Le langage SQL

Introduction :

Nous savons que, pour créer une base de données, il est d'abord nécessaire de la modéliser à l'aide d'un MCD, puis d'en déduire le modèle relationnel et, enfin, de procéder à une normalisation du schéma ainsi obtenu. Une fois ces étapes effectuées, il est enfin temps de concrétiser la base de données et de l'utiliser.

Dans ce cours, nous allons découvrir un langage utilisé pour transmettre des ordres à un SGBD relationnel : SQL. Il se décompose selon trois grandes familles d'opérations : la gestion des tables et des vues, la manipulation des données en table et, pour terminer, l'interrogation et la recherche dans les tables.
Commençons par créer les relations de notre modèle relationnel, autrement dit, la structure de la base. Les tables qui en résulteront seront vides, mais prêtes à accueillir les futures données.

1 La description des données

Le langage utilisé pour créer la structure d'une base de données est un LDD (Langage de description des données). SQL, en plus d'autres langages, est un LDD.

Astuce

SQL est le sigle de *Structured query language* qui signifie en français « Langage de requêtes structuré ».

Examinons les requêtes que nous pouvons adresser au SGBD pour créer la structure de notre base de données.

a. Création de la structure de la base de données

La structure de la base de données comporte l'ensemble de ses tables. Il est donc nécessaire de les créer une à une.
La création d'une table est une opération qu'il faut effectuer avec attention. C'est lors de cette étape que, en plus de ses colonnes, on définit les contraintes d'intégrité de domaine, de relation et de référence.

La syntaxe employée est la suivante :

```
CREATE TABLE nomTable (
        colonne1 type de donnée,
        colonne2 type de donnée,
        colonne3 type de donnée,
        colonne4 type de donnée,
        ....
        PRIMARY KEY (colonnex, colonney, ...),
        FOREIGN KEY (colonnea, colonneb,...) REFERENCES nomTableX (colonnei, colonnej...)
        ON DELETE option
        ...);
```

À retenir

- Chaque instruction se termine obligatoirement par un « ; ».
- L'orthographe des termes écrits en lettre capitales uniquement doit être respectée. Ce sont des mots-clés composant la syntaxe.
- Les termes soulignés sont obligatoires.
- Les autres termes sont les noms qui sont affectés librement aux attributs et aux tables.

Attention

Cette syntaxe n'est pas exhaustive, mais nous l'avons volontairement limitée aux notions que nous avons jugées les plus utiles pour ce cours.

Type de donnée

Le type de donnée constitue en lui-même une contrainte de domaine. Il précise la nature d'un attribut :

- **Numérique**
  - INT (codé sur 32 bits) ;
  - SMALLINT (codé sur 16 bits) ;
  - REAL (taille qui dépend du SGBD) ;
  - FLOAT(n) (réel représenté sur n bit).
- **Caractère**
  - CHAR(n) (chaîne de longueur n, codage ASCII 1 octet) ;
  - VARCHAR(n) (chaîne de longueur maximale n, codage ASCII 1 octet) ;
  - NCHAR(n) (chaîne de longueur n, codage UNICODE 2 octets) ;
  - NVARCHAR(n) (chaîne de longueur maximale n, codage UNICODE 2 octets).
- **Binaire**
  - BOOLEAN (booléen) ;
  - BLOB (*Binary large object* qui permet de stocker tout type binaire tel que photo, document mis en forme...).
- **Date**
  - DATE (date) ;
  - TIME(n) (heure, n, optionnel, étant le nombre de décimales pour représenter une fraction de seconde).

Clé primaire

Le terme « PRIMARY KEY » désigne le ou les attributs qui constituent la clé primaire de la relation. C'est une contrainte de relation qui garantit l'unicité de la clé primaire.

Clé étrangère

- Le terme « FOREIGN KEY » désigne le ou les attributs qui constituent une clé étrangère dans la relation. C'est une contrainte de référence qui garantit une cohérence entre les données.
- Le terme « REFERENCE » sert à désigner la table et la ou les colonnes de cette table dont la clé étrangère est issue.
- Enfin, le terme « ON DELETE » permet de préciser l'option choisie en cas de suppression de la clé étrangère dans la table d'où celle-ci provient.

