

BASES DE DONNÉES AVANCÉES

5 catégories de commandes

2

- DDL - Data Definition Language
 - ▣ définition des éléments de la base de données : tables, champs, clés,...
- **DML - Data Manipulation Language**
 - ▣ manipulation des données : insertion, suppression, modification, extraction, ...
- DQL - Data Query Language
 - ▣ gestion des droits d'accès aux données
- DCL - Data Control Language
 - ▣ gestion des transactions
- SQL intégré

Commandes DML

3

- Langage de manipulation des données
 - ▣ **SELECT** : affichage de données spécifiques
 - ▣ **INSERT** : insertion de nouveaux enregistrements dans une table
 - ▣ **UPDATE** : mise à jour des données
 - ▣ **DELETE** : suppression d'enregistrements

Insertion d'enregistrements

4

□ Commande **INSERT INTO**

□ Syntaxe :

```
INSERT INTO nom_table [ ( nom_colonne [, ...] ) ]  
  { DEFAULT VALUES |  
    VALUES ( { expression | DEFAULT } [, ...] ) |  
    requête  
  } ;
```

□ Déclaration des valeurs à insérer

□ Implicite

□ Explicite

□ Par défaut

□ Résultat d'une requête

Exemple d'INSERT

5

- Déclaration implicite des valeurs
 - ▣ Valeurs saisies dans l'ordre des attributs de la relation
 - ▣ Une valeur pour chaque attribut
 - ▣ Valeurs possibles :
 - Valeur spécifique
 - NULL
 - DEFAULT

```
INSERT INTO emp  
VALUES (7499,'ALLEN','SALESMAN',7698,'20-FEB-81',1600,300,30);
```

Exemple d'INSERT

6

- Déclaration explicite des valeurs
 - ▣ Définition de l'ordre des champs
 - ▣ Choix des champs (!! default values??)

```
INSERT INTO emp (empno, name, job,mgr,hiredate,sal,comm,deptno)  
VALUES (7499,'ALLEN','SALESMAN',7698,'20-FEB-81',1600,null,30);
```

↙ Enregistrements identiques ↗

```
INSERT INTO emp (name, job,mgr,hiredate,sal,deptno,empno)  
VALUES ('ALLEN','SALESMAN',7698,'20-FEB-81',1600,30,7499);
```

Exemple d'INSERT

7

- Insertion d'enregistrements issus d'une autre table
 - ▣ Requête **SELECT** (aussi complexe que nécessaire!!)

```
INSERT INTO emp
      SELECT * from nath.emp
      WHERE comm IS NOT NULL;
```

- ▣ À n'utiliser que dans des cas particuliers
 - Attention à la redondance et l'incohérence...

Mise à jour des données

8

- Commande **UPDATE**

- Modification des valeurs d'une table
 - ▣ Pour chaque colonne spécifiée
 - ▣ Pour chaque enregistrement qui satisfait la condition (si exprimée)
 - ▣ Conservation des valeurs des autres données

Mise à jour des données

9

□ Syntaxe

```
UPDATE nom_table [AS nom_alias]  
  SET { nom_colonne = { expression | DEFAULT } |  
      ( nom_colonne [, ...] ) = ( { expression | DEFAULT } [, ...] )  
      } [, ...]  
[ FROM liste_sources ]  
[ WHERE condition ] ;
```

- **FROM** : liste les sources contenant les attributs nécessaires au WHERE
- **WHERE** : spécifie les lignes à mettre à jour

Exemples d'UPDATE

10

□ Modification simple

▣ Utilisation des valeurs existantes

```
UPDATE emp SET empno=empno+1;
```

▣ Restriction sur les lignes avec WHERE

```
UPDATE emp SET comm=0 WHERE comm IS NULL;
```

Exemples d'UPDATE

11

□ Modification sur plusieurs colonnes

```
UPDATE emp SET comm=0, empno=empno+1 WHERE comm IS NULL;
```

Modification identique

```
UPDATE emp SET (comm,empno)=(0,empno+1) WHERE comm IS NULL;
```

