Nouri Ahmad Asaf

Organisation et structures des données 1
Commandes Synthèse

Table des matières

Légende :

Tout ce qui est en rouge est quelque chose qui change en fonction de l'utilisateur et qui correspond à ses propres valeurs, exemple :

SELECT * FROM nom_de_la_table;

Tout ce qui est en bleu est des commandes exemple :

SELECT * FROM nom_de_la_table;

Tout ce qui est en orange est le type exemple : varchar, numeric, char, decimal, date, int, bigint, etc…

Et tout ce qui est en mauve est une contrainte qui donne des « exigences » à la colonne qu'on place après le type de la colonne,et on peut avoir plusieurs contraintes pour une même colonne(pas besoin de mettre des virgule), les contraintes peuvent également être places à la fin d'une création de table par exemple, pour qu'elle soit mise pour toute la table (ne pas oublier de mettre un virgule entre les contraintes de la table et les colonnes) :

- NOT NULL : obligatoire, donc l'inverse qui est NULL veut dire que ce n'est pas obligatoire, donc **facultatif**.
- PRIMARY KEY : identifiant principal (clé primaire), **pour chaque table on doit avoir une clé primaire !** (sauf si c'est une clé primaire composite)
- UNIQUE : valeur unique (appelé aussi clé secondaire), elle peut être utiliser plusieurs fois dans la même table et comme sont l'indique elle doit avoir une valeur unique.
- FOREIGN KEY : clé étrangère, on la rajoute pour une colonne de cette manière-là :

nom_de_la_cle_etrangere varchar(10) REFERENCES nom_de_lautre_table(la_cle_primaire_de_lautre_table)

Ou pour la table de cette façon-là (le nom de la colonne de la clé étrangère peut-être ≠) :

Nom_cle_etranger_peut_etre_different_mais_pas_le_type numeric(3),
FOREIGN KEY (la_cle_primaire_de_lautre_table) REFERENCES nom_de_lautre_table(la_cle_primaire_de_lautre_table)

- CHECK : contraintes additionnelles, on utilise cette contrainte dans le but de vérifier que la valeur donnée correspond à la condition, si ce n'est pas le cas la valeur est rejeté exemple :

CREATE TABLE exemple_table (nom varchar(20) PRIMARY KEY,age int CHECK(age >=18)) ;

Dans ce cas-là, la valeur qu'on va entrer doit être supérieur à 18, si non la colonne ne l'acceptera pas, et ceux même si on met les données de la colonne à jour.

Les noms (de bases de données, tables, colonnes...) doivent :

- Commencer par une lettre
- Contenir entre 1 et 30 caractères
- Contenir seulement A—Z, a—z, 0—9, _, $, et #
- Être différents (pour un même utilisateur)
- Ne pas être des mots réservés

Ils doivent être en snake-case exemple :

Ne pas mettre DateDeNaissance **<** mettre date_de_naissance

Les commandes SQL ne sont pas sensibles à la casse donc on peut mettre les commandes en MAJUSCULE ou en minuscule sa revient à la même chose.

Commandes de création :

On utilise la commande CREATE TABLE, pour comme son nom l'indique crée une table :

```
CREATE TABLE nom_de_la_table (colonne1 varchar(30) PRIMARY KEY, colonne2 numeric(10)) ;
```

On doit placer les colonnes qu'on veut ajouter entre la parenthèse (et on n'oublie pas de les séparer avec des virgules).

Les chiffres entre parenthèse après le type correspondent au nombre de caractère qu'on peut stocker.

Commande default :

On peut avoir des parties optionnelles d'une commandes comme par exemple default 'kilo', qui permet de définir l'unité de la colonne par défaut :

```
CREATE TABLE article (nom varchar(30),poids decimal(6,2),unite varchar(15) default 'kilo') ;
```

Commandes de modification/suppression :

On utilise la commande ALTER pour modifier un objet, dans notre cas on veut modifier la table donc on utilise la commande ALTER TABLE ensuite on spécifie la table, exemple :

```
ALTER TABLE nom_de_la_table
ADD COLUMN colonne3 varchar(5) UNIQUE;
```

ADD COLUM est une commande qui ajoute une colonne, qu'on peut utiliser qu'en passant par ALTER TABLE.

