



Data Engineering

LTAT.02.007



Ass Prof. Riccardo Tommasini
Fabiano Spiga, Hassan Eldeeb, Mohamed Ragab

Relational DBs (PostgreSQL)

Lab. 02

```
[~]$ whoami
```

- Mohamed Ragab, PhD Candidate, Starting **3rd** year.
- Masters in Information Systems, “**Trust management in Social Networks**”.
- My PhD Centered around “**Large Graph Processing, Querying and Optimizations**”.
- I also work on Benchmarking Big Data engines (Spark) for Processing Large graph Datasets.



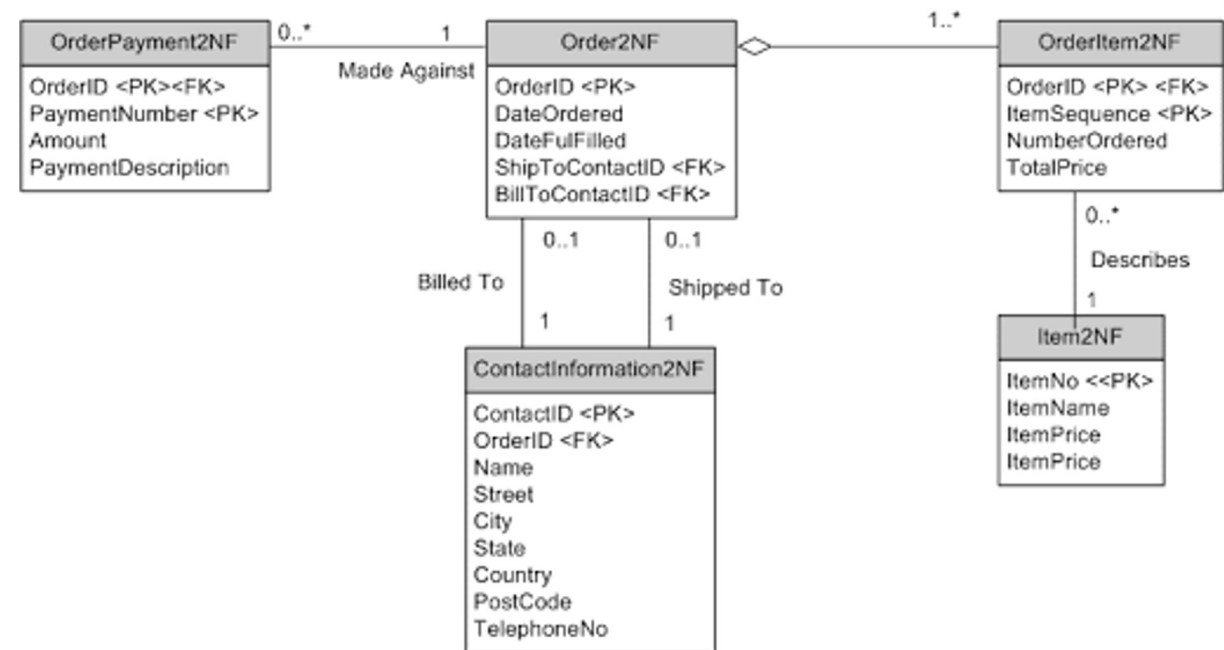
Lab Agenda

- What is the Relational Model?
- What is Relational DBs?!
- What is DBMS With Examples?
- What is PostgreSQL?
- SQL language
 - **Data Definition Lang. DDL**(Create DB, Create Tables,..)
 - **Data Query Lang. DQL** (Selections, Projections, Sorting, Filtering, SQL Joins,..)
 - **Data Manipulation Language DML** (Insert, Update, Delete)



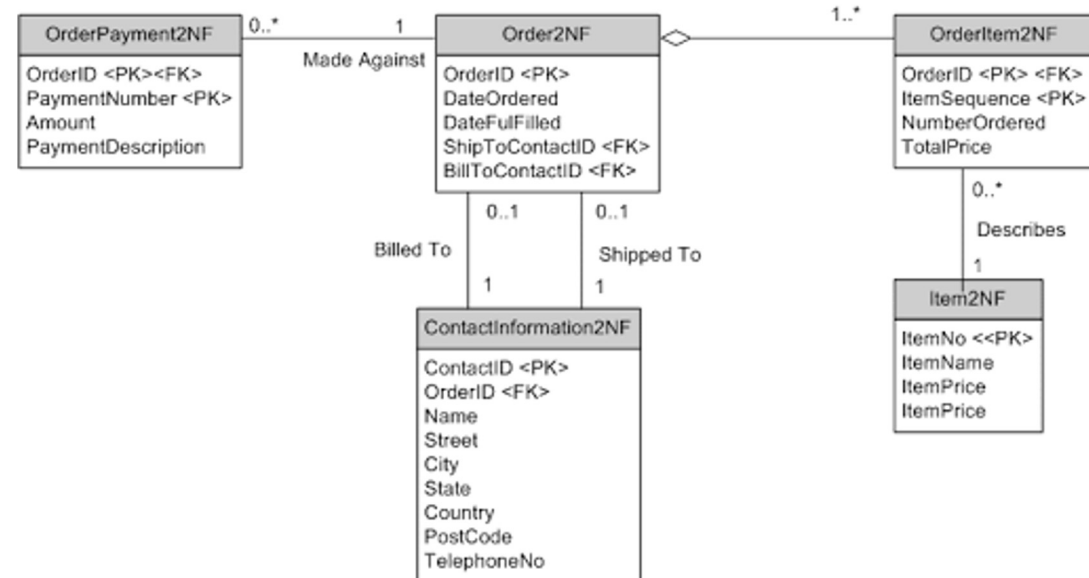
What is Relational Model?

- **RELATIONAL MODEL (RM)** represents the database as a collection of relations.
- A relation is nothing but a “**table**” of values.
- Every row in the table represents a collection of **related data values**.
- These rows in the table denote a **real-world entity** or relationship.



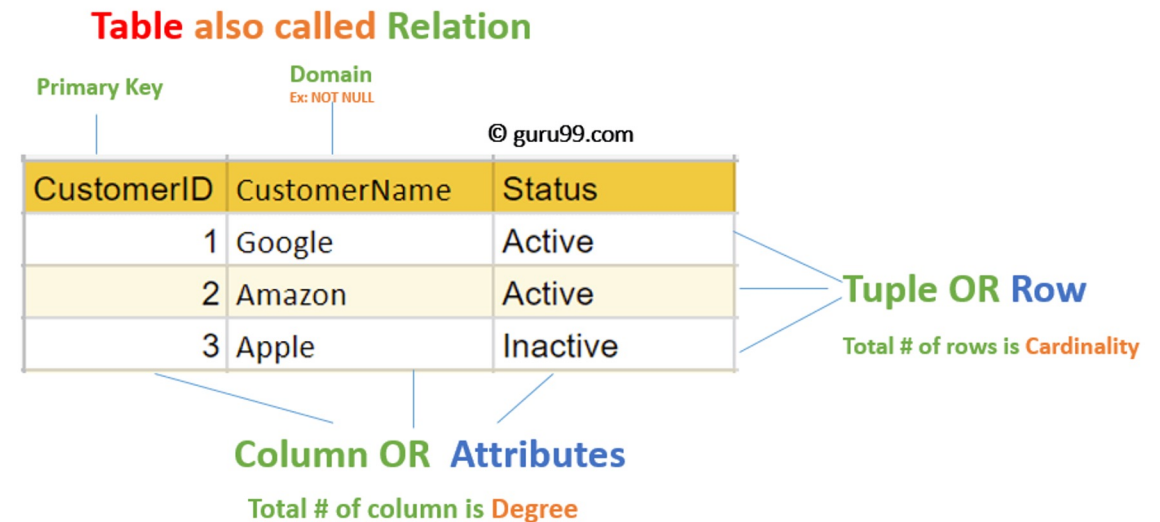
The Relational Model Cont.