Exemple

Si nous reprenons le modèle relationnel de la base de données de notre club de cuisine (voir cours précédents du même chapitre), voici sa traduction en une commande SQL qui permet de créer les tables *ville* et *adherent* de notre base.

Schéma relationnel	Commande SQL
Ville(idVille, nomVille, codePostal) avec idVille : clé primaire	CREATE TABLE ville(idVille INT NOT NULL AUTO_INCREMENT, nomVille CHAR(50), codePostal INT, PRIMARY KEY (idVille));
Adherent(idAdherent, nomAdherent, prenom, mail, numero, rue, complement, idVille, passe, telephone) avec idAdherent : clé primaire idVille : clé étrangère en référence à idVille de Ville	CREATE TABLE adherent(idAdherent INT NOT NULL AUTO_INCREMENT, nomAdherent CHAR(50), prenom CHAR(50), mail CHAR(30),numero CHAR(10), rue CHAR(50), complement CHAR(50), idVille INT,

```
    passe CHAR(30),
    telephone INT,

    PRIMARY KEY (idAdherent),
    FOREIGN KEY (idVille)
    REFERENCES ville(idVille) ON
    DELETE NO ACTION
    ;
```

On remarquera la consigne « AUTO_INCREMENT » qui laisse l'initiative au SGBD d'affecter une valeur numérique à la colonne *idAdherent*, s'incrémentant de 1 en 1 au fur et à mesure que l'on ajoute des lignes dans la table *adherent* (idem pour *idVille*).
On notera également la précision « NOT NULL » qui évite qu'une ligne soit insérée sans clé primaire (dans le cas contraire, il serait impossible de l'identifier !).
Enfin, on constate que l'on choisit de ne supprimer aucun enregistrement dans la table *ville* si sa colonne *idVille* figure comme clé étrangère dans une ligne de la table *adherent*.

Nous avons ainsi créé les deux tables vides suivantes :

idVille	nomVille	codePostal

idAdherent	nomAdherent	prenom	mail	numero	rue	complement

b. Suppression d'une table

La syntaxe pour supprimer une table, et par conséquent son contenu, est la suivante : `DROP TABLE nomTable;`.

Il va de soi que si des contraintes d'intégrité référentielle ont été posées, dans les autres tables, sur des clés étrangères faisant appel à des colonnes issues de la table que l'on souhaite supprimer, l'opération ne sera pas possible.
Il est donc nécessaire de supprimer au préalable les tables ou les lignes dont les clés étrangères se réfèrent à une ligne de la table que l'on veut supprimer.

Exemple

DROP TABLE ingredient;

Erreur

Requête SQL :

```
DROP TABLE ingredient
```

MySQL a répondu:

```
#1217 - Cannot delete or update a parent row: a foreign key
constraint fails
```

DROP TABLE utilise;

MySQL a retourné un résultat vide (aucune ligne). (Traitement en 0.0080 secondes.)

```
DROP TABLE utilise
```

DROP TABLE ingredient;

MySQL a retourné un résultat vide (aucune ligne). (Traitement en 0.0088 secondes.)

```
DROP TABLE ingredient
```

Dans un premier temps, il n'a pas été possible de supprimer la table *ingredient* car une ou plusieurs de ses lignes était référencée via une clé étrangère (*idIngredient*) dans une autre table (la table *utilise*).
La suppression de la table *utilise*, qui, elle, n'est référencée dans aucune autre table, a été possible.
Cela a alors permis, par la suite, la suppression de la table *ingredient*.

c. Modification d'une table

La commande `ALTER TABLE` permet de modifier la structure d'une table, que ce soit en ajoutant, supprimant ou modifiant ses colonnes.
Cette opération est plus délicate lorsque la table contient déjà des lignes. Il est par conséquent important d'agir prudemment avec cette commande.

