Group 2: Zachary Harrison, Evan Dweck, Yash Bidasaria

1. Schema Refinement

<u>Relational Model From Stage-1</u>

- User(Email: String, Name: String, Password: String);
- Basic(Email: String,DOB: Date);
- Artist(Email: String, Country: String, Year Founded: int, Genre: String, No. of Members: int, Most_Popular_Album_Name: String, Most_Popular_Song_Name: String);
- Song(Email: string, Album.Name: string, Name: String, Plays: int);
- Album(Email: String, Name: String, Release Date: Date, Genre: String, No. of Songs: int); Admin(AdminEmail: string, Name: string);
- Review(Email: string, Song.Name: string, Album.Name: string, AdminEmail: string, Deleted: boolean, Reviewed: boolean, Flagged_Date: date, Review_Date: date); RateAlbums(User.Email: string, Artist.Email: string, Album.Name: string, Emoji: string, Stars: int, date: Date);
- RateSongs(User.Email: string, Artist.Email: string, Album.Name: string, Song.Name: string, Emoji: string, Stars: int, date: Date);

After some discussions within ourselves and with the TAs, we decided to implement integer ids as primary keys of our relations instead of having combined keys with 3-4 attributes. In a real world application, working with a more than one attribute for a key might get too cumbersome. We ended up getting an int id as a primary key for the following relations: User, Song, Review, RateAlbums and RateSongs.

<u>New Schema after implementing such integer ids.</u>

- User(<u>User_ID</u>: int, Email: String, Name: String, Password: String IsArtist: boolean);
- Basic(<u>User_ID</u>: int,DOB: Date);
- Artist(<u>User_ID</u>: int, Country: String, Year Founded: int, Genre: String, No. of Members: int, Most_Popular_Album_Name: String, Most_Popular_Song_Name: String);
- Song(<u>Song_ID</u>: int, User_ID: int, Album.Name: string, Name: String, Plays: int);
- Album(<u>User_ID</u>: int, <u>Name</u>: String, Release Date: Date, Genre: String, No. of Songs: int);
- Admin(<u>Admin_ID</u>: int, AdminEmail: string, Name: string);
- Review(<u>Review_ID</u>: int, User_ID: int, Song_ID: int, Album.Name: string, Admin_ID: int, Deleted: boolean, Reviewed: boolean, Flagged_Date: date, Review_Date: date);
- RateAlbums(<u>RateAlbum_ID</u>: int, Rater_User_ID: int, Owner_User_ID: int, Album.Name: string, Emoji: string, Stars: int, date: Date);
- RateSongs(<u>RateSong_ID</u>: int, Rater_User_ID: int, Owner_User_ID: int, Album.Name: string, Song_ID: int, Emoji: string, Stars: int, date: Date);

Functional Dependencies and Normalization

User
User_ID → Email, Name, Password, IsArtist

Basic
User_ID→ DOB

Artist
User_ID→ Country, Year Founded, Genre, No. of Members, Most_Popular_Song_Name, Most_Popular_Album_Name

Song
Song_ID → User_ID, Album_Name, Name, Plays

$\{FD\}^+$ = { Song_ID → Song_ID,
User_ID → User_ID,
Album_Name → Album_Name,
Name → Name,
Plays → Plays,
Song_ID → User_ID, Album.Name, Name, Plays }

Album
{User_ID, Name } → Release_Date, Genre, No_of_Songs

$\{FD\}^+$ = { {User_ID, Name } → User_ID, Name,
User_ID → User_ID,
Name → Name,
Release_Date → Release_Date,
Genre → Genre,
No_of_Songs → No_of_Songs
{User_ID, Name } → Release Date, Genre, No. of Songs }

Admin
Admin_ID → AdminEmail, Name

$\{FD\}^+$ = { Admin_ID → Admin_ID,
AdminEmail → AdminEmail,
Name → Name,
Admin_ID → AdminEmail, Name }

Review
Review_ID → User_ID, Song_ID, Album.Name, AdminEmail, Deleted, Reviewed, Flagged_Date, Review_Date (FD_1)
Song_ID → User_ID, Album.Name (FD_2)

FD_2 violates BCNF because all the attributes of the Review Relation cannot be obtained by Song_ID, i.e. Song_ID is not a superkey of this relation. Therefore, through formally decomposing, we can transform this relation into two smaller ones.
Let A = all attributes of the relation Review
→Review_1 (A - User_ID, Album.Name)
→**Review_1 (Song_ID, AdminEmail, Deleted, Reviewed, Flagged_Date, Review_Date)**
And,
→Review_2 (Song_ID U (User_ID, Album.Name))
→Review_2 (Song_ID, User_ID, Album.Name)
Although the relation Song already has such attributes, therefore Review_2 is redundant and should not be created.

