Week 5

Accessing Multiple Tables

Storing More Data

- So far our in class examples have used a single movie table
  - What other data could we store related to movies?

movie_id	title	runtime	genre	release_year
1	The Banshees of Inisherin	109	Drama	2022
2	The Truman Show	107	Drama	1998
3	Eternal Sunshine of the Spotless Mind	108	Romance	2004
4	The Dark Knight	152	Action	2008
5	The Grand Budapest Hotel	99	Comedy	2014
6	Spider-Man: Into the Spider-Verse	116	Animation	2018
7	Shrek	89	Animation	2001
8	Spirited Away	125	Animation	2001

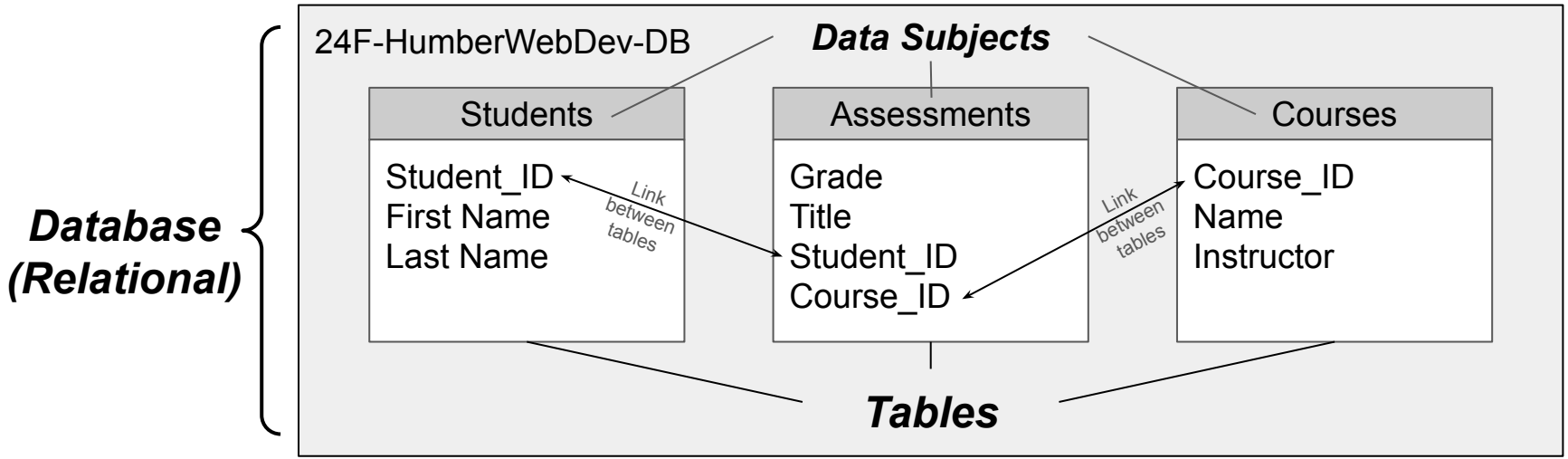
- Expanding this table indefinitely would quickly end up storing duplicated data, leading to data integrity issues

Multiple Tables

- Splitting up data into multiple tables to remove duplication and improve data integrity is known as **normalization**
- Normalization is a deep topic, and there are complex sets of rules which dictate the extent to which a database is judged to be normalized
- For now these are the 2 most important notes:
  - The reason for normalization is to reduce data redundancy and improve data integrity
  - The mechanism for carrying out normalization is arranging data in multiple tables and defining relationships between them (AKA Relational Databases!)

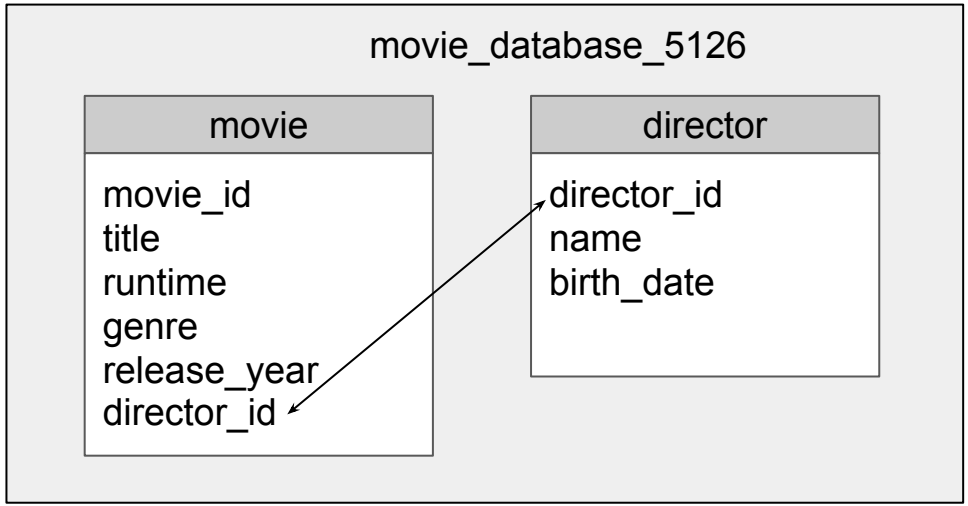
Relational Databases Summary

- **Databases** hold **Tables**
- **Tables** hold pieces of **Data** about a single **subject**
- **Relational Databases** allow **Tables to connect (relate) to each other** to each other



Database Design

- To create Relational Databases we need to complete 2 steps
  - Define our entities (Tables)
  - Connect them together
- An entity represents a real world **object, subject, or concept**
  - Basically a set of data that we want to model within our database



This can be defined as a **one-to-many relationship**

- As in 1 director can belong to many movies

*More on this next week

SQL & Multiple Tables

- SQL can access multiple related tables in a single query using **JOIN** and **ON**
- **JOIN** and **ON** allow 2 tables to output data into a single output table

title	director_id
The Banshees of Inisherin	1
The Truman Show	2

JOIN

director_id	name
1	Martin McDonagh
2	Peter Weir

Connects rows together by the related id columns

ON director_id = director_id

title	director_id	director_id	name
The Banshees of Inisherin	1	1	Martin McDonagh
The Truman Show	2	2	Peter Weir

SQL JOINs[1] and ON[2]

- A **JOIN** is used to combine rows from two or more tables
- **ON** is used to define the connection of the 2 tables, using the related column between them

title	director_id
The Banshees of Inisherin	1
The Truman Show	2

JOIN

director_id	name
1	Martin McDonagh
2	Peter Weir

```
SELECT movie.movie_id, movie.director_id, director.name
    FROM movie INNER JOIN director
    ON movie.director_id = director.director_id;
```

