Création et alimentation de bases de données SQL

stph.scenari-community.org/bdd
sql1.pdf

Stéphane Crozat

12 janvier 2018

Table des matières

Cours

I

SQL* est un langage standardisé, implémenté par tous les SGBDR*, qui permet, indépendamment de la plate-forme technologique et de façon déclarative, de définir le modèle de données, de le contrôler et enfin de le manipuler.

1. Le langage SQL

Définition : SQL

SQL* (pour langage de requêtes structuré) est un langage déclaratif destiné à la manipulation de bases de données au sein des SGBD* et plus particulièrement des SGBDR*.

SQL : LDD, LCD, LMD, LCT

SQL est un langage déclaratif, il n'est donc pas a proprement parlé un langage de programmation, mais plutôt une interface standard pour accéder aux bases de données.

Il est composé de quatre sous ensembles :

- Le Langage de Définition de Données (LDD*, ou en anglais DDL, *Data Definition Language*) pour créer et supprimer des objets dans la base de données (tables, contraintes d'intégrité, vues, etc.).
  Exemple de commandes : `CREATE DROP ALTER`
- Le Langage de Contrôle de Données (LCD*, ou en anglais DCL, *Data Control Language*) pour gérer les droits sur les objets de la base (création des utilisateurs et affectation de leurs droits).
  Exemple de commandes : `GRANT REVOKE`
- Le Langage de Manipulation de Données (LMD*, ou en anglais DML, *Data Manipulation Language*) pou la recherche, l'insertion, la mise à jour et la suppression de données. Le LMD est basé sur les opérateurs relationnels, auxquels sont ajoutés des fonctions de calcul d'agrégats et des instructions pour réaliser les opérations d'insertion, mise à jour et suppression.
  Exemple de commandes : `INSERT UPDATE DELETE SELECT`
- Le Langage de Contrôle de Transaction (LCT, ou en anglais TCL, *Transaction Control Language*) pour la gestion des transactions (validation ou annulation de modifications de données dans la BD)
  Exemple de commandes : `COMMIT ROLLBACK`

Fondamental : Référence SQL : SQL-99 complete, really

Gulutzan and Pelzer, 1999*

https://mariadb.com/kb/en/sql-99)

Complément : Origine du SQL

Le modèle relationnel a été inventé par E.F. Codd (Directeur de recherche du centre IBM de San José) en 1970, suite à quoi de nombreux langages ont fait leur apparition :

- IBM Sequel (Structured English Query Language) en 1977
- IBM Sequel/2
- IBM System/R
- IBM DB2

Ce sont ces langages qui ont donné naissance au standard SQL, normalisé en 1986 au États-Unis par l'ANSI* pour donner SQL/86 (puis au niveau international par l'ISO* en 1987).

Complément : Versions de SQL

- SQL-86 (ou SQL-87) : Version d'origine
- SQL-89 (ou SQL-1) : Améliorations mineures
- SQL-92 (ou SQL-2) : Extensions fonctionnelles majeures (types de données, opérations relationnelles, instruction LDD, transactions, etc.
- SQL-99 (ou SQL-3) : Introduction du PSM* (couche procédurale sous forme de procédure stockées) et d RO*
- SQL-2003 : Extensions XML*
- SQL-2006 : Améliorations mineures (pour XML notamment)
- SQL-2008 : Améliorations mineures (pour le RO notamment)

Remarque : Version SQL et implémentations SGBD

Selon leur niveau d'implémentation de SQL, les SGBD acceptent ou non certaines fonctions.

Certains SGBD ayant entamé certaines implémentations avant leur standardisation définitive, ces implémentations peuvent différer de la norme.

2. Créer des tables en SQL (Langage de Définition de Données)

Objectifs

Maîtriser les bases du SQL pour créer et modifier des tables et des vues.

Le LDD* est la partie du langage SQL qui permet de créer de façon déclarative les objets composant une BD*. Il permet notamment la définition des schémas, des relations, des contraintes d'intégrité, des vues.

Rappel : Le code SQL peut être testé avec Db Disco*

2.1. Exercice : Lab I++

Description du problème

[20 min]

Un laboratoire souhaite gérer les médicaments qu'il conçoit.

- Un médicament est décrit par un nom, qui permet de l'identifier. En effet il n'existe pas deux médicaments avec le même nom. Un médicament comporte une description courte en français, ainsi qu'une description longue en latin. On gère aussi le conditionnement du médicament, c'est à dire le nombre de pilules par boîte (qui est un nombre entier).
- À chaque médicament on associe une liste de contre-indications, généralement plusieurs, parfois aucune Une contre-indication comporte un code unique qui l'identifie, ainsi qu'une description. Une contre-indication est toujours associée à un et un seul médicament.

Exemple de données

Afin de matérialiser notre base de données, nous obtenons les descriptions suivantes :

- Le *Chourix* a pour description courte *« Médicament contre la chute des choux »* et pour description longue *« Vivamus fermentum semper porta. Nunc diam velit, adipiscing ut tristique vitae, sagittis vel odio. Maecenas convallis ullamcorper ultricies. Curabitur ornare. »*. Il est conditionné en boîte de 13.
  Ses contre-indications sont :
  - CI1 : Ne jamais prendre après minuit.
  - CI2 : Ne jamais mettre en contact avec de l'eau.
- Le *Tropas* a pour description courte *« Médicament contre les dysfonctionnements intellectuels »* et pour description longue *« Suspendisse lectus leo, consectetur in tempor sit amet, placerat quis neque. Etiam luctus porttitor lorem, sed suscipit est rutrum non. »*. Il est conditionné en boîte de 42.
  Ses contre-indications sont :
  - CI3 : Garder à l'abri de la lumière du soleil

Question 1

[solution n°1 p.26*]*

Réaliser le modèle conceptuel de données en UML du problème.

