

Chap. 1 : SQL - création d'une base de données

L'objectif de ce cours est de (re)voir les possibilités offertes par le langage SQL pour créer une base de données.

Introduction : les SGBDR

Définition d'un SGBDR

Depuis leur apparition au début des années 1980, les SGBDR constituent le standard incontesté pour le stockage d'informations.

Comment définir un SGBD ?

Un Système de Gestion de Bases de Données est un logiciel qui gère une ou plusieurs base de données.

Une base de données assure le **stockage persistant** de données informatisées. Elle est implantée physiquement sous la forme d'un ou plusieurs fichiers stockés sur le disque dur d'un ordinateur. Un **SGBD** assure structuration, stockage, maintenance, mise à jour et recherche des données dans une base. Il fournit également des interfaces pour différentes formes d'utilisation de la base. On distingue plusieurs types de SGBD selon la technologie de stockage employée.

Qu'est-ce qu'un SGBDR ?

Un SGBD qui utilise les principes du modèle relationnel.

Historique des SGBDR

Les bases de données relationnelles sont nées de travaux effectués dans les laboratoires de la société IBM dans les années 1970. La théorie fondamentale des SGBDR repose sur le **modèle relationnel** initialement décrit par F. Codd en 1970. Il s'agit d'une manière d'organiser les informations sous forme de **relations**, qui se traduisent concrètement par des **tables**.

Les premiers SGBDR commerciaux sont apparus au début des années 1980. Il s'agit de logiciels bien connus comme **ORACLE**, **DB2** (IBM) ou encore Sybase, racheté en 1994 par Microsoft pour devenir **SQL Server**.

Ces logiciels professionnels ont été rejoints depuis les années 1990 par des logiciels *open source* reconnus comme **MySQL** ou **PostgreSQL**.

Avenir des SGBDR

Pas besoin d'une boule de cristal pour prévoir que les SGBDR ne sont pas prêts de perdre leur position de standard dans le domaine du stockage de données :

- La théorie sous-jacente (le modèle relationnel) est simple et rigoureuse.
- Les SGBDR ont depuis longtemps fait la preuve de leurs qualités (performances, robustesss, sécurité).
- Ils sont particulièrement adaptés à la création d'applications de type client/serveur ou à plusieurs couches (n -tiers).

Apparues dans les années 1990, les SGBD orientés objet n'ont pas rencontré le succès. Pour sans doute longtemps encore, le monde objet (standard pour l'écriture de programmes) et le monde relationnel (standard pour le stockage des données) devront cohabiter.

Le langage SQL

Présentation de SQL

Comme vous le savez déjà, SQL signifie **Structured Query Language**. En première approche, on peut donc le définir comme un langage de requête structuré. Le langage SQL constitue une norme universelle pour l'accès aux données d'un SGBDR. Il peut s'interfacer avec de nombreux langages informatiques (Java, C#, PHP...), facilitant l'écriture d'applications qui accèdent aux données d'un SGBDR.

Historique du langage

SQL est apparu en même temps que les premiers SGBDR. Pour la petite histoire, F. Codd, le père de la théorie relationnelle, voulait créer un système où l'interrogation des données utilisait le vocable anglais. On retrouve des traces de cette volonté dans les instructions du langage.

Appelé à l'origine SEQUEL (*Structured English Query Language*), il a été rebaptisé SQL puis normalisé pour la première fois en 1986. Cette version de SQL, souvent nommée **SQL1** ou **SQL86**, présentait :

- Des requêtes compilées puis exécutées depuis un programme d'application.
- Des types de données simples (entiers, réels, chaînes de caractère de taille fixe).
- Des opérations ensemblistes restreintes (union).

Trop restrictive, cette première norme a conduit de nombreux éditeurs de SGBDR à ajouter des extensions propriétaires au langage de requêtes. La seconde mouture de la norme SQL est **SQL92**, appelée parfois **SQL2** et normalisée (quelle surprise) en 1992. Elle apporte plusieurs évolutions majeures :

- Des requêtes dynamiques : exécution immédiate ou différée.
- Des types de données riches (intervalles, dates, chaînes de caractère de taille variable).
- Différents types de jointure (syntaxe JOIN).
- Des opérations ensemblistes plus nombreuses (différence, intersection, union).

