University of Jeddah
College of Computer Science and Engineering
Department of Computer Science and Artificial Intelligence
Introduction to database | CCCS 215

PODCASTS DATABASE

Final report

Prepared for: Course project for Introduction to database - CCCS 215

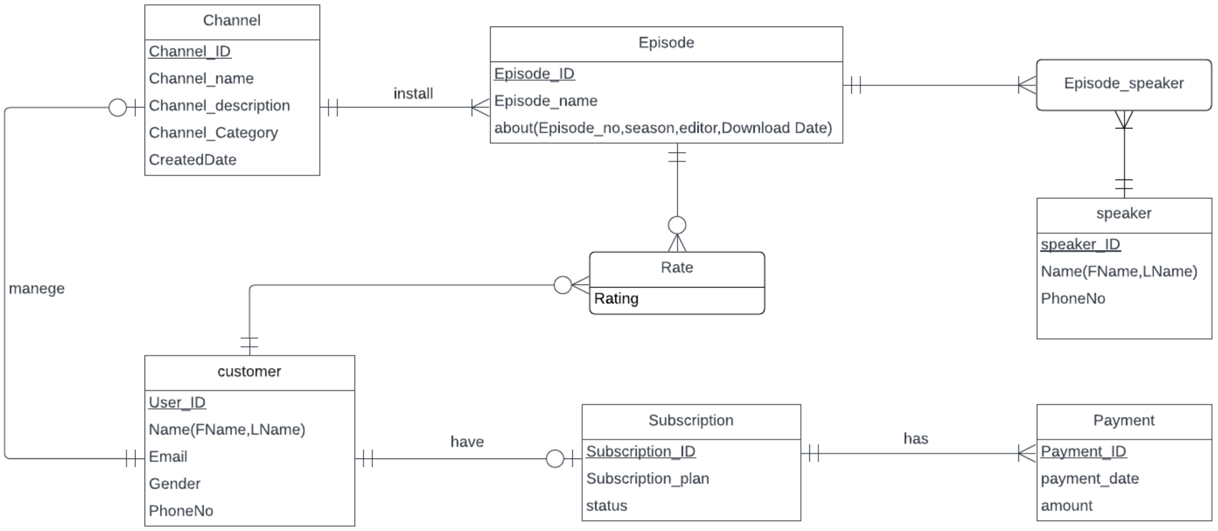
| Podcasts Database

Podcasts are tremendously helpful, impactful, and motivating in the eyes of today's listeners. Building a podcast database is an excellent opportunity to assist several parties, whether they are developers seeking to program a podcast platform or start-ups hoping to launch podcast-related enterprises. Moreover, the podcast provides a very valuable opportunity to benefit from information rich in knowledge, science and culture, and it also allows many and varied options to take advantage of the time in hearing your favorite kind of speech, whether it is in self-development, learning new skills, or even just hearing someone talk to you in a comfortable way.

| Database Entities:

Channel	Each channel must contain one or more episodes. Each channel is linked to one customer
Episode	Each episode has one or more speaker. Each Episode can have one or more ratings
Speaker	Each speaker must participate in one or more podcast episode.
Customer	Each customer can have one or more ratings Each customer may be linked to one channel.
Subscription	Each subscription will be for one and only one customer. Each subscription must have one or more payment.
Payment	Each Payment must be for one and only one subscription.
Episode Speaker (Associative entity)	Describes the relationship between the Episode and the Speaker
Rate (Associative entity)	Describes the relationship between the Episode and the Customer

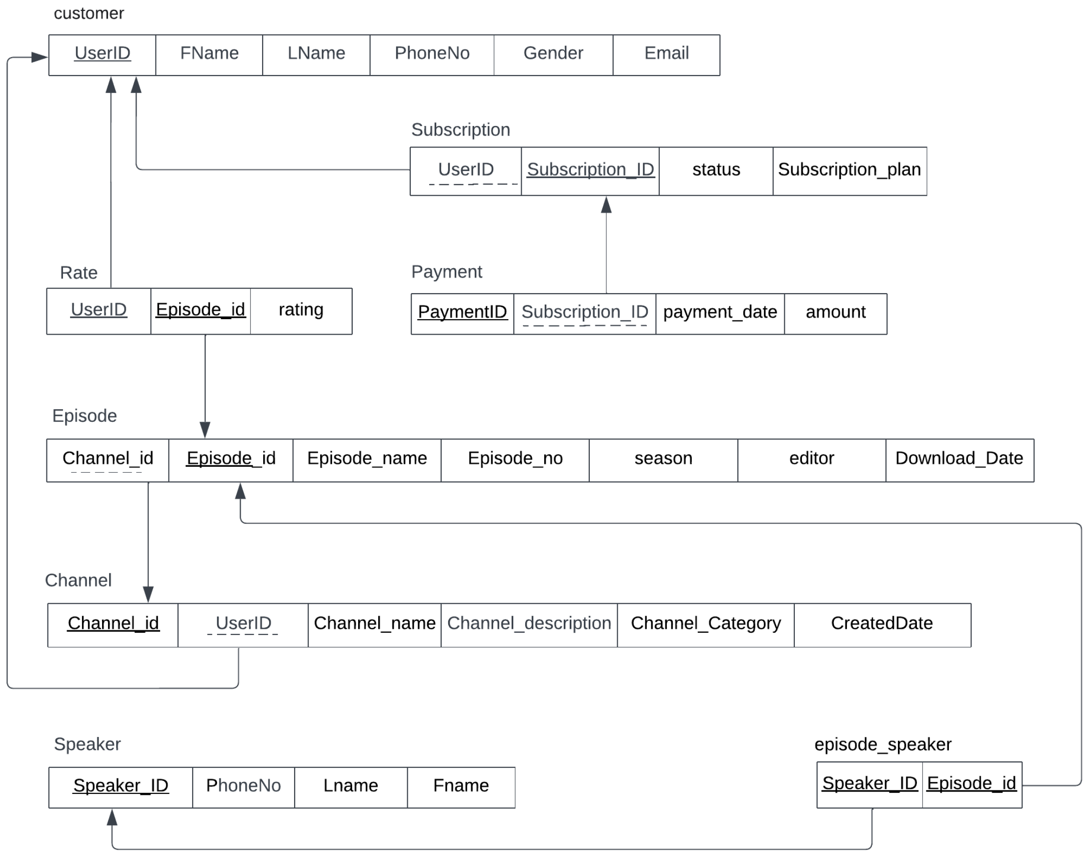
| ER Diagram:



