Module EF2

Bases de données

Licence Professionnelle - 2015/2016

EF2 – Create table

```
CREATE TABLE nom_relation
  (  { définition_attribut
  | contrainte_de_relation } [ , … ]
  )
```

où définition_attribut :

```
nom_att nom_type [ DEFAULT val ]
                [ contrainte_attribut ] [ … ]
```

- [ ] : optionnel
- { … | … } : choix
- [, …] ou […] : répétition de l'élément précédent

EF2 – Types

- Numériques :

  `smallint, integer, numeric(n, d), real,…`

- Caractères :

  `char, char(n), varchar(n), text`

- Dates :

  `date, time, timestamp, interval`

- **`boolean (FALSE, TRUE)`**

EF2 – Contraintes

- **`CONSTRAINT` *`nom_cont`*** : nomme une contrainte
- Contraintes d'attributs :

```
primary key, unique,
not null, check (condition),
references nom_relation(attribut)
```

- Contraintes de relation :

```
primary key (liste_attributs)
unique (liste_attributs)
check (condition)
foreign key (liste_attributs) references
            nom_relation (liste_attributs)
```

EF2 – Vues

- Une vue se crée comme résultat d'une requête

- **CREATE VIEW nom [(liste-attributs)]**
  AS SELECT … FROM …

EF2 – Vues

- Seule la définition de la vue est stockée. Il n'y a pas de création de nouveaux n-uplets
- La vue se manipule comme une relation
- Son contenu est dynamique (calculé à l'exécution)
- Par défaut, seul le **SELECT** est autorisé

EF2 – Vues - exemple

Dans la base Zoo :

Animal (noma, espece, pays, datea, numc)

1) Créer une vue donnant les noms des koalas et les cages dans lesquelles ils se trouvent.

EF2 – Vue - Exemple

```
Animal (noma, espece, pays, datea, numc)

CREATE VIEW VcageKoala
AS   SELECT noma, numc
     FROM Animal
     WHERE espece='koala';
```

EF2 – Droits

- Le SGBD fournit des mécanismes pour accorder ou supprimer des privilèges (droits d'accès) sur des objets

- Par défaut, un utilisateur possède tous les droits sur les objets qu'il a créés

- Les règles d'organisation (RO) définissent qui a accès à quelles données et pour quoi faire

EF2 – Droits sur les objets

- **GRANT {privilège, … | ALL}**
  [(attribut, …)]
  ON {relation | vue} [,…]
  TO {utilisateur|groupe|PUBLIC}[,…]
- **REVOKE … ON … FROM …**

- Les privilèges sont **SELECT, INSERT, UPDATE, DELETE**

EF2 – Droits - Exemples

Dans la base Zoo :

Animal (noma, especе, pays, datea, numc)

EstMalade (noma, nommal, datem, remede)

2) Donner les droits suivants :

- L'utilisateur **veto** peut tout faire sur **EstMalade**
- Les utilisateurs du groupe **gardien** peuvent consulter les animaux et les changer de cage

EF2 – Droits - Exemples

- L'utilisateur **veto** peut tout faire sur **EstMalade**

```
GRANT ALL ON EstMalade TO veto;
```

- Les utilisateurs du groupe **gardien** peuvent consulter les animaux et les changer de cage

```
GRANT SELECT ON Animal TO gardien;
GRANT UPDATE(numc)ON Animal TO gardien;
```

EF2 – Procédures - Objectifs

Une procédure, stockée dans la base, permet de regrouper des instructions

- Pour effectuer un traitement intermédiaire (réutilisable dans une requête ou une autre procédure)
- Pour assurer une interface adaptée (à un utilisateur, ou à une application particulière)
- Pour augmenter les performances en effectuant les traitements sur le serveur plutôt que le client et en évitant les allers/retours entre le client et le serveur

EF2 – Procédures stockées

- Une procédure stockée est une fonction qui regroupe des instructions (SQL ou autres)
- Elle est pré-compilée, stockée dans la base et exécutée à la demande ou automatiquement
- Elle est écrite
  - en langage de requêtes SQL
  - en langage C
  - en langage procédural (par exemple PL/pgSQL)

