

Découverte des SGBD-R et du langage SQL





Définitions SGBD-R et SQL

- **BASE DE DONNÉE (relationnelle)** : Une base de données relationnelle (BDR) est un ensemble d'informations organisées selon des relations prédéfinies à l'aide de clés primaires et étrangères. Les données sont stockées dans une ou plusieurs tables, ce qui facilite la gestion et l'accès aux informations de manière structurée, similaire à un tableau
- **SGBD-R** : Un Système de Gestion de Base de Données Relationnelles (SGBDR) est un logiciel ou un programme utilisé pour créer, mettre à jour et gérer des bases de données relationnelles. Il offre des fonctionnalités pour la manipulation des données telles que l'insertion, la mise à jour, la suppression et la récupération des informations stockées dans la base de données. Le langage le plus utilisé par les SGBD-R est le langage SQL.
- **SQL** : Le langage SQL (structured query language) est un langage de requête structuré utilisé pour interagir avec les bases de données relationnelles. Il permet aux utilisateurs de définir, manipuler et interroger les données dans une base de données

I - La modélisation relationnelle

La modélisation relationnelle est la première étape, avant même de s'attaquer au SGBD-R.

Il s'agit d'organiser la future base de données (BDD) avec une structure appropriée. Je vais travailler uniquement avec des données structurées (type tableau excel).

Pour le projet, j'ai rempli un dictionnaire de données (ci-joint) pour comprendre les deux fichiers excel sur lesquelles je vais travailler. Ce dictionnaire répertorie les différentes colonnes ainsi que leur description, le type de variables (données), leurs tailles (contrainte) et les clés (déjà renseignées (définition slide suivant)).

	Nom des colonnes	Type de données	Taille	Clé	Description
CONTRAT.XLSX	Contrat_ID	INT		Clé primaire	Id unique pour les contrats
	No_voie	INT			Numéro dans la voie pour l'adresse du logement assuré
	B_T_Q	CHAR	1		Indicateur éventuel de répétition pour l'adresse du logement assuré sur un caractère
	Type_de_voie	VARCHAR			Type de voie pour l'adresse du logement assuré: rue, av (Avenue), rte (Route), ...
	Voie	VARCHAR			Libellé de la voie pour l'adresse du logement assuré
	Code_dep_code_commune	VARCHAR	5	Clé secondaire	Concaténation du code département et code commune pour avoir une clé unique
	Code_postal	INT			Code postal pour l'adresse du logement assuré
	Surface	INT			Surface du logement
	Type_local	VARCHAR			Type du logement (maison ou appartement)
	Occupation	VARCHAR			Statut des occupants (propriétaire ou locataire)
	Type_contrat	VARCHAR			Statut du logement (résidence principal/ secondaire ou mise en location)
	Formule	VARCHAR			Formule du contrat (intégral ou classique)
REGION.XLSX	Valeur_declaree_biens	VARCHAR			Valeurs du bien par tranches
	Prix_cotisation_mensuel	INT			Prix mensuel de la cotisation pour le contrat
	Code_dep_code_commune	VARCHAR	5	Clé primaire	Concaténation du code département et code commune pour avoir une clé unique
	reg_nom	VARCHAR			Nom de la région où se situe la commune
	dep_nom	VARCHAR			Nom du département où se situe la commune
	com_nom_maj	VARCHAR			Nom de la commune en majuscule
	dep_code	VARCHAR	3		Code du département de la commune
	reg_code	INT			Code de la région de la commune



