DBMS MID-SEM ASSIGNMENT

Neha Goel (2019066)
Prashasti Agarwal (2019075)
Raghav Nakra (2019083)

Table Of Contents

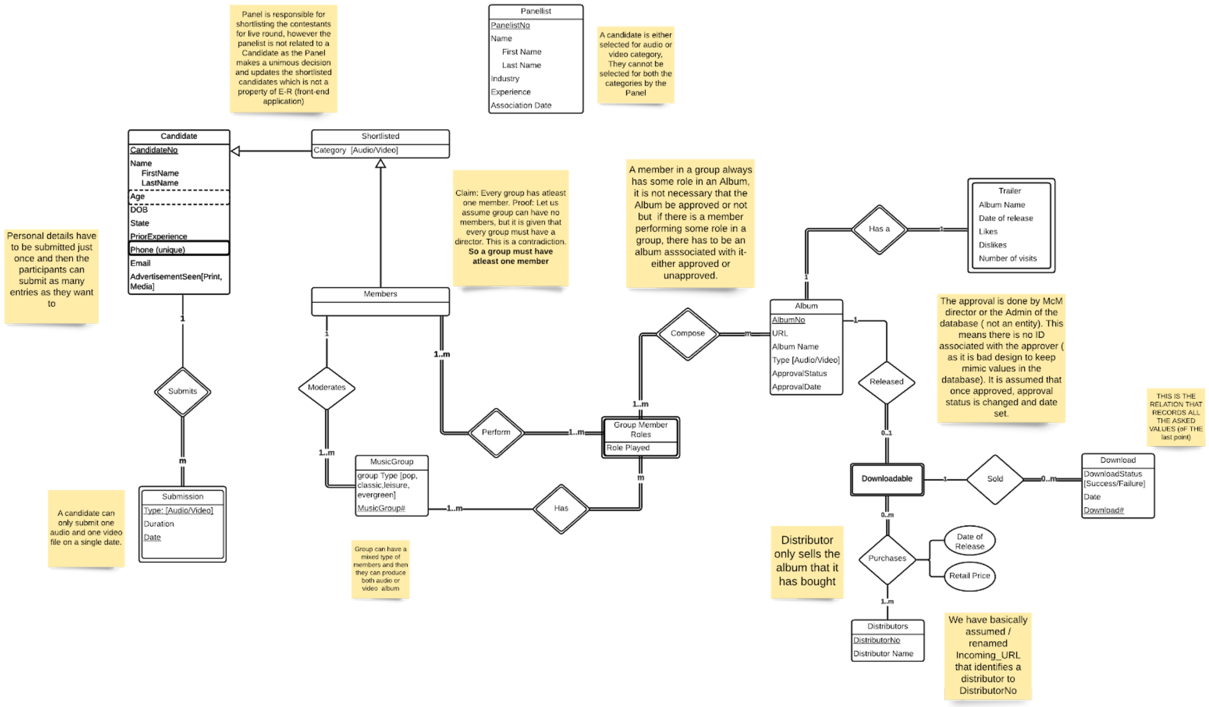
Assumptions

1. Same person can be director or multiple music groups
2. Album Name of multiple albums can be same
3. Considering the worst case scenario by having 2 people sharing the same personal information on a broader perspective.
4. DistributorNo is the "Incoming URL" as mentioned in the question which can uniquely identify each distributor.
5. One person is having only one role for a specific album.
6. Distributors can make multiple downloads for the same album.
7. Other assumptions are mentioned in the ER diagram in the sticky notes

Question 1

Er diagram is as follows:

⇒ The assumptions for various entities are inserted as notes in the E-R for better and easy understanding.



Panel is responsible for shortlisting the contestants for live round, however the panelist is not related to a Candidate as the Panel makes a unimous decision and updates the shortlisted candidates which is not a property of E-R (front-end application)

A candidate is either selected for audio or video category, They cannot be selected for both the categories by the Panel

Claim: Every group has atleast one member. Proof: Let us assume group can have no members, but it is given that every group must have a director. This is a contradiction. **So a group must have atleast one member**

A member in a group always has some role in an Album, it is not necessary that the Album be approved or not but if there is a member performing some role in a group, there has to be an album asssociated with it- either approved or unapproved.

Personal details have to be submitted just once and then the participants can submit as many entries as they want to

The approval is done by McM director or the Admin of the database (not an entity). This means there is no ID associated with the approver (as it is bad design to keep mimic values in the database). It is assumed that once approved, approval status is changed and date set.

THIS IS THE RELATION THAT RECORDS ALL THE ASKED VALUES (oF THE last point)

A candidate can only submit one audio and one video file on a single date.

Group can have a mixed type of members and then they can produce both audio or video album

Distributor only sells the album that it has bought

We have basically assumed / renamed Incoming_URL that identifies a distributor to DistributorNo

A better view can be obtained here

Question 2

1. Candidate (CandidateNo: integer, FirstName: varchar(30), LastName: varchar(30), DOB: date, State: varchar(30), PriorExperience: integer, Email: varchar(50), AdvertisementSeen: enum{“Print”, “Media”})
   Candidate Keys: **Email**
   We are not considering remaining all attributes except Email, CandidateNo combined to be a candidate key as we are assuming all the information can be coincidently the same for 2 people.