[1] https://www.w3schools.com/mysql/mysql_join.asp
[2] https://mode.com/sql-tutorial/sql-joins

ON

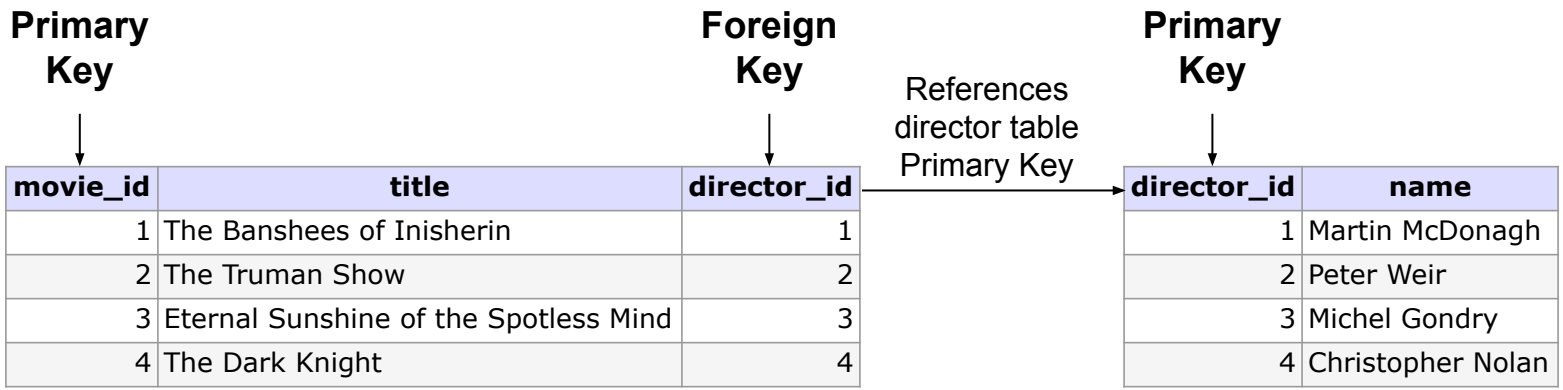
```
SELECT * FROM movie
    INNER JOIN director
    ON movie.director_id = director.director_id;
```

- **ON** indicates how the two tables, *movie* and *director*, relate to each other
- So the query above means:
  - Join the rows from both tables together where the director_id field in the movie table is equal to the director_id field from the director table
- Note: The **SELECT** portion of the statement picks the columns for both tables
- The 2 columns that map to one another, *director_id*, are called "keys"

Keys

There are many ‘Keys’ in SQL, the 2 main keys are:

- Primary Key: a column in a table, that can uniquely identify a row in the table
- Foreign Key: a column in one table that refers to the PRIMARY KEY in another table



movie_id	title	director_id
1	The Banshees of Inisherin	1
2	The Truman Show	2
3	Eternal Sunshine of the Spotless Mind	3
4	The Dark Knight	4

director_id	name
1	Martin McDonagh
2	Peter Weir
3	Michel Gondry
4	Christopher Nolan

INNER JOIN[1]

- **INNER JOIN** selects records that have matching "keys" in both tables
  - Example: Show the movie title and the name of the director for the movie

```
SELECT movie.title, director.name FROM movie
    INNER JOIN director ON movie.director_id = director.director_id;
```

- Query returns only rows from both tables where the director ids match
- **INNER JOIN**: The intersection of 2 tables where the intersection is made of rows that satisfy the **ON** condition

