Séquence A

Principaux ordres en langage SQL

Ceci est un memento des principaux ordres exprimés en langage SQL à savoir utiliser.

Le langage permet de créer des relations, on dit des **tables**, en spécifiant leur nom, leurs attributs, les domaines de valeurs, on dit les **types**, et les contraintes associées à la table.

Les ordres peuvent s'étendre sur plusieurs lignes, les espaces et les indentations ne sont pas significatifs mais ils améliorent la lisibilité. Un ordre se termine par la ponctuation `;`.

A.1 Définition de données

La syntaxe générale de création d'une table est la suivante.

```
CREATE TABLE <nom_table> (<attribut 1> <domaine 1> <contrainte 1>,
  <attribut 2> <domaine 2> <contrainte 2>,
  ...,
  <contrainte globale 1>,
  ...,
  <contrainte globale n>);
```

L'expression de la contrainte portant sur un attribut est optionnelle de même que les contraintes globales qui sont exprimées après la définition de tous les attributs.

A.1.1 Quelques types de données possibles

Pour chaque domaine de valeurs, on présente les principaux types offerts par le langage.

Domaine numérique

Nom du type	Description
`SMALLINT`	**exact** entier signé sur 16 bits
`INTEGER`	**exact** entier signé sur 32 bits
`BIGINT`	**exact** entier signé sur 64 bits
`INT`	**exact** il s'agit d'un alias pour `INTEGER`
`DECIMAL(t, f)`	**exact** nombre décimal signé de `t` chiffres dont `f` après la virgule
`REAL`	**approché** nombre flottant sur 32 bits
`DOUBLE PRECISION`	**approché** nombre flottant sur 64 bits.

Les chaînes de caractères sont délimitées par des guillemets simples `'`. L'apostrophe peut être échappée en la doublant.

Domaine chaînes de caractères

Nom du type	Description
`CHAR(n)`	Chaîne ayant exactement n caractères. Ceux qui manquent sont complétés par des espaces à droite du dernier caractère.
`VARCHAR(n)`	Chaîne ayant au plus n caractères.
`TEXT`	Chaîne de taille quelconque.

Les types de date, d'instant ou de durée permettent d'effectuer des opérations comme ajouter des jours par exemple. À savoir en cas de besoin mais en recherchant la syntaxe adaptée à l'implémentation du langage.

Domaine temporel

Type	Description
`DATE`	une date au format `'AAAA-MM-JJ'`
`TIME`	une heure au format `'hh:mm:ss'`
`TIMESTAMP`	un instant, date et heure, au format `'AAAA-MM-JJ hh:mm:ss'`

La valeur `NULL` est particulière et on peut l'assimiler à `None` en Python. Elle représente une absence de valeur. Nous l'utiliserons essentiellement pour tester si la valeur d'un attribut est présente ou pas avec les expressions `IS NULL` ou `IS NOT NULL`.

Le type booléen est optionnel dans le standard et donc inégalement supporté. Une alternative réside en un type `CHAR(1)` avec les valeurs `'T'` et `'F'` par exemple.

A.1.2 Expression des contraintes d'intégrité

Contraintes d'intégrité

Mot clef	Description
`PRIMARY KEY`	indique qu'un attribut ou un ensemble d'attributs alors décrit entre parenthèses est une clef primaire.
`REFERENCES`	qualifie un attribut comme clef étrangère, suivi du nom de la table de référence et, entre parenthèses, du nom de l'attribut dans cette table.
`UNIQUE`	permet de spécifier qu'un attribut ou un groupe d'attributs est unique sans en faire une clef primaire.
`NOT NULL`	signifie qu'un attribut ne peut pas être `NULL`.
`CHECK`	suivi d'une formule booléenne permet de spécifier des contraintes arbitraires sur les attributs d'une entité. Une contrainte `CHECK` figure **obligatoirement** à la fin de la déclaration de la table.