On peut également utiliser la commande ALTER COLUMN dans ALTER TABLE pour modifier par exemple le type d'une colonne existante :

```
ALTER TABLE nom_de_la_table
ALTER COLUMN colonne1 varchar(50);
```

Pour modifier ou ajouter une contraint on utilise la commande suivante :

```
ALTER TABLE nom_de_la_table
ADD CONSTRAINT nom_contrainte UNIQUE (colonne2);
```

Le nom de la contrainte n'est pas obligatoire mais il est recommandé car cela peut nous aider à résoudre certain bugs (dans le cas où on ne mets pas de nom de contrainte c'est le système qui le met).

On peut supprimer une colonne avec la commande :

```
ALTER TABLE nom_de_la_table
DROP COLUMN colonne3 ;
```

Et enfin on peut supprimer la table avec la commande :

```
DROP TABLE nom_de_la_table ;
```

Commandes d'insertion/modification ou de suppressions de valeur :

On peut insérer des valeurs dans notre table, avec cette commande :

```
INSERT INTO nom_de_la_table (colonne1, colonne2, etc…)

VALUES
        ('valeur1', valeur2, etc…),
        ('ligne2_valeur1', ligne2_valeur2, etc…)
        ('ligne3_valeur1', ligne3_valeur2, etc…) ;
```

On n'est pas obligé de mettre le nom des colonnes et c'est plutôt déconseillé pour réduire les chances d'erreur, mais si on veut insérer les valeurs dans cet ordre-là on peut utiliser cette façon-là.

On peut insérer plusieurs lignes tant qu'on met la virgule entre chaque parenthèse contenant des valeurs.

Si la colonne est de type chaines de caractères ou tout simplement un caractère on doit placer la valeur entre 'quotes'.

Si on ne connaît pas la valeur d'une colonne on peut insérer NULL, mais si la colonne à pour contrainte NOT NULL, alors on est obligé de mettre une valeur.

Pour ce qui est du contenu on peut aussi le « mettre à jour », donc remplacer des données déjà existant par d'autre :

```
UPDATE nom_de_la_table
SET colonne2 = valeur2, colonne3 = valeur3, etc…
WHERE colonne1 = valeur1 ;
```

Donc pour chaque colonne indiquer on remplace la valeur existant par la valeur après l'égale, si on ne précise pas où dans la table avec WHERE c'est toute la table qui va être changer, donc en général et de préférence on indique l'endroit qu'on veut changer avec la clé primaire, c'est donc la clé primaire de la ligne qui nous indique l'emplacement des valeurs des colonnes qu'on veut changer.

Avec DELETE on peut supprimer le contenue d'une table, exemple :

```
DELETE FROM nom_de_la_table
WHERE colonne1='valeur1' ;
```

Dans ce cas-là on efface les valeurs de la ligne qui contient la valeur de la clé primaire 'valeur1'.

Dans ce cas-là aussi si on oublie de spécifier avec le WHERE c'est toute la table qui va être supprimer.

On peut supprimer le contenu d'une table avec la commande (donc on garde la structure) :

```
TRUNCATE TABLE nom_de_la_table ;
```

Commandes d'affichage :

On peut utiliser la commande SELECT afficher la table comme on le souhaite, car avec la commande SELECT on récupère les données voulues pour former une table, ce qui veut dire qu'on afficher une table qui regroupe plusieurs tables dans notre base de données, on peut également afficher une partie de notre table seulement :

SELECT * FROM nom_de_la_table;

L'astérisque * indique qu'on récupère toutes les colonnes de la table.

	numero	nom
1	4	boulangerie
2	5	cremerie
3	1	fruit
4	2	legume
5	3	viande

On peut même choisir l'ordre des colonnes :

```
SELECT *
FROM nom_de_la_table
ORDER BY numero DESC ;
```

Dans se cas numero et nom correspond aux colonnes dans la table nom_de_la_table et numero c'est ceux sur quoi on va se référence pour faire l'ordre, qui est dans ce cas la décroissante (DESC).

	numero	nom
1	5	cremerie
2	4	boulangerie
3	3	viande
4	2	legume
5	1	fruit

On peut aussi bien le faire dans un ordre croissant (ASC)(commande par défaut).

	numero	nom
1	1	fruit
2	2	legume
3	3	viande
4	4	boulangerie
5	5	cremerie

Avec SELECT on peut également afficher une valeur après l'avoir calculé, exemple :

SELECT numero*2, nom FROM nom_de_la_table ORDER BY numero ;

	(Aucun nom de colonne)	nom
1	2	fruit
2	4	legume
3	6	viande
4	8	boulangerie
5	10	cremerie

C'est normal que ça soit afficher (Aucun nom de colonne) car on vient de créer une colonne inexistante, et qu'on peut voir qu'avec SELECT.