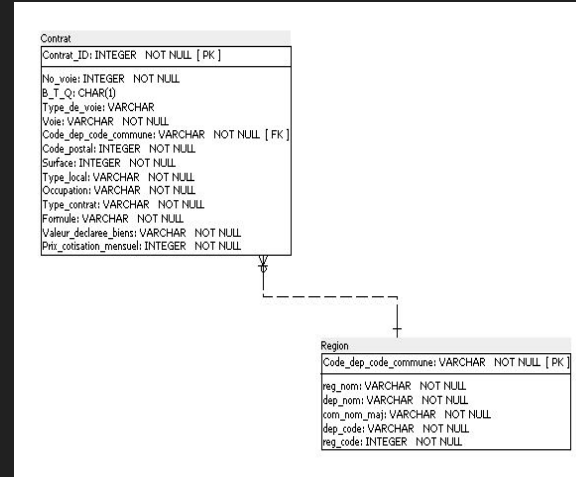
Clés Primaires et étrangères

- **Clé primaire** : C'est la donnée qui permet d'identifier de manière **unique** un enregistrement dans une table. Une clé primaire peut être composée d'une ou de plusieurs colonnes de la table. La clé primaire est indiquée par les initiales anglaises **PK**, pour « primary key »
- **Clé étrangère** : C'est une colonne (ou groupe de colonnes) d'une table qui fait référence à la **clé primaire** d'une autre table. Elle permet de modéliser le lien entre les lignes de ces deux tables. La clé étrangère est désignée par le signe **FK**, pour « Foreign Key ».
- **Clé artificielle** : C'est une colonne que l'on ajoute à la table. Cette colonne n'a pas de réelle signification (contrairement aux autres colonnes), mais sa seule fonction est d'identifier de manière unique les lignes de la table. C'est donc un identifiant. Dans le cas où la clé primaire est trop complexe (trop de colonnes, par exemple).

II - Le schéma relationnel normalisé

Cette étape vise à visualiser les tables de données que j'importerais dans notre SGBD-R, ainsi que les relations qu'elles entretiennent via les clés primaires et étrangères.

J'ai utilisé le logiciel SQL Power Architect pour créer le schéma joint, qui est un diagramme de classes faisant partie du langage de modélisation UML. Il me permet également de générer le code SQL pour la création des tables dans le SGBD-R.



