



# INTRODUCTION À SQL



# PLAN

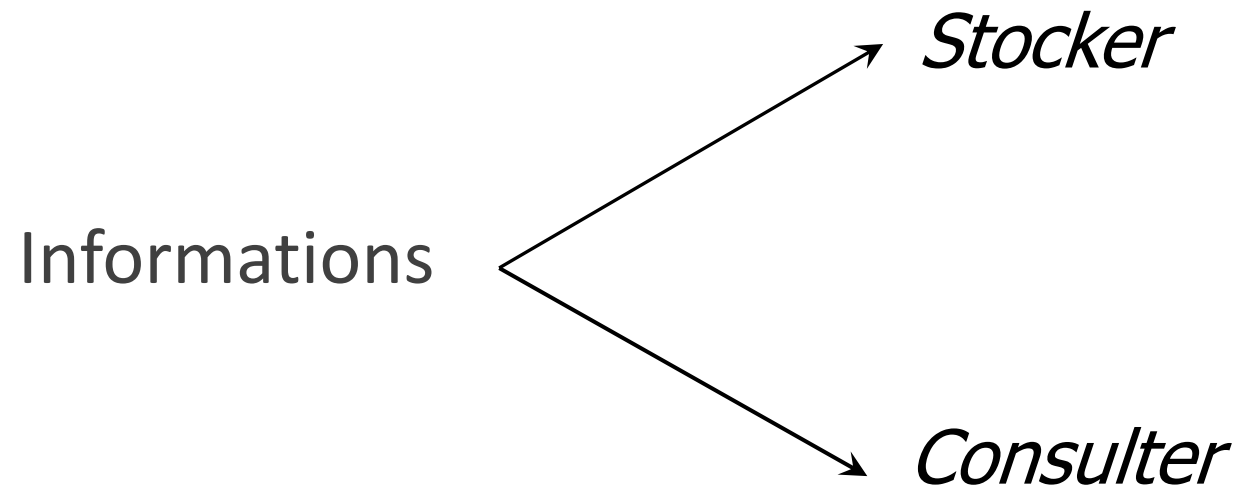
---

- I. Définitions
- II. Modèle relationnel
- III. Langage SQL
- IV. Requête

# Introduction aux Bases de Données et au langage SQL

---

- But des Bases de Données (BDD) :





# Introduction aux Bases de Données et au langage SQL

---

- Base de données (BDD) :
  - Contiennent des ensembles de données ;
  - Organisées suivant un modèle ;
  - Consultable par de nombreux utilisateurs.



# Introduction aux Bases de Données et au langage SQL

---

- Exemple de BDD : Annuaire téléphonique

<b>NOM</b>	<b>PRENOM</b>	<b>TEL.</b>
Benoit	Jean	06 76 45 65 56
Bernard	Francois	06 76 56 68 32
Bourdan	Pierre	06 76 23 54 66

- Données sous forme de tableau et en libre accès.



# Introduction aux Bases de Données et au langage SQL

---

- SGBD : Pour support informatique  
(*Système de Gestion de Bases De Données*)
- Ensemble de logiciels capables de :
  - concevoir, enregistrer, consulter les données sur support informatique ;
  - sécuriser les données :
    - intégrité du contenu
    - droits d'accès





# Introduction aux Bases de Données et au langage SQL

---

- Exemples de SGBD :
  - Microsoft QUERY
  - Microsoft ACCESS
  - MySQL
  - PostgreSQL



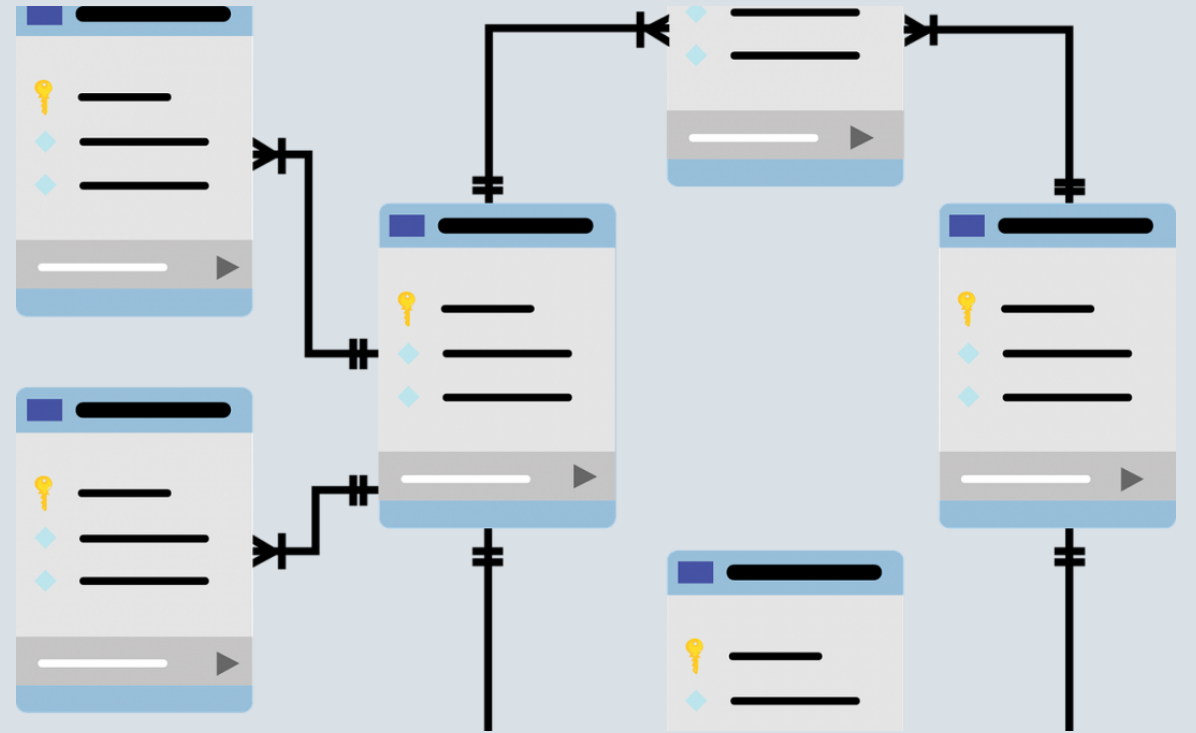
# Introduction aux Bases de Données et au langage SQL

---

- Domaines d'utilisation des SGBD : Traiter un grand nombre d'objets similaires ;
  - Données de clients pour les assurances ;
  - Données patients dans les hôpitaux ;
  - Comptes dans les banques ;
  - Livres dans les bibliothèques ;
  - ...



