Données 24 – SQL : SELECT FROM WHERE

I – DB Brownser

Il s'agit d'un Système de **Gestion de Bases de Données** (**SGBD**) **open source** ne gèrant que **SQLite**.
SQL veut dire **Structured Query Language**. C'est un **langage déclaratif.**

II – PROJECTION : SELECT xxx FROM xxx

Exemple de déclaration de relation / table

→ Implémentation SQL du schéma relationnel de la relation **titre** *(Hors programme)*

```
1   CREATE TABLE titre (
2     id INTEGER,
3     title_type VARCHAR(15) NOT NULL,
4     primary_title VARCHAR(40),
5     original_title VARCHAR(40),
6     start_year INTEGER,
7     end_year INTEGER,
8     runtime_minutes INTEGER,
9     genres TEXT,
10  PRIMARY KEY (id AUTOINCREMENT)
11  );
```

PRIMARY KEY : équivalent à NOT NULL et UNIQUE
On respecte la contrainte d'**UNICITE**.
En plaçant AUTOINCREMENT, on laisse le SGBD gérer cette valeur tout seul.

NOT NULL : on demande au SGBD de vérifier que chaque champ de cet attribut/colonne possède bien une vraie valeur.t NULL est interdit pour cet attribut.

UNIQUE : on demande au SGBD de vérifier que cette valeur n'apparaît bien qu'une seule fois dans cette colonne. **NULL** multiple est possible par contre.

→ Schéma relationnel abstrait de la relation **titre (au programme)**

```
titre ( id INTEGER, title_type VARCHAR(15),                    id souligné car clé primaire.
        primary_title VARCHAR(40), original_title VARCHAR(40),
        start_year INTEGER, end_year INTEGER, runtime_minutes INTEGER, genres TEXT )
```

2.1 Projection avec `SELECT a FROM b` → Choix des attributs (colonnes)

L'opération de projection consiste à récupérer un sous-ensemble des attributs (colonnes).
Attention, l'ordre fourni derrière SELECT a une influence sur l'ordre des données obtenues.

`SELECT primary_title, runtime_minutes` `FROM titre ;`	Projette titre et durée des n-uplets de la table titre

2.2 Clause `DISTINCT` (normalement, il y aurait un rappel avant utilisation le jour du BAC)

`SELECT DISTINCT title_type` `FROM titre;`	Projette un unique exemplaire du type de production des n-uplets de la table titre

III – FILTRAGE : SELECT a FROM b WHERE c

3.1 Clause `WHERE` → Choix des n-uplets (lignes)

La clause **WHERE** permet de **restreindre / filter la sélection des n-uplets** (lignes) récupérés en précisant des conditions.

`SELECT primary_title, runtime_minutes`
`FROM titre`
`WHERE start_year = 2020;`
	Projette titre et durée
	des n-uplets de la relation titre
	qui sont sortis en 2020.

`SELECT primary_title`
`FROM titre`
`WHERE start_year <> 2020 AND runtime_minutes < 120 ;`
	Projette le titre
	des n-uplets de la relation titre
	(qui ne sont pas sortis en 2020) et (qui durent - de 2h).

- Egalité `=` *(== en Python)* — `WHERE start_year = 2020 ;`
- Différence `<>` *(!= en Python)* — `WHERE start_year <> 2020 ;`
- Autres comparaisons `<`, `<=`, `>`, `>=` — `WHERE runtimes_minutes >= 90 ;`
- Opérateurs logiques `AND`, `OR`, `NOT` — `WHERE end_year = 2020 AND start_year = 2018 ;`
- Opérateurs arithmétiques `+`, `-`, `*`, `/`, `%` — `WHERE end_year = start_year + 5 ;`
- Contient une chaîne de caractères — `WHERE primary_title LIKE '%aliens%' ;` *(% symbolise tout)*
- Appartient à un ensemble de valeurs — `WHERE title_type IN ('movie', 'tvSeries') ;`

3,2 – Priorités du AND sur le OR

Nous avions vu en 1er que `a AND b` est équivalent à `a . b` dans l'algèbre de Boole.
Nous avions vu en 1er que `a OR b` est équivalent à `a + b` dans l'algèbre de Boole 0 est Faux, **le reste est Vrai**
a + b – a.b 0 est Faux, 1 est Vrai

IV – Commande **ORDER BY** *(normalement rappel avant utilisation)*

La commande **ORDER BY** permet de trier les n-uplets transmis par la requête SQL.
On pourra trier les données sur une ou plusieurs colonnes, et par ordre croissant ou decroissant.

4.1 Tri simple croissant : suffixe ASC ou rien

```
1    SELECT primary_title, start_year
2    FROM titre
3    WHERE title_type = "movie"
4    ORDER BY start_year;
```

```
1   SELECT primary_title, start_year
2   FROM titre
3   WHERE title_type = "movie"
4   ORDER BY start_year ASC;
```

4.2 Tri en ordre décroissant : suffixe **DESC**.

```
1    SELECT primary_title, start_year
2    FROM titre
3    WHERE title_type = "movie"
4    ORDER BY start_year DESC;
```

```
1   SELECT primary_title, start_year
2   FROM titre
3   WHERE title_type = "movie" AND start_year <> ''
4   ORDER BY start_year DESC;
```

4.3 Tri multiple

Exemple : les n-uplets sont triés par **start_year** décroissant puis en cas d'égalité entre **start_year**, on affiche par ordre alphabétique croissant du **primary_title**.

```
1   SELECT primary_title, start_year
2   FROM titre
3   WHERE title_type = "movie" AND start_year <> ''
4   ORDER BY start_year DESC, primary_title ASC;
```

V – Fonctions d'agrégation

ATTENTION : un seul n-uplet ramené

5.1 MAX()

On projète le premier n-uplet qui correspond à ce maximum.
Pour trouver la durée de film la plus longue :

`SELECT primary_title, MAX(runtime_minutes)`
`FROM titre`
`WHERE title_type = 'movie';`
	Projette **l'un des titres** de durée maximale, et cette durée obtenue des n-uplets de la relation titre qui sont des films.

