

CSE 202, Winter Semester 2021

Database Management System

Mid-Sem Exam

Team No. 97

Dhroov Goel (2019303)

Sakshat Mali (2019327)

Shubham Garg (2019336)

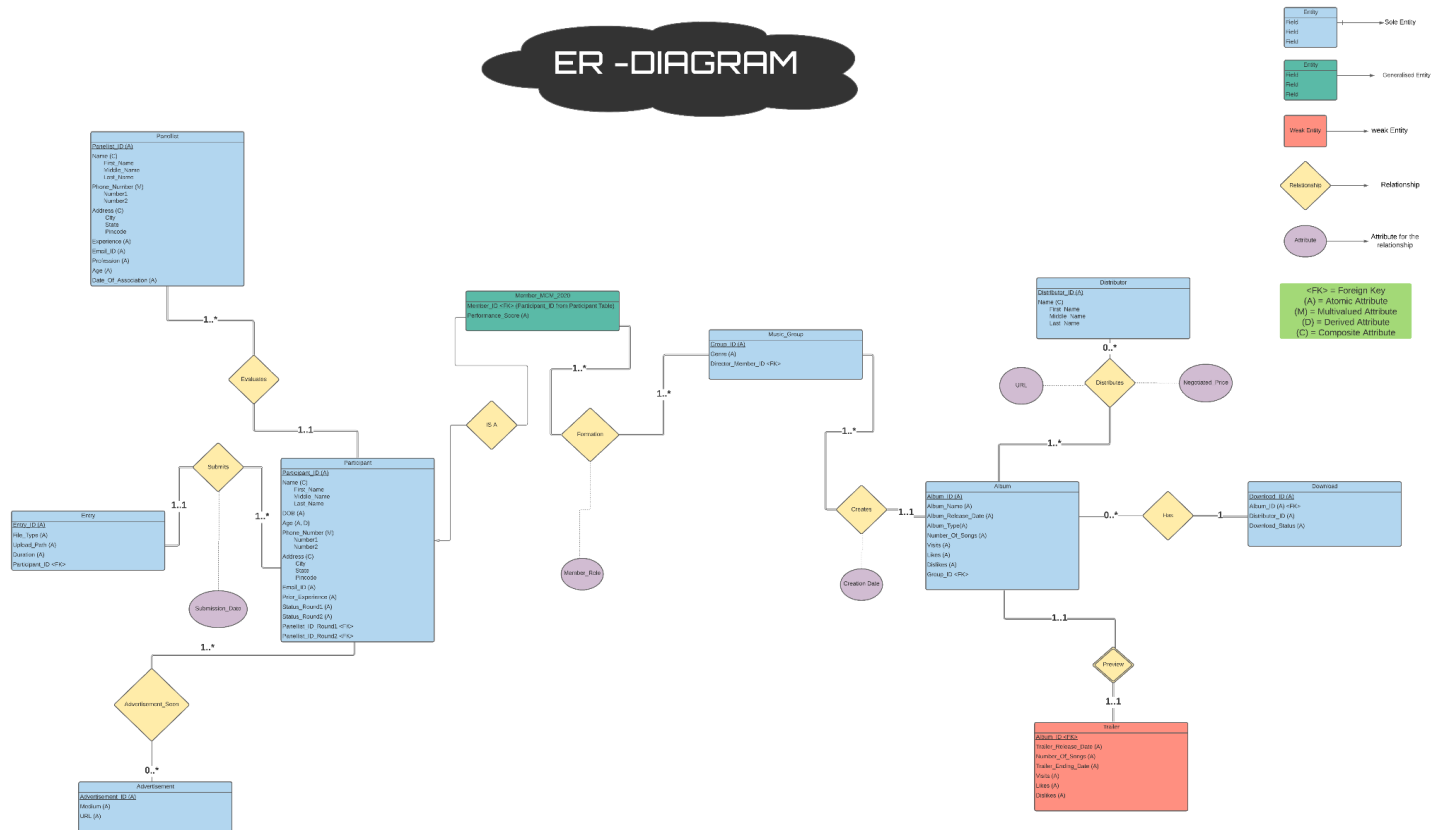
INDEX

| <u>Answer#</u> | <u>Page#</u> |
|-----------------------|---------------------|
| 1 | 3 |
| 2 | 4-6 |
| 3 | 7-8 |
| 4A | 9 |
| 4B | 9 |
| 4C | 9 |
| 4D | 9 |
| 4E | 9 |
| 5A | 10 |
| 5B | 10 |
| 5C | 10 |
| 5D | 11 |
| 5E | 11 |
| 5F | 11-17 |
| 6A | 18 |

| | |
|-----------|-------|
| 6B | 18-19 |
| 6C | 19 |
| 7A | 20-21 |
| 7B | 22-30 |
| 7C | 31 |

ANS 1:-

ER-DIAGRAM



Link to ER-Diagram:-

<https://lucid.app/lucidchart/invitations/accept/a088f1c1-8c1d-450b-bfad-68d34d3fa02b>

