

Bases de données *SQL*

LDD

Invariants et automatismes

BD111
v122b

2021-04-01

Département d'informatique
Faculté des sciences



Christina.Khnaisser@USherbrooke.ca
<http://info.USherbrooke.ca/ckhnaisser>
Luc.Lavoie@USherbrooke.ca
<http://info.USherbrooke.ca/llavoie>

PLAN

- Les invariants (assertions)
- Les déclencheurs (triggers)
- Les automatismes (trigger functions)
- Perspectives
- Exemples
- Références



INVARIANTS

UN EXEMPLE - LE PROBLÈME

Soit la représentation suivante d'une cartothèque :

```
CREATE TABLE Inventaire (  
  idArticle CHAR (8),  
  idClasseur CHAR (4),  
  ....  
  PRIMARY KEY (idArticle)  
);
```

On désire limiter le nombre d'articles par classeur à 40.

INVARIANTS

UN EXEMPLE – UNE SOLUTION

```
CREATE ASSERTION Inventaire_Max AS  
  CHECK (NOT EXISTS (  
    SELECT idClasseur  
    FROM Inventaire  
    GROUP BY idClasseur  
    HAVING COUNT(*) > 40  
  ));
```

Les invariants (ASSERTION) sont des contraintes pouvant référer à plus d'une table et pouvant utiliser toute la « puissance » des expressions relationnelles.

INVARIANTS

SYNTAXE SQL ISO : CREATE ASSERTION

```
<assertion definition> ::=  
    CREATE ASSERTION <constraint name>  
    CHECK ( <search condition> )  
    [ <constraint characteristics> ]
```

```
<constraint characteristics> ::=  
    <constraint check time> [ [ NOT ] DEFERRABLE ]  
    | [ NOT ] DEFERRABLE [ <constraint check time> ]
```

```
<constraint check time> ::=  
    INITIALLY { DEFERRED | IMMEDIATE }
```

INVARIANTS

SYNTAXE SQL ISO : CREATE ASSERTION

<drop assertion statement> ::=
 DROP ASSERTION <constraint name>
 [<drop behavior>]
<drop behavior> ::=
 ...

LE LANGAGE SQL

LES INVARIANTS EN PRATIQUE

- La majorité des éditeurs de SGBD n'ont toutefois pas mis en oeuvre les invariants de BD, imposant aux concepteurs de les « programmer » à l'aide de déclencheurs (TRIGGER) et d'automatismes (TRIGGER FUNCTION).

AUTOMATISMES

RAPPEL : FONCTIONS, PROCÉDURES ET AUTOMATISMES

- Qu'ont en commun une fonction, une procédure et un automatisme ?
 - Ce sont des abstractions paramétrables d'un traitement.
- Qui fait quoi ?
 - Une *fonction* **calcule une valeur** suite à un appel (sur la base de ses paramètres **ou** de l'état de la BD).
 - Une *procédure* **change l'état de la BD** suite à un appel (sur la base de ses paramètres **et** de l'état de la BD).
 - Un *automatisme* change l'état de la BD **suite à un évènement** (sur la base d'un évènement **et** de l'état de la BD).

AUTOMATISMES

SYNTAXE SQL ISO (1/2)

<trigger definition> ::=
CREATE TRIGGER <trigger name>
<trigger action time>
<trigger event> ON <table name>
[REFERENCING <transition list>]
<triggered action>

<trigger action time> ::=
BEFORE | AFTER | INSTEAD OF
<trigger event> ::=
INSERT | DELETE | UPDATE [OF <column name list>]

<triggered action> ::=
[FOR EACH { ROW | STATEMENT }]
[WHEN (<search condition>)]
<triggered SQL statement>

AUTOMATISMES

SYNTAXE SQL ISO (2/2)

<triggered SQL statement> ::=
 <SQL procedure statement> | BEGIN [ATOMIC] <inst> END
<inst> ::=
 { <SQL procedure statement> ; ...}

...