Question 2

[solution n°2 p.26*]*

En mobilisant les règles adéquates, proposer un modèle logique de données correspondant en relationnel. Le repérage des domaines et des clés est obligatoire.

Question 3

[solution n°3 p.26*]*

Créer une base de données en SQL correspondant au modèle relationnel.

Question 4

[solution n°4 p.26*]*

Insérer les données fournies en exemple dans la base de données.

2.2. Création de tables

Rappel

Qu'est ce que le SQL ? (cf. p.4)

Introduction

La création de table est le fondement de la création d'une base de données en SQL.

Définition : Création de table

La création de table est la définition d'un schéma de relation en intension*, par la spécification de tous les attributs le composant avec leurs domaines respectifs.

Syntaxe

```
1 CREATE TABLE nom_table (
2  nom_colonne1 domaine1,
3  nom_colonne2 domaine2,
4  ...
5  nom_colonneN domaineN
6 );
```

Exemple

```
1 CREATE TABLE Personne (
2  Nom VARCHAR(25),
3  Prenom VARCHAR(25),
4  Age NUMERIC(3)
5 );
```

Attention : Contrainte d'intégrité

La définition des types n'est pas suffisante pour définir un schéma relationnel, il faut lui adjoindre la définition de *contraintes d'intégrité*, qui permette de poser les notions de clé, d'intégrité référentielle, de restriction de domaines, etc.

2.3. Domaines de données

Introduction

Un attribut d'une relation est défini pour un certain domaine ou type. Les types de données disponibles en SQL varient d'un SGBD* à l'autre, on peut néanmoins citer un certain nombre de types standards que l'on retrouve dans tous les SGBD.

Fondamental : Les types standard

- INTEGER ou INT, SMALLINT
- NUMERIC(X)

- DECIMAL(X,Y) ou NUMERIC(X,Y)
- FLOAT(X), REAL
- CHAR(X)
- VARCHAR(X)
- DATE (AAAA-MM-JJ)
- DATETIME (AAAA-MM-JJ HH:MM:SS)

Les types numériques standard

- *Les nombres entiers*
  INTEGER (ou INT) et SMALLINT, permettent de coder des entiers sur 4 octets (-2.147.483.648 à 2.147.483.647) ou 2 octets (-32.768 à 32.767).
- *Les nombres entiers*
  NUMERIC(X) désigne un entier de X chiffres au maximum.
- *Les nombres décimaux*
  DECIMAL(X,Y), où X et Y sont optionnels et désignent respectivement le nombre de chiffres maximum pouvant composer le nombre, et le nombre de chiffres après la virgule.
  NUMERIC(X,Y) est un synonyme standard.
- *Les nombres à virgule flottante*
  FLOAT(X), avec X définissant la précision (nombre de bits de codage de la mantisse).
  REAL est un synonyme standard de FLOAT(24).

Conseil : FLOAT versus DECIMAL

Il est conseillé d'utiliser DECIMAL qui est un nombre exact, plutôt que FLOAT qui est un nombre approximatif, si la précision requise est suffisante. FLOAT sera réservé typiquement à des calculs scientifiques nécessitant un degré de précision supérieur.

Les types chaîne de caractères standard

On distingue principalement les types CHAR(X) et VARCHAR(X), où X est obligatoire et désigne la longueur de la chaîne.

- CHAR définit des chaînes de longueur fixe (complétée à droites par des espaces, si la longueur est inférieure à X) ;
- et VARCHAR des chaînes de longueurs variables.

CHAR et VARCHAR sont généralement limités à 255 caractères. La plupart des SGBD proposent des types, tels que TEXT ou CLOB (Character Long Object), pour représenter des chaînes de caractères longues, jusqu'à 65000 caractères par exemple.

Les types date standard

Les types date dont introduits avec la norme SQL2. On distingue :

- DATE qui représente une date selon un format de type "AAAA-MM-JJ" ;
- et DATETIME qui représente une date plus une heure, dans un format tel que "AAAA-MM-JJ HH:MM:SS".

Complément : Les autres types

En fonction du SGBD, il peut exister de nombreux autres types. On peut citer par exemple :

- MONEY pour représenter des décimaux associés à une monnaie,
- BOOLEAN pour représenter des booléens,
- BLOB (pour Binary Long Oject) pour représenter des données binaires tels que des documents multimédia (images bitmap, vidéo, etc.)
- ...

2.4. La valeur NULL

L'absence de valeur, représentée par la valeur `NULL`, est une information fondamentale en SQL, qu'il ne faut pas confondre avec la chaîne de caractère *espace* ou bien la valeur 0. Il ne s'agit pas d'un type, ni d'une contrainte, mais d'une valeur possible dans tous les types.

Fondamental

Par défaut en SQL `NULL` fait partie du domaine, il faut l'exclure explicitement par la clause `NOT NULL` après la définition de type, si on ne le souhaite pas.

Syntaxe

```
CREATE TABLE nom de table (
CREATE TABLE nom_table (
 nom_colonne1 domaine1 NOT NULL,
 nom_colonne2 domaine2,
 ...
 nom_colonneN domaineN NOT NULL
);
```

2.5. Contraintes d'intégrité

Fondamental

- PRIMARY KEY (<liste d'attibuts>)
- UNIQUE (<liste d'attibuts>)
- FOREIGN KEY (<liste d'attibuts>) REFERENCES <nom table>(<nom colonnes>)
- CHECK (<condition>)

Une contrainte d'intégrité est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD*.

Il existe deux types de contraintes :

- sur une colonne unique,
- ou sur une table lorsque la contrainte porte sur une ou plusieurs colonnes.

Les contraintes sont définies au moment de la création des tables.

