Cours de Base De Donnees

Mendy Fatnassi

10 décembre 2020

Table des matières

Chapitre 1 : Generalite

1.1 Introduction

MERISE (Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise) est certainement le langage de spécification le plus répandu dans la communauté de l'informatique des systèmes d'information, et plus particulièrement dans le domaine des bases de données.
Une représentation Merise permet de valider des choix par rapport aux objectifs, de quantifier les solutions retenues, de mettre en œuvre des techniques d'optimisation et enfin de guider jusqu'à l'implémentation.
Merise réussit le compromis difficile entre le souci d'une modélisation précise et formelle, et la capacité d'offrir un outil et un moyen de communication accessible aux non-informaticiens.

Les trois niveaux de représentation des données, sont détaillés ci-dessous :

Niveau conceptuel :

Le modèle conceptuel des données (MCD) décrit les entités du monde réel, en terme d'objets, de propriétés et de relations, indépendamment de toute technique d'organisation et d'implantation des données. Ce modèle se concrétise par un schéma entités-associations représentant la structure du système d'information, du point de vue des données.

Niveau logique :

Le modèle logique des données (MLD) précise le modèle conceptuel par des choix organisationnels. Il s'agit d'une transcription (également appelée dérivation) du MCD dans un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données relationnelle ou réseau, ou autres (cf. section 1.1.2). Les choix techniques d'implémentation (choix d'un SGBD) ne seront effectués qu'au niveau suivant.

Niveau physique :

Le modèle physique des données (MPD) permet d'établir la manière concrète dont le système sera mis en place (SGBD retenu).

1.2 Commande postgres/psql

Pour se connecter a une base de donnes : $psql -U user –d database –h host
ou

$sudo -i -u postgres => cela ouvre l'interpreteur postgres , ensuite taper : $psql et voila nous sommes dans notre base de donnes.

```
Lister les bases de données: \l
Se connecter à une autre base de données : \c nom_base
Detail de tout ce que contient la base : \d
Lister les tables de la base actuelle : \dt
Obtenir les détails d'une table : \dt nom_table
Exécuter un fichier de commandes sql : \i mon_fichier.sql
Obtenir de l'aide postgre : \?
Obtenir de l'aide sql : \help
```

Lorsqu'on se connecte a une BDD on peux gerer le controle de l'acces au niveau de la connexion :
psql -d nomBase
Par defaut nomBase=nomUser .

On peux aussi le faire avec un role :
psql -U nomUser
Par defaut sessionUser=nomUser .

1.3 Catalogues systeme

Les catalogues systemes permettent de stocker toutes les metadonnees des objets contenue dans la base de donnees et chaque base contient son propre

catalogues.
Ils contiennent la liste des tables,utilisateurs,vues ... .
Un catalogues commence par le mot cle pg_nomCatalogue .

Pour afficher les catalogues on peux faire : `\dt *.*`
Ensuite on fait :
SELECT * FROM nomSchema.nomCatalogue ;
SELECT * FROM information_schema.sql_languages ;

Liste Catalogue :
pg_namesapce -> Liste schema `\dn`
pg_views -> Liste vue `\dv`
pg_roles -> Liste role `\du`

Chapitre 2 : Algebre Relationnel

2.1 Forme Normale

1NF => Tout attributs est atomique (non-decomposable).

2NF => Tout attributs ne depend d'une partie de la cles mais de toute la cles.Tout attributs cles ne doivent pas etre defini par des attributs non-cles.

3NF => Pas de DFE(Dependance Fonctionnelle Elementaire) entre attributs cles.tout attributs non-cles ne peux pas definir d'autre attributs non-cles.

4NF => Est du type cles ->-> attribut(muti-valuee)

BCNF=> Est du type cles->attribut

2.2 MEA / MCD

Modele Entité Association ou Modele Conceptuel des Données .

Il est difficile de modéliser un domaine sous une forme directement utilisable par un SGBD. Une ou plusieurs modélisations intermédiaires sont donc utiles, le modèle entités-associations constitue l'une des premières et des plus courantes.
Cardinalité :

-Maillé (non-hiearchique) : 0|1,n et 0|1,n .
L'association devient une table avec importation des 2 cles primaire des tables concernant l'association (2 a n participant).

