CHAPITRE 4 : Langage SQL

PARTIE 1: GERER LA STRUCTURE DES TABLES

I. Introduction

Les ordres **DDL** (*anglais : **D**ata **D**efinition **L**anguage*, en français LDD, langage de Définition de Données) permettent la **gestion des objets d'une base de données** (tables, index, trigger, vues, etc.) : ils ne touchent qu'à la **STRUCTURE D'ACCUEIL DES DONNEES**, pas au contenu[1].

A. Choix du SGBD

Le choix, ou la contrainte imposée d'un SGBD, va définir les types de données à utiliser en lieu et place des domaines des attributs.

B. Règles de passage du Modèle Relationnel au modèle physique

Chaque relation devient une table :

- les attributs deviennent des colonnes, chacune se voyant attribuer un type de donnée le plus proche possible du domaine de l'attribut correspondant
- un attribut clef primaire devient une contrainte d'intégrité d'identité à définir (primary key)
- un attribut clef étrangère devient une contrainte de clé étrangère (foreign key)
- un attribut clef candidate devient un index

C. Représentation du Modèle Physique de Données

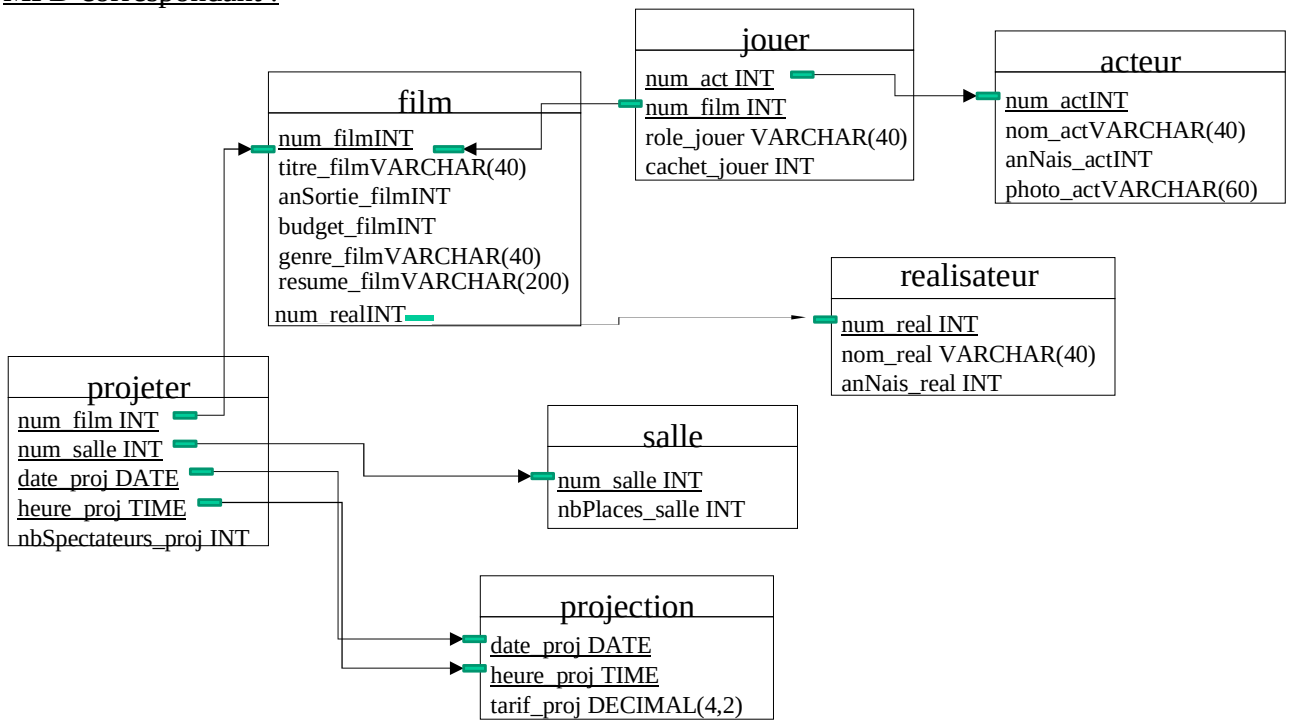
La représentation graphique du MPD très utile pour avoir une représentation de l'ensemble des tables d'une base de données et des liens, exprimés sous forme de clefs étrangères, qui existent entre ces tables. Les types de données compléteront la représentation des tables.