Définition : Contraintes d'intégrité sur une colonne

Les contraintes d'intégrité sur une colonne sont :

- PRIMARY KEY : définit l'attribut comme la clé primaire
- UNIQUE : interdit que deux tuples de la relation aient la même valeur pour l'attribut.
- REFERENCES <nom table> (<nom colonnes>) : contrôle l'intégrité référentielle entre l'attribut et la tabl et ses colonnes spécifiées
- CHECK (<condition>) : contrôle la validité de la valeur de l'attribut spécifié dans la condition dans le cadre d'une restriction de domaine

Définition : Contraintes d'intégrité sur une table

Les contraintes d'intégrité sur une table sont :

- PRIMARY KEY (<liste d'attibuts>) : définit les attributs de la liste comme la clé primaire
- UNIQUE (<liste d'attibuts>) : interdit que deux tuples de la relation aient les mêmes valeurs pour l'ensemble des attributs de la liste.
- FOREIGN KEY (<liste d'attibuts>) REFERENCES <nom table>(<nom colonnes>) : contrôle l'intégrité référentielle entre les attributs de la liste et la table et ses colonnes spécifiées
- CHECK (<condition>) : contrôle la validité de la valeur des attributs spécifiés dans la condition dans le cadre d'une restriction de domaine

Syntaxe

```
CREATE TABLE nom de table (
CREATE TABLE nom_table (
 nom_colonne1 domaine1 <contraintes colonne1>,
 nom_colonne2 domaine2 <contraintes colonne2>,
 ...
 nom_colonneN domaineN <contraintes colonneN>,
 <contraintes de table>
);
```

Exemple

```
CREATE TABLE Personne (
 N°SS CHAR(13) PRIMARY KEY,
 Nom VARCHAR(25) NOT NULL,
 Prenom VARCHAR(25) NOT NULL,
 Age INTEGER(3) CHECK (Age BETWEEN 18 AND 65),
 Mariage CHAR(13) REFERENCES Personne(N°SS),
 UNIQUE (Nom, Prenom)
);
```

Remarque : Clé candidate

La clause UNIQUE NOT NULL sur un attribut ou un groupe d'attributs définit une clé candidate non primaire.

Remarque

Les contraintes sur une colonne et sur une table peuvent être combinées dans la définition d'un même schéma de relation.

Remarque

Une contrainte sur une colonne peut toujours être remplacée par une contrainte sur une table.

2.6. Exemple de contraintes d'intégrité

Exemple

```
CREATE TABLE Personne (
 N°SS CHAR(13) PRIMARY KEY,
 Nom VARCHAR(25) NOT NULL,
 Prenom VARCHAR(25) NOT NULL,
 Age INTEGER(3) CHECK (Age BETWEEN 18 AND 65),
 Mariage CHAR(13) REFERENCES Personne(N°SS),
 Codepostal INTEGER(5),
 Pays VARCHAR(50),
 UNIQUE (Nom, Prenom),
 FOREIGN KEY (Codepostal, Pays) REFERENCES Adresse (CP, Pays)
);

CREATE TABLE Adresse (
 CP INTEGER(5) NOT NULL,
 Pays VARCHAR(50) NOT NULL,
 Initiale CHAR(1) CHECK (Initiale = LEFT(Pays, 1)),
 PRIMARY KEY (CP, Pays)
);
```

Dans la définition de schéma précédente on a posé les contraintes suivantes :

- La clé primaire de Personne est N°SS et la clé primaire de Adresse est (CP, Pays).
- Nom, Prénom ne peuvent pas être null et (Nom, Prénom) est une clé.
- Age doit être compris entre 18 et 65 et Initiale doit être la première lettre de Pays (avec la fonction LEFT qui renvoie la sous chaîne à gauche de la chaîne passée en premier argument, sur le nombre de caractères passés en second argument)
- Mariage est clé étrangère vers Personne et (Codepostal, Pays) est une clé étrangère vers Adresse.

Exemple : Réécriture avec uniquement des contraintes de table

```
CREATE TABLE Personne (
 N°SS CHAR(13) ,
 Nom VARCHAR(25) NOT NULL,
 Prenom VARCHAR(25) NOT NULL,
 Age INTEGER(3) ,
 Mariage CHAR(13),
 Codepostal INTEGER(5),
 Pays VARCHAR(50),
 PRIMARY KEY (N°SS),
 UNIQUE (Nom, Prenom),
 CHECK (Age BETWEEN 18 AND 65),
 FOREIGN KEY (Mariage) REFERENCES Personne(N°SS),
 FOREIGN KEY (Codepostal, Pays) REFERENCES Adresse (CP, Pays)
);

CREATE TABLE Adresse (
 CP INTEGER(5) NOT NULL,
```

```
 Pays VARCHAR(50) NOT NULL,
 Initiale CHAR(1),
 PRIMARY KEY (CP, Pays),
 CHECK (Initiale = LEFT(Pays, 1))
);
```

Ce schéma est strictement le même que le précédent, simplement les contraintes ont toutes été réécrites comme des contraintes de table.

3. Insérer, modifier et supprimer des données en SQL (Langage de Manipulation de Données)

Objectifs

Maîtriser les bases du SQL pour entrer, modifier et effacer des données dans les tables.

3.1. Exercice

[solution n°5 p.27*]*

Quelle valeur renvoie la dernière instruction SQL de la liste ci-dessous :

```
CREATE TABLE t (
 a integer,
 b integer,
 c integer);

INSERT INTO t VALUES (0, 0, 0);

INSERT INTO t VALUES (1, 0, 0);

SELECT sum(a) + count(b) FROM t;
```

3.2. Insertion de données par entrée explicite des valeurs

Le langage SQL fournit des instructions pour ajouter des nouveaux tuples à une relation. Il offre ainsi une interface standard pour ajouter des information dans une base de données.