RateAlbums
RateAlbum.ID→ Rater_User_ID, Owner_User_ID, Album.Name, Emoji, Stars, date

RateSongs
RateSong_ID → Rater_User_ID, Owner_User_ID, Album.Name, Song_ID, Emoji, Stars, date (FD_1)
Song_ID → User_ID, Album.Name (FD_2)

(Where User_ID(FD_2) corresponds to Owner_User_ID(FD_1))
FD_2 violates BCNF because all the attributes of the Review RateSongs cannot be obtained by Song_ID, i.e. Song_ID is not a superkey of this relation. Therefore, through formally decomposing, we can transform this relation into two smaller ones.
Let A = all attributes of the relation Review
→RateSongs_1 (A - User_ID, Album.Name)
→**RateSongs_1 (Rater_User_ID, Owner_User_ID, Song_ID, Emoji, Stars, date)**
And,
→RateSongs_2 (Song_ID U (User_ID, Album.Name))
→RateSongs_2 (Song_ID, User_ID, Album.Name)
Although the relation Song already has such attributes, therefore RateSongs_2 is redundant and should not be created.

UPDATED SCHEMA

- Users(User_ID: int, Email: String, Name: String, Password: String Is_Artist: boolean);
- Basic(User_ID: int,DOB: Date);
- Artist(User_ID: int, Country: String, Year Founded: int, Genre: String, No. of Members: int, Most_Popular_Album_Name: String, Most_Popular_Song_Name: String);
- Song(Song_ID: int, User_ID: int, Album.Name: string, Name: String, Plays: int);
- Album(User_ID: int, Name: String, Release Date: Date, Genre: String, No_of_Songs: int);
- Admin(Admin_ID: int, AdminEmail: string, Name: string);
- Review(Review_ID: int, Song_ID: int, Admin_ID: int, Deleted: boolean, Reviewed: boolean, Flagged_Date: date, Review_Date: date);
- RateAlbums(Rate_Album_ID: int, Rater_User_ID: int, Owner_User_ID: int, Album.Name: string, Emoji: string, Stars: int, Rate_Date: Date);
- RateSongs(Rate_Song_ID: int, Rater_User_ID: int, Song_ID: int, Emoji: string, Stars: int, Rate_Date: Date);

2. SQL Implementation

what are all the rock albums released before 2010?

Query 1:

Select name from Album where genre="Rock" and Release_Date < "2010-01-01 00:00:00";

Output:

Madness
Rolling Hills
Garage

What are all the songs played more than 10 times on Kanye West's College Dropout album?

Query 2:

Select name from Song where plays > 10 and album_name = "College Dropout";

Output:

Gold Digger
Taylor is my friend
Slow Jamz

two tables (2)

How many songs does Death’s most popular album have?
Query 3:
Select No_of_Songs from Album JOIN Artist on
Artist.Most_Popular_Album_Name=Album.Name where Artist.User_ID=11;
Output:
3

What are the names of all of Katy Perry’s albums?
Query 4:
Select Album.Name from Album JOIN Artist on Album.User_ID=Artist.User_ID
where Artist.User_ID=19;
Output:
Spinning around

self join (1)
What are the top ten songs?
Query 5:
Select Song.Name from Song ORDER BY Song.Plays DESC LIMIT 10;
Output:
Dad is right
No more sleeping in
Watermelon soda (we lke that)
Stairway to Heaven
Over the Hills and Far Away
Kashmir
Who ate all the guacamole?
My Broom
Candy
I am dumb

Union, Except or Intersect (2)
Name artists that are from the US and were founded before 2000
Query 6:
Select Users.Name from Users, Artist where Artist.Country="USA" and
Users.User_ID=Artist.User_ID
INTERSECT

Select Users.Name from Users, Artist where Artist.Year_Founded < 2000 and Users.User_ID=Artist.User_ID;

Output:
Foo Fighters
death

What are the albums and songs that are rated 5 stars?

