Introduction au langage SQL

Les bases de données relationnelles

Le langage SQL est un langage universel destiné à travailler sur des bases de données relationnelles. Nous considérerons ici qu'une base de données relationnelle est constituée d'un ensemble de tables, chaque table pouvant être décrite sous la forme d'un tableau où les colonnes correspondent à des variables et les lignes à des enregistrements.

Par exemple, le tableau ci-dessous donne la valeur des variables production (P1), consommation intermédiaire (P2) et valeur ajoutée (B1) de plusieurs secteurs d'activité (AE).

AE	P1	P2	B1
A01	800	300	500
A02	500	400	100
A02	600	450	150
A03	200	50	150
B01	700	400	300
B01	900	500	400

La première ligne du tableau donne le nom des différentes variables, les autres lignes correspondent aux enregistrements. Dans une table, le nombre d'enregistrements n'est pas fixé et est seulement limité par les capacités de la mémoire, mais la contrainte fondamentale est que chaque enregistrement doit avoir la même structure, c'est-à-dire comporter les mêmes variables et cela dans le même ordre. Ainsi, si l'on ajoutait un nouvel enregistrement à cette table, il devrait impérativement comprendre les variables AE, P1, P2 et B1 dans cet ordre.

La commande SELECT

La commande SELECT est l'une des plus importantes du langage SQL, c'est elle qui permet de sélectionner les variables que l'on veut lire dans une table. Elle est toujours accompagnée au minimum de la clause FROM qui permet de préciser sur quelle table on va travailler. La commande SELECT doit être suivie de la liste des variables retenues, les variables étant séparées par des virgules. La clause FROM doit être

suivie du nom de la table. Un ensemble de commandes SQL se nomme une requête et chaque requête doit se terminer par un point-virgule.

Par exemple, si la table ci-dessus s'appelle PROD la requête suivante :

```
SELECT AE, B1
FROM PROD
;
```

renvoie :

AE	B1
A01	500
A02	100
A02	150
A03	150
B01	300
B01	400

On peut aussi avec la commande SELECT sélectionner des combinaisons de variables et leur donner un nom nouveau (un alias) en utilisant la clause AS, c'est un moyen simple de faire certains calculs avec SQL. Par exemple :

```
SELECT AE, 2*P1 AS DP1, P2+B1 AS NP1
FROM PROD
;
```

Renvoie :

AE	DP1	NP1
A01	1600	800
A02	1000	500
A02	1200	600
A03	400	200
B01	1400	700
B01	1800	900

La commande SELECT * permet de sélectionner l'ensemble des variables sans avoir besoin de les détailler dans une liste.

La clause WHERE

La clause WHERE permet de sélectionner les enregistrements correspondant à un certain critère. La clause WHERE vient obligatoirement après la clause FROM et les expressions alphanumériques doivent être encadrées par de simples quotes (et non des guillemets). Par exemple :

```
SELECT AE, B1
FROM PROD
WHERE AE='A02'
;
```

renvoie :

AE	B1
A02	100
A02	150

Ou encore :

```
SELECT AE, B1
FROM PROD
WHERE P1>700
;
```

renvoie :

AE	B1
A01	500
B01	400

Notons ici que, dans une requête SQL, les valeurs alphanumériques doivent impérativement être encadrées par des simples quotes et que les valeurs numériques ne doivent pas être encadrées. De plus, le séparateur décimal doit obligatoirement être un point car la virgule est interprétée comme un séparateur de valeurs. Par exemple, le chiffre qui s'écrit 23,4 en français doit s'écrire 23.4 en SQL.

Il est possible de réaliser des critères complexes en utilisant notamment les opérateurs AND, OR et NOT.

Par exemple :

```
SELECT AE, B1
FROM PROD
WHERE AE='A02' AND P1>500
;
```

renvoie :

AE	B1
A02	150

On peut également travailler avec des listes en utilisant le prédicat IN. La liste des valeurs que peut prendre la variable est placée entre parenthèses, chaque élément de la liste étant séparé des autres par des virgules et placé entre quotes s'il correspond à une valeur alphanumérique. Par exemple :

```
SELECT AE, B1
FROM PROD
WHERE AE IN ('A01','B01')
;
```

renvoie :

AE	B1
A01	500
B01	300
B01	400

Il est également possible d'utiliser l'opérateur LIKE. Celui-ci utilise les deux caractères _ (souligné) et % (pourcentage) pour chercher des chaînes de caractères. Le caractère _ est utilisé pour remplacer un caractère dans une chaîne et le caractère le caractère % remplace 0, 1 ou plusieurs caractères.

