Starter Activity.

Why are relational databases useful in Data Science?

TechTalent Academy

Data Science Course

Introduction to SQL

Lesson Objectives.

- Introduction into SQL
- Creating a relational database
- Manipulating Data
- Joining tables

SQL.

- Structured Query Language
- A programming language used for storing, manipulating, and retrieving data from a relational database

SQL.

- Nowadays SQL is used by companies to manage data from a relational database management system (RDBMS)

- RDMS key components:

1. System administrator: user authentication, database monitoring activity

2. Schema: contains databases, tables, indexes etc.

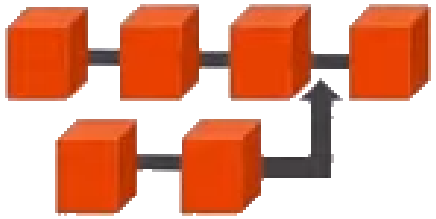
Examples of licensed RDBMS software

Relational vs non-relational databases.

SQL

Document based
Represents hierarchical relationship using a single record – **Schema (collection of databases) – Structured data stored into tables having rows and columns**

MongoDB
(covered in week 4!)



No Schema
No relation between documents - Mostly used for **unstructured data** (JSON format)

Main Data Manipulation Commands.

- Adding table rows
- Saving table changes
- Listing table rows
- Updating table rows
- Restoring table contents
- Deleting table rows
- Inserting table rows with a select subquery

Datatypes in SQL.

techtalent academy

characters

numbers

Type	Value
CHAR()	A FIXED length string (can contain letters, numbers, and special characters). The size parameter() specifies the column length in characters - can be from 0 to 255. Default is 1. E.g. CHAR(10) will allow 10 characters. CHAR stores a fixed size of bytes based on what you specify (). If you know the data will be a fixed amount of characters then choose this.
VARCHAR()	A string of variable length up to 255 characters long (can contain letters, numbers, and special characters). The storage requirements depends on the string length. This is the better choice for names.
INT	An integer range is from -2147483648 to 2147483647.
DECIMAL	A floating point number that can specify the number permissible digits. E.g. (5,2) permits – 999.99 to 999.99. (0,24)
DOUBLE	A long double-precision floating point number. (25,53)
DATE	A date in the YY-MM-DD format.
TIME	A time in the HH:MM:SS format.
DATETIME	Combination of the date and time format. Date first and then time. YY-MM-DD HH:MM:SS
YEAR	A year 1901-2155 in either YY of YYYY format.
TEXT	A string up to 65,535 characters long.
BLOB	A binary type for variable data.
ENUM	A single string value from defined list. For example, ENUM(“Dr”, “Mr”, “Mrs”).
SET	A single/multiple strings value from defined list. For example, ENUM(“Red”, “Green”, “Yellow”).
BOOL	Equal to Boolean: false or true

Relational Database Keywords.

Database: all the of the table structure and records saved.

Table: holds all of the data and information of the databases. This is organised in rows and columns.

Fieldnames: column headings (variable names)

Datatypes: assigned to fields to store the correct type of data

Records: is when all the fields are completed for one entity

What make a database relational?

- **Unique identifiers and constraints:**
  - added to the database to create a link between the different tables
  - provide data consistency and accuracy (avoid data redundancy and errors)

Unique Identifiers – Primary Key.

Tables must have a field which is unique to that record. It is usually called a primary key.

Car_ID	Make	Model	Colour	Price
1	VW	Golf	Grey	£23,800
2	BMW	5 Series	Red	£36,525
3	Mercedes	C Class	Silver	£28,350
4	Ford	Focus	Blue	£24,435
5	Audi	Q5	Black	£39,000

An ID field is usually used as a primary key since it contains unique information

Constraints – Foreign Keys.

Foreign keys are created when a primary key from one table appears in another table. For example, in the Customer table below, **Customer_ID** is the **primary key**. The **primary key** in the Order table is the **Order_ID**.

