Module 2-3

Joins

Objectives

- Normalization
- Keys (Primary, Natural, Surrogate, Foreign)
- Cardinality (1-1, 1-M, M-M)
- SQL Joins (Inner and Left Join)
- Unions
- Create a new database (MovieDB)

Normalization

Normalization

- Normalization is a process by which data structures in a relational database are as efficient as possible, including the elimination of redundancy, the minimisation of the use of null values and the prevention of the loss of information.

Aims of Normalization

- Normalization ensures that the database is structured in the best possible way.
- To achieve control over data redundancy. There should be no unnecessary duplication of data in different tables.
- To ensure data consistency. Where duplication is necessary the data is the same.
- To ensure tables have a flexible structure. E.g. number of classes taken or books borrowed should not be limited.
- To allow data in different tables can be used in complex queries.

Duplication vs Redundant Data

- Duplicated Data: When an attribute has two or more identical values
- Redundant Data: If you can delete data with a loss of information

Stages of Normalization

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

Candidate Key / Primary Key

- A *Candidate Key* is an attribute that can be used to uniquely identify each tuple (a single row in a table) in a relation.
- A relation may have more than one candidate key
- If so, one candidate key is nominated as the primary key

First Normal Form

A table is in its first normal form if it contains no repeating attributes or groups of attributes

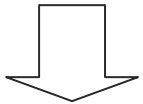
Non-Normalised Table

STUDENT

Number	Name	Classes
001231	William Hartnell	Information Systems, Systems Analysis, Data Communications
001232	Patrick Troughton	Systems Analysis, Data Communications
001233	Jon Pertwee	OO Programming, Systems Analysis, Data Communications
001234	Tom Baker	Systems Analysis, Data Communications

First Normal Form

- To convert data for unnormalised form to 1NF, simply convert any repeated attributes into part of the candidate key
- STUDENT(Number, Name, Classes)



- STUDENT(Number, Name, Classes)

First Normal Form

STUDENT

Number	Name	Classes
001231	William Hartnell	Information Systems
001231	William Hartnell	Systems Analysis
001231	William Hartnell	Data Communications
001232	Patrick Troughton	Systems Analysis
001232	Patrick Troughton	Data Communications
001233	Jon Pertwee	OO Programming
001233	Jon Pertwee	Systems Analysis
001233	Jon Pertwee	Data Communications
001234	Tom Baker	Systems Analysis
001234	Tom Baker	Data Communications

Over to you...

RefNo	Name	Address	Status	AccNo
345	C.J. Date	23, High Street	Business	120768, 348973
543	F.D. Rolland	45, The Ash	Domestic	987654
675	D.R. Howe	17, Low Street	Business	745363, 678453, 348973

Second Normal Form

A table is in the second normal form if it's in the first normal form AND no column that is not part of the primary key is dependant only a portion of the primary key

Second Normal Form

- The concept of functional dependency in central to Normalization and, in particular, strongly related to 2NF.

Functional Dependency

- If ‘X’ is a set of attributes within a relation, then we say ‘A’ (an attribute or set of attributes), is functionally dependant on X, if and only if, for every combination of X, there is only one corresponding value of A
- We write this as :

$$X \rightarrow A$$

Table in 1NF

RefNo	Name	Address	Status	AccNo
345	C.J. Date	23, High Street	Business	120768
345	C.J. Date	23, High Street	Business	348973
543	F.D. Rolland	45, The Ash	Domestic	987654
675	D.R. Howe	17, Low Street	Business	745363
675	D.R. Howe	17, Low Street	Business	678453
675	D.R. Howe	17, Low Street	Business	348973

Second Normal Form

RefNo	AccNo
345	120768
345	348973
543	987654
675	745363
675	678453
675	348973

RefNo	Name	Address	Status
345	C.J. Date	23, High Street	Business
543	F.D. Rolland	45, The Ash	Domestic
675	D.R. Howe	17, Low Street	Business

Over to you...

Supplier#	Part#	City	Quantity
S1	P1	London	1000
S1	P2	London	1500
S1	P3	London	3400
S1	P4	London	2100
S2	P2	Paris	3400
S2	P3	Paris	1000
S4	P1	Nuku alofa	5
S4	P4	Nuku alofa	7

