Bases de données : langage SQL

Pour effectuer une requête dans une base de données via un SGBD, on va utiliser un langage de requêtes nommé SQL (Structured Query Language).

Le langage SQL permet d'effectuer des opérations sur la base (créer une table à l'aide d'un schéma, modifier le schéma d'une table...) mais nous verrons uniquement la partie du langage permettant de rechercher des informations dans une base.

Les requêtes SQL commencent par le mot clé `SELECT` et se terminent par un point virgule. Les tables sur lesquelles on opère sont précisées par `FROM`. Une requête renvoie une liste de n-uplets, que l'on peut identifier à une table.

1 Projection et sélection

Une *projection* consiste à donner tous les éléments d'une table en ne gardant que certaines colonnes :

```
SELECT nom, prenom
FROM artistes;
```

nom	prenom
Adams	Amy
Affleck	Ben
Affleck	Casey
⋮	⋮
Winslet	Kate
Woo	John

Il est possible d'utiliser un *alias* pour chaque colonne :

```
SELECT titre AS film, 2022-annee AS âge
FROM films;
```

film	âge
Vertigo	64
Alien	43
Titanic	25
Sacrifice	36
Volte/Face	25
⋮	⋮

Lorsqu'on souhaite obtenir toutes les colonnes, on dispose d'un *joker* `*`.

```
SELECT *
FROM films;
```

id	titre	annee	idRealisateur	genre	resume	codePays
1	Vertigo	1958	3	Drame	Scottie Ferguson, ancien inspecteur de police, ...	USA
2	Alien	1979	4	Science-fiction	Près d"un vaisseau spatial échoué sur ...	USA
3	Titanic	1997	6	Drame	Conduite par Brock Lovett, une expédition ...	USA
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Le mot-clé `DISTINCT` permet de supprimer les doublons :

```sql
SELECT DISTINCT prenom
FROM artistes;
```

prenom
Amy
Ben
Casey
Jean-Hughes
⋮
Sigourney
Bruce
Kate

2 Sélection

La commande `SELECT` admet une option, `WHERE`, qui permet de sélectionner les éléments vérifiant une condition donnée :

```sql
SELECT *
FROM films
WHERE annee < 1970;
```

id	titre	annee	idRealisateur	genre	resume	codePays
1	Vertigo	1958	3	Drame	Scottie Ferguson, ...	USA
35	Psychose	1960	3	Thriller	Après avoir volé ...	USA
37	Les oiseaux	1963	3	Horreur	Melanie Daniels se rend à ...	USA
41	Pas de printemps pour Marnie	1964	3	Thriller	Marnie est engagée...	USA
42	Fenêtre sur cour	1954	3	Suspense	En repos forcé à cause ...	USA
43	La mort aux trousses	1959	3	Suspense	Roger Thornhill, publiciste, ...	USA
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Il est très aisé de réaliser simultanément une sélection et une projection, et il est possible d'utiliser directement les opérateurs booléens :

```sql
SELECT titre
FROM films
WHERE annee >= 1990 AND codePays <> "USA";
```

titre
Van Gogh
MICROCOSMOS
Jeanne d'Arc
Le cinquième élément
Léon
Nikita
⋮

3 Tri et limite du nombre de lignes

Il est possible de trier les enregistrements renvoyés à l'aide d'une clause `ORDER BY`, par ordre croissant avec le mot clé `ASC` ou par ordre décroissant avec le mot clé `DESC`. Par défaut, les éléments seront triés par ordre croissant (`ASC`).

```sql
SELECT nom, prenom
FROM artistes
ORDER BY anneeNaiss DESC;
```

nom	prenom
Stone	Emma
Chazelle	Damien
Johansson	Scarlett
⋮	
Miland	Ray
Grant	Cary
Hitchcock	Alfred

La commande `LIMIT n` permet de ne renvoyer que les n premiers lignes de la requête.

```sql
SELECT titre
FROM films
ORDER BY annee
LIMIT 5
```

titre
Fenêtre sur cour
Vertigo
La mort aux trousses
Psychose
Les oiseaux

Utilisée après la commande `LIMIT n`, la commande `OFFSET p` permet aussi de sauter les p premières lignes de la requête, puis de sélectionner les n premières lignes restantes.

```sql
SELECT titre
FROM films
ORDER BY annee
LIMIT 5
OFFSET 5
```

titre
Pas de printemps pour Marnie
Les dents de la mer
Le gendarme et les extra-terrestres
Alien
Les bronzés font du ski

4 Opérations ensemblistes

4.1 Réunion, intersection, différence

Il est possible de calculer la réunion, l'intersection et la différence de deux tables ayant le même schéma.