Mise à jour des données

12

- Utilisation d'informations issues d'autres tables
 - ▣ Fonctionnalité PostgreSQL non standard
 - ▣ **FROM** : liste de sources dont les attributs apparaissent dans la clause **WHERE**
 - source : table ou vue
- ▣ !! La table mis à jour n'apparaît pas dans la clause **FROM**

Exemple d'UPDATE

13

- Objectif : Doubler le salaire des personnes travaillant dans une ville comportant un C

```
UPDATE ??? SET ??? FROM ??? WHERE ??? ;
```

```
UPDATE emp SET sal=sal*2 FROM dept WHERE loc like '%C%' ;
```

!!! Mise à jour de tous les enregistrements !!!

```
UPDATE emp SET sal=sal*2 FROM dept  
WHERE loc like '%C%' AND emp.deptno=dept.deptno ;
```

Mise à jour de tous les enregistrements qui se situent dans une ville contenant un 'C'

Suppression de données

14

□ Commande **DELETE**

- ▣ Suppression d'enregistrements d'une table
- ▣ !! DANGER !! Suppression définitive hors transaction...

□ Syntaxe

```
DELETE FROM nom_table [ [ AS ] nom_alias ]  
  [ USING liste_using ]  
  [ WHERE condition ]
```

- ▣ **USING** : - équivalent au **FROM** de **UPDATE**
- non standard
- ▣ **WHERE** : - spécification des lignes à supprimer
- même syntaxe que le WHERE du SELECT

!!! SANS condition TOUS les enregistrements sont supprimés !!!

Exemples de DELETE

15

- Suppression de tous les enregistrements

```
DELETE FROM emp ;
```

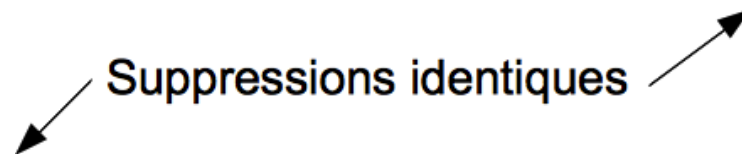
- Suppression selon une condition

```
DELETE FROM emp WHERE hiredate >= '1982-01-01';
```

- Suppression selon un attribut issu d'une autre table

```
DELETE FROM emp WHERE deptno IN  
    (SELECT deptno FROM dept WHERE loc ='CHICAGO');
```

Suppressions identiques



```
DELETE FROM emp USING dept WHERE dept.deptno=emp.deptno  
and loc ='CHICAGO';
```

5 catégories de commande

16

- **DDL - Data Definition Language**
 - ▣ définition des éléments de la base de données : tables, champs, clés,...
- **DML - Data Manipulation Language**
 - ▣ manipulation des données : insertion, suppression, modification, extraction, ...
- **DQL - Data Query Language**
 - ▣ gestion des droits d'accès aux données
- **DCL - Data Control Language**
 - ▣ gestion des transactions
- **SQL intégré**

Commandes DDL

17

- Langage de définition des données
 - ▣ Manipulation des objets SQL
 - **CREATE**: ajout
 - **ALTER** : modification
 - **DROP** : suppression
 - ▣ Les fameux «objets» ...
 - **DATABASE**
 - **TABLE**
 - **VIEW**
 - **ROLE**
 - ...

Création d'une table

18

- Définition d'une nouvelle table
 - ▣ Commande : **CREATE TABLE**
 - ▣ Déterminer le nom
 - Commence par une lettre (*!!pas un mot clé*)
 - Unique dans le schéma de la base de données
 - ▣ Pour chaque attribut de la table, définir:
 - Nom, Type et Contraintes
- Syntaxe

```
CREATE TABLE nom_table (  
  { nom_col   data_type   [DEFAULT default_expr]  [ contrainte_col [...] ]  
    | contrainte_table }  
  [, ... ]  
);
```