Here is the link to view the ER model:

https://lucid.app/lucidchart/8d68a823-6fa3-4317-b130-f38fbc3e424a/edit?viewport_loc=-579%2C-3%2C2952%2C1023%2C0_0&invitationId=inv_aefa36a8-f65e-4770-860e-af17bb1ca98a

| Relational Diagram:



Here is the link to view relational diagram;

https://lucid.app/users/registerOrLogin/free?showLogin=false&invitationId=inv_19ee42df-9692-4d28-b480-0d9d0a3a97ea&productOpt=chart&invitationType=documentAcceptance&returnUrlOverride=%2Flucidchart%2Fc2a600a9-a12b-4541-a483-fdc0871a4db1%2Fedit%3Fviewport_loc%3D-149%252C-32%252C2261%252C915%252C0_0%26invitationId%3Dinv_19ee42df-9692-4d28-b480-0d9d0a3a97ea

| Functional Dependency:

- UserID -> FName, LName, Email, Gender ,PhoneNo
- Channel_ID -> Channel_name , Category , CreatedDate , Channel_description
- Episode_ID -> Episode_title ,Episode_No, season, editor ,download_date
- Episode_id , UserID -> rating
- speaker_ID -> FName, LName, phone_Num
- Subscription_ID ->subscription_ plan, status
- Payment_ID -> amount , payment_date
- Note: episode_speaker consists of composite primary keys, hence no attribute inside the entity

| Normalization:

- First Normal Form

  Our relational model is in the first normal form, there are no composite or multi-valued attributes.

- Second Normal Form

  There are no partial dependencies in our relational model.

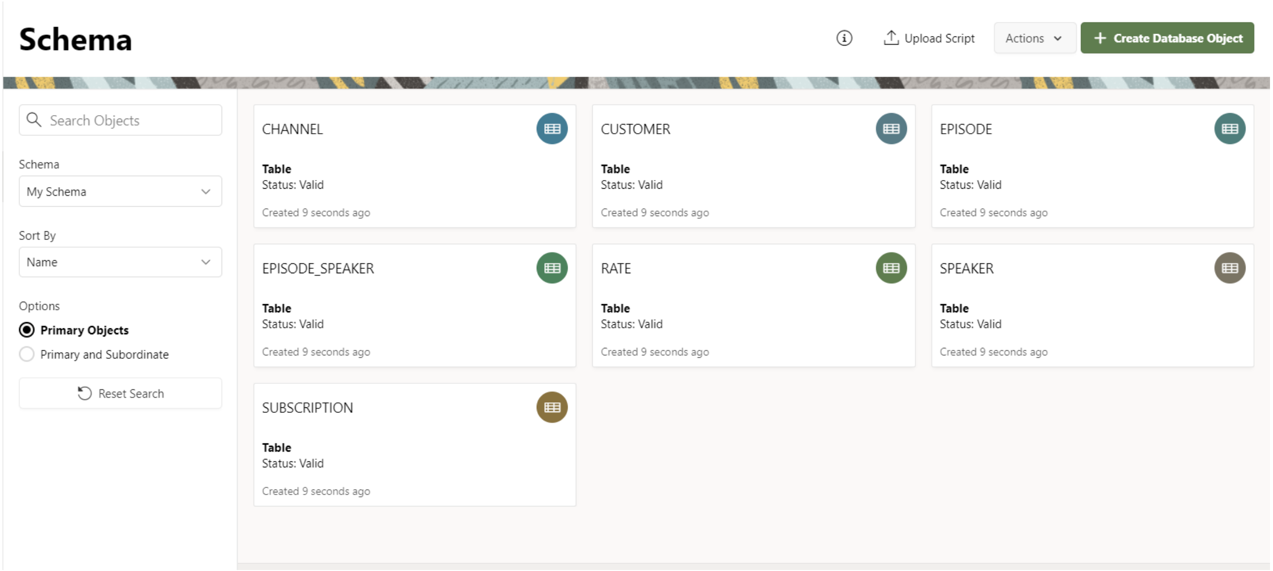
- Third Normal Form

  Since we do not have any transitive dependency to eliminate, it is in the third normal form

SQL statements:

1. Creating tables with constraints:

Schema



Schema: My Schema

Sort By: Name

Options

● **Primary Objects**

○ Primary and Subordinate

CHANNEL
Table
Status: Valid
Created 9 seconds ago

CUSTOMER
Table
Status: Valid
Created 9 seconds ago

EPISODE
Table
Status: Valid
Created 9 seconds ago

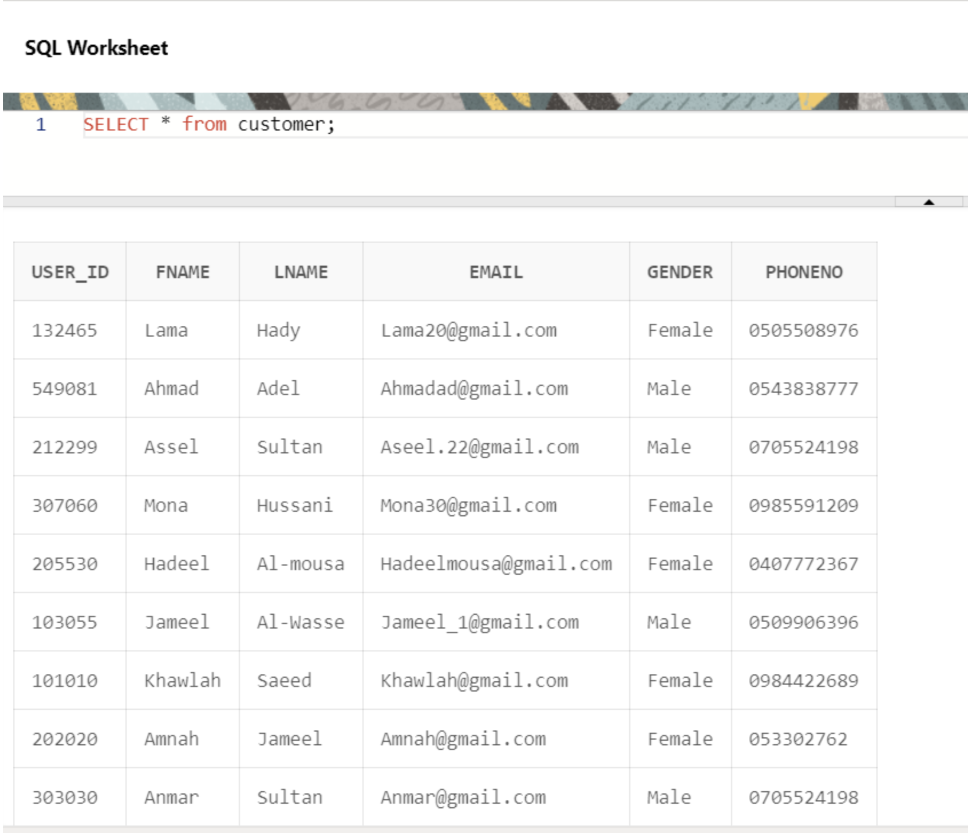
EPISODE_SPEAKER
Table
Status: Valid
Created 9 seconds ago

