Notion de TRIGGER

Triggers pour l'Héritage

André Miralles

Typologie des *Triggers*

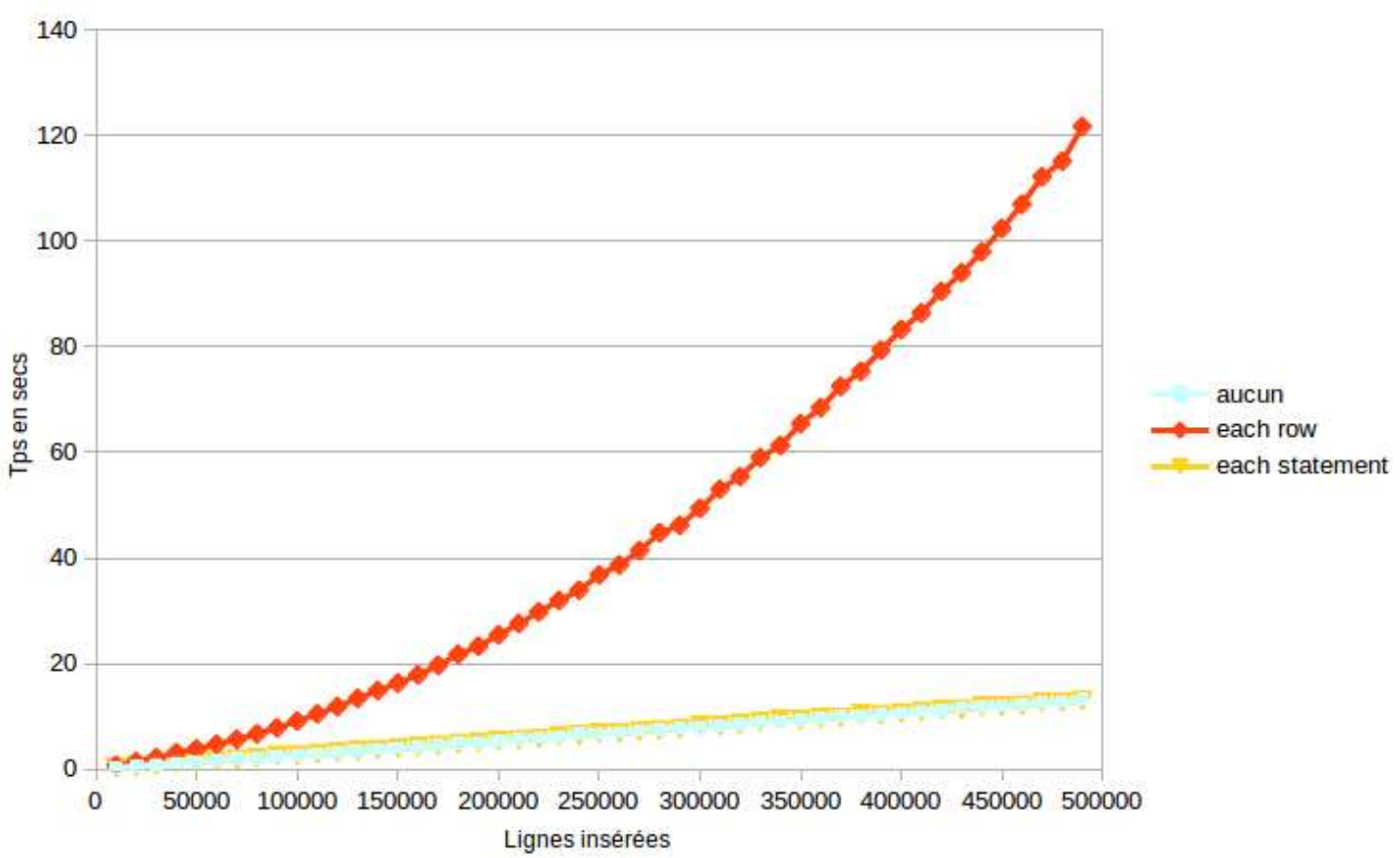
» **Triggers d'état**

> **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**

Instruction *Trigger*

» Syntaxe simplifiée

- > CREATE [CONSTRAINT] TRIGGER NOM_TRIGGER
  - + {BEFORE|AFTER|INSTEAD OF} {event1 [OR event2 ...]}
  - + ON NOM_TABLE
  - + [FOR [EACH] {ROW|STATEMENT}]
  - + EXECUTE {FUNCTION|PROCEDURE} function_name ();