Même si de nouvelles versions du langage (SQL:1999, SQL:2003) ont vu le jour depuis, SQL92 constitue toujours la version de la norme la plus importante et la plus utilisée.

Structure du langage

Le langage SQL se compose d'instructions, ou ordres SQL, que le SGBDR doit exécuter. Remarquons qu'il s'agit d'ordres **logiques** : l'utilisateur ou l'application spécifie ce qu'il veut faire (exemple : ajouter un nouvel enregistrement dans une table) et c'est le SGBDR qui décide comment effectuer cette opération (exemple : ajouter un élément dans son fichier de stockage interne). SQL est totalement indépendant de la manière dont les données sont physiquement organisées dans le SGBDR.

Les principales possibilités du langage SQL concernent :

- La **création** des éléments de la base (tables, index, etc).
- La **recherche** d'informations dans la base.
- La **mise à jour** des données de la base (insertion, mise à jour, suppression).
- Le **contrôle** sur les données de la base (gestion des privilèges).

Création des éléments de la base avec SQL

Préambule

Du modèle relationnel au schéma physique

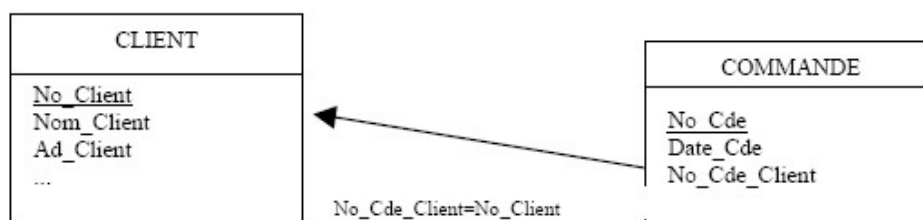
Le modèle de données que SQL manipule est constitué de **tables** correspondant aux **relations** décrites dans le modèle relationnel. Une table est composée de **colonnes** (également appelées **champs**) et contient des **lignes** (également appelées **tuples** ou **enregistrements**).

Même si ce n'est pas obligatoire en pratique, il est très fortement conseillé de définir une **clé primaire** (*primary key* ou PK) pour chaque table.

Qu'est-ce qu'une clé primaire ?

C'est l'ensemble minimal de colonnes qui permet d'identifier de manière unique chaque ligne.

Dans le modèle relationnel, les associations entre les tables sont assurées par des clés **étrangères** (*foreign key* ou FK) qui réfèrent des clés **primaires** d'autres tables.



La présence d'une clé étrangère dans une table interdit l'utilisation de références vers des

données inexistantes. Ce mécanisme est appelé **intégrité référentielle**.

Quel est son intérêt ?

Renforcer la cohérence globale des données de la base.

Choix des types de données

Une fois le schéma physique conçu, il reste à choisir sous quels types stocker les données de la base. Tout comme les langages de programmation de haut niveau (C++, Java, C#), SQL est un langage **fortement typé**. Cela signifie qu'il propose de nombreuses possibilités (différents types de données) pour définir au mieux la façon dont les valeurs sont être stockées dans la base.

Quel est l'intérêt du typage fort ?

S'assurer que les types de données employés décrivent correctement les informations manipulées.

Le tableau ci-dessous présente la liste des principaux types de données.

Type	Description
CHAR(n)	Chaîne de caractères de longueur fixe <i>n</i>
VARCHAR (n)	Chaîne de caractères de n caractères maximum
SMALLINT	Entier stocké (en principe) sur 16 bits
INTEGER	Entier stocké (en principe) sur 32 bits
FLOAT	Nombre réel à virgule flottante (arrondi)
DECIMAL	Nombre réel stocké sous sa forme exacte (pas d'arrondi)
BOOLEAN	Booléen (valeurs TRUE ou FALSE)
DATE	Date sous la forme aaaa-mm-jj
TIME	Heure sous la forme hh:mm:ss
TIMESTAMP	Type horaire Date et Heure

Remarque : en plus de ces types standard, les éditeurs de SGBDR proposent souvent des types spécifiques propres à leur logiciel.