Ajout d'une colonne à une table

```
ALTER TABLE nomTable
ADD COLUMN nouveauChamp TYPEDONNEE;
```

Notons que l'attribut *nouveauChamp* sera affecté avec une valeur « NULL » aux enregistrements déjà existants dans la table.

Exemple

```
ALTER TABLE recette
ADD COLUMN pays CHAR(30);
```

Avant ajout de la colonne *pays* :

idRecette	nomRecette	niveau	duree	categorie	idAdh
1	Mayonnaise	difficile	15	sauce	7
2	Omelette	facile	5	plat	6
3	Pâte brisée	moyen	20	autre	6
4	Quatre-quarts	facile	10	dessert	4
5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2
7	Sole meunière	difficile	20	plat	7

Après ajout de la colonne *pays* :

idRecette	nomRecette	niveau	duree	categorie	idAdh
1	Mayonnaise	difficile	15	sauce	7
2	Omelette	facile	5	plat	6
3	Pâte brisée	moyen	20	autre	6
4	Quatre-quarts	facile	10	dessert	4
5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2

7	Sole meunière	difficile	20	plat	7

Suppression de la colonne d'une table

```
ALTER TABLE nomTable
DROP COLUMN nomChamp;
```

Exemple

```
ALTER TABLE recette
DROP COLUMN duree;
```

Après suppression de la colonne *duree* :

idRecette	nomRecette	niveau	categorie	idAdherent	pays
1	Mayonnaise	difficile	sauce	7	
2	Omelette	facile	sauce	6	
3	Pâte brisée	moyen	sauce	6	
4	Quatre-quarts	facile	sauce	4	
5	Purée de carottes	moyen	sauce	2	
6	Poulet rôti	facile	sauce	2	
7	Sole meunière	difficile	sauce	7	

Modification d'une colonne

Cette opération peut s'avérer délicate s'il s'agit, par exemple, de renommer le nom d'une colonne ou de changer son type. L'appel à une colonne supplémentaire s'avère alors nécessaire.

Exemple

> Supposons que l'on souhaite renommer la colonne *rue* de la table *adherent* pour l'appeler *voie* et que l'on souhaite augmenter sa taille à 100 caractères maximum.
> Il faudra alors procéder comme suit :
>
> ```
> /* création d'une colonne portant le nom désiré et du type désiré */
> ALTER TABLE adherent
> ADD COLUMN voie VARCHAR(100);
>
> /* initialisation de la nouvelle colonne avec les valeurs de l'ancienne colonne */
> UPDATE adherent
> SET voie=rue;
>
> /* suppression de l'ancienne colonne */
> ALTER TABLE adherent
> DROP COLUMN rue;
> ```

L'instruction `UPDATE` qui figure dans cet exemple est étudiée plus en détail dans la suite de ce cours. Elle permet ici, pour chaque ligne existante dans la table *adherent*, de renseigner la valeur du nouveau champ *voie* avec la valeur du champ existant *rue*.

Maintenant que nous sommes en mesure de créer la structure de base, nous allons pouvoir commencer à y enregistrer des données. Un Langage de manipulation des données (LMD) nous est nécessaire.

→ SQL en est un.

2 | La manipulation des données

La manipulation des données consiste en leur insertion, modification et suppression dans les tables de la base. Regardons en détail les instructions

qui nous permettent de faire réaliser ces opérations par le SGBD.

a. L'insertion d'une ligne dans une table

Il s'agit d'enregistrer une nouvelle ligne dans une table. Les mécanismes suivants vont être sollicités :

- contrôle des contraintes d'intégrité lors de l'insertion (domaine, relation et référentielle) ;
- renseignement des champs avec la valeur « NULL » si aucune valeur n'est précisée et si « NULL » est autorisé pour ces champs.

