Création et alimentation de bases de données SQL

Table des matières

I Contexte

Durée : 2h

Environnement de travail : DB Fiddle

Pré-requis : Aucun

La dernière étape de la conception d'une base de donnée relationnelle est son implémentation : c'est le moment où vous choisissez un système de gestion de bases de données.

Le modèle relationnel que vous avez conçu est suffisamment abstrait pour ne pas s'enfermer dans un SGBD particulier. En revanche, dès que l'implémentation est faite, est-il encore possible de faire marche arrière, de changer de SGBD ? La réponse est *« oui »*, et c'est avant tout grâce au langage SQL.

SQL [p.45] est un langage standardisé, implémenté par tous les *SGBDR* [p.45], 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. Dans cette série de modules, vous apprendrez à effectuer toutes les opérations de base avec SQL.

II Le langage SQL

Objectif

- Découvrir les possibilités du langage SQL.

Mise en situation

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* p.45 pour donner SQL/86 (puis au niveau international par l'*ISO* p.45 en 1987).

SQL Az Définition

SQL p.45 (pour langage de requêtes structuré) est un langage déclaratif destiné à la manipulation de bases de données au sein des *SGBD* p.45 et plus particulièrement des *SGBDR* p.45.

Sous-ensembles de SQL : LDD, LCD, LMD, LCT

SQL est un **langage déclaratif**, il n'est donc un langage de programmation complet, 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* p.45, 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* p.45, 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* p.45, 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`

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

Conseil

Le code SQL est fait pour être testé. Cela peut être fait simplement grâce à des interfaces en ligne comme :

- db-fiddle.com[1]
- *dbdisco.crzt.fr*[dbdisco p.46]

Alimenter vos bases de données

Conseil

Une base de données sans données, c'est au mieux une base, mais pas une base de données. Ce n'est pas suffisant pour être testé.

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.

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 très complète : https://docs.postgresql.fr.

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* [p.45] (couche procédurale sous forme de procédure stockées) et du *RO* [p.45]
- SQL-2003 : Extensions *XML* [p.45]
- SQL-2006 : Améliorations mineures (pour XML notamment)
- SQL-2008 : Améliorations mineures (pour le RO notamment)

Les SGBD acceptent ou non certaines fonctions en fonction leur niveau d'implémentation de SQL. Certains SGBD ont implémenté des fonctions avant leur standardisation SQL, en conséquence elles peuvent différer du standard.

1. https://www.db-fiddle.com/

Référence SQL : SQL-99 complete, really

Fondamental

Gulutzan and Pelzer, 1999[Gulutzan and Pelzer, 1999 p.47]

Exercice : Appliquer la notion : Enquête documentaire

[solution n°1 p. 36]

Dans cet exercice, on consultera le début de la documentation de PostgreSQL : docs.postgresql.fr/current/preface.html

Exercice

Quel est le nom du projet original sur lequel se base PostgreSQL ?

- **A** Ingres
- **B** MySQL
- **C** POSTGRES
- **D** SQLite

Exercice

Parmi les propositions suivantes, quelles fonctionnalités SQL sont présentes dans PostgreSQL ?

- **A** Transactions
- **B** Table
- **C** Vues
- **D** Clés étrangères
- **E** Indexage
- **F** Triggers
- **G** Procédure

Exercice

PostgreSQL peut être :

- **A** Librement et gratuitement utilisée
- **B** Modifiée

C Redistribuée

D Revendue

IV Création de tables

Objectif

- Connaître les instructions de création, de modification et de suppression de table.

Mise en situation

L'élément fondamental des bases de données relationnelles est la table, qui correspond aux relations du modèle relationnel.

Les tables sont porteuses de la structure des données, et ce sont elles qui contraignent quelle donnée peut être ajoutée ou non à une base de données.

Le langage SQL fournit des instructions pour créer, modifier et supprimer des tables, que vous allez découvrir dans ce module.

Création de table

Az Définition

La création de table est la définition d'un schéma de relation en *intension* p.44, par la spécification de tous les attributs le composant avec leurs domaines respectifs.