2. Submission (CandidateNo: integer (FK references Candidate (CandidateNo)), Duration: Time() (check Duration ≥ 2 and ≤ 5 mins), SubmissionDate: date, SubmissionType: enum {“Audio”, “Video”})
   Candidate Keys: -
   We are not considering remaining all attributes except CandidateNo, combined to be a candidate key as we are assuming all the information can be coincidently the same for 2 people.

3. PhoneNumber (CandidateNo: integer (FK references Candidate(CandidateNo), PhoneNumber: integer(10))
   Candidate Keys: -
   CandidateNo can’t be a candidate key as they can have multiple values corresponding to them.

4. panellist (panellistNo: integer, FirstName: varchar(30), LastName: varchar(30), Industry: varchar(30), Experience(yrs): Integer, AssociationDate: date)
   Candidate Keys: -
   We are not considering remaining all attributes except CandidateNo, combined to be a candidate key as we are assuming all the information can be coincidently the same for 2 panellist.

5. Shortlisted(CandidateNo: integer (FK references Candidate(CandidateNo)), Category: enum{“Audio”, “Video”})
   Candidate Keys: -

6. Member(MemberNo: integer (FK references Shortlisted(CandidateNo)))
   Candidate Keys: -

7. MusicGroup (ModeratorNo: integer (FK references Member(MemberNo)), Type: enum{“pop”, “classic”, “leisure”, “evergreen”}, MusicGroupID: integer)
   Candidate Keys: -
   ModeratorNo can’t be a candidate key because we are assuming that the same person can be director of multiple groups.

8. Album(AlbumNo: integer, URL: varchar(50), AlbumName: varchar(30), Type: enum{“Audio”/ “Video”}, ApprovalStatus: enum {“Approved”, “Unapproved”} , ApprovalDate: date)
   Candidate Key : URL
   AlbumName can’t be a candidate key as we are assuming 2 Albums to have the same name, since one might get approved while the other does not.

9. Trailer(AlbumNo:integer (FK references Album(AlbumNo)), AlbumName: varchar(30), DateOfRelease: Date, Likes: Integer (check Likes ≤ NumberOfVisits), Dislikes: Integer (check Dislikes ≤ NumberOfVisits), NumberOfVisits: Integer ((check Likes +Dislikes ≤ NumberOfVisits)))
   Candidate Keys: -
   We are not considering remaining all attributes (except AlbumNo) combined to be a candidate key as we are assuming all the information can be coincidently the same for 2 albums.

10. Distributors(DistributorNo: Varchar, DistributorName: varchar(30))
    Candidate Keys: -

11. Downloadable(AlbumNo: integer (FK references Album(AlbumNo) (check Album Status should be approved), DistributorNo: varchar(FK references Distributors(DistributorNo)), DateOfRelease: date, RetailPrice: integer)
    Candidate Keys: -

12. Download(DownloadNo: integer, Date: date, DownloadStatus: enum{“Successful”, “Failure”}, AlbumNo: integer, DistributorNo: varchar(FK (AlbumNo, DistributorNo) references Downloadable(AlbumNo, DistributorNo)))
    Candidate Keys: -

13. Roles (MusicGroupID: integer (FK references MusicGroup(MusicGroupID)), AlbumNo :integer (FK references Album(AlbumNo)), MemberNo: integer (FK references Member(MemberNo)), RolePlayed: varchar(30))
    Candidate Keys: - (AlbumNo, MemberNo, RolePlayed)

Question 3

Yes, our group ended up creating a "Good Design" of McM Sangeet Database.
Reasons:

1. We Handled Redundant Information in Tuples and Update Anomalies
   Each and every tuple is a representation of of one and only one kind of entity, this can be seen with: Candidate and its submission- as we allow multiple submissions per candidate, we are not keeping redundant candidate information, rather associated all the submission (a separate weak entity) to their respective candidates via candidate id, thus preventing redundant information.

2. Attributes of different entities are not mixed in the same relation: the above example illustrates this. Table candidate refers to information only about the candidates and no information about the submissions as they are different entities with their own attributes. Only foreign keys in submission are used to refer it to the candidate so as to get the complete excess information.

⇒ Our schema is easily explainable relation by relation. The attribute names and constraints are highly reasonable and intuitive making it self explanatory.

3. Design a schema that does not suffer from the insertion, deletion and update anomalies.
   There is a very efficient cascading and updating implemented in our database. For example In case a candidate drops out, we simply delete all his submissions, their phone numbers. This ensures seamless operation of the entire database even when such a major thing (dropping out of a candidate) occurs . On updating a candidate Number Value, the effects are reflected in every key that references the candidate.

