Procedural Language for SQL (PL/PGSQL)

André Miralles

Langage *PL/SQL*

- Notion de **bloc** en PL/PGSQL
- Messages et gestion des erreurs
- Procédures stockées
- Fonctions stockées
- Triggers (déclencheurs)
- Transactions
  - Cf. Cours spécifique

Structure d'un *Bloc*

» **Syntaxe**

```
> DO {$Nom Bloc$ | $$}
> DECLARE                                   -- Section facultative
    + Section déclaration
> BEGIN                                     -- Section obligatoire
    + Section Corps-du-bloc
> EXCEPTION                                 -- Section facultative
    + Section traitement des erreurs
> END
> {$Nom Bloc$ | $$};
```

Imbrication des *Blocs*

» **Syntaxe**

```
> DO {$Nom Bloc$ | $$}
> DECLARE                                   -- Section facultative
    + Section déclaration
> BEGIN                                     -- Section obligatoire
    + Section Corps-du-bloc
```

```
> DO {$Nom Bloc$ | $$}
> DECLARE                                   -- Section facultative
    + Section déclaration
> BEGIN                                     -- Section obligatoire
    + Section Corps-du-bloc
> EXCEPTION                                 -- Section facultative
    + Section traitement des erreurs
> END
> {$Nom Bloc$ | $$};
```

```
> EXCEPTION                                 -- Section facultative
    + Section traitement des erreurs
> END
> {$Nom Bloc$ | $$};
```

Messages vers la console et *Erreurs*

- RAISE ;
- RAISE [niveau] USING option = expression [, ...];
- RAISE [niveau] SQLSTATE 'état_sql' [USING option = expression [, ...]];
- RAISE [niveau] nom_condition [USING option = expression [, ...]];
- RAISE [niveau] 'format' [, expression [, ...]] [USING option = expression [, ...]];

- Paramètre NIVEAU
  - DEBUG, LOG, INFO, NOTICE, WARNING ou EXCEPTION
    - Attention : EXCEPTION est le paramètre par défaut

Messages vers la console et *Erreurs*

- **Paramètre OPTION de USING**
  - **MESSAGE**
    - **Texte du message d'erreur**
  - **DETAIL**
    - **Message de détail sur l'erreur**
  - **HINT**
    - **Message de conseil sur l'erreur**
  - **ERRCODE**
    - **Code d'erreur SQLSTATE**
      - **Nom de condition de l'Annexe A**
      - **Codes d'erreurs de PostgreSQL™**
      - **Code SQLSTATE sur cinq caractères**
  - **COLUMN, CONSTRAINT, DATATYPE, TABLE, SCHEMA**
    - **Nature de l'objet**

Messages vers la console et *Erreurs*

- **Exemples**
  - RAISE INFO 'Attention danger';
  - RAISE WARNING 'Erreur sur l''indentifiant % de la table %', id, tableName;
  - RAISE WARNING 'Erreur sur l''indentifiant ' || id || ' de la table %', tableName;
  - RAISE WARNING 'Erreur sur l''indentifiant ' || id || ' de la table ' || tableName;
  - RAISE EXCEPTION 'Cet ID n''existe pas --> %', user_id
    - USING HINT = 'Please check your user id';
  - RAISE division_by_zero;
    - division_by_zero => Cf. message annexe A
  - RAISE SQLSTATE '22012';

Section *Declare*

- **Déclaration de variables**
  - NOM_VARIABLE [CONSTANT] TYPE [COLLATE collation_name] [NOT NULL] [{DEFAULT|:=|=} expression];
  - Paramétre NOM-VARIABLE
    - Chaîne de caractères
  - Paramétre TYPE
    - Integer, Numeric, Varchar, Text, Date, etc.
- **Quelques exemples**
  - Quantite integer DEFAULT 32;
  - Longueur numeric(5);
  - Message CONSTANT varchar := 'Erreur de lecture';
  - Value integer := 10;