RATE
Table
Status: Valid
Created 9 seconds ago

SPEAKER
Table
Status: Valid
Created 9 seconds ago

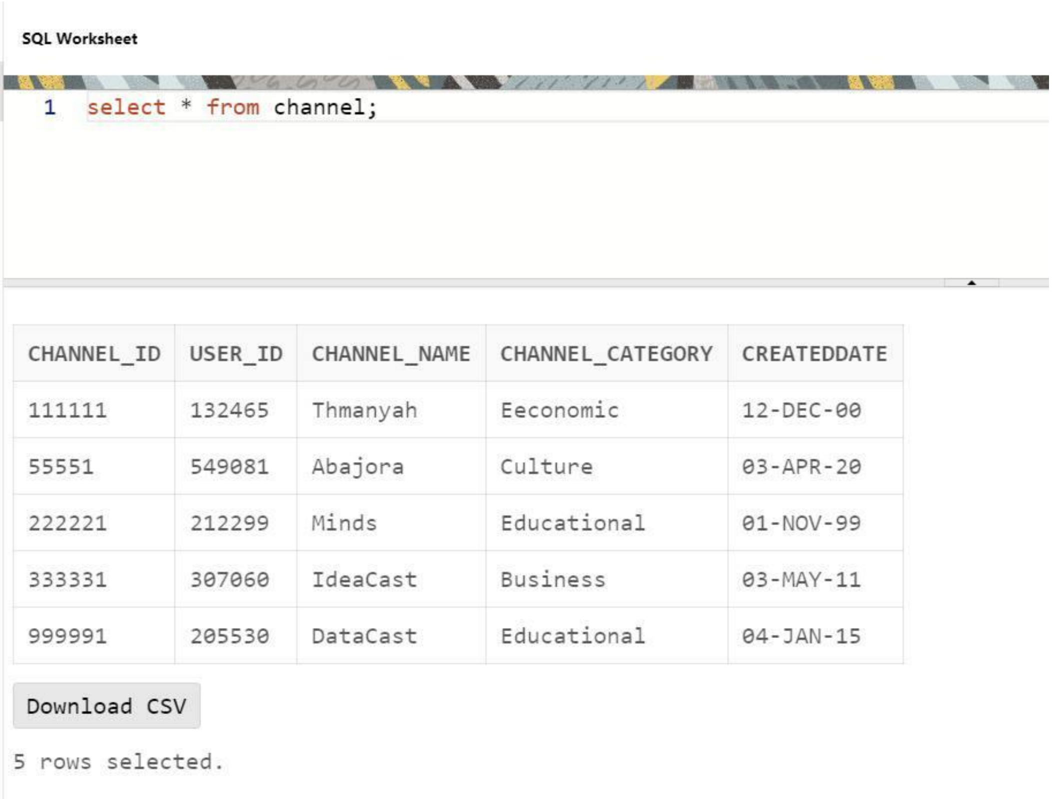
SUBSCRIPTION
Table
Status: Valid
Created 9 seconds ago

2. Inserting data in tables:



```sql
SELECT * from customer;
```

USER_ID	FNAME	LNAME	EMAIL	GENDER	PHONENO
132465	Lama	Hady	Lama20@gmail.com	Female	0505508976
549081	Ahmad	Adel	Ahmadad@gmail.com	Male	0543838777
212299	Assel	Sultan	Aseel.22@gmail.com	Male	0705524198
307060	Mona	Hussani	Mona30@gmail.com	Female	0985591209
205530	Hadeel	Al-mousa	Hadeelmousa@gmail.com	Female	0407772367
103055	Jameel	Al-Wasse	Jameel_1@gmail.com	Male	0509906396
101010	Khawlah	Saeed	Khawlah@gmail.com	Female	0984422689
202020	Amnah	Jameel	Amnah@gmail.com	Female	053302762
303030	Anmar	Sultan	Anmar@gmail.com	Male	0705524198

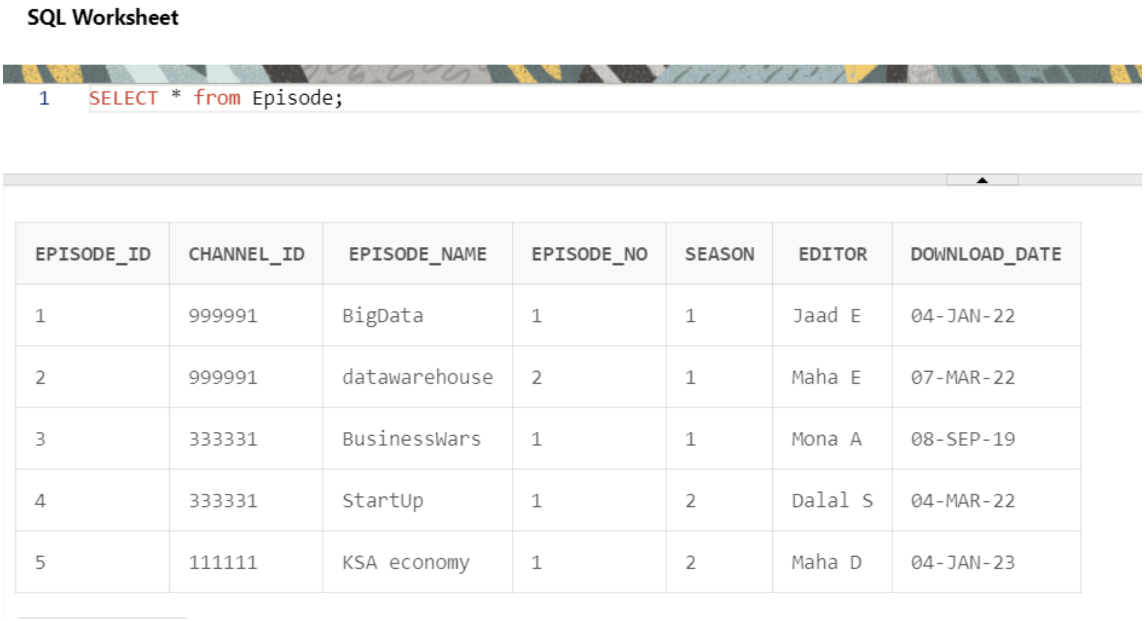


SQL Worksheet

```
select * from channel;
```

CHANNEL_ID	USER_ID	CHANNEL_NAME	CHANNEL_CATEGORY	CREATEDDATE
111111	132465	Thmanyah	Eeconomic	12-DEC-00
55551	549081	Abajora	Culture	03-APR-20
222221	212299	Minds	Educational	01-NOV-99
333331	307060	IdeaCast	Business	03-MAY-11
999991	205530	DataCast	Educational	04-JAN-15