Exemple de CREATE TABLE

19

Contraintes de colonne



```
CREATE TABLE emp
(  empno      integer      NOT NULL,
   name       varchar(10)  NOT NULL,
   job        varchar(10)  NOT NULL,
   mgr        integer,
   hiredate   date,
   sal        numeric(7,2)  DEFAULT 0.00,
   comm       numeric(7,2),
   deptno     integer      NOT NULL,
   PRIMARY KEY(empno)
);
```



Contrainte de table

Contraintes d'intégrité

20

- Assurer précision, logique et validité des données de BDR
- Préciser généralement à la création d'une table
- Quelques exemples :
 - ▣ PRIMARY KEY
 - un ou plusieurs attributs
 - unicité de la clé → unicité de l'enregistrement
 - ▣ UNIQUE
 - valeur d'une colonne unique pour chaque enregistrement
 - ▣ NOT NULL
 - obligation de saisir une valeur dans le champ à chaque nouvel enregistrement

Contrainte de colonne

21

□ Syntaxe de « *contrainte_col* »

```
[ CONSTRAINT nom_contrainte ]  
{ NOT NULL | UNIQUE | PRIMARY KEY |  
  CHECK (condition) |  
  REFERENCES nom_table [ (nom_col) ]  
    [ ON DELETE action ]  
    [ ON UPDATE action ] }
```

- Nom optionnel pour la contrainte
- NOT NULL, UNIQUE, PRIMARY KEY
- CHECK : permet de poser une condition à l'ajout ou la modification de la donnée

Contrainte de colonne

22

□ Syntaxe de « *contrainte_col* »

```
[ CONSTRAINT nom_contrainte ]  
{ NOT NULL | UNIQUE | PRIMARY KEY |  
  CHECK (condition) |  
  REFERENCES nom_table [ (nom_col) ]  
    [ ON DELETE action ]  
    [ ON UPDATE action ] }
```

▣ **REFERENCES** : spécification d'une contrainte de clé étrangère

→ cohérence dans les données

Exemple de CREATE TABLE

23

```
CREATE TABLE emp
(  empno      integer      PRIMARY KEY,
   name       varchar(10)  NOT NULL,
   job        varchar(10)  NOT NULL,
   mgr        integer,
   hiredate   date,
   sal        numeric(7,2)  CHECK (sal>800),
   comm       numeric(7,2),
   deptno     integer      REFERENCES dept (deptno)
                                ON DELETE NO ACTION
                                ON UPDATE CASCADE
);
```

Contraintes de tables

24

□ Syntaxe de «*contrainte_table*»

```
[ CONSTRAINT nom_contrainte ]  
{ UNIQUE ( nom_colonne [, ... ] ) |  
  PRIMARY KEY ( nom_colonne [, ... ] ) |  
  CHECK ( condition ) |  
  FOREIGN KEY ( nom_colonne [, ... ] )  
    REFERENCES nom_table [ ( nom_colonne [, ... ] ) ]  
    [ ON DELETE action ]  
    [ ON UPDATE action ]  
}
```

- Quasi-identique aux contraintes de colonnes
→ Peut s'appliquer sur plusieurs colonnes

Création d'une table

25

- À partir de données existantes
 - ▣ → Requête **SELECT** (complète!)
 - ▣ Recopie de l'ensemble des enregistrements

```
CREATE TABLE emp2 AS SELECT * from emp ;
```

- ▣ Recopie partielle

```
CREATE TABLE emp2 AS SELECT empno,ename,sal  
FROM emp WHERE ename LIKE 'A%';
```

- ▣ !!! Attention copie des données mais perte des contraintes ...

Création d'une table

26

- À partir de structures/contraintes existantes

- ▣ Recopie de la structure + contraintes **NOT NULL**

```
CREATE TABLE emp2 (LIKE emp);
```

- ▣ Recopie de la structure, des contraintes et des indexes

```
CREATE TABLE emp2 (LIKE emp INCLUDING DEFAULTS  
INCLUDING CONSTRAINTS INCLUDING INDEXES);
```

- ▣ !!! Attention copie de la structure mais pas des données ...

Modification d'une table

27

□ ALTER TABLE

□ Possibilité de modification de la structure

- Après création
- Même après insertion de données

□ Modification des colonnes, des contraintes, du propriétaire, ...

