

## Stored Procedures, Views, Indexes, Functions

### View

```
USE `campus_eats_fall2020`;

CREATE OR REPLACE VIEW order_bill AS
SELECT DISTINCT person_id as customer_id,
    ROUND(total_price + delivery_charge) AS order_cost
FROM campus_eats_fall2020.order
GROUP BY person_id;

select * from order_bill;
```

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Schemas:** Local instance MySQL80. The 'student' schema is selected, showing tables like student, Columns, graduation\_y, major, type, and foreign keys.
- SQL Editor:** Displays the SQL code for creating the 'order\_bill' view.
- Result Grid:** Shows the results of the 'select \* from order\_bill;' query, displaying 11 rows of customer\_id and order\_cost.
- Output:** Shows the execution log for the CREATE statement.
- Message:** A note about context help being disabled.

customer_id	order_cost
1	22
2	27
3	19
4	25
5	20
6	10
7	23
8	6
9	24
10	15
11	11

```
1 • USE `campus_eats_fall2020`;
2 • CREATE OR REPLACE VIEW order_bill AS
3     SELECT DISTINCT person_id as customer_id,
4         ROUND(total_price + delivery_charge) AS order_cost
5     FROM campus_eats_fall2020.order
6     GROUP BY person_id;
7 • select * from order_bill;
```

```
76 17:20:30 CREATE OR REPLACE VIEW order_bill AS SELECT DISTINCT person_id as customer_id, ROUND(total_... 0 row(s) affected
```

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

## Procedure 1

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

CALL order\_count(2015,2019,@output\_str);  
Select @output\_str

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Standard icons for file operations like Open, Save, Print, etc.
- Navigator:** Shows the database schema for the 'student' table, including columns (student\_id, person\_id, graduation\_year, major, type), indexes, foreign keys, triggers, and views.
- Procedure Editor:** The 'Procedure 1' tab contains the following SQL code:

```
1 USE campus_reta_fall2020 ;
2 DROP PROCEDURE IF EXISTS order_count;
3 DELIMITER //
4 CREATE PROCEDURE order_count(IN begin_year INT,IN final_year INT, OUT output_str varchar(100))
5 BEGIN
6     DECLARE number_of_orders Varchar(20);
7     SELECT count(*) INTO number_of_orders
8     FROM order
9     WHERE person_id IN (
10         select person_id from student where graduation_year between begin_year and final_year
11     );
12     IF number_of_orders < 0 THEN
13         SET output_str = CONCAT("The number of orders are ");
14     ELSE
15         SET output_str = CONCAT("The number of orders are ", number_of_orders);
16     END IF;
17 END //
18
19 DELIMITER ;
20 CALL order_count(2015,2019,@output_str);
21 Select @output_str
```
- Result Grid:** Shows the output of the stored procedure call: '@output\_str' is displayed as 'The number of orders are 3'.
- Result 6:** Shows the history of the last query: Action Output, showing the call to 'CALL order\_count(2015,2019,@output\_str)' at 17:32:57, with 1 row(s) affected and a duration of 0.016 sec.
- Information:** Shows 'No object selected'.
- Object Info:** Shows 'Session'.
- Status:** 'Query Completed'.

## Procedure 2

```
USE campus_eats_fall2020;
DROP PROCEDURE IF EXISTS
max_min_avg_rating_for_each_feature;
DELIMITER //

CREATE PROCEDURE max_min_avg_rating_for_each_feature (IN
restaurant_id INT(50), OUT max_food INT, OUT min_food INT,
OUT avg_food INT, OUT max_deli INT, OUT min_deli INT, OUT
avg_deli INT)

BEGIN

    SET max_food = 0;
    SET min_food = 0;
    SET avg_food = 0;
    SET max_deli = 0;
    SET min_deli = 0;
    SET avg_deli = 0;

select max(food_rating)
into max_food
from order_rating where order_id
IN (select order_id from `order` where `order`.restaurant_id = 3);

select min(food_rating)
into min_food
```

```
from order_rating where order_id  
IN (select order_id from `order` where `order`.restaurant_id = 5);
```

```
select avg(food_rating)  
into avg_food  
from order_rating where order_id  
IN (select order_id from `order` where `order`.restaurant_id = 7);
```

```
select max(delivery_rating)  
into max_deli  
from order_rating where order_id  
IN (select order_id from `order` where `order`.restaurant_id = 3);
```

```
select min(delivery_rating)  
into min_deli  
from order_rating where order_id  
IN (select order_id from `order` where `order`.restaurant_id = 7);
```

```
select avg(delivery_rating)  
into avg_deli  
from order_rating where order_id  
IN (select order_id from `order` where `order`.restaurant_id = 9);
```

```
END //
```

```
DELIMITER ;
```

CALL

```
max_min_avg_rating_for_each_feature(2,@Max_Food_Rating,@Min_Food_Rating, @Avg_Food_Rating, @Max_Deli_Rating, @Min_Deli_Rating, @Avg_Deli_Rating);
```

```
SELECT @Max_Food_Rating, @Min_Food_Rating, @Avg_Food_Rating, @Max_Deli_Rating, @Min_Deli_Rating, @Avg_Deli_Rating ;
```

