

BASES DE DONNÉES AVANCÉES

Réaliser sa BdD relationnelle

2

- Analyser le problème
 - ▣ De l'analogique au numérique ...
- Concevoir un MEA
 - ▣ Ensemble d'entités et d'associations
- Passer du MEA au MR
 - ▣ Ensemble de schémas de relation
- Choix du SGBD
 - ▣ Implémentation avec le langage SQL

Le langage SQL

3

- SQL : Structured Query Language
 - ▣ Le langage de communication avec un SGBD-R
- De nombreuses fonctionnalités
 - ▣ Gestion, insertion, suppression, modification de données
 - ▣ Opérations arithmétiques et de comparaison
 - ▣ Affichage des données
 - ▣ ...

La norme SQL

4

- 1986 : Approbation ANSI de l'implémentation IBM
- 1987 : Première Norme ISO
- Puis :
 - ▣ 1989, 1992
 - ▣ 1999 (SQL3)
 - Intégration de parties plus avancées (SGBDRO, interface de programmation, gestion d'intégrité des données...)
 - ▣ 2003 (SQL:2003) , 2008 (SQL:2008)
 - Ajouts de manipulations XML
- 2011 (SQL:2011)
 - ▣ What's new? <http://www.sigmod.org/publications/sigmod-record/1203/pdfs/10.industry.zemke.pdf>

PostgreSQL

5

□ Des atouts majeurs

- ▣ Projet de SGBD-R non commercial le plus avancé
- ▣ Projet Open Source toujours en développement
- ▣ Existence de distributions commerciales (support tech.)

PostgreSQL

6

□ Historique

- ▣ 1977 : Projet Ingres débuté par l'Université de Berkeley
 - Relation Technologies/Ingres Corporation
- ▣ 1986 : Nouvelle équipe de l'Université de Berkeley
 - Ingres → Postgres
- ▣ 1996 : Ajout de nouvelles fonctionnalités par la communauté du logiciel libre
 - Postgres → PostgreSQL

PostgreSQL et la norme SQL ?

7

- Version installée à l'institut Claude Chappe
 - ▣ Côté serveur : 8.3.7
 - Documentation : <http://docs.postgresql.fr/8.3/>
 - ▣ Côté clients : psql (PostgreSQL) 8.4.X et 9.0.X
- Conformité avec la norme SQL
 - ▣ <http://docs.postgresql.fr/8.3/features.html>
 - ▣ Un extrait :

PostgreSQL supporte la plupart des fonctionnalités majeures de SQL:2003. Sur les 164 fonctionnalités requises pour une conformité « centrale » complète (*full Core conformance*), PostgreSQL se conforme à plus de 150. De plus, il existe une longue liste de fonctionnalités optionnelles supportées.

SQL : un langage fortement typé

8

- Une donnée \leftrightarrow Un type
- Un type \rightarrow Un ensemble d'opérations applicables
- Type : contrainte définit lors de la création de la table
 - ▣ !! important, y penser dès la modélisation !!
- Plus d'une 30aine de types définis dans PostgreSQL
 - ▣ Certains basés sur une norme ISO
 - ▣ D'autres non standards
 - !! compatibilité avec d'autres SGBDR ...

Types de données

9

- Booléens et binaires
- Caractères
- Numériques
 - ▣ Entiers
 - ▣ Réels
- Date et heure
- ...
- Types spécifiques
 - ▣ Géométrie, adresse réseau...

Le type NULL

10

- **NULL** correspond à :
 - ▣ méta-valeur représentant une absence de valeur
- Affectation :
 - ▣ Tout champ indépendamment de son type
 - !! sauf contrainte **NOT NULL**
- Référencement : le mot clé **NULL**
- **NULL** n'est pas
 - ▣ la valeur booléenne *false*
 - ▣ La chaîne de caractères vide ""
 - ▣ attention : 'NULL' n'est pas **NULL**
- Valeur par défaut d'un champ (si autorisé!)