However, the Customer_ID field also appears in the Order table. **Customer_ID is a foreign key within the Order table.**

primary key

Customer Table

Customer_ID	First_Name	Surname	House_Number	Address	Postcode
1	Suzannah	Downie	1	Database Road	FP12 1DE
2	Georgina	Stanley	65	String Lane	LA76 8ZS
3	Andy	Gray	42	Data Street	SQ57 4TP
4	Satinder	Sohal	34	Boolean Way	TP67 9QO
5	Joe	Bloggs	119	Integer Avenue	AC12 7PU

primary key foreign key

Order Table

Order_ID	Customer_ID	Order_Date	Dispatch_Date	Due_Date
1	5	11/04/2021	13/05/2021	14/05/2021
2	43	15/04/2021	13/05/2021	14/05/2021
3	12	20/04/2021	13/05/2021	14/05/2021
4	5	20/04/2021	13/05/2021	14/05/2021

Exploring available databases on MySQL server.

Write the command **SHOW DATABASES;** in your SQL script

The right panel on VS Code display the different databases available

TTA.session.sql

Detach file from TTA | ▷ Run on active connection |

```
1  SHOW DATABASES;
```

TTA: SHOW DATABASES;

Database

abc Filter...

cars
information_schema
mysql
performance_schema
school
sys
Uni

Click on run active connection to run the command

Creating a new Database.

TTA.session.sql

Detach file from TTA | ▷ Run on active connection

```
1  CREATE DATABASE;
2  USE TTA;
3
```

Note: when closing a statement in SQL use always a semi colon

Database

abc Filter...

- cars
- information_schema
- mysql
- performance_schema
- school
- sys
- TTA
- Uni

Our TTA database has been created

Creating a Table within your Database: Structure.

Now that you have created your database you can start working on the structure by creating a **table** with fieldnames/variables with the correct datatypes. Structure to create a table is:

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
);
```

- The command **CREATE TABLE** is needed, followed by the **name** for the table
- Use the parenthesis **(** to start creating your fieldnames/variables and datatypes
- At the end you must close the parenthesis **)** and add the **;**

Remember syntax is important!

Creating a Table within your Database.

techtalent academy

Once you run the code - view the schemas to see the whole setup.

The code to see the structure of the table is: **EXPLAIN Studentscoring;**

TTA.session.sql

Detach file from TTA | Run on active connection |

```sql
CREATE DATABASE TTA;

USE TTA;

CREATE TABLE Studentscoring(
  ID int (10),
  Student varchar (25),
  Score int (10),
  Age int (10),
  PRIMARY KEY (ID)
);

EXPLAIN Studentscoring;
```

Variables info

Field	Type	Null	Key
ID	int	NO	PRI
Student	varchar(25)	YES	
Score	int	YES	
Age	int	YES	

Use the following online lite SQL server if you cannot use VS code yet: https://www.programiz.com/sql/online-compiler/

Task: Creating a database.

- Create your own database
- Create a table which contains 3 columns

Create a table and insert data

General syntax to create an empty table