4. If there are any anomalies present, then note them so that applications can be made to take them into account.
   Triggers have been implemented to take care of abnormal values. For eg. there is a trigger to make sure that any album which is declared as downloadable must be approved by MCM director, if that is not the case, error is thrown to take care of the entry. Another example is on deletion of a candidate member who is currently moderator of a group from the records, null values have been assigned indicating that the following group needs to have a moderator reassigned as a result of the member deletion.

5. Less number Null Values in Tuples

   As null values are unpredictable, they can indicate anything- unknown value, undiscovered it is dangerous to keep this in a table. Hence we have added several NOT NULL constraints wherever possible to ensure the least number of null values possible.

6. Avoid generating Spurious Tuples at any cost

   There are no fake tuples that mimic anything. There are no sample or representative values, all entries are true to what they represent. All the incoming entries are consistent with the previously entered value thus creating very less or none erroneous join operation results. Foreign keys have been added wherever possible to ensure valid entries to child tables i.e. only those available in the parent table.

Question 4

A.

- $\rho(\text{A, Album})$
- $\rho(\text{D, Downloadable})$
- $\rho(\text{X, A} \bowtie_{\text{A.AlbumNo =D.AlbumNo}}(\text{D}))$
- $\Pi_{\text{AlbumName}}(\sigma_{\text{(X.AlbumType = ‘Audio’)} \cap \text{((DateOfRelease >= ‘2020-01-01’)} \cap \text{(DateOfRelease <= ‘2020-12-31’))}}(\text{X}))$

OR

- $\rho(\text{A, Album})$
- $\rho(\text{D, Downloadable})$
- $\rho(\text{X, A} \bowtie \text{D})$
- $\Pi_{\text{AlbumName}}(\sigma_{\text{(X.AlbumType = ‘Audio’)} \cap \text{((DateOfRelease >= ‘2020-01-01’)} \cap \text{(DateOfRelease <= ‘2020-12-31’))}}(\text{X}))$

B.

- $\rho(\text{S1, Submission})$
- $\rho(\text{S2, Submission})$
- $\text{Candidate} \bowtie_{\text{Candidate.CandidateNo = S1.CandidateNo}} (\sigma_{\text{(S1.CandidateNo = S2.CandidateNo)} \cap \text{(S1.SubmissionType} \neq \text{S2.SubmissionType)}}(\text{S1 X S2}))$

OR

- $\rho(\text{S1, Submission})$
- $\rho(\text{S2, Submission})$

- $\text{Candidate} \bowtie (\sigma_{\text{(S1.CandidateNo = S2.CandidateNo)} \cap \text{(S1.SubmissionType} \neq \text{S2.SubmissionType)}}(\text{S1 X S2}))$

C.

- ρ(R1, Roles)
- ρ(R2, Roles)
- $\text{Candidate} \bowtie_{\text{Candidate.CandidateNo = R1.CandidateNo}} (\sigma_{\text{(R1.MemberNo = R2.MemberNo)} \cap \text{(R1.MusicGroupID} \neq \text{R2.MusicGroupID)}}(\text{R1 X R2}))$

OR

- ρ(R1, Roles)
- ρ(R2, Roles)
- $\text{Candidate} \bowtie (\sigma_{\text{(R1.MemberNo = R2.MemberNo)} \cap \text{(R1.MusicGroupID} \neq \text{R2.MusicGroupID)}}(\text{R1 X R2}))$

D.

- ρ(R1, Roles)
- ρ(R2, Roles)
- $\text{Candidate} \bowtie (\sigma_{\text{Type = 'pop'}}(\text{MusicGroup} \bowtie (\Pi_{\text{MemberNo, MusicGroupID}}(\text{Roles}) - (\Pi_{\text{R1.MemberNo, R1.MusicGroupID}}(\sigma_{\text{(R1.MemberNo = R2.MemberNo)} \cap \text{(R1.MusicGroupID} \neq \text{R2.MusicGroupID)}}(\text{R1 X R2})))))$

E.

- $\rho(\text{T, Download} \bowtie_{\text{Download.AlbumNo = Album.AlbumNo}}(\text{Album}))$
- ρ(A,T)
- ρ(B,T)

- Distributors $\bowtie$ ($\sigma_{\text{(A.DistributorNo = B.DistributorNo)} \cap \text{(A.AlbumType} \neq \text{B.AlbumType)}}$(A X B))

OR

- ρ(T, Download $\bowtie$ Album)
- ρ(A,T)
- ρ(B,T)
- Distributors $\bowtie$ ($\sigma_{\text{(A.DistributorNo = B.DistributorNo)} \cap \text{(A.AlbumType} \neq \text{B.AlbumType)}}$(A X B))

Question 5

A.

```
SELECT
    *
FROM
    Album
WHERE
    AlbumNo IN (SELECT
            AlbumNo
        FROM
            Downloadable
        WHERE
            YEAR(DateOfRelease) = 2020)
        AND AlbumType = 'Audio';
```

B.

```
With TheGroups(C, MemberNo) as (Select count(*) as C, MemberNo
from Roles group by MemberNo having C>1 )
Select * from candidate where CandidateNo in TheGroups.MemberNo;
```