MLD exemple :
acteur (num_act : entier, nom_act : chaine(40), anNais_act : entier)
num_act : clef primaire
realisateur (num_real : entier, nom_real : chaine(40), anNais_real : entier)
num_real : clef primaire
film (num_film : entier, titre_film : chaine(40), anSortie_film : entire, budget_film : entire, genre_film : chaine(40), num_real : entier) num_film : clef primaire
num_real : clef étrangère vers la table realisateur, colonne num_real
jouer (num_act : entire, num_film : entier, role_jouer : chaine(40), cachet_jouer : entier)
num_act, num_film : clef primaire
num_act : clef étrangère vers la table acteur, colonne num_act
num_film : clef étrangère vers la table film, colonne num_film

[1] Attention quand même : quand on supprime une table, son contenu disparait également…

salle (num_salle : entier, nbPlaces_salle : entier)
num_salle : clef primaire
projection (date_proj : date, heure_proj : heure)
date_proj, heure_proj : clef primaire
projeter (num_film : entier, num_salle : entier, date_proj : date, heure_proj : heure, nbSpectateurs_proj)
num_film, num_salle, date_proj, heure_proj : clef primaire
num_salle : clef étrangère vers la table salle, colonne num_salle num_film : clef étrangère vers la table film, colonne num_film
date_proj, heure_proj : clef étrangère vers la table projection, colonnes date_proj, heure_proj

MPD correspondant :



Une fois le MPD réalisé, l'écriture des scripts SQL de création de la base et des tables pourra être effectuée.

II. Créer une table : CREATE TABLE

Créer une table, c'est :
_ Définir la structure d'accueil de données : les colonnes et leur type de données
_ Définir des règles de vérifications de ces données : les contraintes

```
CREATE TABLE nomTable (  nomColonne
typeColonne [contrainteColonne]
[,nomColonne1 typeColonne1 [contrainteColonne1]...]
    [,contrainteTable]
    [,contraintePK]
    [,contrainteFK [,contrainteFK1 ...] ]
);
```

Par exemple, création d'une table 'pays', avec 2 colonnes dont une obligatoire, contrainte de clef primaire (colonne identifiant chaque ligne de manière unique) pour la colonne codePays :

```
CREATE   TABLE   pays   (
codePays   INT   NOT   NULL,
libPAYS  VARCHAR(20),
    CONSTRAINT pk_pays PRIMARY KEY (codePays)
);
```

A. Les noms de tables et colonnes (nomTable, nomColonne)

Les caractères acceptés pour les noms de tables et colonnes sont les caractères alphanumériques (lettres et chiffres), et le séparateur '_' (tiret bas).

Certaines bases de données acceptent le caractère ' ' (espace) : le nom doit alors être entouré de caractères '' (quote) ou [ ] (crochets) afin de délimiter le nom.

Conseil : *utilisez les caractères : [a .. z] ou [A .. Z], [0 .. 9], [_] pour coder les noms de tables et colonnes. Vous limiterez ainsi les problèmes de portage des scripts d'une base de données vers une autre.*

B. Les types de données et les domaines

Les **types de données** représentent le **premier niveau de contrainte d'intégrité** des données stockées. C'est pourquoi l'affectation à une colonne d'un type de données et d'une taille est de première importance.

