DATA SOCIETY®

Introduction to SQL - day 2

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

Module completion checklist

Objective	Complete
Summarize the use of query clauses	
Apply conditional filtering of data	
Query multiple tables - introduction to joins	
Define various joins and when to implement each type	
Understanding set operations	
Understanding and applying string functions	
Understanding and applying numeric functions	

'World' database introduction

- In this module, we will be creating a database called world
- The `world.sql` file contains data for a world database
- This file came from the MYSQL website
- This dataset has information about countries, cities and languages
- Your manager would like a database of things like :
  - Population from cities
  - Aliased columns/tables
  - Other needed information for the annual global report

'World' database schema

- We are going to create a new database called **world**
- We will be using this database to further study SQL

city table

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment
Name	char(35)	NO			
CountryCode	char(3)	NO	MUL		
District	char(20)	NO			
Population	int(11)	NO		0	

country table

Field	Type	Null	Key	Default	Extra
Code	char(3)	NO	PRI		
Name	char(52)	NO			
Continent	enum('Asia','Europe','North America','Africa','Oc...	NO		Asia	
Region	char(26)	NO			
SurfaceArea	float(10,2)	NO		0.00	
IndepYear	smallint(6)	YES		NULL	
Population	int(11)	NO		0	
LifeExpectancy	float(3,1)	YES		NULL	
GNP	float(10,2)	YES		NULL	
GNPOld	float(10,2)	YES		NULL	
LocalName	char(45)	NO			
GovernmentF	char(45)	NO			
HeadOfState	char(60)	YES		NULL	
Capital	int(11)	YES		NULL	
Code2	char(2)	NO			

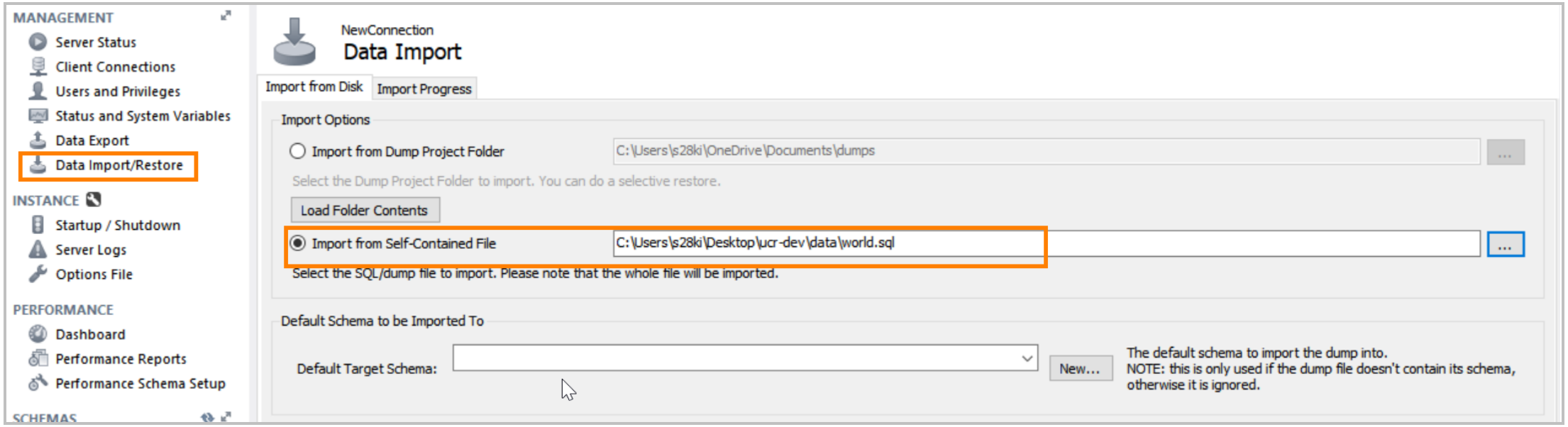
country language table

Field	Type	Null	Key	Default	Extra
CountryCode	char(3)	NO	PRI		
Language	char(30)	NO	PRI		
IsOfficial	enum('T','F')	NO		F	
Percentage	float(4,1)	NO		0.0	

