Bases de données relationnelles

Le modèle relationnel introduit au chapitre précédent est un modèle mathématique permettant de raisonner sur des données tabulées. Il est mis en œuvre par un logiciel particulier, le système de gestion de bases de données (SGBD en abrégé). Un SGBD relationnel est un SGBD utilisant le modèle relationnel pour la représentation des données. L'écrasante majorité des SGBD relationnels utilisent le langage SQL (*Structured Query Language*, langage de requête structuré). Ce dernier permet d'envoyer des ordres au SGDB. Les ordres peuvent être de deux natures. Les *mises à jour* permettent la création de relations, l'ajout d'entité dans ces dernières, leur modification et leur suppression. Les *requêtes* permettent de récupérer les données répondant à des critères particuliers.

Présenter de façon intuitive le langage SQL et les SGBD, tout en restant dans le cadre du programme de terminale, est une tâche complexe. En effet, on ne peut comprendre certains aspects du langage sans connaître un peu le fonctionnement des SGBD et à l'inverse le fonctionnement et les limites des SGBD ne sont explicables qu'avec une certaine pratique de la manipulation de données. Les exemples de code SQL de ce cours utilisent le SGBD libre PostgreSQL. Avec d'autres systèmes, les messages d'erreur peuvent être légèrement différents.

A. SQL : un langage de définition de données

Directement inspiré du modèle relationnel introduit par E. Codd, le langage SQL permet la définition de relations ou tables dans une base de données relationnelle. Ce langage est standardisé par l'ISO, sous la référence ISO/IEC 9075. La dernière version du standard date de 2016.

Le langage SQL permet de créer des tables en spécifiant leur nom, leurs attributs, les types de ces derniers et les contraintes associées à la table. Par exemple, pour créer les tables correspondant à la modélisation finale de la médiathèque obtenue à la fin du chapitre précédent, on peut saisir les ordres suivants :

```
CREATE TABLE usager (nom VARCHAR(90), prénom VARCHAR(90), adresse VARCHAR(300),
                     cp VARCHAR(5), ville VARCHAR(60), email VARCHAR(60),
                     code_barre CHAR(15) PRIMARY KEY);

CREATE TABLE livre (titre VARCHAR(300), éditeur VARCHAR(90), annee INT,
                    isbn CHAR(14) PRIMARY KEY);

CREATE TABLE auteur (a_id INT PRIMARY KEY, nom VARCHAR(90), prénom VARCHAR(90));

CREATE TABLE auteur_de (a_id INT REFERENCES auteur(a_id),
                        isbn CHAR(14) REFERENCES livre(isbn),
                        PRIMARY KEY (a_id, isbn));

CREATE TABLE emprunt (code_barre CHAR(15) REFERENCES usager(code_barre),
                      isbn CHAR(14) PRIMARY KEY REFERENCES livre(isbn), retour DATE);
```

L'interaction avec un SGBD se fait par l'envoi d'une suite d'ordres SQL. Un ordre peut s'étendre sur plusieurs lignes. Les blancs et l'indentation ne sont pas significatifs mais améliorent la lisibilité dans certains cas. Un ordre se termine par un « ; ». Les cinq ordres donnés ci-dessus créent les cinq tables composant notre base de données. Une première remarque de syntaxe est que SQL est insensible à

la casse. On aurait ainsi pu écrire « `create table livre ...` » ou encore « `cReATE TablE LIVrE ...` ». Pour faciliter la lecture, on utilisera des capitales pour les mots clés du langage SQL (`CREATE`, `TABLE`, `PRIMARY`, etc.) et des minuscules pour les noms d'attributs (a_id, prénom, etc.) et de tables. Ces derniers ne pouvant pas contenir d'espace, nous utilisons le caractère « _ » comme séparateur de mots. Enfin, on utilisera la convention, généralement considérée comme une bonne pratique, d'utiliser des noms de tables au singulier. Une autre remarque de syntaxe est que le langage SQL peut sembler « verbeux », à l'inverse d'un langage de programmation comme Python. Les ordres ressemblent à du langage naturel (en anglais). La notation est aussi semblable aux schémas du modèle relationnel. La syntaxe générale d'un ordre `CREATE TABLE` est la suivante :

```
CREATE TABLE nom_table (att1 dom1 contr1?,
            ...
            attn domn contrn?,
            contr_glob1,
            ...
            contr_globn);
```

On donne, à la suite des mots clés `CREATE TABLE`, un nom de table, suivi de la liste des définitions d'attributs entre parenthèses et séparées par des virgules. Une définition d'attribut consiste en un nom d'attribut, un type d'attribut (ou domaine) qui sont tous les deux obligatoires et optionnellement des contraintes sur cet attribut. Si des contraintes portent sur plusieurs attributs à la fois (par exemple pour spécifier que plusieurs attributs forment une clé primaire) on peut placer ces contraintes en fin de liste, avant la parenthèse fermante. Le système lèvera une erreur si la table existe déjà.