ANS 2:-

🔑 -> Primary Key

◆ -> Not Null

◇ -> Can be Null

Primary Key: 🔑 **Participant_ID**

Candidate Key: **Phone_Number**
Participant_ID

Primary Key: 🔑 **Album_ID(Album)**

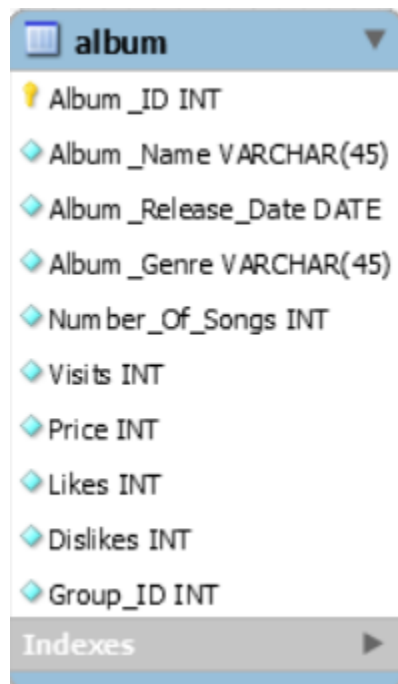
Candidate Key: **Album_ID**

| participant |
|---------------------------------|
| 🔑 Participant_ID INT |
| ◆ First_Name VARCHAR(45) |
| ◇ Middle_Name VARCHAR(45) |
| ◆ Last_Name VARCHAR(45) |
| ◆ DOB DATE |
| ◆ Age INT |
| ◆ Address VARCHAR(100) |
| ◆ Phone_Number DECIMAL (10) |
| ◇ Email_ID VARCHAR(45) |
| ◆ City VARCHAR(100) |
| ◆ State VARCHAR(45) |
| ◆ Pincode DECIMAL (6) |
| ◇ Prior_Experience VARCHAR(200) |
| ◆ Status_Round_1 VARCHAR(45) |
| ◇ Status_Round_2 VARCHAR(45) |
| ◆ Panellist_ID_Round1 INT |
| ◇ Panellist_ID_Round2 INT |
| Indexes ▶ |

| trailer |
|-----------------------------|
| ◆ Trailer_Release_Date DATE |
| ◆ Trailer_Ending_Date DATE |
| ◆ Visits INT |
| ◆ Likes INT |
| ◆ Dislikes INT |
| ◆ Album_ID INT |
| ◆ Number_Of_Songs INT |

Primary Key: 🗝️ **Album_ID**

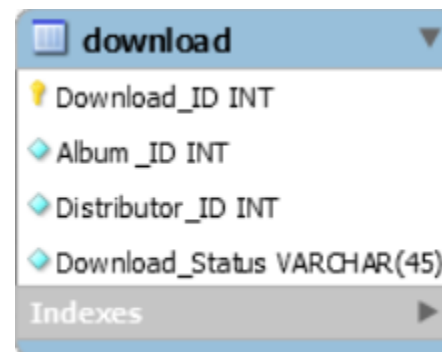
Candidate Key: **Phone_Number**
Album_ID



| album | |
|--------------------|-------------|
| 🗝️ Album_ID | INT |
| Album_Name | VARCHAR(45) |
| Album_Release_Date | DATE |
| Album_Genre | VARCHAR(45) |
| Number_Of_Songs | INT |
| Visits | INT |
| Price | INT |
| Likes | INT |
| Dislikes | INT |
| Group_ID | INT |
| Indexes ▶ | |

Primary Key: 🗝️ **Download_ID**

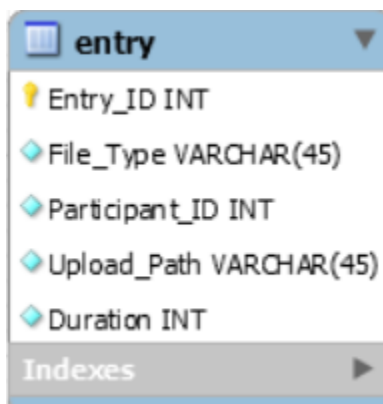
Candidate Key: **Album_ID**
Download_ID



| download | |
|-----------------|-------------|
| 🗝️ Download_ID | INT |
| Album_ID | INT |
| Distributor_ID | INT |
| Download_Status | VARCHAR(45) |
| Indexes ▶ | |

Primary Key: 🗝️ **Entry_ID**

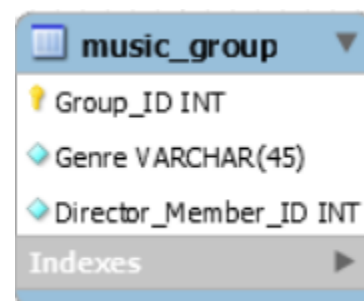
Candidate Key: **Participant_ID**
Entry_ID



| entry | |
|----------------|-------------|
| 🗝️ Entry_ID | INT |
| File_Type | VARCHAR(45) |
| Participant_ID | INT |
| Upload_Path | VARCHAR(45) |
| Duration | INT |
| Indexes ▶ | |

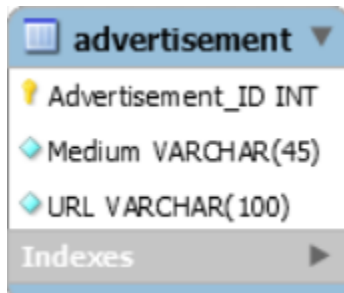
Primary Key: 🗝️ **Group_ID**




Candidate Key: **Group_ID**




| music_group | |
|--------------------|-------------|
| 🗝️ Group_ID | INT |
| Genre | VARCHAR(45) |
| Director_Member_ID | INT |
| Indexes ▶ | |









