POLYTECH LILLE

Université Lille 1 Sciences et Technologies

SQL (Première partie)

Walter RUDAMETKIN

Bureau F011

Walter.Rudametkin@polytech-lille.fr

Les commandes de base sous Unix

Création d'une base ([] facultatif) :

- `createdb nomBase [ -U comptePostgres ]`

Destruction d'une base :

- `dropdb nomBase [ -U comptePostgres ]`

Accès à une base :

- `psql nomBase [ -U comptePostgres ]`

SQL : Structured Query Language

Historique :

- En 70, Travaux de Codd
- De 72 à 75, IBM invente SEQUEL.
- Puis SEQUEL/2 en 77 pour le prototype System-R de SGBDR.
- Ce langage donne naissance à SQL
- Parallèlement, Ingres développe le langage QUEL en 76
- Dès 79, Oracle utilise SQL
- En 81, IBM sort SQL/DS (même année que le PC)
- En 83, IBM sort DB2 (héritier de System-R) exploitant SQL

Le language SQL (2/2)

- Standard de fait, puis véritable standard - **Norme ISO**
- Facile à utiliser car il se réfère toujours aux lignes ou colonnes d'une table
- Objectif des SGBD : applicable aux non-informaticiens ?
- Langage tout en un : consultation mais aussi création, modification, suppression, . . .
- Langage en constante évolution : SQL-86, SQL-89, SQL-92 (SQL 2),
- SQL-99 (SQL 3, non terminé)

Syntaxe des commandes

Les commandes **commencent** par un **mot clé** servant à **nommer l'opération** de base à exécuter.

Chaque commande SQL doit remplir deux exigences :

- Indiquer les données sur lesquelles elle opère (un ensemble de lignes stockées dans une ou plusieurs classes)
- Indiquer l'opération à exécuter sur ces données

Les domaines par défaut

Les **numériques** : integer, smallint, double, float,...

- exemple : 23, -122, 3.4, -6.7,. . .

Les **alphanumériques** : char, varchar(n), char(n), text

- exemples : 'v', 'la vie est un long fleuve ...'

Le **type date** (format configurable)

- exemple : '2002-02-28'

Le type **boolean** : bool

- exemples :'f', 't'

Et bien d'autres encore...

Distinction des commandes

Les **commandes d'administration** (utilisateur-administrateur) :

- création, suppression de tables, *d'index, de vues, de droits d'accès, de fonctions ou procédures, modification de structure de tables.*

Les **commandes d'utilisation :**

- consultation, modification de lignes, suppression de lignes.

Création de tables

Syntaxe postgres (très simplifiée)

- [] : 0 ou 1 occurrence,
- {} 0 ou plusieurs occurrences,
- | : alternative

CREATE TABLE table_name (table_element {, table_element} {, table_constraint })

< table_element> ::= <column_name> <type>
[<column_constraint>*]

type ::= VARCHAR <longueur> | INT | REAL | DATE …

Chaque relation est définie par un nom de relation et une liste d'attributs

Chaque attribut est défini par un nom d'attribut et un type de données

Contrainte d'attribut

column_constraint ::

```
[ CONSTRAINT constraint_name ]
PRIMARY KEY | REFERENCES table_name [ ( column [, ... ] ) ]
```

- Concerne un seul attribut
- valeur NULL impossible : NOT NULL
- Attribut clé : PRIMARY KEY
- Unicité de l 'attribut : UNIQUE
- Valeur par défaut : DEFAULT <valeur>
- Contrainte référentielle : REFERENCES <relation référencée> [(<attribut référencé>)]
- Valeur par défaut : CHECK <condition>

Contrainte de table

table_constraint::

```
[ CONSTRAINT constraint_name ]
PRIMARY KEY ( column_name [, ... ] ) |
FOREIGN KEY ( column_name [, ... ] )
REFERENCES table_name [ ( column [, ... ] ) ]
```

Concerne plusieurs attributs

- Clé composée :
  - PRIMARY KEY (nom_attribut [, nom_attribut]*)
- Contrainte référentielle :
  - FOREIGN KEY (nom_attribut [, nom_attribut]*)
  - REFERENCES nom de relation [(nom_attribut [, nom_attribut]*)]

Exemple de création de tables (1/3)

```
CREATE TABLE utilisateur (
  num_u INTEGER PRIMARY KEY,
  nom VARCHAR(30), prenom VARCHAR(30)
) ;
CREATE TABLE auteur (
  num_a INTEGER,
  nom VARCHAR(30),
  CONSTRAINT cle_auteur PRIMARY KEY (num_a)
) ;
```