- The data are represented as a set of relations.
- In the relational model, data are stored as tables (**Conceptually**).
- However, the **physical storage** of the data is independent of the way the data are logically organized.



Relational Model Concepts

- Tables(Relations)
- Attributes (Fields)
- Tuples (Rows)
- Relation Schema(Table name + attributes)
- Degree (# attributes)
- Cardinality (# of rows)
- Relation key
 - Attribute(s) that uniquely identify each tuple in a relation .



Database Management Systems (DBMSs)

- A software package designed to **define**, **manipulate**, **retrieve** and **manage** data in a database.

ORACLE®



MySQL®



PostgreSQL

Microsoft®
SQL Server®

And Many More...

What is PostgreSQL?

- PostgreSQL is an advanced, enterprise-class, and open-source relational database system.
- It supports both SQL (**relational**) and JSON (**non-relational**) querying.
- It is highly stable database with more than **20years** of development by the **open-source** community.

PostgreSQL



Common Use cases of PostgreSQL

- **A robust database in many applications**
 - PostgreSQL is primarily used as a robust back-end database that powers **many dynamic websites** and **web applications**.
- **General purpose transaction database**
 - Large corporations and startups alike use PostgreSQL as primary databases to support their applications and products.
- **Geospatial database**
 - PostgreSQL with the [PostGIS extension](#) supports geospatial databases for geographic information systems (**GIS**).

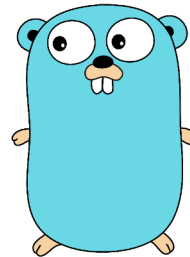


PostgreSQL Language Support



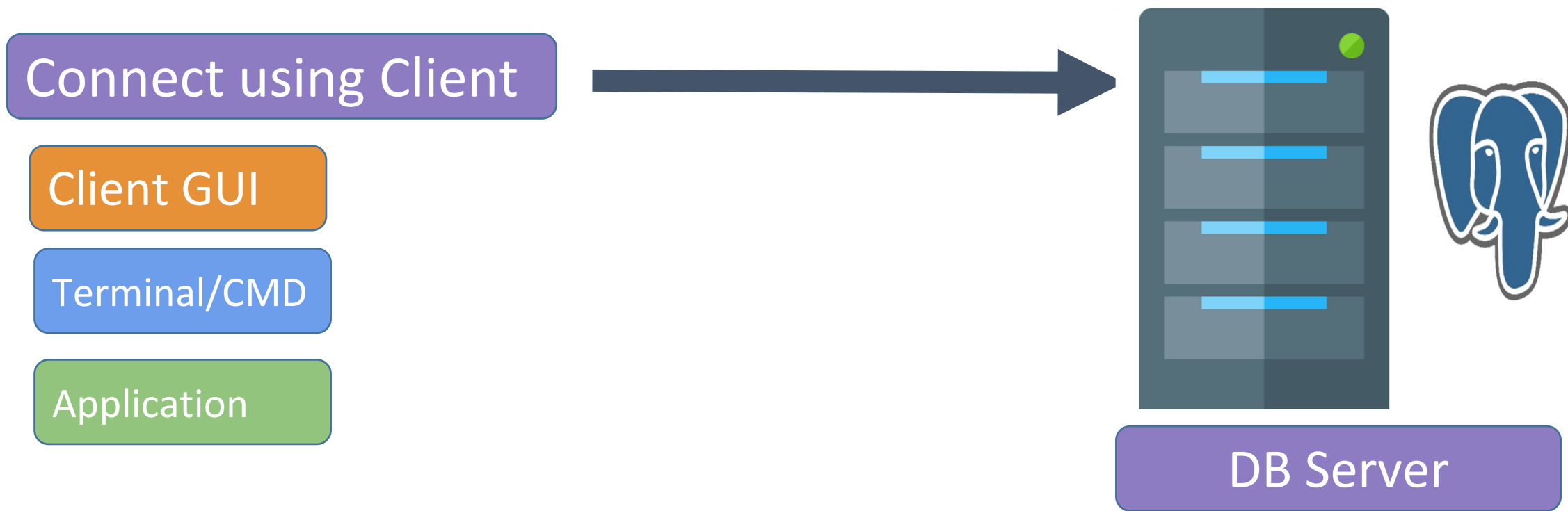
Perl

GO



 **Ruby**

Connecting to DB server



How To Connect to PostgreSQL

- Using the **PSQL Shell**
 - Or the command prompt after adding psql to your path (environment variables)
- Using the **pgAdmin** (The GUI to manage PostgreSQL DBs)
- Using other **third-party client GUI** softwares.