<transition list> ::=
 { <transition table or variable> , ...}
<transition table or variable> ::=
 OLD [ROW] [AS] <old var name>
 | NEW [ROW] [AS] <new var name>
 | OLD TABLE [AS] <old table name>
 | NEW TABLE [AS] <new table name>

AUTOMATISMES

SYNTAXE POSTGRESQL (1/2)

<trigger definition> ::=
CREATE [CONSTRAINT] TRIGGER <trigger name>
<trigger action time>
{ <trigger event> OR' ... } ON <table name>
[FROM <referenced table name>]
[NOT DEFERRABLE | [DEFERRABLE] [INITIALLY {IMMEDIATE|DEFERRED}]]
[REFERENCING <transition list>]
<triggered action>

<trigger action time> ::=
BEFORE | AFTER | INSTEAD OF
<trigger event> ::=
INSERT | DELETE | UPDATE [OF <column name list>]
| TRUNCATE

AUTOMATISMES

SYNTAXE POSTGRESQL (2/2)

<transition list> ::=

{ { OLD | NEW } TABLE [AS] transition_relation_name } [...]

<triggered action> ::=

[FOR EACH { ROW | STATEMENT }]

[WHEN (<search condition>)]

<triggered SQL statement>

<triggered SQL statement> ::=

EXECUTE PROCEDURE function_name (arguments)

AUTOMATISMES (TRIGGER FUNCTION) SYNTAXE – SIMPLIFIÉE (POSTGRESQL)

trigger_function_def ::=

```
CREATE [ OR REPLACE ] FUNCTION name ( [ argument [, ...] ] )  
RETURNS TRIGGER  
LANGUAGE lang_name  
AS body { other_fun_qual }
```

argument ::=

```
[ arg_mode ] [ arg_name ] arg_type [ DEFAULT default_expr ]
```

arg_mode ::=

```
IN | VARIADIC
```

AUTOMATISMES (TRIGGER FUNCTION) SYNTAXE (POSTGRESQL)

- Accès à la valeur de l'objet de référence (ROW ou TABLE)
 - soit avant l'évènement (OLD)
 - soit après l'évènement (NEW)
- Selon la syntaxe suivante
 - <reference> ::=
 - OLD [ROW]
 - | NEW [ROW]
 - | OLD TABLE
 - | NEW TABLE

UN EXEMPLE

LA SOLUTION PAR LE BIAIS D'UN AUTOMATISME DE TABLE

Il faut définir un « TRIGGER » et

```
CREATE TRIGGER Inventaire_Max  
AFTER INSERT OR UPDATE  
ON Inventaire  
EXECUTE PROCEDURE  
    Inventaire_lim ();
```

une « TRIGGER FUNCTION »

```
CREATE FUNCTION Inventaire_lim ()  
    RETURNS TRIGGER  
    LANGUAGE plpgsql AS  
$$  
BEGIN  
IF NOT EXISTS (  
    SELECT idClasseur  
    FROM Inventaire  
    GROUP BY idClasseur  
    HAVING COUNT(*) > 40 ) THEN  
    RETURN NEW; -- on peut procéder  
ELSE  
    RAISE EXCEPTION 'Trop de cartes';  
    RETURN NULL; -- rien à insérer  
END IF;  
END  
$$
```

UN EXEMPLE

LA SOLUTION PAR LE BIAIS D'UN AUTOMATISME DE TABLE ... OPTIMISÉE

Il faut définir un « TRIGGER » et

```
CREATE TRIGGER Inventaire_Max  
AFTER INSERT  
ON Inventaire  
EXECUTE PROCEDURE  
    Inventaire_lim ();
```

```
CREATE TRIGGER Inventaire_Max  
AFTER UPDATE OF idClasseur  
ON Inventaire  
EXECUTE PROCEDURE  
    Inventaire_lim ();
```

une « TRIGGER FUNCTION »

```
CREATE FUNCTION Inventaire_lim ()  
    RETURNS TRIGGER  
    LANGUAGE plpgsql AS  
$$  
BEGIN  
IF NOT EXISTS (  
    SELECT idClasseur  
    FROM Inventaire  
    GROUP BY idClasseur  
    HAVING COUNT(*) > 40 ) THEN  
    RETURN NEW; -- on peut procéder  
ELSE  
    RAISE EXCEPTION 'Trop de cartes';  
    RETURN NULL; -- rien à insérer  
END IF;  
END  
$$
```


