Requêtes SQL

1 Sélection des données

1.1 SQL SELECT

1.1.1 Commande basique

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande `SELECT`, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.
L'utilisation basique de cette commande s'effectue de la manière suivante :

```
SELECT nom_du_champ FROM nom_du_tableau
```

Cette requête SQL va sélectionner (`SELECT`) le champ «nom_du_champ» provenant (`FROM`) du tableau appelé «nom_du_tableau».

Exemple : Si l'on veut connaître tous les noms des éditeurs présent dans notre base de données, il suffit d'effectuer la requête SQL ci-dessous.

```
SELECT nom FROM editeur;
```

De cette manière on obtient le résultat :

	nom
1	iMinds Pty Ltd
2	Educa Books
3	Editions Albert René
4	J'ai Lu
5	LGF/Le Livre de Poche

1.1.2 Obtenir plusieurs colonnes

Il est possible de lire plusieurs colonnes à la fois. Il suffit tout simplement de séparer les noms des champs souhaités par une virgule. Pour obtenir les prénoms et les noms des élèves il faut alors faire la requête suivante :

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

	nom	prenom
1	Rhodes	Lucian
2	Ward	Ralph
3	Mayo	Giacomo
4	Hernandez	Zeus
5	Alston	Levi

Il est possible de retourner automatiquement toutes les colonnes d'un tableau sans avoir à connaître le nom de toutes les colonnes. Au lieu de lister toutes les colonnes, il faut simplement utiliser le caractère «*» (étoile). C'est un joker qui permet de sélectionner toutes les colonnes. Il s'utilise de la manière suivante :

```
SELECT * FROM eleve;
```

1.2 SQL DISTINCT

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT après le mot SELECT.
L'utilisation basique de cette commande consiste alors à effectuer la requête suivante :

```
SELECT DISTINCT ma_colonne
FROM nom_du_tableau
```

Exemple : Si l'on veut connaître toutes les années de parution des livres présents au CDI sans afficher de doublon, il suffit d'effectuer la requête SQL ci-dessous.

```
SELECT DISTINCT annee FROM livre;
```

1.3 SQL WHERE

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

1.3.1 Syntaxe

La commande WHERE s'utilise en complément à une requête utilisant SELECT. La façon la plus simple de l'utiliser est la suivante :

```
SELECT nom_colonnes FROM nom_table WHERE condition
```

Exemple : Si l'on veut connaître les noms et prénoms des élèves de la classe 2-GT5, il suffit d'effectuer la requête SQL ci-dessous.

```
SELECT nom, prenom FROM eleve WHERE classe = '2-GT5';
```

1.3.2 Opérateurs de comparaisons

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques uns des opérateurs les plus couramment utilisés.

Opérateur	Description
=	Egal
<> ou !=	Différent de
> / >=	Supérieur à / Supérieur ou égale à
< / <=	Inférieur à / Inférieur ou égale à
IN	Liste de plusieurs valeurs possible
BETWEEN	Valeur comprise dans un intervalle donnée
LIKE	Recherche spécifiant le début, milieu ou fin d'un mot

Exemple : Pour connaître les livres dont le titre contient le mot «homme», il suffit d'effectuer la requête SQL ci-dessous.

```
SELECT * FROM livre WHERE titre LIKE '%homme%';
```

Le caractère «%» est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère qui contiennent le mot «homme».

1.4 SQL ORDER BY

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.
Une requête où l'on souhaite filtrer l'ordre des résultats utilise la commande ORDER BY de la sorte :

```
SELECT colonne1, colonne2
FROM table
ORDER BY colonne1
```

Exemple : Pour afficher les titre des livres classés par année d'édition, il suffit d'effectuer la requête SQL ci-dessous.

```
SELECT titre, annee FROM livre ORDER BY annee;
```

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne.

```
SELECT titre, annee FROM livre ORDER BY annee DESC;
```

1.5 Fonctions d'agrégation SQL

Les fonctions d'agrégation dans le langage SQL permettent d'effectuer des opérations statistiques sur un ensemble d'enregistrement. Étant données que ces fonctions s'appliquent à plusieurs lignes en même temps, elle permettent des opérations qui servent à récupérer l'enregistrement le plus petit, le plus grand ou bien encore de déterminer la valeur moyenne sur plusieurs enregistrement.