Section *Declare*

- **Déclaration de tableaux**
  - A préciser sur le paramétre TYPE
    - Integer[], Numeric[], Varchar[], Text[], Date[], etc.
- **Exemple**
  - Quantite integer[];
- **Déclarations de variables conforme à une TABLE**
  - NOM_VARIABLE NOM_TABLE%ROWTYPE;
  - NOM_VARIABLE NOM_TABLE.NOM_ATTRIBUT%TYPE;
- **Exemple**
  - Create Table tablePersonne (idPersonne integer, nom varchar, prenom varchar, age integer);
  - Declare
    - tuplePersonne tablePersonne %ROWTYPE;
      - {12, 'Dupond', 'Luc', 21}
    - nomPersonne tablePersonne .name%TYPE;
      - 'Dupond'

Section *Declare*

- **Déclaration du TYPE RECORD**
  - Similaire à une variable de TYPE
  - Pas de structure interne
    - Affectation possible d'un tuple ou d'une valeur simple dans la même variable

- **Quelques exemples**
  - Create Table tablePersonne (idPersonne integer, nom varchar, prenom varchar, age integer);
  - Declare
    - tuplePersonne Personne%ROWTYPE;
    - nomPersonne Personne.nom%TYPE;
    - currentValue record;
  - Begin
    - currentValue := tuplePersonne;
    - currentValue := nomPersonne;

Section *Corps du bloc*

- Instructions d'**affectation**
- Instructions du **langage SQL**
  - CLOSE, COMMIT, DELETE, FETCH, INSERT, LOCK, OPEN, ROLLBACK, SAVEPOINT, SELECT, SET, TRANSACTION, UPDATE
- Instructions de contrôle **conditionnelles** ou **répétitives**
- Instructions de **gestion de curseurs**
- Instructions de **gestion des erreurs**

Instructions d'*Affectation d'une variable*

- **Exemple de bloc**

```
DO $$
DECLARE
    nbRow INTEGER;
    Nom VARCHAR(50);
    currentActor tablePersonne%ROWTYPE;
BEGIN
    nbRow := 100;
    Nom := 'Arthur';
    currentActor.idPersonne := 12;
    currentActor.nom := 'Dupond';
    currentActor.prenom := 'Luc';
    currentActor.age := 28;
END
$$;
```

Instruction d'*Affectation d'UN Tuple*

- **Syntaxe Requête SELECT ... INTO ...**
  - SELECT col1, col4
    - INTO Var1, Var2
      - FROM tableName [condition];
  - SELECT col1, col4
    - FROM tableName [condition];
      - INTO Var1, Var2

- **Exemple**
  - DO $$
  - DECLARE
    - nomActor tablePersonne.nom%TYPE;
    - prenomActor tablePersonne.prenom%TYPE;
    - currentActor tablePersonne%ROWTYPE;
  - BEGIN
    - SELECT nom, prenom INTO nomActor, prenomActor
      - FROM tablePersonne WHERE idPersonne = 20;
    - SELECT idPersonne, nom, prenom, age INTO currentActor
      - FROM tablePersonne WHERE idPersonne = 30;
  - END $$;

Structures de *Contrôle*

- **Instructions conditionnelles**
  - **IF condition**
    - **THEN instructions;**
    - **END IF;**
  - **IF condition**
    - **THEN instructions;**
    - **ELSE instructions;**
    - **END IF;**
  - **IF condition**
    - **THEN instructions;**
      - **ELSIF condition**
        - **THEN instructions;**
        - **ELSE instructions;**
    - **END IF;**

Structures de *Contrôle*

- **Instructions conditionnelles**
  - **CASE condition**
    - **WHEN expression_1 [, expression_2, ...] THEN**
      - **instructions;**
    - **[WHEN ...]**
    - **[ELSE**
      - **instructions]**
    - **END CASE;**