A.1.3 Insertion de valeurs

L'insertion de nouvelles valeurs dans une table s'effectue au moyen de l'ordre `INSERT INTO` avec la syntaxe suivante

```
INSERT INTO <nom_table> VALUES (valeurs 1),
  (valeurs 2),
  ...,
  (valeurs n);
```

Chaque n-uplet écrit entre parenthèses représente une nouvelle ligne dans la table et doit donc respecter le schéma de la relation.

Les contraintes d'intégrité sont vérifiées par le SGBD au moment de l'insertion. Une instruction `INSERT` qui violerait ces contraintes conduirait à un message d'erreur et les données correspondantes ne seraient pas ajoutées à la table.

A.1.4 Suppression d'une table

Enfin, il est possible de supprimer une table avec la syntaxe

```
DROP TABLE <nom_table>;
```

Cet ordre renverra un message d'erreur si cette table sert de référence comme clef étrangère d'une autre table car il y aurait violation d'une contrainte de référence. Attention donc à supprimer les tables dans le bon ordre en remontant celui des contraintes de référence.

A.2 Sélection de données

La sélection de données consiste en l'écriture de **requêtes SQL** permettant de trouver toutes les données de la base qui vérifient certaines conditions.

La **clause** `SELECT` permet de réaliser des **sélections**, de certains enregistrements, et des **projections**, de certains champs, avec la syntaxe générale suivante

```
SELECT <nom_attribut_1>, <nom_attribut_2>,
  ..., <nom_attribut_n>
  FROM <nom_table>
  WHERE <conditions>
;
```

Le résultat d'une telle requête est une table mais celle-ci n'est pas persistante.

L'**analyse syntaxique** de cette requête commence par sa partie `FROM` qui va désigner la table, les attributs sont nécessairement issus de cette table et les conditions portent sur des attributs issus de cette table.

Le mot clef `DISTINCT` écrit directement après la clause `SELECT` permet d'éviter le renvoi de doublons lorsqu'il en existe dans la sélection.

L'écriture du *joker* $*$ à la place de la liste des noms d'attributs réduit l'ordre à une **sélection** d'enregistrements.

Dans une **projection**, chaque attribut peut être renommé au moyen du mot clef `AS` avec une syntaxe comme

```
SELECT <nom_attribut_1> AS <alias_1>, <nom_attribut_2>
  FROM <nom_table>
  WHERE <conditions>
;
```

Il est possible de trier les résultats avec la clause `ORDER BY` insérée avant la ponctuation `;` en précisant l'attribut sur lequel porte le tri suivi de l'ordre de rangement avec `ASC` pour croissant, valeur par défaut, ou `DESC` pour décroissant.

Un tri sur plusieurs attributs est aussi possible en les enchaînant après la ponctuation `,`.

Enfin, la clause `LIMIT` suivie d'un nombre entier réduit le nombre d'enregistrements affichés.

```
SELECT *
  FROM <nom_table>
  ORDER BY <nom_attribut_1> ASC, <nom_attribut_2> DESC
  LIMIT 20
;
```

La clause `WHERE` est optionnelle. L'expression des conditions qui figurent après cette clause est booléenne. Elle peut être construite à partir d'opérateurs de comparaison, d'opérateurs arithmétiques, de constantes, de noms d'attributs et d'opérateurs logiques.

L'opérateur spécial `LIKE` et notamment le *joker* `%`, qui signifie « toute chaîne de caractères », permettent de comparer des chaînes de caractères. Il s'agit d'une recherche de *pattern* :

- `'K%'` renvoie tous les enregistrements commençant par `'K'` ;
- `'%K%'` renvoie tous les enregistrements contenant `'K'` ;
- `'K[L-Q]%'` renvoie tous les enregistrements commençant par `'K'` suivi d'une lettre comprise entre `'L'` et `'Q'`, suivi d'au moins un caractère ;
- `'Kl_mn'` renvoie tous les enregistrements de 5 caractères commençant par `'Kl'` et terminant par `'mn'`.