On peut aussi utiliser les « alias » qui crée un nom fictif à la colonne si elle n'existe pas, ou qu'elle est trop longue :

SELECT numero AS num, nom FROM nom_de_la_table ORDER BY numero ;

	num	nom
1	1	fruit
2	2	legume
3	3	viande
4	4	boulangerie
5	5	cremerie

Donc le nom de colonne numero est remplacé par num et on n'a pas changer les autres valeurs.

On peut utiliser dans le SELECT également le WHERE pour afficher seulement les valeurs qui nous intéresse comme par exemple :

SELECT * FROM nom_de_la_table WHERE numero < 4 ORDER BY numero;

	numero	nom
1	1	fruit
2	2	legume
3	3	viande

On peut également utiliser d'autre opérateur :

Opérateurs de comparaison

Opérateur	Signification
=	Égal
>	Plus grand
>=	Plus grand ou égal
<	Plus petit
<=	Plus petit ou égal
<>	Différent
BETWEEN ... AND ...	Entre deux valeurs (bornes incluses)
IN (...)	Égal à au moins un élément de la liste
LIKE	Correspond à un modèle (chaîne de caractères)
IS NULL	Est une valeur inconnue

+ Combinaisons des conditions : AND et OR

Quand on utilise le <> on ne doit pas oublier d'enlever le ORDER BY sinon ça ne marchera pas.

Le IN(…), on peut l'utiliser pour par exemple dire que les valeurs que je cherche doivent être égale aux valeurs que j'ai mis entre parenthèse.

Le LIKE est utilisé pour chercher des chaines de caractère ou des dates où on ne connait pas une partie, exemple (ne pas oublier de mettre ce qu'on cherche entre 'quotas') :

Si je ne connais pas tout le mot fruit et que je veux trouver le mot je tape :

SELECT nom FROM nom_de_la_table WHERE nom like '_ruit' ;

	nom
1	fruit

A se moment là il pourra trouver toutes les chaines de caractère qui se termine par ruit, on peut également utiliser % si on à plusieurs mots à trouver.

Le IS NULL c'est pour qu'on puisse voir toutes les valeurs qui sont inconnue.

On utilise le NOT pour inverser les opérateur, exemple IS NOT NULL va chercher toutes les valeurs qui ne sont pas NULL, NOT IN (…) va chercher tous les valeurs qui ne sont pas dans la parenthèse, etc…

On peut également ajouter la commande DISTINCT qui retire toutes les valeurs semblables :

```
SELECT numero FROM nom_de_la_table ORDER BY numero;
```

	numero
1	1
2	2
3	3
4	4
5	5
6	6
7	6
8	7

```
SELECT DISTINCT numero FROM nom_de_la_table ORDER BY numero;
```

	numero
1	1
2	2
3	3
4	4
5	5
6	6
7	7

Comme on peut le voir on à plus les deux 6 comme au-dessus.

Commandes sur les fonctions :

Tout ce qui est en rose est une fonction.

Les fonctions sont généralement utilisées pour l'affichage donc avec la commande SELECT.

Quelques fonctions :

- LOWER = Mettre en minuscule tout ce qui se trouve entre parenthèse.
- UPPER = Mettre en majuscule tout ce qui se trouve entre parenthèse.
- CONCAT = Rassembler deux colonnes en un, exemple :

```
SELECT CONCAT (nom, ' ',numero) FROM nom_de_la_table;
```

	(Aucun nom de colonne)
1	boulangerie4
2	cremerie5
3	fromagerie6
4	fruit1
5	laitrie7
6	legume2
7	sucrerie6
8	viande3

Cela peut être utile si on à deux colonnes nom et prénom pour pouvoir mettre un nom complet, obligation d'utiliser les parenthèses et les quotas du milieu.

- LEN = Compte le nombre de caractère que contient la chaîne de caractères (ne pas oublier la parenthèse et les quotas !).
- REPLACE = remplace le caractère spécifié par un autre, exemple : REPLACE('Hellk wkrld','k','o') → Hello world
- ROUND = Réduit le chiffre en fonction du chiffre après la virgule, exemple : ROUND(123.456,2) → 123.46
- LEFT = Fait la même action que ROUND, mais il le fait pour les chaînes de caractères.
- LTRIM = Enlève les espaces dans une chaînes de caractères.