1- PSQL Shell

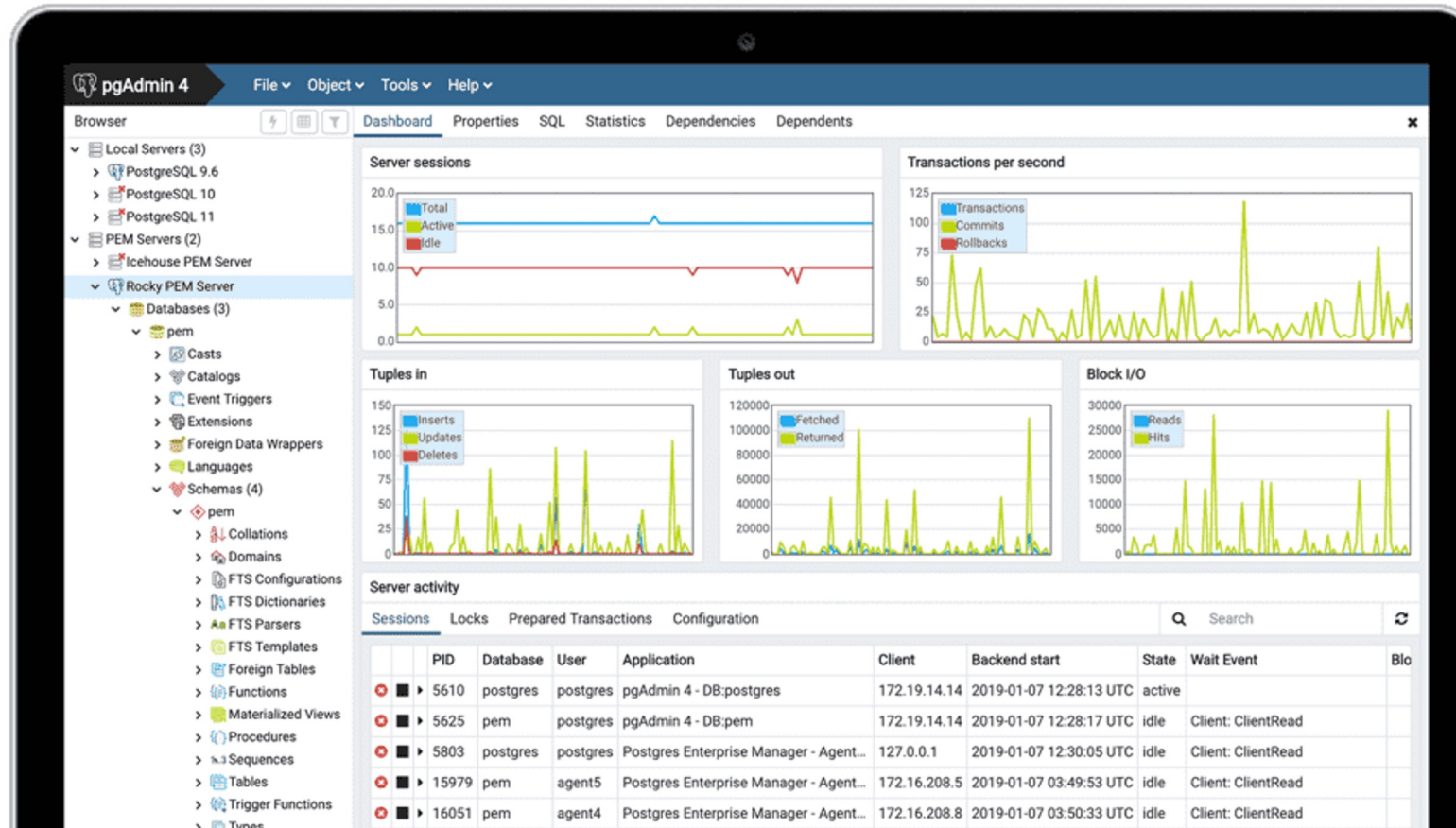
```
postgres=# \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
exercises	postgres	UTF8	Estonian_Estonia.1257	Estonian_Estonia.1257	
moviedb	postgres	UTF8	Estonian_Estonia.1257	Estonian_Estonia.1257	
postgres	postgres	UTF8	Estonian_Estonia.1257	Estonian_Estonia.1257	
socialmedai	postgres	UTF8	Estonian_Estonia.1257	Estonian_Estonia.1257	
socialmedia	postgres	UTF8	Estonian_Estonia.1257	Estonian_Estonia.1257	
template0	postgres	UTF8	Estonian_Estonia.1257	Estonian_Estonia.1257	=c/postgres + postgres=CTc/postgres
template1	postgres	UTF8	Estonian_Estonia.1257	Estonian_Estonia.1257	=c/postgres + postgres=CTc/postgres

(7 rows)

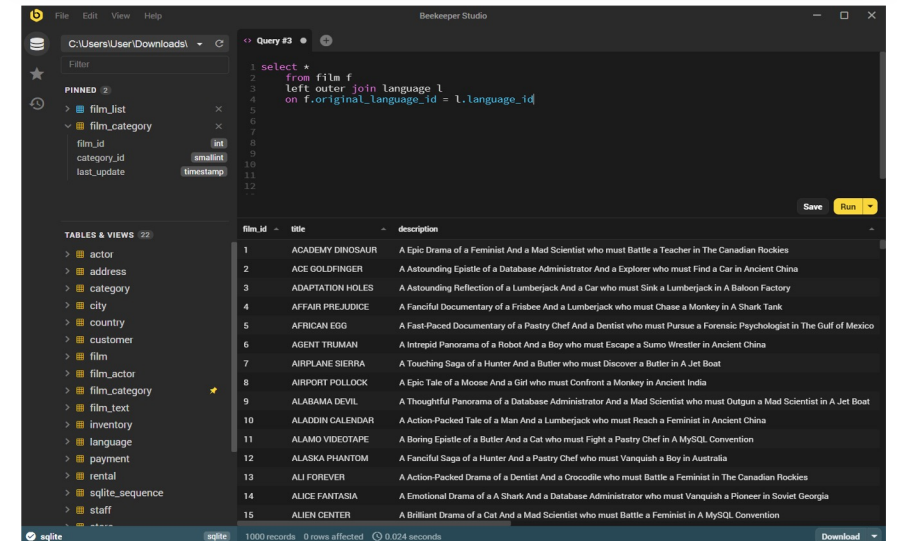
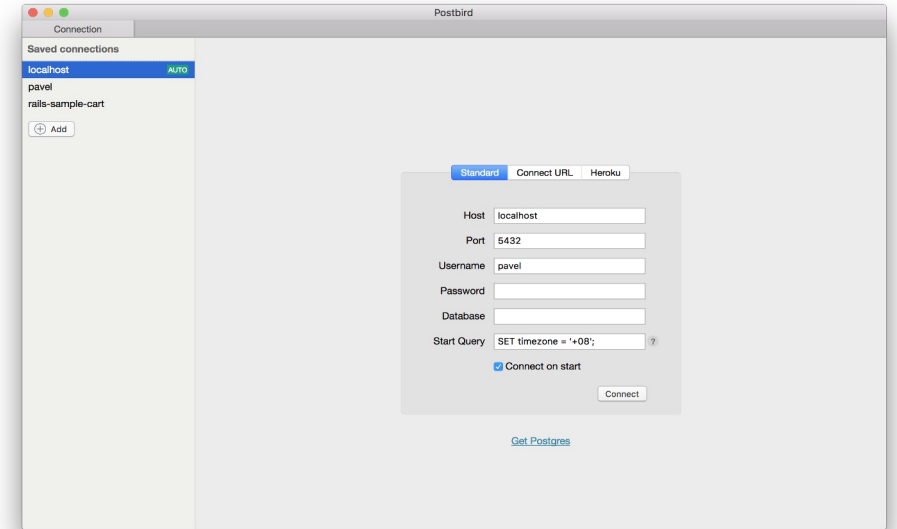
- To get more familiar with PSQL SHELL commands, You can follow this link: (<https://www.postgresql.org/docs/current/app-psql.html>)

2. pgAdmin: the GUI of PostgreSQL



3. Other GUI Client Options

- Paxa/postbird
 - Open source PostgreSQL GUI client for macOS, Linux and Windows
 - <https://github.com/paxa/postbird>
- Beekeeper Studio
 - Open Source SQL Editor and Database Manager
 - <https://www.beekeeperstudio.io/>

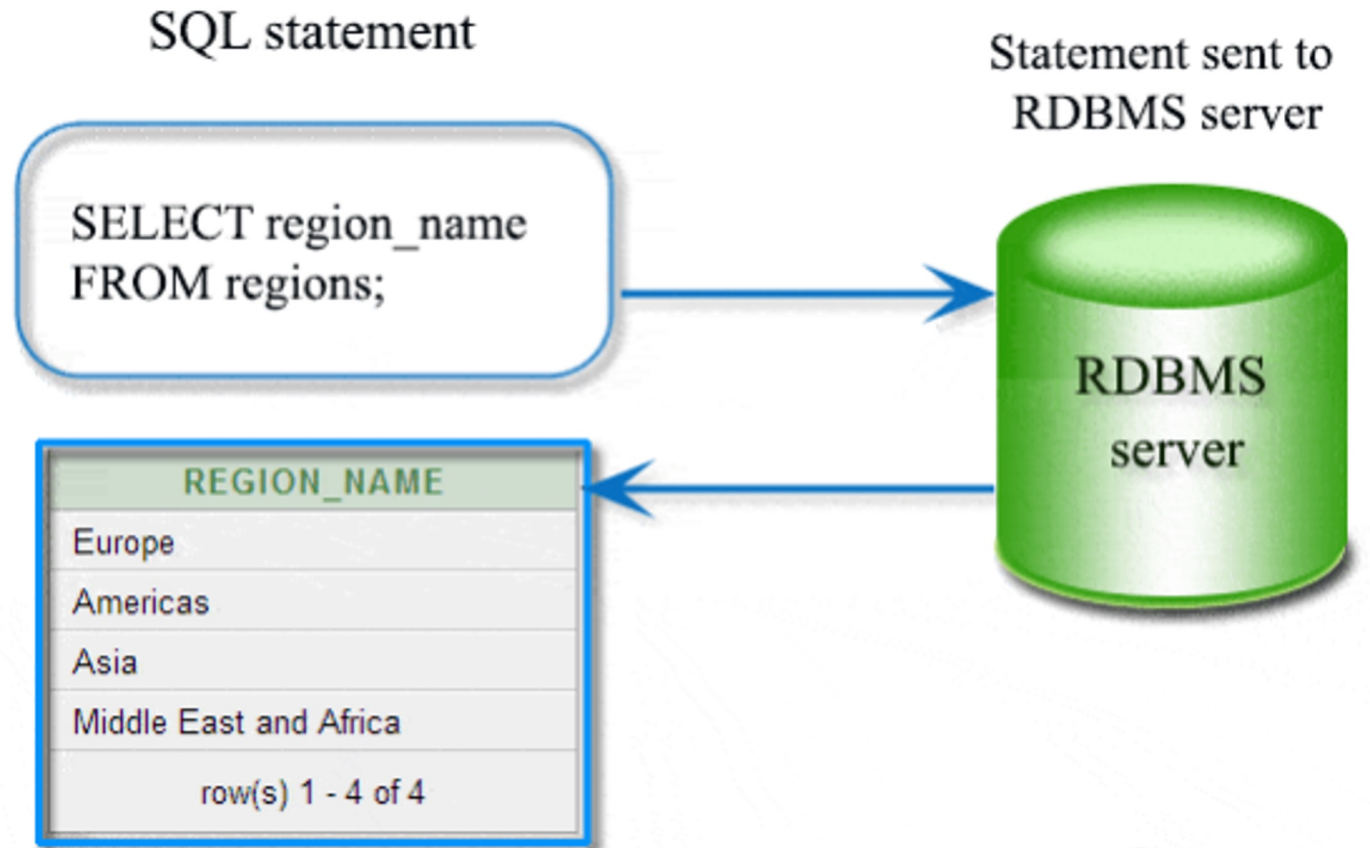


What is SQL?

- SQL stands for Structured Query Language.
- It is the standard language for relational database management systems.
- SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.
- Most of the common relational database management systems that use SQL .



How SQL Works ?



Creating a Database