Il existe deux moyens d'ajouter des données, soit par fourniture directe des valeurs des propriétés du tuple à ajouter, soit par sélection des tuples à ajouter depuis une autre relation.

Syntaxe : Insertion directe de valeurs

```
INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>)
VALUES (<Liste ordonnée des valeurs à affecter aux propriétés spécifiées ci-
 dessus>)
```

Exemple

```
INSERT INTO Virement (Date, Montant, Objet)
VALUES (14-07-1975, 1000, 'Prime de naissance');
```

Remarque

- Les propriétés non valorisées sont affectées à la valeur NULL.
- Il est possible de ne pas spécifier les propriétés à valoriser, dans ce cas, toutes les propriétés de la relation seront considérées, dans leur ordre de définition dans la relation (à n'utiliser que dans les cas les plus simples).

Attention : Chaînes de caractères

- Les chaînes de caractère doivent être protégées par des apostrophes : `'`
- Pour insérer une apostrophe, doubler le caractère : `''`

Exemple

```
1 INSERT INTO livre (titre) VALUES ('L''Attrape-cœurs')
```

Attention : Dates

La saisie des dates peut se faire de plusieurs façons dépendante du SGBD, la méthode la plus systématique consiste à utiliser la fonction `TO_DATE(chaîne,format)` où la chaîne de caractère respecte le format.

Par exemple `TO_DATE('20170330','YYYYMMDD')` ou `TO_DATE('30-03-2017','DD-MM-YYYY')` désignent tous les deux le 30 mars 2017.

Exemple

```
1 INSERT INTO livre (pubdate) VALUES (TO_DATE('20170330','YYYYMMDD'))
```

Complément

Fonctions SQL (cf. p.23)

Complément

https://www.postgresql.org/docs/current/static/functions-formatting.html

3.3. Insertion de valeurs par l'intermédiaire d'une sélection

Syntaxe : Insertion de données par sélection de valeurs existantes dans la base

```
1 INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>)
2 SELECT ...
```

L'instruction SELECT projetant un nombre de propriétés identiques aux propriétés à valoriser.

Exemple

```
1 INSERT INTO Credit (Date, Montant, Objet)
2 SELECT Date, Montant, 'Annulation de débit'
3 FROM Debit
4 WHERE Debit.Date = 25-12-2001;
```

Dans cet exemple tous les débits effectués le 25 décembre 2001, sont re-crédités pour le même montant (et à la même date), avec la mention annulation dans l'objet du crédit. Ceci pourrait typiquement réalisé en cas de débits erronés ce jour là.

Remarque

- Les propriétés non valorisées sont affectées à la valeur NULL.
- Il est possible de ne pas spécifier les propriétés à valoriser, dans ce cas, toutes les propriétés de la relation seront considérées, dans leur ordre de définition dans la relation (à n'utiliser que dans les cas les plus simples).

3.4. Mise à jour de données

Le langage SQL fournit une instruction pour modifier des tuples existants dans une relation.

Syntaxe : Mise à jour directe de valeurs

```
1 UPDATE <Nom de la relation>
2 SET <Liste d'affectations Propriété=Valeur, Propriété=Valeur>
3 WHERE <Condition pour filtrer les tuples à mettre à jour>
```

```
1 UPDATE r
2 SET a=1, b='x'
3 WHERE c=0
```

Exemple : Mise à jour directe de valeurs

```
1 UPDATE Compte
2 SET Monnaie='Euro'
3 WHERE Monnaie='Franc'
```

Exemple : Mise à jour par calcul sur l'ancienne valeur

```
1 UPDATE Compte
2 SET Total=Total * 6,55957
3 WHERE Monnaie='Euro'
```

3.5. Suppression de données

Le langage SQL fournit une instruction pour supprimer des tuples existants dans une relation.

Syntaxe

```
1 DELETE FROM <Nom de la relation>
2 WHERE <Condition pour filtrer les tuples à supprimer>
```

Exemple : Suppression de tous les tuples d'une relation

```
1 DELETE FROM FaussesFactures
```

Exemple : Suppression sélective

```
1 DELETE FROM FaussesFactures
2 WHERE Auteur='Moi'
```

4. Supprimer et modifier des tables en SQL (Langage de Définition de Données)

Objectifs

Maîtriser les bases du SQL pour créer et modifier des tables et des vues.

Le LDD* permet de créer les objets composant une BD* de façon déclarative. Il permet notamment la définition des schémas des relations, la définition des contraintes d'intégrité, la définition de vues relationnelles.

4.1. Suppression d'objets

Il est possible de supprimer des objets de la BD*, tels que les tables ou les vues.

Syntaxe

```
1 DROP <type objet> <nom objet>
```

Exemple

```
1 DROP TABLE Personne;
2 DROP VIEW Employe;
```

4.2. Modification de tables

Introduction

L'instruction ALTER TABLE permet de modifier la définition d'une table (colonnes ou contraintes) préalablement créée.

Cette commande absente de SQL-89 est normalisée dans SQL-92

Syntaxe : Ajout de colonne

```
1 ALTER TABLE <nom de table>
2 ADD <définition de colonne>
```

Syntaxe : Suppression de colonne

```
1 ALTER TABLE <nom de table>
```

```
2 DROP <nom de colonne>
```

Syntaxe : Ajout de contrainte

```
1 ALTER TABLE <nom de table>
2 ADD <définition de contrainte de table>
```

Remarque : Modification de table sans donnée sans la commande ALTER

Pour modifier une table ne contenant pas encore de donnée, la commande ALTER n'est pas indispensable, l'on peut supprimer la table à modifier (DROP) et la recréer telle qu'on la souhaite. Notons néanmoins que si la table est référencée par des clauses FOREIGN KEY, cette suppression sera plus compliquée, car il faudra également supprimer et recréer les tables référençantes (ce qui ce complique encore si ces dernières contiennent des données).