Structures de *Contrôle*

- **Instructions répétitives**
  - **LOOP**
    - **instructions;**
    - **END LOOP;**
  - **LOOP**
    - **instructions;**
    - **EXIT WHEN condition;**
    - **...**
    - **END LOOP;**
  - **LOOP**
    - **instructions;**
    - **IF condition THEN EXIT;**
      - **END IF;**
    - **...**
    - **END LOOP;**

Structures de *Contrôle*

- **Instructions répétitives**
  - FOR iIndice IN [REVERSE] iMin … iMax
    - LOOP
      - instructions;
    - END LOOP;
  - FOR Tuple IN (requête)
    - LOOP
      - instructions;
    - END LOOP;
  - WHILE condition
    - LOOP
      - instructions;
    - END LOOP;

Section *Exception*

- **Exceptions machine**
  - Cf. https://www.postgresql.org/docs/11/errcodes-appendix.html

Code d'erreur	Nom de l'erreur
01000	warning
02000	no_data
23000	integrity_constraint_violation
23503	foreign_key_violation
28P01	invalid_password
40000	transaction_rollback

- **Exceptions utilisateur**

Exception *Système*

- **Syntaxe**
  - BEGIN
    - Instructions
    - EXCEPTION
      - WHEN condition1 [OR condition2 ...]
        - THEN instructions gestion erreurs
      - [WHEN condition3 [OR condition4 ...]
        - THEN instructions_gestion_erreurs ...]
  - END;

- **Exemple**
  - BEGIN
    - y := 12/ 0;
    - EXCEPTION
      - WHEN division_by_zero
        - THEN RAISE INFO 'Division par zéro';
  - END;

Exception *Utilisateur*

- **Syntaxe**
  - BEGIN
    - Instructions;
    - IF condition THEN
      - RAISE EXCEPTION('Message');
      - END IF;
    - Instructions;
  - END;

- **Exemple**
  - BEGIN
    - IF count(idPersonne)=0 THEN
      - RAISE EXCEPTION('La table % est vide', 'Personne');
      - END IF;
  - END;

Commande *Procedure*

- **Création Procédure**
  - CREATE [OR REPLACE] PROCEDURE NOM_PROCEDURE
    - ({IN|OUT|INOUT} PARAM DATATYPE, …)
    - [LANGUAGE {plpgsql|sql|…}]
    - AS

> {$Nom Bloc$ | $$}	
> DECLARE + Section déclaration	-- Section facultative
> BEGIN + Section Corps-du-bloc	-- Section obligatoire
> EXCEPTION + Section traitement des erreurs	-- Section facultative
> END	
> {$Nom Bloc$ | $$};	

Commande *Procedure*

- **Suppression Procédure**
  - DROP PROCEDURE NOM_PROCEDURE;
- **Appel Procédure**
  - BEGIN
    - …
    - CALL NOM_PROCEDURE(…);
    - …
  - END;