EF2 – Procédures stockées

- **CREATE FUNCTION** *nom*

```
( [[modearg] nomarg typearg [, ...] ] )
      [ RETURNS type_resultat ]
  AS $$definition$$
  LANGUAGE nom_lang ;
```

- **DROP FUNCTION** *nom*

```
( [ [modearg] [nomarg] typearg [, ...] ] )
```

EF2 – PL/pgSQL

- Langage procédural qui permet d'écrire :
  - des instructions SQL
  - des structures de contrôle (if, while, …)

- Avantages :
  - Allier la puissance d'un langage de programmation à la facilité d'utilisation du SQL
  - Permettre des traitements complexes

EF2 – PL/pgSQL - paramètres

- **CREATE FUNCTION** ***nom***

```
( [ [modearg] nomarg typearg [, ...] ] )
    [ RETURNS type_resultat ] ...
```

- Les modes pour les paramètres sont :

  IN (par défaut), **INOUT, OUT**

- **RETURNS** ***void*** : la fonction n'a pas de résultat
- Dans appel de fonction et dans un **DROP FUNCTION**, les paramètres résultats (**OUT**) n'apparaissent pas.

EF2 – PL/pgSQL - paramètres

- **CREATE FUNCTION** ***nom***

([[*modearg*] *nomarg typearg* [, ...]])
[RETURNS *type_resultat*]

- Les types utilisables sont :
  - les mêmes qu'en SQL
  - **nom_table.nom_att%TYPE**

  et les types lignes (n-uplet)
  - **nom_table%ROWTYPE** (pas pour le résultat)
  - **RECORD**

EF2 – exemple

Dans la base Zoo :

Animal (noma, espece, pays, datea, numc)

EstMalade (noma, nommal, datem, remede)

3) Créer une fonction qui renvoie le nombre de maladies qu'ont eues les animaux d'une espèce donnée :

```
CREATE FUNCTION nbmaladies
(IN esp Animal.espece%TYPE) RETURNS numeric (3)
AS $$
...
$$ LANGUAGE 'plpgsql' ;
```

EF2 – PL/pgSQL - définition

- La définition (le corps) d'une fonction est un bloc :

```
[DECLARE déclarations]
BEGIN
    instructions;
END;
```

- Commentaires :

```
-- sur une ligne
/* bloc éventuellement sur
plusieurs lignes*/
```

EF2 – PL/pgSQL - déclarations

- Toutes les variables doivent être déclarées, sauf les variables des boucles FOR

```
nom [CONSTANT] type [NOT NULL]
                    [DEFAULT expression];
```

- Les types utilisables sont :
  - les mêmes qu'en SQL
  - **nom_table.nom_att%TYPE**
  - **nom_table%ROWTYPE**
  - **RECORD**

EF2 – PL/pgSQL - instructions

- **NULL;**
- Affectation : **:=**
- Instruction conditionnelle :

```
IF … THEN … ELSIF … ELSE … END IF;
```

- Itérations

```
WHILE …
LOOP
   …
END LOOP;
```

```
LOOP
   …
EXIT WHEN …
END LOOP;
```

```
FOR … IN [REVERSE]…
LOOP
   …
END LOOP;
```

EF2 – PL/pgSQL - instructions

- Toutes les commandes SQL qui ne renvoient pas de résultat :

  `INSERT, UPDATE, GRANT, CREATE TABLE, …`

- Un **`SELECT`** ne peut donc pas être exécuté sauf si on affecte le résultat

  `SELECT … INTO … FROM …`

- On peut utiliser les variables et les paramètres dans les requêtes.

EF2 – PL/pgSQL - instructions

- Affichage d'un message ou d'une erreur

```
RAISE NOTICE 'message'[, …];
RAISE EXCEPTION 'message'[, …];
```

Le message est éventuellement suivi d'expressions qui remplacent les % du message.