OR

```
SELECT
    *
FROM
    Candidate
WHERE
    CandidateNo IN (SELECT DISTINCT
            A.MemberNo
        FROM
            Roles A,
            Roles B
        WHERE
            A.MemberNo = B.MemberNo
                AND—NOT (A.MusicGroupID = B.MusicGroupID));
```

OR

```
SELECT
    *
FROM
    Candidate
WHERE
    CandidateNo IN (SELECT DISTINCT
            A.MemberNo
        FROM
            Roles A INNER JOIN
            Roles B
        ON
            A.MemberNo = B.MemberNo
                AND NOT (A.MusicGroupID = B.MusicGroupID));
```

C.

```
SELECT
    *
FROM
    Candidate
WHERE
    CandidateNo IN (SELECT
            MemberNo
        FROM
            Roles
        WHERE
            MemberNo NOT IN (SELECT DISTINCT
                    A.MemberNo
                FROM
                    Roles A INNER JOIN
                    Roles B
                ON
                    A.MemberNo = B.MemberNo
                        AND NOT (A.MusicGroupID =
B.MusicGroupID))
                AND MusicGroupID IN (SELECT
                    MusicGroupID
                FROM
```

```
                        MusicGroup
                    WHERE
                        Type = 'pop'));
```

D.

```
SELECT
    *
FROM
    candidate
WHERE
    candidateNo IN (SELECT
            A.CandidateNo
        FROM
            Submission A INNER JOIN
            Submission B
        ON
            A.CandidateNo = B.CandidateNo
                AND NOT (A.SubmissionType = B.SubmissionType));
```

OR

```
Select * from candidate where CandidateNo IN (Select CandidateNo
from Submission where SubmissionType= “Audio” intersects Select
CandidateNo from Submission where SubmissionType= “Video”)
```

E.

```
With NewTable(MediaForm,count)as (Select AdvertisementSeen, count(*)
from submission INNER JOIN Candidate ON Candidate.CandidateNo =
submission.CandidateNo group by AdvertisementSeen) Select max(count)as
"Number of entries", MediaForm from NewTable ;
```

F. (BONUS)

```
      import mysql.connector
      import sqlite3

      mydb = mysql.connector.connect(host = "127.0.0.1", user = "root",
passwd = "12345", port = "3306", database = "midsem")

      mycursor = mydb.cursor(buffered=True)
      sql_command='''
      with sometable(Amount, DistributorNo, Type) as
      (Select count(*), DistributorNo, AlbumType  from Download inner join
album on Download.AlbumNo=album.AlbumNo and
Download.DownloadStatus="Successful"
       group by AlbumType, DistributorNo)
       Select max(Amount) over(),Type  from sometable where type="Audio"
      UNION
       Select max(Amount) over(),Type  from sometable where type="Video";
'''

      sql_command2='''
      with sometable(Amount, DistributorNo, Type) as
      (Select count(*), DistributorNo, AlbumType  from Download inner join
album on Download.AlbumNo=album.AlbumNo and
Download.DownloadStatus="Successful"
       group by AlbumType, DistributorNo)
      Select DistributorNo, Type, Amount as "Copies_Sold"  from sometable
where type= (%s) and Amount= (%s)
      '''
      mycursor.execute(sql_command)
      cursor2=mydb.cursor(buffered=True)
      query="Select DistributorName from Distributors where DistributorNo=
(%s);"
      cursor3=mydb.cursor(buffered=True)
      query2="""Select distinct MemberNo, Candidate.FirstName as \"Name\"
from Roles,
      Candidate where Roles.MemberNo=Candidate.CandidateNo and  AlbumNo in
(Select Download.AlbumNo
      from Download INNER JOIN Album ON Album.AlbumNo=Download.AlbumNo
where Download.DistributorNo= (%s) and Album.AlbumType=(%s));"""
      cursor4=mydb.cursor(buffered=True)
      for row in mycursor.fetchall():
              maximum,AlbumType=row
              cursor4.execute(sql_command2,(AlbumType,maximum,))
              for row2 in cursor4.fetchall():
                      DistributorNo,AlbumType, Sold_Items = row2
                      DistributorName = ""
                      cursor2.execute(query,(DistributorNo,))
                      for row2 in cursor2.fetchone():
                          DistributorName=row2
                          print("Distributor No. ",DistributorNo,"
Distributor  Name ", DistributorName," Album Type ", AlbumType,"Items Sold
", Sold_Items,"\n")
                          print("Members Associated :\n")
                          cursor3.execute(query2,
(DistributorNo,AlbumType,))
                          for row3 in cursor3.fetchall():
                              MemberNo,MemberName=row3
                              print(MemberNo,MemberName,"\n")
      mydb.close()
```