The screenshot shows the MySQL Workbench interface with the 'Procedure 2' tab selected. The code pane contains the following stored procedure definition:

```
1 USE campus_restaurants;
2 DELIMITER //
3 CREATE PROCEDURE max_min_avg_rating_for_each_feature (IN restaurant_id INT(10), OUT max_food INT, OUT min_food INT, OUT avg_food INT, OUT max_deli INT, OUT min_deli INT, OUT avg_deli INT)
4 BEGIN
5     SET max_food = 0;
6     SET min_food = 0;
7     SET avg_food = 0;
8     SET max_deli = 0;
9     SET min_deli = 0;
10    SET avg_deli = 0;
11
12    select max(food_rating)
13    into max_food
14    from order_rating where order_id
15    IN (select order_id from order where order.restaurant_id = ?);
16
17    select min(food_rating)
18    into min_food
19    from order_rating where order_id
20    IN (select order_id from order where order.restaurant_id = ?);
21
22    select avg(food_rating)
23    into avg_food
24    from order_rating where order_id
25    IN (select order_id from order where order.restaurant_id = ?);
26
27    select max(delivery_rating)
28    into max_deli
29    from order_rating where order_id
30    IN (select order_id from order where order.restaurant_id = ?);
31
32    select min(delivery_rating)
33    into min_deli
34    from order_rating where order_id
35    IN (select order_id from order where order.restaurant_id = ?);
36
37    select avg(delivery_rating)
38    into avg_deli
39    from order_rating where order_id
40    IN (select order_id from order where order.restaurant_id = ?);
41
42    END //
43    DELIMITER ;
44
45 CALL max_min_avg_rating_for_each_feature(2,@Max_Food_Rating,@Min_Food_Rating, @Avg_Food_Rating, @Max_Deli_Rating, @Min_Deli_Rating, @Avg_Deli_Rating);
46
47 SELECT @Max_Food_Rating, @Min_Food_Rating, @Avg_Food_Rating, @Max_Deli_Rating, @Min_Deli_Rating, @Avg_Deli_Rating;
```

The result grid shows the output of the stored procedure:

	@Max_Food_Rating	@Min_Food_Rating	@Avg_Food_Rating	@Max_Deli_Rating	@Min_Deli_Rating	@Avg_Deli_Rating
10	1	5	9	4	8	

The screenshot shows the MySQL Workbench interface with the 'Procedure 2' tab selected. The code pane contains the same stored procedure definition as the previous screenshot. The result grid shows the output of the stored procedure:

	@Max_Food_Rating	@Min_Food_Rating	@Avg_Food_Rating	@Max_Deli_Rating	@Min_Deli_Rating	@Avg_Deli_Rating
10	1	5	9	4	8	

The status bar at the bottom indicates "Query Completed".

## Function

```
DROP FUNCTION IF EXISTS driver_expertise;
DELIMITER //
CREATE FUNCTION driver_expertise
(
rating INT
)
RETURNS varchar(150)
DETERMINISTIC
BEGIN
    DECLARE rating_feedback varchar(150);
    IF rating = 1 THEN
        SET rating_feedback = "Bad driver";
    ELSEIF rating = 2 THEN
        SET rating_feedback = "Moderate driver";
    ELSEIF rating = 3 THEN
        SET rating_feedback = "Decent driver";
    ELSEIF rating = 4 THEN
        SET rating_feedback = "Great driver";
    ELSEIF rating = 5 THEN
        SET rating_feedback = "Excellent driver";
    END IF;
    RETURN rating_feedback;
END//
DELIMITER ;
```

select driver\_id, driver\_expertise(rating) from driver;

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance MySQL80
- Tables:** campus\_eats\_fall2020 (selected), delivery, driver
- Function:** driver\_expertise
- Code:**

```
1 • DROP FUNCTION IF EXISTS driver_expertise;
2 • DELIMITER //
3 • CREATE FUNCTION driver_expertise
4 • (
5 •     rating INT
6 • )
7 • RETURNS varchar(150)
8 • DETERMINISTIC
9 • BEGIN
10 •     DECLARE rating_feedback varchar(150);
11 •     IF rating = 1 THEN
12 •         SET rating_feedback = "Bad driver";
13 •     ELSEIF rating = 2 THEN
14 •         SET rating_feedback = "Moderate driver";
15 •     ELSEIF rating = 3 THEN
16 •         SET rating_feedback = "Decent driver";
17 •     ELSEIF rating = 4 THEN
18 •         SET rating_feedback = "Great driver";
19 •     ELSEIF rating = 5 THEN
20 •         SET rating_feedback = "Excellent driver";
21 •     END IF;
22 •     RETURN rating_feedback;
23 • END//
```
- Result Grid:** Shows the output of the function for ratings 1 through 5:

driver_id	driver_expertise(rating)
1	Great driver
2	Decent driver
3	Decent driver
4	Decent driver
5	Great driver
- Action Output:** Shows the creation of the function:

#	Time	Action
106	17:56:32	CREATE FUNCTION driver_expertise ( rating INT ) RETURNS varchar(150) DETERMINISTIC BEGIN DE... 0 row(s) affected

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance MySQL80
- Tables:** campus\_eats\_fall2020 (selected), delivery, driver
- Query:** select driver\_id, driver\_expertise(rating) from driver;
- Result Grid:** Shows the output of the query for ratings 1 through 5:

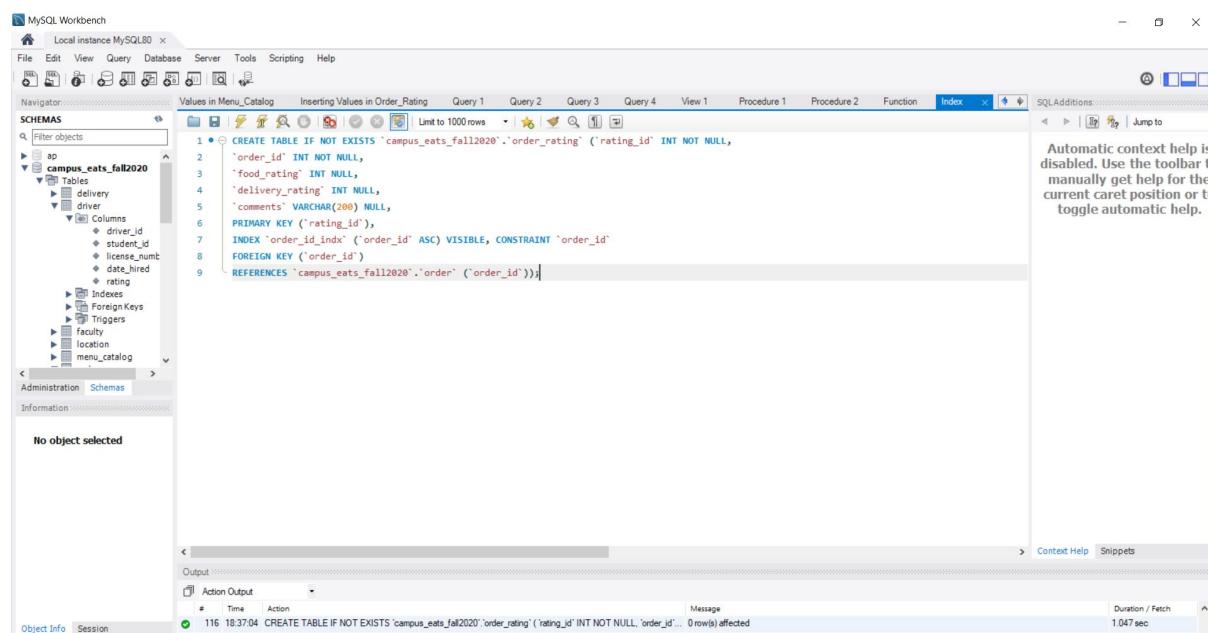
driver_id	driver_expertise(rating)
1	Great driver
2	Decent driver
3	Decent driver
4	Decent driver
5	Great driver
- Action Output:** Shows the creation of the function:

#	Time	Action
106	17:56:32	CREATE FUNCTION driver_expertise ( rating INT ) RETURNS varchar(150) DETERMINISTIC BEGIN DE... 0 row(s) affected

## Index

CREATE TABLE IF NOT EXISTS

```
'campus_eats_fall2020'.`order_rating`(`rating_id` INT NOT NULL,  
`order_id` INT NOT NULL,  
`food_rating` INT NULL,  
`delivery_rating` INT NULL,  
`comments` VARCHAR(200) NULL,  
PRIMARY KEY(`rating_id`),  
INDEX `order_id_idx`(`order_id` ASC) VISIBLE, CONSTRAINT  
`order_id`  
FOREIGN KEY(`order_id`)  
REFERENCES `campus_eats_fall2020`.`order`(`order_id`))
```



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is "campus\_eats\_fall2020".
- Tables:** The table "order\_rating" is selected.
- Query Editor:** The SQL code for creating the table is displayed:

```
CREATE TABLE IF NOT EXISTS `campus_eats_fall2020`.`order_rating`(`rating_id` INT NOT NULL,  
`order_id` INT NOT NULL,  
`food_rating` INT NULL,  
`delivery_rating` INT NULL,  
`comments` VARCHAR(200) NULL,  
PRIMARY KEY(`rating_id`),  
INDEX `order_id_idx`(`order_id` ASC) VISIBLE, CONSTRAINT  
`order_id`  
FOREIGN KEY(`order_id`)  
REFERENCES `campus_eats_fall2020`.`order`(`order_id`))
```
- Output Tab:** The output shows the execution results:

#	Time	Action	Message	Duration / Fetch
116	18:37:04	CREATE TABLE IF NOT EXISTS `campus_eats_fall2020`.`order_rating`(`rating_id` INT NOT NULL, `order_id`... 0 row(s) affected		1.047 sec