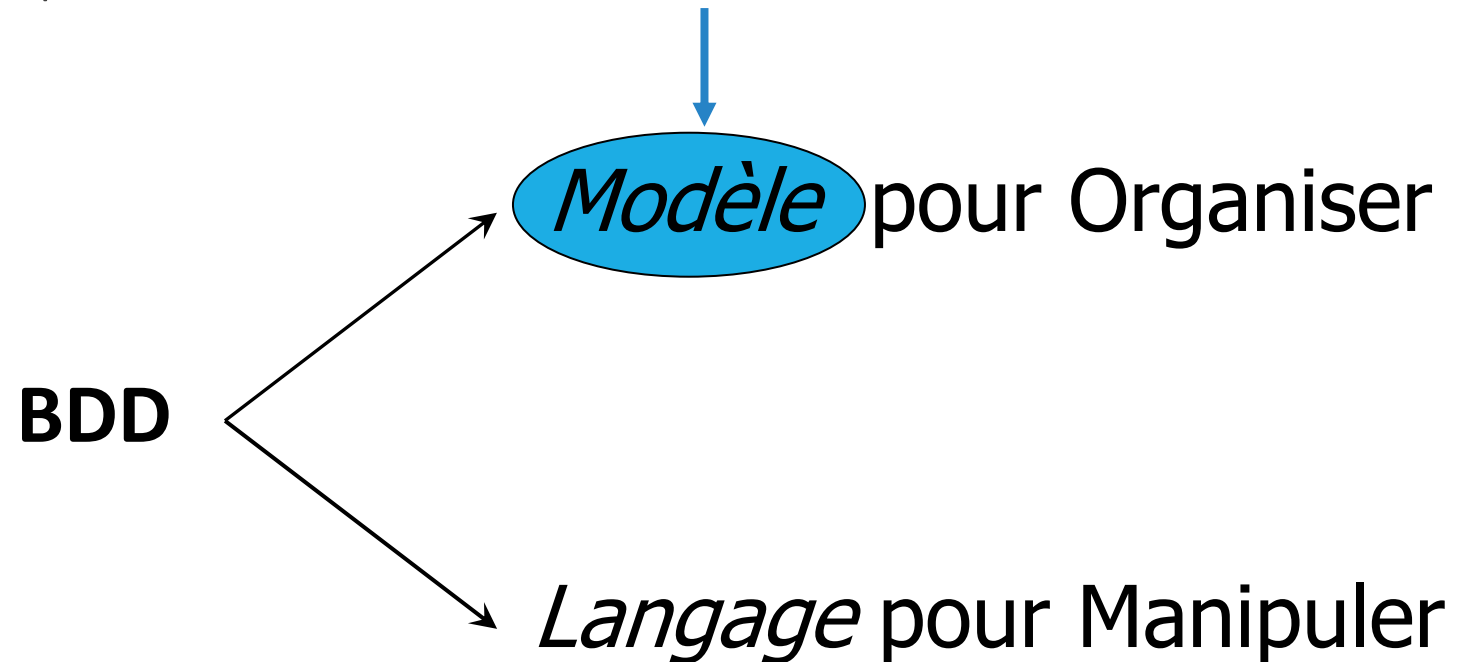
# Modèle relationnel



# Modèle relationnel

---

- But des BDD : gérer des informations.
- Deux questions : comment les stocker, et comment les consulter.





# Modèle relationnel

---

- BDD relationnelle = collection de relations
- relation = table (tableau)
- comment se définit une relation ?
  - Une relation est une table identifiée par un nom et qui se découpe en lignes et en colonnes ;
  - Chaque colonne porte un nom attribut ;
  - Chaque ligne (appelée occurrence) correspond a un objet qui est décrit suivant les valeurs que prennent ses attributs ;

# Modèle relationnel

*Nom* : ANNUAIRE

ANNUAIRE

*Attributs* :  
(NOM, PRENOM, TEL)

NOM	PRENOM	TEL
Benoit	Jean	06 76 45 65 56
Benoit	Francois	06 76 56 68 32
Bourdan	Jean	06 76 23 54 66
Bouvier	Jacqueline	06 76 18 34 35

*Occurrences* :

- 4 abonnés
- décrits suivant valeurs d'attributs

NOM	PRENOM	TEL	AGE
Benoit	Jean	06 76 45 65 56	
Benoit	Francois	06 76 56 68 32	32
Bourdan	Jean	06 76 23 54 66	
Bouvier	Jacqueline	06 76 18 34 35	



# Modèle relationnel

---

- Clé/identifiant d'une relation:
  - attribut(s)
  - Si la valeur de la clé est fixée, une seule occurrence possède cette valeur.
- Question : quelle est la clé d'ANNUAIRE ?
  - Nom
  - Prenom
  - Tel

NOM	PRENOM	TEL
Benoit	Jean	04 76 45 65 56
Benoit	Francois	04 76 56 68 32
Bourdan	Jean	04 76 23 54 66
Bouvier	Jacqueline	04 76 18 34 35



# Modèle relationnel

---

- Clé/identifiant d'une relation:
  - attribut(s)
  - Si la valeur de la clé est fixée, une seule occurrence possède cette valeur.
- Question : quelle est la clé d'ANNUAIRE ?
  - Nom
  - Prenom
  - Tel

NOM	PRENOM	TEL
Benoit	Jean	04 76 45 65 56
Benoit	Francois	04 76 56 68 32
Bourdan	Jean	04 76 23 54 66
Bouvier	Jacqueline	04 76 18 34 35

1 n° téléphone → 1 seul abonné



# Modèle relationnel

## Schéma de relation

---

- Schéma d'une relation :

Nom relation ( clé, attribut1, attribut2, ... )

- Schéma de l'ANNUAIRE :

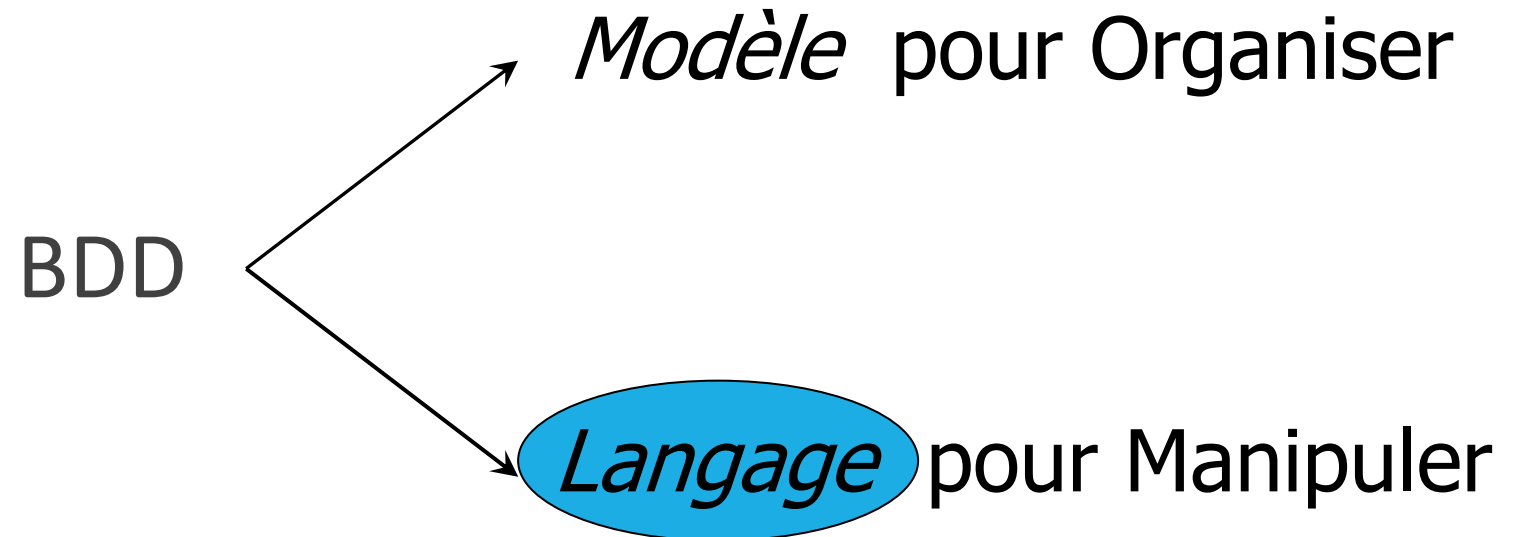
ANNUAIRE ( TEL, PRENOM, NOM )