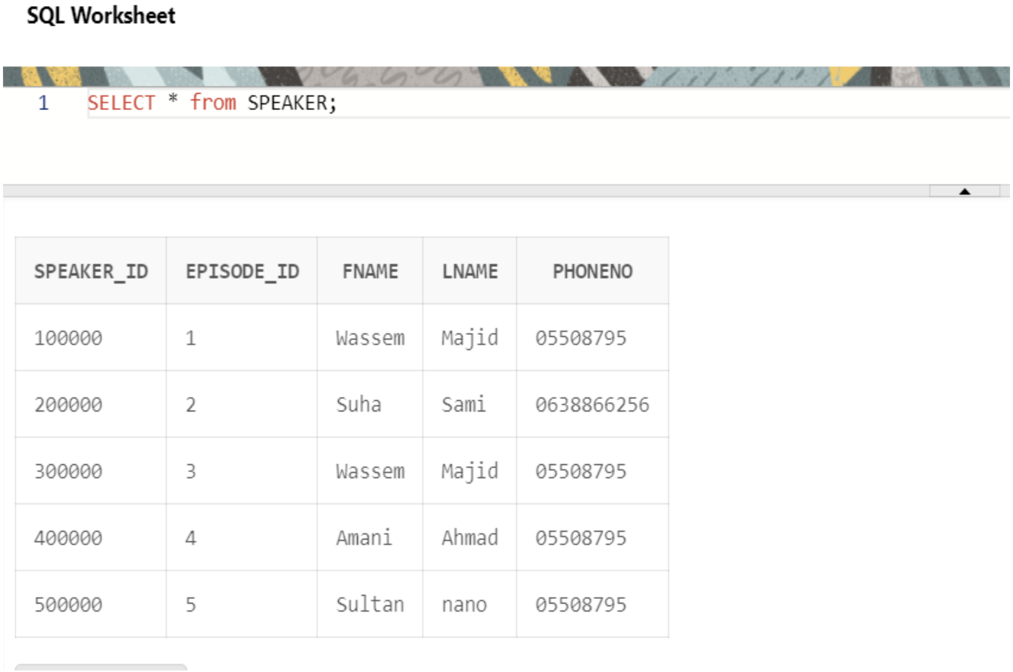
5 rows selected.



SQL Worksheet

```
SELECT * from Episode;
```

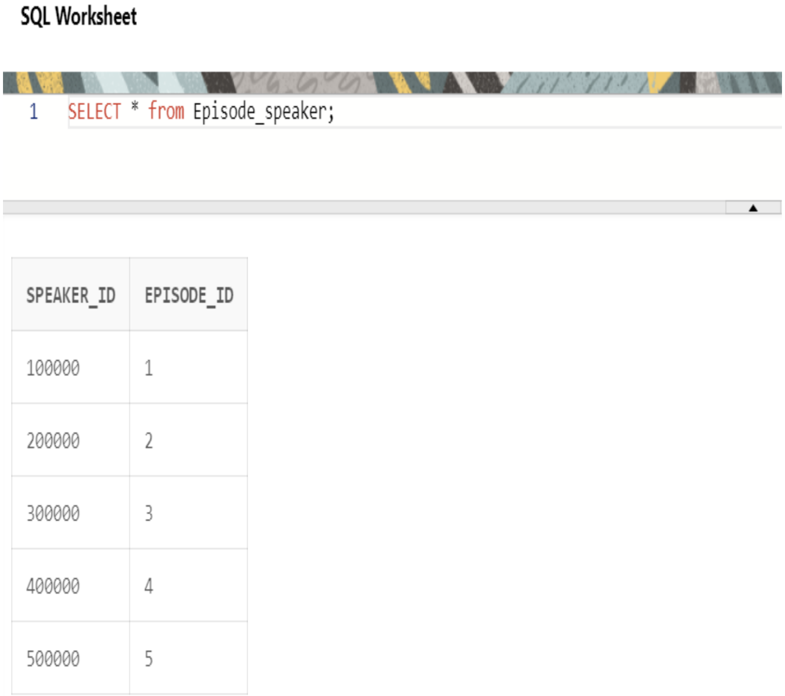
EPISODE_ID	CHANNEL_ID	EPISODE_NAME	EPISODE_NO	SEASON	EDITOR	DOWNLOAD_DATE
1	999991	BigData	1	1	Jaad E	04-JAN-22
2	999991	datawarehouse	2	1	Maha E	07-MAR-22
3	333331	BusinessWars	1	1	Mona A	08-SEP-19
4	333331	StartUp	1	2	Dalal S	04-MAR-22
5	111111	KSA economy	1	2	Maha D	04-JAN-23



SQL Worksheet

```sql
SELECT * from SPEAKER;
```

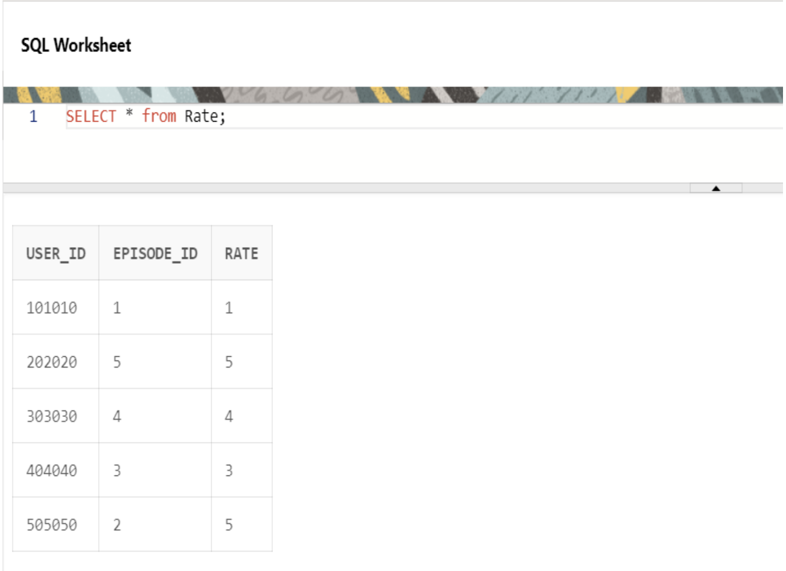
SPEAKER_ID	EPISODE_ID	FNAME	LNAME	PHONENO
100000	1	Wassem	Majid	05508795
200000	2	Suha	Sami	0638866256
300000	3	Wassem	Majid	05508795
400000	4	Amani	Ahmad	05508795
500000	5	Sultan	nano	05508795