1.5.1 Liste des fonctions d'agrégation

Les fonctions d'agrégation sont des fonctions idéales pour effectuer quelques statistiques de bases sur des tables. Les principales fonctions sont les suivantes :

- AVG() pour calculer la moyenne sur un ensemble d'enregistrement
- COUNT() pour compter le nombre d'enregistrement sur une table ou une colonne distincte
- MAX() pour récupérer la valeur maximum d'une colonne sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumérique
- MIN() pour récupérer la valeur minimum de la même manière que MAX()
- SUM() pour calculer la somme sur un ensemble d'enregistrement

1.5.2 Utilisation simple

L'utilisation la plus générale consiste à utiliser la syntaxe suivante :

```
SELECT fonction(colonne) FROM table
```

La fonction COUNT() possède une subtilité. Pour compter le nombre total de ligne d'une table, il convient d'utiliser l'étoile "*" qui signifie que l'ont cherche à compter le nombre d'enregistrement sur toutes les colonnes. La syntaxe serait alors la suivante :

```
SELECT COUNT(*) FROM table
```

Exemple : Compter le nombre de livres de la base de données :

```
SELECT COUNT(*) FROM livre;
```

L'année de publication la plus récente :

```
SELECT MAX(annee) FROM livre;
```

1.6 Jointure SQL

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.
Pour deux tables A et B avec une clef commune, la syntaxe pour obtenir leur jointure est :

```
SELECT *
FROM A
JOIN B ON A.key = B.key
```

Exemple : Pour afficher les titres des livres et les noms des éditeurs, il suffit d'effectuer la requête SQL ci-dessous.

```
SELECT livre.titre, editeur.nom
FROM livre JOIN editeur ON livre.siret = editeur.siret;
```

On peut également associer plus de 2 tables, si l'on veut par exemple obtenir la liste des livres empruntés avec le nom et prénom de l'élève correspondant et la date retour, il suffit d'effectuer la requête SQL ci-dessous.

```
SELECT livre.titre, eleve.nom, eleve.prenoms, emprunt.date_ret
FROM livre
JOIN emprunt ON livre.isbn = emprunt.isbn
JOIN eleve ON eleve.num_etu = emprunt.num_etu;
```

La commande USING
Dans les exemples précédents les colonnes, les colonnes qui permettent de faire les jointures ont les mêmes noms. Dans ce cas on peut utiliser la commande `USING` de la manière suivante.

```
SELECT livre.titre, editeur.nom
FROM livre JOIN editeur USING(siret);
```

```
SELECT livre.titre, eleve.nom, eleve.prenoms, emprunt.date_ret
FROM livre
JOIN emprunt USING(isbn)
JOIN eleve USING(num_etu);
```

2 Modification des données

2.1 Suppressions de lignes

La commande `DELETE` en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associé à `WHERE` il est possible de sélectionner les lignes concernées qui seront supprimées.
La syntaxe pour supprimer des lignes est la suivante :

```
DELETE FROM nom_de_table WHERE condition
```

Exemple : Si un elève rend le livre *1984*, dont l'isbn est 978-0547249643, il faut supprimer la ligne correspondant dans la table `emprunt`.

```
DELETE FROM emprunt WHERE isbn = '978-0547249643';
```

Si l'élève *Head Bryar*, dont le numéro d'étudiant est 160307224881 rend tous ces livres, il faut supprimer les lignes correspondantes.

```
DELETE FROM emprunt WHERE num_etu = 160307224881;
```

2.2 Mise à jour

La commande `UPDATE` permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec `WHERE` pour spécifier sur quelles lignes doivent porter la ou les modifications. La syntaxe basique d'une requête utilisant `UPDATE` est la suivante :

```
UPDATE table
SET nom_colonne_1 = 'nouvelle valeur'
WHERE condition
```

Exemple : Si l'on souhaite par exemple prolonger de 30 jours l'emprunt du livre *1984*, dont l'isbn est 978-0547249643, il suffit d'effectuer la requête SQL ci-dessous.

```
UPDATE emprunt SET date_ret = date_ret + 30
WHERE isbn = '978-0547249643';
```

Références

[1] Thibaut Balabonsky, Sylvain Conchon, Jean-Christophe Filliâtre, Kim Nguyen, *Spécialité Numérique et sciences informatiques : 24 leçons avec exercices corrigés - Terminale*. `http://www.nsi-terminale.fr/`.

[2] *Cours et tutoriels sur le langage SQL*. `https://sql.sh/`.