# Langage SQL



# SQL - langage relationnel

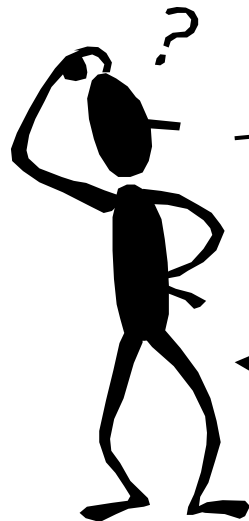
---



# SQL - langage relationnel

---

- SQL : Structured Query Language
- Langage d'interrogation (Anglais)
- Inventé par IBM en 1973



Requête SQL

Réponse : relation



SGBD



# SQL - langage relationnel

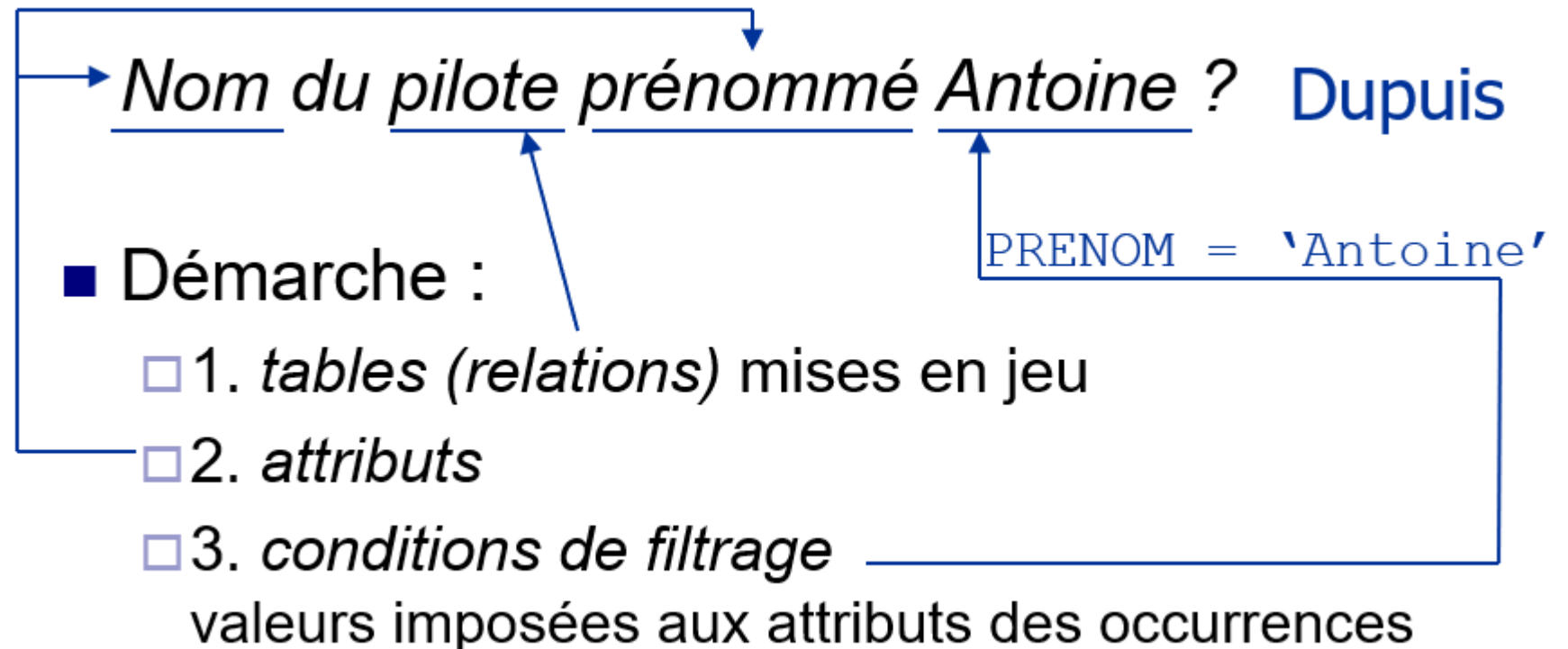
---

- Structured Query Language (SQL)
- langage d'interrogation (Anglais)
- inventé par IBM (1973)



# SQL - langage relationnel

- Requête en Français :





# SQL - langage relationnel

---

- Requête en SQL :

3 parties (SFW)

Select <liste attributs à afficher en résultat>

From <listes tables>

Where <condition de filtrage des occurrences>;



# SQL - langage relationnel

---

- Requête en SQL exemple

- En Français :

Nom du pilote prénommé Antoine ?

- Traduction SQL :

```
SELECT nom FROM pilote WHERE prenom = 'Antoine';
```

```
SELECT age from collection SORT ASC LIMIT 1;
```



# Questions

---

- Qu'est ce qu'un SGBD ?
- De quoi se compose un relation/table ?
- Qu'est ce que SQL ?
- Avec quoi se termine une requête SQL ?



# Questions

---

- Qu'est ce qu'un SGBD ? logiciel servant à stocker, à manipuler ou gérer des données dans une base de données.
- De quoi se compose un relation/table ?
- Qu'est ce que SQL ?
- Avec quoi se termine une requête SQL ?





# Questions

---

- Qu'est ce qu'un SGBD ? logiciel servant à stocker, à manipuler ou gérer des données dans une base de données.
- De quoi se compose un relation/table ? Nom de la table, attributs, Occurrences/lignes.
- Qu'est ce que SQL ?
- Avec quoi se termine une requête SQL ?





