Discussion 1

Intro, Relations, SQL

Agenda

I. Welcome!
II. Getting started
III. Sets, Relations, Tables
IV. SQL

Welcome!

This is a online course (online Zoom lectures, sections, and office hours)

Course website: http://www.cs186berkeley.net/

Projects are involved coding assignments

- **Project 0 is due Thurs 9/3!** (Not graded but is **mandatory**)
- **Project 1 is assigned (due next week, Thurs 9/10)!**

Vitamins are **required** weekly quizzes that keep you up to date with lectures

- **Vitamin 1** due this ~~Friday~~ Sunday at 11:59PM

Pre-semester Quiz due Friday at 11:59 PM

Midterms will be **Week 7** and **Week 11** (TBD)

Accounts

Piazza: http://piazza.com/berkeley/fall2020/cs186

Gradescope, bCourses: Automatic enrollment for enrolled students

GitHub: See Project 0 for GitHub Classroom setup

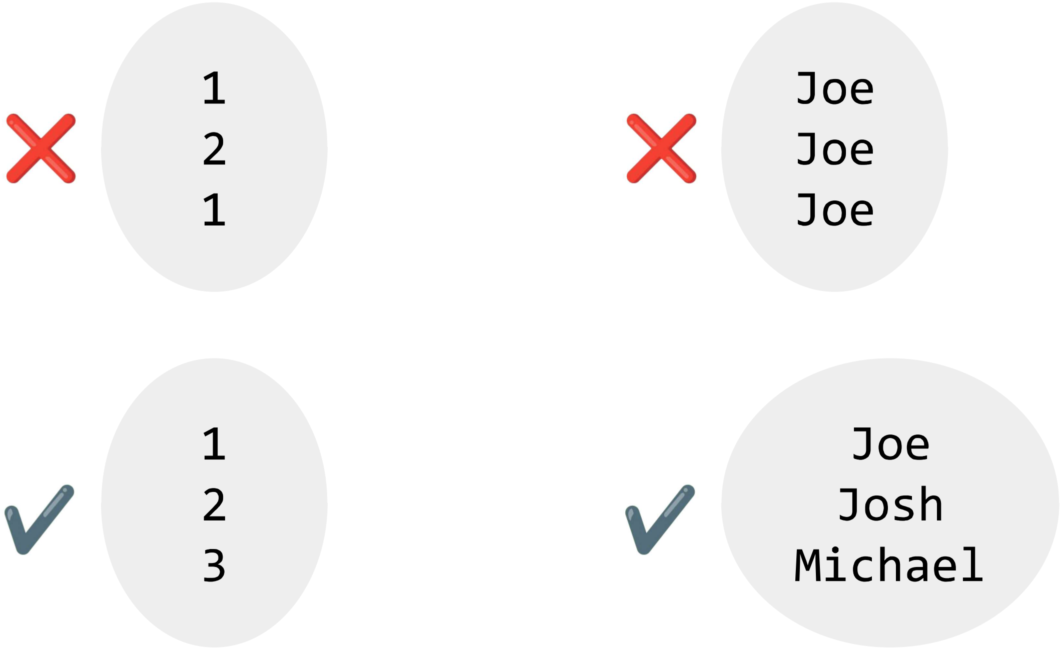
Email cs186berkeley.issues@gmail.com if you have any issues

Questions?

Sets, Relations, and Tables

What are sets?

A **set** is a well-defined collection of distinct objects.



❌ 1 2 1

❌ Joe Joe Joe

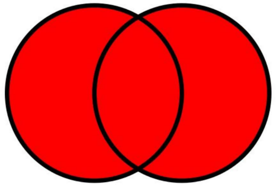
✓ 1 2 3

✓ Joe Josh Michael

Set operations

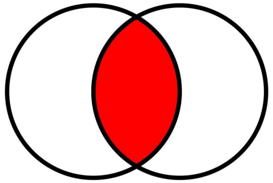
The **union** of sets X and Y is the collection of all elements in **either** set.