Query 7:
Select DISTINCT Album_Name from RateAlbums where stars=5 UNION Select Song.Name from Song, RateSongs where RateSongs.stars = 5 and Song.Song_ID=RateSongs.Song_ID;

Output:
Balling
I am famous
Loyalty
Madness
Welcome 2 my barn party

aggregation (3)

What is the average rating of all the songs?

Query 8:
```
Select AVG(Stars)
   FROM RateAlbums
   WHERE Owner_User_ID = 15;
```
Output:
4.0

How many songs does User 11 (death) have?

Query 9:
```
Select Count(*)
   FROM Song S
   Where S.User_ID=11;
```
Output:
3

What is the max song rating in the database?

Query 10:
```
Select Max(Stars)
   From RateSongs;
```

Output:
5

3. FDs and Normal Forms

3.1. No. B $\rightarrow$ C does not hold.
The second FD A $\rightarrow$ C shows that the FD AB $\rightarrow$ C is redundant. C can be obtained directly from A, and B is not required.

Student_Name	CS_Class	Class_Room
Yash	564	CS2120
Yash	540	PSY123

The above table shows that the FD {Student_Name, CS_Class} $\rightarrow$ Class_Room holds.
Also CS_Class $\rightarrow$ Class_Room holds.
But Student_Name $\rightarrow$ Class_Room does not hold or make sense.

3.2. The FD: AB $\rightarrow$ E breaks the requirements for BCNF in R.
Therefore, R can be decomposed into R_1(R-E) and R_2(AB U E)
R_1(ABCD), R_2(ABE)
The FD: A $\rightarrow$ C breaks the requirements for BCNF in R_1
Therefore, R_1 can be decomposed into R_3(R_1-C) and R_4(A U C)
R_3(ABC), R_4(AC)
All of these relations are in BCNF now, namely: R_2(ABE), R_3(ABC), R_4(AC)

3.3. $\{FD\}^+$ = {AB $\rightarrow$ C, AC $\rightarrow$ B, AD $\rightarrow$ E, BC $\rightarrow$ A, B$\rightarrow$F, AC $\rightarrow$ F} (not including trivial fds)

3.3.1. {AB, BC, ABDE, EF}

3.3.1.1. The intersection for all these relations is null, therefore this is not a lossless join

A	B
540	E Hall
524	Union South

This shows a lossy join.

3.3.1.2. Dependencies are not preserved in the decomposition because the union of all the closures of fds in the decomposition does not give the closure of fd in R.

E	F
564	CS
564	Union South

The FD B → F does not hold on this table. E is not dependent on F and vice versa.And F is not present in any table with B.

3.3.2. {ABCDE, BCDEF}

3.3.2.1. The intersection of the two relations gives one of these. That is (ABCDE intersection with BCDEF) → ABCDE
BCDE → BCDEF is true because (B → F) (the rest is a trivial fd)

3.3.2.2. The FDs of ABCDE = {AB → C, AC → B, AD → E, BC → A}
The FDs of BCDEF = {B → F}
The closure of these two is {AB → C, AC → B, AD → E, BC → A, B→F, AC → F} Therefore, this is dependency preserving.

3.3.3. {ABC, ADE, BF}

3.3.3.1. The intersection for all these relations is null, therefore this is not a lossless join

3.3.3.2. FDs of ABC = {AB → C, AC → B, BC → A}.
FDs of ADE = AD → E
FDs of BF = B → F
The closure of the union of all these FDs =
{AB → C, AC → B, AD → E, BC → A, B→F, AC → F} (not including trivial fds) Therefore, this is dependency preserving.

3.4. (bonus question)
A = relation in BCNF
B = relation in 3NF
Proving A → B
If a relation just has one key it has to be a superkey. One key also denotes just one non trivial fd. Therefore this relation is in BCNF. And all relations in BCNF are in 3NF.
Proving B → A
As mentioned earlier, if a relation is just has one key, that means the key is a superkey. One key also denotes one non trivial fd. And BCNF is just a subset of the 3NF with the 'superkey' restraint about non-trivial fds. This shows that this relation is in BCNF.