Primary Key:  **Advertisement_ID**
Candidate Key: **Advertisement_ID**




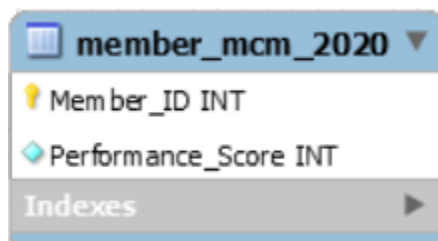
| advertisement | |
|---|----------------------|
|  | Advertisement_ID INT |
|  | Medium VARCHAR(45) |
|  | URL VARCHAR(100) |
| Indexes | |



Primary Key:  **Panellist_ID**
Candidate Key: **Phone_Number**
Panellist_ID




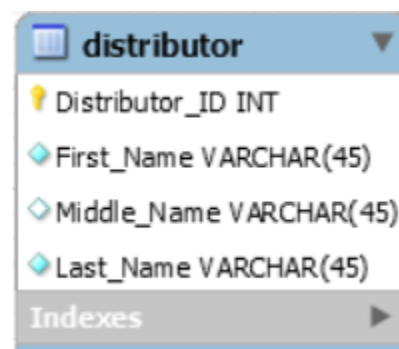
| panellist | |
|---|--------------------------|
|  | Panellist_ID INT |
|  | First_Name VARCHAR(45) |
|  | Middle_Name VARCHAR(45) |
|  | Last_Name VARCHAR(45) |
|  | Address VARCHAR(45) |
|  | City VARCHAR(45) |
|  | State VARCHAR(100) |
|  | Pincode DECIMAL(6) |
|  | Phone_Number VARCHAR(45) |
|  | Email_ID VARCHAR(100) |
|  | Profession VARCHAR(45) |
|  | Experience VARCHAR(200) |
|  | Date_Of_Association DATE |
|  | Age INT |
| Indexes | |





Primary Key:  **Member_ID**
Candidate Key: **Member_ID**



| member_mcm_2020 | |
|---|-----------------------|
|  | Member_ID INT |
|  | Performance_Score INT |
| Indexes | |

Primary Key:  **Distributor_ID**
Candidate Key: **Distributor_ID**



| distributor | |
|---|-------------------------|
|  | Distributor_ID INT |
|  | First_Name VARCHAR(45) |
|  | Middle_Name VARCHAR(45) |
|  | Last_Name VARCHAR(45) |
| Indexes | |

ANS 3:-

Yes, we were able to make a good-design of the McM Sangeet Company database.

We were able to do so as we followed almost all the steps required for making a good database design.

We started by dividing the gathered information into the major entities (real-world objects) such as Participant, Entry, Advertisement, Member_Mcm_2020, ...etc. Then we carefully thought about all the attributes of the entities from the gathered information. Then we looked for any primary key into our entity and if not then we made a **Entity#** as the key wherever required.

Then we refined our design by removing all the errors by answering all the queries that can be formed out of the given situation. Then, we applied all the normalisation rules to see if there are any redundancies and repetition in our design.

Firstly, we broke down all the multivalued attributes like phone_no, from our tables and converted them into atomic attributes, to remove data redundancy and repetition from our tables.

Then for the second normalization, we tried to remove the partial dependencies from our relation. For this, we made all the non-prime attributes to be fully functional dependent on the **Candidate Key** for each entity table.

Then for the third normal form, we tried to remove all the transitive dependencies from the entity tables i.e. we tried to form a design such that no non-key column is dependent on each other. For this we mapped all the non-key with the primary key, and made new tables if a certain non-key was dependent on any other non-key to reduce redundancy and repetition.

Then for the next normal form i.e. **BCNF**, we tried to make 'X' as the super key in the function dependency $X \rightarrow Y$ for each entity table, to avoid the redundancies caused by the attributes of the table.

Using these normalization techniques we were able to come with a design that we think as an appropriate design.

ANS 4:-

A.)

$$\sigma_{\text{Album_Type} = \text{'Audio'} \wedge \text{Album_Release_Date} \geq \text{'2020-01-01'} \wedge \text{Album_Release_Date} \leq \text{'2020-12-31'}} (\text{Album})$$

B.)

$$\begin{aligned} & \text{Participant} \bowtie (\Pi_{\text{Participant_ID}} (\sigma_{\text{File_Type} = \text{'Audio'}} ((\text{Participant}) \bowtie (\text{Entry}))) \\ & \cap \Pi_{\text{Participant_ID}} (\sigma_{\text{File_Type} = \text{'Video'}} ((\text{Participant}) \bowtie (\text{Entry})))) \end{aligned}$$

C.)

$$\begin{aligned} & \rho_{T_1} ((\text{Formation}) \bowtie (\text{Member_MCM_2020})) \\ & \rho_{T_2} ((\text{Formation}) \bowtie (\text{Member_MCM_2020})) \\ & \text{Member_MCM_2020} \bowtie (\Pi_{T_1.\text{Member_ID}} (\sigma_{T_1.\text{Member_ID} = T_2.\text{Member_ID} \cap T_1.\text{Group_ID} \\ & \neq T_2.\text{Group_ID}} (T_1 \bowtie T_2))) \end{aligned}$$

D.)

$$\begin{aligned} & \text{Member_MCM_2020} \bowtie (\Pi_{\text{Member_ID}} (\sigma_{\text{Genre} = \text{'Pop'}} ((\text{Music_Group}) \bowtie (\text{Formation}) \\ & \bowtie (\text{Member_MCM2020}))) - \Pi_{\text{Member_ID}} (\sigma_{\text{Genre} \neq \text{'Pop'}} ((\text{Music_Group}) \\ & \bowtie (\text{Formation}) \bowtie (\text{Member_MCM2020})))) \end{aligned}$$

E.)

