



Trigger sur DB2 z/OS



Réunion du Guide DB2A
Jeudi 22 janvier 2009
ASG, Paris-La Défense (92)
France

Laurent VEILLET - FMP
Guide DB2 B - Avril 2007
Guide DB2 A – Janvier 2009

Trigger



Définition

- Opérations «déclenchées» par la mise à jour d'une table
- Ordres SQL déclencheurs :
 - INSERT
 - UPDATE
 - DELETE
- Opérations possibles :
 - ordres SQL
 - procédure stockée
 - fonction UDF
- Exécution entièrement managée par le SGDB

Trigger



Contexte

- Spécifications des triggers comme une **extension objet** des bases relationnelles dans la norme **SQL3** (1999), donc portable sur les autres SGDB selon leur niveau d'implémentation
- Fonctionnalité disponible sur DB2 z/OS depuis la V6
- Tests effectués sur la V7

Trigger



Syntaxe

```
CREATE TRIGGER nom_schema.nom_trigger
NO CASCADE BEFORE | AFTER
INSERT | DELETE | UPDATE [ OF liste_colonne ]
ON nom_table
[REFERENCING OLD [ROW] [AS] nom_anciennes_valeurs_ligne
| NEW [ROW] [AS] nom_nouvelles_valeurs_ligne
| OLD TABLE [AS] nom_anciennes_valeurs_table
| NEW TABLE [AS] nom_nouvelles_valeurs_table]
FOR EACH STATEMENT | ROW MODE DB2SQL
[WHEN condition]
BEGIN ATOMIC
code_trigger
END $
```

Trigger

Exemple orienté objet

➔ approvisionnement si quantité dispo. de produit < 10% du stock max.

```
CREATE TRIGGER sch.tr_approv
```

```
AFTER
```

```
UPDATE OF qt_dispo,stock_max
```

```
ON t_produit
```

```
REFERENCING NEW AS n
```

```
FOR EACH ROW
```

```
MODE DB2SQL
```

```
WHEN (n.qt_dispo < 0.10 * n.stock_max)
```

```
BEGIN ATOMIC
```

```
    CALL ps_approv
```

```
    (n.stock_max - n.qt_dispo , n.produit_no) ;
```

```
END
```

```
-- nom du trigger
```

```
-- activation
```

```
-- déclencheur
```

```
-- table mise à jour
```

```
-- nom de transition variable : n
```

```
-- granularité
```

```
-- mode d'exécution
```

```
-- condition
```

```
-- début corps du trigger
```

```
-- action : appel procédure stockée
```

```
-- fin corps de trigger
```

Trigger



Autres utilisations

- Data conditioning : modifier les données avant la mise à jour
- Data integrity : quand une table fille a plusieurs tables parentes
- Data propagation : audits, historiques, agrégats
- Data validation : plage de valeur complexe (hors check constraint)

Trigger



Restrictions

- Peut être coûteux en CPU si mal optimisé
- Pas rétroactif sur les données déjà présentes dans la table
- Pas actif pour le LOAD, sauf en SHRLEVEL CHANGE
- Interdit sur les tables du catalogue
- Rollback impossible si l'action déclenchée par le trigger est hors du contrôle de DB2 (PS hors RSS commit control, envoi d'un mail)

Trigger

SQL corps de trigger

Table 77. Allowable SQL statements

SQL statement	Trigger activation time	
	BEFORE	AFTER
fullselect	X	X
CALL	X	X
SIGNAL	X	X
VALUES	X	X
SET transition variable	X	
INSERT		X
DELETE (searched)		X
UPDATE (searched)		X
REFRESH TABLE (v8)		X

Trigger

Combinaisons d'options

Table 76. Allowable combinations of attributes in a trigger definition

Granularity	Activation time	Triggering SQL operation	Transition variables allowed	Transition tables allowed
FOR EACH ROW	BEFORE	DELETE	OLD	-
		INSERT	NEW	-
		UPDATE	OLD, NEW	-
	AFTER	DELETE	OLD	OLD_TABLE
		INSERT	NEW	NEW_TABLE
		UPDATE	OLD, NEW	OLD_TABLE, NEW_TABLE
FOR EACH STATEMENT	BEFORE	DELETE	-	-
		INSERT	-	-
		UPDATE	-	-
	AFTER	DELETE	-	OLD_TABLE
		INSERT	-	NEW_TABLE
		UPDATE	-	OLD_TABLE, NEW_TABLE

Trigger

SET transition-variable

Dans un trigger BEFORE, permet de modifier les données mise à jour

Exemple : à chaque UPDATE, forcer *last_upd* à CURRENT TIMESTAMP

```
CREATE TRIGGER trigger_1
NO CASCADE BEFORE UPDATE ON tab1
REFERENCING NEW AS n
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    SET n.last_upd = CURRENT TIMESTAMP;
END#
```

Trigger

SIGNAL SQLSTATE

- Interdire une opération (trigger before)
- Forcer un rollback (trigger after)
- Alimenter la SQLCA :
 - retourne un SQLCODE – 438
 - code utilisateur de 5 caractères dans SQLSTATE
 - message utilisateur dans SQLERRMC