Vocabulaire. Strictement parlant, une relation du modèle relationnel et une table SQL ne sont pas des concepts équivalents. En particulier, une table peut contenir des doublons. En effet, il n'est pas obligatoire de spécifier une clé primaire lors de la création d'une table SQL. Sans clé primaire, une table peut contenir plusieurs copies du même n-uplet, sans que cela pose problème, chose qui n'est a priori pas autorisé pour les ensembles du modèle relationnel (car justement, ce sont des ensembles). Nous ignorerons ces différences dans le cadre du programme de terminale et utiliserons les termes « tables » et « relations » indistinctement. De façon analogue, en SQL les attributs d'une relation sont appelés des colonnes et les entités des lignes.

B. Types de données en SQL

Les domaines abstraits du modèle relationnel correspondent à des types de données du langage SQL. En voici quelques types « classiques ».

Types numériques. Le standard SQL définit plusieurs types numériques. Ces derniers sont soit des types numériques exacts, soit des types numériques approchés. La table détaille les différents types numériques. La plupart de ces types représentent fidèlement les entiers ou flottants « machine » manipulables directement par le processeur. Une exception notable est le type `DECIMAL(t, f)` qui permet de représenter de manière exacte un nombre à virgule d'une taille donnée. Ce type est particulièrement important, car il permet par exemple de représenter des sommes d'argent sans erreurs d'arrondis. Ainsi, le type `DECIMAL(5,2)` permet de stocker des valeurs décimales de 5 chiffres, dont deux après la virgule, soit des valeurs entre −999,99 et 999,99. Une utilisation judicieuse de ce type permettra de réaliser une contrainte de domaine sur un attribut numérique. Le standard ne supporte que les nombres en base 10, sans autoriser des notations hexadécimales ou octales comme dans les autres langages.

Types textes. Le standard SQL définit plusieurs types permettant de stocker des chaînes de caractères. Malheureusement, ces derniers sont supportés de manière inégale dans les divers SGBD. On se limitera aux plus communs, donnés ci-après. Le type `CHAR(n)` permet de définir des chaînes de caractères de taille exactement n. La taille maximale acceptée pour n dépend des différents systèmes, mais ils supportent tous au moins 8000 caractères. La taille minimale est 1. Ce type est approprié lorsque l'on veut stocker des chaînes de taille fixe et connue, par exemple les isbn de nos livres qui font exactement 14 caractères, de la forme `'XXX-XXXXXXXXXX'` (trois chiffres, un tiret, dix chiffres). Attention, si l'on stocke une chaîne

nom du type	exact/approché	description
SMALLINT	exact	entier 16 bits signé
INTEGER	exact	entier 32 bits signé
INT	exact	alias pour INTEGER
BIGINT	exact	entier 64 bits signé
DECIMAL(t. f)	exact	décimal signé de t chiffres dont f après la virgule
REAL	approché	flottant 32 bits
DOUBLE PRECISION	approché	flottant 64 bits

Figure 1 – Types numériques en SQL

de taille inférieure à n, cette dernière est complétée par la droite avec des espaces. Ainsi, la chaîne 'hello' stockée dans une colonne de type CHAR(10) sera convertie en 'hello␣␣␣␣␣' (où le caractère « ␣ » représente un espace). Le type VARCHAR(n) permet quant à lui de définir des chaînes de taille au plus n. La valeur de n suit les mêmes règles que pour le type CHAR. Enfin, le type TEXT permet de stocker des chaînes de caractères de taille variable, sans fixer de taille maximale a priori. En pratique, il est équivalent à VARCHAR(n) pour la plus grande valeur de n supportée par le système. Les chaînes de caractères littérales sont délimitées par des guillemets simples « ' ». Le caractère guillemet peut être échappé en le doublant. Par exemple, la chaîne constante « c'est moi » s'écrira 'c''est moi'. Les autres caractères n'ont pas besoin d'être échappés. Une chaîne peut en particulier contenir un retour chariot (et donc être écrite sur plusieurs lignes).

nom du type	description
CHAR(n)	chaîne d'exactement n caractères (les caractères manquants sont complétés par des espaces)
VARCHAR(n)	chaîne d'au plus n caractères
TEXT	chaîne de taille quelconque

Figure 2 – Types textes en SQL

Type booléen. Le type BOOLEAN est inégalement supporté par les différents systèmes, qu'ils soient commerciaux ou libres. Cela est dû au fait que le standard SQL laisse ce type comme optionnel. Les SGBD sont donc libres de ne pas le proposer. Une alternative possible est d'utiliser CHAR(1) et de se servir de deux caractères distincts (par exemple 'T' et 'F') pour représenter des booléens. Une autre alternative est d'utiliser un type numérique exact et de considérer la valeur 0 comme fausse et les autres valeurs comme vraies.