Connexion au SGBDR et authentification

Avant de pouvoir créer une nouvelle base, il faut au préalable se connecter au SGBDR qui l'héberge. Cela implique de s'identifier en tant qu'utilisateur du SGBDR et de disposer des droits adéquats. La procédure de connexion est spécifique au SGBDR utilisé.

Ne pas confondre un utilisateur d'OS (Windows, Linux, etc) et un utilisateur du SGBDR. L'utilisateur SGBDR est plus proche de la notion de **rôle**, et plusieurs utilisateurs d'OS, ou même



plusieurs applications (un site Web, un client graphique, etc) peuvent se connecter à un SGBDR sous le même utilisateur.

Création de la base proprement dite

A l'exception des plus simples (Access), un SGBDR est capable de gérer plusieurs bases de données. La première étape est donc bien souvent de créer la base elle-même, avant d'y ajouter des tables. Il n'existe pas d'ordre SQL standardisé pour créer une base. Cependant, la plupart des éditeurs se sont mis d'accord pour accepter l'ordre **CREATE DATABASE**.

Exemple : **CREATE DATABASE** LYCEE;

L'instruction ci-dessus crée une nouvelle base nommée **LYCEE**.

Création d'une table

Syntaxe

La création d'une table dans une base de données existante s'effectue grâce à l'instruction **CREATE TABLE**.

La syntaxe de cette instruction est :

```
CREATE TABLE <nom de la table> (  
    <nom du champ 1>    <type>    <attributs>,  
    <nom du champ 2>    <type>    <attributs>,  
    ...  
);
```

Remarques :

- ✓ Les ordres SQL sont **insensibles** à la casse.
- ✓ Le mot entre <> signifie qu'il faut remplacer ce mot par une valeur (sans les <>).

Exemple :

```
CREATE TABLE ETUDIANT (  
    ID                INTEGER,  
    NOM                VARCHAR(50),  
    PRENOM            VARCHAR(50),  
    DATE_NAISSANCE    DATE  
);
```

Que fait l'instruction ci-dessus ?

Elle crée une table nommée ETUDIANT contenant 4 champs : ID, NOM, PRENOM et DATE_NAISSANCE.

Contraintes de table

Dans un **CREATE TABLE**, il est possible de préciser des contraintes qui restreignent les conditions d'acceptation des données de la table. Les différentes contraintes possibles sont rassemblées dans le tableau suivant.

Nom	Définition
NOT NULL	Obligation de valeur
DEFAULT	Valeur par défaut
CHECK	Validation de valeur
UNIQUE	Unicité de valeur
PRIMARY KEY	Clé primaire
FOREIGN KEY	Clé étrangère

Contraintes sur les valeurs des colonnes

Exemple :

```
CREATE TABLE ETUDIANT (  
  ID                INTEGER          NOT NULL,  
  NOM               VARCHAR(100)     NOT NULL,  
  PRENOM            VARCHAR(100)     NOT NULL,  
  NUM_SECU          CHAR(15)         NOT NULL UNIQUE,  
  SEXE              CHAR(1)          NOT NULL,  
  DATE_NAISSANCE    DATE,  
  REDOUBLANT        BOOLEAN          DEFAULT FALSE,  
  CONSTRAINT CK_ETU_SEX CHECK (SEXE IN ('M', 'F'))  
);
```

Voici la description des attributs utilisés :

- **NOT NULL** signifie que le champ doit toujours avoir une valeur.
- **UNIQUE** signifie que la valeur du champ, si elle existe, doit être unique sur toute la table.
- **DEFAULT** donne la valeur par défaut du champ en l'absence de valeur explicite.
- **CHECK** définit une restriction sur les valeurs autorisées.