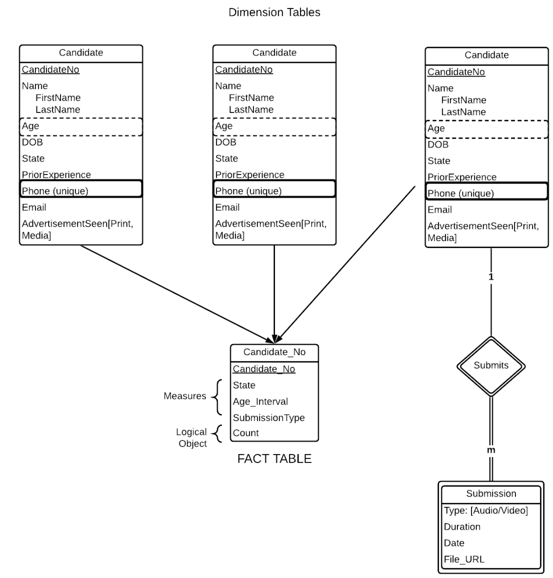
Question 6

A.

We need to design a multidimensional data model. (Implementation of OLAP to be quick and efficient)

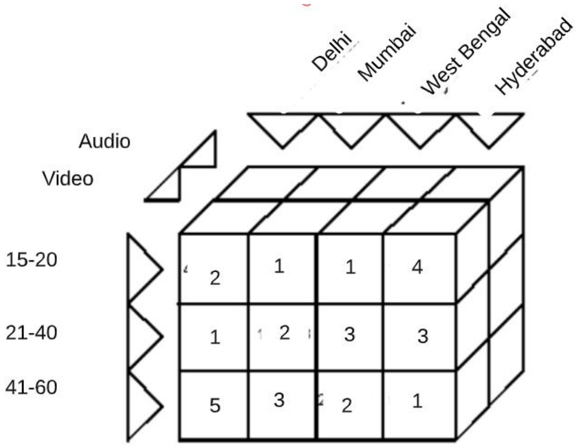
1. Logical Model - We organise data in the form ofs **data cubes** this is because each measure(columns) have the same relation to the logical object (count in this case) and can be easily analysed and displayed together**.** These data cubes are filled with **measures** to provide information about analytics. Edges of the cubes are called **dimension**s that contain unique values to categorize data- each dimension has a table associated to it (**called the dimension Table**) to store the data concerning only that dimension (our case AlbumType, State, ageInterval) . The cube contains various **levels and hierarchy (level is a position in a hierarchy)** of information. **Trend**s are recognised by analysing data level by level.Cubes can also contain **attributes** that convey extra information about the data for displaying purposes.
2. Relational Model- It is basically a **Star Schema (**the most preferred schema for organising data**). " A star schema is a convention for organizing the data into dimension tables, fact tables, and materialized views. Ultimately, all of the data is stored in columns, and metadata is required to identify the columns that function as multidimensional objects."** In our design, the dimension Tables (1) Candidate that contain information about Album Type (2) Candidate - AgeInterval (3) Candidate- State all 3 contribute to make the fact

table “statistics” that contains various data groupings



Star Schema to implement the required statistics.

3. **Conceptually the multidimensional data can be modelled is as follows in the form of a data cube:**



	Delhi	Mumbai	West Bengal	Hyderabad
15-20	2	1	1	4
21-40	1	2	3	3
41-60	5	3	2	1

4. To analyse based on any of the given measures, one could slice and dice the cube to get the information required. For example if we slice the cube then we get groupings based on States and SubmissionType on dicing we get information about age groups and SubmissionType and so on.

5. For modelling Zones and States we can use pivots to model them in rows and columns placing Zones as Columns and States as rows

B.

USING GROUP BY CUBE()

This is to display different combinations of data in a single table. This includes grouping only by State, Grouping only by AlbumType, Grouping only by age, Group by (State,AlbumType) , Group by (Age,AlbumType) , Group by (State,age) and then all 3.

```
Create table statistics as (Select State, SubmissionType,
CASE
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN 15
AND 20 THEN '15-20'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN 21
AND 25 THEN '21-25'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN 26
AND 30 THEN '26-30'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN 31
AND 40 THEN '31-40'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN 41
AND 100 THEN '41-100'
END as Age_Interval, Count(*) as Count  from Candidate INNER JOIN
Submission  ON Candidate.CandidateNo= Submission.CandidateNo group by
CUBE (State,SubmissionType,Age_Interval)) ;
```

C.

Assuming that question asks that we don't need to analyse the previous data, we just need to group the entries by zone and state.

