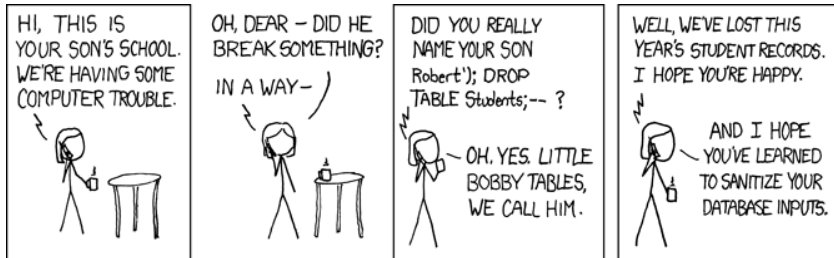


SQL 1 : Requêtes sur une table

SELECT, WHERE, DISTINCT, ORDER BY, AS

Quentin Fortier

September 14, 2022



Gérer beaucoup de données

Problème : comment gérer de façon efficace un grand nombre de données?

Gérer beaucoup de données

Problème : comment gérer de façon efficace un grand nombre de données?

Exemples :

- ❶ Les astres connus, avec leur taille, poids...
- ❷ Les espèces connues, avec leur taxonomie, famille...
- ❸ Les utilisateurs d'un site web, avec leur mot de passe, préférences...
- ❹ Les élèves d'un lycée, avec leurs notes, options...

Quelques solutions

On peut stocker ces données dans :

- ① Un tableur : trop limité.
- ② Une liste Python : trop désorganisé.
- ③ Une **base de donnée**.

Base de donnée

Une **base de donnée** est un ensemble de **tables**. On peut y extraire des informations à l'aide de **requêtes**.

Base de donnée

Base de donnée

Une **base de donnée** est un ensemble de **tables**. On peut y extraire des informations à l'aide de **requêtes**.

Table

Une **table** est un tableau à 2 dimensions dont les colonnes sont les **attributs** et les lignes les **enregistrements**.

Base de donnée

Base de donnée

Une **base de donnée** est un ensemble de **tables**. On peut y extraire des informations à l'aide de **requêtes**.

Table

Une **table** est un tableau à 2 dimensions dont les colonnes sont les **attributs** et les lignes les **enregistrements**.

Exemple : une base de donnée astre contient deux tables planete et etoile.

La table planete possède des attributs nom, rayon, poids...

Chaque enregistrement de planete correspond aux informations sur une planète.

Base de donnée astre

nom	rayon (km)	poids (kg)	etoile
'Terre'	6400	6×10^{24}	'Soleil'
'Jupiter'	70000	2×10^{27}	'Soleil'
'Proxima b'	?	?	'Proxima Centauri'
...

Table planete

nom	type	duree_vie (année)	galaxie
'Soleil'	'Naine jaune'	10^{10}	'Voie lactée'
'Proxima Centauri'	'Naine rouge'	?	'Voie lactée'
...

Table etoile

Domaine

Chaque attribut a un **domaine** : l'ensemble des valeurs que peut prendre cet attribut.

Chaque attribut a un **domaine** : l'ensemble des valeurs que peut prendre cet attribut.

Dans la table `planete` :

- ① `nom` est un attribut ayant pour domaine l'ensemble des chaînes de caractères.
- ② `rayon` est un attribut ayant pour domaine \mathbb{N} .
- ③ ...

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.

Parmi les clés possibles on en choisit une qu'on nomme **clé primaire**.

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.

Parmi les clés possibles on en choisit une qu'on nomme **clé primaire**.

nom	type	duree_vie (année)	galaxie
'Soleil'	'Naine jaune'	10^{10}	'Voie lactée'
'Proxima Centauri'	'Naine rouge'	?	'Voie lactée'
'Kepler-22'	'Naine jaune'	?	'Voie lactée'

Table étoile

Clés possibles?

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.

Parmi les clés possibles on en choisit une qu'on nomme **clé primaire**.

nom	type	duree_vie (année)	galaxie
'Soleil'	'Naine jaune'	10^{10}	'Voie lactée'
'Proxima Centauri'	'Naine rouge'	?	'Voie lactée'
'Kepler-22'	'Naine jaune'	?	'Voie lactée'

Table étoile

Clés possibles? **nom**

nom	pays	latitude	longitude
'Hanoï'	'Viêt Nam'	21°	104°
'Valence'	'France'	45°	5°
'Valence'	'Espagne'	39°	0°
'Quito'	'Equateur'	0°	-78°
'Singapour'	'Singapour'	0°	104°
'Valence'	'France'	45°	1°

Table ville

Clés possibles?

nom	pays	latitude	longitude
'Hanoï'	'Viêt Nam'	21°	104°
'Valence'	'France'	45°	5°
'Valence'	'Espagne'	39°	0°
'Quito'	'Equateur'	0°	-78°
'Singapour'	'Singapour'	0°	104°
'Valence'	'France'	45°	1°

Table ville

Clés possibles?