$$\begin{aligned} & \text{Distributor} \bowtie \\ & (\Pi_{\text{Distributor_ID}} (\sigma_{\text{Album_Type} = \text{'Audio'}} ((\text{Distributes}) \bowtie (\text{Album}) \bowtie (\text{Distributor}))) \cap \\ & \Pi_{\text{Distributor_ID}} (\sigma_{\text{Album_Type} = \text{'Video'}} ((\text{Distributes}) \bowtie (\text{Album}) \bowtie (\text{Distributor})))) \end{aligned}$$

ANS 5:-

A.)

```
SELECT * FROM Album WHERE Album_Type = 'Audio' and  
year(Album_Release_Date)=2020;
```

B.)

```
Select distinct P.Participant_id , P.First_Name From Participant  
as P,(Select M.Member_ID From Member_MCM_2020 as  
M,(Select F1.Member_ID from Formation as F1, Formation as F2  
where F1.Member_ID = F2.Member_ID and F1.Group_ID <>  
F2.Group_ID) as A where M.Member_ID = A.Member_ID)as B  
where P.Participant_ID = B.Member_ID;
```

C.)

```
Select distinct P.Participant_id, P.first_Name From Participant as  
P ,(Select Y.Member_ID From Formation as F2, Music_Group as  
G2 , (Select X.Member_ID, X.Count From (Select F.Member_ID ,  
Count(*) as Count From Music_Group as G, Formation as F  
where G.Group_ID= F.Group_ID Group by F.Member_ID order  
by count asc) as X where X.Count = 1) as Y where Y.Member_ID =  
F2.Member_ID and F2.Group_ID = G2.Group_ID and  
G2.Genre='Pop') as Z where P.Participant_ID = Z.Member_ID;
```

D.)

```
Select P.Participant_id , P.First_Name From Participant AS P,  
(Select Distinct E1.Participant_id FROM Entry as E1,Entry AS E2  
WHERE E1.Participant_id=E2.Participant_id AND E1.File_Type  
<> E2.File_Type )AS A WHERE P.Participant_id =  
A.Participant_id;
```

E.)

```
Select T2.Advertisement_ID, Count, T2.URL, T2.Medium From  
(Select * From (Select A.Advertisement_ID , Count(*) as Count  
From Advertisement_Seen as  
A Group by A.Advertisement_ID ORDER BY Count DESC) as B  
LIMIT 1) as T1, Advertisement as T2 where T2.Advertisement_ID  
= T1.Advertisement_ID;
```

F.)

- **Distributors associated with Max downloads of Audio Album:**

```
select distinct Di.Distributor_ID, Di.First_Name FROM Distributor as  
Di,(Select Ds.Distributor_ID FROM distributes As Ds,(select * from  
(Select A.album_ID, B.Count FROM album AS A, (Select  
D1.Album_ID , Count(*) As count From (Select * FROM Download  
Where Download_Status = 'Pass') AS D1 group by D1.album_ID  
order by count desc) AS B WHERE A.Album_ID = B.Album_ID AND  
A.Album_Type = 'Audio' and year(A.Album_Release_Date)=2020)  
AS Aud_Table limit 1 ) AS Aud_Lim WHERE Aud_lim.Album_ID =  
Ds.Album_ID) AS Aud_Dis_ID WHERE Aud_Dis_ID.Distributor_ID  
= Di.Distributor_ID
```

Python program for embedded sql:-

```

import mysql.connector

mydb = mysql.connector.connect(
    host = "localhost",
    user = "root",
    passwd = "Butterss@123",
    database = "Sangeeta_Company"
)

mycursor = mydb.cursor()

mycursor.execute(

"select distinct Di.Distributor_ID, Di.First_Name FROM
Distributor as Di, (Select Ds.Distributor_ID FROM distributes As
Ds, (select * from (Select A.album_ID, B.Count FROM album AS A,
(Select D1.Album_ID , Count(*) As count From (Select * FROM
Download Where Download_Status = 'Pass') AS D1  group by
D1.album_ID order by count desc) AS B WHERE A.Album_ID =
B.Album_ID AND A.Album_Type = 'Audio' and
year(A.Album_Release_Date)=2020) AS Aud_Table limit 1 ) AS
Aud_Lim WHERE Aud_lim.Album_ID = Ds.Album_ID) AS Aud_Dis_ID
WHERE Aud_Dis_ID.Distributor_ID = Di.Distributor_ID"

)

myresult = mycursor.fetchall()

for row in myresult:
    print(row)

```

Screenshot of the output:-

```

PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM> c++; cd 'c:\Users\Dhroov\Desktop\DBMS_MID_SEM'; & 'C:\Users\Dhroov\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\python.exe' 'c:\Users\Dhroov\.vscode\extensions\ms-python.python-2021.2.625869727\python\files\lib\python\debugpy\launcher' '62146' '--' 'c:\Users\Dhroov\Desktop\DBMS_MID_SEM\embedded_sql.py'
(2, 'Sutherland')
(3, 'cello')
(4, 'Siouxie')
(5, 'Ado')
PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM>

```

- **Distributors associated with Max downloads of Video Album:**

select distinct Di.Distributor_ID, Di.First_Name FROM Distributor as Di,(Select Ds.Distributor_ID FROM distributes As Ds,(select * from (Select A.album_ID, B.Count FROM album AS A, (Select D1.Album_ID , Count(*) As count From (Select * FROM Download Where Download_Status = 'Pass') AS D1 group by D1.album_ID order by count desc) AS B WHERE A.Album_ID = B.Album_ID AND A.Album_Type = 'Video' and year(A.Album_Release_Date)=2020) AS vid_Table limit 1) AS vid_Lim WHERE vid_Lim.Album_ID = Ds.Album_ID) AS vid_Dis_ID WHERE vid_Dis_ID.Distributor_ID = Di.Distributor_ID