Type des dates, durées et instants. À première vue anodine, la gestion des dates et du temps est un problème excessivement complexe, source de nombreux bugs. Le standard SQL propose donc de nombreux types temporels permettant de représenter des dates, des heures et des durées. Les valeurs de ces types s'écrivent comme de simples chaînes de caractères. Une fonctionnalité intéressante est la possibilité d'utiliser l'addition pour ajouter des jours à une valeur de type DATE. Si d est une expression de type DATE, alors d + 10 représente la date 10 jours après d. Cette opération produit une valeur de type DATE et donc prend correctement en compte les changements de mois, d'années et les années bissextiles.

Valeur NULL. Une valeur notée NULL existe en SQL. Elle représente une absence de valeur et peut donc être utilisée à la place de n'importe quelle autre valeur, quel que soit le type attendu. Son utilisation est similaire à la constante None du langage Python, mais son comportement est complexe et peut être source d'erreurs. Il est déconseillé de l'utiliser dans le cadre d'une initiation aux bases de données. En particulier, SQL interdit l'utilisation de NULL comme valeur pour une clé primaire. En revanche, elle est autorisée pour les clés étrangères. La valeur NULL permet donc de violer la contrainte de référence. La

nom du type	description
DATE	une date au format 'AAAA-MM-JJ'
TIME	une heure au format 'hh:mm:ss'
TIMESTAMP	un instant (date et heure) au format 'AAAA-MM-JJ hh:mm:ss'

FIGURE 3 – Types temporels en SQL

seule chose que l'on fera donc avec des attributs potentiellement NULL est de les tester au moyen des expression *e* IS NULL ou *e* IS NOT NULL. Attention cependant, le test *e* = NULL ne produit pas le résultat booléen comme on pourrait s'y attendre mais renvoie toujours NULL quelle que soit la valeur de *e*.

C. Spécification des contraintes d'intégrité

Les contraintes d'intégrité jouant un rôle fondamental dans le modèle relationnel, il est naturel de pouvoir les spécifier en SQL. Nous montrerons ici comment définir les quatre types de contraintes d'intégrité étudiées au chapitre précédent.

Clé primaire. Les mots clés PRIMARY KEY permettent d'indiquer qu'un attribut est une clé primaire. Nous écrirons par exemple :

```
CREATE TABLE personne (id INT PRIMARY KEY, nom VARCHAR(99), prénom VARCHAR(99));
```

Si l'on souhaite utiliser plusieurs attributs comme clé primaire, on peut spécifier la contrainte après les attributs :

```
CREATE TABLE point (x INT, y INT, couleur VARCHAR(30), PRIMARY KEY (x, y));
```

Clé étrangère. Un attribut peut être qualifié de clé étrangère en utilisant le mot clé REFERENCES suivi de la table où se trouve la clé primaire et de son nom.

```
CREATE TABLE employé (emp INT REFERENCES personne (id), dept VARCHAR(90),
                      suph INT REFERENCES personne (id));
```

La table employé associe un employé (attribut *emp*), qui est une personne, à son département dans l'entreprise (ressources humaines, support informatique, comptabilité, etc.) et à son supérieur hiérarchique (attribut *suph*), qui est aussi une personne. Ce lien est matérialisé par le fait que *emp* et *suph* sont des clés étrangères. Il est à noter que la plupart des SGBD ne supportent pas l'utilisation de clés étrangères composites. L'utilité des clés primaires composites se trouve donc amoindrie en pratique.

Unicité, non nullité. Il peut être intéressant de spécifier qu'un groupe d'attributs est unique, sans pour autant en faire une clé primaire. Cette information permet au SGBD plus de vérifications sur les données (cohérence des données) et parfois de traiter ces dernières de façon plus efficace. Cela peut être spécifié au moyen du mot clé UNIQUE. Une autre bonne pratique consiste à déclarer qu'un attribut ne peut pas être NULL. Cette valeur spéciale ne pourra donc jamais être utilisée pour remplir des valeurs de la colonne correspondante. Cela peut être fait au moyen du mot clé NOT NULL. Notons que PRIMARY KEY implique obligatoirement NOT NULL. En reprenant l'exemple des utilisateurs de la bibliothèque, on pourrait affiner notre définition de table de la manière suivante :

```
CREATE TABLE usager(nom VARCHAR(90) NOT NULL,
                    prénom VARCHAR(90) NOT NULL, adresse VARCHAR(300) NOT NULL,
                    cp VARCHAR(5) NOT NULL, ville VARCHAR(60) NOT NULL,
                    email VARCHAR(60) NOT NULL UNIQUE,
                    code_barre CHAR(15) PRIMARY KEY);
```

On spécifie de cette façon qu'aucun des attributs n'est optionnel et de plus que email doit être unique dans la table (même s'il n'est pas une clé primaire).