Par exemple :

```
SELECT AE FROM PROD WHERE AE LIKE '_01'
```

renvoie A01 et B01.

SELECT AE FROM PROD WHERE AE LIKE '%2'

renvoie A02/

SELECT AE FROM PROD WHERE AE LIKE 'B%'

renvoie B01/

La clause GROUP BY

On peut également utiliser dans des requêtes SQL des fonctions d'agrégation comme la somme SUM, le nombre COUNT, la moyenne AVG, la valeur maximale MAX ou la valeur minimale MIN. Ce sont ces fonctions d'agrégation qui font tout l'intérêt de SQL pour le comptable national. La fonction COUNT(*) permet de compter tous les enregistrements d'une table. Ces fonctions d'agrégation peuvent être utilisées sur l'ensemble de la table ou sur des regroupements. Dans ce dernier cas, il faudra utiliser la clause GROUP BY qui permet de regrouper les enregistrements par critères. Cette clause GROUP BY doit se placer après la clause WHERE. Par exemple, la requête suivante permet de calculer la somme de la variable B1 pour chaque modalité de l'activité AE :

```
SELECT AE, SUM(B1)
FROM PROD
GROUP BY AE
;
```

Elle renvoie :

AE	B1
A01	500
A02	200
A03	150
B01	700

La clause ORDER BY

La clause ORDER BY permet de trier les résultats d'une requête. Elle se place à la fin de la requête et spécifie la ou les variables selon lesquelles le tri doit être fait. Par défaut, le tri est fait par ordre croissant mais il est possible de trier par ordre décroissant. Ainsi, en ajoutant ASC après le nom de la variable on obtient un tri croissant, en ajoutant DESC on obtient un tri décroissant.

Par exemple, la requête suivante :

```
SELECT AE, P1
FROM PROD
ORDER BY P1 ASC
;
```

renvoie :

AE	P1
A03	200
A02	500
A02	600
B01	700
A01	800
B01	900

La requête suivante associe un tri par ordre croissant sur une variable et un tri décroissant sur une autre variable :

```
SELECT AE, P1
FROM PROD
ORDER BY AE DESC, P1 ASC
;
```

renvoie :

AE	P1
B01	700
B01	900
A03	200
A02	500
A02	600
A01	800

La commande INSERT INTO

Pour entrer des données dans une table SQL on peut utiliser la commande INSERT INTO. Celle-ci peut s'utiliser de deux manières

différentes. La première consiste à introduire les nouveaux enregistrements un par un. Ainsi la commande INSERT INTO doit être suivie en combinaison avec la clause VALUES. La commande INSERT INTO doit être suivie par le nom de la table puis, de manière facultative par la liste des variables mise entre parenthèses. La clause VALUES doit être suivie de la liste des valeurs, mise entre parenthèses, chaque élément étant séparé par une virgule. Les variables alphanumériques doivent être mises entre quotes et les variables numériques doivent utiliser le point comme séparateur décimal. Par exemple :

```
INSERT INTO Prod (AE, P1, P2, B1)
VALUES ('A03', 2000, 800, 1200)
;
```

L'ordre de la liste des valeurs doit correspondre à celui de la liste des variables. Lorsque la liste des variables a été omise dans la commande INSERT INTO l'ordre de la liste des valeurs doit correspondre à celui des variables tel qu'il a été défini au moment de la création de la table.

La deuxième manière d'utiliser la commande INSERT INTO est d'introduire dans une table une série de valeurs à partir d'une autre table. La commande INSERT INTO sera alors utilisée en liaison avec la commande SELECT FROM. Par, exemple, si nous disposons d'une table nommée Production contenant les mêmes variables que la table Prod, la requête suivante permettra d'insérer dans la table Prod tous les enregistrements de la table Production :

```
INSERT INTO Prod
SELECT * FROM Production
;
```

La commande DELETE FROM

Il est souvent utile de détruire des enregistrements d'une table. Cela peut se faire avec la commande DELETE FROM qui est le plus souvent associée à la clause WHERE. La commande DELETE FROM doit être suivie du nom de la table, la clause WHERE fonctionne comme avec la commande SELECT. Par exemple, la requête suivante élimine de la table Prod tous les enregistrements pour lesquels la variable P1 est inférieure à 1000 :

```
DELETE FROM Prod
WHERE P1<1000
;
```

Lorsque la clause WHERE est omise c'est l'ensemble des enregistrements de la table qui est supprimé.