Python program for embedded sql:-

```

import mysql.connector

mydb = mysql.connector.connect(
    host = "localhost",
    user = "root",
    passwd = "Butterss@123",
    database = "Sangeeta_Company"
)

mycursor = mydb.cursor()

mycursor.execute(

```

```

"select distinct Di.Distributor_ID, Di.First_Name FROM Distributor as
Di,(Select Ds.Distributor_ID FROM distributes As Ds,(select * from (Select
A.album_ID, B.Count FROM album AS A, (Select D1.Album_ID , Count(*) As
count From (Select * FROM Download Where Download_Status = 'Pass') AS D1
group by D1.album_ID order by count desc) AS B WHERE A.Album_ID =
B.Album_ID AND A.Album_Type = 'Video' and year(A.Album_Release_Date)=2020)
AS vid_Table limit 1 ) AS vid_Lim WHERE vid_lim.Album_ID = Ds.Album_ID) AS
vid_Dis_ID WHERE vid_Dis_ID.Distributor_ID = Di.Distributor_ID"

)

myresult = mycursor.fetchall()

for row in myresult:
    print(row)

```

Screenshot of the output:-

```

PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM> c:: cd 'c:\Users\Dhroov\Desktop\DBMS_MID_SEM'; & 'C:\Users\Dhroov\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Pyt
hon.3.7_qbz5n2kfra8p0\python.exe' 'c:\Users\Dhroov\.vscode\extensions\ms-python.python-2021.2.625869727\pythonFiles\lib\python\debugpy\launcher' '62159' '---' 'c:\Users\Dhro
ov\Desktop\DBMS_MID_SEM\embedded_sql.py'
(1, 'Tabb')
(2, 'Sutherland')
(3, 'cello')
(4, 'Siouxie')
PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM>

```

- **Member associated with Max downloads of Audio Album:**

Select part.participant_id, part.first_name FROM participant as part,(Select mem.Member_ID From member_mcm_2020 as mem,(Select frm.Member_ID FROM formation AS frm,(select Mus_G.Group_ID FROM music_group As Mus_G, (select * from (Select A.album_ID,A.Group_ID, B.Count FROM album AS A, (Select D1.Album_ID , Count(*) As count From (Select * FROM Download Where Download_Status = 'Pass') AS D1 group by D1.album_ID order by count desc) AS B WHERE A.Album_ID = B.Album_ID AND A.Album_Type = 'Audio' and year(A.Album_Release_Date)=2020)

AS Aud_Table limit 1) As Aud_Lim Where Mus_G.Group_ID = Aud_Lim.Group_ID) AS Group_Max_Aud WHERE Group_Max_Aud.Group_ID = frm.Group_ID) AS Mem_Aud WHERE mem.Member_ID = Mem_Aud.Member_ID) AS Same_Mem_Aud where Same_Mem_Aud.Member_id = part.Participant_id

Python program for embedded sql:-

```
import mysql.connector

mydb = mysql.connector.connect(
    host = "localhost",
    user = "root",
    passwd = "Butterss@123",
    database = "Sangeeta_Company"
)

mycursor = mydb.cursor()

mycursor.execute(

"Select part.participant_id, part.participant_name FROM participant as
part, (Select mem.Member_ID From member_mcm_2020 as mem, (Select
frm.Member_ID FROM formation AS frm, (select Mus_G.Group_ID FROM
music_group As Mus_G, (select * from (Select A.album_ID,A.Group_ID,
B.Count FROM album AS A, (Select D1.Album_ID , Count(*) As count From
(Select * FROM Download Where Download_Status = 'Pass') AS D1 group by
D1.album_ID order by count desc) AS B WHERE A.Album_ID = B.Album_ID AND
A.Album_Type = 'Audio' and year(A.Album_Release_Date)=2020) AS Aud_Table
limit 1 ) As Aud_Lim Where Mus_G.Group_ID = Aud_Lim.Group_ID) AS
Group_Max_Aud WHERE Group_Max_Aud.Group_ID = frm.Group_ID ) AS Mem_Aud
WHERE mem.Member_ID = Mem_Aud.Member_ID) AS Same_Mem_Aud where
Same_Mem_Aud.Member_id = part.Participant_id"

)

myresult = mycursor.fetchall()
```

```
for row in myresult:
    print(row)
```

Screenshot of the output:-

```
PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM> cd 'c:\Users\Dhroov\Desktop\DBMS_MID_SEM'; & 'C:\Users\Dhroov\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.7.9bz5n2kfra8p0\python.exe' 'C:\Users\Dhroov\.vscode\extensions\ms-python.python-2021.2.625869727\pythonFiles\lib\python\debugpy\launcher' '62169' '--' 'c:\Users\Dhroov\Desktop\DBMS_MID_SEM\embedded_sql.py'
(2, 'Heinrick')
(4, 'Amii')
(9, 'Cindie')
PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM>
```