```
CREATE TABLE table_name(
    column1 datatype (# of character accepted),
    column2 datatype (# of character accepted),
    column3 datatype (# of character accepted),
    PRIMARY KEY (column_name)
);
```

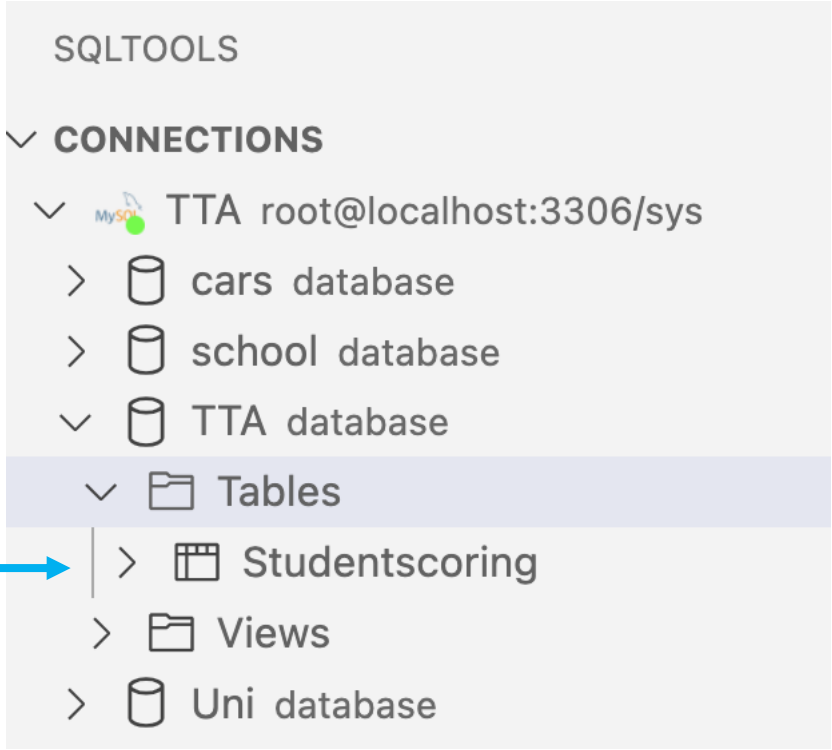
```
create table Studentscoring (
    ID int (10),
    Student varchar(25),
    Score int(10),
    Age int(10),
    primary key(ID)
);
```

Each variable (column name) is being added separated by a colon

Don't forget to close your whole statement with a semi colon

Create a table and insert data cont.

Our table named **Studentscoring** has been well inserted in the Tables folder in the TTA database, but the table is empty and has no data in it yet



Create a table and insert data cont.

1-Table created in our database:

```
create table Studentscoring (
    ID int (10),
    Student varchar(25),
    Score int(10),
    Age int(10),
    primary key(ID)
);
```

2-Insert records into it:

```
insert into Studentscoring
    (ID, Student, Score, Age)
values (1, 'Chris', 78, 23),
       (2, 'Charlotte', 92, 22),
       (3, 'Caleb', 92, 24),
       (4, 'Chantelle', 88, 23)
```

output

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	92	22
3	Caleb	92	24
4	Chantelle	88	23

The table can be created only once!

Main sources of errors.

- **Create <u>databases</u> and <u>tables</u> only once!** If you run the code twice in the script to create again the same table or the same database you will get an error message!

- Don't forget to end your block of code with a semi colon

- Sometimes VS Code loose the database connexion when running queries. To fix the issue use the command **USE database_name** to reconnect to your database

Task: Inserting Records.

- Use **INSERT INTO** to insert at least 2 records into your table

- View all the records in your table

SQL syntax: main clauses (select, from, where).

Main syntax

```
SELECT column-name
FROM table-name
WHERE condition;
```

Table: Studentscoring

ID	Student	Score
1	Chris	78
2	Charlotte	92
3	Caleb	92
4	Chantelle	88

Example

```
SELECT Score
FROM Studentscoring
WHERE Score >78;
```

Conditions and Logical operators.

- Conditions are used to select and filter data of interest

- The WHERE clause is used followed by a logical operator:

Logical operator	description
=	equal
<>	Not equal
<	greater than
>	less than
>=	greater than or equal to
<=	less than or equal to

Working With queries: data selection: Selecting one column

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	92	22
3	Caleb	92	24
4	Chantelle	88	23

```
SELECT Score
FROM Studentscoring
```

output

Score
78
92
92
88

Working With queries: data selection.

Selecting 2 columns

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	92	22
3	Caleb	92	24
4	Chantelle	88	23

```
SELECT SCORE, Age
FROM Studentscoring;
```

output

Score	Age
78	23
92	22
92	24
88	23

Working With queries: data selection.

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	92	22
3	Caleb	92	24
4	Chantelle	88	23