- ① latitude, longitude
- ② nom, longitude
- ③ pays, longitude

On peut résumer la structure d'une table par son schéma :

Schéma

Le schéma d'une table est la donnée de ses attributs, des domaines des attributs et de l'éventuelle clé primaire (soulignée), sous la forme :

table (attribut_1 : type_1, ..., attribut_n : type_n)

On peut résumer la structure d'une table par son schéma :

Schéma

Le schéma d'une table est la donnée de ses attributs, des domaines des attributs et de l'éventuelle clé primaire (soulignée), sous la forme :
table (attribut_1 : type_1, ..., attribut_n : type_n)

Par exemple, le schéma de la table ville avec comme clé primaire (latitude, longitude) est :

ville (nom : chaîne de caractères, pays : chaîne de caractères,
latitude : entier, longitude : entier)

On accède à des informations d'une base de donnée avec un **langage de requêtes**.

Contrairement à un langage de programmation :

- 1 On ne va pas utiliser de variable, boucle...
- 2 On se contente de demander ce que l'on veut obtenir, mais il n'y a pas besoin de dire **comment** l'obtenir : la machine se débrouille.

Langage de requêtes / de programmation

Pour trouver la somme des masses des planètes du système solaire :

① Langage de programmation :

Somme = 0

Pour toute planete p :

 Si p tourne autour du Soleil :

 Augmenter Somme du poids de p

② Langage de requête :

Obtenir la somme des poids des planetes
qui tournent autour du Soleil

Quelques règles en SQL :

- ❶ Chaque requête doit être terminée par un point-virgule ;
- ❷ Pas d'indentation obligatoire comme en Python, mais il est conseillé de bien présenter son code
- ❸ Les commandes peuvent être écrites en majuscules ou minuscules
- ❹ Il est conseillé d'écrire les commandes SQL en majuscules et de donner des noms de tables et colonnes en minuscules

Les attributs peuvent être de type :

- INT : entier
- CHAR(*k*) : chaîne d'au plus *k* caractères
Une chaîne de caractère doit être entourée de guillemets ("exemple") ou apostrophes ('exemple')
- FLOAT : nombre à virgule
- BOOLEAN : booléen (en fait soit 0 soit 1)

Création d'une table

Remarque : la création de table n'est pas au programme.

Exemple : je veux créer une table de mes élèves avec leur nom, prénom, classe, option, école intégrée.

Création d'une table

Remarque : la création de table n'est pas au programme.

Exemple : je veux créer une table de mes élèves avec leur nom, prénom, classe, option, école intégrée.

Il n'y a pas de clé possible! Dans ce cas, on peut créer un attribut qui fera office de clé primaire.

On peut dire quelle est la clé primaire en écrivant :

PRIMARY KEY (clé) dans la création de table.

Création d'une table

```
CREATE TABLE eleve (  
  id INT AUTO_INCREMENT,  
  PRIMARY KEY (id),  
  nom CHAR(20),  
  prenom CHAR(20),  
  annee_entree INT,  
  option_info BOOLEAN,  
  classe_sup CHAR(5),  
  classe_spe CHAR(5),  
  classe_spe2 CHAR(5),  
  ecole CHAR(20)  
);
```

Création d'une table

```
INSERT INTO eleve (nom, prenom, classe_sup, annee_entree, option_info)
VALUES ('Turing', 'Alan', 'MPSI2', 2015, TRUE),
       ('Gödel', 'Kurt', 'MPSI1', 2015, TRUE),
       (NULL, 'Euclide', 'MPSI2', 2015, FALSE),
       ('Newton', 'Isaac', 'PCSI2', 2015, FALSE),
       ('Curie', 'Marie', 'PCSI1', 2015, FALSE);
```

SELECT

Pour afficher des colonnes d'une table, on utilise :

```
SELECT colonne_1, ..., colonne_n FROM table;
```

SELECT

Par exemple, pour obtenir seulement les noms et prénoms des élèves :

```
SELECT nom, prenom FROM eleve;
```

SELECT

Par exemple, pour obtenir seulement les noms et prénoms des élèves :

```
SELECT nom, prenom FROM eleve;
```

On obtient :

nom	prenom
Turing	Alan
Gödel	Kurt
NULL	Euclide
Newton	Isaac
Curie	Marie

SELECT

Par exemple, pour obtenir seulement les noms et prénoms des élèves :

```
SELECT nom, prenom FROM eleve;
```

On obtient :

nom	prenom
Turing	Alan
Gödel	Kurt
NULL	Euclide
Newton	Isaac
Curie	Marie

Pour afficher la table entière, on peut utiliser * plutôt que donner le nom de chaque colonne :

```
SELECT * FROM eleve;
```

SELECT

On peut faire des calculs dans les requêtes :

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3))  
FROM planete;
```

Que fait cette requête?

SELECT

On peut faire des calculs dans les requêtes :

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3))  
FROM planete;
```

Que fait cette requête?