→ {1, 2, 3} ∪ {3, 4, 5} = {1, 2, 3, 4, 5}



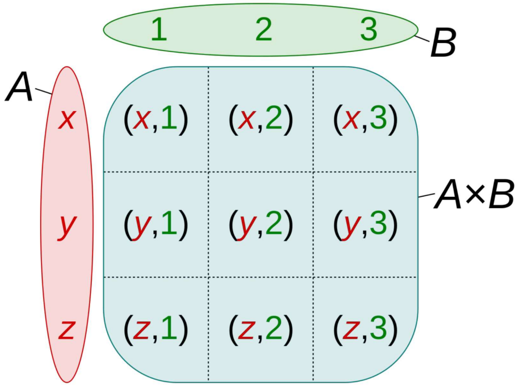
The **intersection** of sets X and Y is the collection of all elements that exists in **both** sets.

→ {1, 2, 3} ∩ {3, 4, 5} = {3}



The **cartesian product** of sets X and Y is the collection of all possible ordered pairs (x, y) where x is in X and y is in Y.

→ {1, 2} × {3, 4} = {(1, 3), (1, 4), (2, 3), (2, 4)}



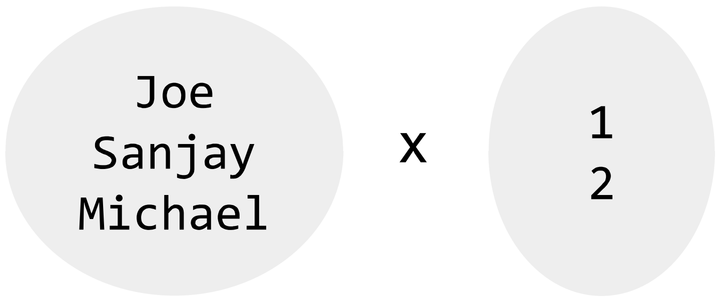
Relations and tables

Sets and *relations* are mathematical concepts

➜ A **set** is a well-defined collection of distinct objects

➜ An ***n*-ary relation** on n sets is any subset of Cartesian products of the sets

A SQL **table** with n columns is an n-ary relation.



Joe	1
Joe	2
Sanjay	1
Sanjay	2
Michael	1
Michael	2

cartesian product

name	num
Joe	1
Sanjay	1
Michael	1

one possible ***relation***

SQL

SQL for single tables queries

```
SELECT [DISTINCT] <column list>
FROM <table1>
[WHERE <predicate>]
[GROUP BY <column list>]
[HAVING <predicate>]
[ORDER BY <column list> [DESC/ASC]]
[LIMIT <amount>];
```

- SQL is declarative - you describe *what* you want in the output, and the DBMS decides *how* it's fetched.

Logical Processing Order

1. `FROM <table1>` - which table are we drawing data **from**
2. `[WHERE <predicate>]` - only keep rows **where** `<predicate>` is satisfied
3. `[GROUP BY <column list>]` - **group** together rows **by** value of columns in `<column list>`
4. `[HAVING <predicate>]` - only keep groups **having** `<predicate>` satisfied
5. `SELECT <column list>` - **select** columns in `<column list>` to keep
    a. `[DISTINCT]` - keep only **distinct** rows (filter out duplicates)
6. `[ORDER BY <column list> [DESC/ASC]]` - **order** the output **by** value of the columns in `<column list>`, `ASC`ending by default
7. `[LIMIT <amount>]` - **limit** the output to just the first `<amount>` rows

- DBMS may execute a query in an equivalent but *different* order
- For multi-table queries: perform joins with `FROM`

Logical Processing Order

```
FROM test_table
```

test_table

a	b
1	0
2	1
3	0
4	1
5	0
6	1
7	2
8	3

Take data `FROM test_table`.

Logical Processing Order

```
FROM test_table
WHERE a > 2
```

test_table

a	b
1	0
2	1
3	0
4	1
5	0
6	1
7	2
8	3

a	b
3	0
4	1
5	0
6	1
7	2
8	3

Take data `FROM test_table` and keep rows `WHERE a > 2`.

Logical Processing Order

```
FROM test_table
WHERE a > 2
GROUP BY b
```

a	b
3	0
4	1
5	0
6	1
7	2
8	3

a	b
3, 5	0
4, 6	1
7	2
8	3

We `GROUP BY b`, but there may be multiple values of `a` per group, so we can't use `a` directly anymore. We can, however, use it with *aggregate functions* (`MIN`, `MAX`, `SUM`, `AVERAGE`, `COUNT`). Using aggregate functions without a `GROUP BY` clause = everything in one group.