```
SELECT Score
FROM Studentscoring
WHERE Score >78;
```

output

Score
92
92
88

The **WHERE** clause allow us to add some conditions when selecting the data

And & Or logical operator.

When Selecting values you can use more than one criteria to filter results, this is done by using adding an AND or OR to your query

A wild card * is used to select everything from the table

```
SELECT *
FROM Studentscoring
WHERE Score>88 AND Age>23;
```

Try with the OR operator!

Output

ID	Student	Score	Age
3	Caleb	92	24

And & Or Logical operator.

- **AND** as well as **OR** follow specific logic when applied
- The **AND** operator requires **BOTH** conditions to be true in order to return a result
- The **OR** function requires **EITHER** to be true to return a result
- Using the query

  SELECT * FROM Table

  WHERE A = 1

  ?

  B = X
- Substituting our “?” for an AND or an OR will give different results

Table		
A	B	C
1	X	Blue
2	Y	Red
1	Y	Yellow
2	X	Green

Results (AND)		
A	B	C
1	X	Blue

Results (OR)		
A	B	C
1	X	Blue
1	Y	Yellow
2	X	Green

Working With queries: data modification.

```
UPDATE Studentscoring
SET Score = 90
WHERE Student="Charlotte";
```

UPDATE and **SET** keywords are used to modify records in the table

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	92	22
3	Caleb	92	24
4	Chantelle	88	23

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	90	22
3	Caleb	92	24
4	Chantelle	88	23

Her score has been modified

Working With queries: data modification.

```
UPDATE Studentscoring
SET Score = 89, Age= 21
WHERE Student="Caleb";
```

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	90	22
3	Caleb	92	24
4	Chantelle	88	23

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	90	22
3	Caleb	89	21
4	Chantelle	88	23

Caleb's data has been modified

Working With queries: data modification.

```
DELETE FROM Studentscoring
WHERE Student="Caleb";
```

DELETE keyword is used to delete records in the table

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	90	22
3	Caleb	89	21
4	Chantelle	88	23

Studentscoring

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	90	22
4	Chantelle	88	23

Caleb's data have been removed completely

Working With queries: data modification.

```
ALTER TABLE Studentscoring
ADD Country varchar(25);
```

ALTER keyword modify records in the table

Studentscoring

ID	Student	Score	Age	Country
1	Chris	78	23	
2	Charlotte	90	22	
4	Chantelle	88	23	

New column named Country added

Working With queries: data modification.

Modifying filed names: when we insert data into tables, it is a good practice to add only non null values. To add this constraint to our table, we can alter the table datatype using the keyword **ALTER**

```
ALTER TABLE Studentscoring
MODIFY COLUMN Student varchar (25) NOT NULL;

EXPLAIN Studentscoring;
```

Field	Type	Null	Key
abc Filter...	abc Filter...	abc Filter...	abc Filter.
ID	int	NO	PRI
Student	varchar(25)	NO	
Score	int	YES	
Age	int	YES	

This has been changed from YES TO NO

Note: NOT NULL keyword can be added initially when creating a table after giving datatype variables similarly to here

Working With queries: data modification.

Modifying filed names: the **ALTER** keyword can also be used to drop records from a table.

```
ALTER TABLE Studentscoring
DROP Age;
```

Studentscoring

ID	Student	Score
1	Chris	78
2	Charlotte	92
3	Caleb	92
4	Chantelle	88

Working With queries: data sorting.

```
SELECT * FROM Studentscoring
ORDER BY Student DESC;
```

Sort the data by descending order

ID	Student	Score	Age
1	Chris	78	23
2	Charlotte	90	22
4	Chantelle	88	23

```
SELECT * FROM Studentscoring
ORDER BY Student ASC;
```

Sort the data by ascending order

ID	Student	Score	Age
4	Chantelle	88	23
2	Charlotte	90	22
1	Chris	78	23

Arithmetic Operators.

- Perform operations within parentheses

  (a+b)

- Perform power operations

  a^b

- Perform multiplications and divisions

  a*b or a/b

- Perform additions and subtractions

  a+b or a-b

BODMAS

Working With queries: calculations.