Remarque : Modification de table avec données sans la commande ALTER

Pour modifier une table contenant des données, la commande ALTER n'est pas indispensable. On peut en effet :

1. Copier les données dans une table temporaire de même schéma que la table à modifier
2. Supprimer et recréer la table à modifier avec le nouveau schéma
3. Copier les données depuis la table temporaire vers la table modifiée

4.3. Exemple de modifications de tables

Table initiale

Soit une table initiale telle que définie ci-après.

```
1 CREATE TABLE Personne (
2  pk_n NUMERIC(4),
3  nom VARCHAR(50),
4  prenom VARCHAR(50),
5  PRIMARY KEY (pk_n)
6 );
```

Modifications

On décide d'apporter les aménagements suivants à la table : on définit "nom" comme UNIQUE et on supprime le champ "prenom".

```
1 ALTER TABLE Personne
2 ADD UNIQUE (nom);
3
4 ALTER TABLE Personne
5 DROP prenom;
```

Table finale

La table obtenue après modification est identique à la table qui aurait été définie directement telle que ci-après.

```
1 CREATE TABLE Personne (
```

```
    pk_n NUMERIC(4),
    nom VARCHAR(50),
    PRIMARY KEY (pk_n),
    UNIQUE (nom)
);
```

Exercices

1. The show

[30 minutes]

Soit le schéma relationnel suivant décrivant un système de réservations de places de spectacles :

```
1 SPECTACLE (#nospectacle:int, nom:str, durée:minutes, type:
  {théâtre|danse|concert})
2 SALLE (#nosalle:int, nbplaces:int)
3 REPRESENTATION (#date:timestamp, #nospectacle=>SPECTACLE, #nosalle=>SALLE, prix:
  decimal)
```

1.1. Exercice : Étude du schéma

[solution n°6 p.27*]*

1.1.1. Exercice

Le schéma permet de gèrer un espace de spectacles composés d'un ensemble de salles.

- Vrai
- Faux

1.1.2. Exercice

La durée du spectacle est donnée en minutes.

- Vrai
- Faux

1.1.3. Exercice

Le type SQL `TIMESTAMP` désigne un instant à la seconde près, par exemple : `20/03/2017 10:37:22`.

- Vrai
- Faux

1.1.4. Exercice

On peut supposer que le champ `date` désigne le début d'une représentation.

- Vrai
- Faux

1.1.5. Exercice

Deux spectacles identiques peuvent être joués en même temps dans deux salles différentes.

- Vrai
- Faux

1.1.6. Exercice

Deux spectacles différents peuvent être joués en même temps dans la même salle.

- Vrai
- Faux

1.2. Exercice :

Question 1

[solution n°7 p.28*]*

Retro-concevoir le MCD en UML.

Question 2

[solution n°8 p.28*]*

Proposer des contraintes d'intégrité réalistes pour ce schéma (en français).

Question 3

[solution n°9 p.29*]*

Proposer une définition du schéma en SQL qui prenne en compte certaines de ces contraintes.

Question 4

Insérer des données réalistes dans votre schéma afin de vérifier son bon fonctionnement.

2. Exercice : Du producteur au consommateur

[30 min]

Soit le modèle relationnel suivant :

```
1 Producteur(#raison_sociale:chaîne(25), ville:chaîne(255))
2 Consommateur(#login:chaîne(10), #email:chaîne(50), nom:chaîne(50),prenom:chaîne
  (50), ville:chaîne(255))
3 Produit(#id:entier, description:chaîne(100), produit-par=>Producteur, consomme-
  par-login=>Consommateur, consomme-par-email=>Consommateur)
```

On ajoute que :

- `(nom,prenom,ville)` est une clé candidate de `Consommateur`
- Tous les produits sont produits
- Tous les produits ne sont pas consommés

Question 1

[solution n°10 p.29*]*

Rétro-concevez le modèle conceptuel sous-jacent à ce modèle relationnel.

Question 2

[solution n°11 p.30*]*

Établissez le code LDD standard permettant d'implémenter ce modèle en SQL.

Question 3

[solution n°12 p.30*]*

Insérez les données dans votre base de données correspondant aux assertions suivantes :

- L'entreprise de Compiègne "Pommes Picardes SARL" a produit 4 lots de pommes, et 2 lots de cidre.
- Il existe trois utilisateurs consommateurs dans la base, donc les adresses mails sont :
  Al.Un@compiegne.fr - Bob.Deux@compiegne.fr - Charlie.Trois@compiegne.fr
  Ce sont des employés de la ville de Compiègne qui habitent cette ville. Leur mail est construit sur le modèle Prenom.Nom@compiegne.fr. Leur login est leur prénom.

Question 4

[solution n°13 p.31*]*

Modifiez les données de votre base de données pour intégrer les assertions suivantes :

- 1 lots de pommes a été consommés par Al Un.
- 2 lots de pomme ont été consommé par Bob Deux.
- Tous les lots de cidre ont été consommés par Al Un.

Question 5

[solution n°14 p.31*]*

Charlie Trois n'ayant rien consommé, modifiez votre base de données afin de le supprimer de la base.

Devoir

III

1. Exercice : Jeanne et Serge

[30 min]

Vous avez en charge la réalisation d'une base de données pour gérer un tournoi de Volley-Ball. Les équipes, identifiées par un nom, sont composées de deux à six joueurs dont on gère le nom et le prénom. Un match oppose deux équipes, à une date donnée et toujours renseignée, dont une est gagnante, et sont joués en deux sets gagnants (le score est donc toujours de 2 pour le gagnant et 0 ou 1 pour le perdant, on décide donc de ne gérer que le score du perdant). On notera que deux équipes se rencontrent au plus une fois.