Logical Processing Order

```
  FROM test_table
   WHERE a > 2
    GROUP BY b
HAVING COUNT(*) >= 2
```

a	**b**
3, 5	0
4, 6	1
7	2
8	3

a	**b**
3, 5	0
4, 6	1

Throw away groups that have fewer than 2 rows in them.

Logical Processing Order

```
SELECT     b AS c
  FROM test_table
   WHERE a > 2
   GROUP BY b
HAVING COUNT(*) >= 2
```

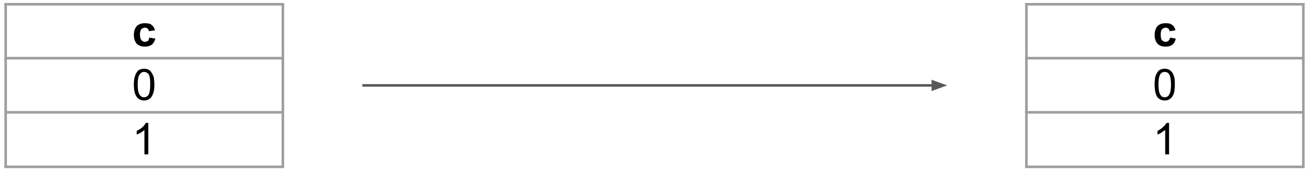
a	b
3, 5	0
4, 6	1

c
0
1

We use an *alias* here: `b AS c` selects `b`, but then calls it `c` afterwards. We can use this alias in any step *after* this one (so not in `WHERE`, `GROUP BY`, `HAVING`).

Logical Processing Order

```
SELECT DISTINCT b AS c
FROM test_table
WHERE a > 2
GROUP BY b
HAVING COUNT(*) >= 2
```



c
0
1

c
0
1

No duplicates here, but if there were any, they would be removed. Duplicates are removed by exact match *on the entire row at this point*.

Logical Processing Order

```
SELECT DISTINCT b AS c
FROM test_table
WHERE a > 2
GROUP BY b
HAVING COUNT(*) >= 2
ORDER BY c DESC
```

c
0
1

→

c
1
0

Sort the output by the columns (just `c` here) (numerically for integers, lexicographically for strings) in either `ASC`ending (low to high) or `DESC`ending (high to low) order. Default is `ASC`.

Note: order of output is *not* guaranteed unless you have an `ORDER BY` clause.

Logical Processing Order

```
SELECT DISTINCT b AS c
FROM test_table
WHERE a > 2
GROUP BY b
HAVING COUNT(*) >= 2
ORDER BY c DESC
LIMIT 1
```

c
1
0

c
1

Return just the first row.

A Note On GROUP BY

- Consider the following **Classes** table and
  - Applying the query: **SELECT <*TBD*> FROM Classes GROUP BY Dept;**
  - This is what the table looks like once we perform the GROUP BY

Dept	**Course**	**Capacity**
CS	186	715
CS	161	485
EE	16B	580
EE	105	40
DATA	100	1375

Dept	**Course**	**Capacity**
CS	186	715
CS	161	485
EE	16B	580
EE	105	40
DATA	100	1375

A Note On GROUP BY

- The role of the SELECT statement is to **squash** each group into a **single row**
  - In general, SELECT clauses are applied to ***each row***, but when grouping is involved they are applied to ***each group***

Are the following valid?

Dept	**Course**	**Capacity**
CS	186	715
CS	161	485
EE	16B	580
EE	105	40
DATA	100	1375

SELECT Dept, Course FROM Classes GROUP BY Dept;

Dept	**Course**
CS	**?**
EE	**?**
DATA	**?**

NO

SELECT Dept, SUM(Capacity) FROM Classes GROUP BY Dept;

Dept	**SUM(Cap.)**
CS	1200
EE	620
DATA	1375

YES

String Comparison

`LIKE`: following expression follows SQL specified format

- **_: Any single character**
- **%: Zero, one, or more characters**
- for a *perfect* string match

Examples:

- LIKE ‘z%’ starts with z
- LIKE ‘z_’ exactly 2 letters, 1st is z
- LIKE ‘_z%’ 2nd letter is a z

~ : following expression follows regex format

- **. : Any single character**
- *** : Zero, one, or more of the character preceding the symbol**
- ^: Match at start of string (If used outside `[]`)
- Looks for *any* pattern in the string that fits

Examples:

- ~ ‘z.*’ *contains* z
- ~ ‘^z.*’ starts with z

Note: ~ cannot be used in SQLite (which Project 1 will be using)

Practice: Single-Table Queries

Question 1

Find the names of 5 songs that spent the least weeks in the top 40, ordered from least to most. Break ties by song name in alphabetical order.