Create new 'world' database

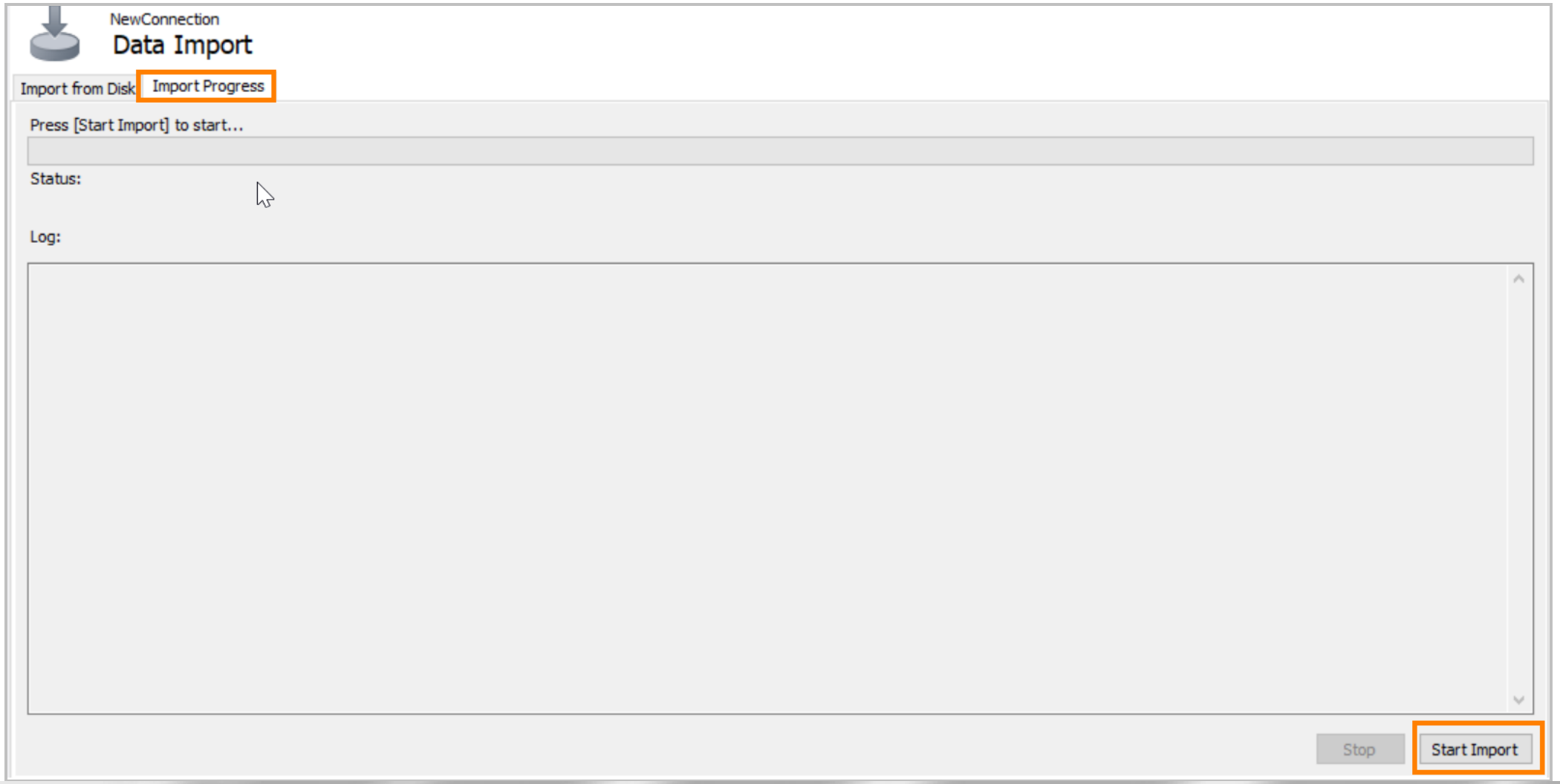
Import data from .sql file

- We are going to import the schema and data from the .sql file
- Click on the **Data import/Restore** and navigate to the data folder to select the `world.sql` file



Import data from .sql file

- Click on **import progress** and **start import**



Query clauses

- There are six clauses that are used for data manipulation and querying
- **The order of the clause matters**

Clause name	Definition
SELECT	Determines which columns to include in the query's result set
FROM	Identifies the table from which to draw the data and how the tables should be joined
WHERE	Filters out unwanted data
GROUP BY	Used to group rows by common column values
HAVING	Filters out unwanted groups
ORDER BY	Orders the rows ascending or descending of the final result set by one or more columns

SELECT & FROM

- **SELECT** could be used to select all columns in a table

```
-- Switch to world database for convenience
USE world;

-- To select all data, use * operator
SELECT * FROM city;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabol	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238

- **SELECT** could be used to select some columns in a table

```
-- To select specific columns.
SELECT Name, Population FROM city;
```

Result Grid | Filter Rows: | Export:

Name	Population
Kabul	1780000
Qandahar	237500
Herat	186800
Mazar-e-Sharif	127800
Amsterdam	731200
Rotterdam	593321
Haag	440900
Utrecht	234323
Eindhoven	201843
Tilburg	193238

SELECT & FROM - various data operations

- Other operations that can be done on the final result set include:
  - Add a new column with same data across all rows
  - Perform mathematical operations on numeric columns
  - Perform string operations on character columns