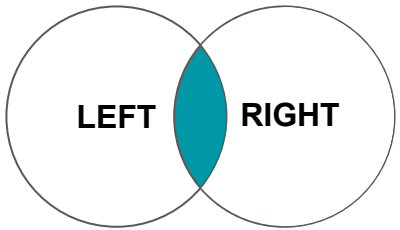
left	id
data	1
data	2

+

id	right
1	data
3	data

=

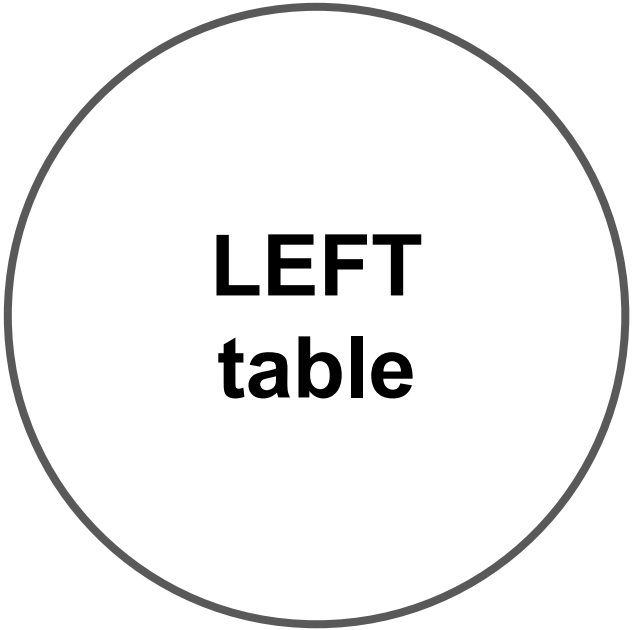
left	id	id	right
data	1	1	data

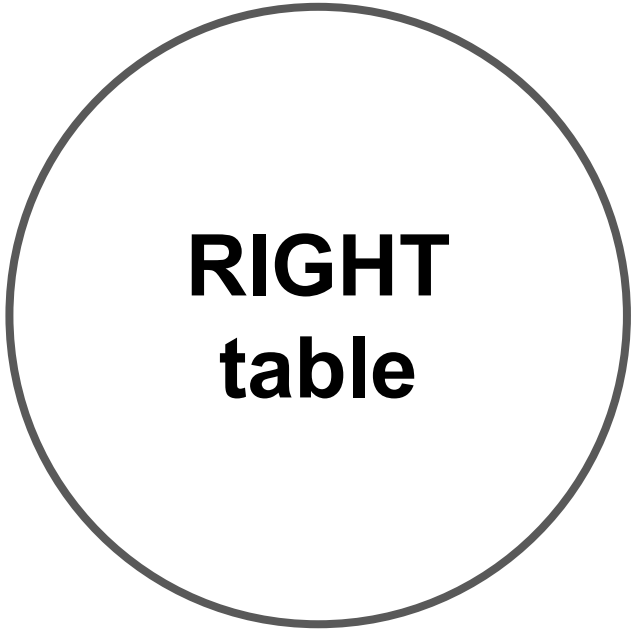


[1] https://www.w3schools.com/mysql/mysql_join_inner.asp

left	id
data	1
data	2

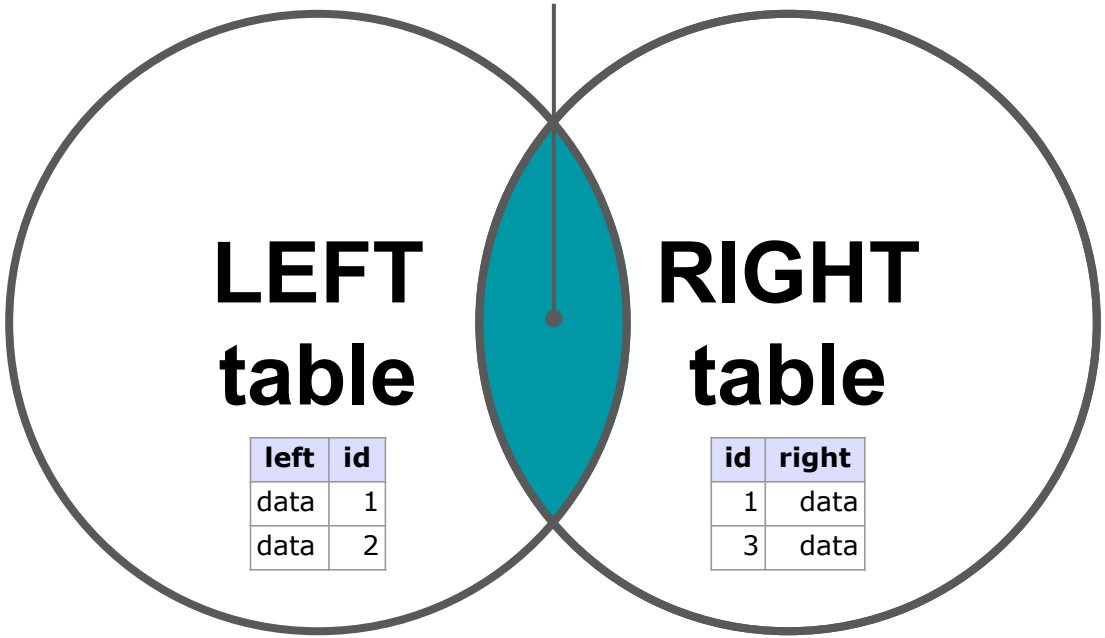
id	right
1	data
3	data





left	id	id	right
data	1	1	data

Joined table rows
where “keys” match



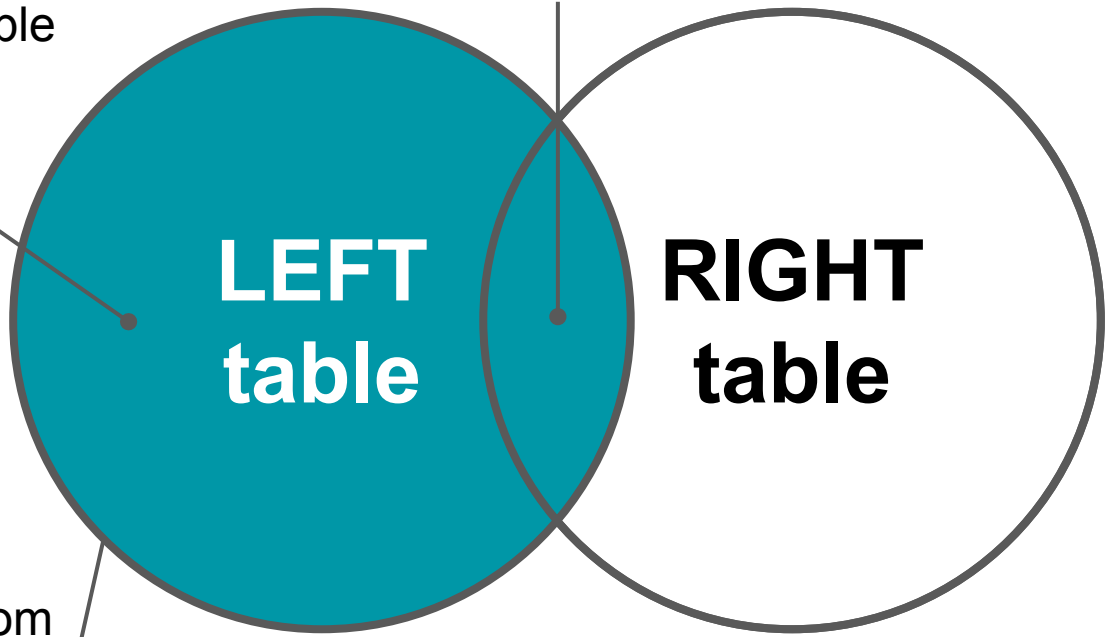
INNER JOIN

left	id	id	right
data	1	1	data

Joined table rows where “keys” match

left	id	id	right
data	2	*NULL*	*NULL*

Joined table rows **without** right table “keys” match



left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*

All rows of data from left table + right data if it exists

LEFT JOIN

LEFT JOIN[1]

- **LEFT JOIN** keyword returns all records from the left table, and the records that fulfill the **ON** condition from the right table
- If there is no matching value from right table, *NULL* fills the rows cells
  - Example: “Show me all the movies, and show me all the directors for the movies, if they exist”

```
SELECT movie.title, director.name FROM movie
    LEFT JOIN director ON movie.director_id=director.director_id;
```