Table in Second Normal Form

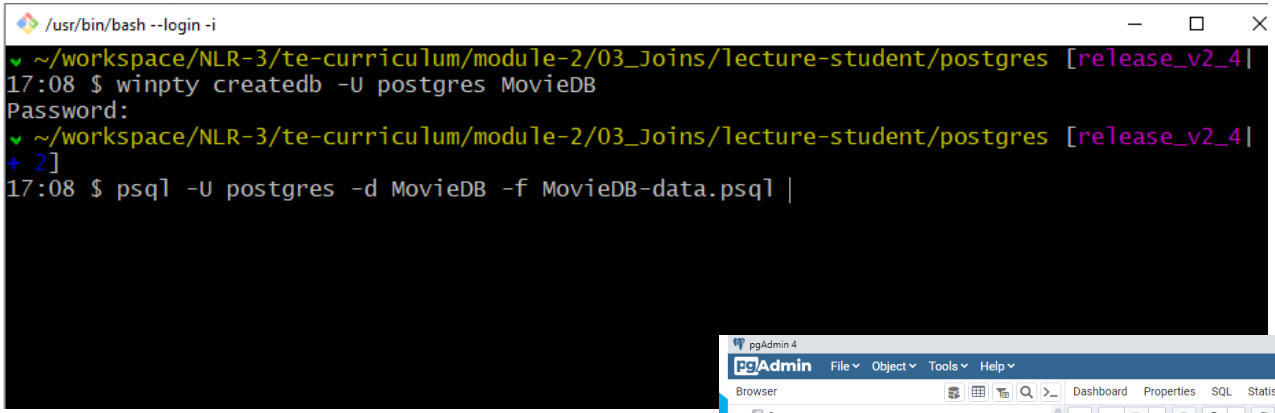
Supplier#	City
S1	London
S2	Paris
S4	Nuku alofa

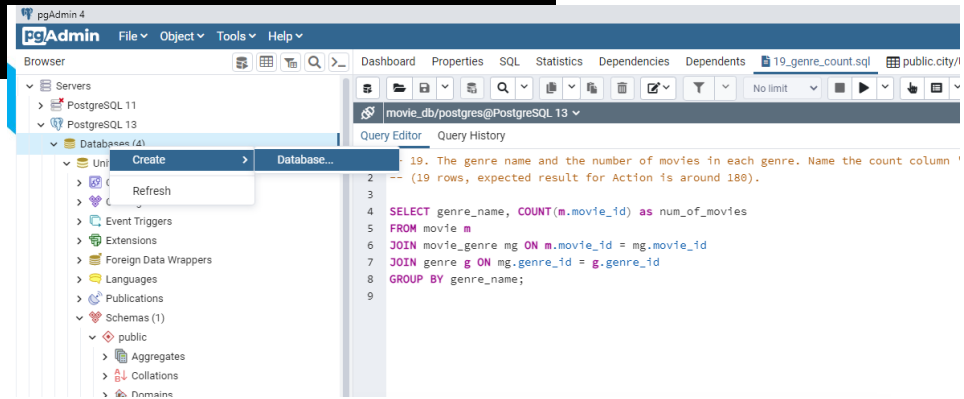
Supplier#	Part#	Quantity
S1	P1	1000
S1	P2	1500
S1	P3	3400
S1	P4	2100
S2	P2	3400
S2	P3	1000
S4	P1	5
S4	P4	7

Third Normal Form

A table is in the third normal form if it is the second normal form and there are no *non-key columns* dependant on *other non-key columns* that could not act as the primary key.

Create the MovieDB and insert some data





Amazon Scenario

Users table

Records for each user

Shipping_Addresses table

Shipping Addresses for each user

Purchases table

User purchases information

Products table

Product Data

Keys

In a relational database, all rows must be unique. The column or combination of columns that make it unique are referred to as **key(s)**.

- **Natural Key**: From real world data, SSN's, customer account numbers, driver license numbers
- **Surrogate Key**: Keys artificially created by an application to make a row unique

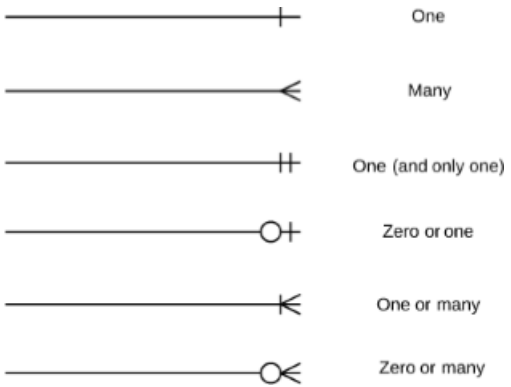
Keys

In a relational database, all rows must be unique. The column or combination of columns that make it unique are referred to as **key(s)**.