```
SELECT ID,                  -- select ID
'City_population',          -- add a new column with value City_population
Population/100,             -- divide the population column by 100
LOWER(CountryCode)          -- convert the country code column to lower case
FROM city;
```

ID	City_population	Population/100	LOWER(CountryCode)
1	City_population	17800.0000	afg
2	City_population	2375.0000	afg
3	City_population	1868.0000	afg
4	City_population	1278.0000	afg
5	City_population	7312.0000	nld
6	City_population	5933.2100	nld
7	City_population	4409.0000	nld
8	City_population	2343.2300	nld

DISTINCT - remove duplicates

- A query might return duplicate rows of data
- To return *only* distinct rows, we use the **DISTINCT** clause

```
SELECT Language FROM countrylanguage;

-- Returns 984 rows
```

Result Grid

	language
▶	Dutch
	English
	Papiamento
	Spanish
	Balochi
	Dari
	Pashto
	Turkmenian
	Uzbek
	Ambo

```
SELECT DISTINCT Language FROM countrylanguage;

-- Returns 457 rows
```

Result Grid

	Language
▶	Dutch
	English
	Papiamento
	Spanish
	Balochi
	Dari
	Pashto
	Turkmenian
	Uzbek
	Ambo

	#	Time	Action	Message
✔	11	15:02:52	select language from countrylanguage LIMIT 0, 1000	984 row(s) returned
✔	12	15:05:33	select distinct Language from countrylanguage LIMIT 0, 1000	457 row(s) returned

AS - alias

- The **AS** clause is used for aliasing table names and column names
- A **column alias** gives a new name to the existing column in the final result set
- A **table alias** gives a new name to the existing table

```
SELECT ci.Name AS city_name,    -- alias column name as city_name
ci.District AS city_district    -- alias column district as city_district
FROM city AS ci;                -- alias table city as ci
```

	city_name	city_district
▶	Kabul	Kabol
	Qandahar	Qandahar
	Herat	Herat
	Mazar-e-Sharif	Balkh
	Amsterdam	Noord-Holland
	Rotterdam	Zuid-Holland
	Haag	Zuid-Holland
	Utrecht	Utrecht

WHERE - filter conditions

- MySQL allows you to use filter conditions with the **WHERE** clause to filter the data

Condition type	Syntax
Logical condition	`OR, AND, NOT`
Equality or matching condition	`=`
Comparison condition	`<, <=, >, >=, <>, !=`
Range condition	BETWEEN & AND
Membership condition	IN
Null condition	IS NULL; IS NOT NULL;

WHERE - filter data

```
-- Select all data from 'countrylanguage' table where the language is official to the country and
-- the percentage spoken is greater than 70%.
SELECT * FROM countrylanguage -- table name
  WHERE                       -- WHERE clause to filter
  IsOfficial = 'T'            -- equality condition '='
  AND                         -- logical condition 'AND'
  Percentage > 70;            -- comparison condition '>'
```

Result Grid | Filter Rows: | Edit

CountryCode	Language	IsOfficial	Percentage
ALB	Albaniana	T	97.9
ANT	Papiamento	T	86.2
ARG	Spanish	T	96.8
ARM	Armenian	T	93.4
ASM	Samoan	T	90.6
AUS	English	T	81.2
AUT	German	T	92.0
AZE	Azerbaijani	T	89.0
BDI	Kirundi	T	98.1
BGD	Bengali	T	97.7

WHERE - filter data

```sql
-- Select name and population of the cities
where population is between 180000 and 190000.

SELECT Name, Population FROM city
WHERE population BETWEEN 180000 AND 190000;
```

Name	Population
Herat	186800
Batna	183377
Santiago del Estero	189947
Mymensingh	188713
Liège	185639
São Leopoldo	189258
Marília	188691
São Carlos	187122

```sql
-- Select all the data from countrylanguage of
three countries USA, Australia, and India.

SELECT * FROM countrylanguage
WHERE CountryCode IN ('USA', 'AUS', 'IND');
```

CountryCode	Language	IsOfficial	Percentage
IND	Punjabi	F	2.8
IND	Tamil	F	6.3
IND	Telugu	F	7.8
IND	Urdu	F	5.1
USA	Chinese	F	0.6
USA	English	T	86.2
USA	French	F	0.7
USA	German	F	0.7

ORDER BY

- Use **ORDER BY** to arrange data in ascending order (default)

```
-- Select all data from the 'city' table order
by population in ascending order.

SELECT * FROM city
  ORDER BY Population;
```

ID	Name	CountryCode	District	Population
2912	Adamstown	PCN	–	42
2317	West Island	CCK	West Island	167
3333	Fakaofo	TKL	Fakaofo	300
3538	Città del Vaticano	VAT	–	455
2316	Bantam	CCK	Home Island	503
2728	Yaren	NRU	–	559
62	The Valley	AIA	–	595
2805	Alofi	NIU	–	682
1791	Flying Fish Cove	CXR	–	700
2806	Kingston	NFK	–	800

- Use **ORDER BY** to arrange data in descending order

```
-- Select all data from the 'city' table order
by population in descending order.