```
CREATE DATABASE Dbname;
```

Example:

```
CREATE DATABASE tartupurchases;
```

- Let's Try it on Jupyter Notebook -----
>



Create our first Table (Customer)

CREATE TABLE table_name (column_Name + DataType + constraints if any)

Example:

```
CREATE TABLE Customer (  
id INT PRIMARY KEY NOT NULL,  
name TEXT NOT NULL,  
country Text NOT NULL,  
email Text )
```

Customer		
-	id	[INT] PRIMARY KEY NOT NULL
-	name	[TEXT] NOT NULL
-	country	[TEXT] NOT NULL
-	email	[TEXT]

Load Some data to the table (Insert)

- **Not our focus though!!**

INSERT INTO customer (id, name, country, email)

VALUES (1, “Mohamed Ragab”, “Egypt”, “ragab@ut.ee”)

- Use the “Mockrow” website for generating mock data (<https://www.mockaroo.com/>)



Field Name	Type	Options
<input type="text" value="id"/>	Row Number	blank: 0 % <input type="text" value="fx"/> ×
<input type="text" value="first_name"/>	First Name	blank: 0 % <input type="text" value="fx"/> ×
<input type="text" value="last_name"/>	Last Name	blank: 0 % <input type="text" value="fx"/> ×
<input type="text" value="email"/>	Email Address	blank: 0 % <input type="text" value="fx"/> ×
<input type="text" value="gender"/>	Gender	blank: 0 % <input type="text" value="fx"/> ×
<input type="text" value="ip_address"/>	IP Address v4	blank: 0 % <input type="text" value="fx"/> ×

Rows: Format: Line Ending: Include: header BOM

Want to save this for later? [Sign up for free.](#)

Query your Table (get All customers)

- To Fetch/Retrieve data from DB Table we use:

```
SELECT * FROM table_name;
```

- '*' means get all columns from that table.

Example:

```
SELECT * FROM customer;
```

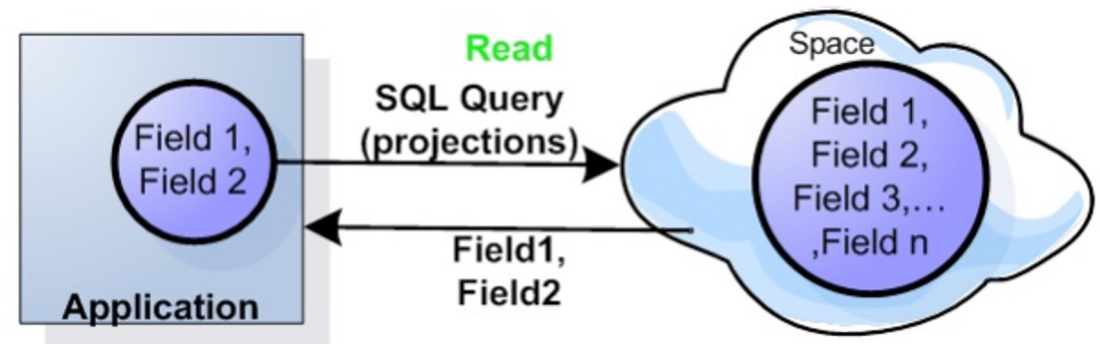


Query your Table (**project** on some fields)

- Projection retrieves only the specified columns.

```
SELECT  
col_name1, col_name2,...col_name_n  
FROM table_name;
```

```
SELECT name, country FROM customer;
```



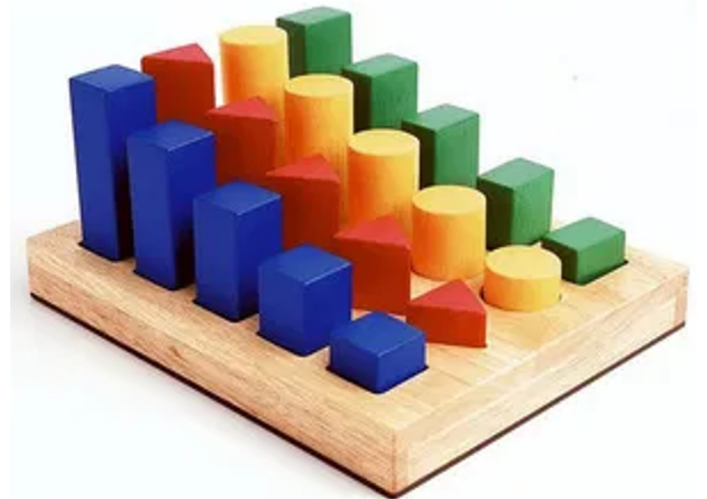
Sorting results (**ORDER BY**)

- Use **ORDER BY** clause to sort results by some columns.

```
SELECT * FROM customer ORDER BY name ASC
```

```
SELECT * FROM customer ORDER BY name DESC
```

- Take Care of **ASC**, and **DESC** for Ascending and Descending sortings.



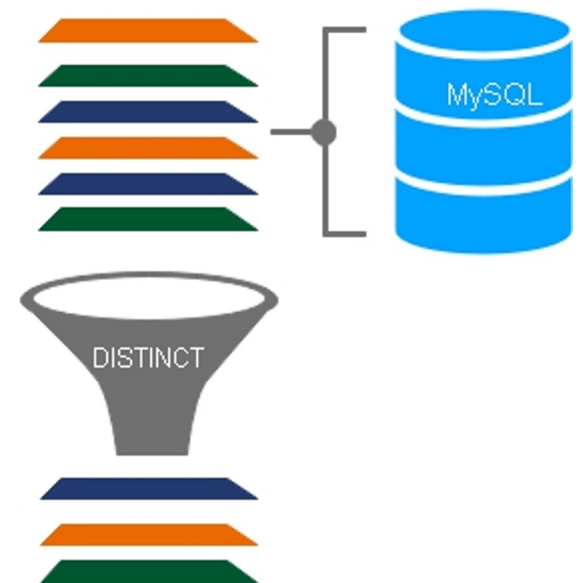
Unique Results (**DISTINCT**)

- The **SELECT DISTINCT** statement is used to return only distinct (different) values.

```
SELECT DISTINCT column1, column2, ...  
FROM table_name;
```

Example:

```
SELECT DISTINCT country from customer;
```



Filtering the Results (WHERE)

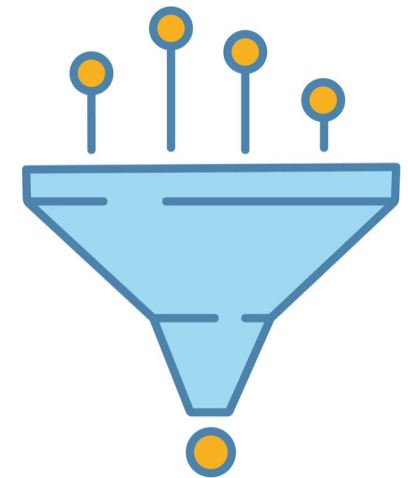
- The **WHERE** clause is used to filter records.
- It's used to extract only those records that fulfill a specified condition.

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

Example:

Get only the customers who have emails (filter out who don't have).