On peut définir à 5 le nombre de familles de types de données pour SQL (norme SQL92 ou pour certains non normalisés) :

- **CHAINES DE CARACTERES :**
  - codage ASCII ou EBCDIC (1 octet par caractère codé)
    - CHARACTER(nn) ou **CHAR(nn)**, suivi de la longueur entre parenthèse : longueur fixe (comblée à droite par des espaces) ;
    - CHARACTER VARYING ou **VARCHAR(nn)**, suivi de la longueur maximale entre parenthèse : longueur variable (1 octet en plus) ;
  - codage UNICODE (2 octets par caractère codé)
    - NATIONAL CHARACTER ou **NCHAR(nn)**, suivi de la longueur entre parenthèse : longueur fixe (comblée à droite par des espaces)
    - NATIONAL CHARACTER VARYING ou **NVARCHAR(nn)**, suivi de la longueur maximale entre parenthèse : longueur variable (1 octet en plus pour stocker la longueur de la chaîne)
- **NOMBRES :**
  - entiers :

INTEGER ou **INT** : nombres entiers sur 32 bits (en général), de – 2.147.483.648, soit -(2^{31}) à +2.147.483.648, soit (2^{31} – 1)

SMALLINT : nombres entiers sur 16 bits, de –32.768, soit -(2^{15}) à +32.768, soit (2^{15} – 1)

- Les nombres réels :

DECIMAL, DEC, **NUMERIC**, MONEY, CURRENCY, NUMBER : nombre décimal exact, avec la précision du nombre de chiffres du nombre et du nombre de chiffres de la partie décimale : NUMERIC(10,4) : le nombre aura 10 chiffres significatifs dont 4 après la virgule ;

REAL, FLOAT, DOUBLE PRECISION : **nombre décimal approché** (la conversion en binaire fait perdre une certaine précision[2]) ; ce type ne devrait jamais être utilisé pour des montants ou des quantités précises !

- **DATES ET HEURES :**
  - Dates et heures sont stockées sous une forme spécifique à chaque SGBD, et présentée à l'utilisateur sous plusieurs formes

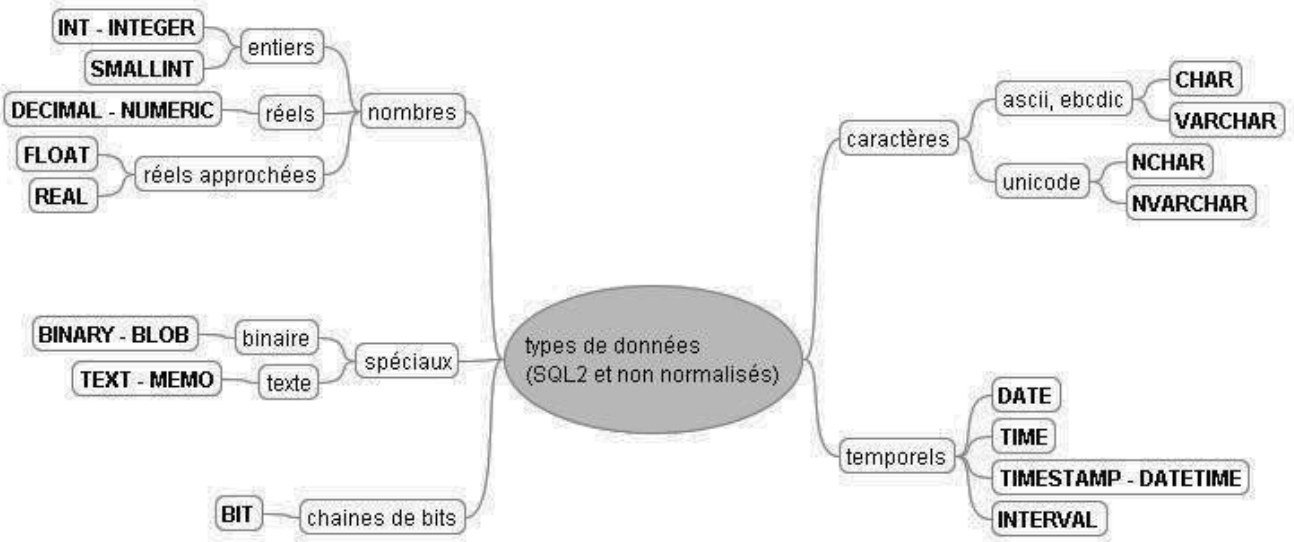
DATE : stockage de la date seule, 'AAAA-MM-JJ' ou 'JJ/MM/AAAA'

TIME : stockage de l'heure seule, 'HH :MN :SS.mmm'

DATETIME / TIMESTAMP : stockage de la date et de l'heure sous forme : JJ/MM/AAAA HH:MN:SS.mmm

INTERVAL : stockage d'un intervalle de dates : nombre de jours, de mois, d'année,etc.