-Hiéarchique : 1,1 et 0|1,n .
Pas d'attriburts dans l'association (2 participant pas plus) .

-Semi-Hiéarchique : 0,1 et 0|1,n
Possibilite d'attributs dans le type association :
- SI attribut ALORS cas non-hiearchique
- SINON cas hiearchique

2.3 MR / MLD

Modele Relationnnel ou Modele Logique des Données .

Chapitre 3 : Commande sql

3.1 Fonction d'agregation

COUNT : compte du nombre de valeurs non NULL.
SUM : somme de valeurs numériques ou intervalle.
AVG : moyenne de valeurs numériques ou intervalle.
MIN & MAX : valeur minimale/maximale de valeurs numériques, chaîne de caractères ou date.

3.2 Les jointures

S'utilise dans la clause **FROM** .

-Jointure croisee : **CROSS JOIN**
-Jointure interne : **INNER JOIN/NATURAL JOIN**
-Jointure externe : **[RIGHT/LEFT/FULL] OUTER JOIN**

3.3 Les clauses

Il existe 9 clauses donc 7 optionnel et elle possedent des ordres de priorite , commme si dessous :

FROM
WHERE
GROUP BY
HAVING
SELECT
UNION | INTERSECT | EXCEPT
ORDER BY
OFFSET
LIMIT

3.3.1 La Clause GROUP BY

Permet de classer des donnees par groupe , sous-enemble de lignes ayant la meme valeur pour les attributs precises .

Exemple :
SELECT nomAtr FROM nomTbl WHERE nomAtr='louis' GROUP BY nomAtr ORDER BY ASC ;

La clause GROUP BY peux etre preceder de la clause ORDER BY pour preciser un ordre de trie croissant/decroissant (ASC/DESC) et on peux aussi la combiner avec 2 autres clause LIMIT et OFFSET .

LIMIT : nombre de n-uplets a afficher .
OFFSET : Debut , premier n-uplets a afficher .

3.3.2 La Clause HAVING

HAVING est au GROUP BY ce que le WHERE est au SELECT .Si on utilise HAVING on ne met pas de WHERE car c'est lui qui donne des restriction sur les groupes .

Exemple :
SELECT nomAtr FROM nomTbl GROUP BY nomAtr HAVING nomAtr='louis' ;

3.4 Operateur Logique

Dans la clause WHERE on peux evalue un attributs avec : `=, < ,> ,<= ,>=`

On peux aussi utilise des mots cles specifique :
-Etendue : **BETWEEN** valeur1 **AND** valeur2
-Appartenance : **IN** (ensemble_val)
-Correspondance a un modele : **LIKE** modele
-Valeur nul : **IS NULL**

On peux utiliser la negation **NOT** : NOT BETWEEN , NOT IN , IS NOT NULL

Chapitre 4 : DDL - Data Definition Language

4.1 CREATE ALTER DROP

Permet de manipuler des objets SQL CREATE,ALTER et DROP.
En SQL un objets peux etre une DATABASE,SCHEMA,TABLE et VIEW.

CREATE
Definition des éléments de la base de données : tables,champs, clés mais on peux aussi creer d'autre type d'objet que des tables comme une base de donnes une fonction , une vue etc ...

```
CREATE nom_table(
id_ent integer primary key,
nom varchar(50)
);
```

On peux aussi creer une base de donne : CREATE DATABASE nomBase ;

DROP
Supprime des objets SQL .On peux utiliser les option RESTRICT ou CASCADE .
RESTRICT : Suppression impossible si un autre objet en depend .
CASCADe : Suppression des vues et des contraintes de cle etrangeres .

DROP TABLE nomTable ;
DROP DATABASE nomBase ;

ALTER
Ajout,Suppression,Modification de la structure de la table ou de ces tuples.

