INTRODUCTION

Les fonctions SQL exécutent une liste arbitraire d'instructions SQL,renvoyant le résultat de la dernière requête de la liste.Dans la suite nous allons manipuler les fonctions SQL.Après nous allons créer deux tables dont l'identifiant de la seconde table aura pour type le nom de la première table.

1 Création des fonctions SQl

1.1 Création des la base de donnée

Dans cette partie nous avons crée une base de donnée ifnti qui a pour tables etudiant et enseignant

```
1 drop database ifnti;
2 create database ifnti;
3
4 \c ifnti
5 create table etudiant(idetudiant integer primary key,nom varchar,prenom varchar,nivea
  varchar);
6 create table enseignant(idenseignant integer primary key,nom varchar,prenom varchar);
7 insert into etudiant values(100,'sankara','sarata','l2'),(101,'akoh','sawana','l1');
8 insert into enseignant values(100,'SHAM','youssaou'),(101,'SANY','koumai')
```

1.2 Création des fonctions sql

1.2.1 Création des fonctions sql ayant pour type de retour void

Pour créer une fonction sql ayant pour type de retour void on utilise la syntaxe suivante **CREATE FUNCTION function_name() returns void as ' '.** Cette fonction ne retourne rien.
Voici notre première tentative de création de la fonction insert-etudiant

```
*/
create function insert_etudiant(ide integer,nom varchar,prenom varchar,niveau varchar)
returns void as
        insert into etudiant(idetudiant,nom,prenom,niveau)values(ide,nom,prenom,niveau)
        LANGUAGE SQL;
select insert_etudiant(102,'beveri','farida','l2');
```

Une erreur de syntaxe est engendré car nous avons omis les cotes (") entre le **as** et **language sql**.

```
ifnti=> \i test.sql
psql:test.sql:11: ERROR:  syntax error at or near "insert"
LINE 11:  insert into etudiant(idetudiant,nom,prenom,niveau)values(id...
          ^
psql:test.sql:12: ERROR:  syntax error at or near "LANGUAGE"
LINE 1: LANGUAGE SQL;
        ^
psql:test.sql:13: ERROR:  function insert_etudiant(integer, unknown, unknown, un
known) does not exist
LINE 1: select insert_etudiant(102,'beveri','farida','l2');
               ^
HINT:  No function matches the given name and argument types. You might need to
add explicit type casts.
```

Pour corriger cette erreur nous avons mis les cotes (") entre le **as** et **language sql**.

```
create function insert_etudiant(ide integer,nom varchar,prenom varchar,niveau varchar)
returns void as'
        insert into etudiant(idetudiant,nom,prenom,niveau)values(ide,nom,prenom,niveau)
        'LANGUAGE SQL;
select insert_etudiant(102,'beveri','farida','l2');
```

Après l'exécution nous avons obtenus le résultat suivant

```
add explicit type casts.
ifnti=> \i test.sql
CREATE FUNCTION
 insert_etudiant
-----------------

(1 row)

ifnti=> table ed

ifnti=> table etudiant ;
 idetudiant |   nom   | prenom | niveau
------------+---------+--------+--------
        100 | sankara | sarata | l2
        101 | akoh    | sawana | l1
        102 | beveri  | farida | l2
(3 rows)

ifnti=>
```

1.2.2 Les fonctions ayant pour type de retour une ligne

Les fonctions SQL peuvent prendre des paramètres en entré et retourner une seule ligne.Pour déclarer les paramètres on écrit le nom du paramètre suivit de son type mais pour le retour nous précisions le type de retour

.Exemple 1:Nous allons crée une fonction qui ne prend rien en paramètre et retourne une chaîne

```
create or replace function niveau_etu() returns varchar as'
        select niveau from etudiant ;'
       LANGUAGE SQL;
select  niveau_etu();
```

```
 niveau_etu
------------
 l2
(1 row)

ifnti=>
```

Exemple 2:Nous allons crée une fonction qui prend l'identifiant d'un enseignant en paramètre et retourne son nom qui est une chaîne

```
create or replace function nom_enseignant(id integer) returns varchar as'
        select nom from enseignant where idenseignant=id ;'
       LANGUAGE SQL;
select  nom_enseignant(100);
```

```
ifnti=> \i test.sql
CREATE FUNCTION
 nom_enseignant
----------------
 SHAM
(1 row)

ifnti=>
```

1.3 Fonctions SQL avec des paramètres en sortie

Nous pouvons définir des fonctions sql dont sont type de retour est indiqué dans les paramètres .Pour declarer un paramètre en sortie on utilise le mot-clé **OUT** et les paramètres en entrés est précéder de du mot-clé **IN**.Le mot-clé IN est optionnel et le OUT est obligatoire exemple 1

```
create or replace function niveau_etudiant(nom_ varchar,prenom_ varchar,OUT niveau varchar) as
        select niveau from etudiant where nom=nom_ and prenom=prenom_ ;'
       LANGUAGE SQL;
select  niveau_etudiant('sankara','sarata');
```

```
ifnti=> \i test.sql
CREATE FUNCTION
 niveau_etudiant
-----------------
 l2
(1 row)

ifnti=>
```