Ex. de création de tables (2/3)

```
CREATE TABLE editeur (
  num_e INTEGER PRIMARY KEY,
  nom VARCHAR(30), adresse1 VARCHAR(30),
  adresse2 VARCHAR(30), code_postal integer, ville
  VARCHAR(30)
) ;
CREATE TABLE livre (
  num_l INTEGER,
  titre text,
  n_auteur INTEGER REFERENCES auteur,
  PRIMARY KEY (num_l)
) ;
```

Ex. de création de tables (3/3)

```
CREATE TABLE reserve (
  num_l INTEGER REFERENCES livre,
  num_u INTEGER REFERENCES utilisateur,
  PRIMARY KEY (num_l, num_u)
) ;
CREATE TABLE emprunte (
  num_l INTEGER REFERENCES livre,
  num_u INTEGER REFERENCES utilisateur,
  PRIMARY KEY (num_l, num_u)
) ;
CREATE TABLE edite_par (
  num_l INTEGER references livre,
  num_e INTEGER references editeur,
  date_edition date,
  PRIMARY KEY (num_l,num_e, date_edition)
) ;
```

Insertion de lignes

Syntaxe (simplifiée) :

```
INSERT INTO table [ ( column {, ...} ) ]
VALUES ( expression {, ...} )
```

- Les valeurs doivent être fournies **dans l'ordre de déclaration** des attributs de la liste ou, s'il n'y en n'a pas, celui défini à la création.
- Si la liste d'attributs est incomplète, les attributs non spécifiés sont insérés avec des valeurs **NULL** ou **DEFAULT**

Exemples d'insertion

```
INSERT INTO <nom_relation> [(nom_attribut {, nom_attribut} )] VALUES ( valeur {, valeur} ) ;
```

On insère une ligne complète :

- `INSERT INTO auteur VALUES (1, 'Uderzo') ;`

On insère une ligne complète en spécifiant les colonnes :

- `INSERT INTO auteur (nom, num_a) VALUES ('Franquin', 2) ;`

On insère une ligne incomplète :

- `INSERT INTO livre(num_l, titre) VALUES (1, 'L \'Odyssée d\'Astérix');`

Suppression de lignes

Syntaxe :

```
DELETE FROM table_name [ WHERE condition ]
```

Exemples :

- `DELETE FROM livres where num_l=1 ;`
- `DELETE FROM utilisateurs ;`

Modification de lignes

Syntaxe :

- `UPDATE table_name SET col = expression {, col = expression} [ WHERE condition ]`

Modifier une colonne, pour une ligne :

- `UPDATE livre SET auteur=1 WHERE num_l=1`

Modifier plusieurs colonnes pour une ligne :

- `UPDATE auteur SET nom='Victor Hugo', num_a=4 where num_a=3`

Modifier toutes les lignes :

- `UPDATE personnel SET salaire=salaire+0.10*salaire`

Modifier une table

De plus en plus de possibilités au cours des versions (ex : supprimer une colonne)

Syntaxes :

- `ALTER TABLE nom_table ADD [ COLUMN ] nom_colonne type`
- `ALTER TABLE nom_table RENAME [ COLUMN ] nom_colonne TO nouveau_nom`
- `ALTER TABLE nom_table RENAME TO nouveau_nom_table`
- `ALTER TABLE nom_table DROP [ COLUMN] nom_colonne`
- `ALTER TABLE nom_table OWNER to nouveau_proprietaire`
- …

POLYTECH LILLE

Université Lille 1 Sciences et Technologies

Définition de contraintes

Walter RUDAMETKIN

Bureau F011
Walter.Rudametkin@polytech-lille.fr

Notion de contrainte d'intégrité

Objectif : Assurer la cohérence logique de la base de données

Définition : Assertion vérifiée par les données de la base à tout moment

Classification des CIs

Structurelles

- liées au modèle relationnel
- Exemple : unicité valeur de clé, ...

Comportementales

- liées aux applications
- Exemple : « la moyenne des salaires n'est pas inférieure à 1500 »

Intra-relation

- met en jeu une seule relation
- Exemple : non nullité d'un attribut

Inter-relation

- met en jeu plusieurs relations
- Exemple : intégrité référentielle

Les types de contraintes

Normalisation SQL-92

Les *contraintes de domaine* définissent les valeurs prises par un attribut.