Question 1

Réaliser le modèle conceptuel de la BD en utilisant le formalisme UML. Énoncer les éventuelles contraintes non représentées sur le schéma.

Question 2

Traduire le MCD en modèle logique relationnel. Énoncer les éventuelles contraintes non représentables en relationnel.

Question 3

Proposer une implémentation SQL du modèle relationnel, en intégrant le maximum de contraintes. Énoncer les éventuelles contraintes non représentables en SQL.

Contenus annexes

> Fonctions SQL

Rappel

Par opposition aux fonctions de calcul SQL qui s'appliquent sur toute la table pour réaliser des agrégats (en ne renvoyant qu'une seule valeur par regroupement), les fonctions "mono-ligne" sont des fonctions au sens classique, qui s'appliquent à une ou plusieurs valeurs et renvoient une valeur en retour.

Les fonctions "mono-ligne" :

- Manipulent des éléments de données
- Acceptent des arguments en entrée et retournent des valeurs en sortie
- Agissent sur chaque ligne
- Retournent un seul résultat par ligne
- Peuvent modifier les types de données

Exemple

- Traitement de chaîne
    - CONCAT, SUBSTR, LENGTH, INSRT, LPAD, TRIM
    - LOWER, UPPER, INITCAP
- Traitement de date
    - MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY
    - `SELECT sysdate FROM dual`
    - Opérations mathématiques sur les dates : `SELECT sysdate + 10 FROM dual`
- Traitement numérique
    - ROUND, TRUNC
    - FLOOR, CEIL
    - MOD
- Conversion
    - Conversion implicite
    - Conversion explicite : TO_DATE, TO_NUMBER, TO_CHAR
- Générales
    - NVL (par exemple NVL(X,0) renvoie 0 si X vaut Null)
    - CASE WHEN condition1 THEN valeur1 WHEN condition2 THEN valeur2 ELSE valeur3 END

- Imbrication de fonctions : F3(F2(F1(col,arg1),arg2),arg3)

Méthode

Les fonctions mono-ligne sont utilisées pour :

- Transformer les données
- Formater des dates et des nombres pour l'affichage
- Convertir des types de données de colonnes
- ...

Exemple : Extraction de chaîne

La fonction `SUBSTR(X, A, B)` renvoie les B caractères à partir du caractère A dans la chaîne X.

Complément

- *Fonctions SQL*
- Vous pouvez consulter Oracle : SQL*, page 9 à 12, pour avoir une description plus détaillée des fonctions disponibles sous Oracle.

Rappel : BD "Gestion des intervenants" : Schéma relationnel

```
tIntervenant (#pknom:varchar, prenom:varchar, poste:integer)
tCours (#pkannee:2000..2100, #pknum:integer, titre:varchar, type:C|TD|TP,
fkintervenant=>tIntervenant, debut:date)
```

Exemple : BD "Gestion des intervenants" : Question avec CASE

```
SELECT pknum AS cours,
  CASE
    WHEN type='C' THEN 'Cours'
    WHEN type='TD' THEN 'Travaux dirigés'
    WHEN type='TP' THEN 'Travaux pratiques'
  END AS type_label
FROM tCours
```

```
COURS TYPE_LABEL
----- -----------------
    1 Cours
    2 Travaux dirigés
```

Questions de synthèse

A quoi sert le LDD ?

.....
.....
.....
.....
.....
.....
.....

En quoi le LDD est il un langage déclaratif ?

.....
.....
.....
.....
.....
.....
.....

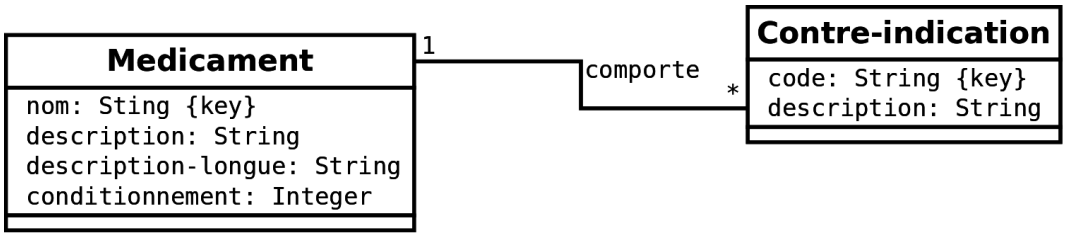
Quel rapport y-a-t il entre le LDD et le concept de relation ?

.....
.....
.....
.....
.....
.....
.....

Solutions des exercices

> **Solution** n°1

Exercice p. 6



> **Solution** n°2

Exercice p. 6

```
Medicament (#nom:varchar, description:varchar, description_longue:varchar,
conditionnement:number)
Contre_indication(#code:varchar, description:varchar, medicament=>Medicament)
```

> **Solution** n°3

Exercice p. 6

```
CREATE TABLE Medicament (
nom varchar,
description varchar,
description_longue varchar,
conditionnement integer,
PRIMARY KEY (nom)
);

CREATE TABLE Contre_indication (
code varchar,
description varchar,
medicament varchar,
PRIMARY KEY (code),
FOREIGN KEY (medicament) REFERENCES Medicament (nom)
);
```

> **Solution** n°4

Exercice p. 6

```
INSERT INTO Medicament (nom,description,description_longue,conditionnement)
VALUES ('Chourix','Médicament contre la chute des choux','Vivamus...',13);
INSERT INTO Contre_indication (code,description,medicament)
VALUES ('CI1','Ne jamais prendre après minuit','Chourix');
INSERT INTO Contre_indication (code,description,medicament)
VALUES ('CI2','Ne jamais mettre en contact avec de l''eau','Chourix');
```

```
7
8 INSERT INTO Medicament (nom,description,description_longue,conditionnement)
9 VALUES ('Tropas','Médicament contre les dysfonctionnements intellectuels',
  'Suspendisse...',42);
10 INSERT INTO Contre_indication (code,description,medicament)
11 VALUES ('CI3','Garder à l''abri de la lumière du soleil','Tropas');
```