# Questions

---

- Qu'est ce qu'un SGBD ? logiciel servant à stocker, à manipuler ou gérer des données dans une base de données.
- De quoi se compose un relation/table ? Nom de la table, attributs, Occurrences/lignes.
- Qu'est ce que SQL ? Langage de manipulation des données.
- Avec quoi se termine une requête SQL ?



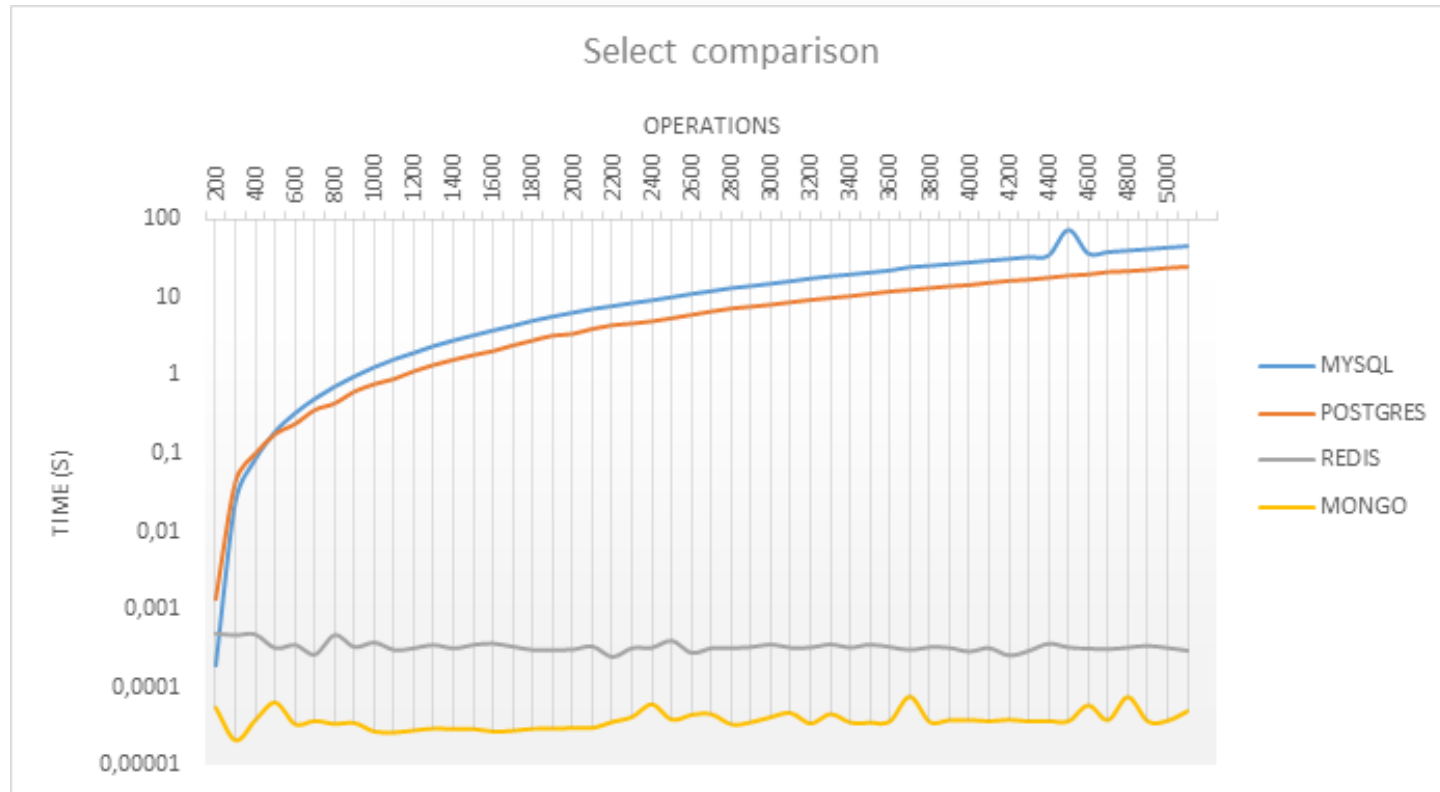
# Questions

---

- Qu'est ce qu'un SGBD ? logiciel servant à stocker, à manipuler ou gérer des données dans une base de données.
- De quoi se compose un relation/table ? Nom de la table, attributs, Occurrences/lignes.
- Qu'est ce que SQL ? Langage de manipulation des données.
- Avec quoi se termine une requête SQL ? ;

	PostgreSQL 10	MySQL 8
Common Table Expression (CTE)	Yes	Yes (Newly Added)
Declarative Partitioning	Yes (Newly Added)	Yes
Full-text Search	Yes	Yes
Geographic Information System (GIS) / Spatial Reference System (SRS)	Yes	Yes (Upgraded)
JSON	Yes	Yes (Upgraded)
Logical Replication	Yes (Newly Added)	Yes
Semi-Synchronous Replication	Yes (Newly Added)	Yes
Window Functions	Yes	Yes (Newly Added)

# Comparaison de Vitesse de select entre SGBD





PostgreSQL

# PostgreSQL

LIEN : <https://www.postgresql.org/download/>

## Downloads

### PostgreSQL Downloads

---

PostgreSQL is available for download as ready-to-use packages or installers for various platforms, as well as a source code archive if you want to build it yourself.

### Packages and Installers

Select your operating system family:





# PostgreSQL

1

Windows installers 

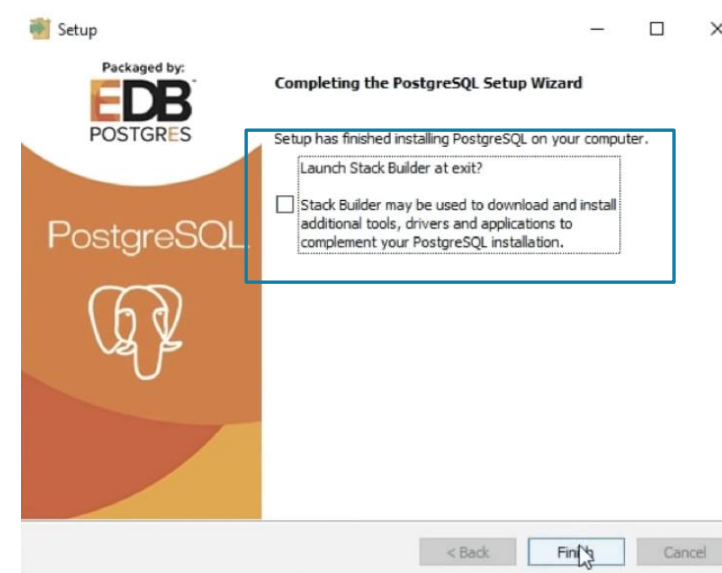
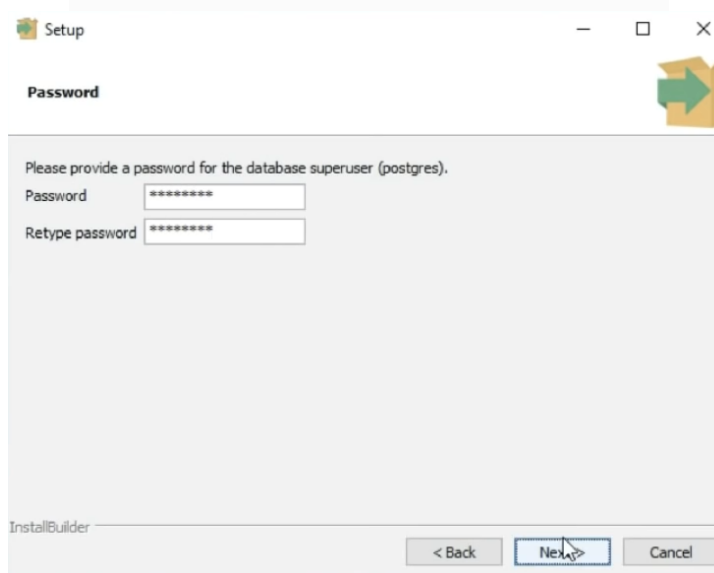
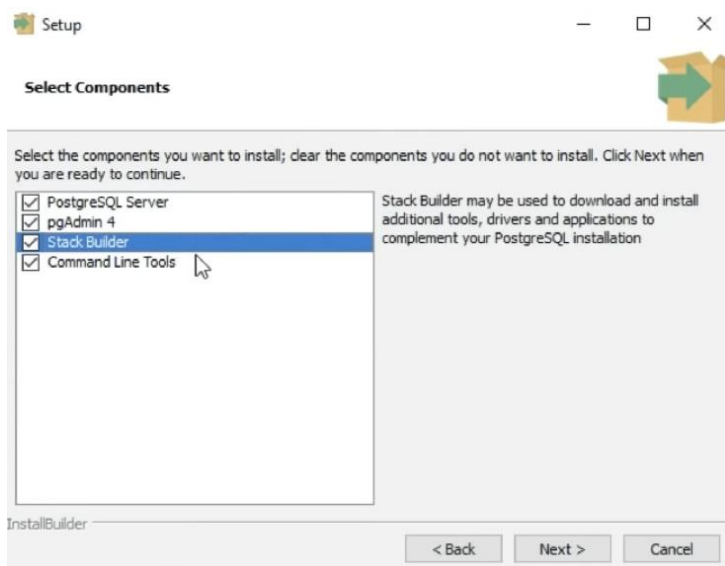
Interactive installer by EDB

**Download the installer** certified by EDB for all sup

## PostgreSQL Database Download

Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64
13.2	N/A	N/A	<b>2</b> <a href="#">Download</a>	<a href="#">Download</a>
12.6	N/A	N/A	<a href="#">Download</a>	<a href="#">Download</a>

# PostgreSQL



Faire l'installation avec les paramètres par défaut



# PostgreSQL

## SQL SHELL

- Lancez le shell et exécutez les commandes suivantes :
  - help
  - \?
  - \l

# PostgreSQL

## SQL SHELL CRÉER UN BASE DE DONÉES

- Pour créer une base de données :
  - `CREATE DATABASE test;`
  - `\l`
- Pour se connecter à la base de données :
  - `\c test`
  - `\c postgres`

# PostgreSQL

## SYNTAXE CRÉATION DE TABLES

```
CREATE TABLE nom_de_la_table
(
    colonne1 type_donnees,
    colonne2 type_donnees,
    colonne3 type_donnees,
    colonne4 type_donnees
)
```



```
CREATE TABLE utilisateur
(
    id INT PRIMARY KEY NOT NULL,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    email VARCHAR(255),
    date_naissance DATE,
    pays VARCHAR(255),
    ville VARCHAR(255),
    code_postal VARCHAR(5),
    nombre_achat INT
)
```

# PostgreSQL

## CRÉATION DE TABLES

- Créer table dans la base de données test :
  - \c test
  - CREATE TABLE person (id INT, first\_name VARCHAR(50), last\_name VARCHAR(50), gender VARCHAR(7), date\_of\_birth DATE);
  - \d
  - \d person



# PostgreSQL

## CRÉATION DE TABLE AVEC CONTRAINTES

- Créer table avec contraintes dans la base de données test :
  - \c test
  - CREATE TABLE client (id BIGSERIAL NOT NULL PRIMARY KEY, first\_name VARCHAR(50) NOT NULL, last\_name VARCHAR(50) NOT NULL, gender VARCHAR(7) NOT NULL, date\_of\_birth DATE NOT NULL, email VARCHAR(150) );
  - \d
  - \d person

# PostgreSQL

## INSERTION DE DONNÉES

- Insertion de donnée à la table person2 dans la base de données test :
  - \c test
  - INSERT INTO person2(first\_name, last\_name, gender, date\_of\_birth, email) VALUES ('Jack', 'Darel', 'Male', date '1991-01-01', 'jack@gmail.com');
  - INSERT INTO person2(first\_name, last\_name, gender, date\_of\_birth, email) VALUES ('Lise', 'Brook', 'Female', date '1995-04-03', 'lise@gmail.com');

# PostgreSQL


## SÉLECTION DE DONNÉES

- Sélection de données de la table person2 dans la base de données test :
  - \c test
  - select \* from person2;
  - select first\_name from person2;
  - select first\_name, last\_name from person2;
  - select \* from person2 where date\_of\_birth > '1992-01-01';
  - select \* from person2 where first\_name = 'Lise';

# PostgreSQL

## INSERTION DE DONNÉES

- Maintenant nous allons insérer plusieurs lignes à notre table,
- Pour générer les données nous allons utiliser le site suivant : <https://www.mockaroo.com/>
- Sur le site :
  - Supprimer l'id, ip\_adress;
  - Ajouter 'date\_of\_birth' de type datetime, format yyyy-mm-dd;
  - Mettre 30% de nulles dans email;
  - Ajouter un champs 'country\_of\_birth' de type country;
  - Changer le format des données à générer à 'SQL' et rajouter le nom de la table 'person2';
  - Cocher la case 'include CREATE TABLE' et appuyer sur PREVIEW, ensuite DOWNLOAD;




email	Email Address	blank:	30%	Σ	×
-------	---------------	--------	-----	---	---

# PostgreSQL

## INSERTION DE DONNÉES

- Aller sur : <https://code.visualstudio.com/>
- Télécharger l'éditeur et l'installer avec les paramètres par défaut;
- Ouvrir le fichier SQL dans Vscodé et effectuer les changements :
- Il est possible de copier tout le code et le coller directement sur le Shell, mais ici on va faire autrement;
- Sur le Shell, taper : \? Et chercher Input/Output

Rechercher la fonction : execute commande from file : \i File



```
create table person2 (  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    email VARCHAR(150),  
    gender VARCHAR(50) NOT NULL,  
    date_of_birth DATE NOT NULL,  
    country_of_birth VARCHAR(50) NOT NULL  
);
```

# PostgreSQL

## INSERTION DE DONNÉES

- Récupérer le chemin absolu du fichier sql : 'C:\Users\Downloads\person2.sql' et changer le sens des slash;
- \c test;
- \i C:/Users/Downloads/person2.sql
- Comment expliquez-vous l'erreur ?

```
psql:C:/Users/ahamid/Downloads/person2.sql:1008: ERREUR: la colonne « country_of_birth » de la relation « person2 » n'existe pas
LIGNE 1 : ...st_name, last_name, email, gender, date_of_birth, country_of...
                                     ^
test=#
```

- Comment résoudre l'erreur ?





