Fonctionnalités des SGBD relationnels

Guillaume Aubian

Université Paris Panthéon-Assas

SGBD relationnel : rappel

Un SGBD est dit relationnel s'il respecte une norme composée de trois conditions :

- Toutes les informations sont représentées par des valeurs contenues dans des relations (tables).
- Les jointures établissent les liens entre les relations.
- Le SGBD permet :
  - la sélection d'occurrences (SELECTION),
  - la sélection d'attributs (PROJECTION),
  - l'opération de JOINTURE.

SGBD relationnel : conditions

Un SGBD relationnel est **totalement relationnel** s'il respecte en outre deux conditions suivantes :

- Il permet d'utiliser les (autres) opérateurs de l'algèbre relationnelle (excepté la division).
- Il permet de prendre en compte la contrainte d'unicité des clés et l'intégrité référentielle.

ACCESS et MySQL sont totalement relationnels.

Base et métabase

- Une base de données comporte des relations, dont les descriptions sont elles-mêmes mémorisées dans une base de données appelée **métabase**.
- Si on crée une nouvelle relation, on agit sur la métabase ; si on crée une occurrence d'une relation, on agit sur la base.
- Le langage SQL permet non seulement d'interroger les bases de données, mais aussi, par d'autres instructions, d'agir sur la métabase et sur la base.

Gestion de la métabase : création de relations

- La plupart des SGBD permettent de créer les relations de manière interactive (Access, MySQL).
- La création d'une relation peut cependant se faire directement en SQL via `CREATE TABLE`.
- `CREATE TABLE` crée une relation, donc agit sur la métabase.

Gestion de la métabase : clés

- Lors de la création d'une relation, de nombreux SGBD imposent de préciser le(s) attribut(s) clé(s).
- Une relation peut avoir :
  - une **clé primaire** (un ou plusieurs attributs),
  - des **clés secondaires** (moyens alternatifs d'accès rapide).
- La clé primaire assure l'impossibilité de donner la même clé à deux occurrences différentes d'une même relation.

Exemple : clé primaire (1 attribut)

`Etudiant_eco(N° étudiant, Nom-étudiant, Année-étude)`

N° Étudiant	Nom-étudiant	Année-étude
843	Carole	1
521	Peter	3
864	Paulo	1
790	Carole	2

Clé primaire composée (plusieurs attributs)

Les clés primaires nécessitent parfois plusieurs attributs pour satisfaire au principe d'unicité.

Exemple (facturation : numéros recommencent à 1 chaque année) :

Année	N° commande	montant
2019	1	15000
2019	2	10000
2020	1	4000
2020	2	3000

Une identification unique de la facture devra comprendre l'année et le numéro de la commande.

CREATE TABLE : clé primaire (1 attribut)

- `idEleve` défini comme clé primaire
- `char(5)` : chaîne de 5 caractères
- `DateDeNaissance` au format date

```
CREATE TABLE eleve (idEleve int(4)
PRIMARY KEY, Nom varchar(50),
Prenom varchar(50), Genre int(1),
DateDeNaissance date, CodePostal
char(5), Ville varchar(50));
```

INT(x)

`INT(x)` :

- ne change pas les capacités de stockage ;
- agit sur l'affichage (padding à gauche si moins de chiffres).
- Si vous stockez un nombre avec un nombre de chiffres inférieur au nombre défini, le caractère par défaut sera ajouté à gauche du chiffre, pour qu'il prenne la bonne taille. Le caractère par défaut est l'espace

CHAR vs VARCHAR

- `CHAR` : taille fixe (réserve toujours la taille définie).
- `VARCHAR` : taille variable (s'adapte à la longueur réelle).

Quand utiliser :

- `CHAR` : codes de taille fixe (ex. code postal).
- `VARCHAR` : adresses, mails, noms, descriptions, etc.

CREATE TABLE : clé primaire composée

```
CREATE TABLE produits
(numproduit CHAR(6),
nomproduit VARCHAR(20),
prixproduit DECIMAL(5,2),
couleurproduit VARCHAR(10),
PRIMARY KEY(numproduit,couleurproduit));
```

- `DECIMAL(5,2)` : format décimal avec 2 décimales
- `VARCHAR(10)` : chaîne d'au plus 10 caractères

DECIMAL(p,s) : précision et arrondi (MySQL)

Pour `DECIMAL(5,3)` : 5 chiffres significatifs dont 3 après la virgule. Exemples : `12,354`, `-54,258`.

Remarques MySQL :

- peut stocker jusqu'à `999,999` dans certains cas (détails d'implémentation).
- hors intervalle : remplace par la valeur la plus proche supportée.
- trop de décimales : arrondi à l'échelle définie.

Exemples :

- `DECIMAL(5,3) : 1012,454 -> 999,999`
- `DECIMAL(5,3) : 45,4583 -> 45,458`

Clé secondaire : UNIQUE

En SQL, les éventuelles clés secondaires sont déclarées par `UNIQUE` :

- vérifie que la valeur saisie n'existe pas déjà ;
- garantit que toutes les valeurs d'un attribut seront différentes.