5 catégories de commandes

11

- 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

5 catégories de commande

12

- **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

13

- Langage de définition des données
 - ▣ Manipulation des **objets SQL**
 - ▣ 3 commandes :
 - **CREATE** : création d'un nouvel « objet »
 - **ALTER** : modification
 - **DROP** : suppression

Quelques objets du SGBD

14

□ 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

□ TABLE

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

□ VIEW

- ▣ Table virtuelle

Les catalogues systèmes

15

- Tables 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)
- *et classique*
 - ▣ *Possibilité de modifier ou supprimer les catalogues (!!)*

Bases de données

16

□ Mot clé **DATABASE**

- Conteneur pour un 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

17

- 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
```

Bases de données

18

□ Création d'une nouvelle DATABASE

- ▣ Depuis n'importe quelle database
- ▣ Recopie de la base modèle *template1*
 - *Notamment ensemble des catalogues nécessaires*

```
template1=# \dt *.*
```

Liste des relations			
Schéma	Nom	Type	Propriétaire
information_schema	sql_features	table	postgres
information_schema	sql_implementation_info	table	postgres
information_schema	sql_languages	table	postgres
information_schema	sql_packages	table	postgres
information_schema	sql_parts	table	postgres
information_schema	sql_sizing	table	postgres
information_schema	sql_sizing_profiles	table	postgres

(7 lignes)

19

```
template1=# select * from information_schema.sql_languages ;
 sql_language_source | sql_language_year | sql_language_conformance |
-----+-----+-----+
 ISO 9075            | 1999              | CORE                      |
 ISO 9075            | 1999              | CORE                      |
 ISO 9075            | 2003              | CORE                      |
 ISO 9075            | 2003              | CORE                      |
(4 lignes)
```

sql_language_binding_style	sql_language_programming_language
DIRECT	
EMBEDDED	C
DIRECT	
EMBEDDED	C

Catalogues par défaut

20

□ sql_features

E011	Numeric data types		
E011	Numeric data types	01	INTEGER and SMALLINT data types
E011	Numeric data types	02	REAL, DOUBLE PRECISION, and FLOAT data types
E011	Numeric data types	03	DECIMAL and NUMERIC data types
E011	Numeric data types	04	Arithmetic operators
E011	Numeric data types	05	Numeric comparison
E011	Numeric data types	06	Implicit casting among the numeric data types
E021	Character data types		
E021	Character string types	01	CHARACTER data type
E021	Character string types	02	CHARACTER VARYING data type
E021	Character string types	03	Character literals
E021	Character string types	04	CHARACTER_LENGTH function
E091	Set functions	01	AVG
E091	Set functions	02	COUNT
E091	Set functions	03	MAX
E091	Set functions	04	MIN
E091	Set functions	05	SUM
E091	Set functions	06	ALL quantifier
E091	Set functions	07	DISTINCT quantifier
E101	Basic data manipulation		
E101	Basic data manipulation	01	INSERT statement
E101	Basic data manipulation	03	Searched UPDATE statement
E101	Basic data manipulation	04	Searched DELETE statement

Catalogues par défaut

21

□ sql_features

E051	Basic query specification	01	SELECT DISTINCT
E051	Basic query specification	02	GROUP BY clause
E051	Basic query specification	04	GROUP BY can contain columns not in <select list>
E051	Basic query specification	05	Select list items can be renamed
E051	Basic query specification	06	HAVING clause
E051	Basic query specification	07	Qualified * in select list
E051	Basic query specification	08	Correlation names in the FROM clause
E051	Basic query specification	09	Rename columns in the FROM clause
E061	Basic predicates and search conditions		
E061	Basic predicates and search conditions	01	Comparison predicate
E061	Basic predicates and search conditions	02	BETWEEN predicate
E061	Basic predicates and search conditions	03	IN predicate with list of values
E061	Basic predicates and search conditions	04	LIKE predicate
E061	Basic predicates and search conditions	05	LIKE predicate ESCAPE clause
E061	Basic predicates and search conditions	06	NULL predicate
E061	Basic predicates and search conditions	07	Quantified comparison predicate
E061	Basic predicates and search conditions	08	EXISTS predicate
E061	Basic predicates and search conditions	09	Subqueries in comparison predicate
E061	Basic predicates and search conditions	11	Subqueries in IN predicate
E061	Basic predicates and search conditions	12	Subqueries in quantified comparison predicate
E061	Basic predicates and search conditions	13	Correlated subqueries
E061	Basic predicates and search conditions	14	Search condition
E071	Basic query expressions		
E071	Basic query expressions	01	UNION DISTINCT table operator
E071	Basic query expressions	02	UNION ALL table operator
E071	Basic query expressions	03	EXCEPT DISTINCT table operator
E071	Basic query expressions	05	Columns combined via table operators need not have exactly the same data type
E071	Basic query expressions	06	Table operators in subqueries
E081	Basic Privileges		

Bases de données

22