La commande UPDATE

La commande UPDATE permet de modifier les enregistrements d'une table. Sa syntaxe la plus courante est, par exemple, la suivante :

```
UPDATE PROD
SET P1 = 500
WHERE B1<400
```

Ici nous avons donné la valeur 500 à P1 dans tous les enregistrements de la table PROD où B1 est inférieur à 400.

La commande UPDATE permet ainsi de modifier un enregistrement s'il est bien spécifié par la clause WHERE.

Si la clause WHERE n'est pas spécifiée, c'est l'ensemble des enregistrements qui est modifié.

Les fonctions

SQL permet d'utiliser de nombreuses fonctions qui ne sont pas toutes normaliisées et qui peuvent varier d'une version à l'autre. Nous ne présenterons ici que certaines parmi les plus utiles acceptées par le SQL d'Access.

Les fonctions de date

NOW : renvoie la date d'aujourd'hui, par exemple 16/01/2019 ;
DAY : renvoie le jour d'une date. Dans notre exemple, DAY(NOW) et DAY('16/01/2019') renvoient 16 ;
MONTH : renvoie le mois d'une date. Dans notre exemple, MONTH(NOW) renvoie 1 ;
YEAR : renvoie l'année d'une date. Par exemple, YEAR(NOW) renvoie 2019.

Les fonctions numériques

ABS : valeur absolue ;
EXP : exponentielle ;
LOG : logarithme népérien ;
ROUND : arrondi, par exemple ROUND(10.6)=11.

Les fonctions de texte

LEFT : renvoie la partie gauche d'un texte, par exemple LEFT('Bonjour',3) renvoie *Bon* ;
MID : renvoie une partie d'un texte, par exemple MID('Bonjour',3,2) renvoie *nj* ;

RIGHT : renvoie la partie droite d'un texte, par exemple RIGHT('Bonjour',3) renvoie *our* ;
TRIM : supprime les espaces à droite et à gauche d'une chaîne ;
LEN : la longueur d'une chaîne de caractères, par exemple LEN('Bonjour') renvoie 7 ;
INSTR : renvoie la première position d'un texte à l'intérieur d'une chaîne, par exemple INSTR('Bonjour','nj')=3, INSTR('Bonjour','o')=2, si le texte n'est pas trouvé INSTR renvoie 0 ;
REPLACE : remplace une partie de texte par un autre, par exemple REPLACE('Bonjour', 'jour', 'ne nuit') renvoie *Bonne nuit* ;
SPACE : renvoie des espaces, par exemple 'A' & SPACE(2) & 'B' renvoie *A B*.

Les jointures

L'une des grandes forces de SQL est de permettre le travail avec plusieurs tables. Par exemple, supposons que les données des tables ci-dessus correspondent à des volumes et que nous désirions calculer des valeurs à partir d'indices de prix. Nous pouvons stocker nos indices de prix dans une table PRIX qui se présentera de la manière suivante :

AE	INDICE
A01	110
A02	120
A03	100
B01	120

Pour obtenir les valeurs de la production, nous pouvons combiner la table PROD qui contient les volumes de la production et la table PRIX qui contient les indices de prix. Nous allons, pour cela, réaliser une jointure. Cette jointure va créer de nouveaux enregistrements qui reprendront les variables sélectionnées de chacune des deux tables. Chaque enregistrement de la première table sera associé à chaque enregistrement de la deuxième table, c'est-à-dire que la jointure réalise un produit cartésien. Dans notre exemple, la table PROD comporte 6 enregistrements et la table PRIX 4 enregistrements. La jointure va donc générer 6 x 4 =24 enregistrements.

