Term Assignment

LIT
LIMERICK INSTITUTE
OF TECHNOLOGY

Joe Morais

K00254840

Software Development Year 2

Data Design & Programming

1.0 EER Model

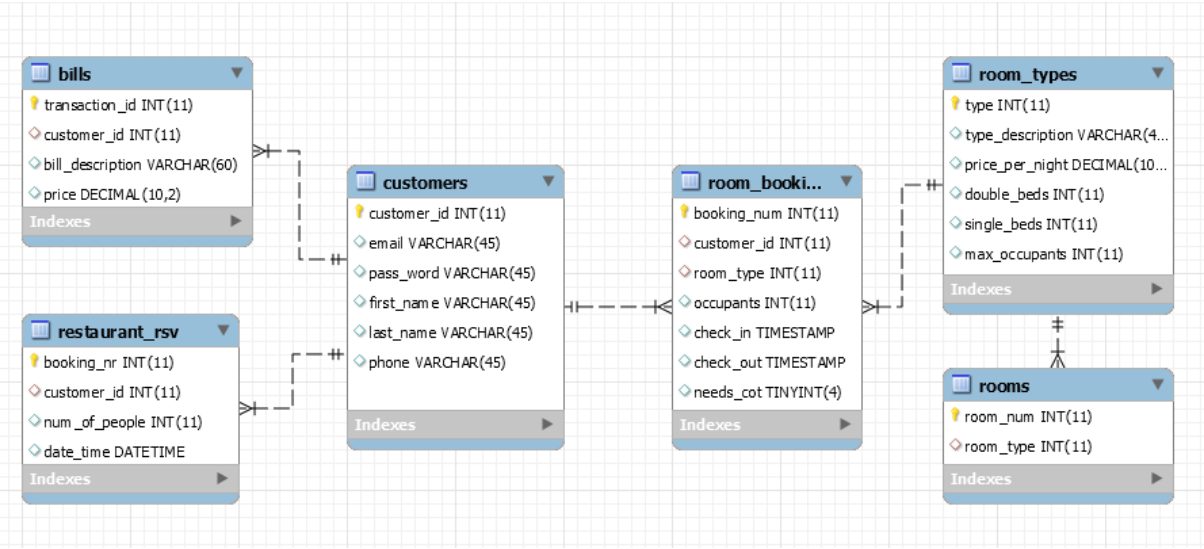


Table	**customers**
Purpose	To store required information on hotel customers.
Fields	• customer_id: Auto generated field used to uniquely identify customer accounts. • email: Stored customer email. • pass_word: Used to grant access to a specific account. • first_name: Stored customer first name. • last_name: Stored customer last name. • phone: Stored customer contact phone number.

Table	**room_bookings**
Purpose	To store required information on room bookings/reservations.
Fields	• booking_num: Auto generated field used to uniquely identify bookings. • customer_id: Used to identify who the booking is for. • room_type: Used to identify which room the customer is booking. • occupants: Used to specify the number of occupants in the room. • check_in: Booking check-in date. • check_out: Booking check-out date. • needs_cot: Specifies if room requires a cot.