Syntaxe

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

Exemple

```
CREATE TABLE personne (
nom TEXT,
prenom TEXT,
age NUMERIC(3)
);
```

Contrainte d'intégrité

Complément

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

Suppression de table

Syntaxe

```
DROP TABLE <nom de la table>;
```

Exemple

```
DROP TABLE personne;
```

Complément

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

Exercice : Appliquer la notion

[solution n°2 p. 37]

```sql
CREATE TABLE voiture (
marque TEXT,
num_immat TEXT,
kilometrage NUMERIC,
nb_portes INTEGER
);
```

Que réalise l'instruction SQL suivante ?

Elle crée une table nommée [ ], comportant une [ ] et un numéro d'immatriculation (tout deux de type [ ]), un kilométrage (de type [ ]) et un nombre de [ ] (de type `INTEGER`).

VI Domaines de données

Objectif

- Savoir les domaines pré-définis pour les attributs des tables.

Mise en situation

Supposez que vous êtes sur le point de créer une base de données pour gérer des événements sportifs. Dans la table événement, vous allez avoir plusieurs attributs, comme la date et l'heure de l'événement, son nom, son prix, son lieu, etc.

Si vous devez faire des calculs sur la date, par exemple pour programmer un rappel une semaine avant, il faut pouvoir distinguer le jour, le mois et l'année, et connaître les règles de calcul sur les dates.

Or, si on se représente mentalement ce que doit être une date, une heure ou un prix, comment indiquer au langage SQL la nature des données, pour qu'il *« comprenne »* comment faire des calculs par exemple ?

C'est grâce aux **types standards SQL**, que vous allez découvrir dans ce module.

Un attribut d'une relation est défini pour un certain domaine ou type. Les types de données disponibles en SQL varient d'un *SGBD* p.45 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")
- `TIME` ("HH:MM:SS")
- `DATETIME` ("AAAA-MM-JJ HH:MM:SS")

Les types numériques standard

Complément

- **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 maximale de la chaîne.

- `CHAR` définit des chaînes de longueur fixe (complétée à droites par des espaces, si la longueur est inférieure à X) ;
- `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`".

Attention

`DATETIME` n'existe pas sous PostgreSQL (on utilise TIMESTAMP à la place).

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 Object) pour représenter des données binaires tels que des documents multimédia (images bitmap, vidéo, etc.)
- Etc.

VII Exercice : Appliquer la notion

Question

[solution n°3 p. 37]

Un ticket de cinéma contient les information suivante :

- du nom film (tronqué à 24 caractères),
- un numéro de salle,
- un prix,
- une date de la séance,
- une heure de la séance.

Écrire la requête SQL permettant de créer cette table dans la base de données.

VIII La valeur NULL

Objectif

- Savoir représenter l'absence de valeur en SQL.

Mise en situation

Dans une table, il est possible qu'un attribut n'ait pas valeurs pour certains enregistrements.

Imaginez par exemple une table qui stocke des informations sur des festivals de musique, passés et à venir.

Au moment où un festival s'organise, le lieu n'est pas forcément connu : ça ne doit pas empêcher l'enregistrement du festival, dont le lieu n'aura pas de valeur jusqu'à ce qu'il doit décidé.

Mais comment représenter l'absence de valeur en SQL ? Avec une case vide ? Un zéro ? Un caractère particulier ? Dans ce module, vous verrez que SQL possède une valeur spéciale, appelée valeur nulle, pour représenter l'absence de valeur.

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 caractères *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
);
```

Exemple

```sql
CREATE TABLE book (
title VARCHAR(255),
subtitle INTEGER,
author VARCHAR(255),
year INTEGER
);

INSERT INTO book VALUES ('Odyssée', NULL, 'Homère', -800);
INSERT INTO book (title) VALUES ('Tristan et Iseut')  ;
```