Cependant, il n'est en général, pas intéressant de réaliser toutes les combinaisons possibles entre les deux tables. Ainsi, dans notre exemple, nous allons associer un enregistrement du tableau PROD à l'enregistrement de la table PRIX qui a la même activité afin d'obtenir

une valeur qui a un sens. Aussi, la jointure devra-t-elle être accompagnée d'une clause établissant un lien entre les deux tables afin de n'associer que des enregistrements de même activité. Dans la commande SELECT on fera apparaître les variables des deux tables préfixées du nom des tables, dans la clause FROM les noms des deux tables apparaîtront séparées d'une virgule. Ainsi, par exemple, la requête suivante :

```
SELECT PROD.AE, PROD.P1, PRIX.INDICE, PROD.P1*PRIX.INDICE/100
AS VALEUR
FROM PROD, PRIX
WHERE PROD.AE=PRIX.AE
;
```

renvoie :

AE	P1	INDICE	VALEUR
A01	800	110	880
A02	500	120	600
A02	600	120	720
A03	200	100	200
B01	700	120	840
B01	900	120	1080

Comme une même variable peut apparaître dans plusieurs tables différentes, il est nécessaire, dans la liste des variables de la requête SQL, de préciser la table à laquelle la variable appartient. Pour cela, on fait précéder le nom de la variable du nom de la table à laquelle elle appartient avec un point pour séparer le nom de la table du nom de la variable. Par exemple, PROD.AE désigne la variable AE appartenant au fichier PROD. Ici, nous aurions également pu choisir PRIX.AE qui est la variable AE de la table PRIX.

On peut également utiliser des alias de nom de tables dans les requêtes pour alléger leur écriture. L'utilisation d'alias est même obligatoire lorsque les noms de tables ont une extension, par exemple en .csv. Ainsi la requête précédente peut encore s'écrire :

```
SELECT P.AE, P.P1, X.INDICE, P.P1*X.INDICE/100 AS VALEUR
FROM PROD P, PRIX X
WHERE P.AE=X.AE
;
```

L'alias est défini dans la clause WHERE et il apparaît après le nom de la table, séparé par un espace.

La commande INNER JOIN

Une autre manière, peut-être plus claire, d'effectuer une jointure entre plusieurs tables est d'utiliser la commande INNER JOIN. Par exemple, la requête précédente peut également s'écrire :

```
SELECT PROD.AE, PROD.P1, PRIX.INDICE, PROD.P1*PRIX.INDICE/100
AS VALEUR
FROM PROD INNER JOIN PRIX
ON PROD.AE=PRIX.AE
;
```

On a ainsi remplacé la virgule séparant les deux tables par INNER JOIN et on a remplacé le WHERE par ON.

Ce type de jointure qui associe deux tables en sélectionnant des enregistrements ayant des valeurs communes dans les deux tables est appelé une jointure interne.

Les auto-jointures

Il est possible et souvent intéressant d'effectuer une jointure entre une table et elle-même. Supposons, par exemple, que nous voulions calculer un prix moyen entre les activités A01 et A02 en donnant un poids de 0,4 à A01 et de 0,6 à A02. Nous pouvons effectuer la requête suivante :

```
SELECT 'A01-A02' as AE, 0.4*p.INDICE+0.6*x.INDICE as INDICE
FROM PRIX p, PRIX x
WHERE p.AE='A01' AND x.AE='A02'
;
```

Nous avons ici utilisé deux alias différents pour la table PRIX afin de sélectionner une première fois A01 et une seconde A02. Notons l'usage des cotes simples pour indiquer des valeurs alphanumériques et l'usage du point comme séparateur décimal. Nous avons également donné le nom "A01-A02" à AE pour ce nouvel enregistrement.