- GETDATE() = Cette fonction renvoie la date et l'heure actuelle, on peut aussi extraire séparément l'année (YEAR(GETDATE())), le mois (MONTH(GETDATE())) et le jour (DAY(GETDATE())) d'une colonne (utiliser cette fonction après le SELECT).

	date_et_heure_actuelles
1	2023-12-13 00:17:45.093

Fonctions de groupements :

- AVG = C'est une fonction qui permet d'avoir la moyenne d'une colonne, exemple :

  SELECT AVG(numero) AS moyenne FROM nom_de_la_table ;

	numero	nom
1	1	fruit
2	2	legume
3	3	viande
4	4	boulangerie
5	5	cremerie
6	6	fromagerie
7	6	sucrerie
8	7	laitrie

→

	moyenne
1	4.250000

- SUM = Permet d'avoir la somme d'une colonne numérique, exemple :

  SELECT SUM(numero) AS somme FROM nom_de_la_table ;

	numero	nom
1	1	fruit
2	2	legume
3	3	viande
4	4	boulangerie
5	5	cremerie
6	6	fromagerie
7	6	sucrerie
8	7	laitrie

→

	somme
1	34

- COUNT() = Avec cette fonction on peut voir le nombre de ligne d'une colonne par exemple :
  SELECT COUNT(*) AS total_employes FROM employes ;

  Ou plus spécifiquement le nombre de ligne non NULL :
  SELECT COUNT(nom) AS total_noms_non_nuls FROM employes ;

  Ou encore qui compte les lignes d'un choix spécifique comme :
  SELECT COUNT(*) AS total_femmes FROM employes
  WHERE genre = 'Femme' ;

  Et finalement si on rajoute DISTINCT on pourra compter tous les éléments différents de la colonne :
  SELECT COUNT(DISTINCT nom) FROM employes ;

- MAX() et MIN() = La plus grande et la plus petite valeur de la colonne.

Commandes de sous-requête :

Les sous-requête sont des requêtes qui nous permettent d'avoir la condition pour la requête principale, exemple :

```
SELECT nom, salaire FROM employes
WHERE salaire > (SELECT AVG(salaire) FROM employes);
```

Dans notre exemple ici on affiche la colonne nom et le colonne salaire, et on dit qu'on prend les salaires supérieur à la moyenne, car la sous-requête qui se trouve entre parenthèse elle sélectionne la moyenne (avec AVG()) de la colonne salaire de la table employes.

On peut aussi sélectionner la moyenne des salaires sans forcément passer par des opérateur, exemple :

```
SELECT nom, salaire, (SELECT AVG(salaire) FROM employes) AS salaire_moyen
FROM employes ;
```

Opérateur	Signification
IN	Égal à n'importe quel élément de la liste
ANY	Doit être précédé de =, <>, >, <, <=, >=. Doit être vrai pour au moins une valeur de la liste
ALL	Doit être précédé de =, <>, >, <, <=, >=. Doit être vrai pour toutes les valeurs de la liste

Avec ce tableau là on peut voir des nouveaux opérateurs qu'on utilise pour les sous-requête (sauf IN qui est pour tous les types de requêtes)

SI par exemple on veut trouver n'importe quels salaires inférieurs aux salaires de nos employés dans le département Ventes :

```
SELECT nom, salaire FROM employes WHERE salaire > ANY
(SELECT salaire FROM employes WHERE departement = 'Ventes') ;
```

Ou si on veut voir tous les salaires inférieurs aux salaires du département Ventes :

```
SELECT nom, salaire FROM employes WHERE salaire > ALL
(SELECT salaire FROM employes WHERE departement = 'Ventes') ;
```

Commandes de groupement :

Avec les groupements on peut faire en sorte de groupé plusieurs lignes de la table en quelques-uns, par exemple si on a dans un magasin des clients vient acheter souvent un produit on peut voir la quantité qu'il à acheter pour ce produit-là :

```
SELECT client, SUM(quantite) AS quantite_totale FROM commandes
GROUP BY client ;
```

Donc on demande de sélectionner la colonne client puis de faire une autre colonne avec l'alias quantite_totale où on mettra la somme de la quantité par client.

On peut aussi faire la même chose mais avec plusieurs colonnes.