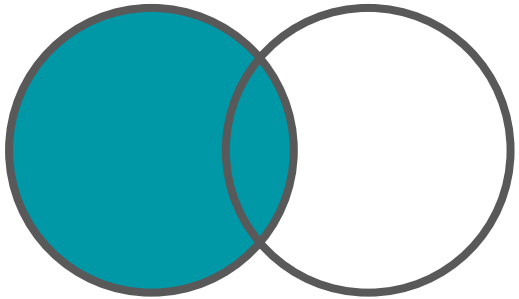
left	id
data	1
data	2

+

id	right
1	data
~~3~~	~~data~~

=

left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*



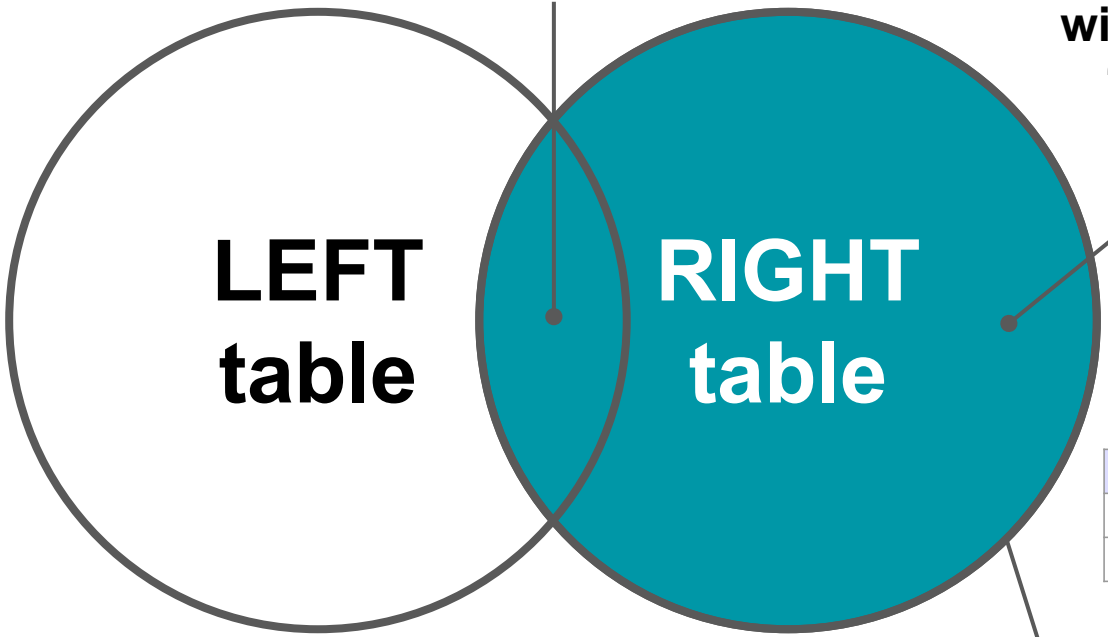
[1] https://www.w3schools.com/mysql/mysql_join_left.asp

left	id	id	right
data	1	1	data

Joined table rows where “keys” match

left	id	id	right
NULL	*NULL*	3	data

Joined table rows **without** left table “keys” match



left	id	id	right
data	1	1	data
NULL	*NULL*	3	data

All rows of data from right table + left data if it exists

RIGHT JOIN

RIGHT JOIN[1]

- **RIGHT JOIN** keyword returns all records from the right table, and the records that fulfill the **ON** condition from the left table
- If there is no matching value from left table, *NULL* fills the rows cells
  - Example: “Show me all the directors, and all the movies they directed, if they exist”

```
SELECT movie.title, director.name FROM movie
    RIGHT JOIN director ON movie.director_id=director.director_id;
```

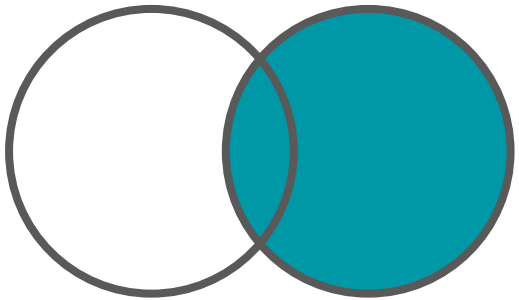
left	id
data	1
~~data~~	~~2~~

+

id	right
1	data
3	data

=

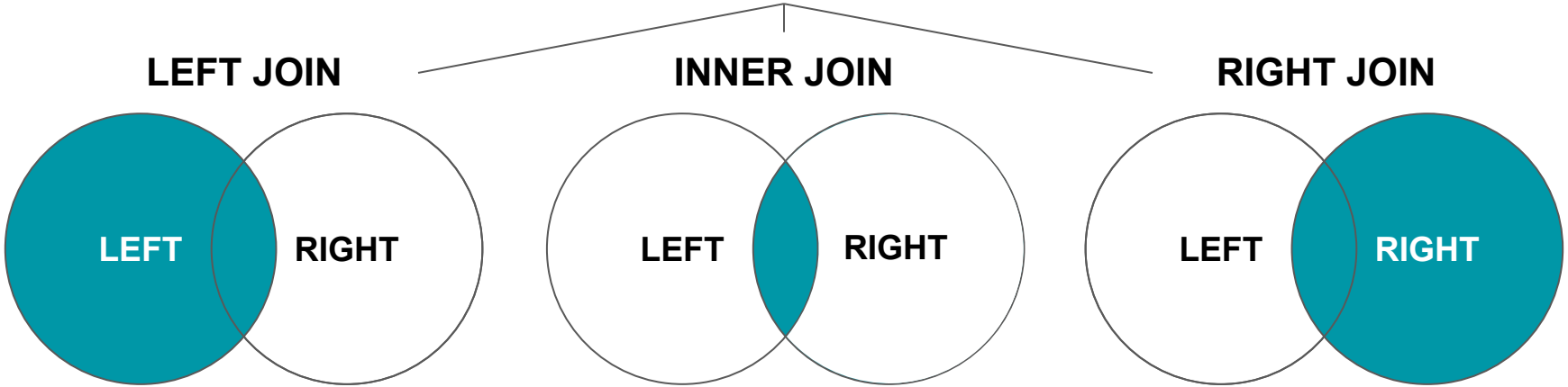
id	right	left	id
1	data	data	1
3	data	*NULL*	*NULL*



[1] https://www.w3schools.com/mysql/mysql_join_right.asp

JOINS

3 Main Joins
in MySQL



UNION[1]

- **UNION** operator combines the results of two or more SELECT statements
- **UNION** runs queries in order, then stacks results on top of each other
  - Every **SELECT** statement within **UNION** **MUST HAVE** the same number of columns
  - The columns **MUST HAVE** the same order and data types (*MySQL is not strict on this)

column_1	column_2
q1_c1_output	q1_c2_output

UNION

column_1	column_2
q2_c1_output	q2_c2_output

↓

column_1	column_2
q1_c1_output	q1_c2_output
q2_c1_output	q2_c2_output

- **Note: UNION will eliminate exact duplicates**
  - **Use UNION ALL to keep duplicates**