SQL Worksheet

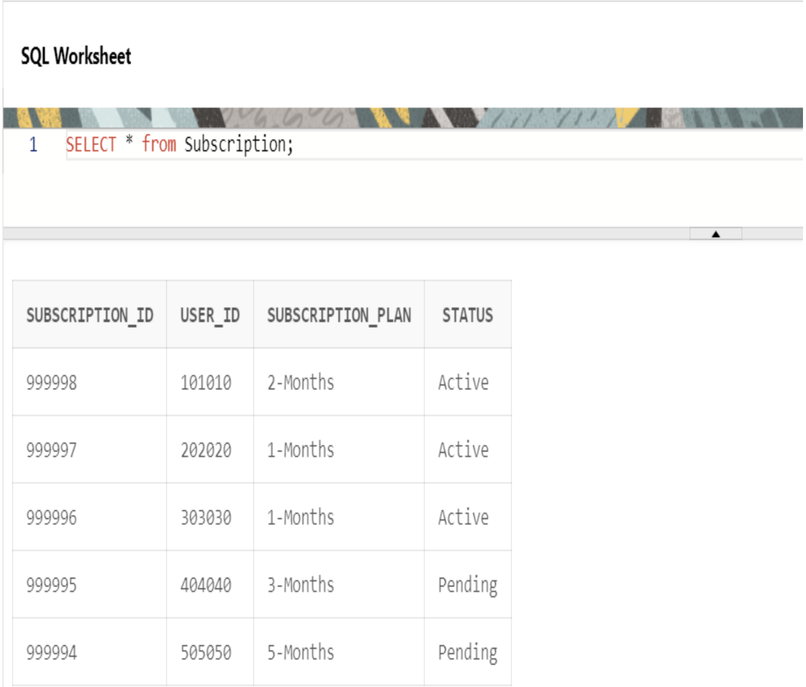
```sql
SELECT * from Episode_speaker;
```

SPEAKER_ID	EPISODE_ID
100000	1
200000	2
300000	3
400000	4
500000	5



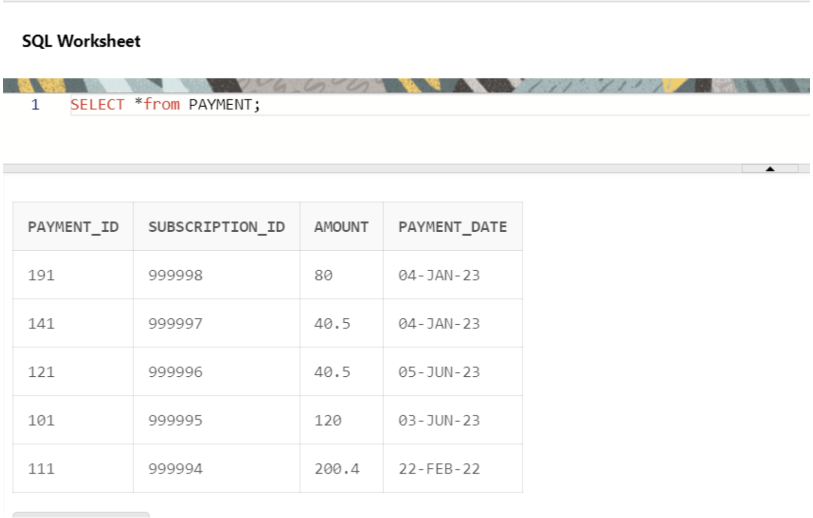
```sql
SELECT * from Rate;
```

USER_ID	EPISODE_ID	RATE
101010	1	1
202020	5	5
303030	4	4
404040	3	3
505050	2	5



```sql
SELECT * from Subscription;
```

SUBSCRIPTION_ID	USER_ID	SUBSCRIPTION_PLAN	STATUS
999998	101010	2-Months	Active
999997	202020	1-Months	Active
999996	303030	1-Months	Active
999995	404040	3-Months	Pending
999994	505050	5-Months	Pending

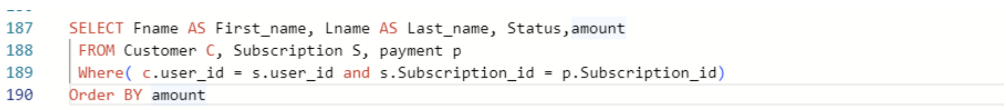


```sql
SELECT *from PAYMENT;
```

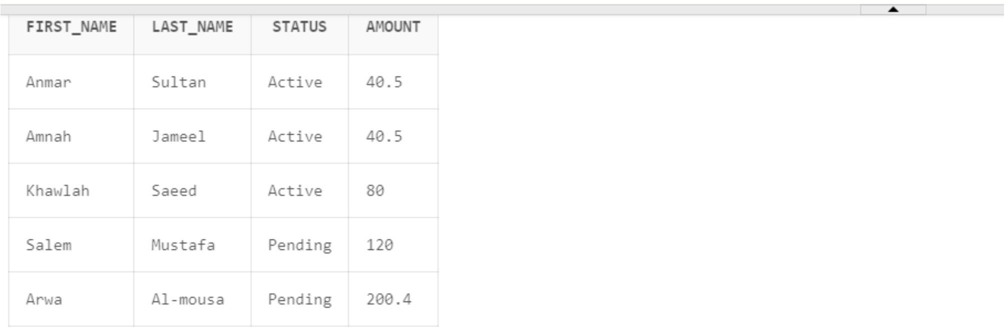
PAYMENT_ID	SUBSCRIPTION_ID	AMOUNT	PAYMENT_DATE
191	999998	80	04-JAN-23
141	999997	40.5	04-JAN-23
121	999996	40.5	05-JUN-23
101	999995	120	03-JUN-23
111	999994	200.4	22-FEB-22

3. Queries:

- Where and order by to display first name, last name, status, and amount from three different tables and the order done based on the amount

