

DATA SOCIETY®

Introduction to SQL - Part 3

*"One should look for what is and not what he thinks should be."
-Albert Einstein.*

Warm up

- Using what you've learned thus far, can you explain how this nerdy joke works?

```
SELECT * FROM users WHERE clue > 0  
0 rows returned
```

Welcome back!

- In the last session we worked on SQL's DDL commands and implemented some simple querying and filtering commands
- Today you will:
 - query datasets and perform data manipulation using SQL commands
 - learn to implement join and set operations

Module completion checklist

Objective	Complete
Query multiple tables using joins	
Combine tables using set operations	
Apply string and numeric functions	
Work with temporal data	
Apply aggregate functions	
Generate groups	

Introduction to joins

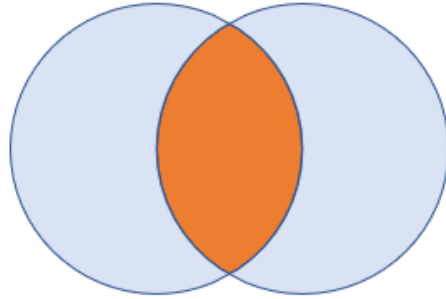
- Until now, we have only queried one table at a time
- If we want to fetch data from multiple tables, we must use **JOIN**
- Joins can be done if there is a common column in each of the tables, which is called a **joining attribute**
- Here some types of joins:

Join type	Description
Inner join	Returns records that have matching values in both the tables
Left outer join	Returns all records from the left table and matched records from the right table
Right outer join	Returns all records from the right table and matched records from the left table
Full outer join	Returns all records from both the tables

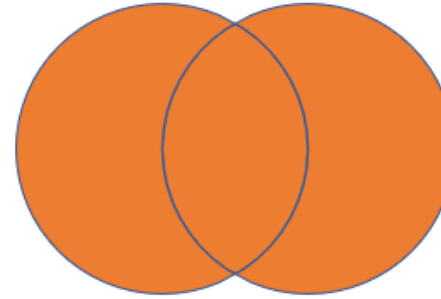
- *Note: the joining attribute will be a primary key in one table and foreign key in the other tables*

Types of joins

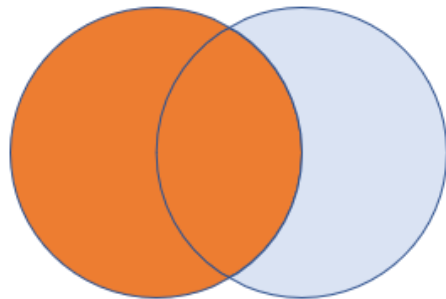
INNER JOIN



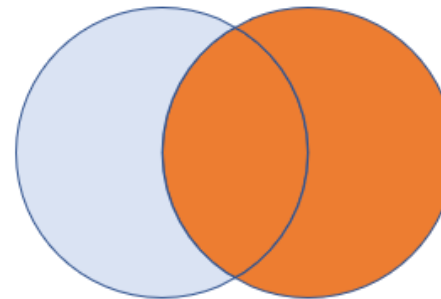
FULL OUTER JOIN



LEFT OUTER JOIN



RIGHT OUTER JOIN



Our scenario

- We will create and use a different database to learn joins
- This database will be created using the `shipping.sql` script that is in your `data` folder
- The file contains information on customers and their orders

Create database and import data

- First, create a new database named `shipping`

```
-- Create a new database transaction.  
CREATE SCHEMA shipping;
```

- Import data to this database from the `shipping.sql` script that is in your `data` folder (just as we did with `world.sql` script earlier)
- Then, switch to the new database

```
-- Switch to shipping database.  
USE shipping;
```


INNER JOIN

- Let's join customer and orders on customer_id column

```
SELECT cu.customer_id AS customer_customer_id,
       cu.customer_name, cu.customer_city,
       o.customer_id AS orders_customer_id,
       o.orders_id, o.item_id
FROM customer AS cu
INNER JOIN orders AS o ON cu.customer_id = o.customer_id;
```

-- select customer_id from customer
-- select attributes from customer
-- select customer_id from orders
-- select attributes from orders
-- alias customer as cu
-- perform inner join, alias orders, and
-- specify joining attribute

Result

#	customer_customer_id	customer_name	customer_city	orders_customer_id	orders_id	item_id
1	DC101	Jack	Washington DC	DC101	1100	1
2	DC101	Jack	Washington DC	DC101	1110	2
3	MD109	Monica	Baltimore	MD109	2330	5