Nous obtenons :

AE	INDICE
A01-A02	116

Nous pouvons également intégrer le résultat dans la table PRIX :

```
INSERT INTO PRIX
SELECT 'A01-A02' as AE, 0.4*p.INDICE+0.6*x.INDICE as INDICE
FROM PRIX p, PRIX x
WHERE p.AE='A01' AND x.AE='A02'
;
```

AE	INDICE
A01	110
A02	120
A03	100
B01	120
A01-A02	116

Les commandes LEFT JOIN et RIGHT JOIN

Reprenons l'exemple de la table PROD :

AE	P1	P2	B1
A01	800	300	500
A02	500	400	100
A02	600	450	150
A03	200	50	150
B01	700	400	300
B01	900	500	400

Supposons que nous voulions lui associer la table SAL donnant le nombre de salariés par activité :

AE	NOMBRE
A01	30
A02	20
B01	40
B02	50

Nous voulons associer à chaque enregistrement de la table PROD un enregistrement de la table SAL. Pour cela, nous pouvons utiliser la commande LEFT JOIN de la manière suivante :

```
SELECT p.AE, p.P1, s.NOMBRE
FROM PROD p LEFT JOIN SAL s
ON p.AE=s.AE
;
```

La commande LEFT JOIN a la même syntaxe que la commande INNER JOIN. Nous obtenons alors la table suivante :

AE	P1	NOMBRE
A01	800	30
A02	500	20
A02	600	20
A03	200	
B01	700	40
B01	900	40

Dans ce tableau, on constate qu'il n'y a pas de valeur pour NOMBRE correspondant à l'activité A03. Ceci s'explique par l'absence d'enregistrement correspondant dans la table SAL. On voit ici la particularité de la commande LEFT JOIN, elle a repris tous les enregistrements de la table PROD, y compris ceux qui n'ont pas de correspondant dans la table SAL. Si l'on remplaçait la commande LEFT JOIN par la commande INNER JOIN, on trouverait la même table, sauf qu'elle n'aurait pas d'enregistrement correspondant à l'activité A03. C'est ce qui fait toute la différence entre la commande LEFT JOIN qui définit ce que l'on appelle une jointure externe et la commande INNER JOIN qui définit une jointure interne.

La commande RIGHT JOIN reprend tous les enregistrements de la table de droite et lui associe ceux de la table de gauche qui lui correspondent. Si on remplace LEFT JOIN par RIGHT JOIN dans la requête précédente, c'est-à-dire :

```
SELECT p.AE, p.P1, s.NOMBRE
FROM PROD p RIGHT JOIN SAL s
ON p.AE=s.AE
;
```

on obtient le résultat suivant :

AE	P1	NOMBRE
A01	800	30
A02	500	20
A02	600	20
B01	700	40
B01	900	40
		50

On voit ici que la jointure RIGHT JOIN donne une table qui a plus d'enregistrements que la table SAL. En fait, elle fait la jointure interne INNER JOIN et elle rajoute les enregistrements de SAL qui n'ont pas de correspondant dans PROD. Ici nous voyons que la dernière ligne ne comprend que la valeur 50 et ne montre pas AE. C'est simplement parce que nous avons sélectionné dans notre requête le AE provenant de la table PROD, ici il est plus pertinent de sélectionner la variable AE provenant de la table SAL :

```
SELECT s.AE, p.P1, s.NOMBRE
FROM PROD p RIGHT JOIN SAL s
ON p.AE=s.AE
;
```

On obtient alors :

AE	P1	NOMBRE
A01	800	30
A02	500	20
A02	600	20
B01	700	40
B01	900	40
B02		50

Ainsi, la commande LEFT JOIN fait d'abord une jointure INNER JOIN et rajoute les enregistrements de la table de gauche qui n'ont pas d'équivalent dans la table de droite, la commande RIGHT JOIN fait d'abord une jointure INNER JOIN et rajoute les enregistrements de la table de droite qui n'ont pas de correspondant dans la table de gauche.

Les commandes UNION et UNION ALL

On peut également fusionner deux tables en une seule. Supposons que nous ayons une deuxième table SAL1 portant sur le nombre de salariés :

AE	NOMBRE
B01	40
B02	50
C01	60
C02	80

Si nous voulons fusionner les tables SAL et SAL1, nous pouvons utiliser la commande UNION :

```
SELECT * FROM SAL
UNION
SELECT * FROM SAL1
;
```

Nous obtenons :

AE	NOMBRE
A01	30
A02	20
B01	40
B02	50
C01	60
C02	80

Nous voyons que les enregistrements correspondant à B01 et B02 qui étaient identiques dans les deux tables SAL et SAL1 n'ont été repris qu'une fois. Si nous voulons reprendre l'ensemble des enregistrements sans exclure les doublons, nous devons utiliser la commande UNION ALL :

```
SELECT * FROM SAL
UNION ALL
SELECT * FROM SAL1
;
```