ALTER TABLE maTable ADD COLUMN nomColonne typeColonne ;
ALTER TABLE maTable DROP COLUMN nomColonne ;

Pour L'ajout des clés etrangere il y a 2 facons de faire , soit dans la definition de la table :

```
CREATE nom_table(
id_ent integer primary key,
nom varchar(50),
id_table2 integer references nom_table2, #ou
id_table2 integer references nom_table2(id_table2),
PRIMARY KEY (id_ent,id_table2)
);
```

Ou alors on peux specifier une contrainte de table apres création de celle-ci :

Contraintes de valuation obligatoire :

ALTER TABLE maTable ALTER COLUMN maColonne memeType NOT NULL ;

ou pour psql

ALTER TABLE maTable ALTER COLUMN maColonne TYPE nomType ;
ALTER TABLE maTable ALTER COLUMN maColonne SET NOT NULL ;

Contraintes de clé primaire :

ALTER TABLE maTable ADD CONSTRAINT PK_maTable PRIMARY KEY (colonnesConstituantLaCléPrimaire) ;

Contraintes de clé étrangère :

ALTER TABLE maTable ADD CONSTRAINT FK_maTable_colCleEtrangere FOREIGN KEY(colCleEtrangere) REFERENCES tableCleAReferencer(colClePrimaire) ;

Contraintes d'unicité :

ALTER TABLE maTable ADD CONSTRAINT UQ_maTable_colTupleUnique UNIQUE (colTupleUnique) ;

4.2 Schema

Une base de donnees peux contenir plusieur schema , qui permet a plusieur utilisateur de travailler sur la meme BDD mais sur des schema different .
En gros un schema sur un espace de nommage qui fait references a la BDD . Le schema par defaut est le public .
Catalogue : pg_namespace

Commande psql : `\dn` -> Liste des schema .

Creation/Suppression
CREATE SCHEMA nomSchema ;
DROP SCHEMA nomSchema [CASCADE|RESTRICT] ;

Creation objet
CREATE OBJET nomSchema.nom_objet () ;
On accede a l'objet d'un schema comme cela : nomSchema.nomObjet

Definir le proprietaire
CREATE SCHEMA nomSchema AUTHORIZATION nomUser ;
ALTER SCHEMA nomSchema OWNER TO newUser ;

Afficher le chemin de parcours des schema : SHOW search_path ;
Modification du chemin : SET search_path TO "louis","$user",public ;

4.3 Vue/VIEW

Une vue correspond a une table virtuelle dans lequelle on vas stocker un resultat d'une requete . Si par exemple on doit souvent faire appele a une requete specifique , on la stockera dans une vue et au lieu de retaper la requete a chaque fois , on feras appelle a la vue .
Catalogue : SELECT * FROM pg_views .

Commmande psql : `\dv` -> liste des vues .

Creation/Suppression :
CREATE VIEW name_emp AS SELECT ename,dname FROM emp NATURAL JOIN dept ;
DROP VIEW name_emp ;
Ensuite pour utiliser la vue on fait un appele classique dans une requete , on peux la combiner avec des clauses si l'on veux detaille l'appelle a la requete :
SELECT * FROM name_emp WHERE ename LIKE `'%A%'` ;

Chapitre 5 : DML-Data Manipulation Language

Manipulation des données : insertion, suppression, modification, extraction, ..

INSERT :

-Creer une nouvelle donnee :
```
INSERT INTO nom_table(nom,age)VALUES('user',5);
```

UPDATE :

Modification des valeurs d'un enregistrement :
```
UPDATE table SET nom_colonne_1 = 'nouvelle valeur' WHERE condition;
```

DELETE :

Suppression d'un enregistrement :
```
DELETE FROM nom_table WHERE condition;
```

Chapitre 6 : DQL-Data Query Language

6.1 Gestion des droits/privileges

Permet de definir des droits pour certain utilisateur de la base , droit de suppression,creation ou d'insertion de donnees sur une ou des tables,schema,vue etc ... .

GRANT : Donne des privilileges .
`GRANT privilege ON objet TO [username|GROUP groupname | PUBLIC] ;`

REVOKE : Supprime des priviliges .
`REVOKE privilileges ON objet FROM {username |GROUP groupname | PUBLIC};`

Pour 'privileges' on peux avoir les valeurs suivante :
SELECT(lecture) , INSERT(insertion/ecriture) , UPDATE , DELETE , ALL .

Pour 'objet' on peux avoir les valeurs suivantes :
ON [DATABASE|TABLE|SCHEMA|LANGUAGE|FUNCTION] nomObjet .