UN EXEMPLE

LA SOLUTION PAR LE BIAIS D'UN AUTOMATISME DE ROW

Il faut définir un « TRIGGER » et

```
CREATE TRIGGER Inventaire_Max
BEFORE
  INSERT OR UPDATE OF idClasseur
ON Inventaire
FOR EACH ROW
EXECUTE PROCEDURE
  Inventaire_lim ();
```

une « TRIGGER FUNCTION »

```
CREATE FUNCTION Inventaire_lim ()
  RETURNS TRIGGER
  LANGUAGE plpgsql AS
$$
BEGIN
  IF (SELECT COUNT(idArticle)
      FROM Inventaire
      WHERE (idClasseur=NEW.idClasseur)) < 40 THEN
    RETURN NEW; -- on peut procéder
  ELSE
    RAISE EXCEPTION 'Trop de cartes';
    RETURN NULL; -- rien à insérer
  END IF;
END
$;
```

AUTOMATISMES

QUELS ÉVÈNEMENTS AVEC QUOI ?

Le tableau suivant récapitule les types d'automatismes pouvant être utilisés sur les tables et les vues:

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables and <i>foreign tables</i>	Tables, views , and <i>foreign tables</i>
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables and <i>foreign tables</i>	Tables, views , and <i>foreign tables</i>
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

Source : <https://www.postgresql.org/docs/14/sql-createtrigger.html>

EXEMPLE DE DONNÉES

Étudiant

<u>matricule</u>	<u>nom</u>	<u>adresse</u>
15113150	Paul	>Δ ^ϕ σ ^ϕ b
15112354	Éliane	Blanc-Sablon
15113870	Mohamed	Tadoussac
15110132	Sergeï	Chandler

Activité

<u>sigle</u>	<u>titre</u>
IFT159	Analyse et programmation
IFT187	Éléments de bases de données
IMN117	Acquisition des médias numériques
IGE401	Gestion de projets
GMQ103	Géopositionnement

TypeÉvaluation

<u>code</u>	<u>description</u>
IN	Examen intra
FI	Examen final
TP	Travail pratique
PR	Projet

Résultat

<u>matricule</u>	<u>TE</u>	<u>activité</u>	<u>trimestre</u>	<u>note</u>
15113150	TP	IFT187	20133	80
15112354	FI	IFT187	20123	78
15113150	TP	IFT159	20133	75
15112354	FI	GMQ103	20123	85
15110132	IN	IMN117	20123	90
15110132	IN	IFT187	20133	45
15112354	FI	IFT159	20123	52

ÉVALUATION EXEMPLE 1 (v1)

Un étudiant ne peut pas
s'inscrire à plus de 5
cours au même trimestre

```
CREATE FUNCTION Cours_lim
RETURNS TRIGGER
LANGUAGE plpgsql AS
$$
BEGIN
    IF EXISTS (
        SELECT * FROM Resultat
        GROUP BY matricule, trimestre
        HAVING COUNT(DISTINCT activite) > 5 )
    THEN
        RAISE EXCEPTION 'Trop de cours';
        RETURN NULL; -- rien à insérer
    ELSE
        RETURN NEW; -- on peut procéder
    END IF;
END
$$

CREATE TRIGGER Cours_dec_lim
BEFORE INSERT OR UPDATE ON Resultat
FOR EACH STATEMENT
EXECUTE PROCEDURE Cours_lim();
```

ÉVALUATION EXEMPLE 1 (v2)

Un étudiant ne peut pas
s'inscrire à plus de 5
cours au même trimestre

```
CREATE TRIGGER Cours_dec_lim
  BEFORE
  INSERT
  OR
  UPDATE OF matricule, trimestre, activite
  ON Resultat
  FOR EACH STATEMENT
  EXECUTE PROCEDURE Cours_lim();
```