[1] https://www.w3schools.com/mysql/mysql_union.asp

left	id	id	right
data	1	1	data

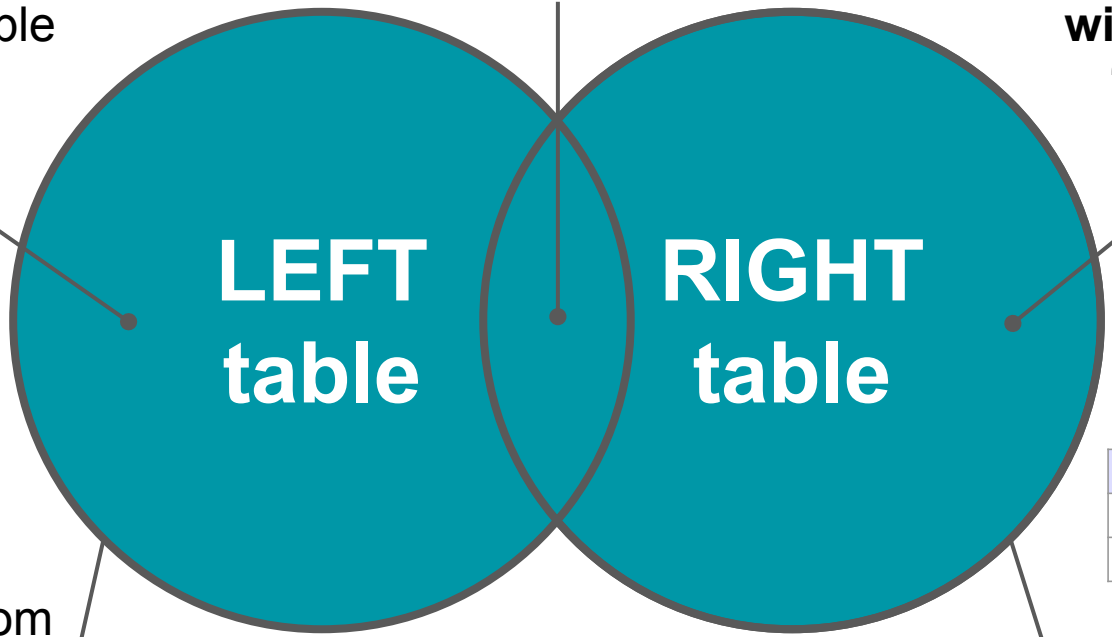
Joined table rows where “keys” match

left	id	id	right
data	2	*NULL*	*NULL*

Joined table rows **without** right table “keys” match

left	id	id	right
NULL	*NULL*	3	data

Joined table rows **without** left table “keys” match



left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*

All rows of data from left table + right data if it exists

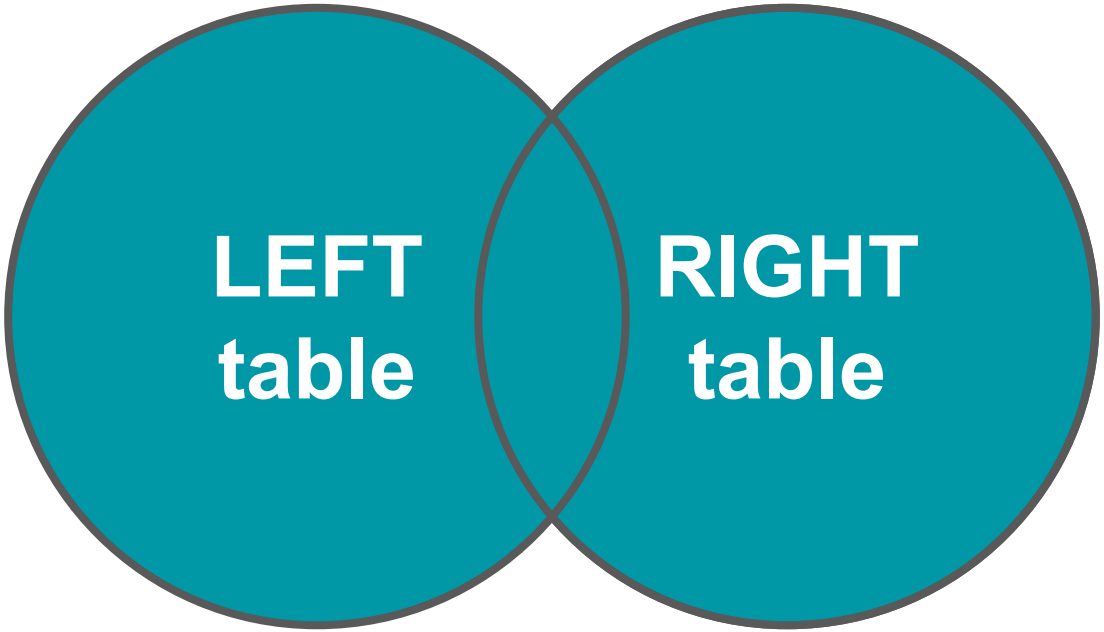
left	id	id	right
data	1	1	data
NULL	*NULL*	3	data

All rows of data from right table + left data if it exists

FULL JOIN

left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*
NULL	*NULL*	3	data

Joined table rows where “keys” match or do not match



FULL JOIN

FULL JOIN[1]

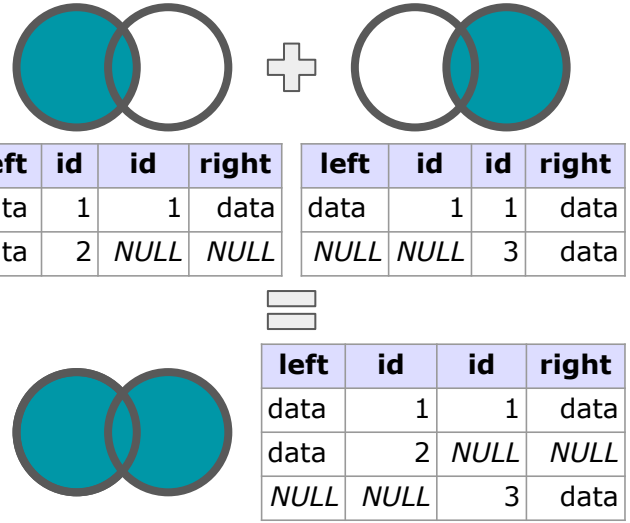
- **FULL JOIN** is both a **LEFT** and **RIGHT JOIN** in 1 query
  - returns all matching records from both tables whether the other table “key” matches or not
- **FULL JOIN** does not exist in MySQL but does in other DBMSs
  - In MySQL we can use **LEFT/RIGHT JOIN** to create a **FULL JOIN**
- **LEFT JOIN** + **RIGHT JOIN**:

```
SELECT * FROM movie LEFT JOIN director
     ON movie.director_id=director.director_id

     UNION

     SELECT * FROM movie RIGHT JOIN director
     ON movie.director_id=director.director_id;
```