On peut employer cet opérateur spécial après la clause `WHERE` avec cette syntaxe

```
SELECT *
  FROM <nom_table>
  WHERE <attribut_str> LIKE <motif>
;
```

Enfin, l'opérateur `IN` permet de rechercher tous les enregistrements dont un champ possède une valeur appartenant à une liste donnée ou le contraire s'il est précédé de `NOT`.

```
SELECT *
  FROM <nom_table>
  WHERE <attribut_1> IN (<valeur1>, <valeurs2>)
;
```

Il existe aussi l'opérateur `BETWEEN` pour rechercher dans un intervalle défini par ses deux bornes `<valeur1> AND <valeur2>`.

A.2.1 Agrégation de données

Les fonctions d'agrégation permettent d'appliquer une fonction à l'ensemble des valeurs d'un champ et de renvoyer le résultat comme une table formée d'un seul enregistrement et d'un seul champ.

Elles s'emploient après la clause `SELECT`.

Exemples de fonctions mobilisables

- `COUNT` permet d'obtenir le nombre de résultats renvoyés
- `AVG`, pour *average*, permet de calculer la moyenne des valeurs d'un champ
- `SUM` permet de calculer la somme des valeurs d'un champ
- `MIN` et `MAX` permettent de trouver le minimum et le maximum parmi les valeurs d'un champ.

Un exemple de syntaxe

```
SELECT SUM(<attribut_1>) AS total
  FROM <nom_table>
  WHERE <conditions>
;
```

où les valeurs du champ `<attribut_1>` sont d'un type numérique compatible avec la fonction `SUM`.

A.2.2 Jointure de tables

Étant données deux tables `<tableA>` et `<tableB>`, l'opération de jointure consiste à créer toutes les combinaisons d'enregistrements de `<tableA>` et de `<tableB>` ayant un champ de même valeur. Cet attribut est généralement une clé primaire ou une clef étrangère.

La clause `JOIN` permet de réaliser cette opération de **jointure** selon la syntaxe générale que voici

```
SELECT <attribut_1>, <attribut_2>
  FROM <tableA> JOIN <tableB>
  ON <tableA>.<cléA> = <tableB>.<cléB>
  WHERE <conditions>
;
```

Cette requête renvoie les enregistrements communs aux deux tables selon le critère posé.

Les attributs peuvent être issus d'une table ou de l'autre. Afin d'éviter toute confusion et de rendre la syntaxe de la requête plus explicite, l'usage recommande de préfixer les noms des attributs par celui de la table dont ils sont issus. Pour soulager la saisie et la lecture, il convient d'attribuer un alias à la table avec le mot clef `AS`.

La syntaxe générale devient alors

```
SELECT <tA>.<attribut_1>, <tB>.<attribut_2>
  FROM <tableA> AS tA
  JOIN <tableB> AS tB
    ON <tA>.<cléA> = <tB>.<cléB>
  WHERE <conditions>
;
```

et il est possible d'ajouter une deuxième clause `JOIN` après la première pour réaliser une jointure avec une troisième table. On peut ainsi composer des requêtes plus complexes par jointure entre des tables liées par une relation.

Il faut savoir qu'il existe d'autres types de jointures mais celles-ci ne sont pas au programme de la spécialité.

A.2.3 Mise à jour de données

L'**ajout de données** dans une table a déjà été présenté. Nous allons compléter cette introduction et introduire la mise à jour de données ainsi que leur suppression.

On peut ajouter des données dans une table à partir du résultat d'une requête de sélection.

```
INSERT INTO <tableB>
  (<attribut_1>, <attribut_2>, ...)
  SELECT <attribut_1>, <attribut_2>, ...
  FROM <tableA>
  WHERE <conditions>
;
```

Dans cette requête, la première ligne d'attributs fait référence à ceux de la table `<tableB>` tandis que ceux exprimés après la cluase `SELECT` réfèrent à la table `<tableA>`.