```
SELECT * FROM customer WHERE email != '' ;
```



SQL AND, OR and NOT Operators

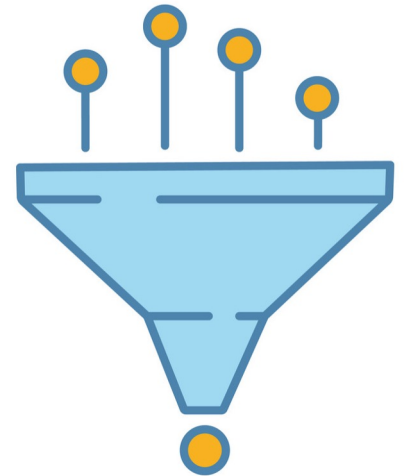
- The WHERE clause can be combined with AND, OR, and NOT operators.

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND condition2 AND condition3 ...;
```

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR condition3 ...;
```

Example:

```
SELECT * FROM customer WHERE country = 'Egypt' AND email !=''
```



Pagination in SQL ("LIMIT" and "OFFSET")

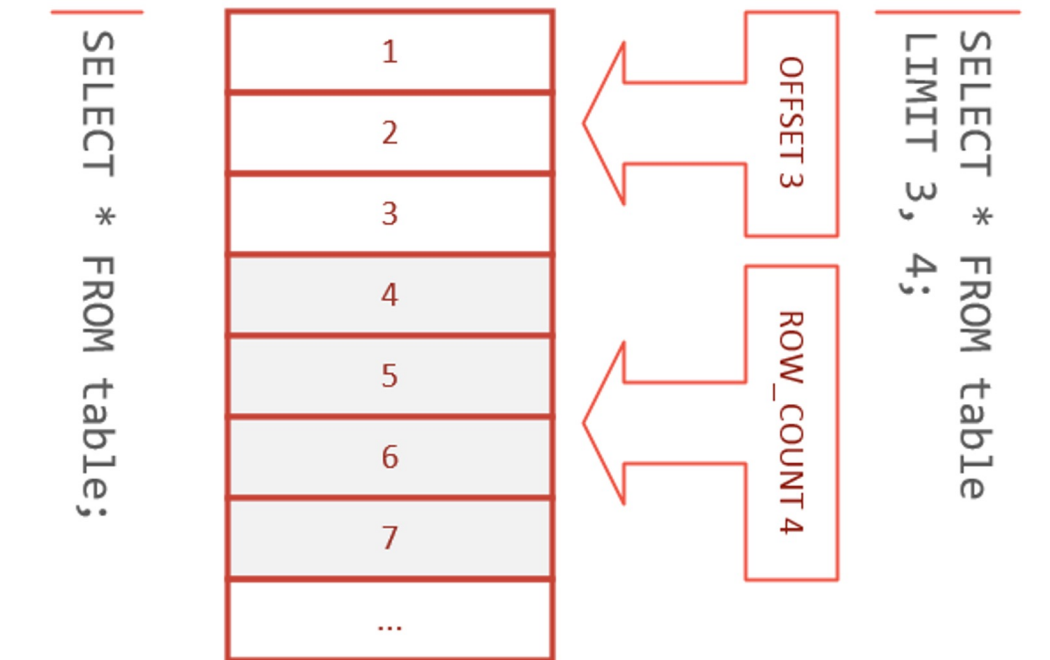
- LIMIT and OFFSET allow you to retrieve just a portion of the rows that are generated by the rest of the query.

SELECT select_list

FROM table_expression

[LIMIT { number | ALL }] [OFFSET number]

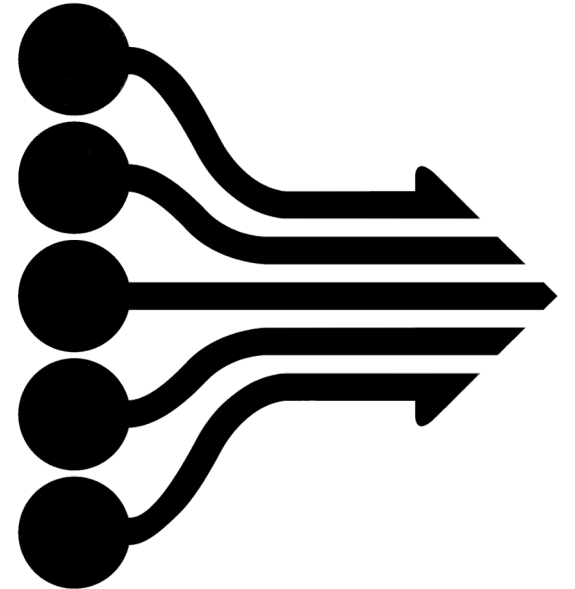
- Example in The Notebook 😊



Aggregations and GROUP BY

- The **GROUP BY** statement groups rows that have the same values into summary rows.
 - Example: "Find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (**COUNT**, **MAX**, **MIN**, **SUM**, **AVG**) to group the result-set by one or more columns.