- *UNION removes the duplicate rows from the intersection



left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*

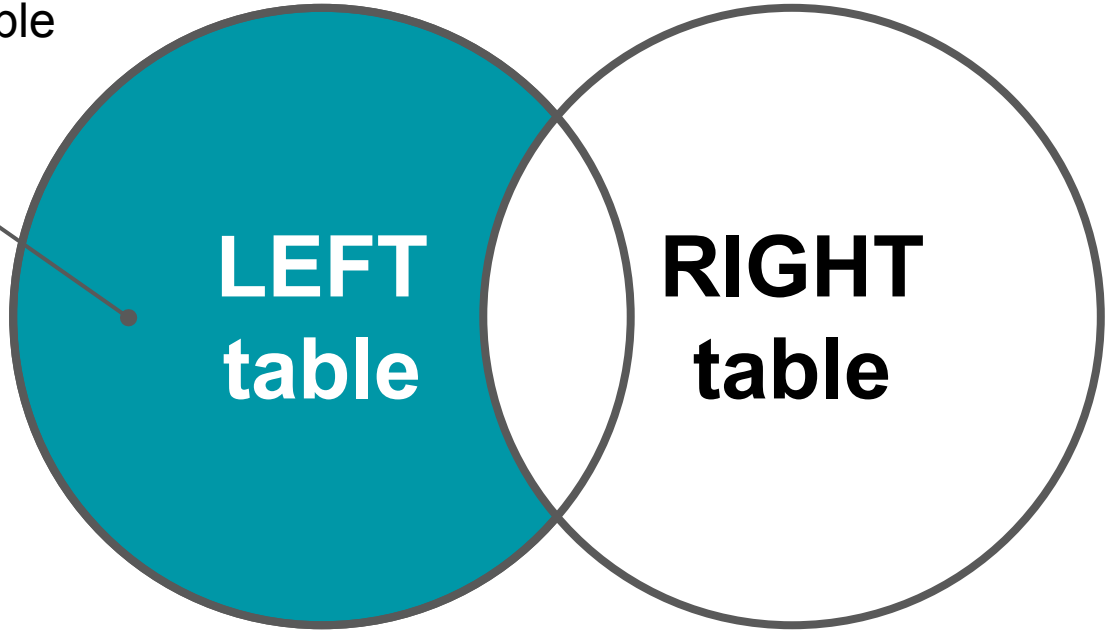
left	id	id	right
data	1	1	data
NULL	*NULL*	3	data

left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*
NULL	*NULL*	3	data

[1] https://www.w3schools.com/sql/sql_join_full.asp

left	id	id	right
data	2	*NULL*	*NULL*

Joined table rows **without** right table “keys” match

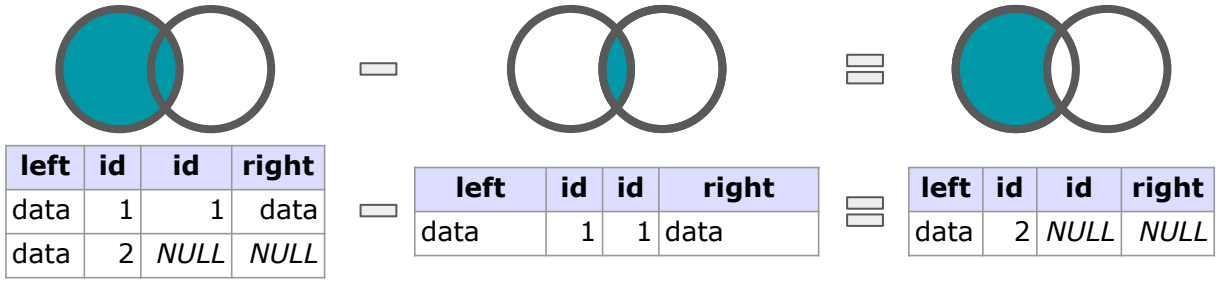


EXCLUSIVE LEFT JOIN

Exclusive LEFT JOIN

- Exclusive **`LEFT JOIN`** returns all records from the left table, but **NOT** the records that fulfill the **`ON`** condition from the right table
  - Example: “Show me all the movies, *without* directors assigned to them”

```
SELECT movie.title, director.name FROM movie
    LEFT JOIN director ON movie.director_id=director.director_id
    WHERE director.director_id IS NULL;
```