□ Usages courants

- Ajout/suppression d'une colonne
- Modification du type d'une colonne
- Changement de nom : table ou colonne
- Ajout/suppression/modification d'une contrainte
- ...

Exemples d'ALTER TABLE

28

□ Ajout d'une colonne

```
ALTER TABLE emp ADD COLUMN grade INTEGER NOT NULL;
```

□ Suppression d'une colonne

```
ALTER TABLE emp DROP COLUMN grade ;
```

□ Ajout d'une valeur par défaut

```
ALTER TABLE emp ALTER COLUMN comm SET DEFAULT 0;
```

□ Ajout d'une contrainte NOT NULL

```
ALTER TABLE emp ALTER COLUMN comm SET NOT NULL;
```

Exemples d'ALTER TABLE

29

- Modification du nom d'une colonne

```
ALTER TABLE emp RENAME COLUMN ename TO nom_emp;
```

- Modification du nom d'une table

```
ALTER TABLE emp RENAME TO listing_employees;
```

- Ajout d'une contrainte de table

```
ALTER TABLE emp ADD CONSTRAINT earn_too_much  
CHECK(coalesce(sal+comm,sal)<10000);
```

- Suppression d'une contrainte de table

```
ALTER TABLE emp DROP CONSTRAINT earn_too_much ;
```

Suppression d'une table

30

□ DROP TABLE

- Suppression d'une table
- Sa structure
- Son contenu
- DELETE pour supprimer uniquement le contenu
- Possible uniquement par le propriétaire

□ Syntaxe

```
DROP TABLE nom_table [,...] [RESTRICT/CASCADE];
```

- Suppression automatique des index, triggers ou contraintes qui en dépendent

Suppression d'une table

31

□ DROP TABLE

- ▣ Destruction d'une table
- ▣ Sa structure
- ▣ Son contenu
- ▣ **DELETE** pour supprimer uniquement le contenu
- ▣ Possible uniquement par le propriétaire

□ Syntaxe

```
DROP TABLE nom_table [,...] [RESTRICT/CASCADE];
```

- ▣ **RESTRICT** : suppression impossible si un autre objet en dépend
- ▣ **CASCADE** : suppression des vues et des contraintes de clés étrangères

Quelques objets du SGBD

32

□ TABLE

- ▣ Structure permettant de stocker les données
- ▣ Les catalogues...

□ DATABASE

- ▣ Ensemble nommé d'objets SQL
- ▣ Niveau hiérarchique le plus élevé d'organisation des objets du SGBD

□ SCHEMA

- ▣ Espace de nommage dans une base de données

□ ROLE

- ▣ Notion d' « utilisateur » ou « groupe d'utilisateurs »
- ▣ Avant la 8.1 : USER et GROUP

Les catalogues systèmes

33

- Relations particulières ...
 - ▣ Nom commence par **pg_**
 - ▣ Stockage de toutes les métadonnées des objets contenus dans la base de données
 - ▣ Permet d'obtenir la liste des tables, des utilisateurs, des vues...
 - ▣ Chaque catalogue système est propre à la base de données (sauf rares exceptions)
 - ▣ Recopie depuis la base modèle *template1*
- *et classique*
 - ▣ *Possibilité de modifier ou supprimer les catalogues (!!)*

Bases de données

34

□ Mot clé **DATABASE**

- Ensemble nommé d'objets SQL
- Chaque objet appartient à une seule base (sauf certains catalogues système)
- Se manipule au même titre que les autres objets
- **CREATE/ALTER/DROP DATABASE**

□ Les bases de données sont séparées physiquement

- Accès uniquement aux objets de la base à laquelle on est connecté
- !! Partage de ressources pour différents utilisateurs → travail au sein de la même base

Bases de données

35

- Connexion au serveur sur UNE base de données
 - ▣ Gestion du contrôle d'accès au niveau de la connexion

```
psql -d nom_de_la_base
```