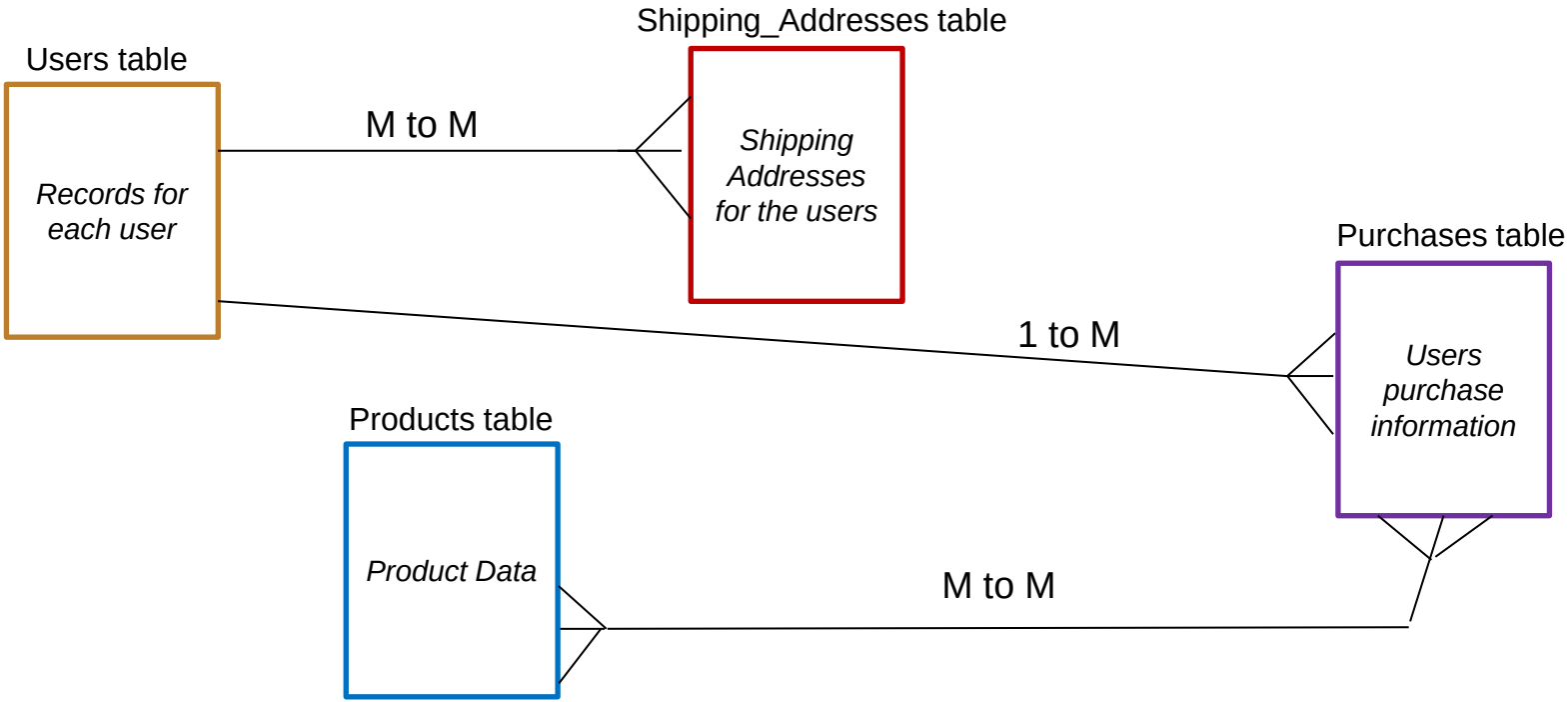
- **Primary Key**: column or columns in a table that uniquely identify the row. These cannot be duplicated.
  - If you say that SSN is your key, there cannot be more than one row with the same SSN.
- **Foreign Key**: A key that exists in another table, in which the latter is the primary key.

Cardinality

- Describes relationship between two tables
- Relationship between a row in one table and a row of another table.
- Options are one or many
- 1 to 1, 1 to M, M to M