Si on a trois colonne categorie, fournisseur et prix, et qu'on prend la moyenne des prix par catégorie de fournisseur, exemple :

```
SELECT categorie, fournisseur, AVG(prix) AS prix_moyen FROM produits
GROUP BY categorie, fournisseur ;
```

Dans cet exemple on a trois colonne categorie où on trouve les catégorie de produit d'un magasin, fournisseur où on trouve les fournisseurs de chaque catégorie et on à l'alias prix_moyen où on trouve le prix moyen de chaque catégorie par fournisseurs.

Attention on ne peut pas sélectionner d'autre colonne que ceux concernés par le groupement.

```
SELECT produit, categorie, fournisseur, AVG(prix) AS prix_moyen FROM produits
GROUP BY categorie, fournisseur ;
```

On peut aussi donner des conditions au groupement avec la commande HAVING :

```
SELECT client, SUM(quantite) AS quantite_totale FROM commandes
GROUP BY client
HAVING SUM(quantite) > 15;
```

Dans cette commande on trouve HAVING qui donne la condition au groupement de seulement prendre la somme des valeurs supérieur à 15.

Commandes jointures :

Pour commencer si on prend tout simplement deux tables comme ceci :

```
SELECT * FROM categorie, test ;
```

On va avoir la combinaison des deux tables où on aura un produit cartésien qui fait qu'on multiplie le nombre de ligne dans une table par le nombre de ligne dans l'autre table, ce qui nous donne toutes les combinaisons de lignes possibles, donc dans cet exemple c'est 8 x 3 = 24 :

	numero	nom	categorie	prix	fourniseure
1	1	fruit	lait	30	lidel
2	1	fruit	légume	10	delhaze
3	1	fruit	vetement	20	zara
4	2	legume	légume	10	delhaze
5	2	legume	vetement	20	zara
6	2	legume	lait	30	lidel
7	3	viande	légume	10	delhaze
8	3	viande	lait	30	lidel
9	3	viande	vetement	20	zara
10	4	boulangerie	légume	10	delhaze
11	4	boulangerie	lait	30	lidel
12	4	boulangerie	vetement	20	zara
13	5	cremerie	vetement	20	zara
14	5	cremerie	lait	30	lidel
15	5	cremerie	légume	10	delhaze
16	6	fromagerie	légume	10	delhaze
17	6	fromagerie	lait	30	lidel
18	6	fromagerie	vetement	20	zara
19	6	sucrerie	lait	30	lidel
20	6	sucrerie	vetement	20	zara
21	6	sucrerie	légume	10	delhaze
22	7	laitrie	vetement	20	zara
23	7	laitrie	légume	10	delhaze
24	7	laitrie	lait	30	lidel

Pour effectuer une jointure il faut avoir les mêmes noms de colonne dans les deux tables qui feront le lien avec ce genre de commande :

```
SELECT employes.emp_id, employes.nom, employes.code_departement,
departements.nom_departement FROM employes INNER JOIN departements ON
employes.code_departement = departements.code_departement;
```

Donc on commence par sélectionné les colonnes des deux tables qui nous intéressent en précisant de quelle table elle s'agit de la table employes ou de la table departements ensuite on mets dans le FORM la première table qui est dans ce cas employes ensuite on ajoute INNER JOIN puis la deuxième table qui est departements, ensuite on ajoute le ON où on va préciser qu'elles sont les colonnes qui font le lien, dans ce cas-là on a code.departement qui se trouve dans les deux tables (si on avait inverser les emplacements entre employes et departements au niveau du FORM et du INNER JOIN on aurait eu les valeurs de la colonne code_departement qui se trouve dans la table départements).

On peut aussi ajouter plusieurs jointures de tables, en ajoutent tout simplement les colonnes qu'on veut ajouter et les liens avec plusieurs INNER JOIN et ON :

```
SELECT employes.emp_id, employes.nom , departements.nom_departement,
projets.nom_projet

FROM employes
INNER JOIN departements ON employes.departement_id = departements.departement_id

INNER JOIN projets ON employes.projet_id = projets.projet_id;
```

On peut aussi faire des jointures avec WHERE :

```
SELECT employes.emp_id, employes.nom, employes.code_departement,
departements.nom_departement

FROM employes, departements

WHERE employes.code_departement = departements.code_departement;
```

Utilisions avec plusieurs tables :

```
SELECT employes.emp_id, employes.nom, departements.nom_departement,
projets.nom_projet

FROM employes, departements, projets

WHERE employes.departement_id = departements.departement_id

  AND employes.projet_id = projets.projet_id ;
```