SELECT * FROM city
  ORDER BY Population
  DESC;
```

ID	Name	CountryCode	District	Population
1024	Mumbai (Bombay)	IND	Maharashtra	10500000
2331	Seoul	KOR	Seoul	9981619
206	São Paulo	BRA	São Paulo	9968485
1890	Shanghai	CHN	Shanghai	9696300
939	Jakarta	IDN	Jakarta Raya	9604900
2822	Karachi	PAK	Sindh	9269265
3357	Istanbul	TUR	Istanbul	8787958
2515	Ciudad de México	MEX	Distrito Federal	8591309
3580	Moscow	RUS	Moscow (City)	8389200
3793	New York	USA	New York	8008278

ORDER BY

- **Sort by expressions**
  - Sometimes, you may need to sort by some other criterion

```
-- Select Name and IndepYear from country
-- and order by last letter of name
-- (order by last letter means the first
-- letter from right).
SELECT Name, IndepYear
  FROM country
  ORDER BY RIGHT(Name, 1);
```

Name	IndepYear
Samoa	1962
Yugoslavia	1918
South Africa	1910
Zambia	1964
Central African Republic	1960
Czech Republic	1993
Dominican Republic	1844
Switzerland	1499

- **Sort by numeric placeholders**
  - We can reference columns by their position rather than name to sort

```
-- Select all data from city in
-- ascending order of 5th and 3rd column.
SELECT * FROM city ORDER BY 5, 3;
```

ID	Name	CountryCode	District	Population
2912	Adamstown	PCN	–	42
2317	West Island	CCK	West Island	167
3333	Fakaofo	TKL	Fakaofo	300
3538	Città del Vaticano	VAT	–	455
2316	Bantam	CCK	Home Island	503
2728	Yaren	NRU	–	559
62	The Valley	AIA	–	595
2805	Alofi	NIU	–	682

NULL

- **NULL** values are appropriate:
  - When values are not available or applicable
  - When values are not yet known, but will be added later
  - When values are undefined
- To test whether an expression is null, use **IS NULL** or **IS NOT NULL** operators

NULL

```sql
-- Select Name and IndepYear from country
-- where IndepYear is null.
SELECT Name, IndepYear FROM country
WHERE IndepYear IS NULL;
```

Name	IndepYear
Aruba	NULL
Anguilla	NULL
Netherlands Antilles	NULL
American Samoa	NULL
Antarctica	NULL
French Southern territories	NULL
Bermuda	NULL
Bouvet Island	NULL

```sql
-- Select Name and IndepYear from country
-- where IndepYear is  not null.
SELECT Name, IndepYear FROM country
WHERE IndepYear IS NOT NULL;
```

Name	IndepYear
Afghanistan	1919
Angola	1975
Albania	1912
Andorra	1278
United Arab Emirates	1971
Argentina	1816
Armenia	1991
Antigua and Barbuda	1981

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Summarize the use of query clauses	✔
Apply conditional filtering of data	✔
Query multiple tables - introduction to joins	
Define various joins and when to implement each type	
Understanding set operations	
Understanding and applying string functions	
Understanding and applying numeric functions	

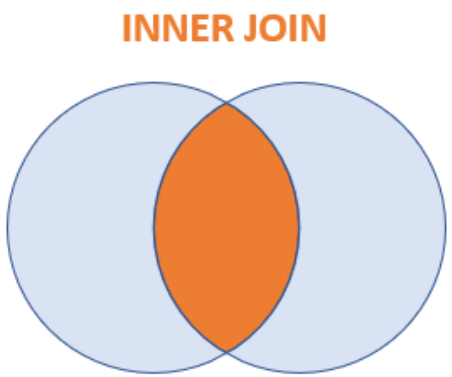
Introduction to joins

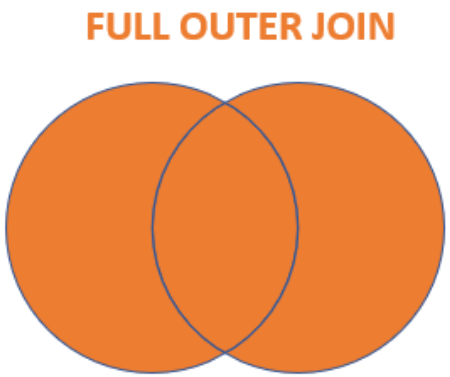
- Until now, we queried only one table at a time
- What if we want to fetch data from multiple columns?
- We use **JOIN** to query multiple tables by combining them
- Join can be done if there is a common column in the joining tables called **joining attribute**

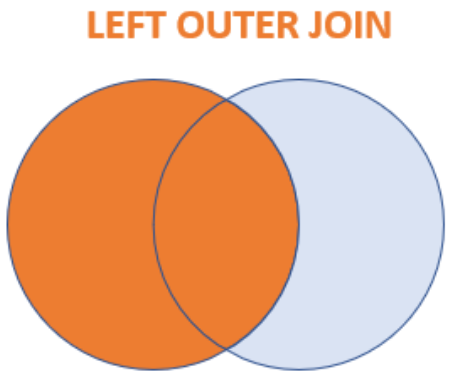
Note: the joining attribute will be a primary key in one table and foreign key in the other tables

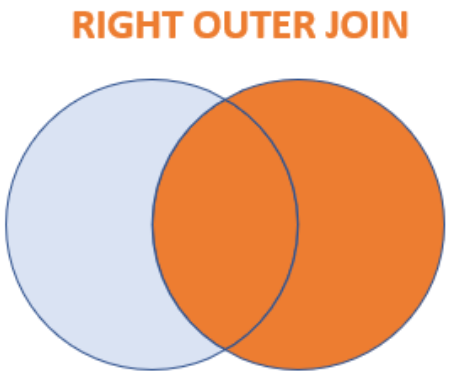
Join type	Description
Inner join	Returns records that have matching values in both the tables
Left outer join	Returns all records from the left table and matched records from the right table
Right outer join	Returns all records from the right table and matched records from the left table
Full outer join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

Types of joins









Note: full outer join is not supported in current version of MySQL

- **JOIN** uses of the concept of a **table alias**

Create shipping database

- We will use a different database to understand joins better
- First, create a new database `shipping`

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

- Import data into 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,              -- select customer_id from customer
cu.customer_name, cu.customer_city,                         -- select attributes from customer
o.customer_id AS orders_customer_id,                        -- select customer_id from orders
o.orders_id, o.item_id                                      -- select attributes from orders
FROM customer AS cu                                         -- alias customer as cu
INNER JOIN orders AS o ON cu.customer_id = o.customer_id;   -- perform inner join, alias orders, and
                                                            -- specify joining attribute
```