- Résultat : **RETURN …** ou **RETURN QUERY …**

EF2 – exemple

```
CREATE FUNCTION nbmaladies
 (IN esp Animal.espece%TYPE) RETURNS numeric (3)
AS $$ DECLARE nb numeric(3);
BEGIN
      select count(*)
                              into nb
      from EstMalade E, Animal A
      where E.noma=A.noma and espece = esp;

      return nb;
END;
$$ LANGUAGE 'plpgsql' ;
```

EF2 – exemple

Dans la base Zoo, un gardien est malade.

Gardien(numg, nomg, adresse, dateg)

Soccupe(numg, numc)

4) Ecrire une fonction qui supprime ce gardien de la relation **Soccupe** et renvoie son nom et son adresse pour les formalités administratives.

EF2 – exemple

```
CREATE FUNCTION GMalade (IN num numeric(2),
        OUT nom varchar(10), OUT adr varchar (10))
AS $$ BEGIN
  select nomg, adresse into nom, adr
  from Gardien where numg = num;

  if not found then
    raise exception '% n''existe pas', num;
  end if;

  delete from Soccupe where numg = num;
END; $$ LANGUAGE 'plpgsql' ;
```

EF2 – Appel de fonction

- **`SELECT nom_fonction (…);`**
- **`SELECT …`**
  `FROM nom_fonction (…)`
  `WHERE …;`
- Les paramètres effectifs doivent être donnés dans l'ordre (notation par position)
- Des erreurs de syntaxe peuvent apparaître à l'exécution

EF2 – exemples

```
CREATE FUNCTION nbmaladies
(IN esp Animal.espece%TYPE) RETURNS numeric (3)…
```

- `SELECT nbmaladies('koala');`

```
CREATE FUNCTION GMalade (IN num numeric(2),
      OUT nom varchar(10), OUT adr varchar (10))…
```

- `SELECT Gmalade(1);`
- `SELECT nom from Gmalade(1);`
- `SELECT (Gmalade(1)).nom;`

EF2 – PL/pgSQL- Compléments

- L'instruction **FOR** peut permettre de parcourir les lignes résultats d'une requête **SELECT**.

```
FOR var IN requête
LOOP   …
END LOOP;
```

- ***var*** doit être déclaré de type ***record***

EF2 – exemple

Dans la base Zoo :

Animal (noma, especе, pays, datea, numc)

5) Ecrire une fonction qui affiche pour chaque animal le nombre d'animaux de sa cage et de la même espèce.

EF2 – exemple

```
CREATE FUNCTION NbMemeCageEspece() RETURNS void
AS $$ DECLARE res RECORD; nb numeric(2);
BEGIN
  for res in   select noma, espece, numc
               from Animal
  loop
     select count(*)-1 into nb from Animal
     where numc=res.numc and espece=res.espece;
     raise NOTICE '% cohabite avec % animaux de
son espèce', res.noma, nb;
  end loop;
END; $$ LANGUAGE 'plpgsql' ;
```

IUT2 Grenoble Université Pierre-Mendès-France

EF2 – PL/pgSQL- Compléments

- En PL/pgSQL, toute requête qui ne renvoie pas de n-uplet peut être exécutée avec **PERFORM** :

  - Appel de fonction dans une fonction :

```
PERFORM nom_fonction (…);
```

  - Exécution d'un **SELECT** sans **INTO** :

```
PERFORM … FROM … WHERE …
```

(utile si effet de bord)

EF2 – PL/pgSQL - Compléments

- Une fonction peut renvoyer un ensemble de lignes (ou une table)
  - … **RETURNS SETOF** ***nom_table***
  - … **RETURNS TABLE** (***nom type*** [, …])
- Les lignes résultats sont générées :
  - une par une par des

    SELECT … INTO … suivis de **RETURN NEXT …;**
  - ou comme résultat d'une requête par un

    RETURN QUERY … ;
- La fonction se termine par : **RETURN;**

EF2 – exemple

Dans la base Zoo :