# PostgreSQL

## INSERTION DE DONNÉES

- On a pas la colonne qu'on veut insérer;
- Vérifier cela en utilisant `\d person`;

## SUPPRESSION DE TABLE

- `DROP TABLE person2;`
- `DROP TABLE person;`
- Vérifier avec : `\d`

# PostgreSQL

## INSERTION DE DONNÉES

- Insérer maintenant les données à partir du fichier .sql:

\i C:/Users/Downloads/person2.sql

- **Comment voir le contenu de la table person2 ?**
- On remarque l'absence de la colonne 'id', il faut le rajouter sur le fichier .sql :
- Revenir sur le Shell, Drop la table et recharger le fichier .sql;
- Vérifier la présence de la colonne 'id';

```
create table person2 (  
    id BIGSERIAL NOT NULL PRIMARY KEY,  
    first_name VARCHAR(50) NOT NULL
```

# PostgreSQL

## SORTING/ORDERING

- 1 2 3 4 5 6 → ASC      ///      6 5 4 3 2 1 → DESC
- Sur le shell : `select * from person2 ORDER BY country_of_birth;`
- `select * from person2 ORDER BY country_of_birth ASC;` → ASC est la valeur par défaut
- `select * from person2 ORDER BY country_of_birth DESC;`
- `select * from person2 ORDER BY id DESC;`
- `select * from person2 ORDER BY first_name;`
- `select * from person2 ORDER BY first_name DESC;`
- **Faire la même chose avec l'email, que constatez-vous ?**
- `select * from person2 ORDER BY first_name, email;`

# PostgreSQL

## REMOVING DUPLICATES

- `select country_of_birth from person2 order by country_of_birth;`
- On a des valeurs qui se répètent là.
- `select distinct country_of_birth from person2 order by country_of_birth;`
- `select distinct country_of_birth from person2 order by country_of_birth DESC;`

# PostgreSQL

## CONDITION WHERE

- `select * from person2 where gender = 'Female';`
- `select * from person2 where gender = 'Male'`
- `select * from person2 where gender = 'Male' and country_of_birth = 'Russia';`
- `select * from person2 where gender = 'Male' and (country_of_birth = 'Russia' OR country_of_birth = 'China') ;`
- `select * from person2 where gender = 'Male' and (country_of_birth = 'Russia' OR country_of_birth = 'China') and date_of_birth > '2021-01-01';`
- `select * from person2 where gender = 'Female' and (country_of_birth = 'Russia' OR country_of_birth = 'China') and date_of_birth > '2021-01-01';`

# PostgreSQL

## COMPARISON OPERATORS

- =
- <
- <=
- >
- >=
- <>
- `select 1 <> 2;`
- `select 1 <> 1;`
- `select 'PROFESSEUR' <> 'professeur';`



# PostgreSQL

## LIMIT / OFFSET / FETCH

- `select * from person2 limit 10;`
- `select * from person2 offset 5 limit 10;` ➔ l'id commence à 6.
- `select * from person2 offset 5;`
- `select * from person2 offset 5 fetch first 2 row only;`

# PostgreSQL

## IN KEYWORD

- `select * from person2 where country_of_birth = 'China' or country_of_birth = 'Russia' or country_of_birth = 'France';`
- `select * from person2 where country_of_birth in ('China', 'Russia', 'France');`
- `select * from person2 where country_of_birth in ('China', 'Russia', 'France') order by country_of_birth limit 5;`

# PostgreSQL

## BETWEEN KEYWORD

- `select * from person2 where date_of_birth between date '2000-01-01' and data '2020-12-31';`

## LIKE KEYWORD

- `select * from person2 where email like '%.com' limit 10;`
- `select * from person2 where email like '%@google.com' limit 10;`
- `select * from person2 where email like '%@google.%' limit 10;`
- `select * from person2 where email like 'd%' limit 10;`
- `select * from person2 where country_of_birth like 'p%' limit 10;` ➔ SQL est sensible à la casse.
- `Select * from person2 where country_of_birth ilike 'p%' limit 10;`

# PostgreSQL

## GROUP BY

- `select country_of_birth , count(*) from person2 group by country_of_birth;`
- `select country_of_birth , count(*) from person2 group by country_of_birth order by country_of_birth;`
- `select country_of_birth , count(*) from person2 group by country_of_birth order by count desc;`

## HAVING

- `select country_of_birth , count(*) from person2 group by country_of_birth having count(*)> 5 order by count asc;`
- `select country_of_birth , count(*) from person2 group by country_of_birth having count(*)>= 5 order by count asc;`

# PostgreSQL

## CAR TABLE

- Revenir sur : <https://www.mockaroo.com/>
- Simuler les données suivantes : **id** (type: Row Number), **make** (type: car make), **model** (type: car model), **price** (type: number)

- Pour le champs **money**, choisir des valeurs entre 10000 et 100000 :

price	Number	↕	min:	10000	max:	10000
-------	--------	---	------	-------	------	-------

- Sur votre ide effectuer les changements suivants :

```
create table car (  
  id BIGSERIAL NOT NULL PRIMARY KEY,  
  make VARCHAR(100) NOT NULL,  
  model VARCHAR(100) NOT NULL,  
  price NUMERIC(19,2) NOT NULL
```

Price en double précision

- Après avoir enregistré, lancer la requête sur le Shell : \i ...