```sql
SELECT *
FROM book;
```

title	subtitle	author	year
Odyssée	null	Homère	-800
Tristan et Iseut	null	null	null

Remarque

On peut insérer une valeur nulle de façon explicite en « *valuant* » une colonne avec `NULL` ou bien implicitement en ne *valuant* pas du tout la colonne.

IX Exercice : Appliquer la notion

Question

[solution n°4 p. 37]

Une adresse postale est composée :

- d'un numéro de l'adresse - un entier nécessairement renseigné,
- d'un indice de répétition - une chaîne de caractères,
- d'un nom de la voie - une chaîne de caractères nécessairement renseignée,
- d'un code postal - un entier nécessairement renseigné,
- d'un nom officiel de la commune - une chaîne de caractères nécessairement renseignée,
- d'un complément du nom de la voie - une chaîne de caractères.

Écrire la requête SQL permettant de créer à la table associée.

On supposera que les chaînes de caractères ont une longueur maximale de 50.

X Contraintes d'intégrité

Objectifs

- Connaître les différentes contraintes de cohérence sur les données ;
- Connaître la syntaxe SQL pour ajouter des contraintes.

Mise en situation

Lors de la création d'une table, les attributs et leurs types ne suffisent pas à exprimer les contraintes qui existent sur les données.

Imaginez par exemple une table qui stocke des relevés de température pour différentes villes, chaque jour. On aura une table qui stocke les relevés, et une table qui stocke les villes.

Dans cette situation, il y a des contraintes fortes : il ne peut pas y avoir plus d'un relevé par jour pour une ville et il faut que la ville concernée existe dans la base.

Alors comment exprimer ces obligations que doivent respecter les données insérées, et qui assurent leur cohérence ? C'est ce que vous allez découvrir dans ce module.

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* p.45.

On définit généralement des contraintes sur les tables au moment de leur création.

Contraintes d'intégrité

Az 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,
nom_colonne2 domaine2,
...
nom_colonneN domaineN,
<contraintes de table>
);
```

Exemple

```
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)
);
```

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`, `prenom` ne peuvent pas être `null` et `(nom, prenom)` 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`.

Clé candidate

Méthode

La clause `UNIQUE` associée à `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).

Contraintes d'intégrité sur une colonne

Complément

Lorsque la contrainte ne s'applique que sur une colonne, il existe une syntaxe dédiée

- `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.

Réécriture avec uniquement des contraintes de colonnes

Exemple

```sql
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)
);
```

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

XI Exercice : Appliquer la notion

Question

[solution n°5 p. 38]

Soit la création d'une table nommée `voiture`, comportant une marque et un numéro d'immatriculation (tous deux de type `TEXT`), un kilométrage (un réel de type `NUMERIC`) et un nombre de portes (un entier, `INTEGER`) :

```
CREATE TABLE voiture (
marque TEXT,
num_immat VARCHAR(15),
kilometrage NUMERIC,
nb_portes INTEGER
);
```

On pose que :

- le numéro d'immatriculation de la voiture est la clé primaire,
- que le nombre de portes est compris entre 3 et 5,
- et que le kilométrage est positif.

Modifier l'instruction pour y inclure ces nouvelles contraintes.

Indice :

On pourra utiliser :

```
CHECK (nom_colonne BETWEEN inf AND sup)
```

XII Insertion de données

Objectifs

- Savoir insérer des données dans une table ;
- Connaître le format des données à insérer en fonction des types.

Mise en situation

Une fois les tables d'une base de données créées et leurs contraintes bien définies, vous allez enfin pouvoir stocker les données à proprement parler.

SQL fournit une syntaxe puissante pour ce stockage, appelée insertion, permettant d'insérer un grand lot de données d'un seul coup et même de choisir quels sont les attributs que vous voulez renseigner.

En revanche, il y a des contraintes à respecter sur le format des données, en fonction de leur type : une date s'écrit différemment d'une chaîne de caractères et d'un nombre.

Introduction

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

Insertion 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

```
CREATE TABLE transfer (
number INTEGER NOT NULL,
amount INTEGER NOT NULL,
object TEXT,
PRIMARY KEY (number)
);

INSERT INTO transfer (number, amount, object)
VALUES (1, 1000, 'Prime de naissance');
```

```
SELECT *
FROM transfer;
```

```
| number | amount | object             |
| ------ | ------ | ------------------ |
| 1      | 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).