Exemple : UNIQUE

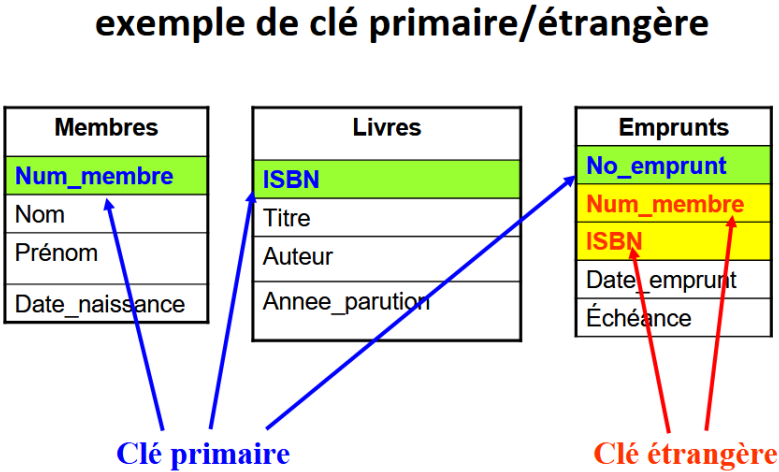
```
CREATE TABLE Etudiant (
num_etudiant INT(6) PRIMARY KEY,
nom VARCHAR(20),
adresse VARCHAR(30),
date_naissance DATE,
code_diplôme INT(3),
num_ss INT(13) UNIQUE);
```

Notion de clé étrangère

Clé étrangère : dans une table, référence à la clé primaire d'une autre table.

Cette référence n'est pas nécessairement unique.

Exemple clé étrangère



Exemple : clé primaire / clé étrangère

- **Membres** : `Num_membre` (clé primaire)
- **Livres** : `ISBN` (clé primaire)
- **Emprunts** :
  - `No_emprunt` (clé primaire)
  - `Num_membre` (clé étrangère vers Membres)
  - `ISBN` (clé étrangère vers Livres)

Intégrité référentielle : définition

- Les jointures naturelles entre relations doivent être déclarées (dans la requête / avant d'effectuer une requête).
- L'intégrité référentielle est respectée si, lors de la création d'une occurrence d'une relation dont un attribut fait référence à un attribut clé d'une autre relation, la valeur correspondante est déjà définie.

Exemple : intégrité référentielle (VOL / PILOTE / AVION)

Une occurrence de `VOL` comporte `Code_Avion` et `Code_Pilote`. Elle ne peut être créée que si les occurrences `PILOTE` et `AVION` référencées existent déjà.

Création d'une occurrence : INSERT

```
INSERT INTO eleve VALUES(16,'Martin','Mazarine',2,
'2002-06-01','75016','Paris');
```

Rappel table :

```
CREATE TABLE eleve (idEleve int(4) PRIMARY KEY, Nom
varchar(50), Prenom varchar(50), Genre int(1), DateDeNaissance
date, CodePostal char(5), Ville varchar(50));
```

INSERT de plusieurs occurrences

```
INSERT INTO eleve VALUES
(18, 'Ossobucco', 'Omer', 1, '2001-07-03', '75018', 'Paris'),
(22, 'Spaghetti', 'Sophie', 2, '1999-09-07', '75002', 'Paris');
```

Contraintes d'intégrité (à la création)

Une contrainte d'intégrité contraint la modification des tables pour garantir des données conformes.

En SQL, dès `CREATE TABLE` via :

- `DEFAULT`, `NOT NULL`, `CHECK`,
- `AUTO_INCREMENT`,
- `ENUM`.

DEFAULT

- définit une valeur par défaut lorsqu'un attribut n'est pas renseigné ;
- doit être suivie de la valeur à affecter.

NOT NULL

- spécifie qu'un attribut doit être saisi ;
- le SGBD refusera d'insérer une occurrence si un attribut `NOT NULL` est absent.

DEFAULT et NOT NULL : exemple

```
CREATE TABLE CLIENTS(
NumClient INT(4) PRIMARY KEY,
Nom VARCHAR (20) NOT NULL,
Age INTEGER NOT NULL,
Adresse VARCHAR (25) ,
Salaire DECIMAL (8, 2) DEFAULT 3000.00
);
```

- `Nom` et `Age` obligatoires
- `Salaire` par défaut : 3000.00

DEFAULT : insertion sans valeur

Pour ne saisir aucune valeur pour `Salaire` :

```
INSERT INTO clients VALUES(10,'Dupont',25,'Rue de Rivoli',DEFAULT);
```

Ou :

```
INSERT INTO clients(Numclient,Nom,Age,Adresse) VALUES
(10,'Dupont',25,'Rue de Rivoli');
```

$\Rightarrow$ `Salaire = 3000.00`

CHECK()

- comporte une condition logique ;
- permet de faire un test sur un attribut.