Imaginons par exemple que l'on souhaite recopier une table pour l'abriter avant une manipulation. Cette syntaxe avec le *joker* * et sans la clause `WHERE` pourrait faire l'affaire mais attention car elle ne fait que copier les données sans transmettre les contraintes d'intégrité. Il faut donc préalablement créer une table de secours qui hérite de ces contraintes avant de réaliser la copie des enregistrements.

Les deux ordres SQL qui permettent la copie intégrale d'une table suivent la syntaxe suivante

```
CREATE TABLE <tableSecours>
  (LIKE <tableInitiale> INCLUDING ALL)
;
INSERT INTO <tableSecours>
  (SELECT * FROM <tableInitiale>)
;
```

La **suppression d'enregistrements** a été entrevue en exercice de création de tables. La syntaxe générale

```
DELETE FROM <nom_table>
  WHERE <conditions>
;
```

permet de supprimer tous les enregistrements de la table `<nom_table>` qui vérifient les conditions énoncées après la clause `WHERE` si aucune contrainte d'intégrité n'est violée par cet ordre sinon la table reste en l'état.

Attention, sans clause `WHERE` tous les enregistrements de cette table sont effacés et il reste une table vide.

La **mise à jour** consiste à remplacer les valeurs de certains attributs par d'autres valeurs. La syntaxe générale est la suivante :

```
UPDATE <nom_table>
  SET <attribut_1> = <valeur_1>,
  SET <attribut_2> = <valeur_2>,
  ...,
  SET <attribut_n> = <valeur_n>
  WHERE <conditions>
;
```

Les expressions <valeur_i> de mise à jour peuvent mentionner des noms d'attributs et il y a alors substitution de la valeur courante de ces attributs comme pour l'évaluation d'une expression en algorithmique. En combinant des noms d'attributs avec des opérateurs compatibles avec leurs types, des mises à jour un peu complexes sont alors réalisables.

A.2.4 Requêtes imbriquées

La clause `SELECT` renvoie un résultat sous la forme d'une table. Dès lors, on peut envisager de réaliser une nouvelle requête portant sur les attributs de ce résultat considéré comme un intermédiaire.

La requête renvoyant le résultat intermédiaire est désignée comme **sous-requête** et elle est exprimée entre parenthèses. La requête exploitant le résultat est la **requête principale**.

L'exécution s'effectue depuis la sous-requête la plus profonde dans l'imbrication vers la requête principale, chacune se fondant sur les résultats renvoyés par la précédente.

- Placée après la clause `SELECT`, une sous-requête peut renvoyer un attribut calculé.
- Placée après `FROM` une sous-requête permet de pré-sélectionner les enregistrements sur lesquels opérera la requête principale.
- Placée après la clause `WHERE` une sous-requête permet d'établir une ou des valeurs possibles sur lesquelles opérera la condition.

  Dans ce cas, la condition est exprimée avec `IS` si une seule valeur est renvoyée par la sous-requête ou sinon avec `IN`.

On rencontrera ainsi le plus souvent l'un de ces trois squelettes d'ordre SQL :

```
SELECT ...,
  (SELECT ... FROM ... WHERE ...) AS ...
  FROM ...
  WHERE ...
;
```

pour le premier cas

```
SELECT ... FROM
  (SELECT ... FROM ... WHERE ...) AS ...
  WHERE ...
;
```

pour le second et

```
SELECT ...
  FROM ...
  WHERE ... IN[IS]
  (SELECT ... FROM ... WHERE ...)
;
```

pour le troisième.

A.3 Pour finir

Le programme s'arrête là. Si vous rencontrez des besoins pour exprimer des opérations ou si vous avez la curiosité de découvrir d'autres possibilités offertes par le langage SQL, vous pouvez consulter ce cours en ligne https://sql.sh/ qui est en français.

Un gestionnaire de bases de données accessible pour débuter est DB Browser for SQLite que vous pouvez télécharger depuis la page WEB suivante https://sqlitebrowser.org/ qui détaille les installations selon le système d'exploitation.

Il est aussi possible d'utiliser cette solution en ligne https://sqliteonline.com/.