INNER JOIN - cont'd

Customer table

customer_id	customer_name	customer_age	customer_city
DC101	Jack	36	Washington DC
LA675	Sara	19	Falls Church
MD109	Monica	28	Baltimore
MD607	Nick	40	Columbia

Orders table

orders_id	customer_id	item_id	quantity	price
1100	DC101	1	2	2.86
1110	DC101	2	1	2.45
2330	MD109	5	5	3.56
4450	VG565	5	5	3.56

Result

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

LEFT OUTER JOIN

```
SELECT cu.customer_id AS customer_customer_id,          -- select customer_id from customer
cu.customer_name, cu.customer_city,                     -- select attributes from customer
o.customer_id AS orders_customer_id,                    -- select customer_id from orders
o.orders_id, o.item_id                                  -- select attributes from orders
FROM customer AS cu                                     -- alias customer as cu
LEFT JOIN orders AS o ON cu.customer_id = o.customer_id; -- perform left join alias orders and
                                                        -- define joining attribute
```

LEFT OUTER JOIN - cont'd

Customer table

customer_id	customer_name	customer_age	customer_city
DC101	Jack	36	Washington DC
LA675	Sara	19	Falls Church
MD109	Monica	28	Baltimore
MD607	Nick	40	Columbia

Orders table

orders_id	customer_id	item_id	quantity	price
1100	DC101	1	2	2.86
1110	DC101	2	1	2.45
2330	MD109	5	5	3.56
4450	VG565	5	5	3.56

Result

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

RIGHT OUTER JOIN

```
SELECT cu.customer_id AS customer_customer_id,              -- select customer_id from customer
cu.customer_name, cu.customer_city,                         -- select attributes from customer
o.customer_id AS orders_customer_id,                        -- select customer_id from orders
o.orders_id, o.item_id                                      -- select attributes from orders
FROM customer AS cu                                         -- alias customer as cu
RIGHT JOIN orders AS o ON cu.customer_id = o.customer_id;   -- perform right join, alias orders and
                                                            -- define joining attribute
```

RIGHT OUTER JOIN - cont'd

Customer table

customer_id	customer_name	customer_age	customer_city
DC101	Jack	36	Washington DC
LA675	Sara	19	Falls Church
MD109	Monica	28	Baltimore
MD607	Nick	40	Columbia

Orders table

orders_id	customer_id	item_id	quantity	price
1100	DC101	1	2	2.86
1110	DC101	2	1	2.45
2330	MD109	5	5	3.56
4450	VG565	5	5	3.56

Result

customer_customer_id	customer_name	customer_city	orders_customer_id	orders_id	item_id
DC101	Jack	Washington DC	DC101	1100	1
DC101	Jack	Washington DC	DC101	1110	2
MD109	Monica	Baltimore	MD109	2330	5
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** clause and **ON** clause
- It makes use of the **WHERE** clause and `'='` operator
- This is the most advanced version of join to write optimized query
- We will use the `world` database from now

ANSI join syntax - cont'd

```
-- 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
ABW	Aruba	Dutch
ABW	Aruba	English
ABW	Aruba	Papiamento
ABW	Aruba	Spanish
AFG	Afghanistan	Balochi
AFG	Afghanistan	Dari
AFG	Afghanistan	Pashto
AFG	Afghanistan	Turkmenian

Joining more than two tables

- Join can be done on more than two tables
- There should be a common column at least across two tables
- Let's join all three tables of the world database