LEFT OUTER JOIN

```
SELECT cu.customer_id AS customer_customer_id,  
       cu.customer_name, cu.customer_city,  
       o.customer_id AS orders_customer_id,  
       o.orders_id, o.item_id  
FROM customer AS cu  
LEFT JOIN orders AS o ON cu.customer_id = o.customer_id;
```

-- select customer_id from customer
-- select attributes from customer
-- select customer_id from orders
-- select attributes from orders
-- alias customer as cu
-- perform left join alias orders and
-- define joining attribute

Result

#	customer_customer_id	customer_name	customer_city	orders_customer_id	orders_id	item_id
1	DC101	Jack	Washington DC	DC101	1110	2
2	DC101	Jack	Washington DC	DC101	1100	1
3	LA675	Sara	Falls Church	NULL	NULL	NULL
4	MD109	Monica	Baltimore	MD109	2330	5
5	MD607	Nick	Columbia	NULL	NULL	NULL

RIGHT OUTER JOIN

```
SELECT cu.customer_id AS customer_customer_id,  
       cu.customer_name, cu.customer_city,  
       o.customer_id AS orders_customer_id,  
       o.orders_id, o.item_id  
FROM customer AS cu  
RIGHT JOIN orders AS o ON cu.customer_id = o.customer_id;
```

-- select customer_id from customer
-- select attributes from customer
-- select customer_id from orders
-- select attributes from orders
-- alias customer as cu
-- perform right join, alias orders and
-- define joining attribute

Result

#	customer_customer_id	customer_name	customer_city	orders_customer_id	orders_id	item_id
1	DC101	Jack	Washington DC	DC101	1100	1
2	DC101	Jack	Washington DC	DC101	1110	2
3	MD109	Monica	Baltimore	MD109	2330	5
4	NULL	NULL	NULL	VG565	4450	5

ANSI join syntax

- There is another way of writing join queries called the **ANSI SQL standard**
- It does not use the **JOIN** and **ON** clauses
- It makes use of the **WHERE** clause and the '=' operator
- It is the most advanced version of join to write optimized queries

ANSI join syntax - cont'd

- Let's switch back to the `world` database to practice the ANSI join syntax

```
-- Fetch the country name and the language spoken from the world database using right join.  
-- We will use CountryCode in countrylanguage table  
-- and Code in country table as the common joining attribute.  
USE world;  
SELECT cl.CountryCode, c.Name, cl.Language      -- refer the attributes using the alias name  
FROM country AS c,                             -- country table is aliased as c  
countrylanguage AS cl                         -- country language aliased as cl  
WHERE cl.CountryCode = c.Code;                 -- join on the country code
```



#	CountryCode	Name	Language
1	ABW	Aruba	Dutch
2	ABW	Aruba	English
3	ABW	Aruba	Papiamentu
4	ABW	Aruba	Spanish
5	AFG	Afghanistan	Balochi
6	AFG	Afghanistan	Dari
7	AFG	Afghanistan	Pashto
8	AFG	Afghanistan	Turkmenian

Joining more than two tables

- Join can be done on more than two tables so long as there is a common column across them
- Let's join all three tables of the world database

```
-- Fetch the city name, country name and the language spoken in each city.  
-- We will use country code as the common joining attribute.  
SELECT ci.Name, co.Name, cl.Language          -- selecting attributes  
FROM city AS ci, country AS co,              -- aliased tables  
countrylanguage AS cl                       -- aliased tables  
WHERE ci.CountryCode = co.Code               -- join countrycode of city and country  
AND co.Code = cl.CountryCode;                -- join countrycode of country and countrylanguage
```

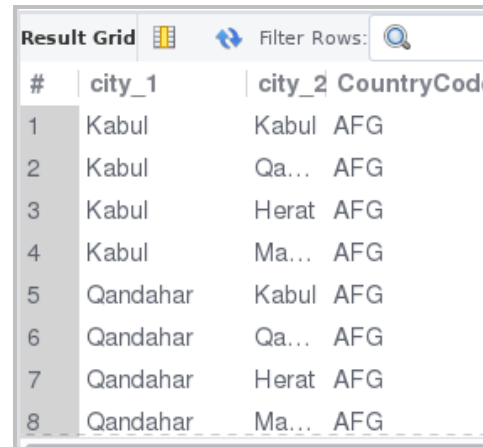


#	Name	Name	Language
1	Oranjestad	Aruba	Dutch
2	Oranjestad	Aruba	English
3	Oranjestad	Aruba	Papiamentu
4	Oranjestad	Aruba	Spanish
5	Kabul	Afghanistan	Balochi
6	Kabul	Afghanistan	Dari
7	Kabul	Afghanistan	Pashto
8	Kabul	Afghanistan	Turkmenian