Nous obtenons alors :

AE	NOMBRE
A01	30
A02	20
B01	40
B02	50
B01	40
B02	50
C01	60
C02	80

Nous avons réalisé ici l'union de deux tables qui avaient les mêmes colonnes, ce qui est généralement le cas le plus logique. Mais, ce qui importe réellement pour SQL, c'est uniquement le nombre de colonnes, si bien qu'il est possible de faire l'union de deux tables ayant le même nombre de colonnes mais pas le même nom dans les deux tables. Dans ce cas, SQL affectera les noms des colonnes de la première table à sa table de résultats. Par exemple, la requête :

```
SELECT * FROM SAL
UNION
SELECT * FROM PRIX
;
```

donne le résultat suivant :

AE	NOMBRE
A01	30
A02	20
B01	40
B02	50
A01	110
A02	120
A03	100
B01	120

Les sous-requêtes

Il est possible et souvent très utile d'utiliser des sous-requêtes à l'intérieur d'une requête.

Avec le prédicat IN

Par exemple, supposions que nous voulions sélectionner les enregistrements de la table PROD qui ont une activité correspondante dans la table SAL. Nous pouvons le faire grâce à la requête suivante :

```
SELECT *
FROM PROD
WHERE AE IN
(SELECT AE FROM SAL)
;
```

On obtient :

AE	P1	P2	B1
A01	800	300	500
A02	500	400	100
A02	600	450	150
B01	700	400	300
B01	900	500	400

Cette table reprend tous les enregistrements de la table PROD à l'exception de celui correspondant à "A03" puisque cette valeur de AE n'est pas dans la table SAL.

Il est également possible de sélectionner les enregistrements de PROD qui n'ont pas d'activité correspondante dans la table SAL. Pour cela, nous pouvons utiliser NOT IN au lieu de IN dans notre requête précédente :

```
SELECT *
FROM PROD
WHERE AE NOT IN
(SELECT AE FROM SAL)
;
```

Nous obtenons :

AE	P1	P2	B1
A03	200	50	150

Avec le prédicat EXISTS

Il est possible de parvenir aux résultats précédents en utilisant le prédicat EXISTS. Celui-ci sélectionne les enregistrements d'une table pour lesquels une requête renvoie un résultat différent de NULL. Par exemple, on peut sélectionner les enregistrements de la table PROD qui ont une activité correspondante dans la table SAL grâce à la requête suivante :

```
SELECT *
FROM PROD p
WHERE EXISTS
(SELECT s.AE
FROM SAL s
WHERE p.AE=s.AE)
;
```

On obtient comme précédemment :

AE	P1	P2	B1
A01	800	300	500
A02	500	400	100
A02	600	450	150
B01	700	400	300
B01	900	500	400

On peut également utiliser le prédicat NOT EXISTS pour sélectionner les enregistrements de PROD qui n'ont pas d'activité correspondante dans la table SAL.

```
SELECT *
FROM PROD p
WHERE NOT EXISTS
(SELECT s.AE
FROM SAL s
WHERE p.AE=s.AE)
;
```

On obtient :

AE	P1	P2	B1
A03	200	50	150

Le prédicat EXISTS est plus général que IN puisqu'il permet de travailler avec des requêtes retournant plus d'une colonne.

Avec la clause FROM

Une requête renvoie une table, celle-ci peut être utilisée dans une requête comme une table existant dans la base de données.

Supposons, par exemple, que nous voulions créer à partir des tables PROD, SAL et SAL1 une table reprenant AE, P1 et NOMBRE pour toutes les valeurs de SAL et SAL1. Cela est possible grâce à la requête :

```
SELECT s.AE, p.P1, s.NOMBRE
FROM PROD p RIGHT JOIN
(SELECT * FROM SAL
UNION
SELECT * FROM SAL1) s
ON p.AE=s.AE
;
```

Cette requête renvoie :

AE	P1	NOMBRE
A01	800	30
A02	600	20
A02	500	20
B01	900	40
B01	700	40
B02		50
C01		60
C02		80

Pour utiliser une sous-requête dans une requête SQL, il est indispensable de lui donner un nom, ici nous avons donné le nom "s" à la sous-requête.

Auteur : Francis Malherbe