Par exemple, considérons deux tables `livres_alice` et `livres_bob` de schéma (titre,auteur).

```sql
SELECT * FROM livres_alice;
```

titre	auteur
Un mari idéal	Oscar Wilde
Le Mariage forcé	Molière
Le Rouge et le Noir	Stendhal
Les Châtiments	Victor Hugo
Zadig	Voltaire

```
SELECT * FROM livres_bob;
```

titre	auteur
La Mare au diable	George Sand
Le Bourgeois gentilhomme	Molière
Le Rouge et le Noir	Stendhal
Zadig	Voltaire

L'ensemble des livres possédés par Alice ou Bob est donné par la requête

```
SELECT * FROM livres_alice
UNION
SELECT * FROM livres_bob;
```

titre	auteur
La Mare au diable	George Sand
Le Bourgeois gentilhomme	Molière
Le Mariage forcé	Molière
Le Rouge et le Noir	Stendhal
Les Châtiments	Victor Hugo
Un mari idéal	Oscar Wilde
Zadig	Voltaire

Les livres communs d'Alice et Bob sont obtenus avec la requête

```
SELECT * FROM livres_alice
INTERSECT
SELECT * FROM livres_bob;
```

titre	auteur
Le Rouge et le Noir	Stendhal
Zadig	Voltaire

Les livres d'Alice que ne possède pas Bob sont donnés par la requête

```
SELECT * FROM livres_alice
EXCEPT
SELECT * FROM livres_bob;
```

titre	auteur
Le Mariage forcé	Molière
Les Châtiments	Victor Hugo
Un mari idéal	Oscar Wilde

4.2 Produit cartésien

Il est possible de calculer le produit cartésien de plusieurs tables de schémas disjoints

Par exemple, considérons les deux tables suivantes :

```
SELECT * FROM fromages;
```

fromage
Cantal
Bleu de Gex

```
SELECT * FROM vins;
```

vin
Chablis
Volnay
Fitou

Le produit cartésien de ces tables s'obtient avec la requête

```
SELECT *
FROM fromages, vins
```

fromage	vin
Cantal	Chablis
Cantal	Volnay
Cantal	Fitou
Bleu de Gex	Chablis
Bleu de Gex	Volnay
Bleu de Gex	Fitou

5 Jointures

Reprenons les tables `films` et `artistes`. Déterminer les films réalisés par Clint Eastwood nécessite d'utiliser des informations provenant des deux tables. Il est possible de d'utiliser un produit cartésien suivi d'une sélection.

```
SELECT titre
FROM artistes, films
WHERE artistes.id = films.idRealisateur
AND artistes.nom = "Eastwood"
AND artistes.prenom = "Clint"
```

Il est possible de reformuler la requête précédente en utilisant l'instruction spécifique `JOIN ... ON`

```
SELECT titre
FROM artistes
JOIN films
ON artistes.id = films.idRealisateur
WHERE artistes.nom = "Eastwood"
AND artistes.prenom = "Clint";
```

titre
Impitoyable
Les pleins pouvoirs

titre
⋮

Il est possible de réaliser une jointure d'une table avec elle-même ; on parle alors d'*auto-jointure*. Cela demande de donner un alias à chacun des exemplaires de la table.
Par exemple, pour obtenir les films dont le réalisateur est celui de *Vertigo* :

```
SELECT f1.titre
FROM films f1
JOIN films f2
ON f1.idRealisateur = f2.idRealisateur
WHERE f2.titre = "Vertigo";
```

titre
Fenêtre sur cour
La mort aux trousses
Les oiseaux
Pas de printemps pour Marnie
Psychose
Vertigo
⋮

6 Fonctions d'agrégation

Les requêtes vues jusqu'ici portaient sur les éléments de la relation ; il est aussi possible d'exprimer des conditions sur des ensembles d'éléments. On dispose pour cela de *fonctions d'agrégation* qui portent sur des sous-ensembles de données (aussi appelés agrégats).

Nom	Description
`COUNT()`	Effectif de chaque sous-ensemble
`MAX()`	Maximum de l'attribut en argument
`MIN()`	Minimum de l'attribut en argument
`SUM()`	Somme de l'attribut en argument
`AVG()`	Moyenne de l'attribut en argument

6.1 Un exemple pour la suite

On dispose d'une base de données comportant des informations sur les communes, départements et régions en France. Cette base de données contient trois tables : `villes`, `departements` et `regions`.

La table `regions` contient trois colonnes :

- `id` : unidentifiant, sous forme d'entier ;
- `nom` : le nom de la région ;
- `cheflieu` : le code INSEE du chef-lieu de la région.

