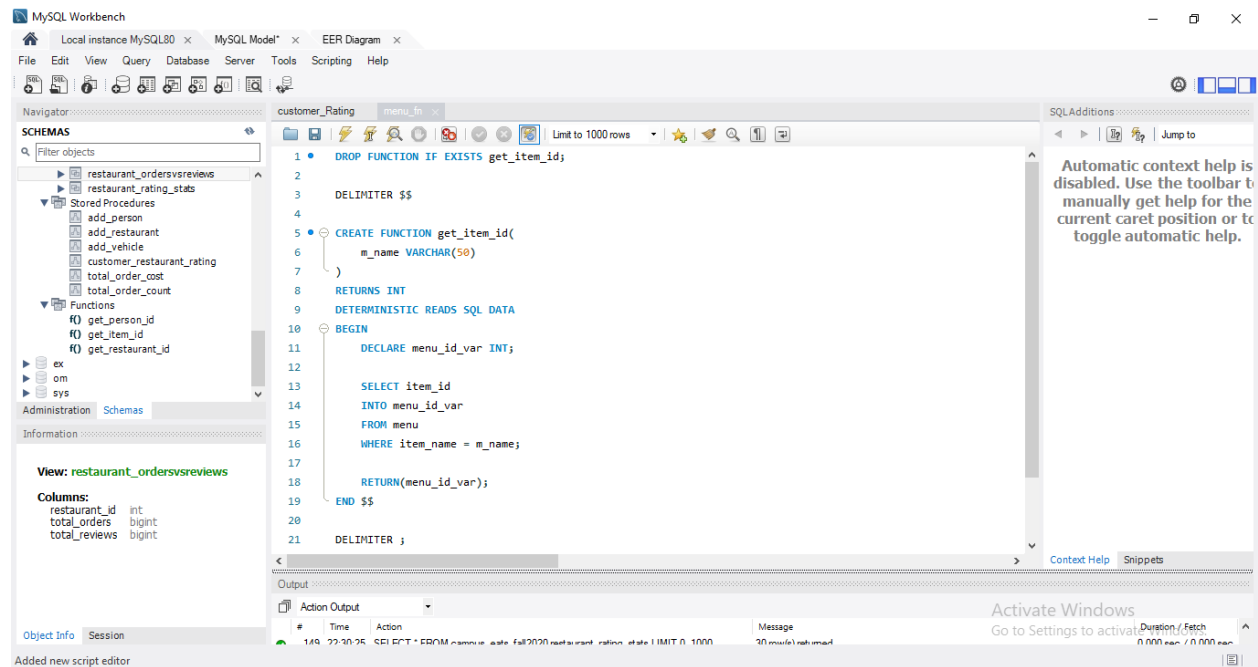
restaurant_id	total_orders	total_reviews
96	2	1
97	4	2
98	3	1
99	2	0
100	3	1

Functions:

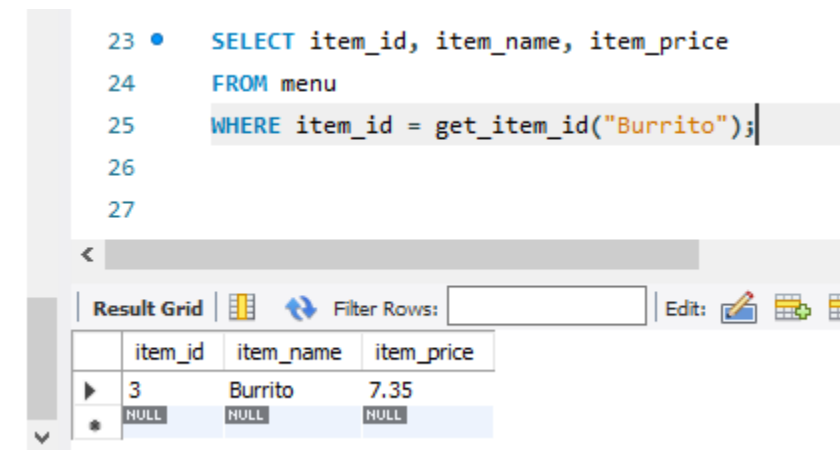
1. Name: `get_item_id`

Purpose: When inputted the name of the item, it will return the id corresponding to the item.

