FORMATION AFPA

- DEVELOPPEUR LOGICIEL –

(NIVEAU III)

JJP

Les bases de données : SQL DATABASE

SOMMAIRE

Les bases de données : SQL DATABASE

1.] SQL CREATE DATABASE

La création d'une base de données en SQL est possible en ligne de commande. Même si les systèmes de gestion de base de données (SGBD) sont souvent utilisés pour créer une base, il convient de connaître la commande à utiliser, qui est très simple.

Syntaxe

Pour créer une base de données qui sera appelé « ma_base » il suffit d'utiliser la requête suivante qui est très simple :

```
CREATE DATABASE ma_base
```

Base du même nom qui existe déjà

Avec MySQL, si une base de données porte déjà ce nom, la requête retournera une erreur. Pour éviter d'avoir cette erreur, il convient d'utiliser la requête suivante pour MySQL :

```
CREATE DATABASE IF NOT EXISTS ma_base
```

L'option IF NOT EXISTS permet juste de ne pas retourner d'erreur si une base du même nom existe déjà. La base de données ne sera pas écrasée.

Options

Dans le standard SQL la commande CREATE DATABASE n'existe normalement pas. En conséquent il revient de vérifier la documentation des différents SGBD pour vérifier les syntaxes possibles pour définir des options. Ces options permettent selon les cas, de définir les jeux de caractères, le propriétaire de la base ou même les limites de connexion.

Les bases de données : SQL DATABASE

2.] SQL DROP DATABASE

En SQL, la commande DROP DATABASE permet de supprimer totalement une base de données et tout ce qu'elle contient. Cette commande est à utiliser avec beaucoup d'attention car elle permet de supprimer tout ce qui est inclus dans une base: les tables, les données, les index …

Syntaxe

Pour supprimer la base de données « ma_base », la requête est la suivante :

```
DROP DATABASE ma_base
```

Attention : cela va supprimer toutes les tables et toutes les données de cette base. Si vous n'êtes pas sûr de ce que vous faites, n'hésitez pas à effectuer une sauvegarde de la base avant de supprimer.

Ne pas afficher d'erreur si la base n'existe pas

Par défaut, si le nom de base utilisé n'existe pas, la requête retournera une erreur. Pour éviter d'obtenir cette erreur si vous n'êtes pas sûr du nom, il est possible d'utiliser l'option IF EXISTS. La syntaxe sera alors la suivante :

```
DROP DATABASE IF EXISTS ma_base
```

Les bases de données : SQL DATABASE

3.] SQL CREATE TABLE

La commande CREATE TABLE permet de créer une table en SQL. Un tableau est une entité qui est contenu dans une base de données pour stocker des données ordonnées dans des colonnes. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacun des colonne (entier, chaîne de caractères, date, valeur binaire …).

Syntaxe

La syntaxe générale pour créer une table est la suivante :

```
CREATE TABLE nom_de_la_table
(
    colonne1 type_donnees,
    colonne2 type_donnees,
    colonne3 type_donnees,
    colonne4 type_donnees
)
```

Dans cette requête, 4 colonnes ont été définies. Le mot-clé « type_donnees » sera à remplacer par un mot-clé pour définir le type de données (INT, DATE, TEXT …). Pour chaque colonne, il est également possible de définir des options telles que (liste non-exhaustive) :

- **NOT NULL :** empêche d'enregistrer une valeur nulle pour une colonne.
- **DEFAULT :** attribuer une valeur par défaut si aucune données n'est indiquée pour cette colonne lors de l'ajout d'une ligne dans la table.
- **PRIMARY KEY :** indiquer si cette colonne est considérée comme clé primaire pour un index.