Les *contraintes d'intégrité d'entité* précisent la clé primaire de chaque table

Les *contraintes d'intégrité référentielle* assurent la cohérence entre les clés primaires et les clés étrangères

Contrainte de domaine (1/4)

NOT NULL :

Toute valeur de l'attribut X est connue

- `CREATE TABLE personnel (nom TEXT NOT NULL, prenom TEXT)`

- `INSERT INTO personnel(nom) VALUES ('dupont')`

  → **CORRECTE : prenom est NULL**

- `INSERT INTO personnel(prenom) VALUES('henri')`

  → **ERREUR : nom n'est pas renseigné**

Contrainte de domaine (2/4)

DEFAULT :

« L'attribut X a, par défaut, la valeur Y »

```
CREATE TABLE article (
   num INT NOT NULL,
   quantite INT DEFAULT 1,
   date_creation DATE DEFAULT now()
)
```

Note : La clause NOT NULL est implicite (pour DEFAULT) (sauf si DEFAULT NULL !)

Contrainte de domaine (3/4)

UNIQUE :

« Toutes les valeurs de l'attribut X sont différentes »

- Éviter les redondances, utile pour les clés
- `CREATE TABLE article (num INT NOT NULL UNIQUE, nom TEXT ...`

- La clé peut être constituée de plusieurs attributs :

```
CREATE TABLE reserve_par (num_client INT
NOT NULL, num_livre INT NOT NULL,
UNIQUE (num_client,num_livre))
```

Contrainte de domaine (4/4)

CHECK : spécifier une contrainte qui doit être vérifiée à tout moment par les tuples de la table :

```
CREATE TABLE personnel ( num INT NOT NULL
UNIQUE, age INT CHECK (age >= 18), sexe CHAR
DEFAULT 'F' CHECK (sexe IN ('M','F')),
ageFuturePromotion INT CHECK (ageFuturePromotion
> age))
```

- La clause CHECK peut se placer après la définition de tous les attributs.
- Il est préférable de nommer la contrainte (facultatif)
- Utilisation de sous-requêtes SQL (> postgres 7.4.5)

```
CONSTRAINT moy_age CHECK ((select avg(age) from
personnel) > 35))
```

Les contraintes d'intégrité d'entité

- Permet de spécifier la clé primaire
- Analogue à NOT NULL UNIQUE
- Génère un index
- Peut être spécifié à part lorsque la clé est constituée de plusieurs attributs (idem clause UNIQUE)

Exemple :

```
CREATE TABLE personnel (num INT PRIMARY
KEY, … )
```

Contr. d'intégrité référentielle (1/5)

Définition :

- Un attribut (ou groupe d'attributs) d'une relation apparaît comme clé dans une autre relation

  ⇒ contrainte inter-relation

Exemple :

- Une personne est affectée à un poste

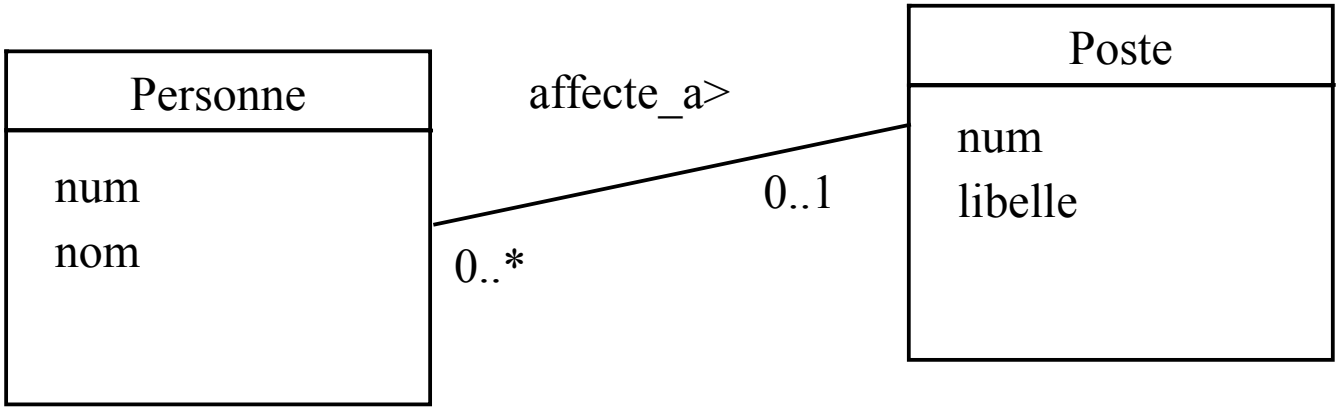
Vérification :