- ▣ Par défaut, *nom_de_la_base* = *nom_user*
- Lister l'ensemble des bases de données

```
SELECT datname FROM pg_database;  --!objet commun aux BDs !
```

```
\l      commande postgresQL  
-l      option sur la commande unix psql
```

Les schémas en postgresSQL

36

- Mot clé **SCHEMA** (!! Non standard SQL)
- Notion d'espace de nommage
 - ▣ Aucun conflit de noms entre les objets de différents schémas d'une même base de données
 - ▣ Catalogue : pg_namespace ;
- Une base de données contient un ou plusieurs schémas
 - ▣ Tous les objets de la base de données sont contenus dans un schéma
- Utilité ?
 - ▣ Différents utilisateurs sur une même base de données sans interférence
 - ▣ Organisation des objets de la base : créer des groupes logiques pour faciliter la gestion

Manipulation de SCHEMA

37

□ Création

```
CREATE SCHEMA nom_schema ;
```

□ Accéder à un objet d'un schéma

```
nom_schema.nom_objet
```

□ Création d'un objet dans un schéma

```
CREATE OBJET nom_schema.nom_objet (...);
```

□ Définir le propriétaire

```
CREATE SCHEMA nom_schema AUTHORIZATION nom_utilisateur ;  
ALTER SCHEMA nom_schema OWNER TO new_user ;
```

□ Supprimer un schéma

```
DROP SCHEMA nom_schema [cascade|restrict];
```

Le schéma par défaut

38

- ❑ Schéma *public*
- ❑ Par défaut dans toute nouvelle base de donnée
- ❑ Non obligatoire, peut être supprimé
- ❑ Si aucun schéma n'est créé dans une base, tous les objets seront créés dans *public*

```
CREATE TABLE ma_table (...);
```

↙ équivalent **par défaut** ↘

```
CREATE TABLE public.ma_table (...);
```

Dans quel schéma je travaille par défaut?

39

- Chemin de parcours des schémas : `search_path`

```
SHOW search_path ;
```

- ▣ Liste des schémas par ordre de recherche

```
search_path
-----
"$user",public
(1 ligne)
```

- Si aucun prefix sur l'objet : recherche dans chacun des schémas de `search_path` de manière ordonnée

→ pre-fixage obligatoire si l'objet est dans un schéma non précisé dans le `search_path`

- Modification du chemin

```
SET search_path TO "toto", "$user", public;
```

- ▣ Le schéma toto doit exister
- ▣ Modification locale à la connexion sur la database courante

Liste des schémas et des tables

40

□ Liste des schémas

```
\dn
```

□ Lister les tables de la base de données

```
\dt
```

- ▣ Recherche uniquement les tables dans le search_path
- ▣ Si même nom → ne se voient pas

```
\dt *.*
```

- ▣ Affichage de toutes les tables

□ Remarques :

- ▣ chaque schéma est dépendant de la base de données et n'existe pas dans les autres ...
- ▣ Le search_path se réinitialise lors d'une nouvelle connexion

ROLE

41

- Mot clé **ROLE** (!! Non standard SQL)
- Entité qui peut posséder des objets et avoir des droits
 - ▣ Rôle de la base \neq utilisateur du système d'exploitation
 - ▣ Global aux bases de données
 - ▣ Affecte les droits d'accès sur ses objets pour les autres rôles
- Catalogue des rôles : **pg_roles**

```
SELECT rolname FROM pg_roles;
```

```
\du
```

ROLE - Utilisateur

42

- Un « user » est un rôle qui a le droit de connexion
 - ▣ Détermination des privilèges par rapport au rôle spécifié lors de la connexion

```
psql -U nom_user
```

- ▣ session_user : nom_user
- L'utilisateur initial : *postgres*
 - ▣ Rôle par défaut d'un SGBD postgresSQL nouvellement installé
 - ▣ Droits d'un super-utilisateur :
 - Aucune vérification de droits avant l'exécution d'une action

ROLE - Groupement

43

- Un rôle peut être à la fois une personne ou un groupe de personnes
 - ▣ Intérêt : partage commun de droits
- Possibilité de changer de rôle

```
SET ROLE to group_user ;
```