left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*

−

left	id	id	right
data	1	1	data

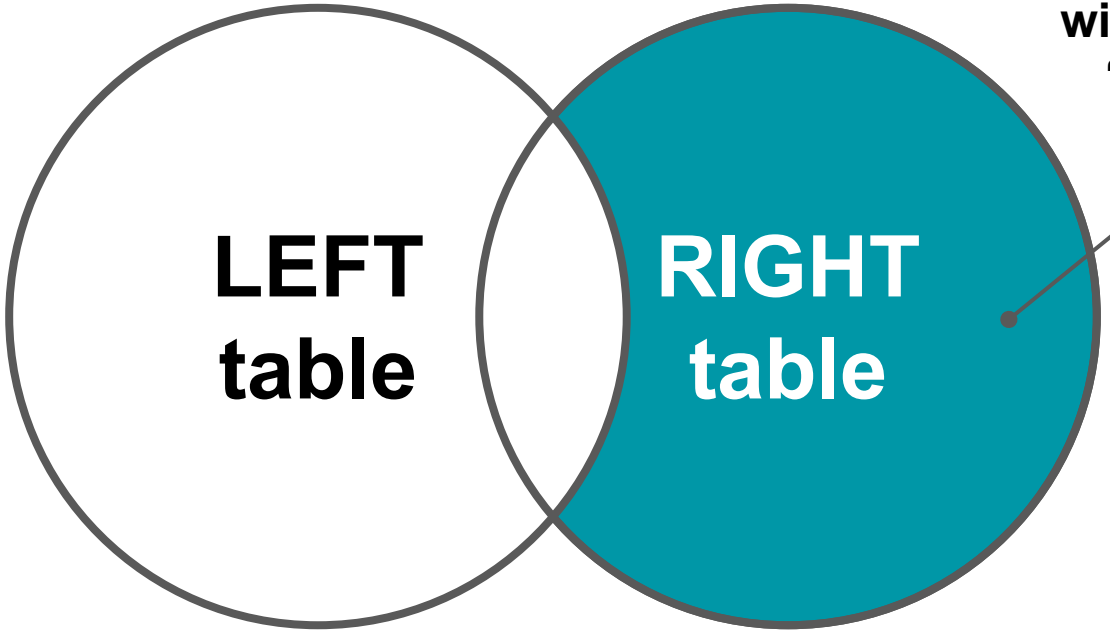
=

left	id	id	right
data	2	*NULL*	*NULL*

*(left join) - (inner join) = left joined table where “keys” do **NOT** match*

left	id	id	right
NULL	*NULL*	3	data

Joined table rows **without** left table "keys" match

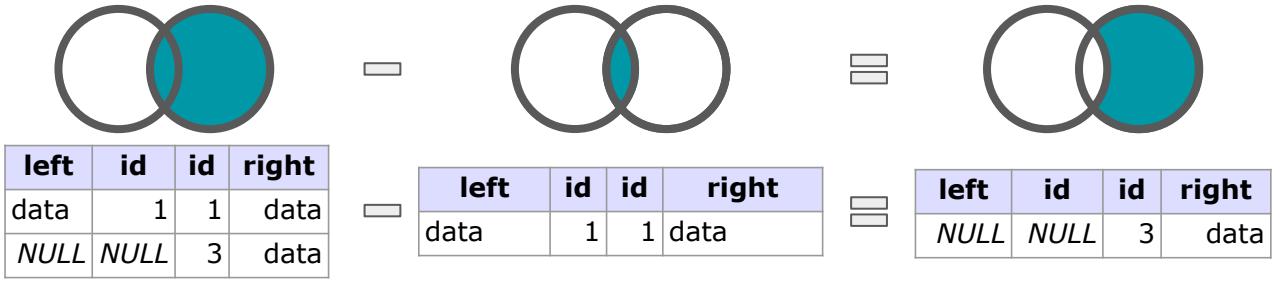


EXCLUSIVE RIGHT JOIN

Exclusive RIGHT JOIN

- Exclusive **`RIGHT JOIN`** returns all records from the left table, but **NOT** the records that fulfill the **`ON`** condition from the left table
  - Example: “Show me all the movies, *without* directors assigned to them”

```
SELECT movie.title, director.name FROM movie
    RIGHT JOIN director ON movie.director_id=director.director_id
    WHERE movie.director_id IS NULL;
```



left	id	id	right
data	1	1	data
NULL	*NULL*	3	data

−

left	id	id	right
data	1	1	data

=

left	id	id	right
NULL	*NULL*	3	data

*(right join) - (inner join) = right joined table where “keys” do **NOT** match*

ON or WHERE in JOIN

- **ON** is used to **filter data before the JOIN** occurs, while both tables are separated. Example:

```
SELECT movie.title, director.name FROM movie
    LEFT JOIN director ON movie.director_id=director.director_id
    AND director.name<>"Wes Anderson";
```

- **WHERE** is used to **filter data after the JOIN** occurs, once 2 tables become 1 Example:

```
SELECT movie.title, director.name FROM movie
    LEFT JOIN director ON movie.director_id=director.director_id
    WHERE director.name<>"Wes Anderson";
```

JOIN - Aliases

- To shorten JOIN queries we can use aliases for our query

```
SELECT m.title, d.name FROM movie m
    INNER JOIN director d
    ON m.director_id=d.director_id;
```

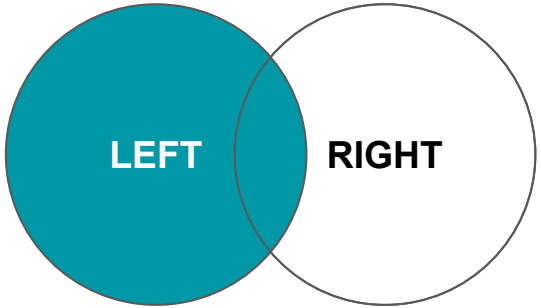
JOIN - 2+ Tables

- We can also join more than 2 tables in a single query

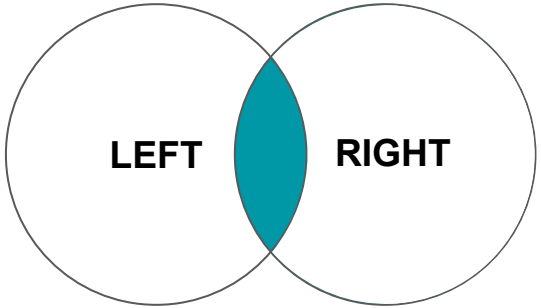
```
SELECT t1.column, t2.column, t3.column FROM table_1 t1
    INNER JOIN table_2 t2
    ON t1.id = t2.id
    INNER JOIN table_3 t3
    ON t1.id = t3.id;
```

JOINS[1]