L'attribut `id` est la clé primaire de cette table.

```
SELECT *
FROM regions;
```

id	nom	cheflieu
1	Guadeloupe	97105
2	Martinique	97209
3	Guyane	97302
4	La Réunion	97411

id	nom	cheflieu
⋮	⋮	⋮

La table `departements` contient quatre colonnes :

- `id` : un identifiant, sous forme de chaîne de caractères ;
- `nom` : le nom du département ;
- `region` : l'identifiant de la région à laquelle est rattachée le département ;
- `cheflieu` : le code INSEE du chef-lieu du département.

L'attribut `id` est la clé primaire de cette table.

```
SELECT *
FROM departements;
```

id	nom	region	cheflieu
01	Ain	84	01053
02	Aisne	32	02408
03	Allier	84	03190
04	Alpes-de-Haute-Provence	93	04070
05	Hautes-Alpes	93	05061
⋮	⋮	⋮	⋮

La table `villes` contient six colonnes :

- `id` : un identifiant, correspondant au code INSEE de la commune ;
- `dep` : l'identifiant du département dans lequel se trouve la commune ;
- `nom` : le nom de la commune ;
- `codepostal` : le code postal ;
- `population` : le nombre d'habitants de la commune ;
- `surface` : la superficie du territoire communal en kmš.

L'attribut `id` est la clé primaire de cette table.

```
SELECT *
FROM villes;
```

id	dep	nom	codepostal	population	surface
01001	01	L' Abergement-Clémenciat	01400	767	15.95
01002	01	L' Abergement-de-Varey	01640	239	9.15
01004	01	Ambérieu-en-Bugey	01500	14022	24.6
01005	01	Ambérieux-en-Dombes	01330	1627	15.92
01006	01	Ambléon	01300	109	5.88
01007	01	Ambronay	01500	2570	33.55
⋮	⋮	⋮	⋮	⋮	

6.2 Les fonctions d'agrégation

6.2.1 Premiers exemples

Le nombre de communes en France :

```
SELECT count(*) FROM villes;
```

count(*)
35409

La population communale maximale :

```
SELECT max(population) FROM villes;
```

max(population)
2243833

La population communale minimale :

```
SELECT min(population) FROM villes;
```

min(population)
0

La population communale moyenne :

```
SELECT avg(population) FROM villes;
```

avg(population)
1860.0842723601345

La population totale :

```
SELECT sum(population) FROM villes;
```

sum(population)
65863724

6.2.2 GROUP BY

Pour grouper les éléments suivants certains attributs (et non grouper tous les éléments de la relation), on utilise la clause `GROUP BY` suivie des attributs selon lesquels on souhaite grouper les éléments.

Par exemple, pour obtenir à partir de la relation villes le nombre de communes et la population totale par département (décrit par son numéro) :

```
SELECT dep, count(*), sum(population)
FROM villes
GROUP BY dep;
```

dep	count(*)	sum(population)
01	408	625820
02	804	539734
03	317	342961
04	198	161241
05	167	139125
⋮	⋮	⋮
976	17	212645

Si on veut le nom du département plutôt que son identifiant, on utilise une jointure :

```
SELECT d.nom, count(*), sum(population)
FROM villes JOIN departements as d
ON dep = d.id
GROUP BY d.nom;
```

nom	count(*)	sum(population)
Ain	408	625820
Aisne	804	539734
Allier	317	342961
Alpes-Maritimes	163	1083312
⋮	⋮	⋮
Yvelines	262	1421670

6.2.3 ORDER BY

Il est possible d'utiliser pour le tri le résultat d'une fonction d'agrégation ; la clause `ORDER BY` doit être située en fin de requête (après `GROUP BY`).

```
SELECT r.nom, sum(population)
FROM villes AS v JOIN departements AS d
JOIN regions AS r
ON v.dep = d.id AND d.region = r.id
GROUP BY r.id
ORDER BY sum(population) DESC ;
```

nom	sum(population)
Île-de-France	12046136
Auvergne-Rhône-Alpes	7730962
Hauts-de-France	6003965
Nouvelle-Aquitaine	5856698
⋮	⋮
Guyane	252338
Mayotte	212645

6.2.4 HAVING

On peut souhaiter réaliser une sélection sur un critère dépendant du résultat d'une fonction d'agrégation. Il n'est pas possible d'ajouter ce critère dans la clause `WHERE` , qui ne peut utiliser que des attributs de la relation. On ajoute alors une clause `HAVING` (suivie de la condition) immédiatement après la clause `GROUP BY`.

```
SELECT d.id, d.nom, sum(population)
FROM villes JOIN departements AS d
ON villes.dep = d.id
GROUP BY d.id
HAVING sum(population) > 1500000
ORDER BY d.id ;
```

id	nom	sum(population)
13	Bouches-du-Rhône	2006069
33	Gironde	1524347
59	Nord	2603472
69	Rhône	1796624
75	Paris	2243833
92	Hauts-de-Seine	1597770

id	nom	sum(population)
93	Seine-Saint-Denis	1571028