ÉVALUATION – EXEMPLE 2 (v1)

```
CREATE VIEW Inscription AS
  SELECT DISTINCT matricule, nom, adresse, activite, trimestre
  FROM Resultat JOIN Etudiant USING (matricule) ;

CREATE OR REPLACE FUNCTION Inscription_ins ()
  RETURNS TRIGGER
  LANGUAGE plpgsql AS
  $$
  BEGIN
    INSERT INTO Etudiant (matricule, nom, adresse)
      VALUES (NEW.matricule, NEW.nom, NEW.adresse);
    INSERT INTO Resultat (matricule, te, activite, trimestre, note)
      VALUES (NEW.matricule, 'TP', NEW.activite, NEW.trimestre, 0);
    RETURN NEW;
  END;
  $$;

CREATE TRIGGER Inscription_ins_dec
  INSTEAD OF INSERT ON Incription
  FOR EACH ROW EXECUTE PROCEDURE Inscription_ins () ;

INSERT INTO Incription (matricule, nom, adresse, activite, trimestre)
  VALUES ('18773250', 'Adam', 'Laval', 'IGE401', '20133') ;
```

ÉVALUATION – EXEMPLE 2 (v2)

```
CREATE OR REPLACE FUNCTION Inscription_ins()
  RETURNS TRIGGER
  LANGUAGE plpgsql AS
  $$
  BEGIN
    IF EXISTS(SELECT * FROM Etudiant WHERE matricule= NEW.matricule) THEN
      IF EXISTS (
        SELECT * FROM Etudiant
          WHERE matricule = NEW.matricule AND
            (nom <> NEW.nom OR adresse <> NEW.adresse)
        ) THEN
        RAISE EXCEPTION 'Incohérence de nom ou d''adresse' ;
        RETURN NULL ;
      ELSE
        -- ne rien faire
      END IF ;
    ELSE
      INSERT INTO Etudiant (matricule, nom, adresse)
        VALUES (NEW.matricule, NEW.nom, NEW.adresse);
    END IF;
    INSERT INTO Resultat (matricule, te, activite, trimestre, note)
      VALUES (NEW.matricule, 'TP', NEW.activite, NEW.trimestre, 0);
    RETURN NEW;
  END;
  $$
```

AUTOMATISMES

AUTRES EXEMPLES POSTGRESQL

○ Exemples

- <http://docs.postgresql.fr/11/sql-createtrigger.html>

AUTOMATISMES PERSPECTIVES

- L'absence du CREATE ASSERTION dans la majorité des dialectes SQL est généralement motivée par le cout induit par leur vérification systématique. Quand on y regarde de plus près, force est de constater que le compilateur est toujours capable de faire aussi bien, sinon mieux, que le programmeur lorsque vient le temps de traduire l'assertion en automatisme. En ce qui concerne le déclencheur, cela est aussi le plus souvent le cas, surtout si le compilateur s'inspire des techniques développées pour les langages fonctionnels.
- Il appartient donc aux utilisateurs de faire des pressions sur les éditeurs pour que ceux-ci passent à l'action!
- Les automatismes et les déclencheurs demeurent toutefois pertinents dans d'autres situations.

RÉFÉRENCES

- Elmasri et Navathe (4^e éd.), chapitre 7
- Elmasri et Navathe (6^e éd.), chapitre 4
- [Loney2008]
Loney, Kevin ;
Oracle Database 11g: The Complete Reference.
Oracle Press/McGraw-Hill/Osborne, 2008.
ISBN 978-0071598750.
- [Date2012]
Date, Chris J. ;
SQL and Relational Theory: How to Write Accurate SQL Code.
2nd edition, O'Reilly, 2012.
ISBN 978-1-449-31640-2.
- Le site d'Oracle (en anglais)
 - http://www.oracle.com/pls/db10g/portal.portal_demo3?selected=5
 - http://docs.oracle.com/cd/B19306_01/server.102/b14200/toc.htm
- Le site de PostgreSQL (en français)
 - <http://docs.postgresqlfr.org>