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 1

Find the names of 5 songs that spent the least weeks in the top 40, ordered from least to most. Break ties by song name in alphabetical order.

```
SELECT song_name
FROM songs
ORDER BY weeks_in_top_40 ASC,
song_name ASC
LIMIT 5
```

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 2

Find the name and first year active of every artist whose name starts with the letter ‘B’.

Tables

Songs
(**song_id**, song_name, album_id,
weeks_in_top_40)

Artists
(**artist_id**, artist_name,
first_year_active)

Albums
(**album_id**, album_name, artist_id,
year_released, genre)

Question 2

Find the name and first year active of every artist whose name starts with the letter ‘B’.

```
SELECT artist_name,
first_year_active
FROM artists
WHERE artist_name LIKE 'B%'
```

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 2

Find the name and first year active of every artist whose name starts with the letter ‘B’.

```
SELECT artist_name,
first_year_active
FROM artists
WHERE artist_name ~ '^B.*'
```

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 3

Find the total number of albums released per genre.

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 3

Find the total number of albums released per genre.

```
SELECT genre, COUNT(album_id)

FROM Albums

GROUP BY genre;
```

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 4

Find the total number of albums released per genre. Don’t include genres with a count less than 10.

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 4

Find the total number of albums released per genre. Don't include genres with a count less than 10.

```
SELECT genre, COUNT(*)
FROM Albums
GROUP BY genre
HAVING COUNT(*) >= 10;
```

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 5

Find the most popular album genre that is released in the year 2000. Assume there are no ties.

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 5

Find the most popular album genre that is released in the year 2000. Assume there are no ties.

```
SELECT genre
FROM albums
WHERE year_released = 2000
GROUP BY genre
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Tables

Songs
(song_id, song_name, album_id,
weeks_in_top_40)

Artists
(artist_id, artist_name,
first_year_active)

Albums
(album_id, album_name, artist_id,
year_released, genre)

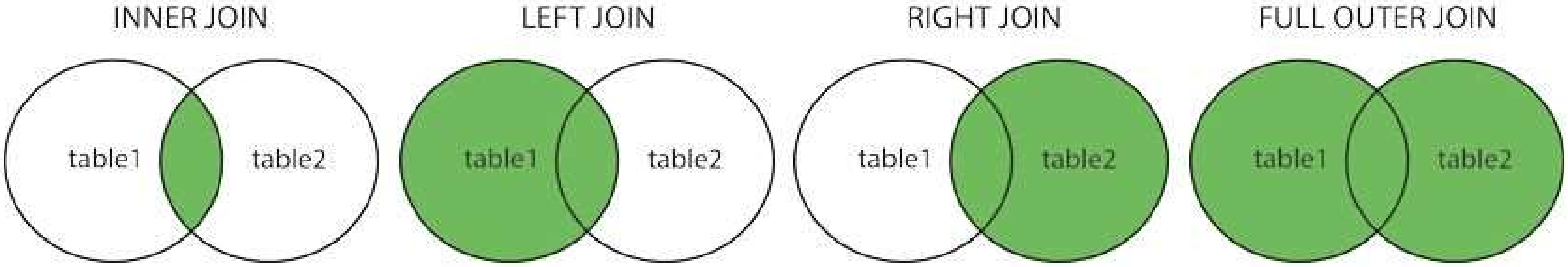
SQL Joins

Join Variants

- The different types of joins determine what we do with rows that don't ever match the "join condition"

```
SELECT * FROM
T1 INNER JOIN T2
ON T1.a = T2.a;
```

Join Condition



Inner Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	Age	Year
Brian	20	Junior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	Age	Year
Brian	20	Junior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	Age	Year
Brian	20	Junior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	Age	Year
Brian	20	Junior
Kimberly	22	Freshman

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior
Kimberly	22	Freshman

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	Age	Year
Brian	20	Junior
Kimberly	22	Freshman

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	Age	Year
Brian	20	Junior
Kimberly	22	Freshman
Kimberly	18	Freshman

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	Age	Year
Brian	20	Junior
Kimberly	22	Freshman
Kimberly	18	Freshman

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Inner Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior
Kimberly	22	Freshman
Kimberly	18	Freshman

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages INNER JOIN standing
ON ages.Name = standing.Name;
```