```
nath=# create database test;
```

```
CREATE DATABASE
```

```
nath=# \c test
```

```
Vous êtes maintenant connecté à la base de données « test ».
```

```
test=# \dt *.*  
  
Liste des relations
```

Schéma	Nom	Type	Propriétaire
information_schema	sql_features	table	postgres
information_schema	sql_implementation_info	table	postgres
information_schema	sql_languages	table	postgres
information_schema	sql_packages	table	postgres
information_schema	sql_parts	table	postgres
information_schema	sql_sizing	table	postgres
information_schema	sql_sizing_profiles	table	postgres

(7 lignes)

Les schémas en postgresSQL

23

- 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

24

□ 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

25

- ❑ 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?

26

- 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 base de donnée courante

Liste des schémas et des tables

27

□ 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

Création d'une table

28

- 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

29

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

30

- Assurer précision, logique et validité des données de la 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

31

□ 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

32

□ 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

33

```
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

34

□ 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

Contrainte de clé étrangère

35

- Assure l'intégrité des données par un lien entre :
 - ▣ un attribut et celui d'une autre table (cible)
 - ▣ plusieurs attributs et ceux correspondants dans une autre table
- Obligation pour l'attribut de prendre ses valeurs parmi celles de l'attribut cible
 - ▣ Défini l'ordre dans l'insertion des enregistrements : table cible avant table référant
- Obligation d'unicité sur les attributs cibles
 - ▣ A minima contrainte *unique*

Contrainte de clé étrangère

36

□ Au moment de la création de la table :

▣ Contrainte de colonne :

```
nom_attribut type_attribut REFERENCES nom_table_cible (nom_attr_cible)
```

▣ Contrainte de table :

```
FOREIGN KEY nom_attribut REFERENCES nom_table_cible (nom_attr_cible)
```

□ Après création de la table :

```
ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte FOREIGN KEY  
(nom_attribut) REFERENCES nom_table_cible (nom_attribut_cible);
```

Contrainte de clé étrangère

37

- **Spécification du comportement**
 - ▣ En cas de modification de l'attribut cible
`ON UPDATE nom_action`
 - ▣ En cas de suppression de l'attribut cible
`ON DELETE nom_action`
- **Liste des actions :**
 - ▣ NO ACTION* : ne permet pas la modification
 - ▣ RESTRICT : idem NO ACTION mais ne peut être différée
 - ▣ CASCADE : entraine la modification de la table source
 - ▣ SET NULL : perte de la valeur
 - ▣ SET DEFAULT : mise à jour avec la valeur par défaut

* Valeur par défaut

Contrainte de clé étrangère

38

- Type de concordance
 - ▣ À chaque nouvelle valeur insérée, une comparaison est faite avec les valeurs des colonnes de référence
 - ▣ 3 types :
 - MATCH FULL : Si une valeur de la clé est nulle alors toute la valeur de la clé doit être nulle
 - MATCH SIMPLE : valeur par défaut
 - MATCH PARTIAL : pas implémenté...
 - ▣ Indication à noter après le nom de la colonne cible

Création d'une table

39

- 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

40

- 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

41

□ 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

42

□ 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

43

- 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

44

□ DROP TABLE

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

Suppression d'une table

45

□ Syntaxe

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

- Suppression automatique des index, triggers ou contraintes qui en dépendent
- **RESTRICT** : suppression impossible si un autre objet en dépend
- **CASCADE** : suppression des vues et des contraintes de clés étrangères

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
```

5 catégories de commandes

48

- 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

49

- Langage de manipulation des données
 - ▣ 3 commandes de base:
 - **INSERT** : insertion de nouvelles données dans une table
 - **UPDATE** : mise à jour des données
 - **DELETE** : suppression d'enregistrements
 - ▣ La commande **SELECT**
 - consultation de la base de données
 - associé à des mots clés et des clauses pour trouver et visualiser quasiment toutes les informations possibles
 - instruction la plus puissante et la plus complexe!

Insertion de données

50

□ 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

51

- 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

52

- 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

53

- 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

54

- 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

55

□ 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

56

□ 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

57

□ 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

58

- 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

59

- 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

60

□ 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

61

- Suppression de tous les enregistrements