```
SELECT Score, (SUM(Score)/COUNT(Score)) AS AverageScore
FROM Studentscoring
```

Using the **AS** keyword allow us to create a new column name here AverageScore where our calculation is stored

Output

Score	AverageScore
78	85

Working With queries: calculations.

```
SELECT MIN (Score)
FROM Studentscoring;
```

MIN (Score)
78

```
SELECT MIN(Score) AS Lowest
 From Studentscoring;
```

Lowest
78

Using the AS keyword allow us to create a new column name here Lowest where our min value is stored

```
SELECT COUNT(Score)
FROM Studentscoring;
```

COUNT(Score)
3

Working With queries: calculations.

```
SELECT SUM(Score)
FROM StudentScoring;
```

SUM(Score)
350

```
SELECT AVG(Age)
FROM StudentScoring;
```

AVG(Age)
23

```
SELECT COUNT (*)
FROM StudentScoring
WHERE Score >78;
```

COUNT (*)
3

Task: Data Manipulation.

- Practice manipulating your data using some of the examples we have looked at

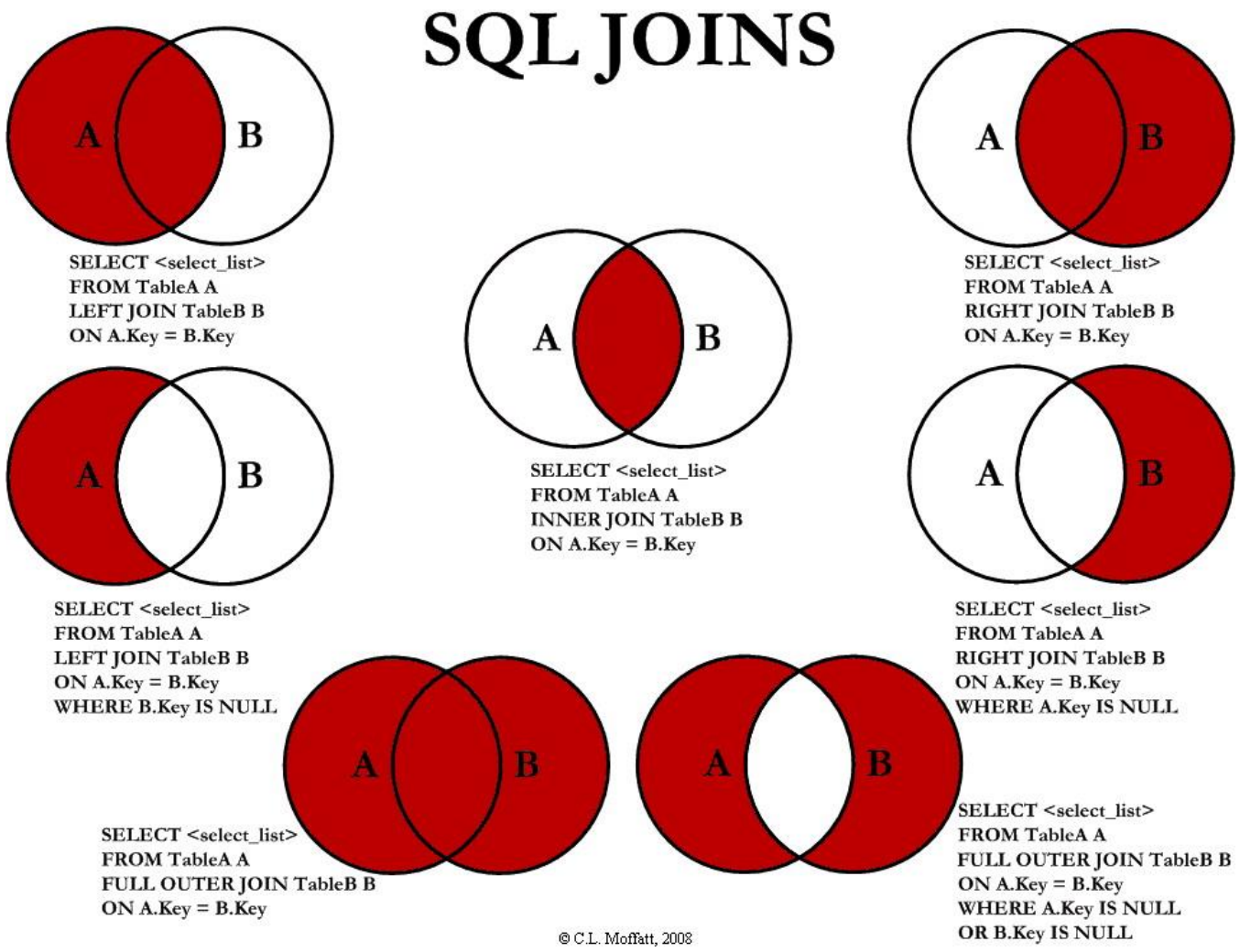
Joining Tables.

- JOIN is performed when data are retrieved from **more than one table at a time**

- JOIN is generally composed of an equality comparison between foreign key and primary key of related tables

Joins.

Note: The Inner join is the most commonly used



INNER JOIN.

Primary keys

Customerinfo

ID_Customerinfo	Country	Gender	Age
9	32	M	UK
2	44	F	Canada
12	36	M	UK
4	56	F	Australia