Contraintes utilisateur. Il est possible de spécifier des contraintes arbitraires sur les attributs d'une même ligne au moyen du mot clé `CHECK`, suivi d'une formule booléenne. Cette contrainte est placée obligatoirement en fin de déclaration avant la parenthèse fermante (et non pas au niveau d'un attribut). Nous donnons ici quelques exemples et reviendrons plus généralement dans le chapitre suivant sur la syntaxe des expressions.

```
CREATE TABLE produit (id INT PRIMARY KEY,
                      nom VARCHAR(100) NOT NULL,
                      quantité INT NOT NULL,
                      prix DECIMAL(10,2) NOT NULL,
                      CHECK (quantité >= 0 AND prix >= 0));
```

Nous définissons ici une table des produits vendus dans un magasin. Ces derniers ont un identifiant (qui est la clé primaire), un nom, une quantité et un prix. Ceux-ci ne peuvent jamais être négatifs (ce qui n'est pas exprimable uniquement au moyen des types `INT` ou `DECIMAL`). On ajoute donc une contrainte `CHECK`.

D. Suppression de tables

Une fois qu'une table est créée, il n'est pas possible d'en créer une autre avec le même nom. Si on souhaite recréer la table, par exemple avec un schéma différent, il faut d'abord supprimer celle portant le même nom. C'est le but de l'instruction `DROP TABLE` :

```
DROP TABLE auteur_de;
```

Cette dernière supprime la table et donc toutes les données qui y sont stockées. Un point important est qu'il n'est pas possible de supprimer une table si elle sert de référence pour une clé étrangère d'une autre table, car cela violerait une contrainte de référence :

```
# DROP TABLE usager;
ERROR: cannot drop table utilisateurs because other objects
       dépend on it
DETAIL: constraint emprunt_code_barre_fkey on table
        emprunt dépends on table usager
```

Nous reproduisons ci-dessus une interaction avec le SGBD PostgreSQL, mais tous les autres systèmes relationnels auront un comportement semblable. Le système indique que la table usager est mentionnée par une contrainte (ici celle de la table emprunt). Le système refuse donc de supprimer la table et renvoie une erreur. Il convient alors de supprimer les tables dans le bon ordre, c'est-à-dire d'abord les tables contenant les clés étrangères, avant les tables contenant les clés primaires référencées.

```
DROP TABLE emprunt;
DROP TABLE auteur_de;
DROP TABLE auteur;
DROP TABLE livre;
DROP TABLE usager;
```

Certains SGBD permettent de spécifier que la suppression d'une table doit détruire automatiquement toutes les tables qui dépendent d'elle. Cette fonctionnalité est commode mais dangereuse et de toute façon inégalement supportée. Il est donc conseillé de spécifier explicitement les commandes de suppression, dans l'ordre adéquat.

E. Insertion dans une table

Nous pouvons enfin aborder l'insertion de nouvelles valeurs dans une table. Cette action s'effectue au moyen de l'ordre `INSERT INTO`.

```
INSERT INTO auteur VALUES (97, 'Ritchie', 'Dennis'),
                          (98, 'Voltaire', ''),
                          (103, 'Toriyama', 'Akira');
```

On spécifie le nom de la table (ici auteur) suivi d'une suite de n-uplets, chacun entre parenthèses. Chaque n-uplet représente une nouvelle ligne de la table. Les valeurs des attributs sont supposés être dans le même ordre que lors de la création de la table. Si on souhaite les passer dans un ordre différent, on peut spécifier l'ordre des attributs avant le mot clé `VALUES`

```
INSERT INTO auteur (prénom, a_id, nom) VALUES ('Jean-Jacques', 200, 'Rousseau');
```

Un point important est que les contraintes d'intégrités sont vérifiées au moment de l'insertion. Une instruction `INSERT` violant ces contraintes conduira donc à une erreur et les données correspondantes ne seront pas ajoutées à la table.

```
# INSERT INTO auteur (97, 'Dumas', 'Alexandre') ;
ERROR:   duplicate key value violates unique constraint
         "auteur_pkey"
DETAIL:  Key (a_id)=(97) already exists.
```

Ici, on essaye d'ajouter un auteur avec la même clé primaire qu'un auteur déjà existant.

```
# INSERT INTO produit VALUES (1, 'ordinateur', 10, -200);
ERROR:   new row for relation "produit" violates check
         constraint "produit_check"
DETAIL:  Failing row contains (1, ordinateur, 10, -200.00).
```

Ici, l'insertion viole la contrainte `CHECK` de la table produit définie plus haut.