Elle affiche le nom et la densité de chaque planète.

SELECT

On peut faire des calculs dans les requêtes :

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3))  
FROM planete;
```

Que fait cette requête?

Elle affiche le nom et la densité de chaque planète.

Exercice

L'attribut rayon est en km.

Écrire une requête pour afficher le rayon de chaque planète en mètres.

DISTINCT

DISTINCT permet d'éviter d'avoir des doublons :

```
SELECT ecole FROM eleves;  
-- affiche plusieurs fois la même école  
-- (si plusieurs élèves l'ont intégré)
```

```
SELECT DISTINCT ecole FROM eleves;  
-- affiche une fois chaque école
```

Fonctions d'agrégations

On peut utiliser des fonctions sur un attribut :

- **SUM(a)** : Somme l'attribut a sur tous les enregistrements.

Exemple : **SELECT SUM(population) FROM world**

Fonctions d'agrégations

On peut utiliser des fonctions sur un attribut :

- **SUM(a)** : Somme l'attribut a sur tous les enregistrements.

Exemple : **SELECT SUM(population) FROM world**

- **MIN(a) / MAX(a)** : Minimum/maximum de a sur tous les enregistrements.

Exemple : **SELECT MAX(population) FROM world**

Fonctions d'agrégations

On peut utiliser des fonctions sur un attribut :

- **SUM(a)** : Somme l'attribut a sur tous les enregistrements.
Exemple : **SELECT SUM(population) FROM world**
- **MIN(a) / MAX(a)** : Minimum/maximum de a sur tous les enregistrements.
Exemple : **SELECT MAX(population) FROM world**
- **COUNT(a)** : Compte le nombre de fois que a est différent de **null**.
Souvent on compte le nombre total d'enregistrements avec **COUNT(*)**.
Exemple :
SELECT COUNT(*) FROM eleve WHERE classe_sup = 'MP2I'.

Fonctions d'agrégations

On peut utiliser des fonctions sur un attribut :

- **SUM(a)** : Somme l'attribut a sur tous les enregistrements.
Exemple : **SELECT SUM(population) FROM world**
- **MIN(a)** / **MAX(a)** : Minimum/maximum de a sur tous les enregistrements.
Exemple : **SELECT MAX(population) FROM world**
- **COUNT(a)** : Compte le nombre de fois que a est différent de **null**.
Souvent on compte le nombre total d'enregistrements avec **COUNT(*)**.
Exemple :
SELECT COUNT(*) FROM eleve WHERE classe_sup = 'MP2I'.
- **AVG(a)** : Moyenne de l'attribut a.
Exemple :
SELECT AVG(note) FROM eleve WHERE classe_sup = 'MP2I'.

Fonctions d'agrégations

On peut utiliser des fonctions sur un attribut :

- **SUM(a)** : Somme l'attribut a sur tous les enregistrements.
Exemple : **SELECT SUM(population) FROM world**
- **MIN(a) / MAX(a)** : Minimum/maximum de a sur tous les enregistrements.
Exemple : **SELECT MAX(population) FROM world**
- **COUNT(a)** : Compte le nombre de fois que a est différent de **null**.
Souvent on compte le nombre total d'enregistrements avec **COUNT(*)**.
Exemple :
SELECT COUNT(*) FROM eleve WHERE classe_sup = 'MP2I'.
- **AVG(a)** : Moyenne de l'attribut a.
Exemple :
SELECT AVG(note) FROM eleve WHERE classe_sup = 'MP2I'.

Il est possible de renommer une colonne avec AS :

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3)) AS densite  
FROM planete;
```


Il est possible de renommer une colonne avec AS :

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3)) AS densite  
FROM planete;
```

Utile pour y faire référence!

WHERE

Il est possible de récupérer seulement les enregistrements vérifiant une condition avec WHERE :

```
SELECT colonne_1, ..., colonne_n FROM table  
WHERE condition;
```

WHERE

Il est possible de récupérer seulement les enregistrements vérifiant une condition avec WHERE :

```
SELECT colonne_1, ..., colonne_n FROM table  
WHERE condition;
```

Dans condition on peut utiliser :

- ❶ = (et non pas ==)
- ❷ <, <=
- ❸ != (ou son équivalent <>)
- ❹ AND, OR
- ❺ LIKE

WHERE

Pour afficher les noms des élèves qui ont été en MPSI2, on écrira :

WHERE

Pour afficher les noms des élèves qui ont été en MPSI2, on écrira :

