Débuter avec SQL

par Baptiste Wicht (home)

Date de publication : Le 23 Octobre 2006

Vous débutez dans la création de base de données SQL ? Alors, cet article est fait pour vous ! Vous y apprendrez à créer des tables, ajouter des données, modifier des enregistrements, ... Bref, toutes les fonctions de base de SQL.

I - Introduction....3
II - Création des tables....4
II-A - Clefs primaires et unicité....4
II-B - Contraintes de type....5
II-C - Possibilité de laisser blanc....5
II-D - Contraintes de validation....6
II-E - Contraintes d'intégrité référentielle....6
III - Manipulation de données....8
III-A - Récupération de données....8
III-B - Jointures....8
III-B-1 - Avec le SQL de base....8
III-B-2 - Avec JOIN....9
III-C - Insertion....10
III-D - Suppression....10
III-E - Modification....11
IV - Conclusion....12
IV-A - Remerciements....12
IV-B - Liens....12

I - Introduction

Les syntaxes que je propose dans ce tutoriel sont parfois épurées pour ne pas troubler le débutant qui ne va pas les employer à fond avant un moment. Si vous voulez les syntaxes complètes des commandes, je vous recommande le tutoriel SQL de A à Z que je propose en fin d'article.

II - Création des tables

Avant de voir comment créer une table, il vous faut savoir ce qu'est une table ? C'est tout simplement une structure de données. Elle contient des enregistrements (lignes) qui ont des valeurs spécifiques à des colonnes.

```
CREATE TABLE tablename (
 colonne type contraintes_pour_colonne,
 ...
 contraintes_pour_table
)
```

Explications sur la syntaxe

- table : C'est tout simplement le nom de la table que vous allez créer
- colonne : Le nom de la colonne
- type : Le type de données que contient la colonne
- contraintes_pour_colonne : Les contraintes pour la colonne
- contraintes_pour_table : Les contraintes pour la table

Comme vous le voyez, ce n'est pas si compliqué de créer une table. Mais les possibilités sont grandes.

On peut aussi définir des valeurs par défaut pour les colonnes avec DEFAULT suivi de la valeur. On peut mettre une valeur par défaut pour chaque colonne.

On va commencer par un exemple, une table recensant des personnes, on a le nom et le prénom de chaque personne, ainsi que son âge :

```
CREATE TABLE personnes (
 nom VARCHAR(50),
 prenom VARCHAR(50),
 age SMALLINT
)
```

Le nombre entre parenthèses est tout simplement la taille du type. Il faut savoir que les types sont variables et on peut donc leur donner la taille que l'on veut sans toutefois dépasser certaines limites.

II-A - Clefs primaires et unicité

Une clef primaire est un moyen d'identifier de manière unique un enregistrement et d'améliorer ainsi les vitesses de recherche et de parcours pour les requêtes.

L'unicité permet d'empêcher qu'une table contienne plusieurs des valeurs qui ne devrait pas exister en double normalement. Pour les colonnes autre que la clé primaire, on pourra utiliser la clause UNIQUE pour dire qu'une colonne (ou un ensemble de colonnes) ne peut pas contenir deux fois la même valeur (ou jeu de valeurs).

Prenons par exemple une table personnes dont chaque enregistrement posséde un id, un nom et un prénom, une langue et un numéro d'AVS. Tout d'abord, réfléchissons à ce qui doit être unique :

- L'id : Oui, il doit être unique, car ce sera l'identifiant de l'enregistrement
- Le nom : Non
- Le prénom : Non, par contre, le couple nom-prénom doit être unique (pas dans tous les cas, je vous l'accorde, mais pour l'exemple, c'est mieux)
- Le numéro d'AVS : Oui (l'AVS est l'assurance vieillesse en Suisse, elle forme une partie du premier pilier)
- La langue : Non

L'id étant l'identificateur de l'enregistrement, il en sera la clé primaire. AVS sera unique et on va créer une contrainte d'unicité sur le couple nom prénom :

```
CREATE TABLE personnes (
 id INT NOT NULL PRIMARY KEY,
 nom VARCHAR(50),
 prenom VARCHAR(50),
 avs INT UNIQUE,
 language VARCHAR(50),
 CONSTRAINT u_nom_prenom UNIQUE (nom, prenom)
)
```

La seule chose de nouveau dans cette commande est la syntaxe pour la contrainte d'unicité sur le couple nom-prénom. En fait, on crée une nouvelle contrainte, on lui donne un nom et on définit quelle est cette contrainte, dans notre cas, on dit qu'on ne peut pas avoir plusieurs fois une personne avec le même couple nom et prénom.

On n'a pas besoin de dire que id est unique, car une clé primaire l'est automatiquement.

