Création et alimentation de bases de données SQL

Table des matières

Cours

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

1.1. Exercice

Parmi les assertions suivantes concernant le langage SQL, sélectionner celles qui sont correctes.

- ☐ Le langage SQL permet d'implémenter physiquement un modèle logique exprimé en relationnel.
- ☐ Le langage SQL permet d'entrer et de sortir des données d'une base de données.
- ☐ Le langage SQL permet de donner et d'enlever des droits en lecture sur les données d'une base de données.
- ☐ Le langage SQL permet de donner et d'enlever des droits en écriture sur les données d'une base de données.
- ☐ Le langage SQL permet de créer une interface graphique utilisable par les utilisateurs finaux.
- ☐ Le langage SQL permet de faire des traitements algorithmiques complexes.
- ☐ Le langage SQL permet de créer une application informatique complète.

1.2. Le langage SQL

SQL ***Définition***

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*) pour 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`

Origine du SQL — *Complément*

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).

Versions de SQL — *Complément*

- 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 du *RO*∗
- SQL-2003 : Extensions *XML*∗
- SQL-2006 : Améliorations mineures (pour XML notamment)
- SQL-2008 : Améliorations mineures (pour le RO notamment)

Version SQL et implémentations SGBD — *Remarque*

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.

1.3. Conseils pour l'apprentissage du SQL

Le SQL, c'est du code informatique, il faut le faire fonctionner — *Conseil*

dbdisco.crzt.fr∗

Alimenter vos bases de données *Conseil*

Pour tester une base de données et voir si elle fonctionne, il faut l'alimenter avec des données, pour vérifier qu'elle permet effectivement de faire ce que l'on souhaite.

Une base de données sans données, c'est au mieux une base, mais ça n'est pas suffisant !

Utiliser une référence en ligne *Conseil*

- Utiliser une référence correspondant à son SGBDR.
- Dans le doute PostgreSQL est un des SGBDR les plus proches du standard et dont la documentation est la mieux faite : https://docs.postgresql.fr

Référence SQL : SQL-99 complete, really *Fondamental*

Gulutzan and Pelzer, 1999∗

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

Réaliser le modèle conceptuel de données en UML du problème.

Question 2

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

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

Question 4

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.3)*

Introduction

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

Création de table ***Définition***

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

```sql
CREATE TABLE nom_table (
  nom_colonne1 domaine1,
  nom_colonne2 domaine2,
  ...
  nom_colonneN domaineN
);
```

Exemple

```sql
CREATE TABLE Personne (
  Nom VARCHAR(25),
  Prenom VARCHAR(25),
  Age NUMERIC(3)
);
```

Contrainte d'intégrité *Attention*

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.

Les types standard *Fondamental*

- 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).

FLOAT versus DECIMAL *Conseil*

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 sont 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".

Types de données PostgreSQL — *Complément*

https://www.postgresql.org/docs/current/static/datatype.html

Les autres types — *Complément*

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

```sql
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.

Contraintes d'intégrité sur une colonne

Définition

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 table 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

Contraintes d'intégrité sur une table

Définition

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_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 CHECK (Age BETWEEN 18 AND 65),
  Mariage CHAR(13) REFERENCES Personne(N°SS),
  UNIQUE (Nom, Prenom)
);
```

Clé candidate

Remarque

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

À condition qu'aucun des attributs du groupe ne soit lui même UNIQUE (sinon la clause de minimalité d'une clé n'est pas respectée).

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

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

Réécriture avec uniquement des contraintes de table ***Exemple***

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

CREATE TABLE Personne (
N°SS CHAR(13) ,
Nom VARCHAR(25) NOT NULL,
Prenom VARCHAR(25) NOT NULL,
Age INTEGER,
Mariage CHAR(13),
Codepostal INTEGER,
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)
);
```

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

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

```sql
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.

Insertion directe de valeurs

Syntaxe

```
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).

Chaînes de caractères

Attention

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

Exemple

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

Dates

Attention

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

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

Complément

Fonctions SQL (cf. p.20)

Complément

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

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

Insertion de données par sélection de valeurs existantes dans la base — *Syntaxe*

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

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

Exemple

```
INSERT INTO Credit (Date, Montant, Objet)
SELECT Date, Montant, 'Annulation de débit'
FROM Debit
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.

Mise à jour directe de valeurs — *Syntaxe*

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

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

Mise à jour directe de valeurs — *Exemple*

```
UPDATE Compte
SET Monnaie='Euro'
WHERE Monnaie='Franc'
```

Mise à jour par calcul sur l'ancienne valeur — *Exemple*

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

3.5. Suppression de données

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

Syntaxe

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

Suppression de tous les tuples d'une relation *Exemple*

```
DELETE FROM FaussesFactures
```

Suppression sélective *Exemple*

```
DELETE FROM FaussesFactures
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

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

Exemple

```
DROP TABLE Personne;
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

Ajout de colonne *Syntaxe*

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

Suppression de colonne — *Syntaxe*

```
ALTER TABLE <nom de table>
DROP <nom de colonne>
```

Ajout de contrainte — *Syntaxe*

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

Modification de table sans donnée sans la commande ALTER — *Remarque*

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).

Modification de table avec données sans la commande ALTER — *Remarque*

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.

```
CREATE TABLE Personne (
 pk_n NUMERIC(4),
 nom VARCHAR(50),
 prenom VARCHAR(50),
 PRIMARY KEY (pk_n)
);
```

Modifications

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

```
ALTER TABLE Personne
ADD UNIQUE (nom);

ALTER TABLE Personne
DROP prenom;
```

Table finale

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

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

Exercices

1. The show

[60 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

Exercice

Le schéma permet de gérer 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

1.2. Exercice

Question 1

Retro-concevoir le MCD en UML.

Question 2

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

Question 3

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

[60 min]

Soit le modèle relationnel suivant :

```
Producteur(#raison_sociale:chaîne(25), ville:chaîne(255))
Consommateur(#login:chaîne(10), #email:chaîne(50), nom:chaîne(50),prenom:chaîne(50), ville:chaîne(255))
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

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

Question 2

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

Question 3

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

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

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

Question 5

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

Contenus annexes

1. 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
- ...

Extraction de chaîne *Exemple*

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

Complément

- Fonctions SQL[1]
- Vous pouvez consulter *Oracle : SQL∗*, page 9 à 12, pour avoir une description plus détaillée des fonctions disponibles sous Oracle.

BD "Gestion des intervenants" : Schéma relationnel *Rappel*

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

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

```
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
```

[1] http://docs.oracle.com/cd/B19188_01/doc/B15917/sqfunc.htm

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 *DB Disco* est une application web permettant de créer une base de données temporaire (les données sont effacées à la déconnexion).

http://pic.crzt.fr/dbdisco

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.

Index