```
SELECT nom FROM eleve WHERE classe_sup = 'MPSI2';
```

Pour afficher les noms des élèves qui sont passés de MPSI à PSI :

WHERE

Pour afficher les noms des élèves qui ont été en MPSI2, on écrira :

```
SELECT nom FROM eleve WHERE classe_sup = 'MPSI2';
```

Pour afficher les noms des élèves qui sont passés de MPSI à PSI :

```
SELECT nom FROM eleve  
WHERE (classe_sup = 'MPSI1' OR classe_sup = 'MPSI2')  
      AND classe_spe = 'PSI';
```

Comment afficher les planètes de rayon supérieur à 50000 km?

WHERE

LIKE permet d'établir une condition sur la forme d'une chaîne de caractères d'un attribut :

```
attribut LIKE motif
```

WHERE

LIKE permet d'établir une condition sur la forme d'une chaîne de caractères d'un attribut :

`attribut LIKE motif`

motif doit être une chaîne de caractères qui peut contenir :

- % : pour n'importe quelle chaîne de caractères
- _ : pour n'importe quel (unique) caractère

WHERE

Que fait la requête suivante?

```
SELECT * FROM eleve WHERE ecole LIKE 'Centrale%';
```

ORDER BY

ORDER BY permet de trier dans l'ordre croissant les enregistrements en fonction d'un attribut. On peut ajouter DESC pour trier dans l'ordre décroissant.

Exemples :

```
SELECT nom FROM eleve ORDER BY nom;
```

ORDER BY

ORDER BY permet de trier dans l'ordre croissant les enregistrements en fonction d'un attribut. On peut ajouter DESC pour trier dans l'ordre décroissant.

Exemples :

```
SELECT nom FROM eleve ORDER BY nom;
```

les noms d'élèves par ordre alphabétique

```
SELECT * FROM planete  
WHERE etoile = 'Soleil'  
ORDER BY poids DESC;
```

ORDER BY

ORDER BY permet de trier dans l'ordre croissant les enregistrements en fonction d'un attribut. On peut ajouter DESC pour trier dans l'ordre décroissant.

Exemples :

```
SELECT nom FROM eleve ORDER BY nom;
```

les noms d'élèves par ordre alphabétique

```
SELECT * FROM planete  
WHERE etoile = 'Soleil'  
ORDER BY poids DESC;
```

les planètes du système solaire de la plus lourde à la plus légère

LIMIT

`LIMIT k` permet de limiter le nombre d'enregistrements aux k premières valeurs. Il est souvent utilisé avec `ORDER BY`.

LIMIT

LIMIT k permet de limiter le nombre d'enregistrements aux k premières valeurs. Il est souvent utilisé avec ORDER BY.

```
SELECT * FROM planete  
WHERE etoile = 'Soleil'  
ORDER BY poids DESC  
LIMIT 3;
```

Donne les 3 planètes les plus lourdes du système solaire.

OFFSET

OFFSET p permet d'afficher les enregistrements à partir du $(p + 1)$ ème.
Il est souvent utilisé avec ORDER BY.

OFFSET

OFFSET p permet d'afficher les enregistrements à partir du $(p + 1)$ ème.
Il est souvent utilisé avec ORDER BY.

```
SELECT * FROM planete  
WHERE etoile = 'Soleil'  
ORDER BY poids  
LIMIT 1  
OFFSET 2;
```

Donne la 3ème planète la plus légère du système solaire.

Récapitulatif

Toutes les commandes optionnelles de SELECT doivent être **écrites dans cet ordre** :

```
SELECT colonne_1, ..., colonne_n  
FROM nom_table  
WHERE conditions  
ORDER BY colonne_i  
LIMIT k  
OFFSET p;
```

`planete (nom, poids, rayon)`

Question

Comment obtenir, dans la table `planete`, la deuxième planète la plus dense connue?

Exercices

`planete (nom, poids, rayon)`

Question

Comment obtenir, dans la table `planete`, la deuxième planète la plus dense connue?

`eleve (nom, annee_entree, ecole, classe_sup,
 classe_spe, classe_spe2)`

Question

Comment obtenir, dans la table `eleve`, les noms des 3 derniers élèves entrés dans une ENS en MP*?

NULL

NULL est un mot-clé signifiant « valeur manquante ».

NULL

NULL est un mot-clé signifiant « valeur manquante ».

Pour savoir si un attribut est NULL :

WHERE ... IS NULL

(et non pas WHERE ... = NULL)

NULL

NULL est un mot-clé signifiant « valeur manquante ».

Pour savoir si un attribut est **NULL** :

WHERE ... IS NULL

(et non pas **WHERE ... = NULL**)

Pour savoir si un attribut n'est pas **NULL** :

WHERE ... IS NOT NULL

(et non pas **WHERE ... <> NULL**)