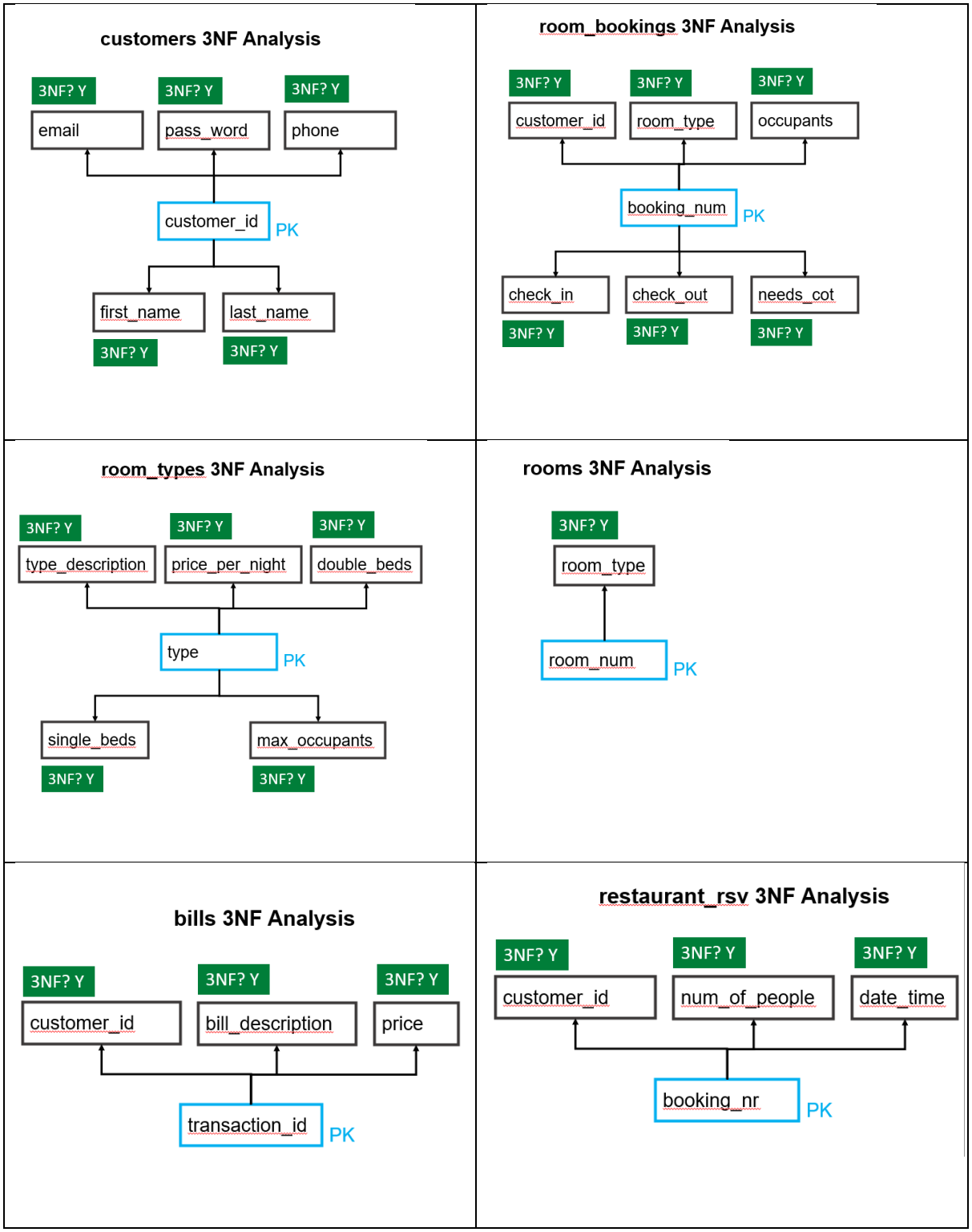
Table	**room_types**
Purpose	To store required detailed information on each room type.
Fields	• type: Unique room type identifier. • type_description: English word description of the room type. • price_per_night: The cost to stay in a particular room type per night. • double_beds: Number of double beds in the room type. • single_beds: Number of single beds in the room type. • max_occupants: The maximum amount of people the room can house.

Table	**rooms**
Purpose	A list of room numbers specifying their specific room type.
Fields	• room_num: Unique room identifier. • room_type: Specifies the room type.

Table	**bills**
Purpose	Used to temporarily store bills that have not been paid for yet.
Fields	• transaction_id: Unique bill identifier. • customer_id: Identifies who the bill belongs to. • bill_description: Describes what the bill is paying for (e.g. dinner, room) • price: The price for the bill.

Table	**restaurant_rsv**
Purpose	Used to store restaurant reservations.
Fields	• booking_nr: Unique reservation identifier. • customer_id: Identifies who the reservation is for. • num_of_people: Specifies the number of people that the reservation is for. • date_time: Specifies the date and time of the reservation.