```
DELETE FROM emp ;
```

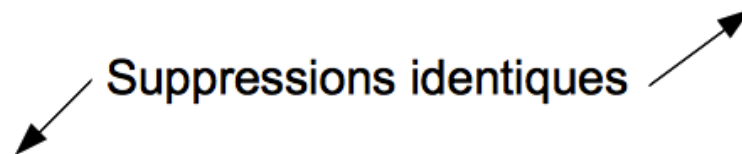
- Suppression selon une condition

```
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';
```

Sélection de données : SELECT

62

- 9 clauses dont 7 optionnelles

```
SELECT [ ALL | DISTINCT ] { * | expression [ AS nom_affiché ] } [, ...]  
FROM nom_table [ [ AS ] alias ] [, ...] (version simplifiée)  
[ WHERE prédicat ]  
[ GROUP BY expression [, ...] ]  
[ HAVING condition [, ...] ]  
[ { UNION | INTERSECT | EXCEPT [ALL] } requête ]  
[ ORDER BY expression [ ASC | DESC ] [, ...] ]  
[ LIMIT { ALL | nombre } ]  
[ OFFSET début ]
```

Les opérateurs de bases

63

□ Projection : clause **SELECT**

- Rappel : En algèbre relationnelle, la projection élimine des attributs d'une relation

- Syntaxe :

```
SELECT att1, att2, ... attN FROM nom_table ;
```

□ Sélection : clause **WHERE**

- Rappel : En algèbre relationnelle, la sélection sur la condition C permet de garder les n-uplets qui satisfont C.

- Syntaxe :

```
SELECT * FROM nom_table WHERE condition ;
```

Exemple : la table *emp*

64

```
SELECT * FROM emp;
```

empno	name	job	mgr	hiredate	sal	comm
7369	SMITH	CLERK	7902	1980-12-17	800	
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300
7521	WARD	SALESMAN	7698	1981-02-22	1250	500
7566	JONES	MANAGER	7839	1981-04-02	2975	
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400
7698	BLAKE	MANAGER	7839	1981-05-01	2850	
7782	CLARK	MANAGER	7839	1981-06-09	2451	
7788	SCOTT	ANALYST	7566	1982-12-09	3000	
7839	KING	PRESIDENT		1981-11-17	5000	
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0
7876	ADAMS	CLERK	7788	1983-01-12	1100	
7900	JAMES	CLERK	7698	1981-12-03	950	
7902	FORD	ANALYST	7566	1981-12-03	3000	
7934	MILLER	CLERK	7782	1982-01-23	13000	

Exemple de Projection et Sélection

65

```
SELECT name, hiredate, sal FROM emp WHERE sal >= 1500 ;
```

empno	name	job	mgr	hiredate	sal	comm
7369	SMITH	CLERK	7902	1980-12-17	800	
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300
7521	WARD	SALESMAN	7698	1981-02-22	1250	500
7566	JONES	MANAGER	7839	1981-04-02	2975	
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400
7698	BLAKE	MANAGER	7839	1981-05-01	2850	
7782	CLARK	MANAGER	7839	1981-06-09	2451	
7788	SCOTT	ANALYST	7566	1982-12-09	3000	
7839	KING	PRESIDENT		1981-11-17	5000	
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0
7876	ADAMS	CLERK	7788	1983-01-12	1100	
7900	JAMES	CLERK	7698	1981-12-03	950	
7902	FORD	ANALYST	7566	1981-12-03	3000	
7934	MILLER	CLERK	7782	1982-01-23	13000	

Exemple de Projection et Sélection

66

```
SELECT name, hiredate, sal FROM emp WHERE sal >= 1500 ;
```

name	hiredate	sal
ALLEN	1981-02-20	1600
JONES	1981-04-02	2975
BLAKE	1981-05-01	2850
CLARK	1981-06-09	2450
SCOTT	1982-12-09	3000
KING	1981-11-17	5000
TURNER	1981-09-08	1500
FORD	1981-12-03	3000

Clause SELECT et ses opérateurs

67

- ***** : tous les champs de la table
- **ALL** : retourne toutes les lignes (par défaut)
- **DISTINCT** : suppression des doublons
- **+, -, *, /** : opérations mathématiques de base
- **||** : concaténation de champs de type caractères
- **AS** : nommer une colonne calculée

Exemples d'op. de la clause SELECT

68

```
SELECT empno || ' ' || name AS id_nom  
FROM emp where job = 'ANALYST';
```

id_nom
7788_SCOTT
7902_FORD