La syntaxe est la suivante :

```
INSERT INTO nomTable
(champx, champY, ....)
VALUES (valeurChampX, valeurChampY,...);
```

À retenir

Quelques précisions méritent notre attention :

- Les valeurs des colonnes doivent être renseignées dans le même ordre que celui de la liste des colonnes donnée juste avant.
- Si la liste des colonnes n'est pas donnée, alors le SGBD se base sur la liste des colonnes de la table, dans l'ordre où celles-ci ont été définies lors de sa création.
- Si certaines colonnes sont omises, elles prendront la valeur 'NULL'.

Exemple

Insérons un ingrédient dans la base de données du club de cuisine :

```
INSERT INTO ingredient
(nomIngredient, type)
VALUES ("oeuf","viande")
```

Nous n'avons pas précisé la valeur de la colonne *idIngredient* car le SGBD se charge lui-même de lui affecter une valeur numérique unique (option "AUTO_INCREMENT" spécifiée lors de la création de la table). Sachant qu'ici, c'est le premier ingrédient que nous insérons dans la table ingredient, la colonne idIngredient prendra la valeur 1.

idIngredient	nomIngredient	type
1	oeuf	viande

b La suppression de la ligne d'une table

Astuce

> Comme lors de la suppression d'une table, la suppression d'une ligne est soumise au respect des contraintes d'intégrité.

L'instruction de suppression permet de supprimer une ou plusieurs lignes d'une table. Pour cela, les lignes concernées par l'opération doivent être désignées. Celles-ci sont identifiées avec une condition.

À retenir

> C'est la clause `WHERE` qui va nous le permettre.

Ainsi, la syntaxe de la commande SQL pour supprimer une ou plusieurs lignes d'une table est la suivante :

```
DELETE FROM nomTable
WHERE condition;
```

La condition consiste en un ou plusieurs critères portant sur le contenu d'une ou plusieurs colonnes. Elle s'exprime à l'aide d'opérateurs de comparaison, comparaison que l'on peut combiner avec d'autres en utilisant des connecteurs logiques.

Opérateurs de comparaison

=	Égal
<>	Différent
<	Inférieur strictement
>	Supérieur strictement
<=	Inférieur ou égal
>=	Supérieur ou égal
BETWEEN <valeur1> AND <valeur2>	Appartient à un intervalle (<valeur2> incluse)
IN <liste de valeurs>	Appartient à un ensemble de valeurs
IS NULL	Colonne non renseignée
LIKE <chaîne>	Recherche chaînes de caractères correspondant à un modèle

Connecteurs logiques

<condition1> AND <condition2>	Les 2 conditions sont vraies simultanément
<condition1> OR <condition2>	Au moins une des 2 conditions est vraie

NOT <condition>	Inverse la condition

Exemple

Supprimons l'ingrédient dont la colonne *nomIngredient* porte la valeur « rutabaga » dans la table *ingredient*&nbp;:

```
DELETE FROM ingredient
WHERE nomIngredient = "rutabaga";
```

Avant suppression :

idIngredient	nomIngredient	type
1	oeuf	viande
2	moutarde	condiment
3	vinaigre	condiment
4	huile	condiment
5	sel	condiment
6	poivre	condiment
7	gruyère râpé	crèmerie
8	huile d'olive	condiment
9	farine	épicerie
10	beurre	crèmerie
11	sucre	épicerie
12	lait	crèmerie
13	carotte	légume

14	crème fraîche	crèmerie
15	cumin	condiment
16	muscade	condiment
17	poulet	viande
18	persil	condiment
19	citron	fruit
20	sole	viande
21	rutabaga	légume
22	levure	épicerie

Après suppression de la ligne :

idIngredient	nomIngredient	type
1	oeuf	viande
2	moutarde	condiment
3	vinaigre	condiment
4	huile	condiment
5	sel	condiment
6	poivre	condiment
7	gruyère râpé	crèmerie
8	huile d'olive	condiment
9	farine	épicerie

10	beurre	crèmerie
11	sucre	épicerie
12	lait	crèmerie
13	carotte	légume
14	crème fraîche	crèmerie
15	cumin	condiment
16	muscade	condiment
17	poulet	viande
18	persil	condiment
19	citron	fruit
20	sole	viande
22	levure	épicerie

Le SGBD n'a pas refusé d'exécuter la commande : l'ingrédient supprimé n'était référencé dans aucune autre table, ou, si c'était le cas, l'option a été choisie, dans les tables concernées (ici la table *utilise*), de supprimer les lignes dont la colonne clé étrangère correspond à l'ingrédient supprimé, ou de renseigner à « NULL » cette clé étrangère.