Exemple

Imaginons que l'ont souhaite créer une table utilisateur, dans laquelle chaque ligne correspond à un utilisateur inscrit sur un site web. La requête pour créer cette table peut ressembler à ceci :

Les bases de données : SQL DATABASE

```
CREATE TABLE utilisateur (
    id_utilisateur INT NOT NULL auto_increment,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    email VARCHAR(255),
    date_naissance DATE,
    pays VARCHAR(255),
    ville VARCHAR(255),
    code_postal VARCHAR(5),
    nombre_achat INT,
    PRIMARY KEY (id_utilisateur)
) ENGINE = MYISAM;
```

Voici des explications sur les colonnes créées :

- **id_utilisateur :** identifiant unique qui est utilisé comme clé primaire et qui n'est pas nulle et dont l'id s'auto incrémente de 1 à chaque nouvel enregistrement.
- **nom :** nom de l'utilisateur dans une colonne de type VARCHAR avec un maximum de 100 caractères au maximum
- **prenom :** idem mais pour le prénom
- **email :** adresse email enregistré sous 255 caractères au maximum
- **date_naissance :** date de naissance enregistré au format AAAA-MM-JJ (exemple :1973-11- 17)
- **pays :** nom du pays de l'utilisateur sous 255 caractères au maximum
- **ville :** idem pour la ville
- **code_postal :** 5 caractères du code postal
- **nombre_achat :** nombre d'achat de cet utilisateur sur le site
- **PRIMARY KEY (id_utilisateur)** : indique que id_utilisateur est la clé primaire
- **ENGINE = MYISAM** : voir chapitre 7

Les bases de données : SQL DATABASE

4.] SQL TRUNCATE TABLE

En SQL, la commande TRUNCATE permet de supprimer toutes les données d'une table sans supprimer la table en elle-même. En d'autres mots, cela permet de purger la table. Cette instruction diffère de la commande DROP qui à pour but de supprimer les données ainsi que la table qui les contient.

A noter : l'instruction TRUNCATE est semblable à l'instruction DELETE sans utilisation de WHERE. Parmi les petite différences TRUNCATE est toutefois plus rapide et utilise moins de ressource. Ces gains en performance se justifie notamment parce que la requête n'indiquera pas le nombre d'enregistrement supprimés et qu'il n'y aura pas d'enregistrement des modifications dans le journal.

Syntaxe

Cette instruction s'utilise dans une requête SQL semblable à celle-ci :

```
TRUNCATE TABLE `table`
```

Dans cet exemple, les données de la table « table » seront perdues une fois cette requête exécutée. Son équivalent est également :

```
DELETE FROM `table`
```

Exemple

Pour montrer un exemple concret de l'utilisation de cette commande, nous pouvons imaginez un système informatique contenant la liste des fournitures d'une entreprise. Ces données seraient tout simplement stockées dans une table « fourniture ».

Table « fourniture » :

id	nom	date_ajout
1	Ordinateur	2013-04-05
2	Chaise	2013-04-14
3	Bureau	2013-07-18

Il est possible de supprimer toutes les données de cette table en utilisant la requête suivante :

```
TRUNCATE TABLE `fourniture`
```

Une fois la requête exécutée, la table ne contiendra plus aucun enregistrement. En d'autres mots, toutes les lignes du tableau présenté ci-dessus auront été supprimées.

Les bases de données : SQL DATABASE

5.] SQL ALTER TABLE

La commande ALTER TABLE en SQL permet de modifier une table existante. Il est ainsi possible d'ajouter une colonne, d'en supprimer une ou de modifier une colonne existante, par exemple pour changer le type.

Syntaxe de base

D'une manière générale, la commande s'utilise de la manière suivante :

```
ALTER TABLE nom_table
instruction
```

Le mot-clé « instruction » ici sert à désigner une commande supplémentaire, qui sera détaillée ci- dessous selon l'action que l'ont souhaite effectuer : ajouter, supprimer ou modifier une colonne.