NULL marque l'**absence de valeur**. Ce n'est donc ni un zéro (qui est une valeur), ni la chaîne vide.

Remarques :

- ✓ Les contraintes de table peuvent être nommées. Ici la contrainte **CHECK** sera nommée **CK_ETU_SEX**. En cas d'absence de nom explicite, le SGBDR générera automatiquement un nom pour la contrainte.
- ✓ Nommer les contraintes est souvent une bonne pratique, car en cas de violation de contrainte, son nom est souvent rapportée dans le message d'erreur.
- ✓ Certains SGBDR ignorent une partie des mots-clés du standard SQL. Par exemple, MySQL n'implémente actuellement pas la contrainte **CHECK**.

Contraintes de clés

Les tables que nous avons créées jusqu'ici ne définissaient pas de clés primaires, ce qui est fortement déconseillé. Le SQL offre la possibilité de désigner un (ou plusieurs) champs comme clé primaire au moment de la création des tables.

Exemple :

```
CREATE TABLE ETUDIANT (  
    ID                INTEGER                NOT NULL PRIMARY KEY,  
    NOM               VARCHAR(100)          NOT NULL,  
    PRENOM            VARCHAR(100)          NOT NULL,  
    NUM_SECU          CHAR(15)              NOT NULL UNIQUE,  
    SEXE              CHAR(1)               NOT NULL,  
    DATE_NAISSANCE    DATE,  
    REDOUBLANT        BOOLEAN               DEFAULT FALSE,  
    CHECK (SEXE IN ('M', 'F'))  
);  
CREATE TABLE NOTE (  
    ID                INTEGER                NOT NULL,  
    VALEUR            FLOAT                 NOT NULL,  
    DATE_DEVOIR       DATE                 NULL,  
    ID_ETU            INTEGER               NOT NULL,  
    CONSTRAINT PK_NOTE PRIMARY KEY (ID),  
    CONSTRAINT FK_NOTE_ETU FOREIGN KEY (ID_ETU) REFERENCES  
        ETUDIANT(ID)  
);
```

Dans l'exemple précédent, la clé étrangère **ID_ETU** de la table **NOTE** fait référence à la clé primaire **ID** de la table **ETUDIANT**.

On voit qu'il existe deux manières de déclarer une clé primaire dans une instruction **CREATE** :

- soit directement à la suite de la déclaration du champ (ici le champ ID de ETUDIANT),
- soit après la déclaration des champs, grâce à l'instruction **CONSTRAINT**.

Remarque :

- ✓ Les noms des colonnes liées par une intégrité référentielle (ici **ID_ETU** dans NOTE et **ID** dans ETUDIANT) peuvent être différents, même si elles ont sémantiquement la même signification. Leur donner le même nom dans les deux tables permet de pouvoir utiliser la syntaxe de la jointure **naturelle**.
- ✓ Afin de faciliter la génération des valeurs de la clé primaire, de nombreux SGBDR permettent de déclarer un champ *auto-incrémenté* à l'aide d'un attribut lors de la création de la table. A chaque insertion de ligne, la valeur du champ est fixée automatiquement. Il n'existe pas de norme SQL sur ce point (exemples de syntaxes : **AUTO_INCREMENT** sous MySQL, **IDENTITY** sous SQL Server).

En cas de présence d'une clé étrangère dans un CREATE TABLE, l'ordre de création des tables est-il important ?

Oui : il faut que la table contenant la clé primaire ait déjà été créée.

Application : au lycée, une classe est identifiée par son numéro (entier). Elle a comme

caractéristiques son intitulé (unique), son effectif et son pourcentage de redoublants (égal à 0 par défaut).

Créez la table correspondante.

```
CREATE TABLE CLASSE (  
  ID          INTEGER          NOT NULL,  
  INTITULE    VARCHAR(10)      NOT NULL UNIQUE,  
  EFFECTIF    SMALLINT(2)      NOT NULL,  
  PCT_RED     FLOAT            NULL DEFAULT 0,  
  CONSTRAINT PK_CLA PRIMARY KEY (ID)  
);
```

Création d'une vue

En SQL, on peut définir une **vue** comme étant un ordre d'extraction de données (**SELECT**) dont le résultat est vu comme une table.