SELF JOIN

- If we are performing join on the **same table**, then it is a **self** join

```
-- Fetch two cities of the same country.  
SELECT a.Name AS city_1,      -- select 1st city name  
       b.Name AS city_2,      -- select 2nd city name  
       a.CountryCode          -- select the country code  
FROM city a, city b           -- alias the table name  
WHERE a.CountryCode = b.CountryCode; -- join on countrycode
```



The screenshot shows a 'Result Grid' window with a search bar and a refresh icon. The table has four columns: '#', 'city_1', 'city_2', and 'CountryCode'. It displays 8 rows of data, showing pairs of cities from the same country (AFG) joined together. The first four rows show 'Kabul' joined with 'Kabul', 'Qandahar', 'Herat', and 'Ma...' (likely Mazar-e-Sharif). The next four rows show 'Qandahar' joined with 'Kabul', 'Qandahar', 'Herat', and 'Ma...'.

#	city_1	city_2	CountryCode
1	Kabul	Kabul	AFG
2	Kabul	Qa...	AFG
3	Kabul	Herat	AFG
4	Kabul	Ma...	AFG
5	Qandahar	Kabul	AFG
6	Qandahar	Qa...	AFG
7	Qandahar	Herat	AFG
8	Qandahar	Ma...	AFG

Equi and non-equi join

- If the join uses only '=' operator, it is called **equi join**
- Majority of the join types we have seen till now are equi join
- If the join uses other comparison operators like '<', '>', '<=', '>=', '!=', then it is **non equi join**

```
-- Select a list of two cities from two different countries.  
SELECT c1.Name AS city_1, c1.CountryCode as country_1,      -- city 1 details  
       c2.Name as city_2, c2.CountryCode as country_2      -- city 2 details  
FROM city c1 INNER JOIN city c2                            -- self join statement  
ON c1.CountryCode != c2.CountryCode;                      -- non equi join using !=
```

#	city_1	country_1	city_2	country_2
1	Rafah	PSE	Kabul	AFG
2	Nablus	PSE	Kabul	AFG
3	Jabaliya	PSE	Kabul	AFG
4	Hebron	PSE	Kabul	AFG
5	Khan Yunis	PSE	Kabul	AFG
6	Gaza	PSE	Kabul	AFG
7	Gweru	ZWE	Kabul	AFG
8	Mutare	ZWE	Kabul	AFG

Module completion checklist

Objective	Complete
Query multiple tables using joins	✓
Combine tables using set operations	
Apply string and numeric functions	
Work with temporal data	
Apply aggregate functions	
Generate groups	

Set operations

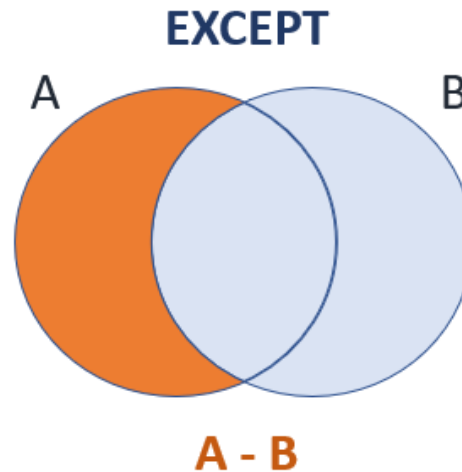
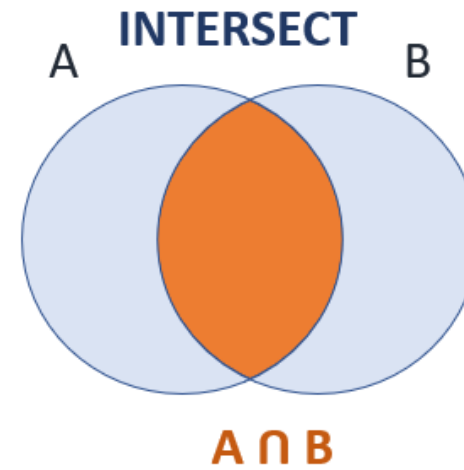
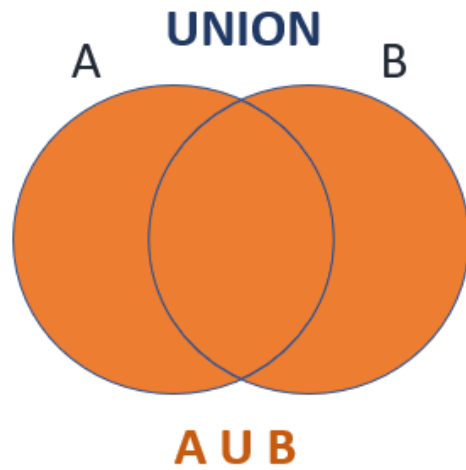
- In addition to joins, we can also combine tables using **set** operators