6.2 Role

Un role est une entite qui peut posseder des objets et avoir des droits .On parle d'utilisateur ou d'un groupe d'utilisateur : USER ou GROUP .
Catalogues des roles : pg_roles .

Commande psql : `\du` -> Liste des roles .

Afficher les Roles :
SELECT roleName FROM pg_roles ;

Changer de Role
SET ROLE TO groupUser ;

Creation/Suppression
CREATE ROLE etudiant LOGIN ; ou CREATE USER etudiant ;
CREATE ROLE prof [CREATEDB|SUPERUSER|CREATEROLE] ; -> Le role prof seras super utilisateur ou peux creer une BDD ou d'autre roles .
DROP ROLE etudiant ;

Modification ALTER ROLE etu WITH PASSWORD 'toto13' ;

Chapitre 7 : DCL-Data Control Language

Gestion des transactions.

Chapitre 8 : Langage procedural Pl/PgSql

On peux combiner des requete sql a une partie procedurale comprennant des boucles , des conditions , des variables etc ... .
Pour cela on creer des bloc d'instruction a travers des fonctions .

8.1 SEQUENCE

Une sequence est une table qui permet le generation d'entier avec des fonction predifini pour manipuler la sequence :
-nextval() : valeur suivante .
-currval() : valeur courante .
-setval() : initialisation de la valeur courante .

On creer une sequence de la facons suivante :
CREATE SEQUENCE nomSeq [Clause] ;

-Clause : peux prendre comme valeur -> [INCREMENT BY val]|[MINVALUE|MAXVAMUE val]|[START[WITH] debut]|[OWNED BY table.colonne] .

Exemple d'utilisation :

```
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; --> initialisation a 3
ou
SELECT setval('mon_num',3,false);
```

8.2 FUNCTION SQL

On utilise le mot cle CREATE pour creer une fonction suivi de son nom,parametres d'entree et de sortie .

Synthaxe :

```
CREATE [OR REPLACE] FUNCTION nomFonc ([argtype [, ...]])
RETURNS type_retour AS $name$
  definition
$name$
LANGUAGE nom_langage ;
```

-nom_langage : SQL ou PLPGSQL selon le type de fonction utilisé.
-definition : requete select ou requete DML (Update...) -> SELECT nomAttr FROM nomTableWHERE condition ;

Une fonction SQL peux avoir 2 type de delimiteur soit avec `$name$` ou alors avec les simple quotes " . Si on utilise les `$$` on fermera notre requete avec point-virgule ';' Sinon avec l'utilisation des quotes on place la requete a l'interieure de celle ci sans le point virgule .

Appel : select nomFonc(argtype) ;
Si argtype est une varchar il faut utiliser les quotes " pour passer la chaine de caractere : select nomFonc('text') ;

Exemple :

```
CREATE FUNCTION ajoute(var1 integer, var2 integer) RETURNS integer AS $$
    SELECT $1 + $2;  //ou SELECT var1+var2 ;
$$ LANGUAGE SQL;

ou

CREATE FUNCTION ajoute(var1 integer, var2 integer) RETURNS integer AS
    'SELECT $1 + $2'
LANGUAGE SQL;

SELECT ajoute(1, 2);
```

Voici une autre fonction qui permet de debiter sur un compte par exemple avec une autre requete que SELECT :

```
CREATE FUNCTION tf1 (integer, numeric) RETURNS numeric AS $$
    UPDATE banque SET balance = balance - $2 WHERE no_compte = $1;
    SELECT balance FROM banque WHERE no_compte = $1;
$$ LANGUAGE SQL;
```

Ici ont a une fonction simple SQL retournant seulement 1 resulat (1lig,1col).

8.3 Type Composite

Un type composite est un nouveau type creer par l'utilisateur , un peux comme une structure en C .

```
CREATE TYPE produit_ajoute AS (somme int, produit int);

CREATE FUNCTION ajoute_n_produit (int, int) RETURNS produit_ajoute
AS 'SELECT $1 + $2, $1 * $2'
LANGUAGE SQL;
```

On auras pour resultat 2 colonne l'une pour la somme et l'autre pour le produit qui seras ranger directement dans notre type composite .