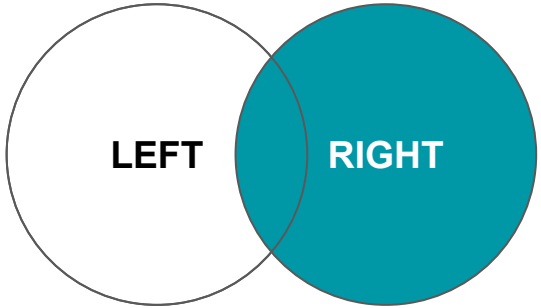
LEFT JOIN



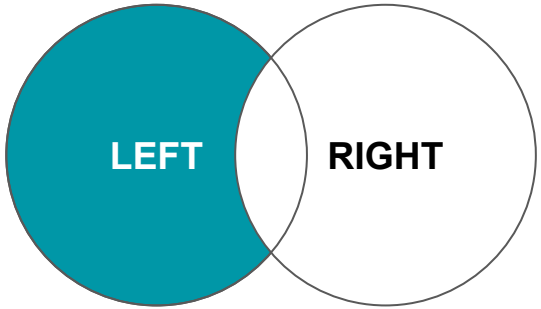
INNER JOIN



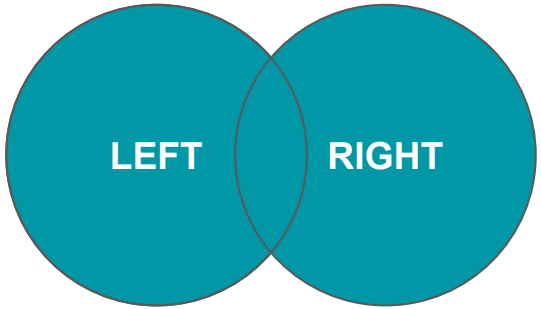
RIGHT JOIN



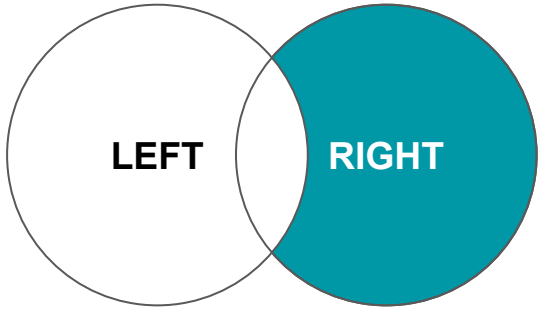
Exclusive LEFT JOIN



FULL JOIN



Exclusive RIGHT JOIN

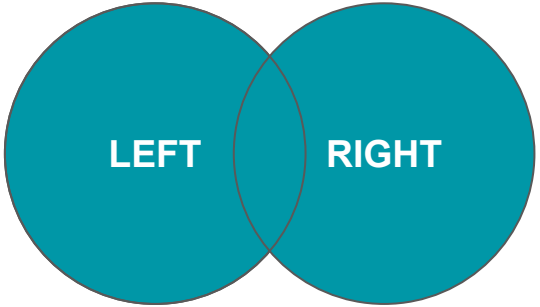


[1] https://www.w3schools.com/mysql/mysql_join.asp

[1] https://www.w3schools.com/mysql/mysql_join_cross.asp
[2] https://www.w3schools.com/mysql/mysql_join_self.asp

Other JOINS

CROSS JOIN[1]

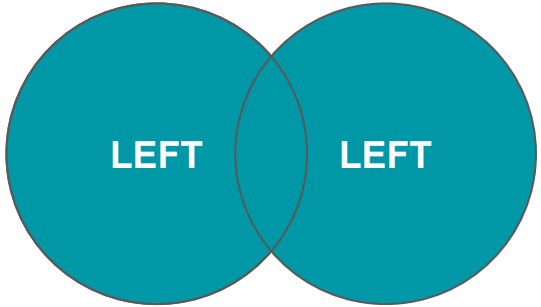


- CROSS JOIN is a full join that does not use the ON keyword
- Therefore it returns every left row matched up with every right row

Cross Join Syntax

SELECT *column_name(s)*
FROM *table1* CROSS JOIN *table2*;

SELF JOIN[2]



- A self join is a regular join, but the table is joined with itself

Self Join Syntax

SELECT *column_name(s)*
FROM *table1 T1, table1 T2*
WHERE *condition*;

Next Week

- Database Design
  - Database design fundamentals
  - Identifying tables & assigning columns
  - Relationships between tables
  - Primary keys and foreign keys
  - Data types in MySQL
- Lab 6 (6%)

Week 5 Terminology

- **Normalization:** the process of organizing data into tables in such a way that the results of using the database are always unambiguous and as intended
- **Primary Key:** a column in a relational database table that's distinctive for each record
- **Foreign Key:** a column or group of columns in a relational database table that provides a link between data in two tables
- **Inner Join:** combine records from two tables whenever there are matching values in a field common to both tables
- **Outer Join:** joins that return matched values and unmatched values from either or both tables
    - Left, Right, Full, Cross are all “outer” joins

SQL Conventions & Keywords

- **SQL Syntax Conventions**
  - When referencing multiple tables in a single query, the columns must be prefixed with the table name and a dot
    - **Eg. movie.title, director.name**

- **Keywords**
  - **INNER JOIN**
  - **ON**
  - **LEFT JOIN**
  - **RIGHT JOIN**
  - **UNION**

left	id	id	right
data	1	1	data

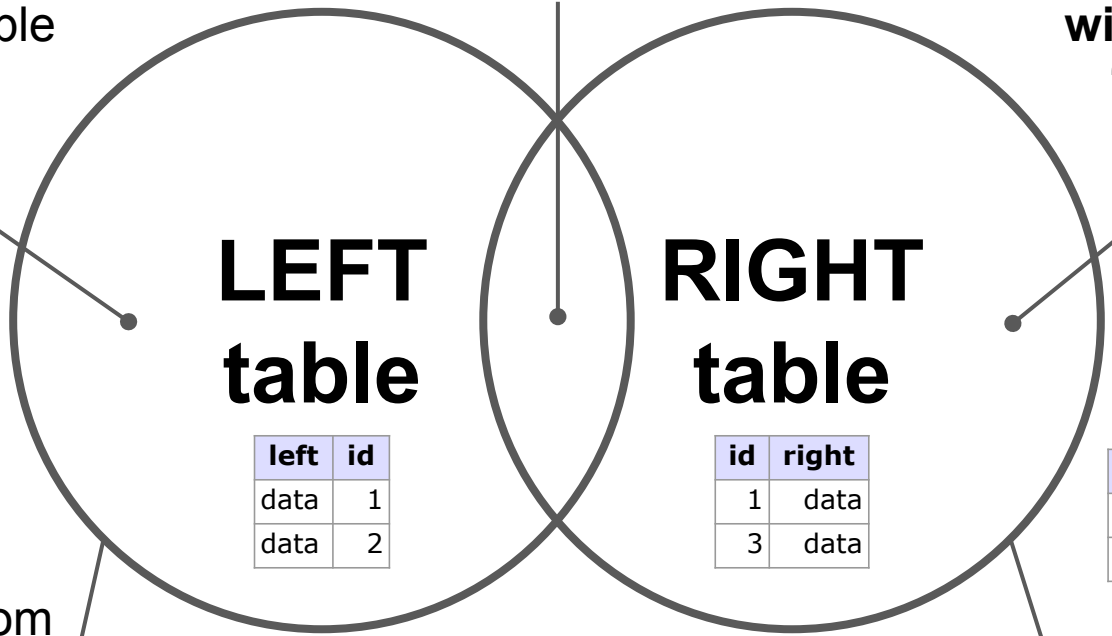
Joined table rows where “keys” match

left	id	id	right
data	2	*NULL*	*NULL*

Joined table rows **without** right table “keys” match

left	id	id	right
NULL	*NULL*	3	data

Joined table rows **without** left table “keys” match



left	id
data	1
data	2

id	right
1	data
3	data

left	id	id	right
data	1	1	data
data	2	*NULL*	*NULL*

All rows of data from left table + right data if it exists

left	id	id	right
data	1	1	data
NULL	*NULL*	3	data

All rows of data from right table + left data if it exists

JOINED TABLES