Commande *Function*

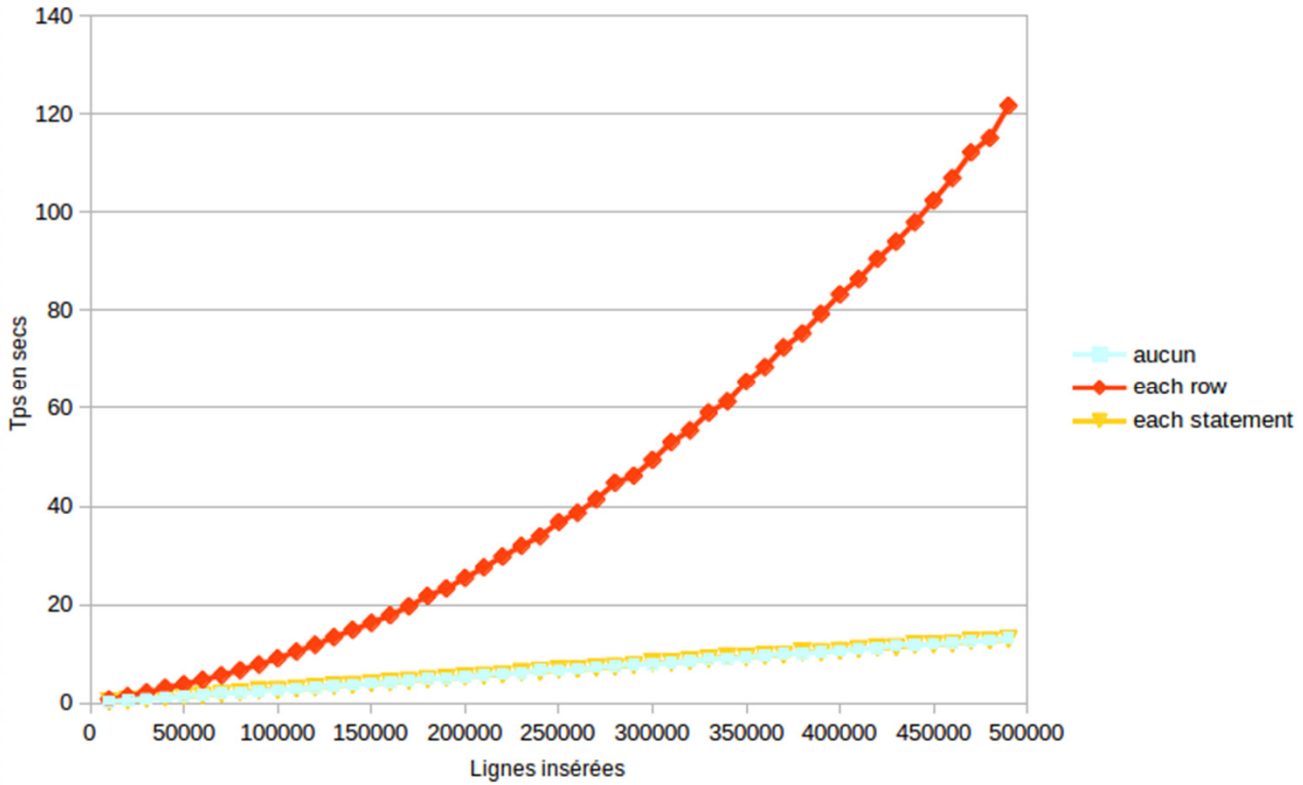
- **Création Fonction**
  - CREATE [OR REPLACE] FUNCTION NOM_FONCTION
    - ({IN|OUT|INOUT} NOM_PARAM DATATYPE, …) RETURNS DATATYPE1
      - [LANGUAGE {plpgsql|sql|…}]
      - AS $$ -- AS Remplace DO
      - DECLARE
        - …
        - RETVAL DATATYPE1;
      - BEGIN
        - …
        - RETVAL := VAL;
        - …
        - RETURN RETVAL;
      - END
      - $$;

Commande *Function*

- **Suppression Fonction**
  - DROP FUNCTION NOM_FONCTION;
- **Appel Fonction**
  - Dans l'éditeur de requête
    - SELECT NOM_FONCTION(…);
  - Dans un bloc
    - BEGIN
      - …
      - SELECT colName INTO Val FROM NOM_FONCTION(…);
      - …
    - END;

Typologie des *Triggers*

- **Triggers d'état**
  - Option STATEMENT
    - Depuis la version 10
  - Pourquoi les avoir introduits ?
    - Temps d'exécution linéaire et légèrement supérieur à l'insertion sans trigger