CHECK : exemple (âge, email)

```
CREATE TABLE clients(
Id_Client integer NOT NULL PRIMARY KEY,
Nom VARCHAR(30) NOT NULL,
Prenom VARCHAR(30),
Age INTEGER, CHECK(age >=18),
Email VARCHAR(50) NOT NULL,
CHECK(Email LIKE '%@%')
);
```

CHECK : attributs obligatoires

Dans la saisie de chaque occurrence :

- `Id_Client`, `Nom`, `Email` doivent être renseignés.

CHECK : contraintes

Dans la saisie de chaque occurrence :

- `Age` doit être ≥ 18
- `Email` doit contenir `@` (et ≤ 50 caractères)

CHECK combiné

```
CREATE TABLE clients(
Id_Client integer NOT NULL PRIMARY KEY,
Nom VARCHAR(30) NOT NULL,
Prenom VARCHAR(30),
Age INTEGER,
Email VARCHAR(50) NOT NULL,
CHECK(age >=18 AND Email LIKE '%@%')
);
```

AUTO_INCREMENT

- la valeur de l'attribut est automatiquement incrémentée à chaque ajout ;
- lors de la saisie : mettre `0` ou `NULL`.

AUTO_INCREMENT : exemple

```
CREATE TABLE fournisseurs (num int
AUTO_INCREMENT PRIMARY KEY, nom
VARCHAR(30), adresse VARCHAR(100));
```

Insertion :

```
INSERT INTO fournisseurs VALUES
(0, 'Dupont', '31 allée des Châtaigniers'),
(0, 'Martin', '22 rue de Rivoli'),
(0, 'Charpentier','15 rue Blanche');
```

Équivalent avec NULL :

```
INSERT INTO fournisseurs VALUES
(null, 'Dupont', '31 allée des Châtaigniers'),
(null, 'Martin', '22 rue de Rivoli'),
(null, 'Charpentier','15 rue Blanche');
```

ENUM

- attribut pour lequel on définit un ensemble de valeurs autorisées (chaînes).

ENUM : exemple

```
CREATE TABLE clients ( IdClient INT AUTO_INCREMENT PRIMARY KEY,
Nom VARCHAR(50) , Prenom VARCHAR(50) , Age INT,
Genre ENUM ('M', 'F', 'Autre', 'Ne pas répondre'));
```

Insertion :

```
INSERT INTO clients VALUES(0,'Dupont', 'Jean', 50, 'M')
```

Ou avec l'index (1 pour 'M', 2 pour 'F', etc.) :

```
INSERT INTO clients VALUES(0, 'Dupont', 'Jean', 50,1)
```

Valeur non autorisée (MySQL) : chaîne vide.

Suppression et modification d'occurrences

Suppression (DELETE) :

```
DELETE FROM eleve WHERE Nom= 'Ossobucco';
```

Modification (UPDATE) :

```
UPDATE eleve SET DateDeNaissance= '1999-12-15'
WHERE Nom= 'Gnocchi';
```

Modification de relations : ALTER / DROP

- modification possible via interface graphique, mais aussi en SQL.
- `ALTER` + opération :
  - ajout : `ADD`
  - modification : `MODIFY`
  - suppression : `DROP`
- suppression d'une relation : `DROP TABLE`

ALTER TABLE : ADD / MODIFY

```
ALTER TABLE eleve ADD DateInscription DATE;
```

Ajoute `DateInscription`.

```
ALTER TABLE eleve MODIFY Ville VARCHAR(100);
```

Modifie le type de `Ville`.

ALTER TABLE : DROP / DROP TABLE

```
ALTER TABLE eleve DROP Ville;
```

Supprime l'attribut `Ville`.

```
DROP TABLE eleve;
```

Supprime la relation `eleve`.

CREATE DATABASE

```
CREATE DATABASE autoecole COLLATE utf8_general_ci ;
```

- crée la base `autoecole`
- `COLLATE utf8_general_ci` : interclassement (comparaison de caractères)
- `utf8_general_ci` supprime les accents et convertit le texte en minuscules.

Exemple complet : base autoecole

```
CREATE DATABASE autoecole COLLATE utf8_general_ci ;
USE autoecole;
CREATE TABLE eleve (idEleve int(3) NOT NULL PRIMARY KEY, Nom
varchar(50), Prenom varchar(50), Genre int(1), DateDeNaissance date,
CodePostal varchar(5), Ville varchar(50));
INSERT INTO eleve VALUES
(8,'Gnocchi' , 'Gwendoline', 2,null, '75008', 'Paris'),
(16, 'Macaroni', 'Mazarine', 2, '2002-06-01', '75016', 'Paris'),
(18, 'Ossobucco', 'Omer', 1, '2001-07-03', '75018', 'Paris'),...
```

USE permet de définir la base de données dans laquelle les relations seront créées.