> **Solution** n°5

Exercice p. 12

Quelle valeur renvoie la dernière instruction SQL de la liste ci-dessous :

```
1 CREATE TABLE t (
2  a integer,
3  b integer,
4  c integer);
5
6 INSERT INTO t VALUES (0, 0, 0);
7
8 INSERT INTO t VALUES (1, 0, 0);
9
10 SELECT sum(a) + count(b) FROM t;
```

6

Contenus de la table :

- Après Create : Ø
- Après Insert n°1 :

0	0	0

- Après Insert n°2 :

0	0	0
1	0	0

On a sum(a) = 1 et count(b) = 2 donc La requête renvoie *3*.

> **Solution** n°6

Exercice p. 19

Exercice

Le schéma permet de gèrer un espace de spectacles composés d'un ensemble de salles.

- ◉ Vrai
- ○ Faux

Exercice

La durée du spectacle est donnée en minutes.

- ◉ Vrai
- ○ Faux

Exercice

Le type SQL `TIMESTAMP` désigne un instant à la seconde près, par exemple : `20/03/2017 10:37:22`.

- ⦿ Vrai
- ○ Faux

Exercice

On peut supposer que le champ `date` désigne le début d'une représentation.

- ⦿ Vrai
- ○ Faux

Exercice

Deux spectacles identiques peuvent être joués en même temps dans deux salles différentes.

- ⦿ Vrai
- ○ Faux

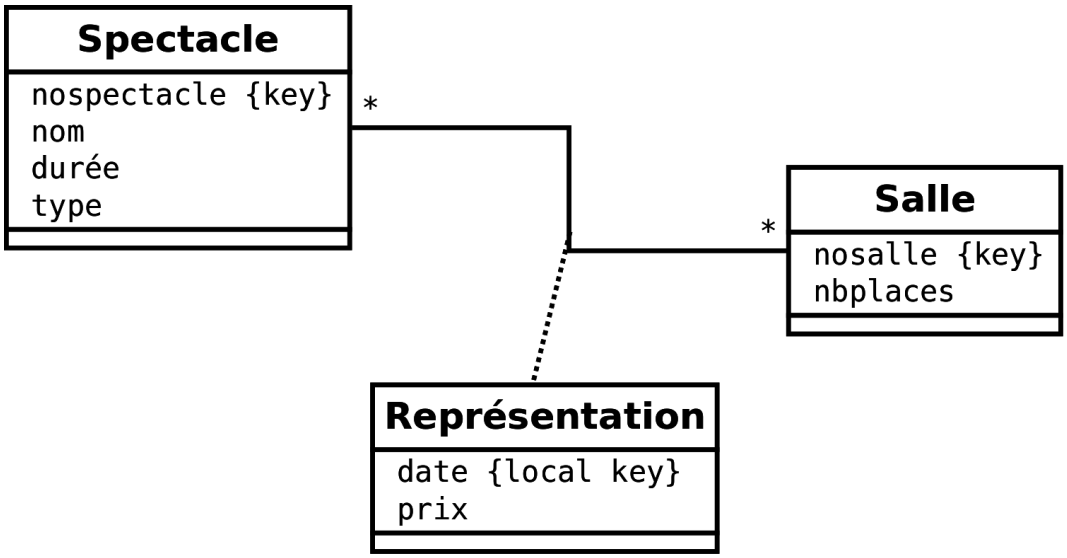
Exercice

Deux spectacles différents peuvent être joués en même temps dans la même salle.

- ⦿ Vrai
- ○ Faux

> Solution n°7

Exercice p. 20



Complément : Classe d'association

- *Classe d'association* *(cf. p.)*
- *Transformation des classes d'association* *(cf. p.)*

> **Solution** n°8

Exercice p. 20

Exemple

- sur SPECTACLES :
  - le nom d'un spectacle est non nul,
  - la durée d'un spectacle est comprise entre 60 minutes et 240 minutes,
- sur SALLES :
  - la capacité d'une salle est comprise entre 100 et 500 places.
- sur REPRESENTATION :
  - le prix d'une représentation est compris entre 10 et 50 €,
  - certaines dates peuvent être interdites (le premier mai par exemple).

> **Solution** n°9

Exercice p. 20

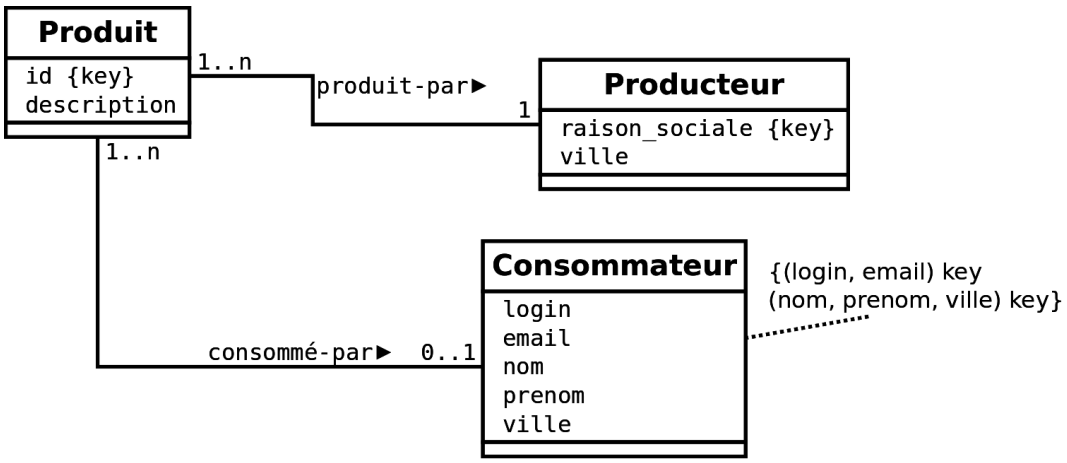
```
CREATE TABLE SPECTACLE (
nospectacle integer,
nom varchar(30) NOT NULL,
duree integer,
type char(7),
PRIMARY KEY (nospectacle),
CHECK (duree>1 and duree<4),
CHECK (type in ('théâtre', 'danse', 'concert'))
);
```

```
CREATE TABLE SALLE (
nosalle integer,
nbplaces integer,
PRIMARY KEY (nosalle),
CHECK (nbplaces>100 and nbplaces<500)
);
```

```
CREATE TABLE REPRESENTATION (
nospectacle integer,
nosalle integer,
rdate date,
prix decimal(3,2),
PRIMARY KEY (nosalle, date, nospectacle),
FOREIGN KEY (nospectacle) REFERENCES SPECTACLE(nospectacle),
FOREIGN KEY (nosalle) REFERENCES SALLE(nosalle),
CHECK (prix>10 and prix<50)
);
```