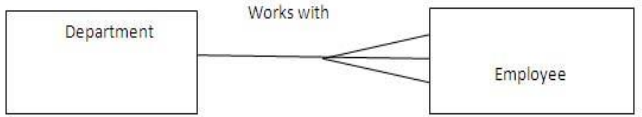


Amazon Scenario

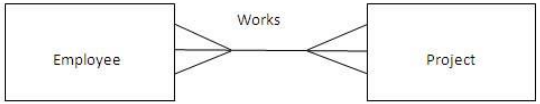


Other examples

a one-to-one relationship



a one-to-many relationship



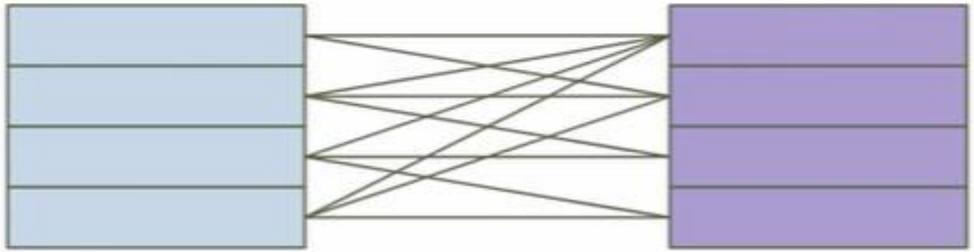
a many-to-many relationship

Many to Many Relationship

Many records in one table

relate to

Many records in another table



Student and Class Tables

student

ID	Name
1	John
2	Mark
3	Sarah
4	Claire

class

ID	Name
2	DB01
5	PH01
7	WEB01
8	WEB02

Student and Class Tables

student

ID	Name	Class ID
1	John	2, 5
2	Mark	5, 7
3	Sarah	8
4	Claire	2, 5, 8

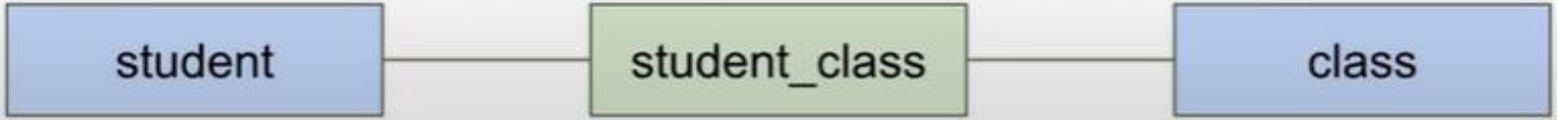
class

ID	Name	Student ID
2	DB01	1, 4
5	PH01	1, 2, 4
7	WEB01	2
8	WEB02	3, 4

Joining Table

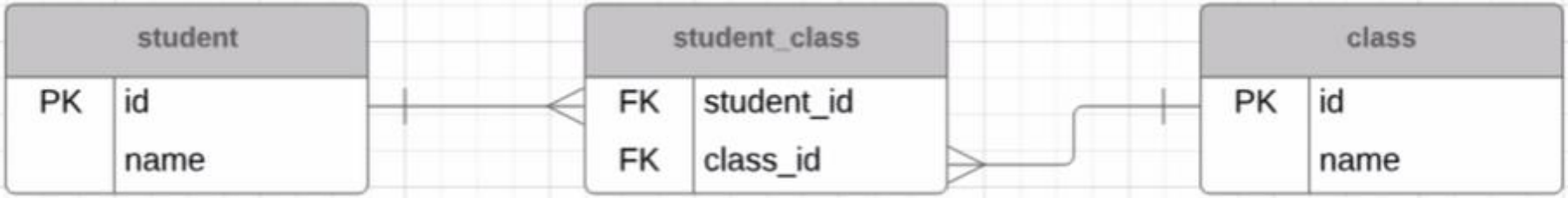
Instead of this:

We have this:



The Joining Table

Captures every instance of student and class


student
PK
id
name
student_class
FK
student_id
FK
class_id
class
PK
id
name

student

ID	Name
1	John
2	Mark
3	Sarah
4	Claire

student_class

Student ID	Class ID
1	2
1	5
2	5
2	7
3	8
4	2
4	5
4	8

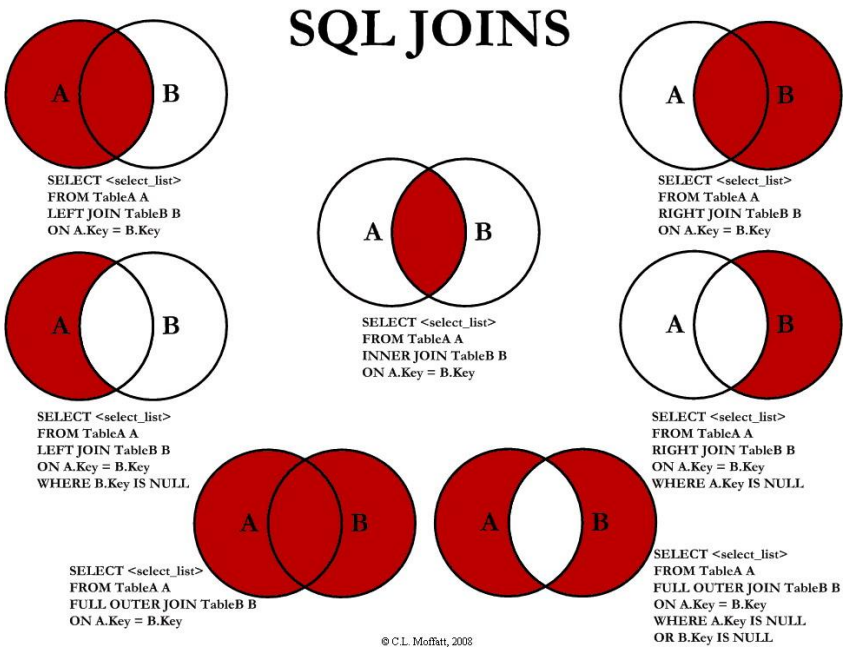
class

ID	Name
2	DB01
5	PH01
7	WEB01
8	WEB02

Joins

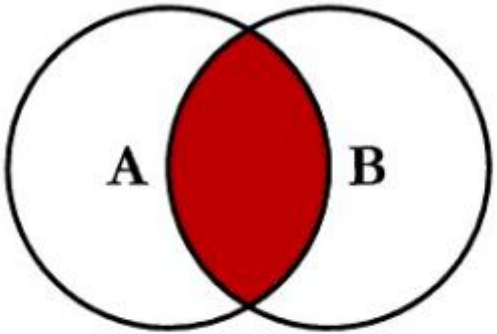
Joins in SQL allow us to pull in data from several tables.

A JOIN is an INNER JOIN



Joins : Inner Join or Join

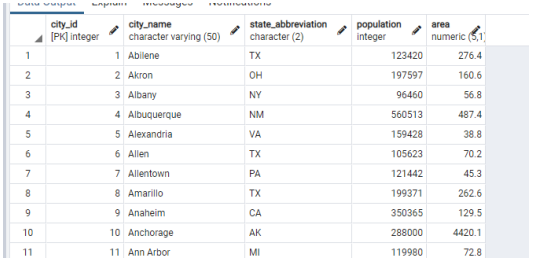
An inner join returns the rows in Table A that has a matching key value in Table B, the Venn Diagram representation is as follows:




```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

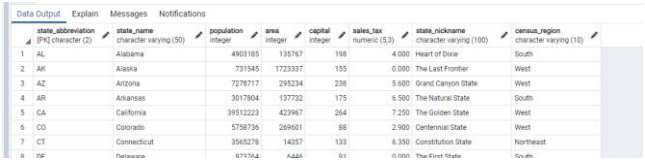
Joins : Inner Join Example

Consider the following example: **We need a SQL query that returns all the cities in Texas. In my result, I want to see the full state name (not the code) followed by the cities in that state**..



	city_id [PK] integer	city_name character varying (50)	state_abbreviation character (2)	population integer	area numeric (5,1)
1	1	Abilene	TX	123420	276.4
2	2	Akron	OH	197597	160.6
3	3	Albany	NY	96460	56.8
4	4	Albuquerque	NM	560513	487.4
5	5	Alexandria	VA	159428	38.8
6	6	Allen	TX	105623	70.2
7	7	Allentown	PA	121442	45.3
8	8	Amarillo	TX	199371	262.6
9	9	Anaheim	CA	350365	129.5
10	10	Anchorage	AK	288000	4420.1
11	11	Ann Arbor	MI	119980	72.8

- The city table contains the list of cities by state abbreviation… but it is missing the full state name.
- The state table contains the list of all states, but it has no data for cities.



Data Output | Explain | Messages | Notifications

	state_abbreviation [PK] character (2)	state_name character varying (50)	population integer	area integer	capital integer	sales_tax numeric (5,3)	state_nickname character varying (100)	census_region character varying (10)
1	AL	Alabama	4903185	135767	198	4.000	Heart of Dixie	South
2	AK	Alaska	731545	1723337	155	0.000	The Last Frontier	West
3	AZ	Arizona	7278717	295234	238	5.600	Grand Canyon State	West
4	AR	Arkansas	3017804	137732	175	6.500	The Natural State	South
5	CA	California	39512223	423967	264	7.250	The Golden State	West
6	CO	Colorado	5758736	269601	88	2.900	Centennial State	West
7	CT	Connecticut	3565278	14357	133	6.350	Constitution State	Northeast
8	DE	Delaware	973764	6446	91	0.000	The First State	South

- What we need to do is combine both tables:
  - Fortunately, these two tables are “related” via the state abbreviation value. Both tables refer to the column as state_abbreviation.

Joins : Inner Join Example

We can combine all of these facts to write a query that combines these two tables:

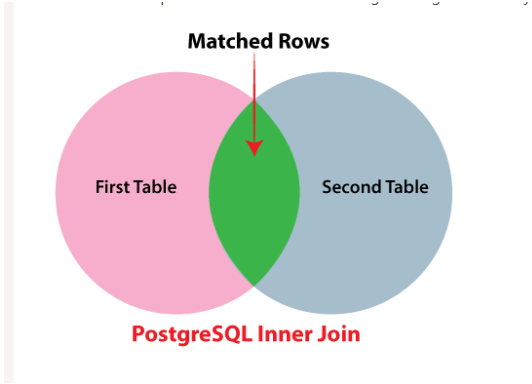
```
SELECT
FROM city c
INNER JOIN state s
ON c.state_abbreviation =
s.state_abbreviation
WHERE c.state_abbreviation =
‘TX';
```

Data Output | Explain | Messages | Notifications

	city_name character varying (50)	state_name character varying (50)
1	Abilene	Texas
2	Allen	Texas
3	Amarillo	Texas
4	Arlington	Texas
5	Austin	Texas
6	Beaumont	Texas
7	Brownsville	Texas
8	Carrollton	Texas
9	College Station	Texas
10	Corpus Christi	Texas
11	Dallas	Texas
12	Denton	Texas
13	Edinburg	Texas
14	El Paso	Texas

luxury_cars	
l_id	INTEGER
luxury_car_names	CHARACTER VARYING(250)

sports_cars	
s_id	INTEGER
sports_car_names	CHARACTER VARYING(250)