```
SELECT column_name(s)  
FROM table_name  
GROUP BY column_name(s);
```



GROUP BY ... HAVING

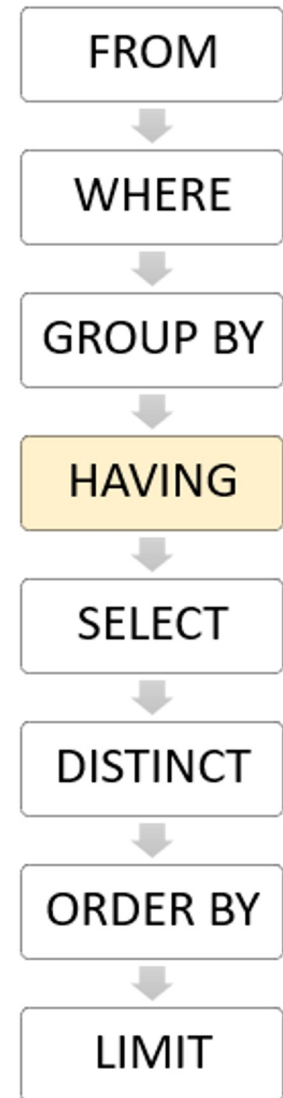
- The HAVING clause specifies a search condition for a group or an aggregate.
- The HAVING clause is often used with the GROUP BY clause to filter groups or aggregates based on a specified condition.

SELECT column1, aggregate_function (column2)

FROM table_name

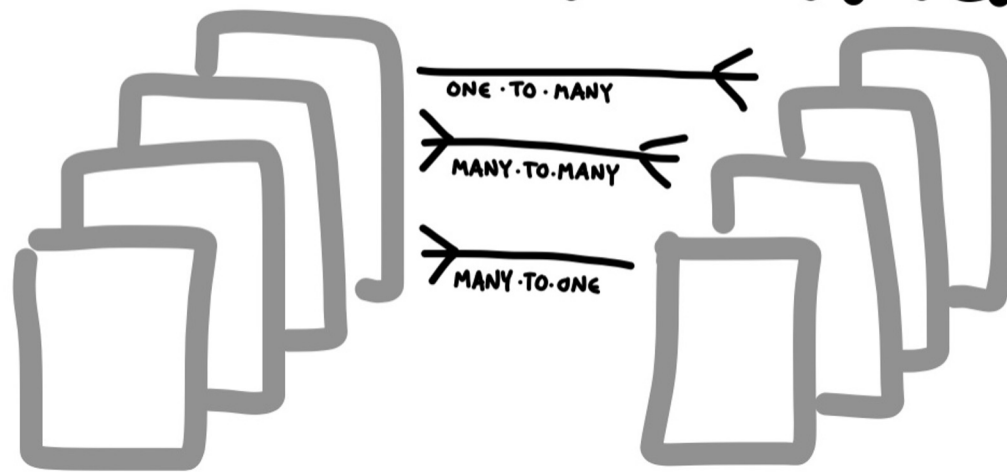
GROUP BY column1

HAVING condition;



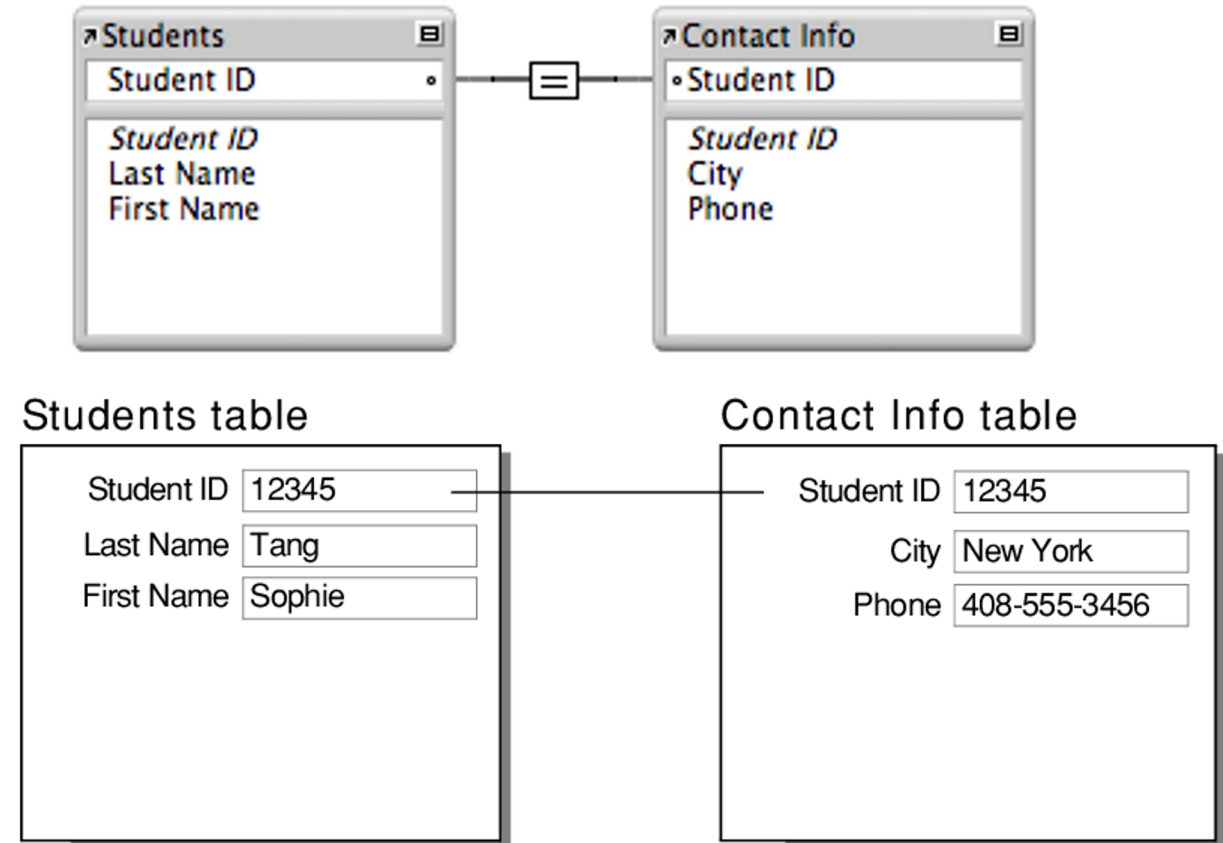
RELATIONSHIPS

RELATIONAL DATABASE



RELATIONSHIPS

- **ONE-to-ONE Relationship:** occurs when one record in a table is associated with one and only one record in another table.
- Example:
 - In a school database:
 - Each student **has only one** student ID.
 - Each student ID **is assigned to only one** person.

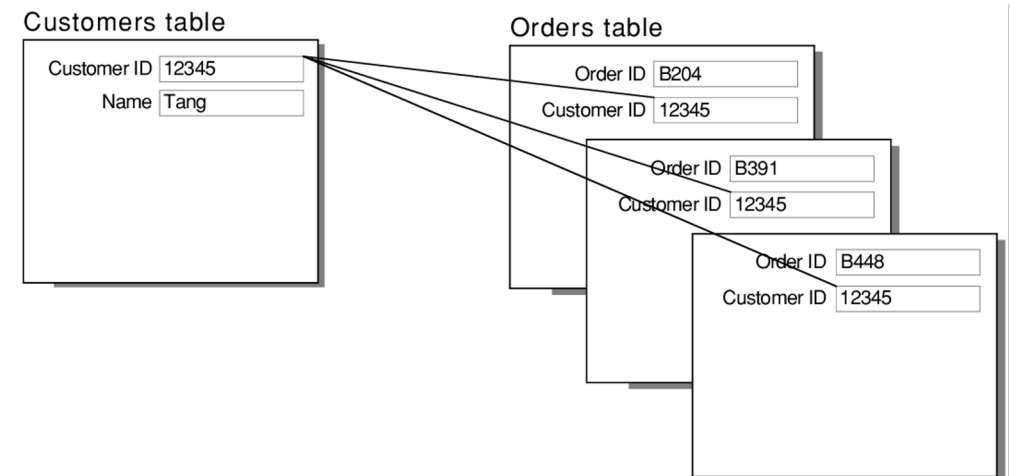
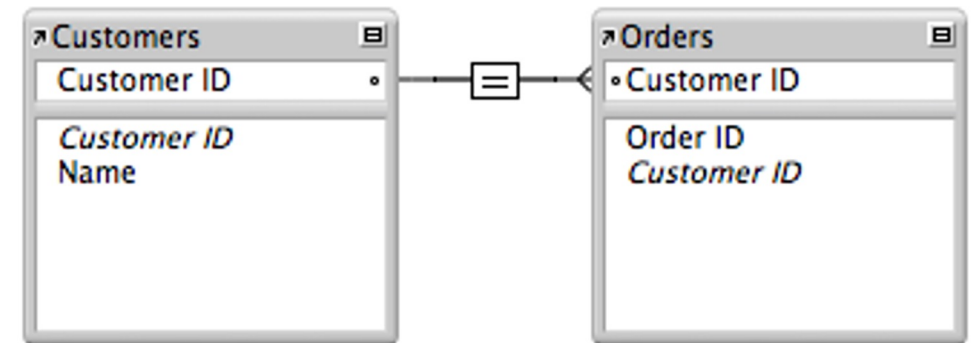


RELATIONSHIPS Cont.

- **ONE-to-Many Relationship:** occurs when one [record](#) in a [table](#) can be associated with one or more records in another table.
- Example:
 - Customer can **make** several/multiple Orders
 - While, Order **can be made** by only one Customer.
- Put the PK in the one-side as a FK in the many-side.

Question?!!!

Why we can't make it the other way around !!?

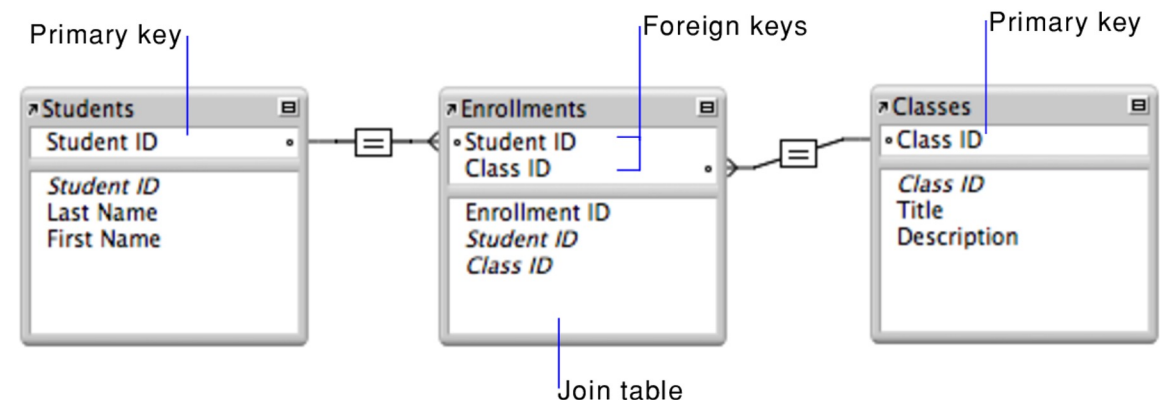


RELATIONSHIPS Cont.

- **Many-to-Many Relationship:** occurs when multiple [records](#) in a [table](#) are associated with multiple records in another table.
- Example:
 - Student can **attend** many classes.
 - Meanwhile, Class **is attended** by many students

Question!!!!

- Assume we have “**grade**” attribute, so to which table we should we put it ?



**THIS DATABASE ISN'T FUN
ANYMORE**

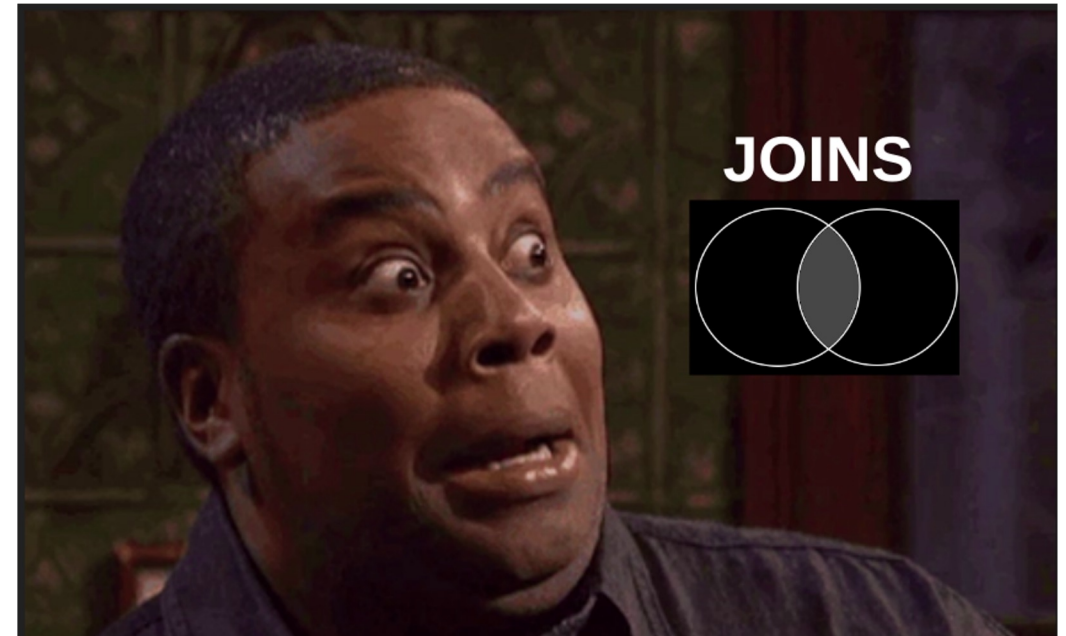


CAN WE PLEASE GO HOME?

Bring Data From Multiple Tables **(SQL JOINS)**

SQL JOINS

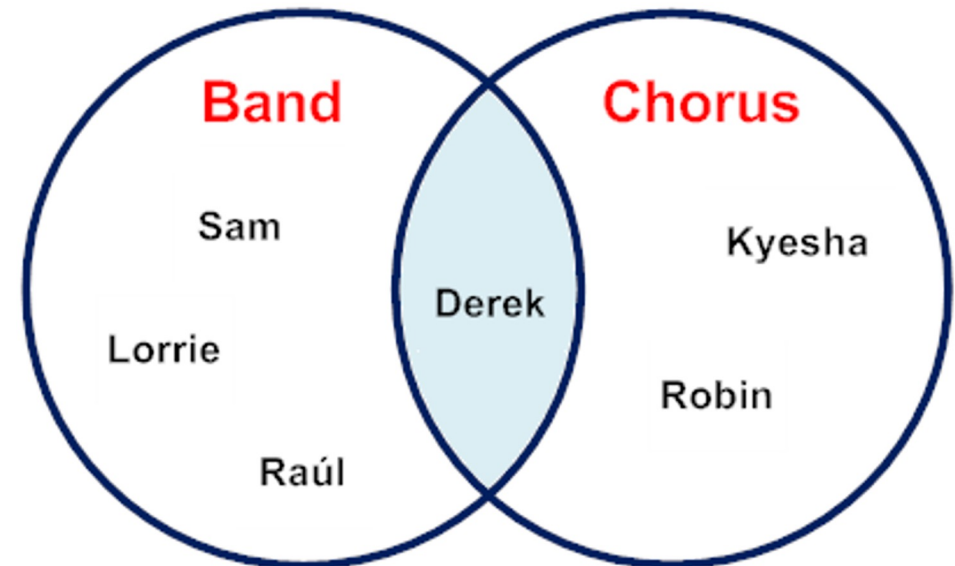
- Joins are one of the key elements of relational DBs.
- They allow us to retrieve data from multiple tables at once.
- Let's go over some different types of Joins.



Joins are like Sets Operations

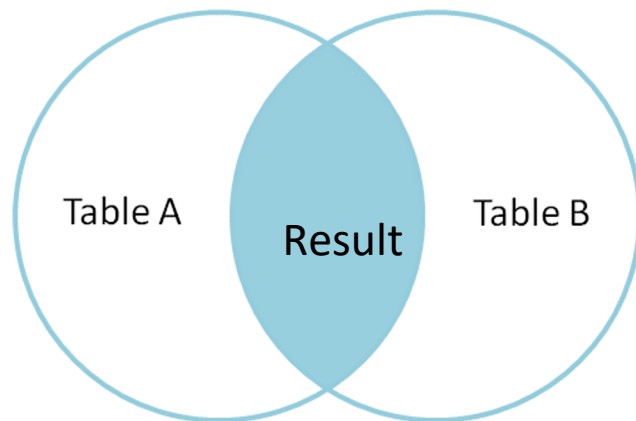
- Sets in python: unordered groups of unique elements.
- JOINS treat rows of data as if they were Sets.
- We can Perform Set operations on the tables.

- Example: Intersect
- Set intersection is the elements common to two sets.
- Here the intersection is {Drek}



INNER JOIN

- The SQL INNER JOIN is similar to the **Set intersection**.
- INNER JOIN selects **rows from table1 and table2** where they **match the selecting column**.



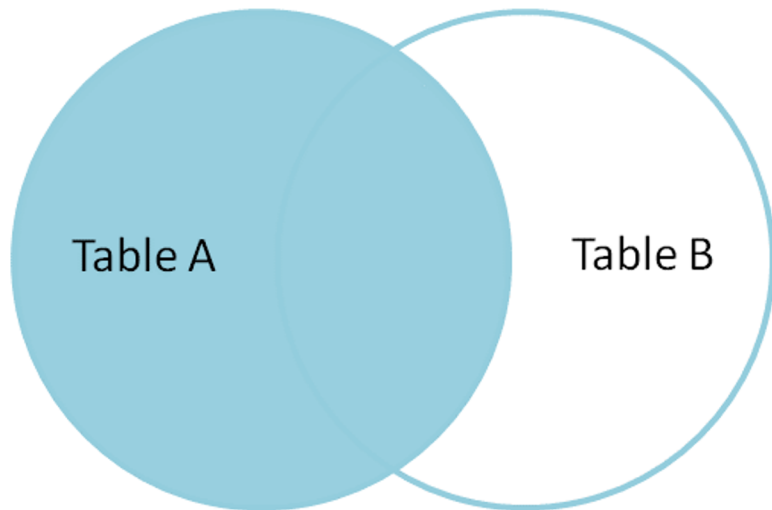
Customers		Orders		
ID	Name	ID	Customer_ID	Product
1	Jose	1	1	Chair
2	Rolf	2	1	Pen
3	Anna	3	1	Monitor
4	Robert	4	3	Headphones