Animal (noma, espece, pays, datea, numc)

6) Réécrire la fonction de la question 7) pour qu'elle retourne les résultats au lieu de les afficher.

```
CREATE FUNCTION NbMemeCageEspece2()
RETURNS TABLE (   nom   Animal.noma%TYPE,
                  nb numeric(2))
AS $$
```

EF2 – exemple

```
DECLARE res RECORD;
BEGIN
  for res in   select noma, espece, numc
               from Animal
  loop
     select count(*)-1 into nb
     from Animal
     where numc=res.numc and espece=res.espece;
     return NEXT res.noma, nb;
  end loop;
  return;
END;
```

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – exemple

Ou

```
CREATE FUNCTION NbMemeCageEspece2() RETURNS
TABLE (nom  Animal.noma%TYPE, nb numeric(2))
AS $$ BEGIN
  return query select noma,(select count(*)
                            from Animal
                            where numc=A.numc
                            and espece = A.espece)
                  from Animal A;
  return;
END; $$ LANGUAGE 'plpgsql' ;
```

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – Curseurs

Toutes les requêtes SQL sont associées à un curseur qui représente la zone mémoire utilisée pour analyser et exécuter la requête.

Un curseur peut être implicite ou explicite

EF2 – Curseurs - Objectifs

Un curseur explicite permet d'encapsuler un **SELECT** pour pouvoir l'exécuter en plusieurs fois

- Pour traiter plus facilement les n-uplets résultats
- Pour ne pas surcharger la mémoire du client
- Pour une fonction qui renvoie un grand ensemble de n-uplets

EF2 – Curseur - déclaration

- Un curseur doit être déclaré.
  - Curseur non lié :

```
nom refcursor
```

  - Curseur lié à une requête :

```
nom CURSOR FOR SELECT ...
```

  - Curseur lié à une requête paramétrée :

```
nom CURSOR (nomarg typearg [,…])FOR SELECT …
```

EF2 – Curseur - ouverture

- Un curseur doit être ouvert avant utilisation.
  - Curseur non lié :

```
OPEN nom FOR requete
```

  - Curseur lié

```
OPEN nom [(paramètres)]
```

EF2 – Curseur - utilisation

- Pour récupérer une ligne d'un curseur :

  FETCH *nom* INTO *cible*

- où ***cible*** est :
  - une variable n-uplet (**…%ROWTYPE**)
  - une variable record
  - une liste de variables séparées par des virgules
- à utiliser, par exemple, dans une boucle :

  LOOP … EXIT WHEN *not found;* END LOOP

EF2 – Curseur - fermeture

- Un curseur doit être fermé après utilisation pour libérer les ressources ou pour pouvoir le ré-ouvrir.

```
CLOSE nom
```

EF2 – Curseur - exemple

Dans la base Zoo :

Animal (noma, espece, pays, datea, numc)

7) Créer une fonction qui affiche les noms des animaux d'une espèce donnée et renvoie l'âge du plus vieux :

```
CREATE FUNCTION RechAnimaux
      (IN esp varchar(10)) RETURNS interval
AS $$
...
$$ LANGUAGE 'plpgsql' ;
```

EF2 – Curseur - exemple

```
DECLARE res RECORD;
        d date DEFAULT now();
        c CURSOR FOR SELECT noma, datea
                     FROM Animal WHERE espece=esp;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO res;
        EXIT WHEN not found;
        IF res.datea < d THEN d := res.datea; END IF;
        RAISE NOTICE '%',res.noma;
    END LOOP;
    CLOSE c;
    RETURN age(d);
END;
```

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – Curseur dans un FOR

- Une variante du **FOR** permet l'itération sur les lignes renvoyées par un curseur.

```
FOR var IN nom_curseur_lié [(paramètres)]
LOOP   …
END LOOP;
```

- **OPEN**, **FETCH**, **CLOSE**, et la déclaration de ***var*** se font automatiquement (de type ***record***).
- A chaque passage dans la boucle, une nouvelle ligne du curseur est affectée à ***var.***