```
SELECT DISTINCT job FROM emp ;
```

job
ANALYST
CLERC
MANAGER
PRESIDENT
SALESMAN

```
SELECT sal+comm FROM emp ;
```

? column ?
1900
1750
2650
1500

La clause FROM

69

- ❑ Clause indispensable pour sélectionner l'ensemble des données sur lesquelles travailler
- ❑ Réellement obligatoire?
 - ▣ Oui : travailler sur des données
 - ▣ Non : appeler une fonction
- ❑ Possibilité de jointure
 - ▣ Travailler sur des données issues de plusieurs tables
 - ▣ Résultat : une relation

Les jointures

70

- Sélection des données de travail sur plusieurs tables
 - ▣ Possibilité de jointure réflexive
- Mot clé : JOIN
- Dans la clause FROM
- 3 types de jointures :
 - ▣ Jointure croisée : CROSS JOIN
 - ▣ Jointure interne : INNER JOIN / NATURAL JOIN
 - ▣ Jointure externe : [RIGHT/LEFT/FULL] OUTER JOIN

Jointure croisée : CROSS JOIN

71

- Produit cartésien entre 2 tables
 - ▣ $R1 \times R2 \rightarrow R3$ regroupant exclusivement toutes les possibilités de combinaison des occurrences de R1 et R2
- Syntaxe :

```
SELECT * FROM table_1 CROSS JOIN table_2
```

 - ▣ identique à :

```
SELECT * FROM table_1, table_2
```
- Quand l'utiliser?
 - ▣ Besoin de toutes les possibilités de combinaison entre des valeurs de différentes tables

Jointure interne : [INNER] JOIN

72

- Retour uniquement des lignes satisfaisant la condition de jointure
- Condition de jointure :
 - ▣ Condition comparant des champs compatibles
 - même type de données
 - !! même signification !!
 - ▣ 2 syntaxes exprimant la condition :
 - **USING** (nom_attr)

```
SELECT * FROM table_1 JOIN table_2 USING nom_attr_12
```

- **ON** table_1.nom_attr_a op_comp table_2.nom_attr_b

- ▣ Possibilité de condition avec conjonction/négation

Jointure naturelle : NATURAL JOIN

73

- Cas particulier d'une jointure interne : Condition de jointure implicite

```
SELECT * FROM table_1 NATURAL JOIN table_2;
```

- ▣ La valeur de TOUS les attributs ayant même nom dans les 2 tables doivent être égales
 - Relie les tables en faisant correspondre toutes les attribut portant le même nom
- À n'utiliser qu'avec prudence...

Remarques sur la jointure interne

74

- Quand l'utiliser?
 - ▣ Lorsque l'on veut faire une sélection a priori sur la « relation de travail »
 - ▣ Équivalent à un l'utilisation d'un CROSS JOIN et d'un WHERE mais moins efficace
- Que se passe-t-il?
 - ▣ Si je fais une jointure interne où aucune des combinaison de tuples ne satisfait la condition?
 - ▣ Si je fais une jointure naturelle sur deux tables n'ayant aucun nom d'attribut commun?
 - ▣ Si je fais une jointure naturelle réflexive?

Jointure externe : OUTER JOIN

75

- Retourne :
 - ▣ Les lignes de chaque table qui satisfont la condition de jointure (idem INNER JOIN)
 - ▣ PLUS : Les lignes de la table [droite/gauche] qui ne la satisfont pas
 - [RIGHT/LEFT/FULL] OUTER JOIN
- Condition de jointure identique à celle du INNER JOIN
- Quand utiliser une jointure externe?
 - ▣ Lorsqu'en plus de la sélection sur la relation de travail, on veut garder toutes les données d'une table en particulier (ou les deux)

Les jointures

76

- Possibilité de plusieurs jointures dans un seul **FROM**
 - ▣ Lecture de gauche à droite
- Possibilité de faire une jointure réflexive
 - ▣ Obligation de renommer (au minimum) une table avec **AS**
 - Attributs disponibles par notation préfixée :
alias_table.nom_attribut
- Possibilité de faire une jointure avec un ensemble de tuples issu d'une requête **SELECT**
 - ▣ Obligation de nommer l'ensemble de tuples avec **AS**
 - À n'utiliser que si nécessaire. Par exemple, pour récupérer un attribut calculé.