Nous pouvons émettre une critique sur le critère de sélection : celui-ci porte sur la colonne *nomIngredient* qui ne constitue pas une clé primaire pour la table *ingredient*. Cela impose au SGBD de parcourir séquentiellement les lignes de la table jusqu'à trouver celle dont la colonne *nomIngredient* répond au critère.

Astuce

Pour être plus efficace et plus fiable, il aurait été préférable de repérer l'ingrédient de nom « rutabaga » par sa colonne *idIngredient* (en l'occurrence, la ligne 21) : le SGBD aurait atteint la ligne directement, grâce à l'index de la table. De plus, il aurait identifié à coup sûr et de manière unique l'ingrédient visé, ce qui est préférable.

c. La modification d'une ou plusieurs lignes d'une table

À retenir

Une opération de modification requiert que l'on précise :

- les colonnes concernées ;
- les nouvelles valeurs prises par ces colonnes ;
- les lignes concernées, via la clause `WHERE`.

Astuce

Comme pour l'insertion ou la suppression d'une ligne dans une table, la modification d'une ligne doit respecter les contraintes d'intégrité spécifiées lors de la création de la structure de la base de données.

La syntaxe d'une modification est la suivante :

```
UPDATE nomTable
SET nomColonne1 = valeur 1, nomColonne2 = valeur2, ...
WHERE condition
```

Exemple

Modifions le nom de la recette dont la colonne *nomRecette* contient « Omelette » et le niveau de difficulté actuellement évalué à « facile », sachant que son identifiant est 2.

```
UPDATE recette
SET nomRecette = "Omelette au fromage", niveau = "moyen"
WHERE idRecette = 2;
```

Avant modification :

idRecette	nomRecette	niveau	duree	categorie	idAdh
1	Mayonnaise	difficile	15	sauce	7
2	Omelette	facile	5	plat	6
3	Pâte brisée	moyen	20	autre	6
4	Quatre-quarts	facile	10	dessert	4
5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2
7	Sole meunière	difficile	20	plat	7

Après modification de la ligne :

idRecette	nomRecette	niveau	duree	categorie	idAdh
1	Mayonnaise	difficile	15	sauce	7
2	Omelette au fromage	moyen	5	plat	6
3	Pâte brisée	moyen	20	autre	6

4	Quatre-quarts	facile	10	dessert	4
5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2
7	Sole meunière	difficile	20	plat	7

Nous savons maintenant enregistrer, modifier et supprimer des données dans une base. L'objectif final de tout ce qui a été réalisé jusqu'ici est cependant de pouvoir **consulter ces données, selon des critères variés**. Nous allons donc aborder maintenant la manière d'**interroger la base**.

3 | Interrogation des données

Pour interroger les données d'une base, on a recours à un LID (Langage d'interrogation des données).

→ SQL intègre un LID.

Lorsque l'on effectue une recherche dans une base de données, le résultat de celle-ci se présente lui-même sous forme d'une table, que l'on nommera par la suite « table résultat ».
Détaillons maintenant la syntaxe d'une instruction de recherche, autrement dénommée **requête**.

a. La projection

La projection consiste à sélectionner la ou les colonnes que l'on souhaite voir apparaître dans la table résultat. C'est l'instruction SELECT qui permet cela.

La clause SELECT

La syntaxe d'une recherche est la suivante :

```
SELECT nomColonne1, nomColonne2, ...
FROM nomTable;
```

Si l'on souhaite que toutes les colonnes apparaissent dans la table résultat, il suffit alors d'employer la syntaxe suivante :

```
SELECT *
FROM nomTable;
```

Exemple

On souhaite obtenir la liste des recettes du club, présentée en trois colonnes : le nom de la recette, sa durée et son niveau de difficulté.