Ajouter une colonne

Syntaxe

L'ajout d'une colonne dans une table est relativement simple et peut s'effectuer à l'aide d'une requête ressemblant à ceci :

```
ALTER TABLE nom_table
ADD nom_colonne type_donnees
```

Exemple

Pour ajouter une colonne qui correspond à une rue sur une table utilisateur, il est possible d'utiliser la requête suivante :

```
ALTER TABLE utilisateur
ADD adresse_rue VARCHAR(255)
```

Supprimer une colonne

Une syntaxe permet également de supprimer une colonne pour une table. Il y a 2 manières totalement équivalente pour supprimer une colonne :

```
ALTER TABLE nom_table
DROP nom_colonne
```

Ou (le résultat sera le même)

```
ALTER TABLE nom_table
DROP COLUMN nom_colonne
```

Modifier une colonne

Pour modifier une colonne, comme par exemple changer le type d'une colonne, il y a différentes syntaxes selon le SGBD.

MySQL

```
ALTER TABLE nom_table
MODIFY nom_colonne type_donnees
```

PostgreSQL

```
ALTER TABLE nom_table
ALTER COLUMN nom_colonne TYPE type_donnees
```

Ici, le mot-clé « type_donnees » est à remplacer par un type de données tel que INT, VARCHAR, TEXT, DATE …

Renommer une colonne

Pour renommer une colonne, il convient d'indiquer l'ancien nom de la colonne et le nouveau nom de celle-ci.

MySQL

Pour MySQL, il faut également indiquer le type de la colonne.

```
ALTER TABLE nom_table
CHANGE colonne_ancien_nom colonne_nouveau_nom type_donnees
```

Ici « type_donnees » peut correspondre par exemple à INT, VARCHAR, TEXT, DATE …

PostgreSQL

Pour PostgreSQL la syntaxe est plus simple et ressemble à ceci (le type n'est pas demandé) :

```
ALTER TABLE nom_table
RENAME COLUMN colonne_ancien_nom TO colonne_nouveau_nom
```

Les bases de données : SQL DATABASE

6.] SQL DROP TABLE

La commande DROP TABLE en SQL permet de supprimer définitivement une table d'une base de données. Cela supprime en même temps les éventuels index, trigger, contraintes et permissions associées à cette table.

Attention : il faut utiliser cette commande avec attention car une fois supprimée, les données sont perdues. Avant de l'utiliser sur une base importante il peut être judicieux d'effectuer un backup (une sauvegarde) pour éviter les mauvaises surprises.

Syntaxe

Pour supprimer une table « nom_table » il suffit simplement d'utiliser la syntaxe suivante :

```
DROP TABLE nom_table
```

A savoir : s'il y a une dépence avec une autre table, il est recommandé de les supprimer avant de supprimer la table. C'est le cas par exemple s'il y a des clés étrangères.

Intérêts

Il arrive qu'une table soit créé temporairement pour stoquer des données qui n'ont pas vocation à être ré-utiliser. La suppression d'une table non utilisée est avantageux sur plusieurs aspects :

- Libérer de la mémoire et alléger le poids des backups
- Éviter des erreurs dans le futur si une table porte un nom similaire ou qui porte à confusion
- Lorsqu'un développeur ou administrateur de base de données découvre une application, il est plus rapide de comprendre le système s'il n'y a que les tables utilisées qui sont présente

Exemple de requête

Imaginons qu'une base de données possède une table « client_2009 » qui ne sera plus jamais utilisé et qui existe déjà dans un ancien backup. Pour supprimer cette table, il suffit d'effectuer la requête suivante :

```
DROP TABLE client_2009
```

L'exécution de cette requête va permettre de supprimer la table.

Les bases de données : SQL DATABASE

7.] Les principales différences entre MyISAM et InnoDB

On se pose souvent la question sur l'utilisation de MyISAM ou d'InnoDB pour une base de données MySQL. Cet article décrit rapidement les principaux avantages et inconvénients de ces deux moteurs de stockage.

MyISAM

Les avantages :

- très rapide pour les requêtes de type SELECT ou INSERT
- il supporte les index fulltext : permet d'effectuer des recherches sur des mots en se basant sur un index spécifique, accélérant ainsi les recherches
- il gère les conflits d'accès (ou lock) : permet de verrouiller une table pour des opérations bien précises
- très facile à administrer : possibilité de recopier directement les fichiers d'un serveur vers un autre, support des tables compressées ...