EF2 – Curseur - exemple

```
DECLARE
        d date DEFAULT now();
        c CURSOR FOR SELECT noma, datea
                     FROM Animal WHERE espece = esp;
BEGIN
    FOR res in c
    LOOP
        IF res.datea < d THEN d := res.datea; END IF;
        RAISE NOTICE '%',res.noma;
    END LOOP;

    RETURN age(d);
END;
```

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – Curseur implicite

- Une autre variante du **FOR** permet même de ne pas déclarer le curseur.

```
FOR var IN requête
LOOP    …
END LOOP;
```

- Le curseur est implicite.
- ***var*** doit être déclaré de type ***record***

EF2 – Curseur - exemple

```
DECLARE
        d date DEFAULT now();
        res record;
BEGIN
    FOR res in      SELECT noma, datea
                    FROM Animal WHERE espece = esp
    LOOP
        IF res.datea < d THEN d := res.datea; END IF;
        RAISE NOTICE '%',res.noma;
    END LOOP;

    RETURN age(d);
END;
```

EF2 – Triggers - Objectifs

Un trigger (déclencheur) permet d'exécuter automatiquement une fonction lorsque certains événements se produisent.

- Pour vérifier des contraintes d'intégrités complexes (dynamiques, sur plusieurs relations, …)
- Pour annuler ou propager des modifications
- Pour tenir un historique des actions sur la base

EF2 – Triggers

- **CREATE TRIGGER** nom_trigger
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE [OR ...] }
ON nom_relation
FOR EACH { ROW | STATEMENT }
EXECUTE PROCEDURE nom_fonction ()

- **DROP TRIGGER** nom_trigger
ON nom_relation

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – Triggers

- Description de l'événement déclencheur :
  - Sur quelle relation?
  - Pour quelle(s) modification(s)?
  - A quel moment? (avant ou après)
  - Pour chaque n-uplet ou instruction (une seule fois)?
- La fonction à exécuter :
  - Définie avant le trigger
  - Sans paramètre, retourne le type **`trigger`**
  - Écrite, par exemple, en **`plpgsql`**
  - **`new`** et **`old`** sont 2 variables spéciales

EF2 – Trigger

- Les variables **`new`** et/ou **`old`** sont déclarées automatiquement et font référence aux n-uplets

règle	new	old
ON INSERT	le nouveau n-uplet à insérer	/
ON UPDATE	la nouvelle valeur du n-uplet	le n-uplet à modifier
ON DELETE	/	le n-uplet à supprimer

EF2 – Trigger- exemple

Dans la base Zoo, on enregistre un nouvel animal.

Animal (noma, espece, pays, datea, numc)

8) Ecrire un trigger qui refuse l'insertion si **datea** est postérieure à la date du jour.

a) Définir l'événement

b) Ecrire la fonction

c) Ecrire le trigger

EF2 – Trigger- exemple

a) Définir l'événement

- Sur quelle relation?
  → **`Animal`**
- Pour quelle(s) modification(s)?
  → **`Insert`**
- A quel moment? (avant ou après)
  → **`Before`**
- Pour chaque n-uplet ou instruction?
  → **`For each row`**

EF2 – Trigger- exemple

b) Ecrire la fonction

```
CREATE FUNCTION f_newanimal() RETURNS trigger
AS $$
BEGIN
  if new.datea > now() then
     raise exception 'insertion refusée';
  end if;

  return ???;  -- pour confirmer l'insertion
END;
$$ LANGUAGE 'plpgsql' ;
```

EF2 – Trigger- exemple

c) Ecrire le trigger

```
CREATE TRIGGER t_newanimal
BEFORE INSERT ON Animal
FOR EACH ROW
EXECUTE PROCEDURE f_newanimal();
```