5.2 MIN

Idem, mais avec la valeur plus petite et toujours le premier enregistrement éventuel.

`SELECT primary_title, MIN(runtime_minutes)`
`FROM titre`
`WHERE title_type = 'movie';`
	Projette **l'un des titres** de durée minimale, et cette durée obtenue des n-uplets de la relation titre qui sont des films.

5.3 AVG()

Pour trouver la durée moyenne des films. Là, aucun intérêt à chercher autre chose que la valeur souvent.

`SELECT AVG(runtime_minutes)`
`FROM titre`
`WHERE title_type = 'movie';`
	Projette la durée moyenne calculée sur les n-uplets de la relation titre qui sont des films.

5.4 SUM()

Pour trouver la durée totale des films produits en 2020 par exemple :

`SELECT SUM(runtime_minutes)`
`FROM titre`
`WHERE title_type = 'movie';`
	Projette la somme des durées calculée à partir des n-uplets de la relation titre qui sont des films.

`SELECT SUM(runtime_minutes) AS total`
`FROM titre`
`WHERE title_type = 'movie';`
	Projette sous le nom de « total » la somme des durées calculée à partir des n-uplets de la relation titre qui sont des films.

5.5 COUNT()

Cette fonction permet de compter le nombre de n-uplets renvoyés par notre requête.

`SELECT start_year, COUNT(*)`
`FROM titre`
`WHERE title_type = 'movie' AND start_year = 2000;`
	Projette l'année recherchée et le nombre de n-uplets récupérés des n-uplets de la relation titre qui sont des films sortis en 2000.

HORS PROGRAMME – Requêtes imbriquées

Comment afficher tous les films de durée maximale ?

On commence par chercher le maximum.
Puis on utilise cette première requête en l'intégrant à l'intérieur d'un autre requête.

1 - Recherche du maximum :

```
SELECT MAX(runtime_minutes)   FROM titre   WHERE title_type = 'movie'
```

2 - Imbrication de cette reqûete dans une autre :

```
SELECT primary_title, runtime_minutes
FROM titre
WHERE runtime_minutes = ( SELECT MAX(runtime_minutes) FROM titre WHERE title_type = 'movie' );
```

HORS PROGRAMME – clause WITH

La clause WITH permet de structurer les requêtes en stockant des résultats intermédiaires dans des alias plutôt que de faire des SELECT de SELECT.

```
WITH duree AS (SELECT MAX(runtime_minutes) FROM titre WHERE title_type = 'movie')

SELECT primary_title, runtime_minutes
FROM titre
WHERE runtime_minutes = duree
```

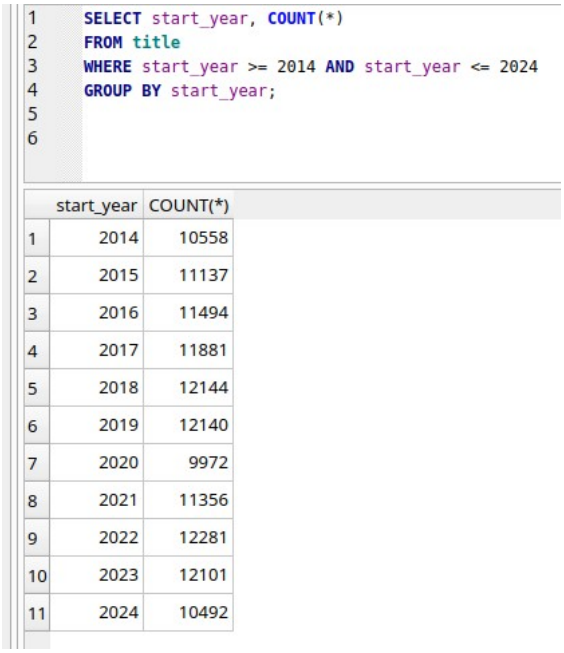
HORS PROGRAMME – clause GROUP BY

Lorsqu'on utilise les fonctions d'agrégation, on peut vouloir regrouper par exemple les films par année pour ensuite projeter le nombre produit en 2020, 2021 et 2022 par exemple.
On peut bien entendu faire 3 requêtes différentes et stocker les résultats.
Mais on peut faire mieux : on peut former des paquets de n-uplets en le regroupant selon un certain critère, puis de lancer les fonctions d'agrégation sur chacun de ces paquets plutôt que sur l'ensemble des n-uplets.

Exemple :

```
SELECT start_year, COUNT(*)
FROM title
WHERE start_year >= 2014 AND start_year <= 2024
GROUP BY start_year;
```

On voit qu'on projète l'année et le nombre de productions
correspondant à cette année
en cherchant dans la table title
les n-uplets dont l'année est entre 2014 et 2024
et en le regroupant par année.



```
1  SELECT start_year, COUNT(*)
2  FROM title
3  WHERE start_year >= 2014 AND start_year <= 2024
4  GROUP BY start_year;
5
6
```

	start_year	COUNT(*)
1	2014	10558
2	2015	11137
3	2016	11494
4	2017	11881
5	2018	12144
6	2019	12140
7	2020	9972
8	2021	11356
9	2022	12281
10	2023	12101
11	2024	10492