```
INSERT INTO transfer (number, amount)
VALUES (2, 200);
```

```
SELECT *
FROM transfer;
```

```
| number | amount | object             |
| ------ | ------ | ------------------ |
| 1      | 1000   | Prime de naissance |
| 2      | 200    |                    |
```

Chaînes de caractères

Méthode

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

Exemple

```
CREATE TABLE book (
  title VARCHAR(255),
  PRIMARY KEY (title)
);

INSERT INTO book (title)
VALUES ('L''Attrape-cœurs');
```

Dates

Méthode

La saisie des dates peut se faire de plusieurs façons dépendantes du SGBD. La méthode la plus simple consiste à entrer la date comme une chaîne au format « *YYYY-MM-DD* » (4 chiffres de l'année, 2 chiffres du mois, 2 chiffres du jour).

Il est plus standard de faire précéder cette notation du mot-clé DATE.

Exemple

```
CREATE TABLE book (
  title VARCHAR(255),
  publication DATE,
  PRIMARY KEY (title)
);

INSERT INTO book (title, publication)
```

```
8 VALUES ('As we may think', DATE '1945-07-14');
```

XIII Exercice : Appliquer la notion

Question

[solution n°6 p. 38]

On se donne la table créée via :

```
CREATE TABLE ticket (
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);
```

De plus, on dispose des données de films suivants.

nom_film	num_salle	prix	date_projection	horaire_projection
I origins	12	7.90	20 Mars 2020	13:37
Drive	34	8.90	14 Avril 2020	21:30
Mr Nobody	7	6.90	4 Mai 2020	10:20

Après avoir créé la table, écrire les requêtes permettant d'insérer ces enregistrements dans la base de données.

Indice :

On peut vérifier les résultats avec la commande :

```
SELECT * FROM ticket;
```

XIV Suppression de données

Objectif

- Savoir supprimer des données d'une table en SQL.

Mise en situation

Même si cela peut paraître étrange, il est pourtant utile de pouvoir supprimer certaines données d'une table.

Imaginez par exemple une base de données qui gère les abonnés à un magazine : lorsqu'une personne se désabonne, on va tout simplement la supprimer de la table abonnements.

Mais un enregistrement dans une table n'est identifié que par son contenu : on ne pourra pas exprimer quelque chose comme *« supprimer le client à la ligne 5 »*. Il faut alors trouver un critère de suppression.

SQL fournit la syntaxe pour supprimer les données qui répondent à une ou plusieurs conditions, et c'est ce que vous allez découvrir dans ce module.

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

Suppression de données

Syntaxe

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

Suppression sélective

Exemple

```
CREATE TABLE fausses_factures (
auteur VARCHAR(255) PRIMARY KEY,
valeur INTEGER
);

INSERT INTO fausses_factures VALUES ('moi', 100);
INSERT INTO fausses_factures VALUES ('toi', 200);
INSERT INTO fausses_factures VALUES ('lui', 500);

DELETE FROM fausses_factures
WHERE auteur='moi';
```

Suppression de tous les tuples d'une relation

Exemple

```
DELETE FROM fausses_factures;
```

XV Exercice : Appliquer la notion

Question

[solution n°7 p. 38]

On se donne la table créée et les enregistrements associés corrigés :

nom_film	num_salle	prix	date_projection	horaire_projection
I Origins	12	7.90	20 Mars 2020	13:37
Drive	34	18.90	14 Avril 2020	21:30
Mr Nobody	47	6.90	4 Mai 2020	10:20
Fight Club	5	7.90	7 Mars 2020	10:20

```sql
CREATE TABLE ticket (
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);