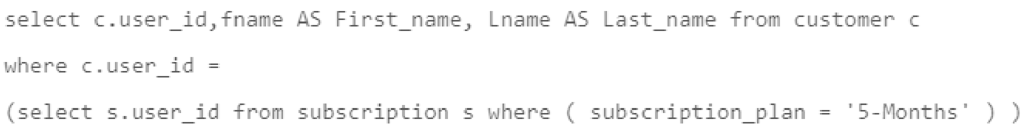


```
SELECT Fname AS First_name, Lname AS Last_name, Status,amount
 FROM Customer C, Subscription S, payment p
 Where( c.user_id = s.user_id and s.Subscription_id = p.Subscription_id)
Order BY amount
```

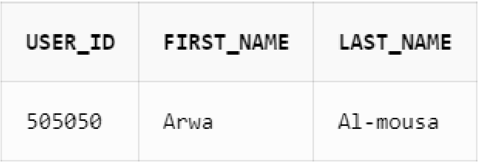


FIRST_NAME	LAST_NAME	STATUS	AMOUNT
Anmar	Sultan	Active	40.5
Amnah	Jameel	Active	40.5
Khawlah	Saeed	Active	80
Salem	Mustafa	Pending	120
Arwa	Al-mousa	Pending	200.4

- Subquery to find the user with subscription plan of 5 months

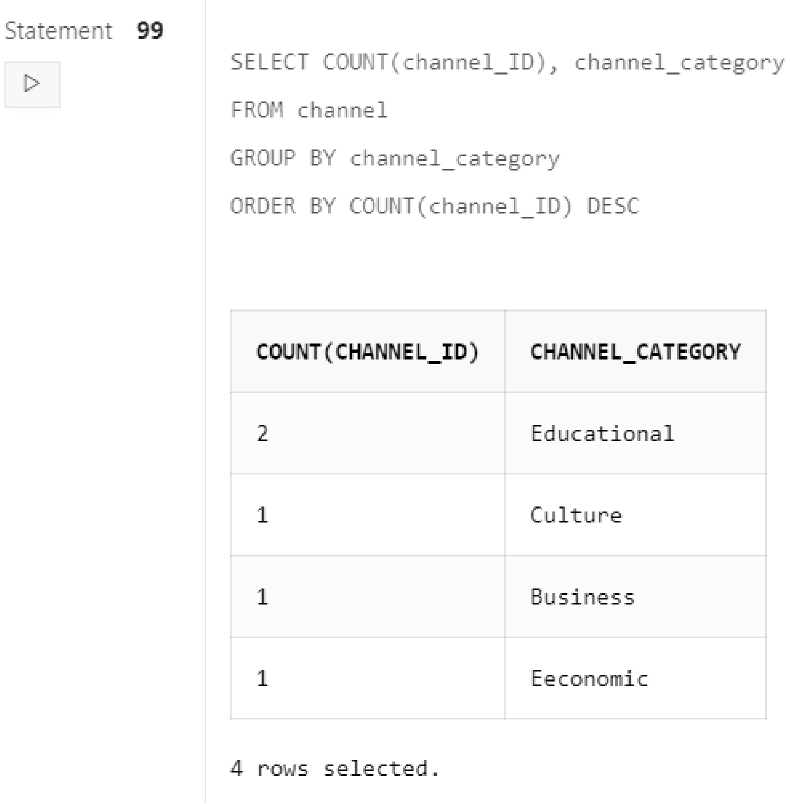


```
select c.user_id,fname AS First_name, Lname AS Last_name from customer c

where c.user_id =

(select s.user_id from subscription s where ( subscription_plan = '5-Months' ) )
```



USER_ID	FIRST_NAME	LAST_NAME
505050	Arwa	Al-mousa

- using group by to count each channel category.



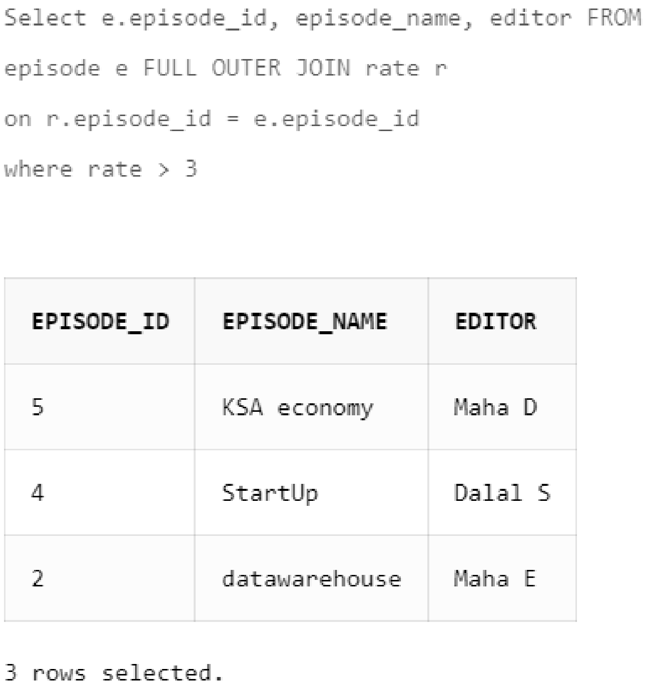
Statement 99

```
SELECT COUNT(channel_ID), channel_category
FROM channel
GROUP BY channel_category
ORDER BY COUNT(channel_ID) DESC
```

COUNT(CHANNEL_ID)	CHANNEL_CATEGORY
2	Educational
1	Culture
1	Business
1	Eeconomic

4 rows selected.

- Join between two tables (rate and episode) for rate > 3

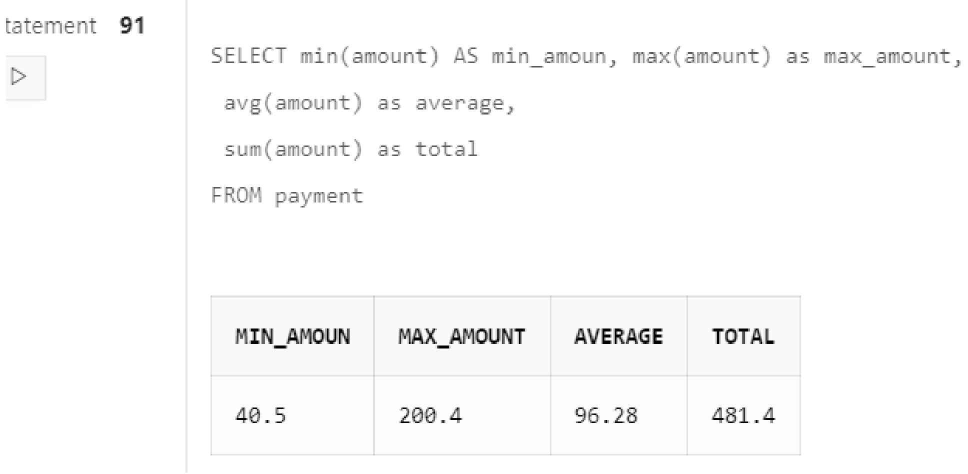


```
Select e.episode_id, episode_name, editor FROM
episode e FULL OUTER JOIN rate r
on r.episode_id = e.episode_id
where rate > 3
```

EPISODE_ID	EPISODE_NAME	EDITOR
5	KSA economy	Maha D
4	StartUp	Dalal S
2	datawarehouse	Maha E

3 rows selected.