- **Member associated with Max downloads of Video Album:**

Select part.participant_id, part.first_name FROM participant as part,(Select mem.Member_ID From member_mcm_2020 as mem,(Select frm.Member_ID FROM formation AS frm,(select Mus_G.Group_ID FROM music_group As Mus_G, (select * from (Select A.album_ID,A.Group_ID, B.Count FROM album AS A, (Select D1.Album_ID , Count(*) As count From (Select * FROM Download Where Download_Status = 'Pass') AS D1 group by D1.album_ID order by count desc) AS B WHERE A.Album_ID = B.Album_ID AND A.Album_Type = 'Video' and year(A.Album_Release_Date)=2020) AS vid_Table limit 1) As vid_Lim Where Mus_G.Group_ID = vid_Lim.Group_ID) AS Group_Max_vid WHERE Group_Max_vid.Group_ID = frm.Group_ID) AS Mem_vid WHERE mem.Member_ID = Mem_vid.Member_ID) AS Same_Mem_vid where Same_Mem_vid.Member_id = part.Participant_id

Python program for embedded sql:-

```
import mysql.connector

mydb = mysql.connector.connect(
```



```

    host = "localhost",
    user = "root",
    passwd = "Butterss@123",
    database = "Sangeeta_Company"
)

mycursor = mydb.cursor()

mycursor.execute(
    "Select part.participant_id, part.participant_name FROM participant as
    part, (Select mem.Member_ID From member_mcm_2020 as mem, (Select
    frm.Member_ID FROM formation AS frm, (select Mus_G.Group_ID FROM
    music_group As Mus_G, (select * from (Select A.album_ID,A.Group_ID,
    B.Count FROM album AS A, (Select D1.Album_ID , Count(*) As count From
    (Select * FROM Download Where Download_Status = 'Pass') AS D1 group by
    D1.album_ID order by count desc) AS B WHERE A.Album_ID = B.Album_ID AND
    A.Album_Type = 'Video' and year(A.Album_Release_Date)=2020) AS vid_Table
    limit 1 ) As vid_Lim Where Mus_G.Group_ID = vid_Lim.Group_ID) AS
    Group_Max_vid WHERE Group_Max_vid .Group_ID = frm.Group_ID ) AS Mem_vid
    WHERE mem.Member_ID = Mem_vid .Member_ID) AS Same_Mem_vid where
    Same_Mem_vid .Member_id = part.Participant_id"
)

myresult = mycursor.fetchall()

for row in myresult:
    print(row)

```

Screenshot of the output:-

```

PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM> c::; cd 'c:\Users\Dhroov\Desktop\DBMS_MID_SEM'; & 'C:\Users\Dhroov\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Py
thon.3.7_qbz5n2kfra8p0\python.exe' 'c:\Users\Dhroov\.vscode\extensions\ms-python.python-2021.2.625869727\pythonFiles\lib\python\debugpy\launcher' '62178' '--' 'c:\Users\Dhro
ov\Desktop\DBMS_MID_SEM\embedded_sql.py'
(4, 'Amii')
(6, 'Chad')
(9, 'Cindie')
PS C:\Users\Dhroov\Desktop\DBMS_MID_SEM>

```

ANS 6:-

A.)

We can model this data by visualising this as a cube where the three axis of the cube would represent the following as:

1. State
2. Type of Album
3. Age Group

All these attributes will belong to the participant who is taking part in the competition and participant will be our aggregated quantity.

As there is no attribute named Age_Group in our table, so we need to create a column for Age_Group with the help of case statement of MySQL. As we will attain the new table, (say Model_Table) having the following entities namely - State, Age_Group and ID; then we need to take its cartesian product with Album and Participant table with the condition that Participant_Id = Model_Table_ID and Participant_ID = Album_ID

B.)

Select age_group , state , File_Type,count(*) as countt from
(Select age_group , state , ent.file_type from
(select

 participant_ID, First_Name, State,
 case when

 Age >= 15 and Age <= 20 then "Group_1"

 when Age >= 21 and Age <= 25 then "Group_2"

 when Age >= 26 and Age <= 30 then "Group_3"

 when Age >= 31 and Age <= 40 then "Group_4"

```

    when Age >= 40 and Age <= 100 then "Group_5"
    else "No Entry Allowed" end as Age_Group
from
    Participant) as Temp , entry as ent where
ent.participant_id=temp.participant_id) as Model_Table
    group by age_group , state , File_Type with rollup;

```

C.)

```

Select zone , age_group , state , File_Type,count(*) as countt from
(Select zone , age_group , state , ent.file_type from
(select
    participant_ID, First_Name, Zone, State,
    case when
    Age >= 15 and Age <= 20 then "Group_1"
    when Age >= 21 and Age <= 25 then "Group_2"
    when Age >= 26 and Age <= 30 then "Group_3"
    when Age >= 31 and Age <= 40 then "Group_4"
    when Age >= 40 and Age <= 100 then "Group_5"
    else "No Entry Allowed" end as Age_Group
from
    Participant) as Temp , entry as ent where
ent.participant_id=temp.participant_id) as Zone_Table
    group by zone , age_group , state , File_Type with rollup;

```

Ans7:-

A.)

Participant Relation:-

- **Insertion Anomaly:**

In the Participant relation, a participant could have multiple entry submissions of media files for which we would need to create multiple tuples to store the File_upload path for each file being uploaded but this would lead to row level redundancy as the mentioned primary key for the relation is Member# and Phone Number. For each entry the primary keys could repeat which should not happen. Either the file_upload_path should be included in the primary key or the relation should be decomposed. Even if we uniquely identify the tuples using the file upload path, a lot of other redundant data would still be stored.

Panelist_Album_Evaluation:-

- **Update Anomaly:**

In the given relation Panelist_Album_Evaluation there is column level duplicacy as for same panelist#, the other attributes such as panelist_Name, Experience etc would get repeated. If there is a need to update 'Experience' attribute for a certain Panelist, then this would result in updation of all those tuples where panelist# is repeated.