INSERT INTO ticket VALUES ('I Origins', 12, 7.90, '2020-03-20', '13:37');
INSERT INTO ticket VALUES ('Drive', 34, 18.90, '2020-04-14', '21:30');
INSERT INTO ticket VALUES ('Mr Nobody', 47, 6.90, '2020-05-04', '10:20');
INSERT INTO ticket VALUES ('Fight Club', 5, 7.90, '2020-03-07', '10:20');
```

Écrire une requête pour supprimer le film *« Fight Club »*, dont la projection a été annulée au dernier moment.

XVI Mise à jour de données

Objectif

- Savoir mettre à jour les données d'une table.

Mise en situation

Les informations stockées dans une base de données sont amenées à évoluer.

Imaginez par exemple une table qui stocke le montant disponible sur le compte en banque des clients d'une banque : ce montant évolue et nécessite constamment d'être mis à jour.

Si une première solution pourrait consister à supprimer la valeur précédente et à insérer les données mises à jour, il existe une syntaxe SQL plus puissante qui permet de mettre à jour les valeurs d'attributs spécifiques, et même de modifier plusieurs enregistrements d'un seul coup.

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

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>;
```

Mise à jour directe de valeurs

Exemple

```
CREATE TABLE compte (
iban VARCHAR(34) PRIMARY KEY,
solde INTEGER,
monnaie VARCHAR(10)
);

INSERT INTO compte VALUES ('DZkk BBBS SSSS CCCC CCCC CCKK', 1024, 'Franc');
INSERT INTO compte VALUES ('DEkk BBBB BBBB CCCC CCCC CC', 2048, 'Franc');
INSERT INTO compte VALUES ('ADkk BBBB SSSS CCCC CCCC CCCC', 4096, 'Dollar');

UPDATE compte
SET monnaie = 'Euro'
WHERE monnaie = 'Franc';
```

Mise à jour par calcul sur l'ancienne valeur

Exemple

```
UPDATE compte
SET solde = solde * 6.55957
WHERE monnaie = 'Euro';
```

XVII Exercice : Appliquer la notion

Question

[solution n°8 p. 39]

On se donne la table ticket et les enregistrement associés :

```sql
CREATE TABLE ticket (
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);

INSERT INTO ticket
VALUES('I origins', 12, 7.90, '2020-03-20', '13:37');

INSERT INTO ticket
VALUES('Drive', 34, 8.90, '2020-04-14', '21:30');

INSERT INTO ticket
VALUES('Mr Nobody', 7, 6.90, '2020-05-04', '10:20');
```

nom_film	num_salle	prix	date_projection	horaire_projection
I origins	12	7.90	20 Mars 2020	13:37
Drive	34	8.90	14 Avril 2020	21:30
Mr Nobody	7	6.90	4 Mai 2020	10:20

On vient de se rendre compte que la table comporte de mauvais enregistrements et que l'on devrait à la place avoir :

nom_film	num_salle	prix	date_projection	horaire_projection
I Origins	12	7.90	20 Mars 2020	13:37
Drive	34	**18.90**	14 Avril 2020	21:30
Mr Nobody	**47**	6.90	4 Mai 2020	10:20

Écrire les requêtes de mise à jour des tables.

XVIII Essentiel

SQL est un langage puissant et standard, utilisable avec toutes les bases de données relationnelles.

Il fournit les éléments de syntaxe pour réaliser toutes les opérations classiques.

D'abord, il prévoit tous les cas de manipulations de la structure des tables : définition, avec `CREATE TABLE`, modification, avec `ALTER TABLE`, suppression, avec `DROP TABLE`, et ajout de contraintes sur les attributs, avec `PRIMARY KEY` et `UNIQUE` par exemple.

Ensuite, il permet de manipuler les données à proprement parler : insertion avec `INSERT INTO`, mise à jour avec `UPDATE`, et suppression avec `DELETE`.

Toutes ces instructions garantissent la cohérence de la base, en interdisant par exemple l'insertion de données qui ne respectent pas la structure et les contraintes des tables.

XIX Quiz

Exercice 1 : Quiz - Culture

[solution n°9 p. 39]

Exercice

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

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

Exercice

Parmi les types SQL suivants, lesquelles sont des domaines d'entiers ?

- [ ] **A** INTEGER
- [ ] **B** TEXT
- [ ] **C** REAL
- [ ] **D** INT
- [ ] **E** DATE

Exercice

Parmi les types SQL suivants, lesquels sont des domaines de nombres réels ?