- Insertion d'une personne

  le poste doit exister
- Suppression d'un poste

  ce poste ne doit pas être affecté

Contr. d'intégrité référentielle (2/5)




```
CREATE TABLE poste (num INT PRIMARY KEY,
libelle TEXT NOT NULL UNIQUE )
```

```
CREATE TABLE personne (num INT PRIMARY KEY, nom
TEXT NOT NULL, num_poste INT REFERENCES poste)
```

Contr. d'intégrité référentielle (3/5)

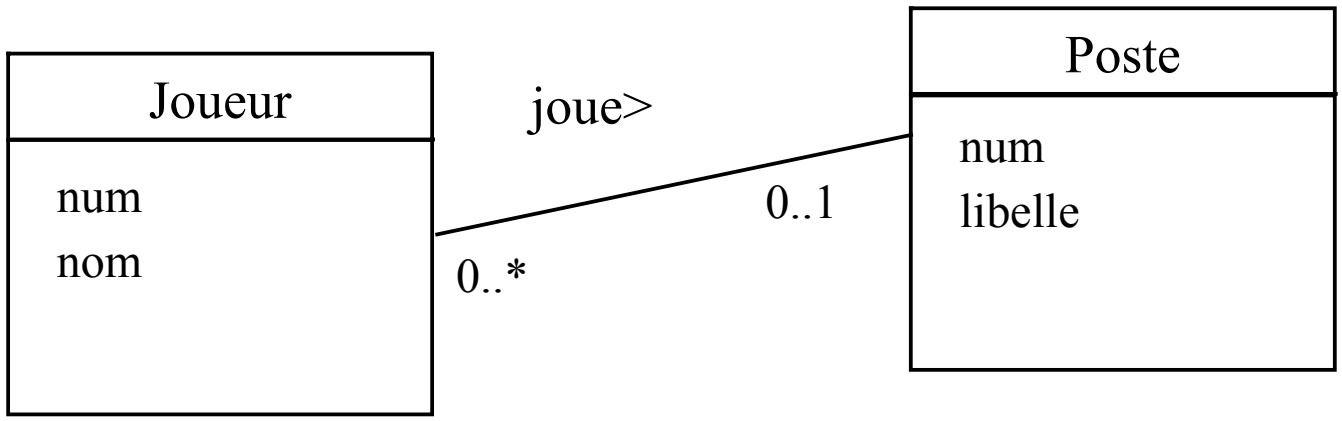
Poste

num	libelle
1	'directeur'
2	'ingénieur'
3	'agent'

Exécution :

```
INSERT INTO personne VALUES (1,'dupont', 1) -> OK
INSERT INTO personne VALUES (2,'durant', 4) -> ERREUR
DELETE FROM poste WHERE num=1 -> ERREUR
UPDATE personne SET num_poste=NULL WHERE num=1 -> OK
DELETE FROM poste WHERE num=1 -> OK
```

Contr. d'intégrité référentielle (4/5)




```
CREATE TABLE poste (num INT PRIMARY KEY, libelle TEXT NOT NULL
UNIQUE )


CREATE TABLE joueur (num INT PRIMARY KEY, nom TEXT NOT NULL)


CREATE TABLE joue (num_joueur INT NOT NULL REFERENCES joueur,
num_poste INT NOT NULL REFERENCES poste,
PRIMARY KEY (num_joueur, num_poste))
```

Contr. d'intégrité référentielle (5/5)

Joueur

num	nom
1	'Blanc'
2	'Barthez'

Poste

num	libelle
1	'goal'
2	'défenseur'
3	'milieu'
4	'attaquant'

Joue

num_joueur	num_poste
1	2
1	3
2	1

Exécution :

```
INSERT INTO joue VALUES (2,1) -> ERREUR
INSERT INTO joue VALUES (2,5) -> ERREUR
DELETE FROM poste where num=3 -> ERREUR
DELETE FROM poste where num=4 -> OK
```

Conclusion

Mécanisme de contraintes très développé

Largement utilisé depuis SQL-92

Syntaxe classique (contraintes nommées) :

```
CONSTRAINT nom [ UNIQUE | NOT NULL | ... ]
```

Mise à jour de contraintes via ALTER TABLE