# PostgreSQL

## MIN MAX AVG

- `select MAX(price) FROM car;`
- `select MIN(price) FROM car;`
- `select AVG(price) FROM car;`
- `select ROUND(AVG(price)) FROM car;`
- `select make, model, MIN(price) from car GROUP BY make, model;`
- `select make, MAX(price) from car GROUP BY make;`

## SUM

- `select SUM(price) from car;`
- `Select make, SUM(price) from car GROUP BY make;`



# PostgreSQL

## ARITHMETIC OPERATORS

- On pourrait faire des calculs ou des statistiques en utilisant les opérateurs arithmétiques :
  - `select 10+2;` - `select 10^2 ;` - `select 10!;`
- `select id, make, model, price, ROUND(price * .10, 2) from car;`

## ALIAS

- `select id, make, model, price, ROUND(price * .10, 2) as ten_percent from car;`

## COALESCE

- `Select coalesce(1);`
- `Select coalesce(null, 1);` #default value will be 1 when null
- `Select coalesce(null, null, null, null, 1);`



# PostgreSQL

## COALESCE

- `select * from person2;` #remarquer les valeurs manquantes dans email
- `Select COALESCE(email) from person2;`
- `select COALESCE(email, 'NOT PROVIDED') from person2;`

# PostgreSQL

## DATA MANIPULATION (EX. COMPUTE AGE)


- \d person2;
- select first\_name, last\_name, gender, country\_of\_birth, date\_of\_birth from person2;
- select first\_name, last\_name, gender, country\_of\_birth, date\_of\_birth, AGE(NOW(), date\_of\_birth) as age from person2;

## PRIMARY KEYS

```
test=# \d person2
```

Colonne	Type	Table % public.person2	Collationnement	NULL-able	Par défaut
id	bigint			not null	nextval('person2_id_seq'::regclass)
first_name	character varying(50)			not null	
last_name	character varying(50)			not null	
email	character varying(150)				
gender	character varying(50)			not null	
date_of_birth	date			not null	
country_of_birth	character varying(50)			not null	

Index :  
"person2\_pkey" PRIMARY KEY, btree (id)



- insert into person2 (id, first\_name, last\_name, gender, email, date\_of\_birth, country\_of\_birth) values (1, 'first', 'last', 'Male', 'first@last.com', '1990-01-01', 'Italy');
- Puis-je appeler la sécurité sociale et demander de changer mon numéro de sécurité sociale et avoir le votre ?

# PostgreSQL

## PRIMARY KEYS \_ ALTER TABLE

- On peut enlever la contrainte d'une table en utilisant alter table, ici on va enlever la contrainte primary\_key
- `ALTER TABLE person2 DROP CONSTRAINT person2_pkey;`
- `insert into person2 (id, first_name, last_name, gender, email, date_of_birth, country_of_birth) values (1, 'first', 'last', 'Male', 'first@last.com', '1990-01-01', 'Italy');` → La même exacte commande qui ne marchait pas.
- `\d person`
- `select * from person2 where id = 1;`
- Remettre la clé primaire : `ALTER TABLE person2 ADD PRIMARY KEY (id);`
- Il faut supprimer les lignes pour rendre la clé primaire : `DELETE FROM person2 WHERE id = 1;`
- `ALTER TABLE person2 ADD PRIMARY KEY (id);`

# PostgreSQL

## UNIQUE CONSTRAINTS

- `select email, count(*) FROM person2 GROUP BY email;` → Remarquer le nombre de lignes vides
- `select email, count(*) FROM person2 GROUP BY email HAVING COUNT(*) > 1;`
- → Ajouter la même première ligne (le code peut être copier sur le fichier .sql) :

email	count
	315
(1 ligne)	

`insert into person2 (first_name, last_name, email, gender, date_of_birth, country_of_birth) values ('Perri', 'Ateridge', 'pateridge0@trellian.com', 'Bigender', '2021-04-04', 'Poland');`

En faisant se traitement deux fois, on essaie de créer une contrainte (unique):

- `ALTER TABLE person2 ADD CONSTRAINT unique_email_address UNIQUE (email);`

Comment faire pour que ça marche ?

- `SELECT * FROM person2 where email = ...`
- `DELETE FROM person2 WHERE id = ...`

Que remarquez vous dans la commande : `\d person2` ?

# PostgreSQL

## UNIQUE CONSTRAINTS

- Supprimer la contrainte : `ALTER TABLE person2 DROP CONSTRAINT unique_email_adress;`
- `\d person;`
- `ALTER TABLE person2 ADD UNIQUE (email);` ➔ Ceci est une deuxième manière de rajouter une contrainte en laissant postgres choisir le nom de la contrainte.
- `\d person;`

# PostgreSQL

## DELETE RECORDS

- Supprimer toutes les entrées : `DELETE FROM person2;`
- `select * from person2;`
- `\i C:/Users/Downloads/person2.sql;`
- `DELETE FROM person where id = 1033;`
- `DELETE FROM person2 where gender = 'Male' AND country_of_birth = 'France';`
- `SELECT * FROM person2 where gender = 'Male' AND country_of_birth = 'France';`
- `SELECT * FROM person2 where country_of_birth = 'France';`

# PostgreSQL

## UPDATE RECORDS

- Mettre à jour une information :

```
UPDATE person2 SET email = 'update@gmail.com' WHERE id = 2000;
```

```
select * from person2 WHERE id = 2000;
```

- Mettre à jour deux informations ou plus :

```
UPDATE person2 SET email = 'update2@gmail.com', first_name = 'professor' WHERE id = 2000;
```

```
select * from person2 WHERE id = 2000;
```

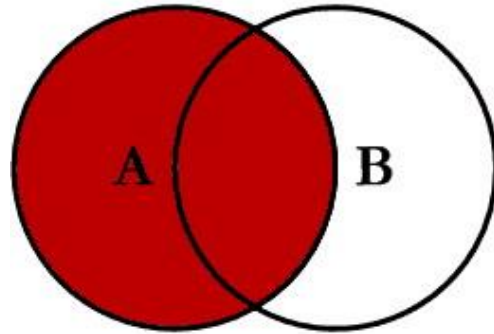


# JOINTURES

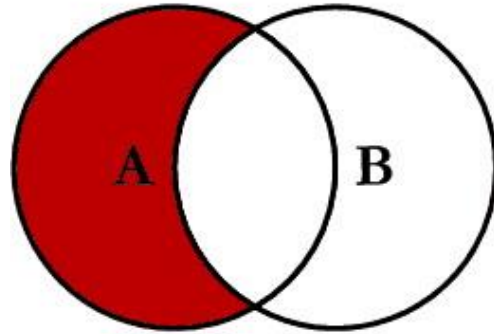
- Les jointures en SQL permettent d'associer plusieurs tables dans une même requête.
- Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.
- En général, les jointures consistent à associer des lignes de 2 tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table.
- Imaginons qu'une base de 2 données possède une table "utilisateur" et une autre table "adresse" qui contient les adresses de ces utilisateurs. Avec une jointure, il est possible d'obtenir les données de l'utilisateur et de son adresse en une seule requête.