- **A** INTEGER
- **B** TIME
- **C** REAL
- **D** INT
- **E** DOUBLE
- **F** FLOAT

Exercice

Parmi les types SQL suivants, lesquels sont des domaines de chaîne de caractères ?

- **A** VARCHAR
- **B** TIME
- **C** TEXT
- **D** INT
- **E** DOUBLE
- **F** CHAR

Exercice 6 : Quiz - Méthode

[solution n°10 p. 41]

Exercice

On peut ajouter des contraintes d'intégrité :

- **A** en définissant des contraintes de colonnes lors de la création d'une table.
- **B** en définissant des contraintes de tables lors de sa création.
- **C** en définissant des contraintes après la création de la table.

Exercice

On peut toujours se ramener à des contraintes de table à partir de contraintes de colonnes.

- **A** Vrai
- **B** Faux

Exercice

On peut toujours se ramener à des contraintes de table à partir de contraintes de colonnes.

- **A** Vrai
- **B** Faux

Exercice 10 : Quiz - Code

[solution n°11 p. 41]

Exercice

Parmi les propositions de requêtes ci-dessous, laquelle est exécutable ?

A

```
CREATE  TABLE identifiant    date_depart     empreinte_co2    ) ;
trajet (     PRIMARY KEY,    DATE NOT NULL,  REAL NOT NULL
```

B

```
CREATE     identifiant date_depart   empreinte_co2   REAL   NOT );
TABLE      INTEGER,    DATE      NOT NULL  CHECK  (empreinte_co2
trajet (               NULL,         > 0)
```

C

```
CREATE   TABLE identifiant    date_depart   DATE empreinte_co2   );
trajet(        INTEGER,       NOT NULL,          REAL > 0
```

Exercice

On se donne la table suivante :

```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))
);
```

Dans les requêtes suivantes, lesquelles comportent des erreurs ?

- **A** `INSERT INTO adresse VALUES (62940, 'FRANCE', 'F');`

B
```
INSERT INTO adresse (cp, pays, initiale) VALUES (0, 'CHINE', 'C');
```

C
```
INSERT INTO adresse VALUES (41894, 'IRLANDE', 'I');
```

D
```
INSERT INTO adresse VALUES (39104, 'Angleterre', 'F');
```

E
```
INSERT INTO adresse (pays, initiale) VALUES ('FRANCE', 'F');
```

F
```
INSERT INTO adresse VALUES (62941, 'FRANCE', 'F');
```

Exercice

Le code de création de la base de données de gestion de film suivante est-il correct ?

```
CREATE TABLE ticket (
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);
```

```
INSERT INTO ticket
VALUES ('I Origins', 12, 6.90, '2020-02-03', '10:30');

INSERT INTO ticket
VALUES ('Drive', 47, 8.90, '2020-08-15', '13:45');

INSERT INTO ticket
VALUES ('Mr Nobody', 12, 16.90, '2020-05-04', '7:20');
```

A Oui, il ne renvoie aucune erreur.

B Non, il y a des erreurs de syntaxe.

Exercice

Soit la base de donnée suivante :

```
CREATE TABLE ticket (
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);
```

```
INSERT INTO ticket
VALUES ('Drive', 47, 8.90, '2020-08-15', '13:45');
```

Que renvoie l'instruction :

```
SELECT nom_film
FROM ticket;
```

Solutions des exercices

Solution n°1

[exercice p. 7]

Exercice

Quel est le nom du projet original sur lequel se base PostgreSQL ?

- **A** Ingres
- **B** MySQL
- **C** POSTGRES
- **D** SQLite

Exercice

Parmi les propositions suivantes, quelles fonctionnalités SQL sont présentes dans PostgreSQL ?

- **A** Transactions
- **B** Table
- **C** Vues
- **D** Clés étrangères
- **E** Indexage
- **F** Triggers
- **G** Procédure

Exercice

PostgreSQL peut être :

- **A** Librement et gratuitement utilisée
- **B** Modifiée
- **C** Redistribuée

D Revendue

Solution n°2

[exercice p. 11]

```
CREATE TABLE voiture (
marque TEXT,
num_immat TEXT,
kilometrage NUMERIC,
nb_portes INTEGER
);
```

Que réalise l'instruction SQL suivante ?