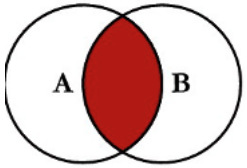
Foreign key

Customer_orders

OrderID	ID_Customerinfo	ItemName	ProductPrice
6333	2	Toaster	17.99
3434	8	Set of 3 Nested Tables	105.99
3433	4	Electric Toothbrush	45.99
1443	3	4 Person Tent	220.99
4678	10	Upright Fan	41.99
5244	22	Henry Hoover	111.99

ID_Customerinfo is in both tables! A join can be performed here

INNER JOIN.



We can join both tables based on similar values contained in ID_Customerinfo, here the ID #2 and #4

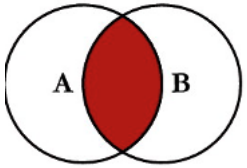
Customerinfo

ID_Customerinfo	Country	Gender	Age
9	32	M	UK
2	44	F	Canada
12	36	M	UK
4	56	F	Australia

Customer_orders

OrderID	ID_Customerinfo	ItemName	ProductPrice
6333	2	Toaster	17.99
3434	8	Set of 3 Nested Tables	105.99
3433	4	Electric Toothbrush	45.99
1443	3	4 Person Tent	220.99
4678	10	Upright Fan	41.99
5244	22	Henry Hoover	111.99

INNER JOIN.



We can join both tables based on similar values contained in both tables, here on ID_Customerinfo #2 and #4

Customerinfo

ID_Customerinfo	Country	Gender	Age
9	32	M	UK
2	44	F	Canada
12	36	M	UK
4	56	F	Australia

Customer_orders

OrderID	ID_Customerinfo	ItemName	ProductPrice
6333	2	Toaster	17.99
3434	8	Set of 3 Nested Tables	105.99
3433	4	Electric Toothbrush	45.99
1443	3	4 Person Tent	220.99
4678	10	Upright Fan	41.99
5244	22	Henry Hoover	111.99

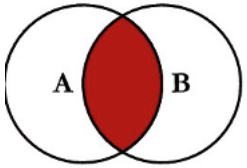
INNER JOIN.

table1

table2

