Exercise 2

Jasper Steinberg

February 17, 2025

1 SELECT Queries in SQL

We solve the task by using DB Browser.

1.1 a)

We run the query

```sql
SELECT songID, name, duration, year, artistID
FROM song;
```

and get the (cropped) output

	songID	name	duration	year	artistID
1	1	Saved	178	2015	1
2	2	Oops!... I Did It Again	221	2000	2
3	3	Don't Start Now	183	2019	3
4	4	Strangers	233	2017	4
5	5	I Went Too Far	294	2016	5
6	6	Blasé	286	2015	1
7	7	Hot Girl Summer	199	2019	9

1.2 b)

We run the query

```sql
SELECT name, year
FROM album
WHERE year < 2017;
```

and get the output

	name	year
1	Free TC	2015
2	Oops!... I Did It Again	2000
3	All My Demons Greeting Me as a …	2016
4	SremmLife 2	2016
5	ANTI	2016
6	I Am Not a Human Being II	2013

1.3 c)

I interpret the logical condition as $2018 < \text{year} < 2021$. The query is then

```sql
SELECT name, year
FROM album
WHERE year > 2017 AND year < 2021
ORDER BY year
```

which returns

	name	year
1	Scorpion	2018
2	Sucker Punch	2019
3	Fine Line	2019
4	thank u, next	2019
5	Future Nostalgia	2020
6	Positions	2020
7	Good News	2020
8	folklore	2020

1.4 d)

We need to merge such that the integrity of the relations are intact.

```sql
SELECT song.name AS songName, artist.name AS featuredName
FROM (featuredOn INNER JOIN song USING (songID))
        INNER JOIN artist ON (artist.artistID = featuredOn.artistID)
ORDER BY artist.name, song.name
```

which returns the (cropped) output

	songName	featuredName
1	Savage Remix	Beyoncé
2	I'm the One	Chance the Rapper
3	Love Me	Drake
4	Work	Drake
5	Saved	E-40
6	Blasé	Future
7	Don't Judge Me	Future
8	Love Me	Future
9	Black Beatles	Gucci Mane

1.5 e)

We merge merge and extract

```sql
SELECT song.name AS songName, album.name AS albumName, song.year AS releaseYear
FROM ((artist INNER JOIN song USING (artistID))
        INNER JOIN songOnAlbum USING (songID))
        INNER JOIN album USING(albumID)
WHERE artist.name = "Ariana Grande"
ORDER BY song.year, album.name, song.name
```

the query returns

	songName	albumName	releaseYear
1	thank u, next	thank u, next	2018
2	7 rings	thank u, next	2019
3	positions	Positions	2020

1.6 f)

We treat the cases of main artist and features artist separately, join by a union clause and then order. In the second part artist takes two different roles.

```sql
SELECT artist.name AS artistName, song.name AS songName
FROM song INNER JOIN artist USING (artistID)
WHERE artist.name = "Ty Dolla Sign"

UNION

SELECT mainArtist.name AS artistName, song.name AS songName
FROM featuredOn
INNER JOIN song USING (songID)
INNER JOIN artist AS featuredArtist ON (featuredArtist.artistID = featuredOn.artistID)
INNER JOIN artist AS mainArtist ON (mainArtist.artistID = song.artistID)
WHERE featuredArtist.name = "Ty Dolla Sign"

ORDER BY artistName, songName
```

the query returns

	artistName	songName
1	Megan Thee Stallion	Hot Girl Summer
2	Ty Dolla Sign	Blasé
3	Ty Dolla Sign	Don't Judge Me
4	Ty Dolla Sign	Love U Better
5	Ty Dolla Sign	Saved

1.7 g)

Extract the info and check for "the" as part of the song name.

```sql
SELECT artist.name AS artistName, song.name as songName
FROM song INNER JOIN artist ON (song.artistID = artist.artistID)
WHERE song.name LIKE "%the%"
```

the query returns

	artistName	songName
1	DJ Khaled	I'm the One

1.8 h)

We group and count, and then check against the maximum of the same group and count.

```
SELECT artist.name as artistName, COUNT(*) AS numberOfFeatures
FROM featuredOn
INNER JOIN artist ON (artist.artistID = featuredOn.artistID)
GROUP BY artist.name
HAVING numberOfFeatures = (
        SELECT MAX(featureCount) FROM (
        SELECT COUNT(*) AS featureCount
        FROM featuredOn
        GROUP BY artistID
        )
)
```

the query returns

	artistName	numberOfFeatures
1	Future	3

2 More Queries in Relational Algebra

2.1 a)