Set type	Syntax	Description
Union	A UNION B	Selects all data from both tables, removes any duplicates
Union all	A UNION ALL B	Selects all data from both tables, does not remove the duplicates
Intersect	A INTERSECT B	Selects data only if it is present in both tables
Except	A EXCEPT B	Selects data only from the first table that is not present in the second table

Note: INTERSECT & EXCEPT operations are not supported in the current version of MySQL

Rules for set operations

- Both datasets must have the **same number** of columns
- The **data type** of each column across the two datasets must be the same



UNION

```
-- Select the code from the country and countrycode from countrylanguage  
SELECT code from country          -- first dataset  
UNION                             -- union operation  
SELECT CountryCode from countrylanguage; -- second dataset
```

#	code
1	ABW
2	AFG
3	AGO
4	AIA
5	ALB
6	AND
7	ANT
8	ARE

Knowledge check 1



Exercise 1



Module completion checklist

Objective	Complete
Query multiple tables using joins	✓
Combine tables using set operations	✓
Apply string and numeric functions	
Work with temporal data	
Apply aggregate functions	
Generate groups	

String functions

- A **string function** is a function that takes a string value as an input
- They are primarily used for string manipulation
- MySQL has a large collection of them that you can read about here:
<https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>
- Some of the most commonly used ones are:

Function	Description
LENGTH	Finds the length of the string
CONCAT or CONCAT_WS	Concatenates two or more string expressions together
SUBSTR or SUBSTRING	Extracts a substring
LOCATE	Finds the position of a substring/character in a string
REPLACE	Replaces all occurrences of a substring/character in a specified string
REVERSE	Reverses a string
STRCMP	Tests whether two strings are the same

LENGTH & CONCAT

- To find the length of a string, use the **LENGTH** function

```
-- Find the string length in language  
-- from CountryLanguage table.  
SELECT Language, LENGTH(Language) AS  
length_language FROM countrylanguage;
```

#	Language	length_language
1	Dutch	5
2	English	7
3	Papiamentu	10
4	Spanish	7
5	Balochi	7
6	Dari	4
7	Pashto	6
8	Turkmenian	10

- To combine two or more strings, use the **CONCAT** function
- To combine strings with a separator, use the **CONCAT_WS** function

```
-- Concatenate Name and District as location from  
-- the city table with a separator '-'.  
SELECT ID, CONCAT_WS("-", Name, District) AS  
location, countrycode FROM city;
```

#	ID	location	countrycode
1	1	Kabul-Kabul	AFG
2	2	Qandahar-Qandahar	AFG
3	3	Herat-Herat	AFG
4	4	Mazar-e-Sharif-Balkh	AFG
5	5	Amsterdam-Noord-Holland	NLD
6	6	Rotterdam-Zuid-Holland	NLD
7	7	Haag-Zuid-Holland	NLD
8	8	Utrecht-Utrecht	NLD

SUBSTR & LOCATE

- To find a substring in a given string, use the **SUBSTR** function

- SUBSTR(string,
start_position, length)

```
-- Select the first three letters of the region  
-- as the region_code from the country table.  
SELECT SUBSTR(region, 1, 3)  
AS region_code FROM country;
```

Result Grid	
#	region_code
1	Car
2	Sou
3	Cen
4	Car
5	Sou
6	Sou
7	Car
8	Mid

- To find the first occurrence of a substring or character in a string, use the **LOCATE** function

- LOCATE(substring, string)

```
-- Locate the position of letter 'a' from  
-- the district column in the city table.  
SELECT LOCATE('a', district)  
AS position_of_a, district FROM city;
```