Elle crée une table nommée `Voiture`, comportant une `marque` et un numéro d'immatriculation (tout deux de type `TEXT`), un kilométrage (de type `NUMERIC`) et un nombre de `portes` (de type INTEGER).

Solution n°3

[exercice p. 15]

```
CREATE TABLE ticket(
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);
```

⊕ Complément

Une requête non standard peut prendre en compte le prix avec le type dédié MONEY :

```
CREATE TABLE ticket(
nom_film VARCHAR(24)
num_salle INTEGER,
prix MONEY,
date_projection DATE,
horaire_projection TIME
);
```

Solution n°4

[exercice p. 18]

Une requête pour réaliser cela peut être :

```
CREATE TABLE adressePostale(
numero INTEGER NOT NULL,
indice_rep VARCHAR(50),
nom_voie VARCHAR(50) NOT NULL,
code_postal INTEGER NOT NULL,
commune VARCHAR(50) NOT NULL,
complement VARCHAR(50)
);
```

Base Adresse Nationale ⊕ Complément

https://adresse.data.gouv.fr/docs/BAN_Descriptif_Donnees.pdf

Solution n°5

[exercice p. 22]

```sql
CREATE TABLE voiture (
marque TEXT,
num_immat VARCHAR(15),
kilometrage NUMERIC,
nb_portes INTEGER,
PRIMARY KEY(num_immat),
CHECK (kilometrage > 0),
CHECK (nb_portes BETWEEN 3 AND 5)
);
```

Solution n°6

[exercice p. 26]

On a les requêtes suivantes :

```sql
INSERT INTO ticket
VALUES('I origins', 12, 7.90, DATE '2020-03-20', '13:37');

INSERT INTO ticket
VALUES('Drive', 34, 8.90, DATE '2020-04-14', '21:30');

INSERT INTO ticket
VALUES('Mr Nobody', 7, 6.90, DATE '2020-05-04', '10:20');
```

```
| nom_film  | num_salle | prix | date_projection          | horaire_projection |
| --------- | --------- | ---- | ------------------------ | ------------------ |
| I origins | 12        | 7.9  | 2020-03-20T00:00:00.000Z | 13:37:00           |
| Drive     | 34        | 8.9  | 2020-04-14T00:00:00.000Z | 21:30:00           |
| Mr Nobody | 7         | 6.9  | 2020-05-04T00:00:00.000Z | 10:20:00           |
```

Solution n°7

[exercice p. 28]

On peut avoir les requêtes suivantes :

```sql
DELETE FROM ticket
WHERE nom_film = 'Fight Club';
```

```
| nom_film  | num_salle | prix | date_projection | horaire_projection |
| --------- | --------- | ---- | --------------- | ------------------ |
| I Origins | 12        | 7.9  | 2020-03-20      | 13:37:00           |
| Drive     | 34        | 18.9 | 2020-04-14      | 21:30:00           |
| Mr Nobody | 47        | 6.9  | 2020-05-04      | 10:20:00           |
```

Solution n°8

[exercice p. 30]

On a les requêtes suivantes :

```sql
UPDATE ticket
SET nom_film = 'I Origins'
WHERE nom_film = 'I origins';

UPDATE ticket
SET prix = 18.90
WHERE nom_film = 'Drive';

UPDATE ticket
SET num_salle = 47
WHERE nom_film = 'Mr Nobody';
```

Solution n°9

[exercice p. 32]

Exercice

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

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

Exercice

Parmi les types SQL suivants, lesquelles sont des domaines d'entiers ?

- **A** INTEGER

B TEXT

C REAL

D INT

E DATE

Exercice

Parmi les types SQL suivants, lesquels sont des domaines de nombres réels ?

A INTEGER

B TIME

C REAL

D INT

E DOUBLE

F FLOAT

Exercice

Parmi les types SQL suivants, lesquels sont des domaines de chaîne de caractères ?