- Aggregate functions to calculate the minimum, maximum, average and sum of the amount



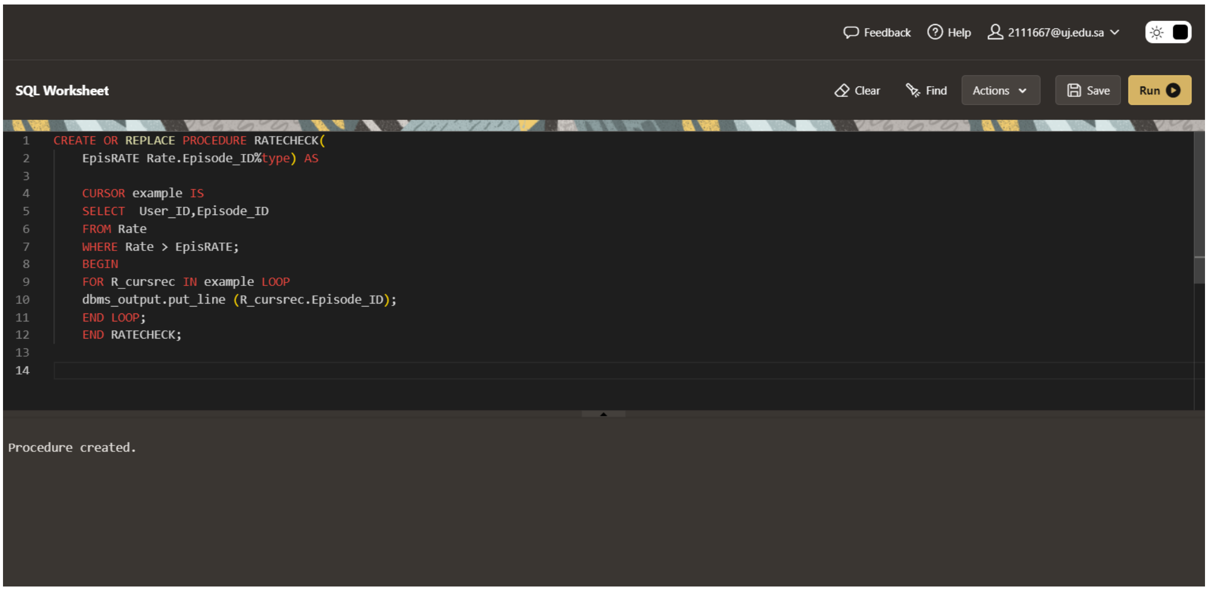
```
SELECT min(amount) AS min_amoun, max(amount) as max_amount,
 avg(amount) as average,
 sum(amount) as total
FROM payment
```

MIN_AMOUN	MAX_AMOUNT	AVERAGE	TOTAL
40.5	200.4	96.28	481.4

4. Procedures:

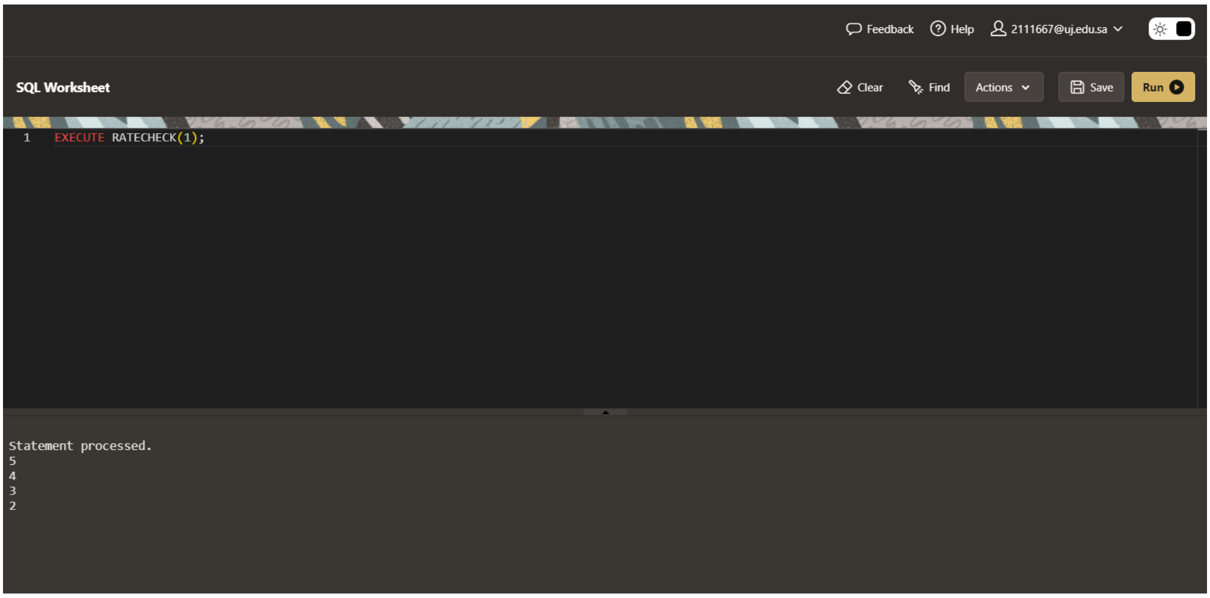
Parameter based Select query stored procedure which return records:

- this procedure going to take a rate from the user then display all the episodes that have a higher rate.



```
CREATE OR REPLACE PROCEDURE RATECHECK(
    EpisRATE Rate.Episode_ID%type) AS

    CURSOR example IS
    SELECT  User_ID,Episode_ID
    FROM Rate
    WHERE Rate > EpisRATE;
    BEGIN
    FOR R_cursrec IN example LOOP
    dbms_output.put_line (R_cursrec.Episode_ID);
    END LOOP;
    END RATECHECK;
```

```
Procedure created.
```