- ▣ Si nom_user fait partie du groupe group_user
 - ▣ current_user prend alors la valeur group_user
- Quel rôle ?

```
SELECT current_user, session_user;
```

```
RESET ROLE;
```

Exemple de ROLE

44

□ Création d'un rôle

```
CREATE ROLE etudiant LOGIN ;  
CREATE USER etudiant ;           -- identique à la ligne précédente  
  
CREATE ROLE prof SUPERUSER ;    -- création d'un super-utilisateur  
  
CREATE ROLE etu_db CREATEDB ;   -- possibilité de créer des bases de données  
  
CREATE ROLE etu_role CREATEROLE ; -- possibilité de créer des rôles
```

□ Modification d'un rôle

```
ALTER ROLE etudiant WITH PASSWORD 'toto13' ;
```

□ Suppression d'un rôle

```
DROP ROLE etudiant;
```

Quelques objets du SGBD (suite)

45

□ VIEW

- ▣ Table virtuelle

□ INDEX

- ▣ Construction physique d'une structure de données sur une ou plusieurs colonnes d'une table

□ SEQUENCE

- ▣ Suite d'entiers qui s'incrémente à chaque appel

□ FUNCTION

- ▣ Opération stockée

VIEW

46

- Mot clé **VIEW**
 - ▣ Correspond à une table « virtuelle »
 - Données non stockées physiquement
 - Requête de création stockée
- Nom différent de celui d'une table, d'un index ou d'une séquence
- Objet dynamique
 - ▣ Modification des données sources de la vue
 - Modification des données de la vue
- Pourquoi/Quand utiliser une vue?
 - ▣ Utilisation fréquente du résultat d'une requête
 - ▣ Alléger des requêtes trop lourdes

Manipulation de VIEW

47

□ Création d'une vue

```
CREATE VIEW name_emp_dept AS SELECT ename,dname  
FROM emp NATURAL JOIN dept;
```

□ Utilisation d'une vue

```
SELECT * FROM name_emp_dept WHERE ename LIKE '%A%';
```

□ Suppression d'une vue

```
DROP VIEW name_emp_dept;
```

□ Lister les vues

```
SELECT * from pg_views ;
```

– catalogue des vues

```
\dv
```

INDEX

48

- Notion non standard
 - ▣ Méthode courante augmentant les performances du SGBD
 - ▣ Méthodes d'indexation : arbres, tables de hachage
 - ▣ Algorithme de recherche spécifique au type d'index
 - ▣ Même intérêt que les index de livre

- Pourquoi utiliser un index ?
 - ▣ ++ : Retrouver une ligne spécifique plus rapidement
 - ▣ – : Ajout une surcharge au SGBD (maj à chaque modif)
 - ▣ → À utiliser à bon escient

INDEX

49

- Un index créé automatiquement sur chaque clé primaire
- Création d'index

```
CREATE INDEX nom_index  
ON nom_table [USING methode] ( nom_colonne [,...] )  
[ WHERE expression ]
```

- ▣ **WHERE** : création d'un index partiel
- Suppression d'index

```
DROP INDEX nom_index [,...] [RESTRICT/CASCADE];
```

- Modification avec **ALTER INDEX**

SEQUENCE

50

- Table permettant la génération d'entier

```
CREATE SEQUENCE nom_seq [INCREMENT BY increment ]  
  [ MINVALUE valeurmin | NO MINVALUE ]  
  [ MAXVALUE valeurmax | NO MAXVALUE ]  
  [ START [ WITH ] début ]  
  [ CACHE cache ]  
  [ [ NO ] CYCLE ]  
  [ OWNED BY { table.colonne | NONE } ]
```

- Utilisation des valeurs avec
 - ▣ *nextval()* : valeur suivante
 - ▣ *curval()* : valeur courante
 - ▣ *setval()* : initialisation de la valeur courante