```
24
25 SELECT L_ID, luxury_car_names, S_ID, sports_car_names
26 FROM Luxury_cars
27 JOIN Sports_cars
28 ON luxury_car_names= sports_car_names;
29
30
31
```

31:1 [712] INS

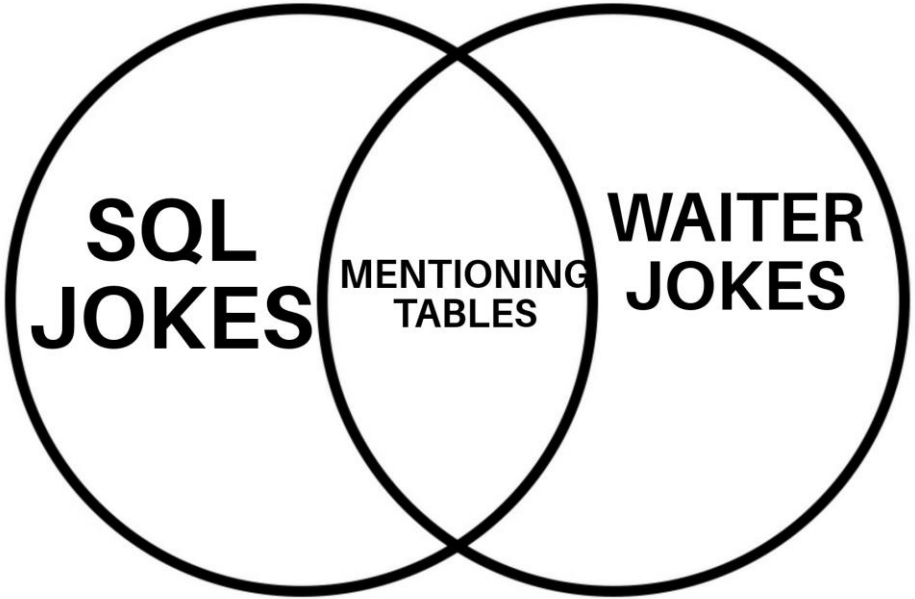
Log | 1: Luxury_cars [2]

*	l_id	luxury_car_names	s_id	sports_car_names
1	1	Chevrolet Corvette	3	Chevrolet Corvette
2	2	Mercedes Benz SL Class	4	Mercedes Benz SL Class