```
SELECT * FROM Customers
INNER JOIN Orders
ON Customers.ID = Orders.Customer_ID
```

Customers.ID	Customers.Name	Orders.ID	Orders.Customer_ID	Orders.Product
1	Jose	1	1	Chair
1	Jose	2	1	Pen
1	Jose	3	1	Monitor
3	Anna	4	3	Headphones

LEFT JOIN

- This selects **all rows from the table1** (on the left), the rows from the table2 (on the right) if they match.
- If they don't match, the data for the right table is blank (NULLS).



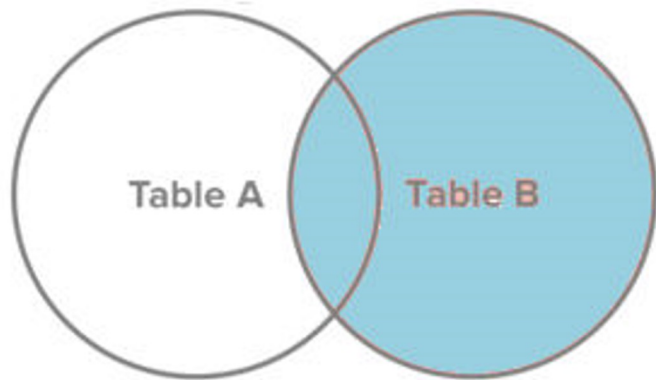
Customers		Orders		
ID	Name	ID	Customer_ID	Product
1	Jose	1	1	Chair
2	Rolf	2	1	Pen
3	Anna	3	1	Monitor
4	Robert	4	3	Headphones