Code:



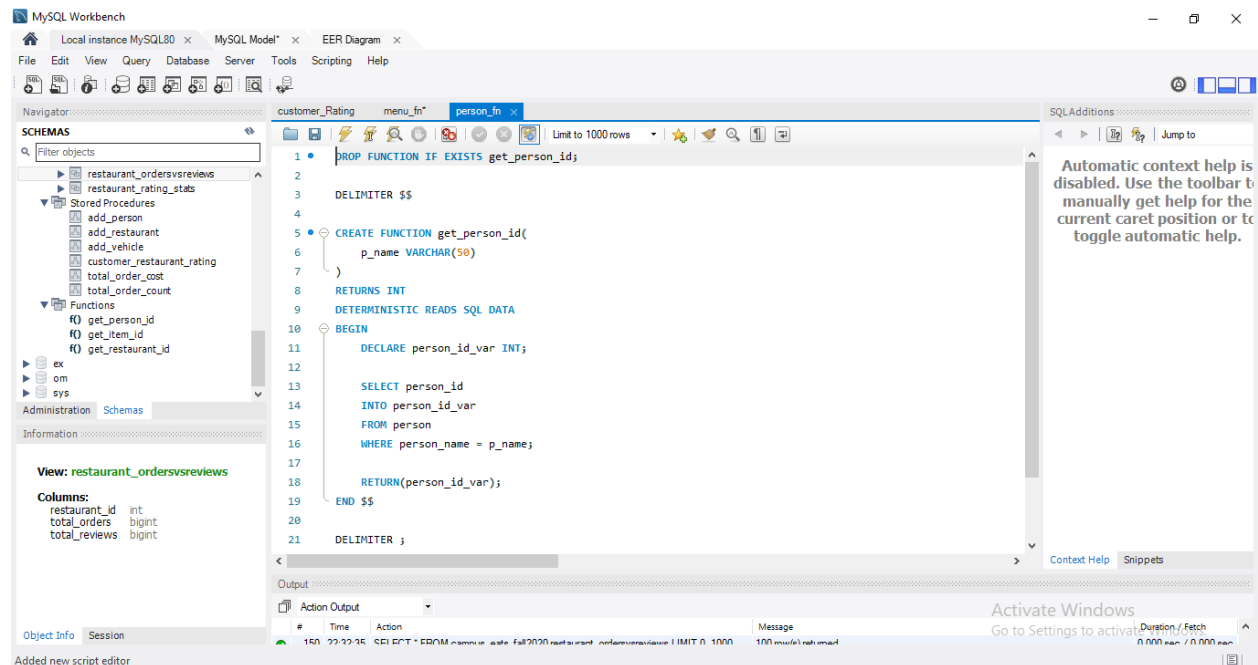
Result:



2. Name: get_person_id

Purpose: When inputted the name of person, it will return the id of the corresponding person.

Code:



Results:

```

23 SELECT person_id, person_name, person_email
24 FROM person
25 WHERE person_id = get_person_id("Keith Turner");

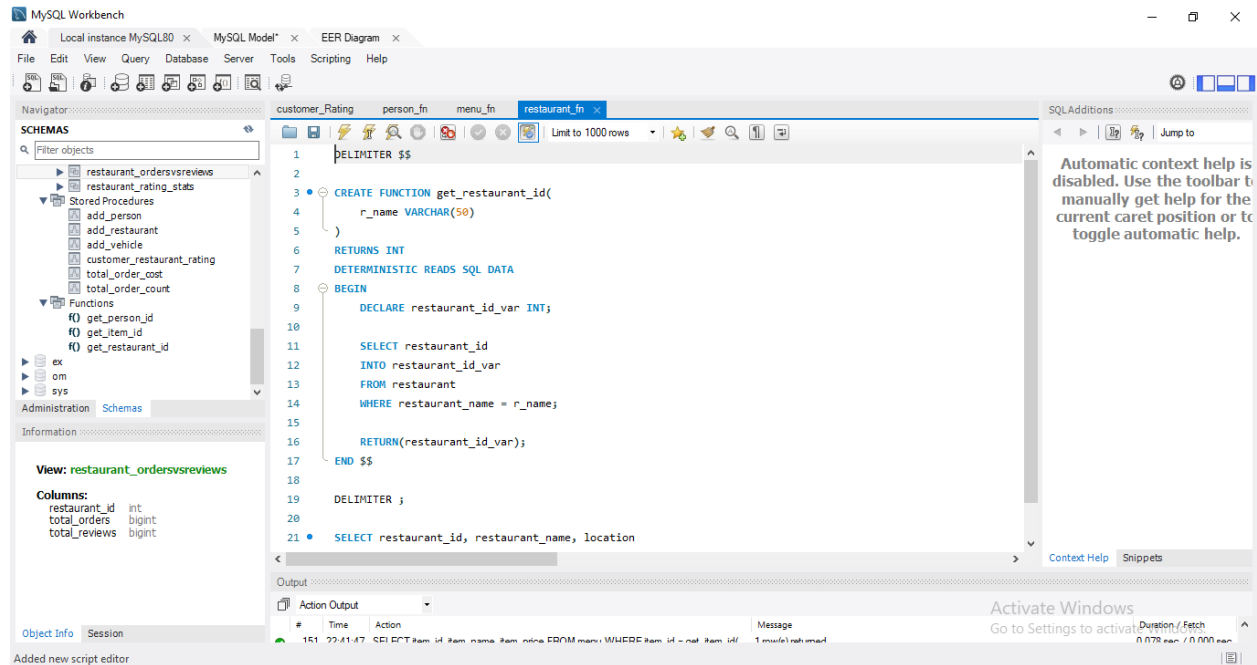
```

person_id	person_name	person_email
1	Keith Turner	shanon08@example.com
NULL	NULL	NULL

3. Name: get_restaurant_id

Purpose: When inputted the name of the restaurant, it will return the restaurant id.