- **Triggers de ligne**
  - Option ROW

Commande *Trigger*

- **Syntaxe**
  - CREATE [CONSTRAINT] TRIGGER NOM_TRIGGER
    - + {BEFORE|AFTER|INSTEAD OF} {event1 [OR event2 ...]}
    - + ON NOM_TABLE_VIEW
    - + [NOT DEFERRABLE | [DEFERRABLE] [INITIALLY IMMEDIATE|INITIALLY DEFERRED]]
      - – Cf. Transparent Clause DEFERRABLE
    - + [REFERENCING {{OLD|NEW} TABLE [AS] Speudo_Table_name } [...]]
    - + [FOR [EACH] {ROW|STATEMENT}]
    - + [WHEN (condition)]
    - + EXECUTE {FUNCTION|PROCEDURE} function_name (arguments);
  - Option event
    - + INSERT
    - + UPDATE [OF column_name [, ...]]
    - + DELETE
    - + TRUNCATE

Commande *Function Trigger*

- **Création Fonction Trigger**
  - CREATE [OR REPLACE] FUNCTION NOM_FONCTION
    - ({IN|OUT|INOUT} NOM_PARAM DATATYPE, …) RETURN TRIGGER
      - [LANGUAGE {plpgsql|sql|…}]
      - AS $$
      - BEGIN
        - …
        - RETURN NULL|NEW|OLD;
      - END $$;
  - La fonction doit retourner NULL ou un tuple ayant exactement la même structure à celle de la table de déclenchement, NEW ou OLD par exemple

Variables des *Function Trigger de ligne* ou *d'état*

- **Variables de la Fonction Trigger**
  - OLD (record), NEW (record)
    - Variables de l'ancien (UPDATE, DELETE) ou du nouveau (INSERT, UPDATE) tuple
  - TG_WHEN (text)
    - Mode de déclenchement du trigger
      - BEFORE, AFTER, or INSTEAD OF
  - TG_OP (text)
    - Type événement de déclenchement du trigger
      - INSERT, UPDATE, DELETE, or TRUNCATE
  - TG_ARGV[] (text), TG_NARGS (integer)
    - Liste et number des arguments de la fonction d'appel (0 premier indice)
  - TG_TABLE_SCHEMA (name), [TG_TABLE_NAME (name) | ~~TG_RELNAME~~(name)], TG_RELID (oid)
    - Nom du schéma + nom et OID de la table de déclenchement du trigger
  - TG_NAME (name), TG_LEVEL (text)
    - Nom et type du trigger => ROW or STATEMENT

Variables des *Function Trigger de ligne* ou *d'état*

- **Valeurs des variables OLD et NEW en fonction du type d'événement**

Type événement	OLD	NEW
INSERT	NULL	Valeur créée
DELETE	Valeur avant suppression	NULL
UPDATE	Valeur avant modification	Valeur après modification

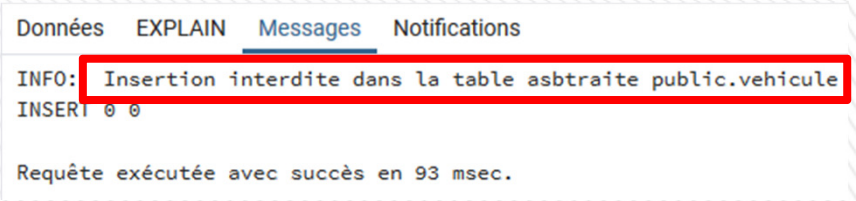
Exemple *Trigger*

- **Création Fonction Trigger**
  - CREATE TABLE vehicule(
    - constructeur varchar,
    - longueur_m real
    -);

- **Création Trigger**
  - CREATE TRIGGER abstractTable
    - BEFORE INSERT
    - ON vehicule
    - FOR EACH ROW
    - EXECUTE PROCEDURE abstractTable();