USING ONLY AGGREGATE FUNCTIONS

```
CREATE TABLE state_zones (
    State VARCHAR(50),
    Zone ENUM('North', 'South', 'East', 'West')
);
with Cities(State) as (select Candidate.State from Candidate
inner join Submission on Submission.CandidateNo=
Candidate.CandidateNo)

Select count(*), Zone from state_zones inner join Cities on
Cities.State=state_zones.state group by Zone;
```

OR

USING PIVOT

```
CREATE TABLE state_zones (
    State VARCHAR(50),
    Zone ENUM('North', 'South', 'East', 'West')
);

-- Assuming we already have a table named state_zones and it
--already contains all the 29 states and their respective zones.

with States(State) as
 (
select Candidate.State from Candidate inner join Submission on
Submission.CandidateNo= Candidate.CandidateNo
)

Select PVT.State,PVT.[“North”],
PVT.[“South”],PVT.[“East”],PVT.[“West”]
From
States,state_zones
where States.State=Zones.Sate
PIVOT
(Count(*) for Zone in (“North”,”South”,”East”,”West”) as PVT;
```

OR

Assuming that questions asks to Analyse by Zone, Age_Intervals and AlbumType instead of State

```
Create table zone_stats as (Select Zones.Zone, SubmissionType,
CASE
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN
15 AND 20 THEN '15-20'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN
21 AND 25 THEN '21-25'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN
26 AND 30 THEN '26-30'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN
31 AND 40 THEN '31-40'
    WHEN DATE_FORMAT(now(), '%Y') - DATE_FORMAT(dob, '%Y')BETWEEN
41 AND 100 THEN '41-100'
END as Age_Interval, Count(*) as Count  from Candidate INNER JOIN
Submission  ON Candidate.CandidateNo=Submission.CandidateNo INNER
JOIN Zones ON Candidate.State=Zones.State group by CUBE
(Zone,SubmissionType,Age_Interval)) ;
```

Question 7

A.

The relational schema does not exhibit good design because it contains redundant information or null values in their stead in some places, attributes of different entities have been mixed in the same relation, it suffers from the insertion, deletion and update anomalies. The examples of each of these issues have been shown in each relation below:

Participant

1. **Redundancy:** Repetition of member personal information in case of multiple entries by same member in table. Also, repetition of member information and entries in case of multiple phone numbers for the same member.
2. **Insertion Anomaly:** To add a new entry, either a new phone number or new Member# is required.
3. **Modification Anomaly:** When member personal information is updated, it may cause **inconsistencies** in case different entries have been submitted by the same person or if multiple phone numbers are given by that person.

4. **Deletion Anomaly:** While deleting an entry, we might delete information about a participant if that participant has submitted just one entry.
5. **Null Values in Tuples:** If a participant wants to submit more than one phone number and just one entry, they may have to submit null in the file upload path attribute.
6. **Relationship instance** - The given relation contains information about at least 3 entities, them being- Members, Submissions, Phone numbers.

Panelist_Album_Evaluation

1. **Insertion Anomaly:** To add a new panelist, a new file upload path (i.e. Submission) is required since it is part of the primary key and cannot be null.
2. **Deletion Anomaly:** Whenever we delete a panelist, the submission (File Upload Path) is also deleted. This means that the submission goes ungraded and there is no method to grade this submission , this creates inconsistency and no record is left for the submission represented by the file upload path.
3. **Relationship instance** - The given relation contains information about at least 2 entities, them being- Panelist and submission.. Entities and their relations are not apart rather all are forced into one single relation.
4. **Redundancy:** In a case of multiple evaluations by a panelist, the entire panelist information is repeated creating a lot of redundant data

Member_Group_Album_Trailer

1. **Redundancy:** Repetition of group information in case there are multiple albums by the same group.
2. **Insertion Anomaly:** To add a group, we need an album too.
3. **Modification Anomaly:** When group information is updated, it may cause **inconsistencies** in case different albums have been made by the same group.
4. **Deletion Anomaly:** While deleting an album, we might delete information about a group if that is the only album made by the group and vice versa, while deleting a group, the album made by it is also deleted.
5. **Relationship instance** - The given relation contains information about at least 4 entities, them being- Members, MemberGroups, Albums and Trailers. Entities and their relations are not apart rather all are forced into one single relation.

Album_Distribution_and_Download

1. **Redundancy:** Repetition of distributor information for multiple downloads.

2. **Insertion Anomaly:** To add a distributor we need a download too (which is not the case in reality since it is possible that a distributor is unable to sell any albums)
3. **Modification Anomaly:** When a distributor's information is updated it may cause **inconsistencies** since the same distributor could be associated with many downloads.
4. **Deletion Anomaly:** When deleting a download, we might delete information about a distributor since it is possible that download is the only one associated with the distributor.
5. **Relationship instance** - The given relation contains information about at least 4 entities, them being- Albums, Distributors, Downloads.

B.

One can end up to a good design (efficient, non redundant, following relational schema representation rules) by Normalising the given 1NF form to the maximum possible. As converting a relation table to relational schema implies we are not having multiple values in a single row, i.e. for each entity in a table we are having atomic values corresponding to each attribute. Normalization refers to a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies, helping us to divide larger tables into smaller ones by linking them through relationships. Hence Normalisation is the answer :).
NOTE KEYS FOR REPRESENTATION:

1NF
2 NF
3 NF
BCNF
Assumptions about F.D's

NORMALISATION

1. Participant(Member#, Name, Age, City, Phone, Email, Prior_Experience, Advt_Seen, Album_Type, Submission Date, File_Upload_Path, Status_Round1, Status_Round2).