Exemple de SEQUENCE

51

```
CREATE SEQUENCE mon_num START 3 INCREMENT BY 2;

SELECT nextval('mon_num');
    -- retourne 3

SELECT nextval('mon_num');
    -- retourne 5

INSERT INTO ma_table VALUES (nextval('mon_num')) ;

ALTER SEQUENCE mon_num RESTART WITH 3 ;
    -- réinitialisation à 3 ;

SELECT setval('mon_num',3,false);
    -- idem
```

FUNCTION

52

- Identificateur demandant l'exécution au sein d'une requête SQL d'une opération programmée
 - ▣ Le résultat en retour s'insère en lieu et place de l'appel de la fonction
- Objet lié à une base de données (tout comme les autres objets : tables, vues, index ...)
- Intérêts :
 - ▣ Étendre les capacités de SQL par la création de nouvelles fonctions
 - ▣ Simplifier l'élaboration des requêtes SQL

FUNCTION

53

□ Syntaxe

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ argtype [, ...] ] )  
RETURNS type_retour  
AS 'definition'  
LANGUAGE 'nom_langage' [ WITH ( attribut [, ...] ) ]
```

□ CREATE OR REPLACE

- ▣ !! le prototype reste inchangé

□ Arguments :

- ▣ Types d'arguments

- ▣ Non obligatoires mais ()

FUNCTION

54

□ Syntaxe

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ argtype [, ...] ] )  
RETURNS type_retour  
AS 'definition'  
LANGUAGE 'nom_langage' [ WITH ( attribut [, ...] ) ]
```

□ *type_retour*

- ▣ Type de données renvoyé par la fonction

□ **AS** '*definition*':

- ▣ Code de la fonction pour le SQL ou les langages procéduraux de type PL/pgSQL
- ▣ Chemin en absolu du fichier contenant le code objet pour le C

FUNCTION

55

□ Syntaxe

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ argtype [, ...] ] )  
RETURNS type_retour  
AS 'definition'  
LANGUAGE 'nom_langage' [ WITH ( attribut [, ...] ) ]
```

□ *LANGUAGE*

- ▣ Précise le nom du langage du corps de la fonction

□ *attribut* :

- ▣ *isstrict* : si un des paramètres est NULL alors la fonction renvoie NULL et n'est pas exécutée
- ▣ *immutable* ...

FUNCTION

56

□ Exemple de déclaration d'une fonction

```
CREATE FUNCTION nom_leader (varchar(20))  
RETURNS varchar(30) AS  
'SELECT nom  
FROM individu join leader on id_ind=id_musicien join cd using (id_cd)  
WHERE titre_cd=$1'  
LANGUAGE 'SQL';
```

□ Appel de la fonction

```
select nom_leader('evidence');
```

```
nom_leader  
-----  
coltrane  
(1 row)
```

!!!! Le langage doit avoir été créé au préalable ...

5 catégories de commande

57

- DDL - Data Definition Language
 - ▣ définition des éléments de la base de données : tables, champs, clés,...
- DML - Data Manipulation Language
 - ▣ manipulation des données : insertion, suppression, modification, extraction, ...
- **DQL - Data Query Language**
 - ▣ gestion des droits d'accès aux données
- DCL - Data Control Language
 - ▣ gestion des transactions
- SQL intégré

Gestion des privilèges

58

- Liste de contrôle d'accès (ACL)
 - ▣ Quels rôles possèdent quels droits sur quels objets ?
 - ▣ Consultée avant toute action
 - ▣ Pour chaque objet :
 - Ensemble de privilèges
 - Ensemble de restriction
 - ▣ Possibilité de modifier les droits sur un objet :
 - Par le super-utilisateur ou le propriétaire
 - Octroi de privilèges : GRANT
 - Restriction de privilèges : REVOKE

Attribution de privilèges

59

□ GRANT

```
GRANT privilege [,...] ON objet [,...] TO  
{ username | GROUP groupname | PUBLIC } [, ...]
```

□ Exemples de valeurs pour *privilege* :

- ▣ SELECT (lecture)
- ▣ INSERT (insertion de lignes)
- ▣ UPDATE (modification de colonnes)
- ▣ DELETE (suppression de lignes)
- ▣ ALL

!!! Attention les droits dépendent des objets ...

Attribution de privilèges

60

□ GRANT

