Chapitre V

Les bases de données relationnelles en pratique : Mise en oeuvre et utilisation

Mise en oeuvre et utilisation d'une base de données : points principaux

- Organisation d'un système de gestion de bases de données ;
- Définition de bases de données et de relations ;
- Contraintes ;
- Définition de vues.

Organisation d'un système de gestion de bases de données

Un système de gestion de bases de données est organisé en plusieurs programmes qui sont exécutés séparément (dans des processus distincts). On distingue

- Un serveur qui gère les données et attend des requêtes SQL, les exécute et en renvoie le résultat,

- Une ou plusieurs applications qui interagissent avec le serveur. Une application peut être :
  - Une simple interface permettant de taper des commandes SQL,
  - Une interface orientée vers la gestion de la base de données,
  - Une interface permettant de d'interroger et de modifier la base de données suivant des opérations prédéfinies.

Un utilisateur peut être

– en interaction directe avec l'application (architecture à deux niveaux), ou

– en interaction avec un troisième programme qui dialogue avec l'application, par exemple un client web interagissant avec une application exploitée dans le contexte d'un serveur web (architecture à trois niveaux).

Note : le serveur et les applications peuvent être exécutés sur des machines différentes

Définition de bases de données et de relations

Un système de gestion de bases de données comme MySQL peut gérer plusieurs bases de données.

– Une base de données est un ensemble de tables géré de façon indépendante et possédant ses propres autorisations d'accès.

– Une requête ne peut pas utiliser des tables de bases de données différentes.

Pour définir une base de données, on peut utilise une interface de gestion, ou des commandes SQL.

create database **drop database**

On précise la base de données à utiliser avec la commande **use**.

Définition de tables (relations)

Lorsque l'on crée une relation, l'information sur le schéma de la relation est conservé dans un *dictionnaire de données* (*data dictionary*).

Il existe des commandes SQL pour gérer les relations figurant dans une base de données.

Création d'une table : **create table** r
$(\cdots, \cdots, \cdots)$

Exemple :

create table *fournisseurs*
(*NOM_F* varchar(20) **not null**,
ADRESSE_F varchar(50) **not null**,
COMB varchar(10),
PRIX int)

Domaines des attributs

Domaines d'attributs prédéfinis :
- Nombres entiers : *int* ou *integer* et *smallint*
- Nombres en virgule flottante : *float*, *real* et *double precision*
- Nombres en virgule fixe : *decimal(i,j)* ou *dec(i,j)* ou *numeric(i,j)*
- Chaînes de caractères : *char(n)* ou *character(n)*, et *varchar(n)* ou *char varying(n)* ou *character varying(n)*
- Chaînes de bits (de booléens) : *bit(n)* et *bit varying(n)*

Domaines d'attributs nouveaux dans SQL2
- Dates : *date* (10 car. 'aaaa-mm-jj')
- Heures : *time* (8 car. 'hh :mm :ss') et *time(i)* ($i + 9$ car. 'hh :mm :ss :f_1. . .f_i') et *time with time zone*
- Dates + heures : *timestamp*
- Intervalles de temps (que l'on peut ajouter ou soustraire à une date, un temps ou un timestamp : *interval*

Définition de domaines :

create domain *EMP_ID_TYPE* **as** char(11) ;

Exemples :

EMPLOYEE (EMP_ID, FNAME, LNAME, BDATE, ADDRESS, SALARY, DEPT_NO)
DEPARTMENT (DNO, DNAME, MGR_ID

```
create table EMPLOYEE
        ( EMP_ID      char(11)       not null,
          FNAME       varchar(15)    not null,
          LNAME       varchar(15)    not null,
          BDATE       date,
          ADDRESS     varchar(30),
          SALARY      decimal(10,2),
          DEPT_NO     int            not null,
        primary key (EMP_ID),
        foreign key (DEPT_NO)   references DEPARTMENT (DNO) ) ;
```

```
create table DEPARTMENT
        ( DNO          int             not null,
          DNAME        varchar(15)     not null,
          MGR_ID       char(11),
          MGR_START    date,
        primary key (DNO),
        unique (DNAME),
        foreign key (MGR_ID)   references EMPLOYEE (EMP_ID) ) ;
```

Contraintes d'intégrité

Un certain nombres d'indications figurant dans la définition d'une relation sont des contraintes d'intégrité sur les données qui peuvent y figurer. On distingue

- Les contraintes *locales* qui ne concernent que le relation définie,
- Les contraintes entre relations ou contraintes de *référence* qui impliquent plus d'une relation.

Contraintes d'intégrité locales

not null impose qu'un attribut ne puisse pas être sans valeur (valeur nulle).

default : spécifie une valeur par défaut

primary key $A_1, \ldots A_n$ **:** spécifie un ensemble d'attributs constituant une clé, et plus précisément une clé que l'on a choisi comme la clé préférentielle : la *clé primaire*.

unique $A_1, \ldots, A_n$ **:** spécifie un ensemble d'attributs constituant une clé (autre que la clé primaire).

Contraintes d'intégrité référentielles et clés étrangères (foreign keys)

Un tuple dans une relation qui fait référence à un tuple d'une autre relation, doit faire référence à un tuple *existant* dans cette relation.

Soit R_1 et R_2 deux schémas de relation

$K = \{A_1, \ldots, A_n\} \subseteq R_1$

K est la clé (primaire) de R_1

$FK = \{B_1, \ldots, B_n\} \subseteq R_2$

$dom(A_i) = dom(B_i)$ pour $1 \leq i \leq n$

FK est une *clé étrangère* de R_2 si pour tout $t(R_2)$ de r_2

- soit il existe dans r_1 un tuple $s(R_1)$ tel que $t[FK] = s[K]$ (t fait référence à s)
- soit $t[FK]$ n'a pas de valeur (valeur nulle).

Autres contraintes d'intégrité

Les contraintes d'intégrité sur les données, peuvent aller bien au delà des contraintes simples ci-dessus :

- dépendances, contraintes de cardinalité,
- contraintes sur les valeurs de un ou plusieurs attributs,....

Ces contraintes peuvent être vérifiées au niveau de l'application ou de la base de données. Dans ce dernier cas, la base de donnée doit disposer des possibilités suivantes :

- Définition d'assertions (requêtes dont le résultat doit toujours être vrai),
- Procédures de la base de données (*stored procedures*),
- Possibilité de déclencher ces procédures lorsque certains évènements surviennent (*triggers*).

La définition de vues

Une *vue* est une relation dérivée d'autres relations enregistrées dans la base de données.

create view $v(A_1, \ldots, A_k)$ **as** $\mathcal{Q}$

où $\mathcal{Q}$ est une requête SQL.

Exemple :

create view *tout-brule-vend*(*COMB*, *PRIX*) **as**
select *COMB*, *PRIX*
from *fournisseurs*
where *NOM_F='Tout-Brule'*

Les vus permettent de fournir une vue partielle des données et d'imposer des restrictions d'accès. Les mises à jour peuvent rarement se faire au niveau de la vue.