Code:



Results:

```

21 • SELECT restaurant_id, restaurant_name, location
22     FROM restaurant
23     WHERE restaurant_id = get_restaurant_id("Wendys");

```

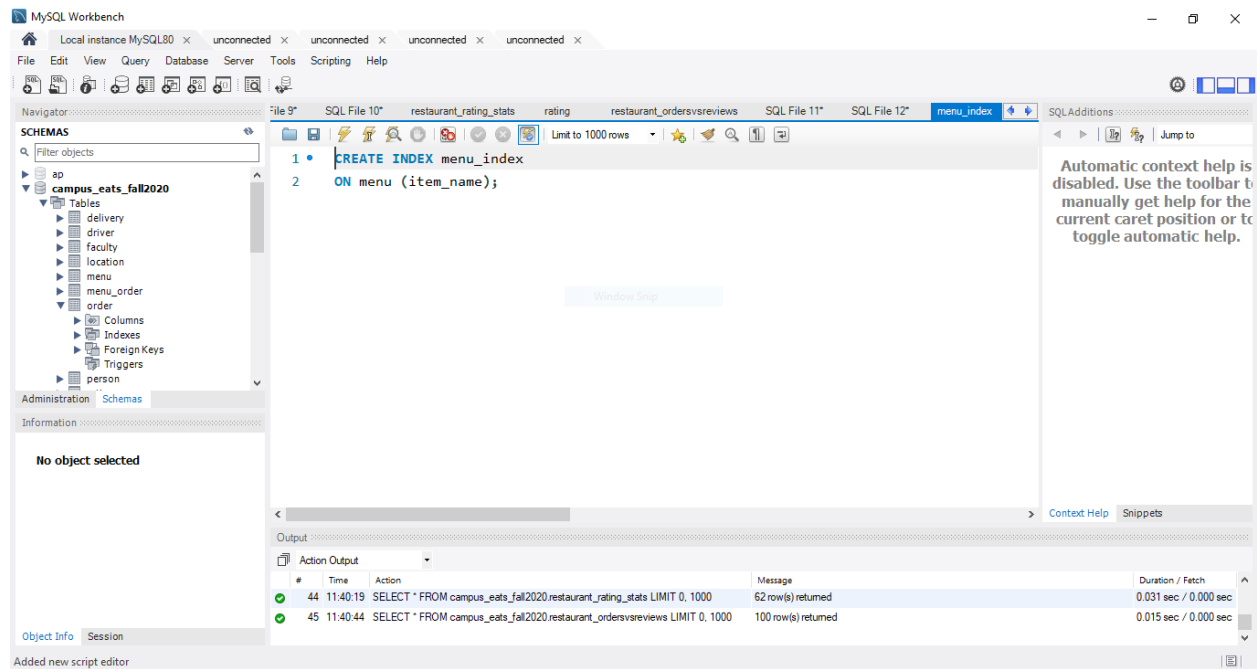
restaurant_id	restaurant_name	location
110	Wendys	UNCC Campus
NULL	NULL	NULL

Indexes:

1. Name: menu_index

Purpose: To improve the query performance, we have indexed the item_name column since it will be used multiple times while placing an order.

Code:



2. Name: restaurant_index

Purpose: To improve the query performance, we have indexed the restaurant_name since it will be used multiple times while placing an order.

Code:

MySQL Workbench

Local instance MySQL80 x unconnected x unconnected x unconnected x unconnected x

File Edit View Query Database Server Tools Scripting Help

Navigator

Filter objects

SCHEMAS

- ap
- campus_eats_fall2020
 - Tables
 - delivery
 - driver
 - faculty
 - location
 - menu
 - menu_order
 - order
 - Columns
 - Indexes
 - Foreign Keys
 - Triggers
 - person

Administration Schemas

Information

No object selected

Object Info Session

Added new script editor

file 10* restaurant_rating_stats rating restaurant_ordersvsreviews SQL File 11* SQL File 12* menu_index restaurant_index

Limit to 1000 rows

```
1 CREATE INDEX restaurant_index
2 ON restaurant (restaurant_name);
```

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
44	11:40:19	SELECT * FROM campus_eats_fall2020.restaurant_rating_stats LIMIT 0, 1000	62 row(s) returned	0.031 sec / 0.000 sec
45	11:40:44	SELECT * FROM campus_eats_fall2020.restaurant_ordersvsreviews LIMIT 0, 1000	100 row(s) returned	0.015 sec / 0.000 sec