Member_Group_Album_Trailer:-

- **Deletion Anomaly:**

In the given relation if we try to delete a particular member from the table then this would result into deletion of all those tuples where the Member# of the member was being repeated. This could result in loss of information of various attributes such as the group id, group name, Album_name and various other attributes could get deleted from the whole database.

Album_Distribution_and_Download:-

- **Updation anomaly:**

For each download# various attributes are being repeated such as Distributor_Name, Distributor_Location etc resulting in column level duplicacy. So updation of a single attribute let's say Distributor_location would result into updation of all those tuples where Distributor# is being repeated.

B.)

First Normal Form:

A relation is in First Normal Form if all the attributes of the relation are single valued . There should be no composite or multivalued attribute in the relation. In order to reach 1NF we should convert every attribute to atomic valued attribute.

Composite:

Name = { First_Name, Middle_Name, Last_Name}

(All the attributes denoting names of Participants, Panelist, Distributors etc should be broken down into first, middle and last.

Location = { State, City, Address, pin etc}

(For attributes such as Distributor_Address)

Multivalued:

In the given problem, the participants are allowed to enter multiple phone numbers, we can store the different phone numbers in separate rows each participant (creating row level duplicacy), store using multiple columns (storing null values if there are not enough phone numbers) or creating a separate table storing participant id and phone number. (no redundancy in this case)

Relations after 1NF:

Participant (Member#, First_Name, Middle_Name, Last_Name, Age, City, Phone, Email, Prior_Experience, Advt_Seen, Album_Type, Submission Date, File Upload Path, Status_Round1, Status_Round2)

Keys: (Member# Phone File Upload Path)

Panelist_Album_Evaluation (Panelist#,
First_Name, Middle_Name, Last_Name, Experience,
Association_Month, Association_Year, File Upload Path)
Keys: (Panelist# File Upload Path)

Member_Group_Album_Trailer (Member#,
First_Name, Middle_Name, Last_Name, Group#, Group_Name,
Member-Role, Group_Music_Class, Album#, Album_Name,
Album_Type, Date_of_Creation, Album_Description,
Group_Leader#, Group_Leader_First_Name,
Group_Leader_middle_Name, Group_Leader_last_Name
Album_Approver#, Approval_Date, Trailer_Release_Date,
Trailer_Release_URL, Incoming_URL_for_View,
View_Date, Comments)
Keys: (Member# Member-Role Album#
Incoming_URL_for_View)

Album_Distribution_and_Download(Album#,
Album_Release_Date, Distributor#, Distributor_Name,
Distributor_Location, Price, Download#,
Incoming_URL_for_Download, Download_Request_Date,
Downloaded_Album#, Download_Status)
Keys: (Download#)

There is still a need to go in higher normal forms as by just decomposing composite or multivalued attributes into atomic attributes is not sufficient to overcome various row and column level duplicacy and redundant data.

Second Normal Form:

For a relation to be in second normal form , it should first satisfy the first normal form. Along with it the second normal form should not contain any partial dependencies or in other words no non-prime attribute should be dependent on any proper subset of any candidate key of the table.

Participant:

Functional Dependency:

Member# -> Fisrt_Name, Age, City, Email, Advt_Seen, Status_Round1, Status_Round2

Phone -> Member#,

File_Upload_Path -> Member#, Album_Type, Submission Date

Candidate Key: File_Upload_Path and Phone

Prime Attributes: File_Upload_Path, Phone

Partial Dependency:

Phone -> Member#, Fisrt_Name, Age, City, Email, Advt_Seen, Status_Round1, Status_Round2

File_Upload_Path -> Member#, Album_Type, Submission Date

After splitting into two tables we get:

Participant (Phone, Member#, Firstt_Name, Age, City, Email, Advt_Seen, Status_Round1, Status_Round2)

Participant_Entry (File_Url ,Member#, Album_Type, Submission Date)

Panelist_Album_Evaluation:

Functional Dependency:

Panelist# -> Panelist_Name, Experience, Association_Month, Association_Year

File_Upload_Path -> Panelist#

Candidate Key: File_Upload_Path

Panelist_Album_Evaluation (Panelist#, Panelist_Name, Experience, Association_Month, Association_Year, File Upload Path)

Member_Group_Album_Trailer:

Functional Dependency:

Group#, Trailer_Release_Date -> Album#

Album# -> Group#

Group# -> Group_Name, Group_Leader#, Group_Leader_Name

Album# -> Album_Name, Album_Type, Date_of_Creation, Album_Description,

Album# -> Group_Leader#, Album_Approver#, Approval_Date

Album# -> Trailer_Release_Date, Trailer_Release_URL, Album_Release_Date

Candidate Key: Member# Album# Member-Role

Incoming_URL_for_View

Partial Dependencies:

Album# -> Group#, Album_Name, Album_Type, Date_of_Creation, Album_Description, Group_Leader#, Album_Approver#, Approval_Date, Trailer_Release_Date, Trailer_Release_URL, Album_Release_Date

Incoming_URL_for_View -> View_Date, Comments, Trailer_Release_Date, Trailer_Release_URL

Album (Album# ,Group#, Album_Name, Album_Type, Date_of_Creation, Album_Description,Group_Leader#, Album_Approver#, Approval_Date,Trailer_Release_Date, Trailer_Release_URL, Album_Release_Date)