- **CHAINES DE BITS :**
  - **BIT** : permet le stockage d'un certain nombre de bits
- **OBJETS VOLUMINEUX :**
  - **BLOB** : Binary Large OBject : permet le stockage d'images, par exemple
  - **TEXT** : permet le stockage de textes volumineux



[2] Vous pouvez vous référer au cours sur la représentation des nombres en virgule flottante (IEEE 754) pour comprendre cette perte de précision lors de la conversion en binaire

Les **domaines** (disponibles dans certains SGBD) permettent de définir de nouveaux types à partir des types de bases, en y adjoignant des contraintes par exemple ; cela permet de réutiliser cette définition pour des colonnes de plusieurs tables (= cohérence de définition).
Cf Créer un domaine de valeurs

C. Les CONTRAINTES de colonnes, ou contraintes verticales

Ces contraintes portent sur la valeur d'une seule colonne :

(1) NOT NULL : donnée obligatoire

Rend obligatoire la présence d'une donnée dans cette colonne

```
CREATE TABLE pays (
codePays INT NOT NULL,
libPAYS VARCHAR(20) );
```

(2) DEFAULT : valeur par défaut

Permet de spécifier une valeur par défaut au cas où aucune valeur n'est fournie pour cette colonne

```
CREATE TABLE pays ( codePays INT NOT
NULL, libPAYS VARCHAR(20),
codeDevise CHAR(4) DEFAULT "euro" );
```

(3) UNIQUE : valeur dans pour la colonne

Permet le contrôle de l'unicité de valeur de cette colonne

```
CREATE TABLE utilisateur (
codeUtil INT, nomUtil
VARCHAR(20), adrEmail
VARCHAR(20) UNIQUE );
```

(4) CHECK : validité du contenu

Permet la validité de la valeur de cette colonne

```
CREATE TABLE chiffreDAff ( jourSem INT CHECK
(joursem BETWEEN 1 AND 7), montant DECIMAL(10,2)
);
```

(5) PRIMARY KEY : Clef primaire

Intégrité de table (ou d'entité) : toute table doit posséder une colonne clef primaire dont la valeur doit être unique.

Contrainte de clef primaire : valeur unique permettant de retrouver une ligne dans une table

```
CREATE TABLE utilisateur ( codeUtil INT NOT NULL
PRIMARY KEY, nomUtil VARCHAR(20), adresseEmail
VARCHAR(20) UNIQUE
);
```

(6) [FOREIGN KEY] REFERENCES : clef étrangère

L'intégrité référentielle : à une valeur d'une colonne d'une table doit correspondre, dans une autre table, à une valeur existante en tant que clef primaire.

Syntaxe :

```
REFERENCES nomTableRef ((colonneTableRef[,…])
[ON UPDATE [CASCADE | SET NULL] ]
```

```
[ON DELETE [CASCADE | SET NULL] ]
[ [NOT] DEFERRABLE]
```

- ON UPDATE : action à prendre en cas de modification de la valeur dans la colonne référencée : modification de la valeur de la ligne ou valeur positionnée à NULL (sans cette clause, la modification est refusée)
- ON DELETE : action à prendre en cas de suppression de la valeur dans la colonne reference : suppression de la ligne, valeur positionnée à NULL (sans cette clause, la suppression est refusée)
- DEFERRABLE : permet de différer la vérification de l'intégrité référentielle à la fin d'une transaction (et non pas immédiatement), et ainsi permettre les dépendances mutuelles (cf. "SET CONSTRAINT nomContrainte DEFERRED" pour le temps d'une transaction.
-