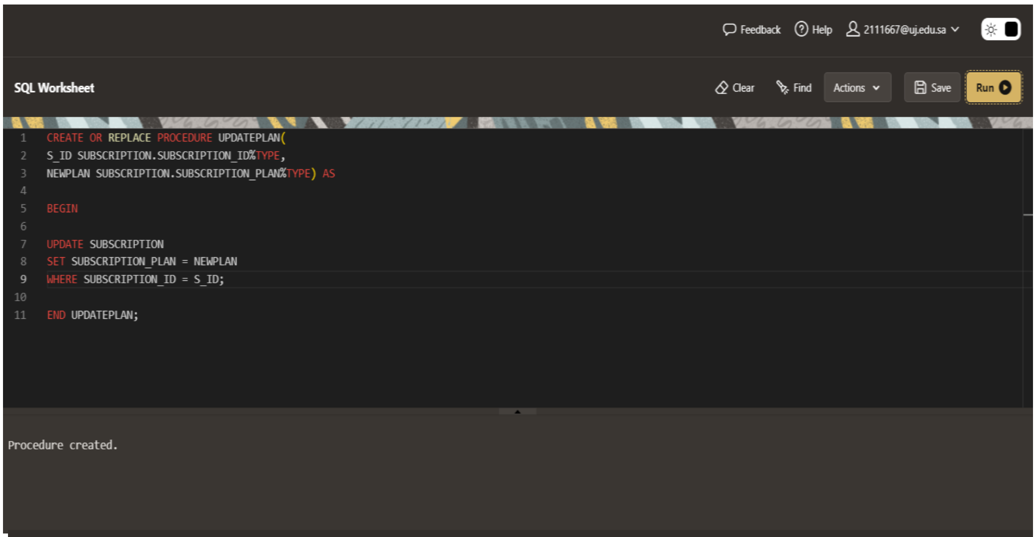
```
EXECUTE RATECHECK(1);
```

```
Statement processed.
5
4
3
2
```

Note: the rate check procedure returns the episode_id, the episode_id contains zeros at the beginning and the oracle server ignores them.

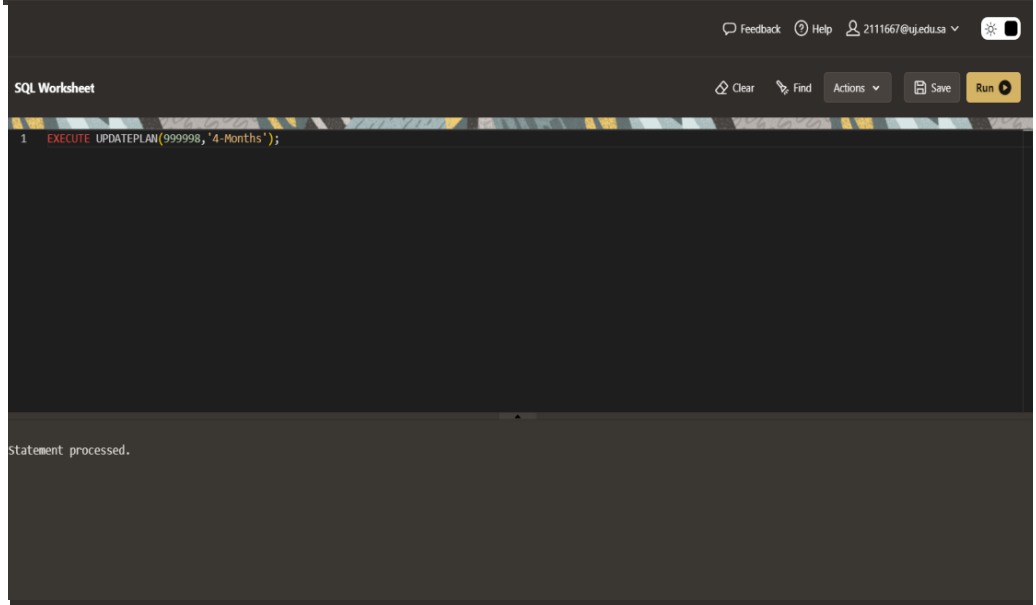
- Update query based stored procedure:

This procedure will take SUBSCRIPTION ID and SUBSCRIPTION PLAN from user then update his plan



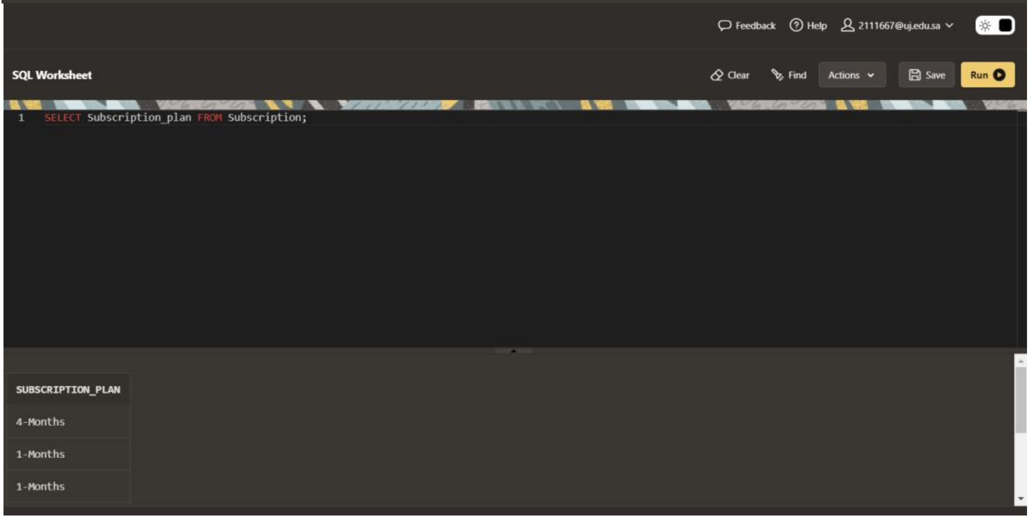
```sql
CREATE OR REPLACE PROCEDURE UPDATEPLAN(
S_ID SUBSCRIPTION.SUBSCRIPTION_ID%TYPE,
NEWPLAN SUBSCRIPTION.SUBSCRIPTION_PLAN%TYPE) AS

BEGIN

UPDATE SUBSCRIPTION
SET SUBSCRIPTION_PLAN = NEWPLAN
WHERE SUBSCRIPTION_ID = S_ID;

END UPDATEPLAN;
```

```
Procedure created.
```



```sql
EXECUTE UPDATEPLAN(999998,'4-Months');
```

```
Statement processed.
```



```sql
SELECT Subscription_plan FROM Subscription;
```

SUBSCRIPTION_PLAN
4-Months
1-Months
1-Months