> **Solution** n°10

Exercice p. 20



Produits

> **Solution** n°11

Exercice p. 20

```
CREATE TABLE  Producteur (
raison_sociale VARCHAR (25),
ville VARCHAR(255),
PRIMARY KEY (raison_sociale)
);

CREATE TABLE Consommateur (
login VARCHAR(10),
email VARCHAR(50),
nom VARCHAR(50) NOT NULL,
prenom VARCHAR(50) NOT NULL,
ville VARCHAR(255) NOT NULL,
PRIMARY KEY (login,email),
UNIQUE (nom,prenom,ville)
);

CREATE TABLE Produit (
id INTEGER,
description VARCHAR(100),
produit_par VARCHAR(25) NOT NULL,
consomme_par_login VARCHAR(10),
consomme_par_email VARCHAR(50),
PRIMARY KEY (id),
FOREIGN KEY (produit_par) REFERENCES Producteur(raison_sociale),
FOREIGN KEY (consomme_par_login,consomme_par_email) REFERENCES Consommateur(login,
email)
);
```

> **Solution** n°12

Exercice p. 21

```
-- Insertion du producteur

```

```
INSERT INTO Producteur (raison_sociale, ville)
VALUES ('Pommes Picardes SARL', 'Compiègne');

-- Insertion des produits

INSERT INTO Produit (id, description, produit_par)
VALUES (1, 'Lot de pommes', 'Pommes Picardes SARL');

INSERT INTO Produit (id, description, produit_par)
VALUES (2, 'Lot de pommes', 'Pommes Picardes SARL');

INSERT INTO Produit (id, description, produit_par)
VALUES (3, 'Lot de pommes', 'Pommes Picardes SARL');

INSERT INTO Produit (id, description, produit_par)
VALUES (4, 'Lot de pommes', 'Pommes Picardes SARL');

INSERT INTO Produit (id, description, produit_par)
VALUES (5, 'Lot de cidre', 'Pommes Picardes SARL');

INSERT INTO Produit (id, description, produit_par)
VALUES (6, 'Lot de cidre', 'Pommes Picardes SARL');

-- Insertion des consommateurs

INSERT INTO Consommateur (login, email, nom, prenom, ville)
VALUES ('Al', 'Al.Un@compiegne.fr', 'Un', 'Al', 'Compiègne');

INSERT INTO Consommateur (login, email, nom, prenom, ville)
VALUES ('Bob', 'Bob.Deux@compiegne.fr', 'Deux', 'Bob', 'Compiègne');

INSERT INTO Consommateur (login, email, nom, prenom, ville)
VALUES ('Charlie', 'Charlie.Trois@compiegne.fr', 'Trois', 'Charlie', 'Compiègne');
```

> **Solution** n°13

Exercice p. 21

```
UPDATE produit
SET consomme_par_login='Al', consomme_par_email='Al.Un@compiegne.fr'
WHERE id=1;

UPDATE produit
SET consomme_par_login='Bob', consomme_par_email='Bob.Deux@compiegne.fr'
WHERE id=2 OR id=3;

UPDATE produit
SET consomme_par_login='Al', consomme_par_email='Al.Un@compiegne.fr'
WHERE description='Lot de cidre';


```

> **Solution** n°14

Exercice p. 21

```
DELETE FROM consommateur
WHERE login='Charlie' AND email='Charlie.Trois@compiegne.fr';
```

Glossaire

Intension

L'intension est l'explicitation d'un domaine par la description de ses caractéristiques (en vue de sa compréhension abstraite, générale).

Elle s'oppose à l'extension qui est l'énonciation exhaustive de l'ensemble des objets du domaine.

- Exemple : Le domaine des couleurs
- Contre-exemple : {bleu, rouge, vert}

Abréviations

ANSI : American National Standards Institute

BD : Base de Données

ISO : International Standardization Organization

LCD : Langage de Contrôle de Données

LDD : Langage de Définition de Données

LMD : Langage de Manipulation de Données

PSM : Persistent Stored Modules

RO : Relationnel-Objet

SGBD : Système de Gestion de Bases de Données

SGBDR : Système de Gestion de Bases de Données Relationnelles

SQL : Structured Query Language

XML : eXtensible Markup Language

Références

dbdisco.crzt.fr *http://dbdisco.crzt.fr*

Bibliographie

Gulutzan Peter, Pelzer Trudy. 1999. *SQL-99 complete, really*. CMP books
.

Webographie

Roegel Denis, *Oracle : SQL*, http://www.loria.fr/~roegel/cours/iut/oracle-sql.pdf, 1999
.