```
25 SELECT L_ID, luxury_car_names, S_ID, sports_car_names
26 FROM Luxury_cars
27 INNER JOIN Sports_cars
28 ON luxury_car_names= sports_car_names;
```

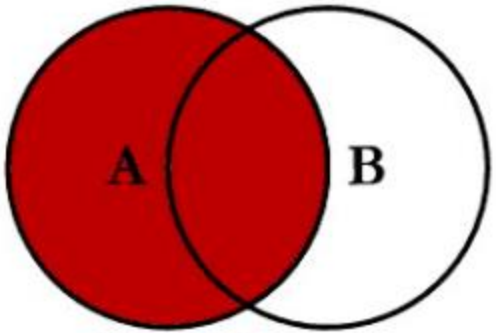
https://www.javatpoint.com/postgresql-join

Let’s write some inner join queries!



Joins : Left Outer Join (can also be called Left Join)

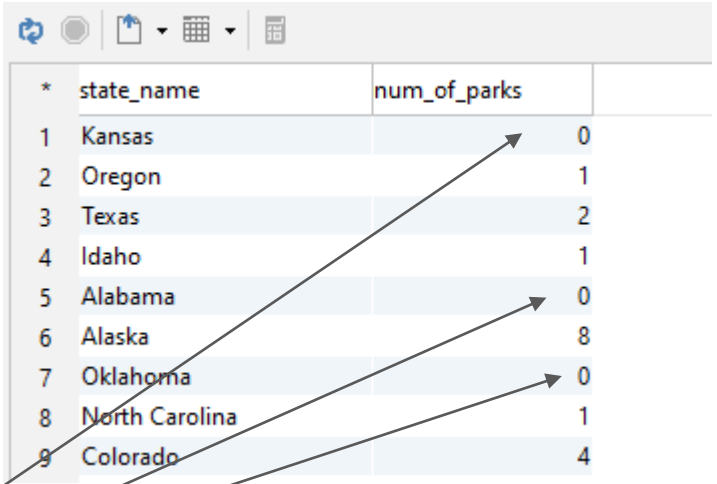
The Left Outer Join returns all the rows on the “left” side table of the join, it will attempt to match to the right side. If there is match… If it can’t find a match it includes it in the result, but with NULL values.




```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

Joins : Left Outer Join Example

```
SELECT state_name, COUNT(park_state.park_id) AS
num_of_parks
   FROM state
LEFT JOIN park_state
   ON state.state_abbreviation =
   park_state.state_abbreviation
GROUP BY state_name;
```



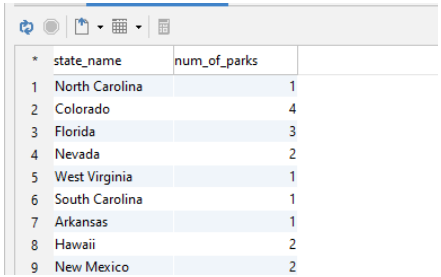
*	state_name	num_of_parks
1	Kansas	0
2	Oregon	1
3	Texas	2
4	Idaho	1
5	Alabama	0
6	Alaska	8
7	Oklahoma	0
8	North Carolina	1
9	Colorado	4

Note that the state_nicknames for Kansas, Alabama, and Oklahoma don't have any parks in the database, so the Left Outer Join instead creates these 0 placeholders.

Joins : Left Join vs Inner Join

With the same data set as the previous slide, let's compare the LEFT OUTER vs an INNER.

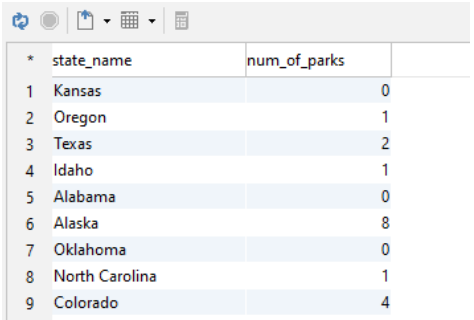
```
SELECT state_name, COUNT(park_state.park_id) AS
num_of_parks
FROM state
JOIN park_state ON state.state_abbreviation =
park_state.state_abbreviation
GROUP BY state_name;
```



*	state_name	num_of_parks
1	North Carolina	1
2	Colorado	4
3	Florida	3
4	Nevada	2
5	West Virginia	1
6	South Carolina	1
7	Arkansas	1
8	Hawaii	2
9	New Mexico	2

With the INNER JOIN, the rows for which there are no matches on the key are dropped from the final result set.

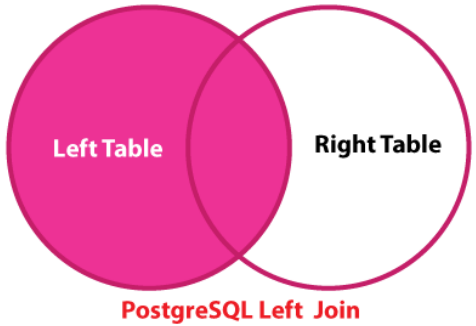
```
SELECT state_name, COUNT(park_state.park_id) AS
num_of_parks
   FROM state
LEFT JOIN park_state
   ON state.state_abbreviation =
   park_state.state_abbreviation
GROUP BY state_name;
```