Result Grid		
#	position_of_a	district
1	2	Kabol
2	2	Qandahar
3	4	Herat
4	2	Balkh
5	11	Noord-Holland
6	10	Zuid-Holland
7	10	Zuid-Holland
8	0	Utrecht

A note on LIKE

- We use the **LIKE** clause to query the database by matching a pattern with string data
- A **wildcard** character is used to substitute any character in the string
- Two important wildcard characters used in conjunction with the **LIKE** clause are:
 - **% percent**, which represents zero, one, or multiple characters
 - **_ underscore**, which represents exactly one character
- The wildcard characters can also be used in combination

Syntax	Description
LIKE 'd%'	Find any value that starts with letter 'd'
LIKE '%a'	Find any value that ends with letter 'a'
LIKE '_e%'	Find any value of any length that has letter 'e' in the second position
LIKE '%is%'	Find any value that has substring 'is' anywhere in the string
LIKE '_r_'	Find three-letter words having letter 'r' in the second position
LIKE 'a%n'	Find any values starting with 'a' and ending with 'n'

LIKE

```
-- Select all the data from the city table
-- where the District starts with
-- letter 'K'.
SELECT * FROM city WHERE District LIKE 'K%';
```

#	ID	Name	CountryCode	District	Population
1	1	Kabul	AFG	Kabul	1780000
2	152	Khulna	BGD	Khulna	663340
3	159	Jessore	BGD	Khulna	139710
4	549	Ouagadougou	BFA	Kadiogo	824000
5	608	Cairo	EGY	Kairo	6789479
6	634	Kafr al-Shaykh	EGY	Kafr al-Shaykh	124819
7	644	Disuq	EGY	Kafr al-Shaykh	91300
8	654	Barcelona	ESP	Katalonia	1503451

```
-- Select all the data from the city table
-- where the District has second letter as
-- 'u' and end with the letter 't'.
SELECT * FROM city WHERE District LIKE '_u%t';
```

#	ID	Name	CountryCode	District	Population
1	146	Sumqayit	AZE	Sumqayit	283000
2	929	Port-au-Prince	HTI	Ouest	884472
3	930	Carrefour	HTI	Ouest	290204
4	931	Delmas	HTI	Ouest	240429
5	950	Padang	IDN	Sumatera Barat	534474
6	963	Mataram	IDN	Nusa Tenggara Barat	306600
7	1029	Ahmedabad	IND	Gujarat	2876710
8	1035	Surat	IND	Gujarat	1498817

Numeric functions

- **Numeric functions** are built-in functions that can be used for mathematical calculations
- MySQL has many such functions that you can read about here:
<https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html>
- Some of the most commonly used ones are:

Function	Description
MOD	Finds the remainder when one number is divided by another
POW	Finds the power of a number
ROUND	Rounds to the nearest number
TRUNCATE	Truncates a given number

MOD & POW

- To find the modulus, use the **MOD** function

```
-- Divide the population by 10 and fetch  
-- the remainder as population_rem_10.  
SELECT MOD(population, 10)  
AS population_rem_10, population FROM city;
```

#	population_rem_10	population
12	8	160398
13	1	153491
14	3	152463
15	4	149544
16	2	148772
17	5	142465
18	0	138020
19	1	135621

- To find the power of a number raised by another number, use the **POW** function
 - POW (m, n) returns m raised to nth power

```
-- Find the cube of the surface area  
-- of the country as Surface_area_cube.  
SELECT POW(SurfaceArea, 3)  
AS Surface_area_cube, SurfaceArea FROM country;
```



#	Surface_area_cube	SurfaceArea
1	7189057	193.00
2	2.77282601924329e17	652090.00
3	1.937697051563e18	1246700.00
4	884736	96.00
5	23758712844992	28748.00
6	102503232	468.00
7	512000000	800.00
8	584277056000000	83600.00

TRUNCATE & ROUND

- To truncate a number, use the **TRUNCATE** function

- `TRUNCATE (number, decimal_places)`



```
-- Find gnp divided by 1000 and
-- truncate to 2 decimal places.
SELECT TRUNCATE(GNP / 1000, 2), GNP from
country;
```

Result Grid  Filter Rows: 		
#	TRUNCATE(GNP / 1000, 2)	GNP
1	0.82	828.00
2	5.97	5976.00
3	6.64	6648.00
4	0.06	63.20
5	3.20	3205.00
6	1.63	1630.00
7	1.94	1941.00
8	37.96	37966.00