To convert to 1NF

Here File_Upload_Path is a multivalued attribute(given the condition that a Participant can submit multiple entries) So we need to make File_Upload_Path a part of Prime Attribute as

well. Also Member# is not a prime attribute as Phone and File_Upload_Path can together determine the uniqueness of an entity. Note : There cannot be multiple phone numbers associated with an upload as phone number has been broken down to atomic quantity.
So the 1NF form is:
Participant(Member#, Name, Age, City, Phone, Email, Prior_Experience, Advt_Seen, Album_Type, Submission Date, File_Upload_Path, Status_Round1, Status_Round2).

1NF to 2NF
In 2NF form all non-key attributes are fully functional dependent on the primary key.

The given Functional Dependencies can be interpreted as:
Phone → Member# (Primary Attribute is dependent to Primary Attribute)
So this is a partial dependency and we can simply add them to a new table

PhoneBook(Phone, Member#)
Primary Key- Phone
Foreign Key: Member# References Member(Member#)
⇒ This table cannot be split further hence PhoneBook i**s in BCNF**

Note: Adding Submission_Date to the dependency as well, as it is highly logical to assume that File_Upload Path Determines Submission_Date
File_Upload_Path → Member#, Album_Type, Submission_Date (Prime Attribute to Prime ,Non-Prime)
So this is a partial dependency and we can simply add them to a new table

Submission(File_Upload_Path,Member#, Album_Type, Submission_Date)
Primary Key- File_Upload_Path
Foreign Key: Member# References Member(Member#)
⇒ This table cannot be split further hence Submission i**s in BCNF**
Member# → Member_Name, Age, City, Email (Non Prime attribute to non-prime Attributes)
⇒ So we need not take care of this is in 2NF
As no information is given about rest of the attributes, they are simply mapped to Prime Attributes, hence, we obtain the table-
Rest(File_Upload_Path, Phone , Member# , Name, Age, City, Email, Prior_Experience, Advt_Seen, Status_Round1, Status_Round2)

2NF to 3NF

2 of the 3 functional dependencies have been considered leaving-

Member# → Member_Name, Age, City, Email (Non Prime attribute to non-prime Attributes)

This results in breaking down the table Rest(File_Upload_Path, Phone , Member# , Name, Age, City, Email, Prior_Experience, Advt_Seen, Status_Round1, Status_Round2)

To

Member(Member# , Member_Name, Age, City, Email)

Primary Key-Member#

⇒ This table cannot be split further hence Member i**s in BCNF**

Also the remaining non prime attributes can be mapped to the primary keys to create the following table:

Information(File_Upload_Path, Phone, Prior_Experience, Advt_Seen, Status_Round1, Status_Round2)

Primary Key - File_Upload_Path, Phone

Foreign Key-File_Upload_Path references Submission(File_Upload_Path),

Foreign Key- Phone references PhoneBook(Phone)

⇒ This table cannot be split further hence Information i**s in BCNF**

Since all the attributes have been broken down to BCNF we need not proceed further.

2. Panelist_Album_Evaluation(Panelist#, Panelist_Name, Experience, Association_Month, Association_Year, File_Upload_Path)

Convert to 1 NF

Already in 1 NF Form

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

1 NF to 2 NF

In 2NF form all non-key attributes are fully functional dependent on the primary key.

The given Functional Dependencies can be interpreted as:

Panelist → Panelist_Name, Experience, Association_Month,Association_Year

so we create a new table with it.

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

Primary Key- Panelist#

Evaluation(File_Upload_Path,Panelist#)
Primary Key- File_Upload_Path
Foreign Key :File_Upload_Path references Submission(File_Upload_Path))
Foreign Key - Panelist references Panelist(Panelist#)

⇒ This table cannot be split further **Panelist and Evaluation are in BCNF**
Since all the attributes have been broken down to BCNF we need not proceed further.

3. Member_Group_Album_Trailer(Member#, Member_Name, Group#, Group_Name, Member-Role, Group_Music_Class, Album#, Album_Name, Album_Type, Date_of_Creation, Album_Description, Group_Leader#, Group_Leader_Name, Album_Approver#, Approval_Date, Trailer_Release_Date, Trailer_Release_URL, Incoming_URL_for_View, View_Date, Comments)

   Convert to 1NF

   Here Group# needs to be added as a primary key as a member can belong to multiple groups (can be seen as a multivalued attribute) and hence needs to be broken down to atomic values and made a primary key

   Member_Group_Album_Trailer(Member#, Member_Name, Group#, Group_Name, Member-Role, Group_Music_Class, Album#, Album_Name, Album_Type, Date_of_Creation, Album_Description, Group_Leader#, Group_Leader_Name, Album_Approver#, Approval_Date, Trailer_Release_Date, Trailer_Release_URL, Incoming_URL_for_View, View_Date, Comments)

   1NF to 2 NF

   The given functional dependencies can be interpreted as-

Note: Adding a new Functional Dependency Member# → MemberName as it is highly logical to assume that memberid can give the name of the member (Inheritance actually Member is a form of Participant Only)

Member# → MemberName (prime attributes to non Prime)

So this is a partial dependency and we can simply create a table.