Une erreur est produite parce que nous avons pas précisé le paramètre de retour donc en appelant la fonction il réclame trois arguments.

exemple 2

```
create or replace function n_etudiant(nom_ varchar,prenom_ varchar,niveau varchar
        select niveau from etudiant where nom=nom_ and prenom=prenom_ ;'
       LANGUAGE SQL;
select  n_etudiant('sankara','sarata');
```

```
ifnti=> \i test.sql
psql:test.sql:15: ERROR:  function result type must be specified
psql:test.sql:16: ERROR:  function n_etudiant(unknown, unknown) does not exist
LINE 1: select  n_etudiant('sankara','sarata');
                ^
HINT:  No function matches the given name and argument types. You might need to
add explicit type casts.
```

1.4 Création d'une fonction sql ayant pour type de retour SETOF sometype

Le type de retour **SETOF sometype** nous permet de retourner plusieurs lignes mais nous avons remarqué le sometype est derivé par le nom de la table et il retourne tous les attributs de la de table.

Nous avons fait une premiere tentative en essayant de sélectionné les noms et les prénoms des étudiants.

```
13 create or replace function select_etudiant(l2 varchar) returns SETOF etudiant as'
14          select nom,prenom from etudiant where niveau=l2;'
15         LANGUAGE SQL;
16 select   select_etudiant('l2')
```

une erreur de retour a été engendré,il est impossible de sélectionner les colonnes d'une table avec le retour **SETOF**.

```
ifnti=> \i test.sql
psql:test.sql:15: ERROR:  return type mismatch in function declared to return et
udiant
DETAIL:  Final statement returns character varying instead of integer at column
1.
```

Pour corriger cette erreur nous avons été obligée de sélectionnée tous les colonnes de la table.

```
create or replace function select_etudiant(l2 varchar) returns SETOF etudiant as'
         select * from etudiant where niveau=l2;'
        LANGUAGE SQL;
select   select_etudiant('l2')
```

Après l'exécution nous avons obtenus le résultat suivant:

```
ifnti=> \i test.sql
CREATE FUNCTION
     select_etudiant
-------------------------
 (102,beveri,farida,l2)
 (100,sankara,sarata,l2)
(2 rows)

ifnti=>
```

2 Fonctions SQL renvoyant TABLE

Dans Une fonction sql nous pouvons retourner un ensemble de ligne avec syntaxe RETURNS TABLE(colonnes).

```
create function info_produit(idp integer)returns table(nom varchar,prix varchar)as
$$
        select libelle,montant from produit where id=idp;
        $$
        LANGUAGE SQL;
select info_produit(10);
```

```
ifnti=> \i test.sql
CREATE FUNCTION
 info_produit
--------------
 (bic,100)
(1 row)

ifnti=>
```

3 Créons les tables personne client

Dans cette partie nous allons crée une table personne(idpersonne,nom,prenom) et une table client(identifiant,adresse)

```
create table personne(idpersonne integer primary key,nom varchar,prenom varchar);
create table client(identifiant personne,adresse varchar);
```

```
ifnti=> table personne ;
 idpersonne | nom | prenom
------------+-----+--------
(0 rows)

ifnti=> table client ;
 identifiant | adresse
-------------+---------
(0 rows)
```

4 Inserons dans les tables personne et clients

Pour faire l'insertion dans une table ayant pour colonne un type composite on ouvre les parenthèses pour saisir les champs correspondant et les séparez par des virgules.

Pour mettre un attribut vide on met les parenthèses du type composite entre guillemets unique et virgule . Pour mettre un attribut la valeur null on met les parenthèses du type composite entre guillemets unique et guillemets doubles pour la valeur nulle et les champs type chaîne.

```
insert into personne values(100,'sankara','sarata');
insert into client values((100,'sankara','sarata'),'sokode');
insert into client values('(101,"akoh",)','sokode'); --enregistrement d'un client ayant pour pr
vide
insert into client values('(102,"beveri","")','sokode');--enregistrement d'un client ayant pour
prenom null
```

```
ifnti=> table personne ;
 idpersonne |   nom   | prenom
------------+---------+--------
        100 | sankara | sarata
(1 row)

ifnti=> table client ;
     identifiant      | adresse
----------------------+---------
 (100,sankara,sarata) | sokode
 (101,akoh,)          | sokode
 (102,beveri,"")      | sokode
(3 rows)

ifnti=>
```

5 Accéder aux types composite

Pour accéder à un champ d'une colonne composite on écrit le nom de colonne point le nom du champ ou (table-name.colonne-composite-name.nom-champ).notons bien le nom de colonne doit être toujours entre parenthèse .exemple

select (identifiant).nom from client;

```
   nom
---------
 sankara
 akoh
 beveri
(3 rows)

ifnti=>
```

select (client.identifiant).prenom from client;

```
 prenom
--------
 sarata


(3 rows)

ifnti=>
```

6 Surcharge des fonctions