2.0 Normalization



3.0 Relational Notation

- **Database:**
  - **K00254840_sd_arms =** {customers, room_bookings, room_types, rooms, bills, restaurant_rsv }
- **Tables:**
  - **customers** = {customer_id, email, pass_word, first_name, last_name, phone}
    - **Data Types**
      - customer_id: INT
      - email: VARCHAR(45)
      - pass_word: VARCHAR(45)
      - first_name: VARCHAR(45)
      - last_name: VARCHAR(45)
      - phone: VARCHAR(45)
  - **room_bookings** = {booking_num, customer_id, room_type, occupants, check_in, check_out, needs_cot}
    - **Data Types**
      - booking_num: INT
      - customer_id: INT
      - room_type: INT
      - occupants: INT
      - check_in: DATETIME
      - check_out: DATETIME
      - needs_cot: BOOLEAN
  - **room_types** = {type, type_description, price_per_night, double_beds, single_beds, max_occupants}
    - **Data Types**
      - type: INT
      - type_description: VARCHAR(45)
      - price_per_night: DECIMAL
      - double_beds: INT
      - single_beds: INT
      - max_occupants: INT
  - **rooms** = {rom_num, room_type}
    - **Data Types**
      - room_num: INT
      - room__type: INT

- **bills** = {transaction_id, customer_id, bill_description, price}
  - **Data Types**
    - transaction_id: INT
    - customer_id INT
    - bill_description: VARCHAR(60)
    - price: DECIMAL
- **restaurant_rsv** = {booking_nr, customer_id, num_of_people, date_time}
  - **Data Types**
    - booking_nr: INT
    - customer_id: INT
    - num_of_people: INT
    - date_time: DATETIME

4.0 Implementation & Testing

Test Data: bills

transaction_id	customer_id	bill_description	price
73	3	Dinner	24.50
74	15	Breakfast	15.60
75	12	Dinner	25.00
76	20	Beer	5.50
77	18	Dinner	26.00
78	17	Lunch	23.10
79	4	Dinner	50.50
80	6	Beer	5.60
81	9	Dinner	21.50
82	10	Lunch	10.50
NULL	NULL	NULL	NULL

Test Data: customers

customer_id	email	pass_word	first_name	last_name	phone
1	ahounsham0@behance.net	RBp9QTbS	Abbe	Hounsham	9839576917
2	kscrimshaw1@netvibes.com	M7W53V02XSq	Karoline	Scrimshaw	7237456463
3	rpirazzi2@symantec.com	RQ7sk7wYwD	Reggie	Pirazzi	8563629684
4	dyuryatin3@dropbox.com	zvEEZCF4yR	Duky	Yuryatin	9463903784
5	atregensoe4@stumbleupon.com	9ly2J2	Alexandro	Tregensoe	8112427590
6	jbygraves5@hubpages.com	8Pyg9ZZ6sm	Janessa	Bygraves	9925927782
7	bkleinstub6@indiatimes.com	rQbXXix2iVG	Becky	Kleinstub	2796243375
8	icaldwall7@toplist.cz	2GE14Jt	Izabel	Caldwall	3159849440
9	mfittall8@yellowpages.com	nXzEX52RaHA	Micaela	Fittall	2577499322
10	cwaddell9@tripod.com	Bg9d8T8c7	Catie	Waddell	2085503819
11	hlarchera@globo.com	ZQcPDTcDSu	Hildy	Larcher	6465681758
12	bcastanob@sciencedirect.com	uSkmUb	Brandi	Castano	8031881551
13	sgandertonc@nymag.com	q6PlaFpY	Sauveur	Ganderton	4757735006
14	pcockerd@macromedia.com	4pnlFbqIfIRA	Peggie	Cocker	1962154659
15	dseaverse@mayoclinic.com	zyoO6DmZQW7	Dennis	Seavers	9542316917
16	nchattawayf@blogger.com	UKFIQ6P	Nataline	Chattaway	7307866791
17	jalamg@yale.edu	voabnbV	Julie	Alam	6603901934
18	jbirdish@cocolog-nifty.com	uWTV1Csbxz	Jimmie	Birdis	7926482553
19	clegieri@mapquest.com	q5TzWEzy	Carilyn	Legier	5069510342
20	acasarilij@moonfruit.com	nSNNcVdZ	Adham	Casarili	2575239081
21	joe.morais@gmail.com	joe123	Joe	Morais	0891234567
24	elon.musk@gmail.com	1234	Elon	Musk	0891234567
NULL	NULL	NULL	NULL	NULL	NULL