Left Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages LEFT JOIN standing
ON ages.Name = standing.Name;
```

Left Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior
Kimberly	22	Freshman
Kimberly	18	Freshman
Lakshya	22	**null**

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages LEFT JOIN standing
ON ages.Name = standing.Name;
```

Right Join, Example

Ages

Name	Age
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	Year
Brian	Junior
Kimberly	Freshman
Ben	Senior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages RIGHT JOIN standing
ON ages.Name = standing.Name;
```

Right Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	22
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior
Kimberly	22	Freshman
Kimberly	18	Freshman
null	**null**	Senior

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages RIGHT JOIN standing
ON ages.Name = standing.Name;
```

Full Outer Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	21
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

```sql
SELECT ages.Name, ages.Age, standing.Year
FROM ages FULL JOIN standing
ON ages.Name = standing.Name;
```

Full Outer Join, Example

Ages

Name	**Age**
Brian	20
Lakshya	21
Kimberly	22
Kimberly	18

Standing

Name	**Year**
Brian	Junior
Kimberly	Freshman
Ben	Senior

Result

Name	**Age**	**Year**
Brian	20	Junior
Kimberly	22	Freshman
Kimberly	18	Freshman
null	**null**	Senior
Lakshya	21	**null**

```
SELECT ages.Name, ages.Age, standing.Year
FROM ages FULL JOIN standing
ON ages.Name = standing.Name;
```

Practice: Multi-Table Joins

Question 1

Find the names of all artists who released a ‘country’ genre album in 2020.

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 1

Find the names of all artists who released a ‘country’ genre album in 2020.

```
SELECT artist_name
FROM Artists AS A
INNER JOIN Albums AS B
ON A.artist_ID = B.artist_id
WHERE genre = ’country’ AND
year_released = 2020
GROUP BY A.artist_id,
artist_name;
```

Tables

Songs
(**song_id**, song_name, album_id,
weeks_in_top_40)

Artists
(**artist_id**, artist_name,
first_year_active)

Albums
(**album_id**, album_name, artist_id,
year_released, genre)

Question 2

Find the name of the album with the song that spend the most weeks in the top 40. Assume there is only one such song.

Tables

Songs
(**song_id**, song_name, album_id,
weeks_in_top_40)

Artists
(**artist_id**, artist_name,
first_year_active)

Albums
(**album_id**, album_name, artist_id,
year_released, genre)

Question 2

Find the name of the album with the song that spend the most weeks in the top 40. Assume there is only one such song.

```
SELECT album_name
FROM Albums AS A INNER JOIN
Songs AS S
ON A.album_id = S.album_id
ORDER BY weeks_in_top_40 DESC
LIMIT 1;
```

Tables

Songs
(**song_id**, song_name, album_id,
weeks_in_top_40)

Artists
(**artist_id**, artist_name,
first_year_active)

Albums
(**album_id**, album_name, artist_id,
year_released, genre)

Question 3

Find the the artist name and the most weeks one of their songs spent in the top 40 for each artist. Include artists that have not released an album.

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Question 3

Find the the artist name and the most weeks one of their songs spent in the top 40 for each artist. Include artists that have not released an album.

```
SELECT artist_name, MAX(weeks_in_top_40)
FROM Artists LEFT JOIN
(Songs INNER JOIN Albums ON
Songs.album_id = Albums.album_id)
ON Artists.artist_id = Albums.artist_id
GROUP BY Artists.artist_id, artist_name;
```

Tables

Songs
(**song_id**, song_name, album_id, weeks_in_top_40)

Artists
(**artist_id**, artist_name, first_year_active)

Albums
(**album_id**, album_name, artist_id, year_released, genre)

Appendix (Tips and Tricks)

- Using an aggregate in **WHERE** is **not** allowed
  - WHERE **count(*)** > **500** is an **invalid** query!
- Don’t use **HAVING** without **GROUP BY**
  - Just use **WHERE** instead!
- If **GROUP BY** is used, you can only **SELECT** columns that are **aggregates** OR are columns used to **group**
- **DISTINCT** removes all **duplicate rows**
  - Watch out for order of operators + logical processing order!
  - If you want to see how many distinct values there are of a column, **DISTINCT COUNT(X)** will not work; use **COUNT(DISTINCT X)**