- To round a number, use the **ROUND** function

- `ROUND (number, decimal_places)`

```
-- Divide gnp by 1000 and round off
-- to 2 decimal places.
SELECT ROUND(GNP / 1000, 2), GNP from country;
```

Result Grid  Filter Rows: 		
#	ROUND(GNP / 1000, 2)	GNP
1	0.83	828.00
2	5.98	5976.00
3	6.65	6648.00
4	0.06	63.20
5	3.21	3205.00
6	1.63	1630.00
7	1.94	1941.00
8	37.97	37966.00

Module completion checklist

Objective	Complete
Query multiple tables using joins	✓
Combine tables using set operations	✓
Apply string and numeric functions	✓
Work with temporal data	
Apply aggregate functions	
Generate groups	

Date functions

- Recall that temporal data is used to store date and time information
- MySQL has built-in functions for date manipulation, which are described here:
<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>
- Some of the most commonly used ones are:

Function	Description
CURDATE	Returns the current date
ADDTIME	Returns the time after a certain time interval is added
DATEDIFF	Returns the difference in days between the two date values
LASTDAY	Returns the last day of the month
DAYNAME	Returns weekday name for the date
EXTRACT	Extracts parts from the date

CURDATE & ADDTIME

- To find today's date, use the **CURDATE** function

```
-- Get current date and display  
-- as Today_date.  
SELECT CURDATE() AS Today_date;
```

Today_date
▶ 2021-04-12

- To add time to the existing time, use the **ADDTIME** function
- `ADDTIME(start_value, time_to_add)`

```
-- Add 2 hours, 10 minutes and 20 seconds  
-- to the current time.  
SELECT CURRENT_TIMESTAMP()  
AS time_now,  
ADDTIME(CURRENT_TIMESTAMP(), "2:10:20")  
AS new_time;
```

time_now	new_time
▶ 2021-04-12 15:32:02	2021-04-12 17:42:22

DATEDIFF & LASTDAY

- Use the **DATEDIFF** function to find the number of days between the two given dates

```
-- Find number of days between September 12th  
-- 2018 and May 12th 2017.  
SELECT DATEDIFF("2018-09-12", "2017-05-12")  
AS date_difference;
```

	date_difference
▶	488

- Use the **EXTRACT** function to get the day part from the date

```
-- Extract the day part from  
-- '2018-05-11 22:12:29'.  
SELECT EXTRACT(DAY FROM '2019-05-11 22:12:29')  
AS day;
```

Result Grid	
#	day
1	11

Module completion checklist

Objective	Complete
Query multiple tables using joins	✓
Combine tables using set operations	✓
Apply string and numeric functions	✓
Work with temporal data	✓
Apply aggregate functions	
Generate groups	

Aggregate functions

- **Aggregate functions** take a collection of values as input and return one value as the output
- There are five built-in aggregate functions:

Function	Definition
MIN	Returns the minimum value
MAX	Returns the maximum value
AVG	Returns the average value
SUM	Returns the sum of values
COUNT	Returns the number of observations

Note: These aggregate functions ignore NULL values!

MIN & MAX

- Use the **MIN** function to find the minimum in a column

```
-- Find the minimum life expectancy  
-- from the country table.  
SELECT MIN(lifeexpectancy)  
AS least_life FROM country;
```

	least_life
▶	37.2

- Use the **MAX** function to find the maximum in a column

```
-- Find the maximum life expectancy  
-- from the country table.  
SELECT MAX(lifeexpectancy)  
AS max_life FROM country;
```

	max_life
▶	83.5

AVG & SUM

- Use the **AVG** function to find the average in a column

```
-- Find the average life expectancy  
-- from the country table.  
SELECT AVG(lifeexpectancy)  
AS average_life FROM country;
```

	average_life
▶	66.48604

- Use the **SUM** function to find the sum of a column

```
-- Find the total population  
-- across all cities.  
SELECT SUM(population)  
AS total_population FROM city;
```

	total_population
▶	1429559884

COUNT

- Use the **COUNT** function to find the total number of observations in the column
- We can also use **DISTINCT** to find the total number of distinct observations
- COUNT (*) counts the total number of rows in the table

```
-- Find the number of countries who got their independence.  
SELECT COUNT(indepyear) AS number_independent_countries FROM country;
```

	number_independent_countries
▶	192

```
-- Find the distinct number of indepyear from the country table.  
SELECT COUNT(DISTINCT indepyear) AS number_distinct_indepyear FROM country;
```

	number_distinct_indepyear
▶	88

Using expressions

- We can also build expressions to use in these aggregate functions

```
-- Find the maximum difference between the gnpold and gnp in the country table.  
SELECT MAX(GNPold - GNP) from country;
```