Test Data: restaurant_rsv

booking_nr	customer_id	num_of_people	date_time
1	21	3	2020-12-24 12:00:00
2	21	1	2020-11-11 12:00:00
3	10	2	2020-10-03 12:00:00
4	11	2	2021-10-03 12:00:00
5	20	2	2020-09-03 12:00:00
6	13	4	2020-11-03 12:00:00
7	17	5	2020-12-13 12:00:00
8	19	6	2021-01-03 12:00:00
9	9	2	2021-02-13 12:00:00
10	3	1	2021-03-05 12:00:00
NULL	NULL	NULL	NULL

Test Data: room_bookings

booking_num	customer_id	room_type	occupants	check_in	check_out	needs_cot
102	21	2	2	2020-12-24 12:00:00	2020-12-24 12:00:00	0
103	13	1	1	2020-12-24 12:00:00	2020-12-24 12:00:00	0
104	9	2	2	2021-01-10 12:00:00	2021-01-11 12:00:00	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Test Data: room_types

type	type_descri	price_per_night	double_beds	single_beds	max_occupants
1	Twin	89.00	0	2	2
2	Double	100.00	1	1	3
3	Family	125.00	1	2	4
NULL	NULL	NULL	NULL	NULL	NULL

Test Data: rooms

room_num	room_type
101	2
102	2
103	2
104	1
105	1
106	3
107	3
108	2
109	2
110	1
201	2
202	2
203	2
204	1
205	1
206	3
207	2
208	2
209	1
210	3
301	1
302	1
303	2
304	2
305	2
306	2
307	1
308	1
309	3
310	3
NULL	NULL

Stored Procedure: book_a_room

This procedure creates a booking for a room. It checks if there are enough rooms of a specific type and if there are available cots for the specified date range. It also adds the room cost to the customer's bill.

```
1 • CALL `k00254840_sd_arms`.`book_a_room`('elon.musk@gmail.com', 2, 2, '2020-12-24 12:00',
2   '2020-12-26 12:00', 0);
3
```

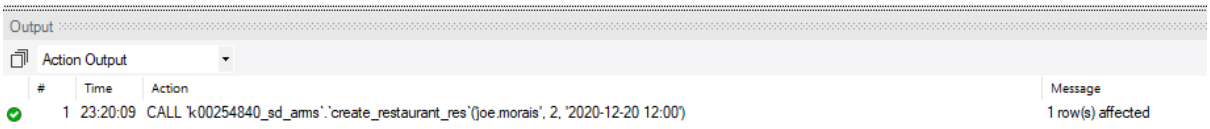

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `book_a_room`(
      IN user_email VARCHAR(45),
    IN type_of_room INT,
    IN num_of_people INT,
    IN check_in_date DATETIME,
    IN check_out_date DATETIME,
    IN need_cot TINYINT)
BEGIN
      # Check room and cot availability
      IF ((count_available_cots(check_in_date, check_out_date) >= 1)
AND
         (count_available_rooms(type_of_room, check_in_date,
check_out_date) >= 1)) THEN
            # Add customer and booking details to room_bookings
            INSERT INTO room_bookings (customer_id, room_type,
occupants, check_in, check_out, needs_cot)
            VALUES (
                  (SELECT customer_id FROM customers WHERE email =
user_email),
                  type_of_room,
                  num_of_people,
                  check_in_date,
                  check_out_date,
                  need_cot);

            # Add room price to customer bill
            INSERT INTO bills (customer_id, bill_description, price)
            VALUES(
                  (SELECT customer_id FROM customers WHERE email =
user_email),
                  'Room booking',
                  (SELECT price_per_night FROM room_types WHERE type =
type_of_room));
      END IF;
    # Print cot error message
      IF (count_available_cots(check_in_date, check_out_date) < 1)
THEN
            SELECT ('There are not enough cots for this date range!')
AS Console;
```