# SQL JOINS

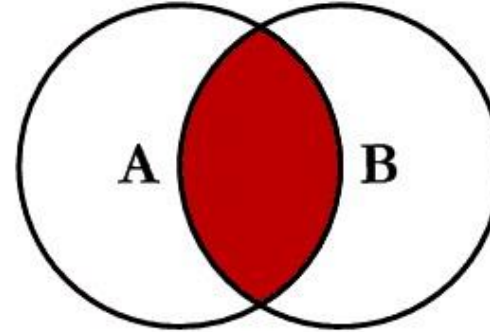
## JOINTURES



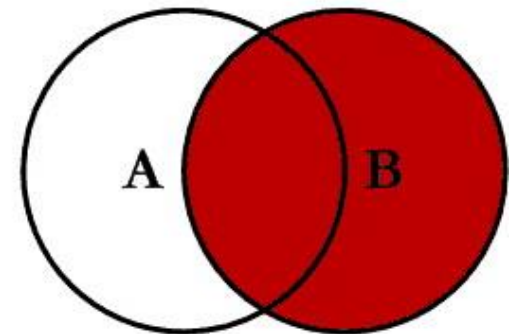
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



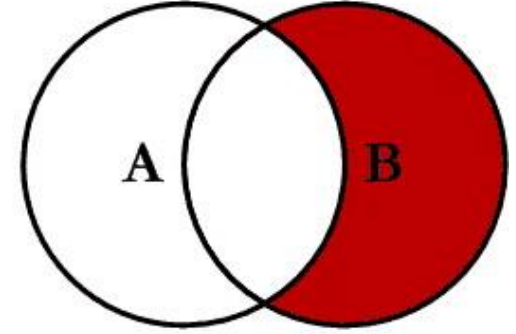
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



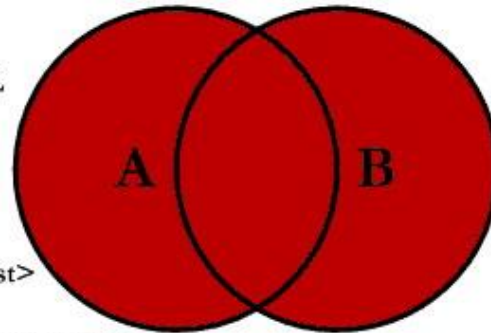
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



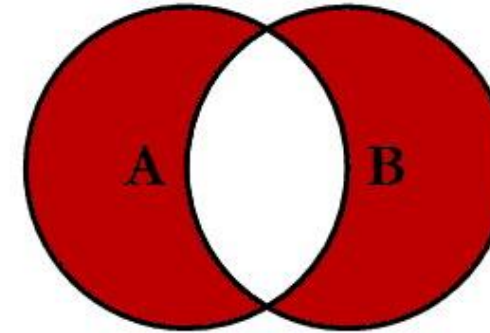
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
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
```

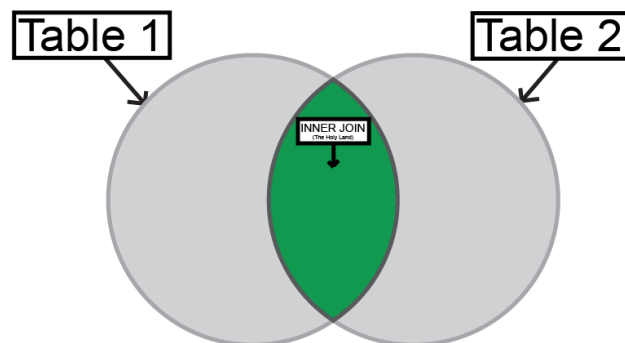
# PostgreSQL

## FOREIGN KEYS & RELATIONSHIPS

- Pour créer des foreign keys on va d'abord supprimer les table avec la fonction :DROP TABLE ...;
- Vérifier en utilisant : \dt → le résultat doit être vide.
- <https://github.com/achfeed/tuto-sql/blob/main/person-car.sql>
- \i C:/Users/ahamid/Downloads/person-car.sql
- \d person
- UPDATE person SET car\_id = 2 WHERE id = 1;
- UPDATE person SET car\_id = 1 WHERE id = 2;
- select \* from person; → Vous devez voir le numéro 2 apparaitre sur la clé 1.
- Pour connaître la voiture de Omar il faut faire : select \* from car; et regarder à quoi correspond son id.

# PostgreSQL

## INNER JOIN

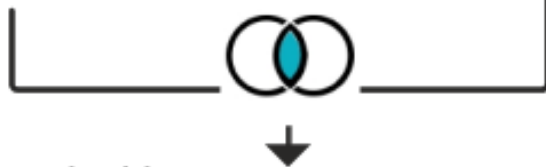


Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

ID	Country
3	Panama
4	Spain



Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

# PostgreSQL

## INNER JOIN

- Select \* from car;
- Select \* from person;
- Select \* from person JOIN car ON person.car\_id = car.id;

```
id | first_name | last_name | gender | email | date_of_birth | country_of_birth | car_id | id | make | price
---+-----+-----+-----+-----+-----+-----+-----+---+---+-----+
2 | Omar | Colmore | Male | | 1921-04-03 | Finland | 1 | 1 | Land Rover | 87665.38
1 | Fernanda | Beardon | Female | fernandab@is.gd | 1953-10-28 | Comoros | 2 | 2 | GMC | 17662.69
Acadia
```

- Meilleur affichage : rentrer la commande : \x
- Select \* from person JOIN car ON person.car\_id = car.id;



# PostgreSQL

## INNER JOIN

- Select first\_name, last\_name, make, model, price from person JOIN car ON person.car\_id = car.id;

# PostgreSQL

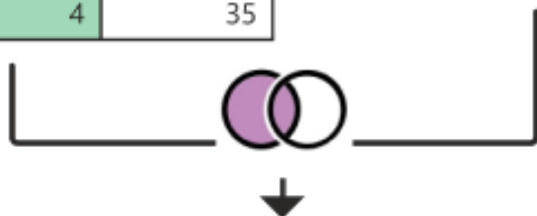
## LEFT / RIGHT JOINS

Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

ID	Country
1	USA
2	Canada
3	Panama



Merged Table

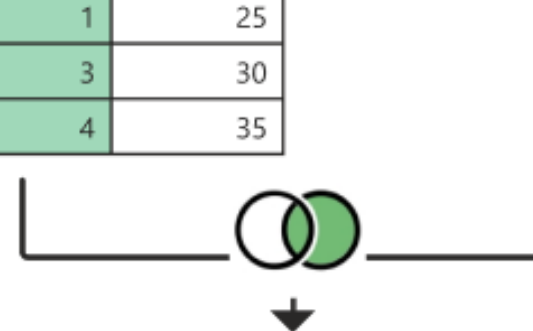
Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/3/2020	3	30	Panama
1/4/2020	4	35	<i>null</i>

Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

ID	Country
3	Panama



Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

# PostgreSQL

## LEFT / RIGHT JOINS

- `Select first_name, last_name, make, model, price from person LEFT JOIN car ON person.car_id = car.id;`
- La requête retourne toutes les lignes dans la table gauche avec les informations de la table droite.
- `Select first_name, last_name, make, model, price from person LEFT JOIN car ON person.car_id = car.id WHERE car.* is null;`
- Comment faire un right join ?



**PRACTICE  
MAKES  
PERFECT**