```
GRANT privilege [,...] ON objet [,...] TO  
{ username | GROUP groupname | PUBLIC } [, ...]
```

□ Exemples de valeurs pour *objet* :

- ▣ bases de données : **ON DATABASE** nom_bd
- ▣ table : **ON [TABLE]** nom_table
- ▣ schéma : **ON SCHEMA** nom_schema
- ▣ langage : **ON LANGUAGE** nom_langage
- ▣ fonctions : **ON FUNCTION** nom_fonction

!!! Attention les droits dépendent des objets ...

Attribution de privilèges

61

□ GRANT

```
GRANT privilege [,...] ON objet [,...] TO  
{ username | GROUP groupname | PUBLIC } [, ...]
```

□ TO :

- ▣ Un utilisateur particulier : *username*
- ▣ Un groupe d'utilisateur : **GROUP** *groupname*
- ▣ Tous les utilisateurs : **PUBLIC**

Restriction des privilèges

62

□ REVOKE : Restriction de privilèges

```
REVOKE privilege [,...] ON objet [,...] FROM  
{ username | GROUP groupname | PUBLIC } [, ...]
```

□ Remarque :

- Le fait de supprimer des droits à public n'affecte pas ceux à qui l'on a octroyé spécifiquement les droits.

Accès aux différents schémas

63

- Donner les droits d'accès à son schéma

```
GRANT usage ON schema mon_schema TO nom_utilisateur ;
```

- Révoquer les droits

```
REVOKE create ON schema public FROM public ;
```



Interdiction de créer des objets dans le schéma public

- !! L'octroi de droits sur le schema n'octroie pas de droits sur les objets de ce schema ...

Création de groupes d'utilisateurs

64

□ Créations de rôles

```
CREATE ROLE group_test;  
CREATE ROLE toto;  
CREATE ROLE tata;  
CREATE ROLE titi;
```

□ Ajout d'utilisateurs au sein d'un groupe

```
GRANT group_test TO toto,tata,titi;
```

□ Suppression d'utilisateurs

```
REVOKE group_test FROM toto;  
ALTER GROUP group_test DROP USER toto; -- idem
```


5 catégories de commandes

65

- DDL - Data Definition Language
 - ▣ définition des éléments de la base de données : tables, champs, clés,...
- DML - Data Manipulation Language
 - ▣ manipulation des données : insertion, suppression, modification, extraction, ...
- DQL - Data Query Language
 - ▣ gestion des droits d'accès aux données
- **DCL - Data Control Language**
 - ▣ gestion des transactions
- SQL intégré

Les transactions

66

- Groupe d'opérations formant une unité indivisible
 - ▣ Toutes effectuées
 - ▣ Ou aucune
- Participe à l'intégrité des données
- Par défaut chaque instruction est elle-même une transaction
- Déclaration explicite par un bloc transactionnel

```
BEGIN ;  
-- listes d'instructions  
COMMIT ;
```

Exemple de transactions

67

- Un virement financier :

```
BEGIN ;  
UPDATE comptes SET balance = balance - 100.00  
  WHERE nom = 'Alice';  
UPDATE comptes SET balance = balance + 100.00  
  WHERE nom = 'Bob';  
COMMIT ;
```

- Les deux opérations sont nécessaires ou aucune ne doit être exécutée.

Gestion des transactions

68

- Possibilité de placer des points de retournement
 - ▣ **SAVEPOINT**
- Possibilité d'annuler des opérations :
 - ▣ **ROLLBACK TO**

```
BEGIN;  
UPDATE comptes SET balance = balance - 100.00  
  WHERE nom = 'Alice';  
SAVEPOINT mon_pointdesauvegarde;  
UPDATE comptes SET balance = balance + 100.00  
  WHERE nom = 'Bob';  
-- oups ... oublions ça et créitons le compte de Wally  
ROLLBACK TO mon_pointdesauvegarde;  
UPDATE comptes SET balance = balance + 100.00  
  WHERE nom = 'Wally';  
COMMIT;
```