```
      END IF;
    # Print room error message
      IF (count_available_rooms(type_of_room, check_in_date,
check_out_date) < 1) THEN
            SELECT ('There are not enough rooms of this type for this
date range!') AS Console;
      END IF;
END
```

Stored Procedure: create_restaurant_res

This procedure creates a restaurant reservation. Further functionality and checks could have been added such as automatic table assignment and calculating the number of tables required for the reservation.

```
1 •  CALL `k00254840_sd_arms`.`create_restaurant_res`('joe.morais', 2, '2020-12-20 12:00');
2
```




```
CREATE DEFINER=`root`@`localhost` PROCEDURE `create_restaurant_res`(
      IN customer_email VARCHAR(45),
    IN num_of_people INT,
    IN date_time DATETIME
)
BEGIN


      INSERT INTO restaurant_rsv(customer_id, num_of_people,
date_time)
    VALUES(
            (SELECT c.customer_id FROM customers c WHERE c.email =
customer_email),
        num_of_people,
        date_time
    );
END
```

Stored Procedure: pay_bills

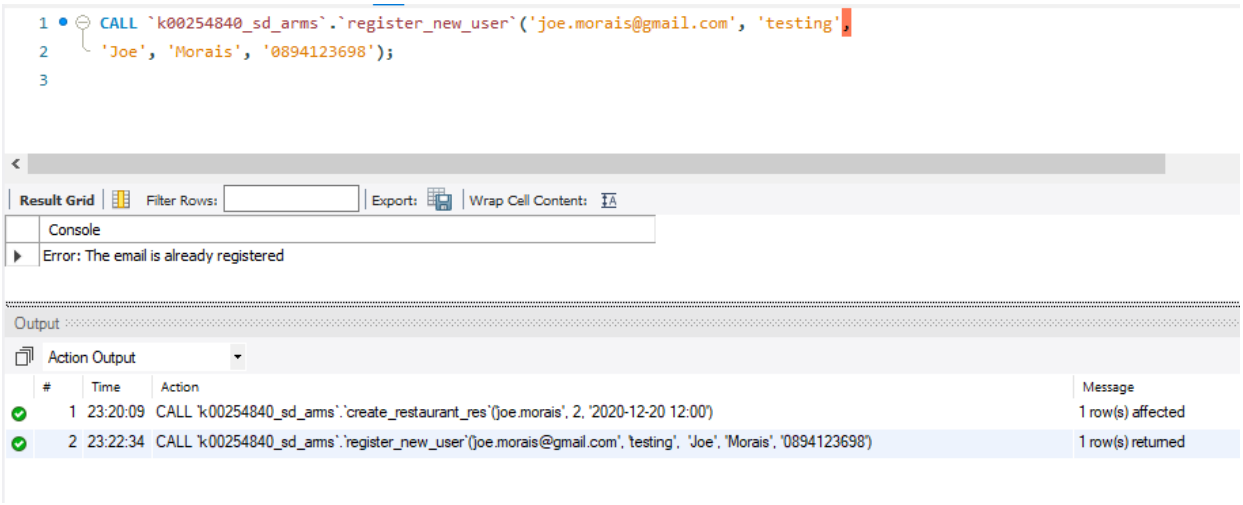
This procedure takes in a user email and returns the total amount owed. It then deletes the records from the bills table.

```
1 • CALL `k00254840_sd_arms`.`pay_bills`('joe.morais@gmail.com');
2
```


```
CREATE DEFINER=`root`@`localhost` PROCEDURE `pay_bills`(
	IN customer_email VARCHAR(45))
BEGIN
	SELECT
		SUM(price) AS Total
	FROM
		bills b
	WHERE
		b.customer_id = (SELECT customer_id FROM customers WHERE
email = customer_email);
	DELETE FROM bills WHERE customer_id = (SELECT customer_id FROM
customers WHERE email = customer_email);
END
```

Stored Procedure: register_new_user

This procedure is used to create new user accounts. If the email is already registered, the procedure displays an error.