Contrainte de clef étrangère : cette valeur, si elle est renseignée, permet de vérifier l'existence d'une clef primaire dans une autre table (ou la même table si association réflexive)

```
CREATE TABLE utilisateur ( codeUtil INT
NOT NULL PRIMARY KEY, nomUtil
VARCHAR(20), adresseEmail
VARCHAR(20)UNIQUE,
        codePays INT REFERENCES pays (codePays) ON DELETE SET
NULL );
```

D. Les CONTRAINTES de tables

Elles sont spécifiées après la définition des colonnes et elles concernent, généralement, plusieurs colonnes de la table. Ces contraintes sont de type : UNIQUE, CHECK, PRIMARY KEY et FOREIGN KEY.

Syntaxe générale :

```
CONSTRAINT nomContrainte typeContrainte contrainte
```

```
CREATE TABLE produit (
 CodeProd INTEGER NOT NULL, versionProd
INTEGER NOT NULL, designation
VARCHAR(40) NOT NULL, codeEAN1
VARCHAR(10) NOT NULL, codeEAN2
VARCHAR(10) NOT NULL, stock
DECIMAL(10,2), stockMini
DECIMAL(10,2), codeFamille INTEGER,
codeSerie INTEGER,
    CONSTRAINT CK_verif_stock CHECK (stock >= stockMini),
    CONSTRAINT UN_ean_unique UNIQUE (codeEAN1, codeEAN2),
    CONSTRAINT PK_produit PRIMARY KEY (codeProd,
versionProd) ,
 CONSTRAINT FK_produit_famille FOREIGN KEY (codeFamille,
codeSerie) REFERENCES famille (codeFamille, codeSerie) );
```

III. Modifier la structure d'une table ou les contraintes associées à une table – ALTER TABLE

```
ALTER TABLE nomTable
     { ADD CONSTRAINT defContrainte
     | DROP [CONSTRAINT] nomContrainte
     | DROP PRIMARY KEY
     | ADD [COLUMN] defColonne
     | DROP [COLUMN] nomColonne
     | ALTER [COLUMN] {SET DEFAULT … | DROP DEFAULT }
);
```

Exemple pour la table « produit »

```
CREATE TABLE produit (  CodeProd INTEGER NOT NULL,      design      VARCHAR(20),      stock DECIMAL(10,2),  codeFamille INTEGER
);
```

Ajouter une colonne :

```
ALTER TABLE produit
     ADD COLUMN stockMini DECIMAL(10,2);
```

Ajouter une contrainte sur des colonnes de la table (ici des contraintes de vérification de cohérence de valeurs sur 2 colonnes de la table):

```
ALTER TABLE produit
     ADD CONSTRAINT CK_verif_stock CHECK (stock >= stockMini);
```

Supprimer la contrainte (les colonnes sont conservées, leurs données également mais elles ne seront plus vérifiées) :

```
ALTER TABLE produit
     DROP CONSTRAINT CK_verif_stock;
```

Supprimer la colonne : **ATTENTION : LES DONNEES SONT PERDUES**

```
ALTER TABLE produit
     DROP COLUMN stockMini;
```

Ajouter une contrainte de clef primaire :

```
ALTER TABLE produit
     ADD CONSTRAINT PK_produit PRIMARY KEY (codeProd);
```

Ajouter une contrainte de clef étrangère :

```
ALTER TABLE produit
     ADD  CONSTRAINT  FK_famille  FOREIGN  KEY  (codeFamille) REFERENCES famille (codeFamille) ;
```

Supprimer une contrainte de clef étrangère :

```
ALTER TABLE produit
     DROP FK_famille ;
```

IV. Supprimer une table – DROP TABLE

ATTENTION : LA SUPPRESSION D'UNE TABLE EFFACE DE MANIERE DEFINITIVE LES DONNEES QUI Y ETAIENT STOCKEES.