```
SELECT *
FROM Customerinfo
INNER JOIN Customer_orders
ON Customerinfo.ID_Customerinfo = Customer_orders.ID_Customerinfo;
```

On the values that are similar in both tables



SQL script link

Customerinfo

ID_Customerinfo	Country	Gender	Age
9	32	M	UK
2	44	F	Canada
12	36	M	UK
4	56	F	Australia

Customer_orders

OrderID	ID_Customerinfo	ItemName	ProductPrice
6333	2	Toaster	17.99
3434	8	Set of 3 Nested Tables	105.99
3433	4	Electric Toothbrush	45.99
1443	3	4 Person Tent	220.99
4678	10	Upright Fan	41.99
5244	22	Henry Hoover	111.99

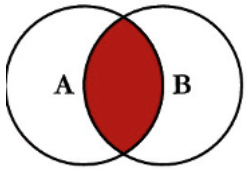
Output

ID_Customerinfo	Country	Gender	Age	OrderID	ID_Customerinfo	ItemName	ProductPrice
2	44	F	Canada	6333	2	Toaster	17.99
4	56	F	Australia	3433	4	Electric Toothbrush	45.99

INNER JOIN.

Same as before but returning few columns

```
SELECT Customer_orders.ID_Customerinfo,Customer_orders.ItemName,
Customerinfo.Gender
FROM Customerinfo
INNER JOIN Customer_orders
ON Customerinfo.ID_Customerinfo = Customer_orders.ID_Customerinfo;
```



Customerinfo

ID_Customerinfo	Country	Gender	Age
9	32	M	UK
2	44	F	Canada
12	36	M	UK
4	56	F	Australia

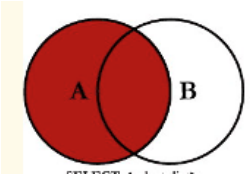
Customer_orders

OrderID	ID_Customerinfo	ItemName	ProductPrice
6333	2	Toaster	17.99
3434	8	Set of 3 Nested Tables	105.99
3433	4	Electric Toothbrush	45.99
1443	3	4 Person Tent	220.99
4678	10	Upright Fan	41.99
5244	22	Henry Hoover	111.99

Output

ID_Customerinfo	ItemName	Gender
2	Toaster	F
4	Electric Toothbrush	F

LEFT JOIN.



We join everything from table 1 (Customerinfo) and the similarities between both tables

```
SELECT *
FROM Customerinfo
LEFT JOIN Customer_orders
ON Customerinfo.ID_Customerinfo = Customer_orders.ID_Customerinfo;
```

Customerinfo

ID_Customerinfo	Country	Gender	Age
9	32	M	UK
2	44	F	Canada
12	36	M	UK
4	56	F	Australia

Customer_orders

OrderID	ID_Customerinfo	ItemName	ProductPrice
6333	2	Toaster	17.99
3434	8	Set of 3 Nested Tables	105.99
3433	4	Electric Toothbrush	45.99
1443	3	4 Person Tent	220.99
4678	10	Upright Fan	41.99
5244	22	Henry Hoover	111.99

Output

ID_Customerinfo	Country	Gender	Age	OrderID	ID_Customerinfo	ItemName	ProductPrice
9	32	M	UK				
2	44	F	Canada	6333	2	Toaster	17.99
12	36	M	UK				
4	56	F	Australia	3433	4	Electric Toothbrush	45.99

Joining Syntax.

- The JOIN is added to the FROM clause of a query

- SYNTAX:

```
SELECT columnlist FROM table1 INNER JOIN table2
        ON table1.column = table2.column
```

- This can still have a where clause to filter information and include more than one table if necessary