	MAX(GNPold - GNP)
▶	405596.00




Module completion checklist

Objective	Complete
Query multiple tables using joins	✓
Combine tables using set operations	✓
Apply string and numeric functions	✓
Work with temporal data	✓
Apply aggregate functions	✓
Generate groups	

GROUP BY

- People are not interested in looking at raw data
- It is useful to form groups when analyzing data, especially when using aggregate functions
- For example, if we wanted to find the value of the least populated city by country (instead of the overall least populated city), our query would look like this:

```
-- Find the least population of a city for each country.  
SELECT MIN(population) AS least_population, countrycode FROM city -- select attributes  
GROUP BY CountryCode; -- group by countrycode
```

Result Grid   Filter Rows: 		
#	least_population	countrycode
1	29034	ABW
2	127800	AFG
3	118200	AGO
4	595	AIA
5	270000	ALB
6	21189	AND
7	2345	ANT
8	114395	ARE

GROUP BY - multi column grouping

- We can also group data by more than one column

```
-- Find the number of cities in each district in each country.  
SELECT CountryCode, District,      -- select the attributes  
count(Name) FROM city              -- find the count  
GROUP BY countrycode, district;    -- group by district and countrycode
```

#	CountryCode	District	count(Name)
1	AFG	Kabul	1
2	AFG	Qandahar	1
3	AFG	Herat	1
4	AFG	Balkh	1
5	NLD	Noord-Holland	5
6	NLD	Zuid-Holland	6
7	NLD	Utrecht	2
8	NLD	Noord-Brabant	4

GROUP BY - ROLLUP

- In multi-column grouping, if we want to perform the additional operation of finding the total value of each group, we use the **ROLLUP** clause
- For example, in addition to grouping by district and country code, if we want to find the total number of cities in each country, use the **ROLLUP** clause

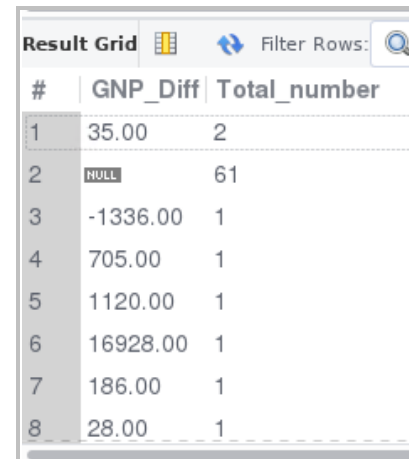
```
SELECT CountryCode, District,      -- select the attributes
COUNT(Name) FROM city            -- find count of cities
GROUP BY countrycode, district WITH ROLLUP; -- use rollup to find grand total
```

Result Grid			
Filter Rows:			
#	CountryCode	District	COUNT(Name)
1	ABW	—	1
2	ABW	NULL	1
3	AFG	Balkh	1
4	AFG	Herat	1
5	AFG	Kabul	1
6	AFG	Qandahar	1
7	AFG	NULL	4
8	AGO	Benguela	2

GROUP BY - expression

- Sometimes we might want to group by a value that is not present in the data table

```
-- Find the number of countries who have the same GNP difference.  
SELECT (GNP-GNPold) AS GNP_Diff,      -- gnp difference expression  
COUNT(*) Total_number FROM country  -- count number of countries  
GROUP BY (GNP-GNPold);                -- group by gnp difference
```