```
CREATE DEFINER=`root`@`localhost` PROCEDURE `register_new_user`(
      IN user_email VARCHAR(45),
      IN user_pass_word VARCHAR(45),
    IN user_first_name VARCHAR(45),
    IN user_last_name VARCHAR(45),
    IN user_phone VARCHAR(45))
BEGIN
      IF (SELECT email FROM customers WHERE email = user_email) =
user_email THEN
            SELECT 'Error: The email is already registered' AS
Console;
    ELSE
            INSERT INTO customers (email, pass_word, first_name,
last_name, phone)
            VALUES (user_email, user_pass_word, user_first_name,
user_last_name, user_phone);
```

```
            SELECT 'The account was created successfully!' AS
Console;
    END IF;
END
```

Stored Procedure: room_details

This is a simple procedure used to return the room types, their prices, and the number of beds. This might be used on the web app to display the available room types to users.

```
CALL `k00254840_sd_arms`.`room_details`();
```

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

type_descri	price_per_night	double_beds	single_beds
Twin	89.00	0	2
Double	100.00	1	1
Family	125.00	1	2

Result 2

Output

Action Output

#	Time	Action	Message
1	23:26:45	CALL `k00254840_sd_arms`.`room_details`()	3 row(s) returned

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `room_details`()
BEGIN
      SELECT
            type_description, price_per_night, double_beds,
single_beds
      FROM
            room_types;
END
```

Stored Procedure: verify_login

This is a simple procedure used to check whether the provided login details are correct.

```
CALL `k00254840_sd_arms`.`verify_login`('joe.morais@gmail.com', 'hmm no');
```

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

Console
Wrong email or password.

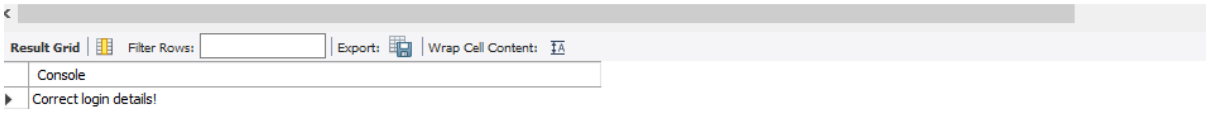
Result 2

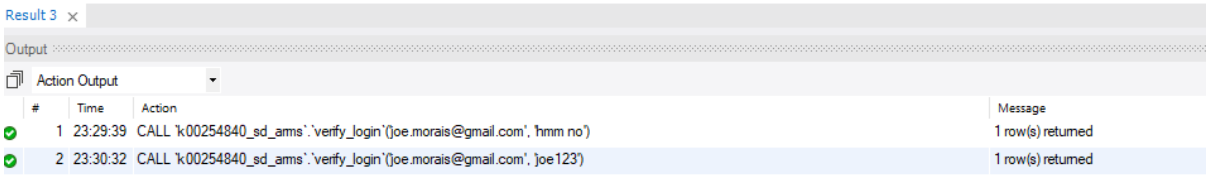
Output

Action Output

#	Time	Action	Message
1	23:29:39	CALL `k00254840_sd_arms`.`verify_login`('joe.morais@gmail.com', 'hmm no')	1 row(s) returned

```
1 • CALL `k00254840_sd_arms`.`verify_login`('joe.morais@gmail.com', 'joe123');
2
```






```
CREATE DEFINER=`root`@`localhost` PROCEDURE `verify_login`(
      IN user_email VARCHAR(45),
      IN user_pass_word VARCHAR(45))
BEGIN
      IF    (SELECT email FROM customers WHERE email = user_email) =
user_email
            AND
            (SELECT pass_word FROM customers WHERE pass_word =
user_pass_word) = user_pass_word
            THEN
                  SELECT 'Correct login details!' AS Console;
    ELSE
            SELECT 'Wrong email or password.' AS Console;
    END IF;
END
```