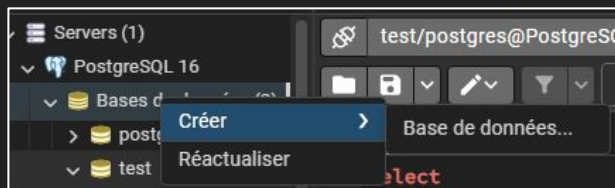
```
Preview SQL Script
Your Target Database is not configured.

CREATE TABLE public.Region (
    Code_dep_code_commune VARCHAR NOT NULL,
    reg_nom VARCHAR NOT NULL,
    dep_nom VARCHAR NOT NULL,
    com_nom_maj VARCHAR NOT NULL,
    dep_code VARCHAR NOT NULL,
    reg_code INTEGER NOT NULL,
    CONSTRAINT region_pk PRIMARY KEY
    (Code_dep_code_commune)
);

CREATE TABLE public.Contrat (
    Contrat_ID INTEGER NOT NULL,
    No_voie INTEGER NOT NULL,
    B_T_Q CHAR(1),
    Type_de_voie VARCHAR,
    Voie VARCHAR NOT NULL,
    Code_dep_code_commune VARCHAR NOT NULL,
    Code_postal INTEGER NOT NULL,
    Surface INTEGER NOT NULL,
    Type_local VARCHAR NOT NULL,
    Occupation VARCHAR NOT NULL,
    Type_contrat VARCHAR NOT NULL,
    Formule VARCHAR NOT NULL,
    Valeur_declaree_biens VARCHAR NOT NULL,
    Prix_cotisation_mensuel INTEGER NOT NULL,
    CONSTRAINT contrat_pk PRIMARY KEY
    (Contrat_ID)
);

ALTER TABLE public.Contrat ADD CONSTRAINT
region_contrat_fk
FOREIGN KEY (Code_dep_code_commune)
REFERENCES public.Region (Code_dep_code_commune)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

III - Création et chargement d'une BDD



	contrat_id [PK] integer	no_vois integer	b.t.q character	type_de_vois character varying	voies character varying
1	100601	190	A	RUE	CENTRALE
2	100602	347	[null]	RUE	DU CHATEAU
3	100603	58	[null]	AV	DU MONT BLAN
4	100604	140	[null]	RUE	DE L'ABBE JOLI

Total de lignes: 1000 sur 30335 Requête terminée 00:00:00.322

	reg_nom character varying	dep_nom character varying	code_dep_code_commune [PK] character varying	com_nom character varying
1	Grand Est	Aube	10001	ABBAYE SQ
2	Grand Est	Aube	10002	AILLEVILLE
3	Grand Est	Aube	10003	AIX VILLE
4	Grand Est	Aube	10004	ALLIBAUDIS

Total de lignes: 1000 sur 38916 Requête terminée 00:00:00.358

C'est à partir d'ici que je vais utiliser le SGBD-R. J'ai travaillé avec PostgreSQL.

Après installation et une première prise en main, j'ai créé une nouvelle base de données que j'ai nommée "test". Ensuite dans l'éditeur de requêtes j'utilise le code récupéré à l'étape précédente pour créer les tables (on peut également utiliser un clic droit comme pour la création de la BDD).

Pour finir, j'ai importé les données en suivant les étapes du menu contextuel "import/export de données..."

Résultats : J'ai une BDD avec deux tables "contrat" et "region" qui sont exploitables pour la suite du projet.

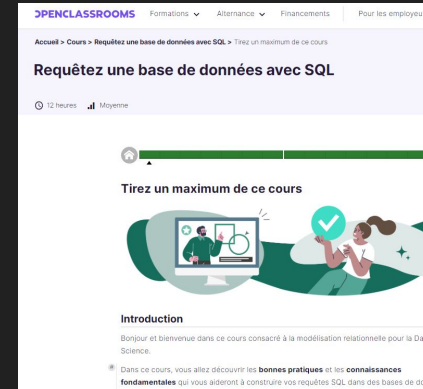
IV - Rédaction de requêtes SQL

Après avoir suivi le cours associé “requêtez une base de donnée avec SQL”, j’ai effectué les premières requêtes demandées. Pour cela j’ai commencé par faire un brouillon sur papier afin de dégager une logique.

Quelles tables/colonnes je vais avoir besoin pour répondre à la question

Est ce que je vais devoir effectuer un filtrage, un regroupement d’information, une jointure...

Est ce que je doit faire des calculs (moyenne, compter le nombre de lignes,...)



Requêtes SQL

Requête 1 : Lister les numéros de contrats (contrat_ID) avec leur surface pour la commune de Caen

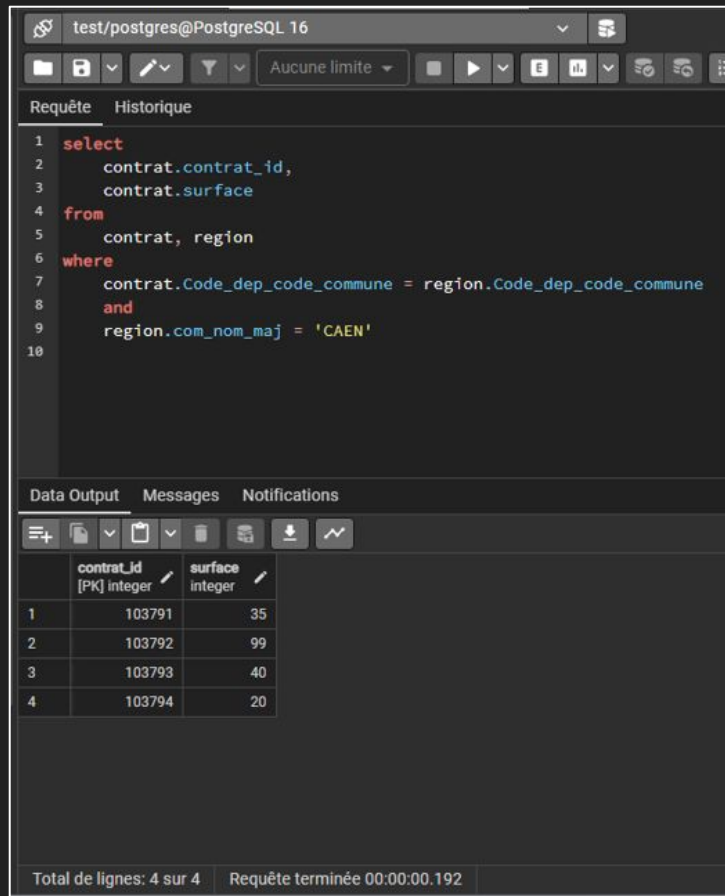
Cheminement logique :

Pour avoir cette information, je doit faire une jointure entre les deux tables car dans la table 'contrat' nous n'avons pas le nom des villes mais uniquement les codes communes.

Ensuite je sélectionne les colonnes demandées, à savoir les numéros de contrat et les surfaces. Pour finir j'effectue un filtrage sur le nom de commune pour garder uniquement "CAEN", sans oublier de mettre dans le filtrage la condition de jointure grâce à la clé étrangère présente dans la table contrat qui correspond à la clé primaire de la table région.

J'aurais pu également faire la jointure avec un "join on" plutôt que le mettre dans le where.

J'obtiens ainsi les identifiants des contrats pour la commune de CAEN, au nombre de 4, ainsi que les surface associé à chaque identifiant.



The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'test/postgres@PostgreSQL 16'. The query editor has tabs for 'Requête' and 'Historique'. The SQL query is as follows:

```
1 select
2   contrat.contrat_id,
3   contrat.surface
4 from
5   contrat, region
6 where
7   contrat.Code_dep_code_commune = region.Code_dep_code_commune
8   and
9   region.com_nom_maj = 'CAEN'
10
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with 4 rows and 2 columns: 'contrat_id [PK] integer' and 'surface integer'.

	contrat_id [PK] integer	surface integer
1	103791	35
2	103792	99
3	103793	40
4	103794	20

At the bottom of the interface, a status bar shows 'Total de lignes: 4 sur 4' and 'Requête terminée 00:00:00.192'.

Requêtes SQL

Requête 2 : Lister les numéros de contrats avec le type de contrat et leur formule pour les maisons du département 71. Cheminement logique similaire à la requête n°1.

test/postgres@PostgreSQL 16

Aucune limite

Requête

Historique

```

1 select
2     contrat.contrat_id,
3     contrat.type_contrat,
4     contrat.formule
5 from
6     contrat, region
7 where
8     contrat.code_dep_code_commune = region.code_dep_code_commune
9     and
10    contrat.type_local = 'Maison'
11    and
12    region.dep_code = '71'

```

Data Output

Messages

Notifications

	contrat_id [PK] integer	type_contrat character varying	formule character varying
1	114768	Residence principale	Integral
2	114779	Residence principale	Classique
3	114782	Residence principale	Classique
4	114812	Residence principale	Integral

Total de lignes: 4 sur 4

Requête terminée 00:00:00.086

le résultat obtenu est le suivant :
Pour le département 71, il y a 4 contrat où le type de local est une maison.

Requête 3: Lister le nom des régions de France.

Ici je peux travailler uniquement avec la table 'region' et avec la seule colonnes 'reg_nom', je n'ai pas besoin de faire de jointure. Je demande a avoir uniquement un retour sans doublons pour cette demande.

Ce qui me donne bien les 18 régions Française (13 métropolitaine et 5 d'outre-mers) plus une valeur pour les collectivités d'outre-mer soit 19 au total.

Requête 4 : Quels sont les 5 contrats qui ont les surfaces les plus élevées ?

Pour cette demande j'ai conservé toutes les colonnes de la table 'contrat'.

J'ai ensuite trié par ordre décroissant les valeurs la colonnes 'surface' puis je n'ai garder que premières pour avoir le résultats demandé.

Requête		Historique	
1	select distinct		
2	region.reg_nom		
3	from		
4	region		

Data Output		Messages		Notifications	
<div> </div>					
	reg_nom				
	character varying				
1	Hauts-de-France				
2	Île-de-France				
3	La Réunion				
4	Bretagne				
5	Mayotte				
15	Normandie				
16	Nouvelle-Aquitaine				
17	Occitanie				
18	Pays de la Loire				
Total de lignes: 19 sur 19					

```
1 select *
2 from contrat
3 order by contrat.surface desc
4 limit 5
```

Requêtes SQL

Requête 5 : Quel est le prix moyen de la cotisation mensuelle ?

J'ai besoin ici uniquement de la colonnes 'prix_cotisation_mensuel' de la table 'contrat'. J'applique juste la fonction "AVG" (average) pour calculer la moyenne de toutes les valeurs de la colonnes que j'ai arrondie à deux chiffres après la virgule pour une meilleure lisibilité et une meilleure exploitation dans le cadre d'analyse business, on parle en euros pour cette information.

Le résultat est donc : **19€33**

Requête	Historique
1	<code>select round (avg(contrat.prix_cotisation_mensuel),2)</code>
2	<code>from contrat</code>

Requête 6 : Quel est le nombre de contrats pour chaque catégorie de prix de la valeur déclarée des biens ? J'ai besoin de faire des agrégats pour chaque valeur de 'valeur_declaree_biens' via un group by et de créer une colonne appelé ici 'nombre_de_contrat' qui représentera la somme du nombre de contrat par groupe d'agrégat créé.