#	GNP_Diff	Total_number
1	35.00	2
2	NULL	61
3	-1336.00	1
4	705.00	1
5	1120.00	1
6	16928.00	1
7	186.00	1
8	28.00	1

GROUP BY & HAVING - filter groups

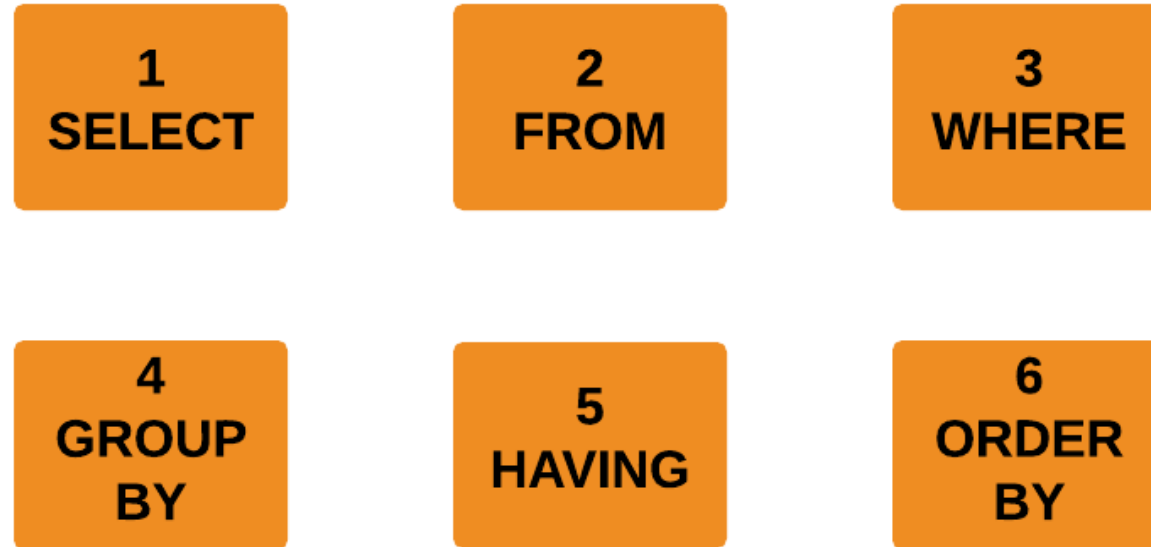
- If we need to filter out unwanted groups, we use the **HAVING** clause

```
-- Find the least populated city from each country;  
-- ignore if the population is less than 30000  
SELECT MIN(population), name, countrycode -- select the attributes  
FROM city GROUP BY countrycode, name -- group by countrycode, name  
HAVING MIN(population) > 30000; -- filter out if population is less than 30000
```

#	MIN(population)	name	countrycode
1	1780000	Kabul	AFG
2	237500	Qandahar	AFG
3	186800	Herat	AFG
4	127800	Mazar-e-Sharif	AFG
5	731200	Amsterdam	NLD
6	593321	Rotterdam	NLD
7	440900	Haag	NLD
8	234323	Utrecht	NLD

Order matters!

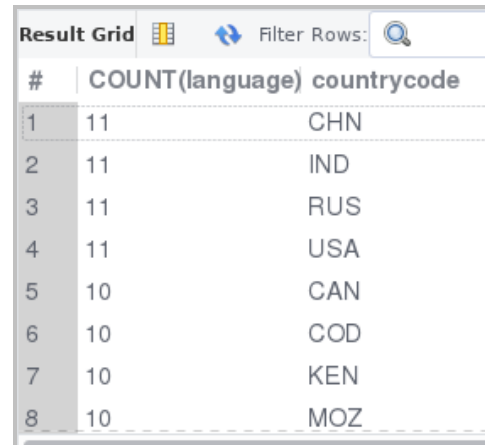
- Always remember that the order of the clauses matters!



Example of a complex query

- Let's write a complex query to know how to make use of all the clauses

```
-- Find the number of unofficial languages for each country and arrange them in descending order.
-- Remove the entry if the number of unofficial languages is one.
SELECT COUNT(language), countrycode -- select count of the languages
FROM countrylanguage                -- from the table countrylanguage
WHERE IsOfficial != 'T'             -- filter out if the language is official
GROUP BY countrycode                -- group by country
HAVING COUNT(language) > 1          -- remove if number is just one
ORDER BY COUNT(language) DESC;      -- order by the count
```



#	COUNT(language)	countrycode
1	11	CHN
2	11	IND
3	11	RUS
4	11	USA
5	10	CAN
6	10	COD
7	10	KEN
8	10	MOZ

Knowledge check 2



Exercise 2



Module completion checklist

Objective	Complete
Query multiple tables using joins	✓
Combine tables using set operations	✓
Apply string and numeric functions	✓
Work with temporal data	✓
Apply aggregate functions	✓
Generate groups	✓

Next steps

In the next session we will:

- Implement SQL subqueries
- Work with views and indexes
- Learn about stored procedures and SQL transactions



This completes our module
Congratulations!