A VARCHAR

B TIME

C TEXT

D INT

E DOUBLE

F CHAR

Solution n°10

[exercice p. 33]

Exercice

On peut ajouter des contraintes d'intégrité :

A en définissant des contraintes de colonnes lors de la création d'une table.

B en définissant des contraintes de tables lors de sa création.

C en définissant des contraintes après la création de la table. Cela est possible grâce aux modifications de tables avec ALTER.

Exercice

On peut toujours se ramener à des contraintes de table à partir de contraintes de colonnes.

A Vrai

B Faux

Cela est toujours possible, il suffit de déplacer la définition des contraintes de colonnes à la fin de la définition de la table.

Exercice

On peut toujours se ramener à des contraintes de table à partir de contraintes de colonnes.

A Vrai

B Faux

Prendre l'exemple de la définition de la contrainte l'unicité des valeurs de deux colonnes d'une table ne peut se faire que via une contrainte de table.

Solution n°11

[exercice p. 34]

Exercice

Parmi les propositions de requêtes ci-dessous, laquelle est exécutable ?

A

```
CREATE TABLE trajet ( identifiant PRIMARY KEY, date_depart DATE NOT NULL, empreinte_co2 REAL NOT NULL ) ;
```

Il manque le domaine pour identifiant.

B ◯

```
CREATE TABLE trajet (
    identifiant INTEGER,
    date_depart DATE NOT NULL,
    empreinte_co2 REAL NOT NULL CHECK (empreinte_co2 > 0)
);
```

C

```
CREATE TABLE trajet(
    identifiant INTEGER,
    date_depart DATE NOT NULL,
    empreinte_co2 REAL > 0
);
```

La contrainte n'est pas correctement exprimée.

Exercice

On se donne la table suivante :

```
CREATE TABLE adresse (
  cp INTEGER NOT NULL,
  pays VARCHAR(50) NOT NULL,
  initiale CHAR(1),
  PRIMARY KEY (cp, pays),
  CHECK (initiale = LEFT(pays, 1))
);
```

Dans les requêtes suivantes, lesquelles comportent des erreurs ?

A

```
INSERT INTO adresse VALUES (62940, 'FRANCE', 'F');
```

B

```
INSERT INTO adresse (cp, pays, initiale) VALUES (0, 'CHINE', 'C');
```

C

```
INSERT INTO adresse VALUES (41894, 'IRLANDE', 'I');
```

D ☐

```
INSERT INTO adresse VALUES (39104, 'Angleterre', 'F');
```

La contrainte sur l'initiale n'est pas vérifiée.

E ☐

```
INSERT INTO adresse (pays, initiale) VALUES ('FRANCE', 'F');
```

Le code postal ne peut pas être nul.

F

```
INSERT INTO adresse VALUES (62941, 'FRANCE', 'F');
```

Exercice

Le code de création de la base de données de gestion de film suivante est-il correct ?

```
CREATE TABLE ticket (
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);
```

```sql
INSERT INTO ticket
VALUES ('I Origins', 12, 6.90, '2020-02-03', '10:30');

INSERT INTO ticket
VALUES ('Drive', 47, 8.90, '2020-08-15', '13:45');

INSERT INTO ticket
VALUES ('Mr Nobody', 12, 16.90, '2020-05-04', '7:20');
```

A Oui, il ne renvoie aucune erreur.

B Non, il y a des erreurs de syntaxe.

Exercice

Soit la base de donnée suivante :

```sql
CREATE TABLE ticket (
nom_film VARCHAR(24),
num_salle INTEGER,
prix FLOAT(2),
date_projection DATE,
horaire_projection TIME
);
```

```sql
INSERT INTO ticket
VALUES ('Drive', 47, 8.90, '2020-08-15', '13:45');
```

Que renvoie l'instruction :

```sql
SELECT nom_film
FROM ticket;
```

Drive

Drive avec un D majuscule est la bonne réponse, car les données sont sensibles à la casse.

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[2]

2. http://pic.crzt.fr/dbdisco/

Bibliographie

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

Index