Requête	Historique
1	<code>select</code>
2	<code>contrat.valeur_declaree_biens,</code>
3	<code>count (*) as nombres_de_contrat</code>
4	<code>from</code>
5	<code>contrat</code>
6	<code>group by</code>
7	<code>contrat.valeur_declaree_biens</code>

	valeur_declaree_biens	nombres_de_contrat
	character varying	bigint
1	50000-100000	696
2	100000+	104
3	25000-50000	6815
4	0-25000	22720

Requête 7 : Quel est le nombre de formule "integral" sur la région Pays de la Loire ?

Pour répondre à cette demande, j'ai besoin de faire deux agrégation : une sur la colonne 'formule' de la table 'contrat' et une sur la colonne 'reg_nom' de la tables 'region'. Comme sur la requête précédente, j'ajoute une colonne qui donneras le nombre d'occurrences pour chaque groupe créé. Ensuite j'effectue un filtrage avec les valeurs demandé dans la question afin d'obtenir notre résultat.

Requête	Historique
1	<code>select</code>
2	<code>region.reg_nom,</code>
3	<code>contrat.formule,</code>
4	<code>count (*) as nombres_de_contrat</code>
5	<code>from</code>
6	<code>region, contrat</code>
7	<code>where</code>
8	<code>contrat.code_dep_code_commune = region.code_dep_code_commune</code>
9	<code>and</code>
10	<code>contrat.formule = 'Integral'</code>
11	<code>and</code>
12	<code>region.reg_nom = 'Pays de la Loire'</code>
13	<code>group by</code>
14	<code>region.reg_nom,</code>
15	<code>contrat.formule</code>

	reg_nom	formule	nombres_de_contrat
	character varying	character varying	bigint
1	Pays de la Loire	Integral	589

Requêtes SQL

Requête 8 : Lister les numéros de contrats avec le type de contrat et leur formule pour les maisons du département 71. exactement la même requête que la n°2.

Requête 9 : Quelle est la surface moyenne des contrats à Paris ?

Pour avoir cette information, j'ai besoin de la colonnes 'surface' de la table 'contrat' sur laquelle je vais effectuer un calcul de

moyenne arrondie à 2 chiffres après la virgule. Ensuite j'ai besoin de faire une jointure entre deux tables pour effectuer notre filtrage. Ce dernier est effectué en recherchant les occurrences pour la ville 'Paris' grâce à l'attribut **like** 'paris%' afin de prendre en compte tous arrondissements parisiens. J'ai rajouter la fonction **lower** a la

colonne 'com_nom_maj' afin de faciliter la recherche et d'éviter les erreurs. Ce qui nous donne comme résultats : une surface moyenne de 51.77 M² pour les contrat à Paris.

Requête









Historique

```
1 select round (avg(contrat.surface),2)
2 from contrat, region
3 where
4     contrat.code_dep_code_commune
5     = region.code_dep_code_commune
6     and
7     lower(region.com_nom_maj) like 'paris%'
```

Data Output

Messages

Notifications



	round numeric	🔒
1	51.77	

Requête 10 : Classements des 10 départements où le prix moyen de la cotisation est le plus élevé.

J'aurais besoin ici des colonnes des noms de département que je vais agréger et du prix des cotisations sur laquelle je vais calculer la moyenne. J'effectue également une jointure car les informations se trouvent dans nos deux tables. Pour finir je trie la colonne des prix moyens de cotisation par ordres décroissant et

je ne garde que les 10 premières valeurs pour obtenir le résultat attendu.

Requête Historique

```
1 select
2   region.dep_nom,
3   round (avg(contrat.prix_cotisation_mensuel),2) as cotisation_mensuel_moyen
4 from
5   contrat, region
6 where
7   contrat.code_dep_code_commune = region.code_dep_code_commune
8 group by
9   region.dep_nom
10 order by
11   cotisation_mensuel_moyen desc
12 limit
13   10
```

Data Output Messages Notifications

	dep_nom character varying	cotisation_mensuel_moyen numeric
1	Paris	36.40
2	Hauts-de-Seine	26.49
3	Val-de-Marne	20.31
4	Yvelines	18.83
5	Rhône	18.49
6	Ain	18.24
7	Alpes-Maritimes	18.14
8	Charente-Maritime	17.32
9	Haute-Savoie	17.15
10	Corse-du-Sud	17.07

Total de lignes: 10 sur 10 Requête terminée 00:00:00.154

Requêtes SQL

Requête 11 : Liste des communes ayant eu au moins 150 contrats.