Names(Member#,MemberName)
Primary Key-Member#
Foreign Key- member# references Participant(Member#)

⇒ This table cannot be split further hence Names i**s in BCNF**

Note: Adding Group_Music_Class to the dependency as well, as it is highly logical to assume that For each group has only one name given to be {pop/leisure..}

Group# → Group_Name, Group_Leader#, Group_Leader_Name, Group_Music_Class (Prime Attributes to Non Prime)

So this is a partial dependency and can be broken down into the table-

Groups(<u>Group#,</u> Group_Name, Group_Leader_Name,Group_Leader#,Group_Music_Class)

Primary Key -Group#

⇒ This table cannot be split further hence Groups i**s in BCNF**

Album# → Album_Name, Album_Type, Date_of_Creation, Album_Description

and

Album# → Group_Leader#, Album_Approver#, Approval_Date (Prime to Non- Prime Attributes)

and

Note: Dropping Album_Release_Date from the provided FD as there is no attribute in given table named "Album_Release_Date".

Album# → Trailer_Release_Date, Trailer_Release_URL

It is a partial Dependency so can be broken down to the following table-

Album (<u>Album#</u> , Album_Name, Album_Type, Date_of_Creation, Album_Description, Album_Approver#, Approval_Date,Trailer_Release_Date, Trailer_Release_URL, Album_Release_Date, Trailer_Release_Date, Trailer_Release_URL)

Primary Key- Album#

⇒ This table cannot be split further hence Album i**s in BCNF**

Album# → Group# (Prime Attribute to Prime Attribute)

So we create a new Table

Production(<u>Album#,</u>Group#)

Primary Key- Album#

Foreign Key - Album# references Album (Album#)

Foreign Key- Group# references Groups(Groups#)

⇒ This table cannot be split further hence Production i**s in BCNF**

Group# ,Trailer_Release_Date → Album#

All the above attributes have been covered and converted to BCNF Form so this FD can simply be ignored.
The remaining rest(Album#, Member#, Member Role, Incoming_Url_For_View, View_Date, Comments)
Comments are not associated to Member# in any way are dependent on Album#, Incoming_URL_For_View, for recognition.
But Album#+Incoming_URL_For_View is not a complete candidate key, so to make it to 2NF, we separate the tables:

Comment(Album# ,Incoming_Url_For_View, Comments, View_Date)

Primary Key- Album#, Incoming_Url_For_View
Foreign Key- Album# references Album(Album#)
⇒ BCNF

Since each member can have different roles to play in an album.
Making Member Roles a primary Key along with Member# and Album#
Roles(Member#, Album#,Member_Roles)
Primary Key- Member#, Album#, Member_Roles
Foreign Key- Album# references Album(Album#)
Foreign Key- Member# references Members(Members#)
⇒ BCNF

4. 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)

To convert to 1NF

Already in 1 NF Form

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)

<u>1NF to 2NF</u>

Already in 2NF Form

In the 2NF form, all the non-primes attributes should be totally dependent on the prime attributes and not any part of those attributes.

The functional dependencies can be viewed as follows:
Download# → All attributes
In this case, Download# can find all the other attributes all by itself. So, we don't find any non-prime attribute which is dependent on any proper subset of the candidate key.

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)

<u>2NF to 3NF</u>

For any relation to be in 3NF form, either the right hand side should have the primary key, or the left hand side should be a superkey, if any one of the 2 doesn't follow then it is not in 3NF.

Note: Adding Album_Release_Date to the dependency as well, as it is can be easily taken from Album Number keeping in mind the Member_Group_Album_Trailer

1. Album# → Album_Release_Date
2. Distributor# → Distributor_Name, Distributor_Location
3. Album#, Distributor# → Album_Release_Date, Price

Clearly, Album# and the other part is transitively dependent on Download# hence they are **not in 3NF form**. This can be broken down further into:

Distributor(<u>Album#, Distributor#</u>, Album_Release_Date, Price)
Primary Key: (Album#, Distributor#)
Foreign Key: Album# references Album(Album#);

Foreign Key:Distributor# references Distribution(Distributor#)

Download(<u>Distributor#</u>, Distributor_Name, Distributor_Location)
Primary Key: Distributor#
Distributor# references Distribution(Distributor#)

⇒ This table cannot be split further hence Distribution and Download **are in BCNF**

C. (BONUS)

Surrogate key acts as a unique identifier for an entity, it is NOT derived from the data given to us, rather it is a meaningless quantity generated artificially by us solely to uniquely identify an entity. Download# is not something derived from some real world entity, it has just been created (by us) to uniquely identify a row since none of the other attributes or set of attributes in the table Album_Distribution_and_Download can uniquely identify a row. This is because an Album can be sold by a single distributor multiple times making none of them (Album# or Distributor#), even together fit for being the primary key. Hence, Download# is used as a surrogate key.