EF2 – Triggers - résultat

Trigger		Pour valider l'action	Pour annuler l'action
FOR EACH ROW	AFTER	RETURN NULL; (valeur de retour ignorée)	RAISE EXCEPTION …
	BEFORE INSERT	RETURN new; *	RETURN NULL; ou RAISE EXCEPTION …
	BEFORE UPDATE	RETURN new; *	
	BEFORE DELETE	RETURN old;	
FOR EACH STATEMENT		RETURN NULL; (valeur de retour ignorée)	RAISE EXCEPTION …

* **new** initial ou modifié ou autre n-uplet de même schéma

EF2 – Trigger- exemple

```
CREATE FUNCTION f_newanimal() RETURNS trigger
AS $$

BEGIN
  if new.datea > now() then
     raise exception 'insertion refusée';
  end if;
  return new;
END;
$$ LANGUAGE 'plpgsql' ;
```

EF2 – Trigger- exemple

Dans la base Zoo, un canari ayant la grippe aviaire et pas de traitement doit être isolé.

Animal (noma, espece, pays, datea, numc)
EstMalade (noma, nommal, datem, remede)

9) Ecrire un trigger qui impose l'isolement à l'infirmerie comme remède.

EF2 – Trigger- exemple

a) Définir l'événement

c) et écrire le trigger (à placer après la fonction)

```
CREATE TRIGGER t_canari
BEFORE INSERT OR UPDATE ON Estmalade
FOR EACH ROW
EXECUTE PROCEDURE f_canari ();
```

EF2 – Trigger- exemple

```
CREATE FUNCTION f_canari() RETURNS trigger
AS $$ DECLARE esp varchar(10);
BEGIN      select espece into esp
           from Animal
           where noma = new.noma;
  if esp='canari' and new.nommal= 'H5N1'
  and new.remede is null
  then
    new.remede := 'infirmerie';
  end if;
  return new;
END; $$ LANGUAGE 'plpgsql' ;
```

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – Trigger- exemple

Gardien(numg, nomg, adresse, dateg)

Soccupe(numg, numc)

10) Afficher un message lorsqu'il existe au moins un gardien qui ne s'occupe d'aucune cage.

Par exemple, lors d'un

```
Update Soccupe set numg=2 where numg=1;
```

EF2 – Trigger- exemple

```
CREATE TRIGGER t_socc
AFTER UPDATE OR DELETE ON Soccupe
FOR EACH STATEMENT
EXECUTE PROCEDURE f_inactif ();

CREATE TRIGGER t_gard
AFTER INSERT ON Gardien
FOR EACH STATEMENT
EXECUTE PROCEDURE f_inactif ();
```

EF2 – Trigger- exemple

```
CREATE FUNCTION f_inactif() RETURNS trigger
AS $$
BEGIN
  if exists (select numg from Gardien
where numg not in (select numg from Soccupe))
  then
    raise NOTICE 'Il existe des gardiens qui
ne s''occupent d''aucune cage';
  end if;
  return NULL;
END; $$ LANGUAGE 'plpgsql' ;
```

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – Triggers - variables

Nom	Type	Valeur
TG_WHEN	text	BEFORE, AFTER
TG_LEVEL	text	ROW, STATEMENT
TG_OP	text	INSERT, UPDATE, DELETE
TG_TABLE_NAME	name	Nom de la table qui a déclenché le trigger

EF2 – Trigger- exemple

```
CREATE FUNCTION f_inactif() RETURNS trigger
…
then
  raise NOTICE ’Suite à un % dans la table %’,
TG_OP, TG_TABLE_NAME;

  raise NOTICE ’il existe des gardiens qui ne
s’’occupent d’’aucune cage’;
end if;
…
```

IUT 2 Grenoble Université Pierre-Mendès-France

EF2 – Trigger

Si plusieurs triggers sont définis pour la même action sur la même table, ils s'exécutent dans l'ordre suivant

FOR EACH STATEMENT, BEFORE
FOR EACH ROW, BEFORE
FOR EACH ROW, AFTER
FOR EACH STATEMENT, AFTER

puis, dans chaque catégorie, dans l'ordre alphabétique.