```
SELECT * FROM Customers
LEFT JOIN Orders
ON Customers.ID = Orders.Customer_ID
```

Customers.ID	Customers.Name	Orders.ID	Orders.Customer_ID	Orders.Product
1	Jose	1	1	Chair
1	Jose	2	1	Pen
1	Jose	3	1	Monitor
3	Anna	4	3	Headphones
2	Rolf			
4	Robert			

RIGHT JOIN

- **Opposite** to **LEFT JOIN**
- This selects all the rows from the table on the right, and then rows from the left if they match.
- If they don't match, the data for the table on the left is blank (NULLS).



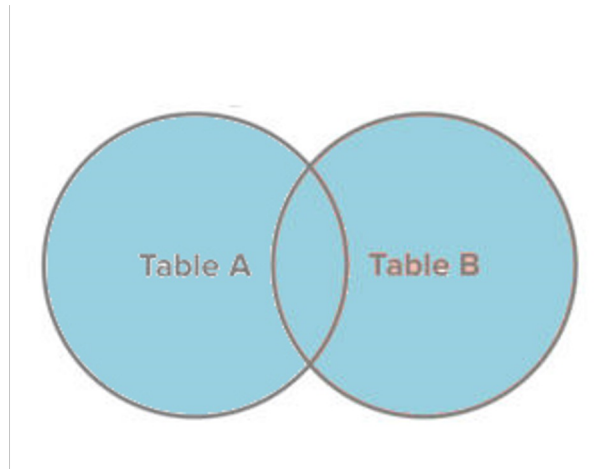
Customers		Orders		
ID	Name	ID	Customer_ID	Product
1	Jose	1	1	Chair
2	Rolf	2		Pen
3	Anna	3	1	Monitor
4	Robert	4	3	Headphones

```
SELECT * FROM Customers
RIGHT JOIN Orders
ON Customers.ID = Orders.Customer_ID
```

Customers.ID	Customers.Name	Orders.ID	Orders.Customer_ID	Orders.Product
1	Jose	1	1	Chair
1	Jose	3	1	Monitor
		2		Pen
3	Anna	4	3	Headphones

FULL JOIN

- This selects all rows from both tables, matching them if there is a match on the selecting column.
- Think of it as a **LEFT** and a **RIGHT** join.



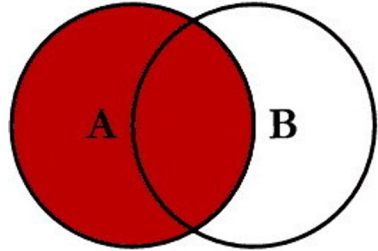
Customers		Orders		
ID	Name	ID	Customer_ID	Product
1	Jose	1	1	Chair
2	Rolf	2		Pen
3	Anna	3	1	Monitor
4	Robert	4	3	Headphones

```
SELECT * FROM Customers
FULL JOIN Orders
ON Customers.ID = Orders.Customer_ID
```

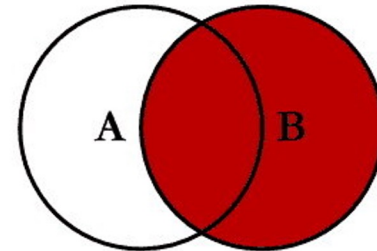
Customers.ID	Customers.Name	Orders.ID	Orders.Customer_ID	Orders.Product
1	Jose	1	1	Chair
1	Jose	3	1	Monitor
2	Rolf	2		Pen
3	Anna	4	3	Headphones
4	Robert			

JOINS with More types..

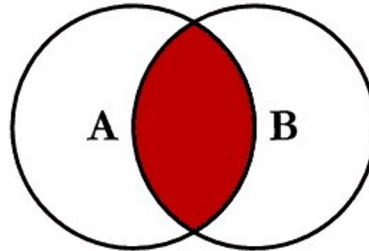
SQL JOINS



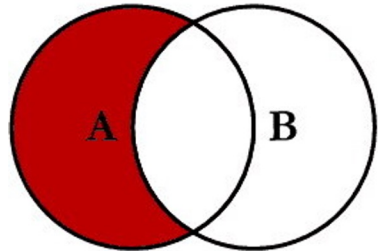
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



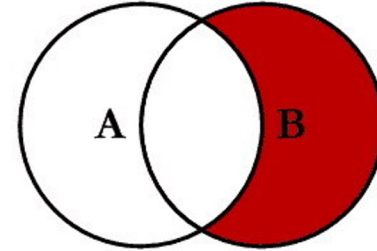
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



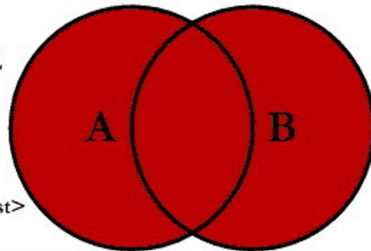
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



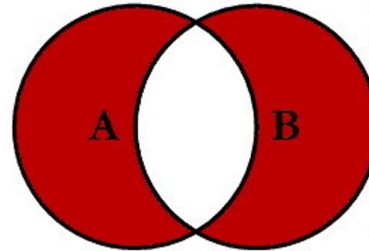
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



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



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

SQL Data Updates (UPDATE)

- The UPDATE statement is used to modify the existing records in a table.

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

Example:

```
UPDATE customer  
SET email = "aisha.kareem@gmail.com"  
WHERE id=3;
```



SQL Data Deletions (DELETE)

- The DELETE statement is used to delete existing records in a table.
- **Note:** Be careful when deleting records in a table!
 - Notice the WHERE clause in the DELETE statement.

```
DELETE FROM table_name WHERE condition;
```

Example:

```
DELETE FROM customer WHERE name="Mohamed Ragab";
```



Now, It's time to say ...

THANK YOU

SEE YOU

NEXT TIME!