Le schéma de la relation recette est le suivant :

```
recette(idRecette, nomRecette, niveau, duree, categorie, idAdherent) avec
        idRecette : clé primaire
        idAdherent : clé étrangère en référence à idAdherent dans adherent
```

La commande SQL sera donc la suivante :

```
SELECT nomRecette, duree, niveau
FROM recette;
```

nomRecette	duree	niveau
Mayonnaise	15	difficile
Omelette au fomage	5	moyen
Pâte brisée	20	moyen
Quatre-quarts	10	facile
Purée de carottes	15	moyen

Poulet rôti	15	facile
Sole meunière	20	difficile

Valeurs distinctes d'une colonne

Si l'on souhaite connaître toutes les valeurs prises par une colonne dans une table, sans que celles-ci soient répétées autant de fois qu'elles s'y retrouvent, on le précise avec la clause `DISTINCT`.

 Exemple

La table adhérent est ainsi constituée :

1	DURAND	Jacques	j.durand@orange.fr	16b	rue brune
2	DUVAL	Paul	paul.d@sfr.fr	135	avenu rouge
3	DELORS	Sophie	sodelors@outlook.com	22a	bd ve
4	VERGER	Jehan	jverger@aol.com	4ter	impas jaune
5	MONVILLE	Fleur	fleur1212@free.fr	9	sente aux loups
6	BRIARD	Paul	paul.briard27@hotmail.fr	191	rue violet
7	LÉGER	Marguerite	mag0506@perso.com	17	place noire

8	FLAMAND	Lise	flise@gmail.com	329	route bleue

On souhaite connaître la liste des prénoms de tous les adhérents du club de cuisine. La commande SQL qui nous vient est :

```
SELECT prenom
FROM adherent;
```

Jacques
Paul
Sophie
Jehan
Fleur
Paul
Marguerite
Lise

Mais si plusieurs adhérent·e·s portent le même prénom, par exemple deux adhérents s'appellent Paul, on retrouvera deux fois le prénom Paul dans la table résultat de la requête. Pour éviter cela, on affine notre commande avec la clause `DISTINCT` :

```
SELECT DISTINCT prenom
FROM adherent;
```

Jacques

Paul
Sophie
Jehan
Fleur
Marguerite
Lise

b. La sélection

La sélection (ou restriction) est le recours à un ou plusieurs critères pour retenir certaines lignes dans la table résultat de la requête.

→ C'est avec la clause `WHERE` que l'on précise ces critères.

Nous avons déjà détaillé cette clause lorsque nous avons étudié les commandes de suppression et de modification de lignes. Ces commandes requièrent en effet de préciser sur quelles lignes elles portent, donc nécessitent la clause `SELECT`. L'usage de `SELECT` dans une requête d'interrogation ne diffère pas de celui dans les requêtes de suppression ou de mise à jour de lignes.

Ainsi, si l'on souhaite effectuer une projection sur une sélection, la commande SQL prendra la forme

```
SELECT colonnex, colonney, ...
FROM nomTable
WHERE conditions
```

Exemple

> La commande SQL qui suit recueille la liste des recettes dont la durée n'excède pas 10 minutes, constituée du nom de la recette, de sa durée et de sa catégorie :

```
SELECT nomRecette, duree, categorie
FROM recette
WHERE (duree <= 10);
```

nomRecette	duree	categorie
Omelette au fromage	5	plat
Quatre-quarts	10	dessert

Conclusion :

Nous achevons ici ce cours relatif aux commandes SQL de base. Nous savons créer des tables, les modifier et les supprimer. Nous avons vu comment insérer, supprimer ou modifier des lignes dans une table. Et nous pouvons également effectuer des recherches dans celle-ci.
Nous ne savons pas encore effectuer des recherches faisant appel à plusieurs tables, ni des calculs ou des tris sur les résultats de nos recherches. C'est ce que nous aborderons dans le cours qui suit.