- > Option event
  - + INSERT
  - + UPDATE [OF column_name [, ...]]
  - + DELETE
  - + TRUNCATE

Instruction *Function Trigger*

- **Syntaxe Fonction Trigger**
  - CREATE [OR REPLACE] FUNCTION NOM_FONCTION() RETURNS 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

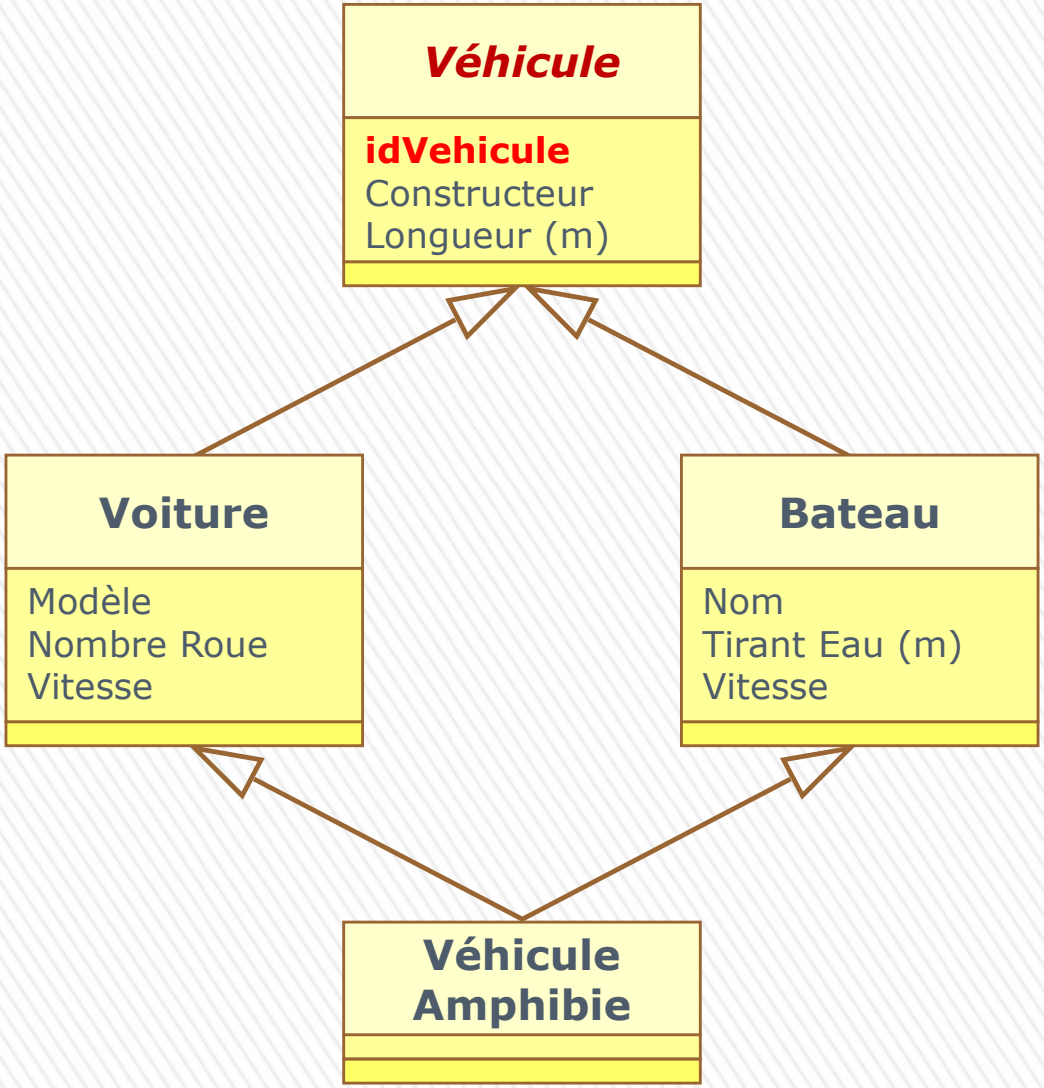
Variables des *Function Trigger de ligne* ou *d'état*