On peut aussi définir des clés primaires sur deux colonnes. Imaginons une table associative (qui relie deux tables) auteurs_livres qui met en relation un auteur et un livre, on va utiliser une clé qui prend les deux valeurs comme index :

```
CREATE TABLE auteurs_livres(
 auteur_id INT,
 livre_id INT,
 CONSTRAINT id PRIMARY KEY (auteur_id, livre_id)
)
```

La syntaxe est la même que pour pour notre contrainte d'unicité.

II-B - Contraintes de type

Une contrainte de type est tout simplement une contrainte qui dit comme quoi les valeurs de cette colonne doivent toutes être du type spécifié.

Pour créer une telle contrainte, il suffit tout simplement de spécifier un type pour la colonne, ainsi une colonne déclarée INT n'acceptera pas de chaînes de caractères.

Comme on l'a vu plus haut, les types sont variables, ils n'ont pas une taille fixe. Par exemple, on peut dire qu'une colonne est une chaîne de caractères de 5 lettres et une autre de 200 lettres. Pour cela, on fait suivre le nom du type par sa taille entre parenthèses.

II-C - Possibilité de laisser blanc

La première chose à spécifier pour une colonne est si on lui laisse ou non le droit d'être laissée vide. Pour dire qu'une colonne ne peut pas être vide, on va utiliser NOT NULL et pour dire qu'une colonne peut éventuellement être vide, on va employer NULL.

Par exemple, dans le cas d'une base de données recensant les membres d'un forum, on est obligé de spécifier le pseudo et le password, mais le sexe peut-être laissé à discrétion du membre. On va donc faire une table comme ça :

```
CREATE TABLE t_users(
 pseudo VARCHAR(20) NOT NULL,
 password VARCHAR(16) NOT NULL,
 sexe VARCHAR(10) NULL
)
```

Ainsi on pourra avoir des personnes ayant renseignés leur sexe et d'autres dont l'info restera secrète.

II-D - Contraintes de validation

Une contrainte de domaine est tout simplement une contrainte qui permet de valider la valeur de la colonne. A chaque fois que l'on va ajouter un élément dans cette colonne, cette condition va se vérifier et l'enregistrement ne se fera qu'en cas de passage de la validation. Néanmoins, ces validations peuvent se révéler lourde, il ne faut donc pas en abuser.

Prenons par exemple, une table tests qui contiendrait une colonne note, on a la contrainte qu'une note est obligatoirement contenue entre 1 et 6.

```
CREATE TABLE tests (
 id INT NOT NULL PRIMARY KEY,
 nom VARCHAR(20),
 note INT CHECK(VALUE BETWEEN 1 AND 6)
)
```

La syntaxe ne montre pas de difficultés particulières, il suffit d'employer le mot clé CHECK suivi de la condition entre parenthères.

On peut définir toutes sortes de prédicats (conditions de validation) pour la validation, après, c'est à vous de créer le prédicat qui vous sert le mieux.

Il faut noter que l'on ne peut contrôler que les valeurs saisies dans l'insertion en cours et qu'une contrainte de validation ne peut pas aller faire des vérifications sur des valeurs préalablement entrées dans la table.

II-E - Contraintes d'intégrité référentielle

Tout d'abord, qu'est ce que l'intégrité référentielle ? C'est un ensemble de règles qu'on définit entre plusieurs tables qui nous permet d'être sûrs qu'un id qui pointe vers une autre table fait toujours référence à une valeur existante.

Bon d'accord, comme ça, ça a l'air incompréhensible, mais vous allez vite comprendre.

Prenons le cas d'une table livres avec un champ auteur_id qui fait référence au champ id de la table auteurs. Le champ auteur_id doit obligatoirement être égal à une valeur présente dans la colonne id d'auteurs. Pour cela, il va nous falloir définir une clé étrangère sur auteur_id :

```
CREATE TABLE livres(
 id INT NOT NULL PRIMARY KEY,
 auteur_id INT FOREIGN KEY REFERENCES auteurs(id)
)
```

On a donc employé FOREIGN KEY pour définir cette contrainte d'intégrité sur le champ auteur_id. On fait suivre FOREIGN KEY de REFERENCES suivi du nom de la table avec le nom du champ entre parenthèses. On est ainsi assuré qu'on ne peut pas insérer un livre qui a un auteur qui n'existe pas.

On peut aussi définir l'action à effectuer lors d'une suppression ou d'une modification de ce vers quoi pointe la clé la étrangère.

Si par exemple vous voulez que si vous supprimez un auteur, tous les livres de cet acteur soient supprimés, il vous suffit de faire :

```
auteur_id INT FOREIGN KEY REFERENCES auteurs(id)
```

```
  ON DELETE CASCADE
```

Si au contraire, vous voulez que la suppression échoue, vous pouvez faire :

```
auteur_id INT FOREIGN KEY REFERENCES auteurs(id)
 ON DELETE NO ACTION
```

Vous pouvez aussi faire la même chose avec ON UPDATE. Donc pour les modifications.