Exemple *Trigger*

- **Création Fonction Trigger**
  - CREATE OR REPLACE FUNCTION abstractTable() RETURNS TRIGGER
    - LANGUAGE plpgsql
    - AS $BODY$
    - BEGIN
      - RAISE INFO 'Insertion interdite dans la table asbtraite %.%', TG_TABLE_SCHEMA, TG_TABLE_NAME;
      - RETURN NULL;
    - END $BODY$;
  - INSERT INTO vehicule VALUES('DACIA', 2.5);



Exemple *Trigger*

- **Création Tables**
  - CREATE TABLE employe(
    - idemploye SERIAL PRIMARY KEY,
    - prenom VARCHAR(40) NOT NULL,
    - nom VARCHAR(40) NOT NULL);

  - CREATE TABLE log_modification_nom_employe(
    - idemploye INT NOT NULL,
    - prenom VARCHAR(40) NOT NULL,
    - nom VARCHAR(40) NOT NULL,
    - date_modif TIMESTAMP(6) NOT NULL
    - CONSTRAINT Tracabilite_nom_employe_FK
      - FOREIGN KEY (idemploye) REFERENCES employe(idemploye));

Exemple *Trigger*

- **Création Fonction Trigger**
  - CREATE OR REPLACE FUNCTION modification_nom_employe_fun() RETURNS trigger
    - LANGUAGE plpgsql
    - AS $BODY$
    - BEGIN
      - IF NEW.nom <> OLD.nom THEN
        - INSERT INTO log_modification_nom_employe(idemploye, prenom, nom, date_modif)
          - VALUES(OLD.idemploye, OLD.prenom, OLD.nom, now());
        - END IF;
      - RETURN NEW;
    - END $BODY$;

- **Création Trigger**
  - CREATE TRIGGER modification_nom_employe_tg
    - BEFORE UPDATE
    - ON employe
    - FOR EACH ROW
    - EXECUTE PROCEDURE modification_nom_employe_fun();

Exemple *Trigger*

- **Création Trigger**
  - CREATE TRIGGER modification_nom_employe_tg
    - BEFORE UPDATE
    - ON employe
    - FOR EACH ROW
    - EXECUTE PROCEDURE modification_nom_employe_fun();

- **Insertion données**
  - INSERT INTO employe(prenom, nom) VALUES('John', 'Doe');
  - INSERT INTO employe(prenom, nom) VALUES('Lily', 'Bush');

- **Update données**
  - UPDATE employe SET nom = 'Brown' WHERE idemploye = 2;

Données | EXPLAIN | Messages | Notifications

	idemploye [PK] integer	prenom character varying (40)	nom character varying (40)
1	1	John	Doe
2	2	Lily	Brown

Données | EXPLAIN | Messages | Notifications

	idemploye integer	prenom character varying (40)	nom character varying (40)	date_modif timestamp without time zone
1	2	Lily	Bush	2019-12-28 23:46:59.934262

Clause *DEFERRABLE*

- Concerne les contraintes UNIQUE, PRIMARY KEY, EXCLUDE et REFERENCES (clé étrangère) et les Triggers
  - Contraintes ~~NOT NULL~~ et ~~CHECK~~ non concernées pour l'instant
- Option NOT DEFERRABLE
  - Valeur par défaut
  - Contrainte vérifiée immédiatement
- Option DEFERRABLE
  - Permet de différer la vérification de la contrainte
    - Sous option INITIALLY IMMEDIATE
      - Contrainte vérifiée après chaque instruction
    - Sous option INITIALLY DEFERRED
      - Contrainte vérifiée à la fin de la transaction

Commande *Set Constraint*

- **Syntaxe SET CONSTRAINTS**
  - SET CONSTRAINTS {ALL|name [, ...]} {DEFERRED|IMMEDIATE}
  - Permet de modifier le moment de vérification des contraintes DEFERRABLE
    - Pour rendre les contraintes faire
      - … ALTER CONSTRAINT …
  - Concerne les contraintes
    - UNIQUE
    - PRIMARY KEY
    - EXCLUDE
    - REFERENCES (clé étrangère)