La table qui doit être supprimée ne doit pas être la cible d'une contrainte de clef étrangère (un message d'erreur sera affiché dans ce cas).

```
DROP TABLE nomTable [ RESTRICT | CASCADE ];
```

L'option 'RESTRICT', par défaut, empêche la suppression si des objets sont liés à cette table. L'option 'CASCADE' permet de forcer la suppression de la table et des objets qui y en dépendent (les contraintes, par exemple).

V. Créer un domaine de valeurs

Un domaine permet de définir un type de données associé à ses contraintes en vue d'une utilisation dans plusieurs tables. Le domaine ainsi crée peut être ensuite utilisé pour définir des colonnes.

```
CREATE DOMAIN nomDomaine AS typeDonnées
[ DEFAULT valeur_par_defaut ]
     [ CONSTRAINT contrainte_de_domaine ]
     [ COLLATE jeu_de_caracteres ]
);
```

Par exemple (sous PostgreSQL) :

```
CREATE DOMAIN jourSem AS INTEGER
     CONSTRAINT CK_verif_jour CHECK ( value BETWEEN 1 AND 7 )
;
```

Puis utilisation dans la création d'une table :

```
CREATE TABLE ca (  jourSemain jourSem NOT
NULL,     montantCA NUMERIC(12,2)
);
```

VI. Créer une séquence de clef auto incrémentée

Cette possibilité offerte par les SGBD, sous différentes formes, permet d'attribuer à une colonne clef primaire ayant un type de données numérique, une valeur croissante (un compteur).
Cela est géré par l'affectation d'un compteur à la table, celui-ci devant être déclaré explicitement ou non.

Par exemple, le mot-clef AUTO-INCREMENT sous MySQL, IDENTITY sous MS-SQL SERVER permet de qualifier une colonne comme étant incrémentée automatiquement.

Par exemple ici, pour le SGBD MySQL :

```
CREATE TABLE produit(
 CodeProd INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  design
VARCHAR(40),  stock DECIMAL(10,2),  codeFamille INTEGER
);
```

D'autres SGBD nécessitent la création préalable d'un compteur (exemple PostgreSQL) :

Création d'une séquence commençant à 1 et incrémentée de 1 à chaque nouvelle ligne (une forme de syntaxe)

```
CREATE SEQUENCE produit_sequence
START WITH 1 INCREMENT BY 1;
```

Création d'une séquence commençant à 1 et incrémentée de 1 à chaque nouvelle ligne (une autre forme de syntaxe)

```
CREATE SEQUENCE produit_sequence
   INCREMENT 2 MINVALUE 1 MAXVALUE 1000;
```

Il faudra ensuite faire référence à cette séquence dans la création de la table concernée :

Par exemple ici, pour le SGBD PostgreSQL :

```
CREATE TABLE produit(
 CodeProd INTEGER DEFAULT nextval('produit_sequence') NOT NULL
PRIMARY KEY,   design   VARCHAR(40),   stock  DECIMAL(10,2),
codeFamille INTEGER
);
```

VII. Créer un index

Un index sert à accélérer les recherches portant sur une ou plusieurs colonnes d'une table fortement sollicitées dans le cadre d'interrogations

Un index est une table système référençant toutes les valeurs d'une colonne (une clef, un identifiant supplémentaire) et la ligne où elles se trouvent dans la table.

Par exemple, sur une table client où chaque ligne possède un numéro de client, il pourra être utile de créer un index sur le nom du client si l'accès aux lignes utilise souvent la valeur de cette colonne.

Un index est créé automatiquement pour chaque table où une clef primaire a été définie.

La syntaxe est spécifique à chaque SGBD, mais on trouvera par exemple :

```
CREATE [UNIQUE] INDEX nomIndex
ON nomTable (nomColonne, ,nomColonne1,...) ;
```

Le mot clef 'UNIQUE' précise qu'on n'autorisera pas de doublons dans cette colonne : cela peut être le cas d'un code INSEE ; par contre dans le cas d'un index sur le nom et le prénom, on pourra autoriser les doublons.