Les inconvénients :

- il ne supporte pas les transactions
- il ne supporte pas les clés étrangères

Note : en plus du fichier .frm, chaque table est représentée par un fichier de données .myd (MYisamData) et un fichier d'index .myi (MYisamIndex).

InnoDB

Les avantages :

- il supporte ACID : permet d'assurer que chaque enregistrement sera complètement réussi, ou complètement échoué, ainsi les risques d'erreurs sont impossibles, même en cas de panne
- il gère les transactions (instructions sql BEGIN, COMMIT, ROLLBACK ...)
- il supporte les clés étrangères et les intégrités référentielles
- il possède un système de récupération automatique en cas de crash

Les inconvénients :

- il ne permet pas les index fulltext
- son administration est un peu plus complexe (gestion de tablespace, paramètres supplémentaires dans le my.cnf ...)
- le moteur de stockage est plus lent que d'autres et gourmand en ressources mémoires et en espace disque

Note : chaque table est représentée par un fichier .frm.

Les bases de données : SQL DATABASE

Changement du moteur de stockage

On gardera à l'esprit que le choix du moteur n'est jamais définitif. Il peut être changé dynamiquement avec la commande ALTER TABLE comme ceci :

```
ALTER TABLE maTable1 ENGINE=INNODB;
ALTER TABLE maTable2 ENGINE=MYISAM;
```

Ce système permet notamment d'utiliser plusieurs types de stockage pour différentes tables d'une même base de données. Cela permettra dans certains cas d'obtenir les meilleurs performances et avantages.

Conclusion

Pour conclure, on optera pour InnoDB principalement lorsque l'on utilisera un système d'information qui n'admet pas les erreurs ou qui doit utiliser des clés étrangères ou des intégrités référentielles.

MyISAM restera quant à lui le meilleur choix dans le cas où l'on fait principalement des requêtes de lecture ou d'insertion.

Les bases de données : SQL DATABASE

8.] LES CLES ETRANGERES

Lorsqu'une table possède une clé étrangère (qui est clé primaire d'une autre table), voici la syntaxe qui doit être écrite pour faire la relation entre les 2 tables :

```
alter table table_1 add constraint FK_Table_1_Table_2 foreign key (id_table_2)
      references table_2 (id_table_2);

Exemple :
alter table utilisateur add constraint FK_Client_Fonction foreign key (id_fonction)
      references fonction (id_fonction);
```

- `FK_Table_1_Table_2` : nom de la relation

Exemple de script complet de création d'une base de donnée (après avoir réalisé l'instruction CREATE DATABASE nom_db;) :

```
/*==============================================================*/
/* DBMS name:      MySQL 5.0                                    */
/* Created on:     28/06/2016 11:11:11                          */
/*==============================================================*/

drop table if exists utilisateur;

drop table if exists fonction;

/*==============================================================*/
/* Table: utilisateur                                           */
/*==============================================================*/
CREATE TABLE utilisateur (
            id_utilisateur          INT NOT NULL auto_increment,
            nom_utilisateur         VARCHAR(100),
            prenom_utilisateur      VARCHAR(100),
            email                   VARCHAR(255),
            date_naissance          DATE,
            pays                    VARCHAR(255),
            ville                   VARCHAR(255),
            code_postal             VARCHAR(5),
            nombre_achat            INT,
            PRIMARY KEY (id_utilisateur)
) ENGINE = MYISAM;

/*==============================================================*/
/* Table: fonction                                              */
/*==============================================================*/
create table fonction
(
            id_fonction             INT NOT NULL auto_increment,
            nom_fonction            VARCHAR(100),
            primary key (id_fonction)
); ENGINE = MYISAM;

alter table utilisateur add constraint FK_Client_Fonction foreign key (id_fonction) references fonction (id_fonction);
```