III - Manipulation de données

Par manipulation de données, on entend la suppression, l'ajout et la modification de données sur la base. Ces requêtes ne sont pas spécialement compliquée. On va voir dans les chapitres qui viennent les moyens de faire ceci.

III-A - Récupération de données

Pour aller chercher des données dans la base, on va employer la requête SELECT dont voici la syntaxe :

```
SELECT * ou liste_colonnne_à_inclure
FROM table
[WHERE predicats]
[ORDER BY ] liste_colonne_pour_tri [ASC | DESC]
```

Explication de la syntaxe

- * ou liste_colonne_à_inclure : C'est en fait ce que vous cherchez. Si vous voulez toutes les colonnes de la table, vous pouvez employer * sinon, spécifier les colonnes dont vous avez besoin.
- table : Le nom de la table dans laquelle vous allez chercher
- predicats : Conditions pour la recherche. Vous pouvez spécifier des conditions pour ne pas prendre toute la table, mais seulement certains enregistrements.
- liste_colonne_pour_tri : Vous pouvez trier les enregistrements dans l'ordre que vous voulez, par colonne.
- ASC ou DESC : C'est tout simplement l'ordre de tri, ASC pour normal et DESC pour à l'envers.

Un petit exemple, vous voulez récupérer dans la table t_auteurs, tous les auteurs de langue francaise et les trier par prénom, de plus, vous voulez tous les champs sauf l'id :

```
SELECT nom, prénom, langue FROM t_auteurs
WHERE langue = 'Français'
ORDER BY prenom
```

III-B - Jointures

Cette partie est un peu complexe que les autres, mais elle est très utile.. Les jointures vous permettront de complexifier les requêtes SELECT pour les rendre plus puissantes et pour récupérer plus d'informations qu'avec une simple requête SELECT.

Les jointures sont un moyen de mettre en relation plusieurs tables. Elles permettent d'exploiter plus à fond le modèle relationnel des bases de données. Cela permet donc de combiner des données depuis plusieurs tables. Il y a deux manières d'exploiter les jointures, soit au moyen de requêtes simples que nous avons déjà vu, soit au moyen de la clause JOIN, ce qui est conseillé.

Un simple utilisateur qui demande un listage de livres aimerait bien avoir le nom de la langue plutôt que son id qui ne lui servira à rien, ainsi avec les jointures, vous lui fournissez le livre avec le nom de la langue dans laquelle il a été écrite.

III-B-1 - Avec le SQL de base

Nous allons apprendre à utiliser ici les jointures avec la clause WHERE, néamoins, il faut savoir que ceci est hors norme SQL2 et que son seul intêret est la compatibilité avec de vieux SGBD. C'est pourquoi, je vous conseille fortement d'utiliser la clause JOIN.

Considérons deux tables :

```
CREATE TABLE t_langues(
 langue_id INT PRIMARY KEY,
 langue_nom VARCHAR(50) UNIQUE
)

CREATE TABLE t_ouvrages(
 ouvrage_id INT PRIMARY KEY,
 ouvrage_titre VARCHAR(150) UNIQUE,
 ouvrage_langue INT,
)
```

Nous voulons récupérer une liste avec les titres des ouvrages et la langue, et tout cela en une seule requête bien sûr. On va donc utiliser une requête SELECT sur deux tables :

```
SELECT langue_nom, ouvrage_titre
FROM t_langues, t_ouvrages WHERE langue_id = ouvrage_langue
```

Il ne faut pas oublier de mettre la condition, sinon, il va renvoyer toutes les langues associées avec tous les auteurs, ce qui nous donnera un nombre de résultats égal au nombre de langues multiplié par le nombre d'ouvrages, ce qui ne sert à rien !

Avec la requête que vous venons de faire, dans le cas ou une langue n'a pas d'ouvrages y référant elle n'est pas présente dans la liste. Nous pourrons résoudre ce problème avec les JOIN.

III-B-2 - Avec JOIN

On va reprendre l'exemple d'avant. On va donc utiliser une requête SELECT, mais avec des JOIN cette fois.

```
SELECT langue_nom, ouvrage_titre
FROM t_langues
 INNER JOIN t_ouvrages
 ON langue_id = ouvrage_langue
```

Concrètement, qu'est ce que fait cette requête : elle sélectionne langue_nom et ouvrage_titre, dans t_langues à laquelle elle ajoute les données de la table t_ouvrage où la langue de l'ouvrage est égale à la langue.

Cette jointure s'appelle la jointure interne.