La syntaxe de cette instruction est :

```
CREATE VIEW <nom de la vue> AS  
  <ordre de sélection>;
```

Exemple :

```
CREATE VIEW ETU_RED AS  
  SELECT ID, NOM, PRENOM, NUM_SECU, SEXE, DATE_NAISSANCE  
  FROM ETUDIANT  
  WHERE REDOUBLANT = TRUE;
```

Que fait la requête-ci-dessus ?

Elle crée une vue qui n'affiche que les étudiants redoublants.

Les vues sont très utiles pour masquer la complexité d'un schéma relationnel. Elles permettent également d'améliorer la sécurité de la base, en ne montrant à chaque utilisateur que la partie nécessaires des données stockées.

Remarque : il est possible de définir une vue affichant les données de plusieurs tables en utilisant des **jointures** dans l'ordre de sélection.

Création d'un index

Les index sont des éléments essentiels pour optimiser les performances en lecture dans une base de données. Un index se définit sur une colonne et permet d'accélérer les recherches sur cette colonne. Son implémentation concrète dépend du SGBDR.

Il existe plusieurs types d'index basés sur des algorithmes plus ou moins complexes (séquentiel, B-Tree, hachage, etc). A proprement parler, les index ne font pas partie du standard SQL. Cependant, la plupart des SGBDR acceptent un ordre **CREATE INDEX** qui permet de le créer.

Exemple :

```
CREATE INDEX I_ETU_NOM ON ETUDIANT (NOM);
```

Lors de la création d'une base, il faut donc réfléchir aux données sur lesquelles porteront le plus souvent les recherches, et créer les index appropriés.

Remarque : beaucoup de SGBDR créent automatiquement un index sur la clé primaire d'une table.

Modification de la structure d'une table

Le langage SQL offre (heureusement) la possibilité de changer la structure d'une table déjà créée. L'ordre SQL associé s'appelle **ALTER TABLE**. Il offre de nombreuses possibilités.

Exemple 1 :

```
ALTER TABLE ETUDIANT ADD ID_CLA INTEGER;
```

Que fait la requête précédente ?

Elle ajoute à la table ETUDIANT un champ entier nommé ID_CLA.

Exemple 2 :

```
ALTER TABLE ETUDIANT ADD CONSTRAINT FK_ETU_CLA FOREIGN KEY  
(ID_CLA) REFERENCES CLASSE(ID);
```

Que fait la requête précédente ?

Elle définit le champ ID_CLA comme clé étrangère de la table ETUDIANT.

Exemple 3 :

```
ALTER TABLE ETUDIANT DROP ID_CLA;
```

La requête précédente **supprime** le champ ID_CLA;

Ecriture d'un script SQL de création

Une base de données contient le plus souvent plusieurs tables reliées entre elles par des contraintes d'intégrité. Pour faciliter sa création, on rassemble souvent toutes les instructions **CREATE TABLE** et **ALTER TABLE** dans un script (fichier texte au format SQL). Exécuter le script depuis l'interface d'un SGBDR permet d'automatiser la génération des tables, contraintes et clés.

Cependant, il faut être vigilant si des relations d'intégrité sont présentes, ce qui est presque toujours le cas. Deux solutions :

1. Créer toutes les tables (instructions **CREATE TABLE**), puis ajouter ensuite les contraintes de clés étrangères (instructions **ALTER TABLE**).

2. Créer au fur et à mesure les tables et les contraintes, en créant les tables où sont présentes des clés primaires **avant** celles où sont présentes des clés étrangères associées.

Suppression d'une table

La suppression d'une table se fait grâce à l'instruction **DROP TABLE**. Elle supprime de la base une table et tout son contenu.

Exemple :

DROP TABLE NOTE;



Cette instruction doit être **manipulée avec précaution**.