```
SELECT columnlist FROM table1 INNER JOIN table2
        ON table1.column = table2.column
        INNER JOIN table3 ON table2.column = table3.column
                WHERE table1.column = criteria
```

Other Joins.

- The INNER JOIN only returns rows for when information is present in both tables and matches the criteria of the join
  - If a foreign key used to connect to another table is not present in that table then no information will be returned.

- Other JOINS include:
  - LEFT JOIN
  - RIGHT JOIN
  - FULL OUTER JOIN

Joins – Left And Right.

- Left and right joins allow us to retrieve records from one table where there are not corresponding key values in the other table.
- This will retrieve **all** of the records from one table and only those that correspond in the other table

LEFT JOIN					
Table A Columns			Table B Columns		
Column1	Column2	Column3	Column1	Column2	Column3
1	xxxxxx	xxxxxx	1	xxxxxx	xxxxxx
2	xxxxxx	xxxxxx	2	xxxxxx	xxxxxx
3	xxxxxx	xxxxxx	3	xxxxxx	xxxxxx
A	xxxxxx	xxxxxx			
B	xxxxxx	xxxxxx			
C	xxxxxx	xxxxxx			

RIGHT JOIN					
Table A Columns			Table B Columns		
Column1	Column2	Column3	Column1	Column2	Column3
1	xxxxxx	xxxxxx	1	xxxxxx	xxxxxx
2	xxxxxx	xxxxxx	2	xxxxxx	xxxxxx
3	xxxxxx	xxxxxx	3	xxxxxx	xxxxxx
			X	xxxxxx	xxxxxx
			Y	xxxxxx	xxxxxx
			Z	xxxxxx	xxxxxx

```
SELECT * FROM TableA
RIGHT JOIN TableB
ON TableA.column1 = TableB.column1
```

```
SELECT * FROM TableA
LEFT JOIN TableB
ON TableA.column1 = TableB.column1
```

Joins – Knowing Left And Right.

- The table which returns all values is determined by the “LEFT” or “RIGHT” keyword in the JOIN when the tables are listed.
  - LEFT = the table before the word JOIN (the one on the left of the join)
  - RIGHT = the table after the word JOIN (the one on the right of the join)
- Fields which do no correspond are returned as **NULL**

LEFT JOIN

Table A Columns			Table B Columns		
Column1	Column2	Column3	Column1	Column2	Column3
1	xxxxxx	xxxxxx	1	xxxxxx	xxxxxx
2	xxxxxx	xxxxxx	2	xxxxxx	xxxxxx
3	xxxxxx	xxxxxx	3	xxxxxx	xxxxxx
A	xxxxxx	xxxxxx			
B	xxxxxx	xxxxxx			
C	xxxxxx	xxxxxx			

```
SELECT * FROM TableA
LEFT JOIN TableB
ON TableA.column1 = TableB.column1
```

RIGHT JOIN

Table A Columns			Table B Columns		
Column1	Column2	Column3	Column1	Column2	Column3
1	xxxxxx	xxxxxx	1	xxxxxx	xxxxxx
2	xxxxxx	xxxxxx	2	xxxxxx	xxxxxx
3	xxxxxx	xxxxxx	3	xxxxxx	xxxxxx
			X	xxxxxx	xxxxxx
			Y	xxxxxx	xxxxxx
			Z	xxxxxx	xxxxxx

```
SELECT * FROM TableA
RIGHT JOIN TableB
ON TableA.column1 = TableB.column1
```

Optional - Sakila Schema.

- **Additonal databses to work on:**
- https://dev.mysql.com/doc/index-other.html
- Open and Run Sakila-schema in VS Code
- Open and Run sakila-data within VS Code

Example Databases

Title	DB Download	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and test/verification suite)	GitHub	View	US Ltr | A4
world database	TGZ | Zip	View	US Ltr | A4
world_x database	TGZ | Zip	View	US Ltr | A4
sakila database	TGZ | Zip	View	US Ltr | A4
airportdb database (large dataset, intended for MySQL on OCI and HeatWave)	TGZ | Zip	View	US Ltr | A4
menagerie database	TGZ | Zip		

Plenary.

techtalent academy

Insert the missing statement to get all the columns from the `Customers` table.

```
        * FROM Customers;
```

Write a statement that will select the `City` column from the `Customers` table.

```
        Customers;
```

Select all records where the `City` column has the value "Berlin".

```
SELECT * FROM Customers
        =          ;
```

What make a database relational.

What make a database relational.