First we need to find out what song don't appear on an album. This will be the set difference between the songID's in song and the songID's in songOnAlbum

$$\text{songIDnotOnAlbum} = \Pi_{\text{songID}}(\text{song}) - \Pi_{\text{songID}}(\text{songOnAlbum}).$$

We can now use these ID's to filter for the desired songs with a selection-operator

$$\text{songNotOnAlbum} = \sigma_{\text{songID} \in \Pi_{\text{songID}}(\text{songIDnotOnAlbum})}(\text{song}).$$

Now we join with artist

$$\text{songNotOnAlbumWithArtist} = \text{artist} \bowtie_{\text{artistID}} \text{songNotOnAlbum}.$$

Now we project and rename

$$\text{artistAndSongName} = \Pi_{\text{artist.name,songNotOnAlbum.name}}(\text{songNotOnAlbumWithArtist})$$
$$\text{songsWithoutAlbums} = \rho_{\text{artistAndSongName(artist.name, songNotOnAlbum.name)}}(\text{artistAndSongName}).$$

2.2 b)

We first pick out the artists with songs from the oughts,

$$\text{oughts} = \Pi_{\text{artist.name, song.name}}\left(\sigma_{2000 \leq \text{year} \leq 2009}(\text{song}) \bowtie_{\text{artistID}} \text{artist}\right).$$

We select the songs from the current decade (which i guess is supposed to be the 2010s), merge with the merged sets of artist that start with B which feature on a song

$$\text{currentArtistsB} = \Pi_{\text{artist.name, song.name}}\left(\sigma_{2010 \leq \text{year} \leq 2019}(\text{song}) \bowtie_{\text{artistID}} \text{featuredOn} \bowtie_{\text{songID}} \sigma_{\text{name = "B\%"}}(\text{artist})\right).$$

The finall query will then be the union of the two

$$\text{final} = \text{oughts} \cup \text{currentArtistsB}$$

2.3 c)

We first join song and artist

$$\text{songArtist} = \text{song} \bowtie_{\text{artistID}} \text{artist}.$$

Then we aggregate with the count function on the grouping artist.name.

$$\text{songArtistCount} = \gamma_{\text{artist.name, COUNT(*)}\rightarrow\text{numberOfSongs}}(\text{songArtist}).$$

Then we can order according to numberOfSongs

$$\text{songArtistCountOrderd} = \tau_{\text{numberOfSongs DESC}}(\text{songArtistCount}).$$

Finally we project the desired information

$$\text{final} = \Pi_{\text{artist.name, song.name}}(\text{songArtistCountOrderd}).$$

3 Introduction to Database Normalization

3.1 a)

We would have to update 4 birth year cells and 4 director name cells. This would mean a total of 8 updates for the given changes.

3.2 b)

We should split the table, a logical way to do it is to split into a film and a director table. This would look like

$$\text{Film}(\underline{\text{FilmID}}, \text{Name}, \text{Year}, \text{DirectorID})$$
$$\text{Director}(\underline{\text{DirectorID}}, \text{DirectorName}, \text{DirectorBirthYear})$$

The functional relations are preserved in each table, and we store the information of the director through a relationship instead of directly.

4 Functional Dependencies, Keys and Closures

4.1 a)

Task	Validity	Explanation
1	True	Trivial relation
2	False	a_3 has two different values b_3 and b_4
3	Unknown	No conflicting values
4	Unknown	No conflicting values
5	False	c_1 corresponds to both d_1 and d_2
6	False	Lots of conflicting values
7	True	Trivial superkey
8	False	The first two rows show that ABC dose not determine D
9	False	D dose not determine C, so it cannot be a candidate key
10	Unknown	All ABD rows are unique, so no way to disprove

Table 1: Caption

4.2 b)

We find the atomic relations of F. We have $D \rightarrow A$ and $B \rightarrow D$, and thus we also have $B \rightarrow A$. We can also use these relations to simplify $ABD \rightarrow C$. Since B determines both A and D, we simply have $B \rightarrow C$. With these relations in mind, $D^+ = \{D, A\}$. For BC^+ we at least know they determine B and C. Since B also determines D and A, we know that BC^+ determines all letters, and thus the entire table, i.e. $BC^+ = ABCD^+$. The same reasoning leads to $AB^+ = BD^+ = ABCD^+$. We see from the atomic relations that B is a candidate key for the table. No other letters can determine the whole table alone, so this is the only one. It is also trivially minimal.