Vous aurez remarqué que l'on n'affiche toujours pas une langue qui n'a pas d'ouvrage référent. Pour pallier à ce problème, on va employer un deuxième type de jointures : les jointures externes, qui sont très utiles justement dans le cas ou il y a des informations qui ne sont pas présentes des deux cotés.

On reprend la requête précedente avec une jointure externe :

```
SELECT langue_nom, ouvrage_titre
FROM t_langues
 LEFT OUTER JOIN t_ouvrages
 ON langue_id = ouvrage_langue
```

LEFT veut dire que l'on rajoute toute les lignes de la table de gauche (t_langues dans notre cas) qui n'ont pas été prises en compte par la condition de jointure. On peut aussi employer RIGHT pour rajouter les lignes de la table de droite ou FULL pour rajouter les lignes de la table de gauche et de celle de droite.

Il existe encore d'autres types de jointures :

- La jointure croisée : CROSS JOIN : Ce n'est rien d'autres que l'union de toutes les données de chaque table, ça revient en fait au même que notre première jointure avec select sans condition.
- La jointure d'union : UNION JOIN : Cela permet d'unir les résultats d'une table avec une autre. En fait, cela liste les résultats d'une table, puis les autres, donc on autant de lignes que l'addition des totaux de lignes des deux tables.

III-C - Insertion

Pour ajouter un ou plusieurs nouveaux enregistrements dans la base de données, il vous faudra employer la requête INSERT. En voici la syntaxe :

```
INSERT INTO table [(colonnes)]
VALUES (valeurs) | SELECT | DEFAULT VALUES
```

Explications sur la syntaxe

- table : le nom de la table cible
- colonnes : les colonnes dans lesquelles on veut insérer quelque chose ou rien si on veut insérer quelque chose dans toute les colonnes
- valeurs : Les valeurs pour chaque colonne
- SELECT : On peut aussi insérer le résultat d'une requête select dans notre table
- DEFAULT VALUES : On va entrer un nouvel enregistrement qui contiendra pour chaque colonne la valeur par défaut de celle-ci.

Exemple : Si on veut insérer une nouvelle personne dans la table personne, on procédera ainsi :

```
INSERT INTO personnes (nom,  prenom)
VALUES ("Wicht", "Baptiste")
```

Ou alors, si on veut insérer une personne avec les valeurs par défaut :

```
INSERT INTO personnes
DEFAULT VALUES
```

Maintenant, un exemple un peu plus complexe en utilisant la clause SELECT. On veut rajouter tous les auteurs dans la table t_personnes :

```
INSERT INTO t_personnes (nom, prenom, age)
SELECT nom, prenom, age FROM t_auteurs
```

III-D - Suppression

Pour supprimmer un ou plusieurs enregistrements d'une table, il suffit d'employer la requête DELETE :

```
DELETE FROM table
[WHERE predicat]
```

Explications sur la syntaxe

- table : La table dans laquelle on veut faire la suppression
- predicat : La condition qui va définir quelles lignes nous allons supprimer

Par exemple, si on veut supprimer tous les auteurs :

```
DELETE FROM t_auteurs
```

Ou alors tous les livres de plus de 1500 pages

```
DELETE FROM t_livres
WHERE pages > 1500
```

III-E - Modification

Pour modifier un ou plusieurs enregistrements existants, c'est la requête UPDATE qui va entrer en jeu :

```
UPDATE table
SET colonne1 = valeur1 [, colonne2 = valeur2 ...]
[WHERE predicat]
```

Explications sur la syntaxe

- table : La table dans laquelle on va faire les modifications
- colonne : la colonne dont on va modifier la valeur
- valeur : La nouvelle valeur
- predicat : La condition pour sélectionner les lignes que nous allons modifier

Par exemple, si on veut que toutes les personnes aient 18 ans, on fera :

```
UPDATE t_personnes
SET age = 18
```

Ou encore, si on veut que toutes les personnes de plus de 18 ans s'appellent Baptiste Wicht, on procédera ainsi :

```
UPDATE t_personnes
SET nom = 'Wicht' , prenom = 'Baptiste'
WHERE age > 18
```

IV - Conclusion

Toutes les requêtes sont très pratiques et assez simple à utiliser (sauf peut-être les jointures). On peut bien sûr aller encore plus loin avec SQL, mais je ne traiterai pas de ça ici. Si vous voulez (ou avez besoin) de plus d'informations sur les autres possibilités SQL, je ne peut que vous conseiller de lire le tutoriel proposé plus bas, il passe en revue toutes les possibilités offertes par SQL.

IV-A - Remerciements

Un énorme merci à **Xo** pour la relecture technique de cet article. Merci également à **loka** pour ses corrections.

IV-B - Liens

- **SQL de A à Z** par SQLPro