*	state_name	num_of_parks
1	Kansas	0
2	Oregon	1
3	Texas	2
4	Idaho	1
5	Alabama	0
6	Alaska	8
7	Oklahoma	0
8	North Carolina	1
9	Colorado	4

luxury_cars	
l_id	INTEGER
luxury_car_names	CHARACTER VARYING(250)

sports_cars	
s_id	INTEGER
sports_car_names	CHARACTER VARYING(250)




```
31
32 SELECT L_ID, luxury_car_names, S_ID, sports_car_names
33 FROM Luxury_cars
34 LEFT JOIN Sports_cars
35 ON luxury_car_names= sports_car_names;
36
37
```

35:36 [853] INS

Log | 1: Luxury_cars [5]

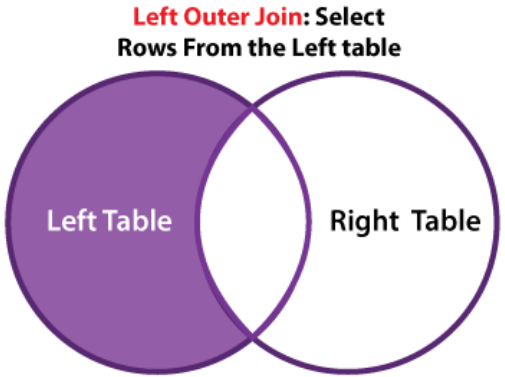
*	l_id	luxury_car_names	s_id	sports_car_names
1	1	Chevrolet Corvette	3	Chevrolet Corvette
2	2	Mercedes Benz SL Class	4	Mercedes Benz SL Class
3	3	Audi A7	(null)	(null)
4	4	Genesis G90	(null)	(null)
5	5	Lincoln Continental	(null)	(null)

```
31
32 SELECT L_ID, luxury_car_names, S_ID, sports_car_names
33 FROM Luxury_cars
34 LEFT OUTER JOIN Sports_cars
35 ON luxury_car_names= sports_car_names;
36
37
```

https://www.javatpoint.com/postgresql-join

luxury_cars	
l_id	INTEGER
luxury_car_names	CHARACTER VARYING(250)

sports_cars	
s_id	INTEGER
sports_car_names	CHARACTER VARYING(250)




```
45
46 SELECT L_ID, luxury_car_names, S_ID, sports_car_names
47 FROM Luxury_cars
48 LEFT JOIN Sports_cars
49 ON luxury_car_names= sports_car_names
50 WHERE S_ID IS NULL;
```

50:22 [1191] INS

Log | 1: Luxury_cars [3]

*	l_id	luxury_car_names	s_id	sports_car_names
1	3	Audi A7	(null)	(null)
2	4	Genesis G90	(null)	(null)
3	5	Lincoln Continental	(null)	(null)

https://www.javatpoint.com/postgresql-join

Unions

A union is a combination of two result sets. The following pattern is used:

[SQL Query 1]

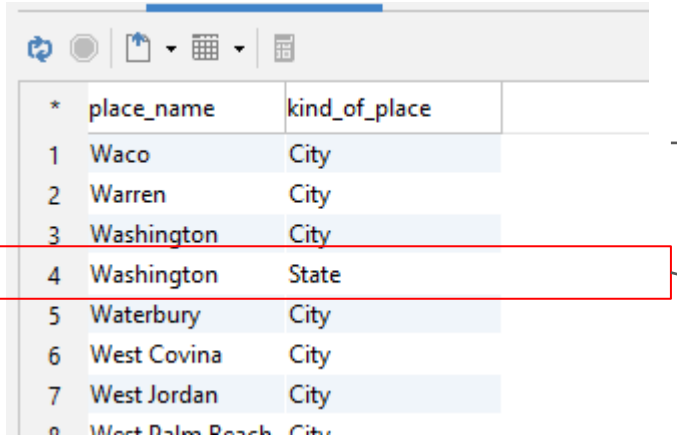
UNION

[SQL Query 2]

Unions Example

Consider the following query:

```
SELECT city_name AS place_name FROM city WHERE city_name LIKE 'W%'
UNION
SELECT state_name AS place_name FROM state WHERE state_name LIKE 'W%'
ORDER BY place_name;
```



*	place_name	kind_of_place
1	Waco	City
2	Warren	City
3	Washington	City
4	Washington	State
5	Waterbury	City
6	West Covina	City
7	West Jordan	City

Result of query first query (returns the City)

Result of query second query (returns the State)

Union All

2) PostgreSQL UNION ALL example

The following statement uses the `UNION ALL` operator to combine result sets from the `top_rated_films` and `most_popular_films` tables:

```
SELECT * FROM top_rated_films
UNION ALL
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957
4	An American Pickle	2020
5	The Godfather	1972
6	Greyhound	2020

In this example, the duplicate row is retained in the result set.