```
-- Fetch the city name, country name and the lauguage 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
Oranjestad	Aruba	Dutch
Oranjestad	Aruba	English
Oranjestad	Aruba	Papiamento
Oranjestad	Aruba	Spanish
Kabul	Afghanistan	Balochi
Kabul	Afghanistan	Dari
Kabul	Afghanistan	Pashto
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
```

city_1	city_2	CountryCode
Kabul	Kabul	AFG
Kabul	Qandahar	AFG
Kabul	Herat	AFG
Kabul	Mazar-e-Sharif	AFG
Qandahar	Kabul	AFG
Qandahar	Qandahar	AFG
Qandahar	Herat	AFG
Qandahar	Mazar-e-Sharif	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
▶	Amsterdam	NLD	Kabul	AFG
	Rotterdam	NLD	Kabul	AFG
	Haag	NLD	Kabul	AFG
	Utrecht	NLD	Kabul	AFG
	Eindhoven	NLD	Kabul	AFG
	Tilburg	NLD	Kabul	AFG
	Groningen	NLD	Kabul	AFG
	Breda	NLD	Kabul	AFG

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Summarize the use of query clauses	✔
Apply conditional filtering of data	✔
Query multiple tables - introduction to joins	✔
Define various joins and when to implement each type	✔
Understanding set operations	
Understanding and applying string functions	
Understanding and applying numeric functions	

Set operations

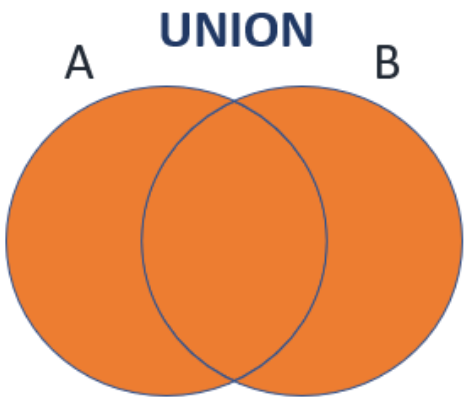
- Until now, we saw how to combine tables using various types of **join**
- Now we will discuss the method to combine tables using **set** operators

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

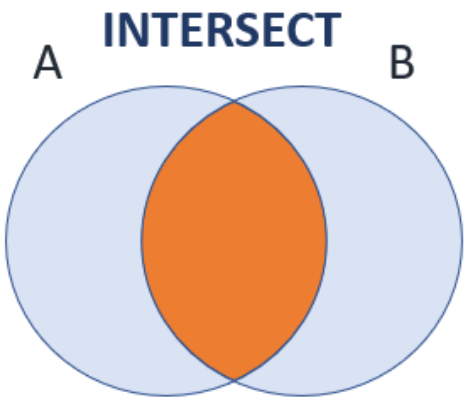
Note: INTERSECT & EXCEPT operations are not supported in the current version of MySQL

Set operations

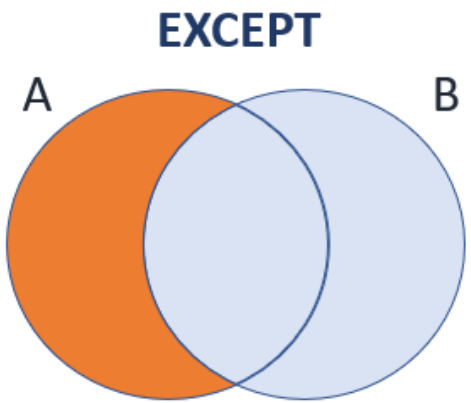
- **Rules for set operations:**
  - Both datasets must have the **same number** of columns
  - The **data type** of each column across the two datasets must be the same



A U B



A ∩ B



A - B

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
▶	ABW
	AFG
	AGO
	AIA
	ALB
	AND
	ANT
	ARE

✔	15	11:11:24	SELECT code from country UNION SELECT CountryCode from countrylanguage	239 row(s) returned
✔	16	11:11:34	SELECT code from country UNION ALL SELECT CountryCode from countrylanguage	1223 row(s) returned

UNION ALL

```
SELECT code from country                    -- first dataset
UNION ALL                                   -- union all operation
SELECT CountryCode from countrylanguage;    -- second dataset
```

	code
▶	ABW
	AFG
	AGO
	AIA
	ALB
	AND
	ANT
	ARE

✔	15	11:11:24	SELECT code from country UNION SELECT CountryCode from countrylanguage	239 row(s) returned
✔	16	11:11:34	SELECT code from country UNION ALL SELECT CountryCode from countrylanguage	1223 row(s) returned

Module completion checklist

Objective	Complete
Summarize the use of query clauses	✔
Apply conditional filtering of data	✔
Query multiple tables - introduction to joins	✔
Define various joins and when to implement each type	✔
Understanding set operations	✔
Understanding and applying string functions	
Understanding and applying numeric functions	

LIKE - pattern match

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

LIKE - pattern match

- The wildcard characters can 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 - pattern match