Album_view (Incoming_URL_for_View, View_Date, Comments,Trailer_Release_Date, Trailer_Release_URL)

Member_Group (Member#, Member_Name, Group#, Group_Name, Member-Role, Group_Music_Class, Group_Leader#, Group_Leader_Name)

Album_Distribution_and_Download:

Functional Dependency:

Distributor# -> Distributor_Name, Distributor_Location

Album#, **Distributor#** -> Price

Album# -> Album_release_date

Download# -> Incoming_URL_for_Download, Download_Request_Date, Downloaded_Album#, Download_Status

Candidate Key: Download#

Album_Distribution_and_Download (Album#, Album_Release_Date, Distributor#, Distributor_Name, Distributor_Location, Price, Download#, Incoming_URL_for_Download, Download_Request_Date, Downloaded_Album#, Download_Status)

There is still a need to go for higher normal Form as 2NF can cause update anomalies which could be countered as we move on to 3NF. We need to remove column level duplicates.

Third Normal Form

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

Participant (Phone, Member#, Firstt_Name, Age, City, Email, Advt_Seen, Status_Round1, Status_Round2)

Candidate Key: Phone

Member# -> Firstt_Name, Age, City, Email, Advt_Seen, Status_Round1, Status_Round2

(violates 3NF)

We split it into:

Participant (Member#, Firstt_Name, Age, City, Email, Advt_Seen, Status_Round1, Status_Round2)

Participant_phone (Phone, Member#)

Participant_Entry (File Url, Member#, Album_Type, Submission Date)
(no change needed)

Now we check:

Panelist_Album_Evaluation (Panelist#, Panelist_Name, Experience, Association_Month, Association_Year, File Upload Path)

File Upload Path -> Paneliest#

Panelist# -> Panelist_Name, Experience, Association_Month, Association_Year

(violate 3NF)

We split this into:

Panelist (Panelist#, Panelist_Name, Experience, Association_Month, Association_Year)

Panelist_Album_Evaluation (Panelist#, File Upload Path)

Now we check:

Album (Album# ,Group#, Album_Name, Album_Type, Date_of_Creation, Album_Description,Group_Leader#, Album_Approver#, Approval_Date,Trailer_Release_Date, Trailer_Release_URL, Album_Release_Date)

Group# -> Group_Leader#
(violate 3NF)

We split this into two tables:

Album (Album# , Album_Name, Album_Type, Date_of_Creation, Album_Description, Album_Approver#, Approval_Date,Trailer_Release_Date, Trailer_Release_URL, Album_Release_Date)

Group_Leader (Group# , Group_Leader#)

Album_view (Incoming URL for View, View_Date, Comments,Trailer_Release_Date, Trailer_Release_URL)
(already in 3NF)

Member_Group (Member#, Group#, Group_Name, Member-Role, Group_Music_Class, Group_Leader#, Group_Leader_Name)

Group# -> Group_Name, Member-Role, Group_Music_Class,
Group_Leader#, Group_Leader_Name
(violates 3NF)

We split this into:

Member_Group (Member#, Group# , Member-Role)

Group (Group#, Group_Name, Group_Music_Class, Group_Leader#,
Group_Leader_Name)

Now we check:

Album_Distribution_and_Download (Album#,
Album_Release_Date, Distributor#, Distributor_Name,
Distributor_Location, Price, Download#, Incoming_URL_for_Download,
Download_Request_Date, Downloaded_Album#, Download_Status)

Distributor# -> Distributor_Name, Distributor_Location

Album#, Distributor# -> Price

(These violates 3NF)

Distributor (Distributor#, Distributor_Name, Distributor_Location)

Album_Dist_Price (Album#, Distributor#, Price)

Download (Download#, Incoming_URL_for_Download,
Download_Request_Date, Downloaded_Album#, Download_Status)

BCNF

Participant (Member#, First_Name, Middle_Name, Last_Name Age, City, Email, Advt_Seen, Status_Round1, Status_Round2)

Participant_Phone (Phone, Member#)

Participant_Entry (File Url ,Member#, Album_Type, Submission Date)

Panellist (Panelist#, Panelist_Name, Experience, Association_Month, Association_Year)

Panelist_Album_Evaluation (Panelist#, File Upload Path)

Album (Album# , Album_Name, Album_Type, Date_of_Creation, Album_Description, Album_Approver#, Approval_Date, Trailer_Release_Date, Trailer_Release_URL, Album_Release_Date)

Group_Leader (Group# , Group_Leader#)

Album_view (Incoming URL for View, View_Date, Comments, Trailer_Release_Date, Trailer_Release_URL)

Member_Group (Member#, Group# , Member-Role)

Group (Group#, Group_Name, Group_Music_Class, Group_Leader#, Group_Leader_Name)

Distributor (Distributor#, Distributor_Name, Distributor_Location)

Album_Dist_Price (Album#, Distributor#, Price)

Download (Download#, Incoming_URL_for_Download, Download_Request_Date, Downloaded_Album#, Download_Status)

As we got all the tables in their optimized form so there is no need to normalize things further.

C.)

For the Album_Distribution_and_Download relation, there are no composite keys that can be formed from the attributes be it, Album#, Distributor#, Download_Album#. Hence there is no way the rows of the Album_Distribution_and_Download can be represented uniquely by composition of the attributes. Hence there is a need for a system-generated key to uniquely represent the rows of the relation table. By using Download# as a primary key we can index each tuple of the relation uniquely. Since we are not using any natural key, and Download# is an artificial key which uniquely identifies each tuple, we can claim that Download# is a surrogate key.