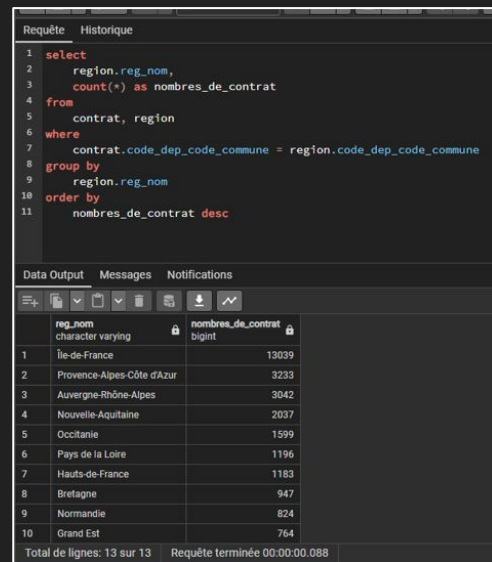
Je vais avoir une nouvelle fois besoin de faire une jointure afin d'obtenir les informations issue des deux tables. Je

vais effectuer une première fonction d'agrégation sur la colonne des nom de commune, puis une deuxième pour calculer la somme des contrats par groupe d'agrégats créés. Je fini cette requêtes par un filtrage dans les agrégations grâce à **'having'** pour ne garder que les résultats supérieurs ou égaux à 150, et je trie par ordre décroissant la colonne

nombre de contrats pour une meilleure lisibilité.

Requête 12 : Quel est le nombre de contrats pour chaque région ?

Je suis le même cheminement que la requête précédente : en effectuant une jointure, une agrégation ; cette fois ci sur le nom des régions ; et en rajoutant une colonnes calculant le nombre de contrat par région avec la fonction **'count'**. Je n'ai pas besoin d'effectuer de filtrage. Et toujours dans un souci d'une meilleure lisibilité, je trie par ordre décroissant la colonne nombre de contrats.



The screenshot shows a SQL query editor with a dark theme. The query is as follows:

```
1 select
2   region.reg_nom,
3   count(*) as nombres_de_contrat
4 from
5   contrat, region
6 where
7   contrat.code_dep_code_commune = region.code_dep_code_commune
8 group by
9   region.reg_nom
10 order by
11   nombres_de_contrat desc
```

Below the query, the results are displayed in a table with two columns: 'reg_nom' (character varying) and 'nombres_de_contrat' (bigint). The table contains 10 rows of data, sorted in descending order of the number of contracts.

reg_nom	nombres_de_contrat
Île-de-France	13039
Provence-Alpes-Côte d'Azur	3233
Auvergne-Rhône-Alpes	3042
Nouvelle-Aquitaine	2037
Occitanie	1599
Pays de la Loire	1196
Hauts-de-France	1183
Bretagne	947
Normandie	824
Grand Est	764

At the bottom of the results, it says: 'Total de lignes: 13 sur 13' and 'Requête terminée 00:00:00.088'.



Fonctions appprises et utiliser dans ce projet

- **SELECT** : chaque requête commençant par SELECT renverra une réponse qui sera toujours formée d'un unique tableau. permet de sélectionner la/les colonne(s) demandée(s).

- **FROM** : Pour sélectionner les tables où appartiennent les colonnes rechercher du SELECT.

- **WHERE** : Permet de filtrer les lignes d'une table selon un certain critère. = égale, < inférieur,...

- **LES OPÉRATEURS LOGIQUES** : Pour combiner plusieurs conditions de filtres grâce au OR (ou), le AND (et) et le NOT (non).

- **JOIN** : Pour lier deux tables avec une jointure. join on pour une jointure interne. left/right/full outer join on pour une jointure gauche/droite/entière externe.

- **GROUP BY** : Pour faire des agrégations sur les lignes

- **HAVING** : Permet le filtrages dans les groupes agréger défini dans le group by.

- **ORDER BY** : Pour trier la/les colonne(s) par ordres croissant ou décroissant.

- **LIKE** : permettre d'effectuer une recherche dans des chaînes de caractères. S'utilise avec '_' pour remplacer un caractère inconnue ou avec '%' pour remplacer 0,1 ou plusieurs caractères.

- **FONCTIONS DIVERS** : count pour calculer le nombre de ligne, AVG (average) pour calculer une moyenne, round pour faire un arrondie, lower pour mettre un texte en minuscule.