```sql
-- 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	Kabul	AFG	Kabol	1780000
152	Khulna	BGD	Khulna	663340
159	Jessore	BGD	Khulna	139710
549	Ouagadougou	BFA	Kadiogo	824000
608	Cairo	EGY	Kairo	6789479
634	Kafr al-Shaykh	EGY	Kafr al-Shaykh	124819
644	Disuq	EGY	Kafr al-Shaykh	91300
654	Barcelona	ESP	Katalonia	1503451
668	L´Hospitalet de Llobregat	ESP	Katalonia	247986

```sql
-- 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
146	Sumqayit	AZE	Sumqayit	283000
929	Port-au-Prince	HTI	Ouest	884472
930	Carrefour	HTI	Ouest	290204
931	Delmas	HTI	Ouest	240429
950	Padang	IDN	Sumatera Barat	534474
963	Mataram	IDN	Nusa Tenggara Barat	306600
1029	Ahmedabad	IND	Gujarat	2876710
1035	Surat	IND	Gujarat	1498817
1040	Vadodara (Baroda)	IND	Gujarat	1031346

String functions

- We are already familiar with displaying the string data in **UPPER** and **LOWER** case
- MySQL has a large collection of string functions to operate on string data types
- String functions: ***https://dev.mysql.com/doc/refman/8.0/en/string-functions.html***
- Here are some of the most commonly used functions:

Function	Description
LENGTH	Finds the length of the string
CONCAT or CONCAT_WS	Concatenates two or more string expressions together
REPLACE	Replaces all occurrences of a substring/character in a specified string
REVERSE	Reverses a string
SUBSTR or SUBSTRING	Extracts a substring
LOCATE	Finds the position of a substring/character in 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 CountryLanuage table.
SELECT Language, LENGTH(Language) AS
length_language FROM countrylanguage;
```

language	length_language
Dutch	5
English	7
Papiamento	10
Spanish	7
Balochi	7
Dari	4
Pashto	6
Turkmenian	10
Uzbek	5

- 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	Kabul-Kabol	AFG
2	Qandahar-Qandahar	AFG
3	Herat-Herat	AFG
4	Mazar-e-Sharif-Balkh	AFG
5	Amsterdam-Noord-Holland	NLD
6	Rotterdam-Zuid-Holland	NLD
7	Haag-Zuid-Holland	NLD
8	Utrecht-Utrecht	NLD
9	Eindhoven-Noord-Brabant	NLD

REPLACE

- To replace characters in a string, use the **REPLACE** function
  - `REPLACE(string, from_string, to_string)`

```
-- Replace the country code 'AFG' with 'AF'
-- in the city table.
SELECT REPLACE(countrycode, 'AFG', 'AF')
AS countrycode from city;
```

countrycode
ABW
AF
AF
AF
AF
AGO
AGO
AGO
AGO

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;
```

region_code
Car
Sou
Cen
Car
Sou
Sou
Car
Mid
Sou

- 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;
```

position_of_a	district
2	Kabol
2	Qandahar
4	Herat
2	Balkh
11	Noord-Holland
10	Zuid-Holland
10	Zuid-Holland
0	Utrecht
9	Noord-Brabant

STRCMP

- To compare two strings, use the **STRCMP** function
- The function returns:
  - `0` if both strings are the same
  - `-1` if the first string is smaller than the second string
  - `+1` if the first string is larger than the second string
  - `NULL` if one or more of the strings is null

```
-- Compare the columns gnp and gnpold in the country table.
-- Label the new column as compared.
SELECT STRCMP(GNP, GNPOld) AS compared FROM country;
```

	compared
	-1
	NULL
	-1
	1
	1
	-1
	-1
	NULL
	1

ENUM

- MySQL uses a special data type called **ENUM**
- **ENUM** is an enumerated list
- It is a string object whose value is chosen from the list of permitted values defined at the time of column creation
- Our `world` database has a column `continent` in the `country` table of **ENUM** datatype

```
-- Explore country table.
DESC country;
```

Field	Type	Null	Key	Default
Code	char(3)	NO	PRI	
Name	char(52)	NO		
Continent	enum('Asia','Europe','North America','Africa','Oceania','Antarctica','South America')	NO		Asia
Region	char(26)	NO		

Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Summarize the use of query clauses	✔
Apply conditional filtering of data	✔
Query multiple tables - introduction to joins	✔
Define various joins and when to implement each type	✔
Understanding set operations	✔
Understanding and applying string functions	✔
Understanding and applying numeric functions	

Numeric functions

- We already saw how to perform basic mathematic calculations on numeric data in MySQL
- There are built-in functions that can be used for mathematical calculations
- Numeric functions: ***https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html***
- Here are some most commonly used functions:

Function	Description
MOD	Finds the remainder when one number is divided by another
POW	Finds the power of a number
SQRT	Finds the square root of a number
LOG	Finds the log of a number
CEIL	Finds the ceiling of a number
FLOOR	Finds the floor 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
7	443727
8	222518
7	183377
5	179055
6	153106
7	128747
1	128281
4	127284
2	117162

- To find the power of a number raised by another number, use the **POW** function
  - `POW(m, n)` returns `m` raised to `n`th 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
7189057	193.00
2.77282601924329e17	652090.00
1.937697051563e18	1246700.00
884736	96.00
23758712844992	28748.00
102503232	468.00
512000000	800.00
584277056000000	83600.00
2.1494227414464e19	2780400.00

SQRT & LOG

- To find the square root of a number, use **SQRT** function

```
-- Find the square root of the surface area.
SELECT SQRT(SurfaceArea)
AS Surface_area_squareroot, SurfaceArea
FROM country;
```

Surface_area_squareroot	SurfaceArea
13.892443989449804	193.00
807.5208975624098	652090.00
1116.5572085656875	1246700.00
9.797958971132712	96.00
169.55235179731363	28748.00
21.633307652783937	468.00
28.284271247461902	800.00
289.1366458960192	83600.00
1667.4531477675766	2780400.00

- To find the log of a number, use the **LOG** function

```
-- Find the log of the gnp as
-- gnp_log from country.
SELECT LOG(GNP) AS GNP_log, GNP FROM country;
```

GNP_log	GNP
6.71901315438526	828.00
8.695506726812653	5976.00
8.802071336535285	6648.00
4.146304313224639	63.20
8.07246736935477	3205.00
7.396335293800808	1630.00
7.57095858316901	1941.00
10.544446301350487	37966.00
12.737400651706617	340238.00

CEIL & FLOOR - control number precision

- To find the ceiling of a number, use the **CEIL** function

```
-- Find the ceil of the life expectancy
-- from the country table.
SELECT CEIL(LifeExpectancy), LifeExpectancy
FROM country;
```

	CEIL(LifeExpectancy)	LifeExpectancy
▶	79	78.4
	46	45.9
	39	38.3
	77	76.1
	72	71.6
	84	83.5
	75	74.7
	75	74.1
	76	75.1

- To find the floor of a number, use the **FLOOR** function

```
-- Find the floor of the life expectancy
-- from the country table.
SELECT FLOOR(LifeExpectancy), LifeExpectancy
FROM country;
```

	FLOOR(LifeExpectancy)	LifeExpectancy
▶	78	78.4
	45	45.9
	38	38.3
	76	76.1
	71	71.6
	83	83.5
	74	74.7
	74	74.1
	75	75.1

TRUNCATE & ROUND - control number precision

- 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;
```

	TRUNCATE(GNP/1000, 2)	GNP
▶	0.82	828.00
	5.97	5976.00
	6.64	6648.00
	0.06	63.20
	3.20	3205.00
	1.63	1630.00
	1.94	1941.00
	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;
```

	ROUND(GNP/1000, 2)	GNP
▶	0.83	828.00
	5.98	5976.00
	6.65	6648.00
	0.06	63.20
	3.20	3205.00
	1.63	1630.00
	1.94	1941.00
	37.97	37966.00

UNSIGNED INT & ABS

- We know that numeric data can either be signed or unsigned
- Default `int` type is signed
- Unsigned integers can be created by specifying **INT UNSIGNED** while creating the table
- We can use the **ABS** function to get absolute value of the number during data manipulation

```
SELECT (GNP-GNPOld), GNP, GNPOld FROM country;
```

	(GNP-GNPOld)	GNP	GNPOld
	781.00	8255.00	7474.00
	-1100.00	97477.00	98577.00
	16383.00	1161755.00	1145372.00
	149.00	6871.00	6722.00
	475.00	7526.00	7051.00
	-405596.00	3787042.00	4192638.00
	992.00	24375.00	23383.00
	-1024.00	9217.00	10241.00
	-141.00	1626.00	1767.00

```
SELECT ABS(GNP-GNPOld), GNP, GNPOld FROM country;
```

	ABS(GNP-GNPOld)	GNP	GNPOld
	781.00	8255.00	7474.00
	1100.00	97477.00	98577.00
	16383.00	1161755.00	1145372.00
	149.00	6871.00	6722.00
	475.00	7526.00	7051.00
	405596.00	3787042.00	4192638.00
	992.00	24375.00	23383.00
	1024.00	9217.00	10241.00
	141.00	1626.00	1767.00

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Summarize the use of query clauses	✔
Apply conditional filtering of data	✔
Query multiple tables - introduction to joins	✔
Define various joins and when to implement each type	✔
Understanding set operations	✔
Understanding and applying string functions	✔
Understanding and applying numeric functions	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today you will:
  - Load your dataset into your workbench
  - Perform data manipulation in MySQL for analysis
  - Find interesting insights from your data
  - Practice working with string and numeric data types

This completes our module

Congratulations!