» **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 du *Véhicule Amphibie*

- Trigger pour **Table Abstraite**
- Trigger unicité de la **Clé Primaire**



Exemple de *Trigger sur Table Abstraite*

» **Etape 1 ==> Création Fonction Trigger**

```
> CREATE OR REPLACE FUNCTION checkAbstractTable() RETURNS TRIGGER
    + LANGUAGE plpgsql
    + AS $$
    + BEGIN
        – RAISE EXCEPTION 'Insertion interdite dans la table abstraite %', TG_TABLE_NAME;
        – RETURN NULL;
    + END $$;
```

» **Etape 2 ==> Création Table**

```
> CREATE TABLE Vehicule (
    + idVoiture integer,
    + Constructeur varchar,
    + Longueur real);
```

» **Etape 3 ==> Création Trigger**

```
> CREATE TRIGGER chekAstractTableVehicule
    + BEFORE INSERT
    + ON Vehicule
    + FOR EACH ROW
    + EXECUTE PROCEDURE checkAbstractTable();
```

Exemple de *Trigger d'unicité de Clé Primaire*

» **Etape 1 ==> Création Fonction Trigger**

```
> CREATE OR REPLACE FUNCTION checkPrimaryKey() RETURNS TRIGGER
    + LANGUAGE plpgsql
    + AS $$
    + DECLARE
        – nbIdVehicule integer;
    + BEGIN
        – SELECT COUNT(idVehicule) FROM Vehicule WHERE idVehicule = NEW. idVehicule INTO
          nbIdVehicule;
        – IF nbIdVehicule > 0 THEN
            » RAISE EXCEPTION 'L''identifiant (%) existe déjà dans la table %.', NEW. idVehicule,
              TG_TABLE_NAME ;
        – END IF;
        – RETURN NEW;
    + END $$;
```

» **Etape 2 ==> Création Table**

```
> CREATE TABLE Voiture(
    + idVoiture integer,
    + Constructeur varchar,
    + Longueur real,
    + Modele varchar,
    + NombreRoue integer,
    + Vitesse real) INHERITS(Vehicule);
```

» **Etape 3 ==> Création Trigger**

```
> CREATE TRIGGER checkPrimaryTableVoiture
    + BEFORE INSERT
    + ON Voiture
    + FOR EACH ROW
    + EXECUTE PROCEDURE checkPrimaryKey();
```

Exemples de *Tests d'insertion*

» Etape 4 ==>Test sur **Table Abstraite**

> **INSERT INTO Vehicule**

+ **SELECT max(idVehicule)+1, 'Citroen', 2.86 FROM Vehicule;**

Data Output | Messages | Notifications

```
ERROR:  Insertion interdite dans la table abstraite Vehicule
CONTEXT:  fonction PL/pgSQL "abstractTable"(), ligne 3 à RAISE

ERREUR:  Insertion interdite dans la table abstraite Vehicule
État SQL : P0001
```

» Etape 4 ==>Test sur **Clé Primaire**

> **INSERT INTO Voiture**

+ **SELECT idVehicule, 'Citroen', 2.86, 'DS', 4, 180 FROM Vehicule LIMIT 1;**

Data Output | Messages | Notifications

```
ERROR:  L'identifiant (1) existe déjà dans la table Voiture.
CONTEXT:  fonction PL/pgSQL "checkPrimaryKey"(), ligne 7 à RAISE

ERREUR:  L'identifiant (1) existe déjà dans la table Voiture.
État SQL : P0001
```