Trigger

SIGNAL SQLSTATE

Exemple : interdire les modifications de *salaire* de plus de 20%

```
CREATE TRIGGER trigger_1
NO CASCADE BEFORE UPDATE OF salaire ON t_employe
REFERENCING OLD AS o
                NEW AS n
FOR EACH ROW MODE DB2SQL
WHEN (n.salaire > (o.salaire * 1.20))
BEGIN ATOMIC
    SIGNAL SQLSTATE '75001' ('Augmentation de salaire > 20%');
END#
```



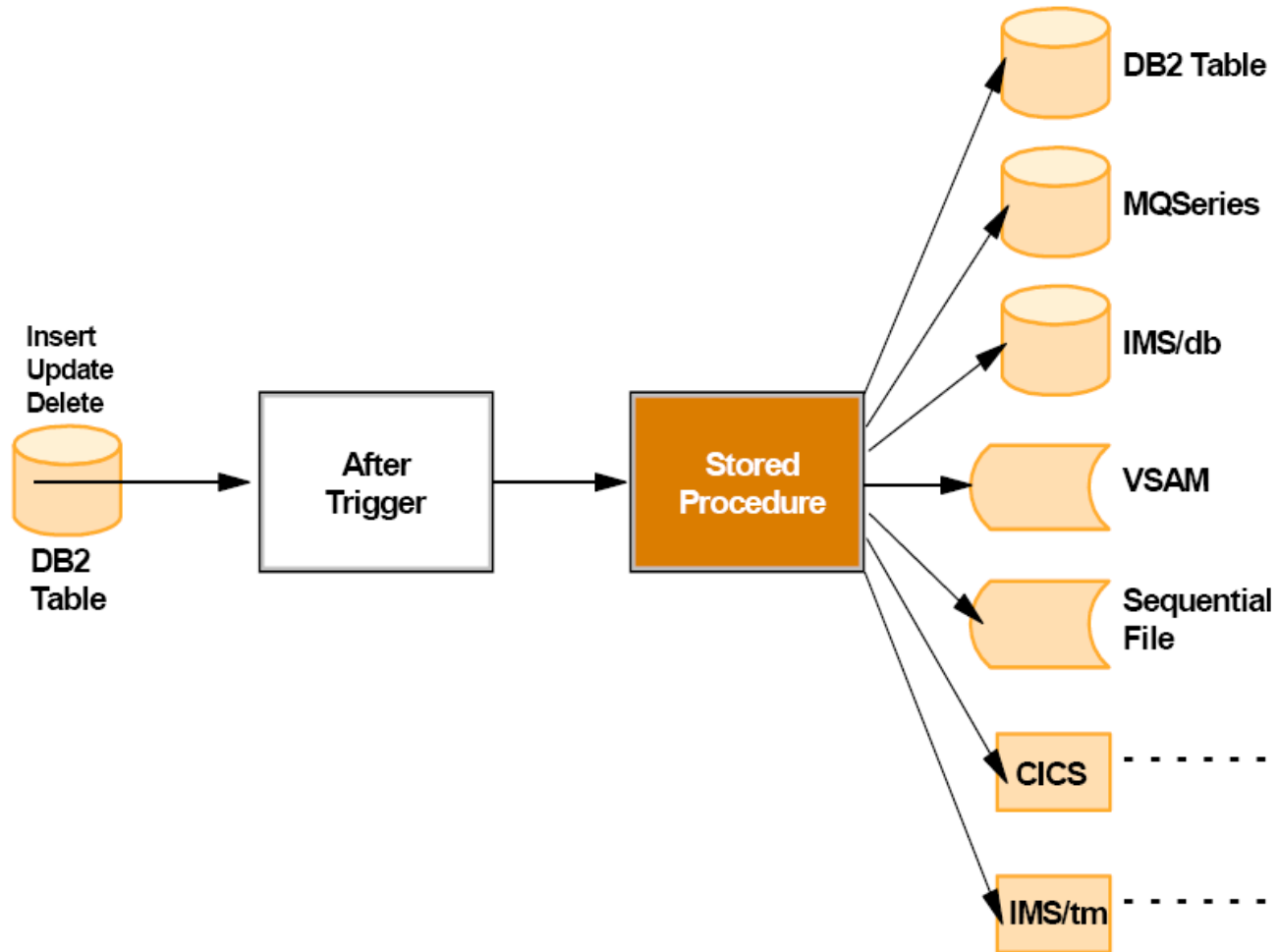
Trigger sur DB2 z/OS



Applications pratiques

Trigger

Data propagation



Trigger

Étendre les fonctionnalités DB2

- Update cascade : update PK ➔ update FK «filles»
- Pendant delete : delete dernière FK ➔ delete PK «mère»
- Gérer l'unicité sur des colonnes longues
- Limiter les mises à jour d'une table par date ou plage horaire

Trigger



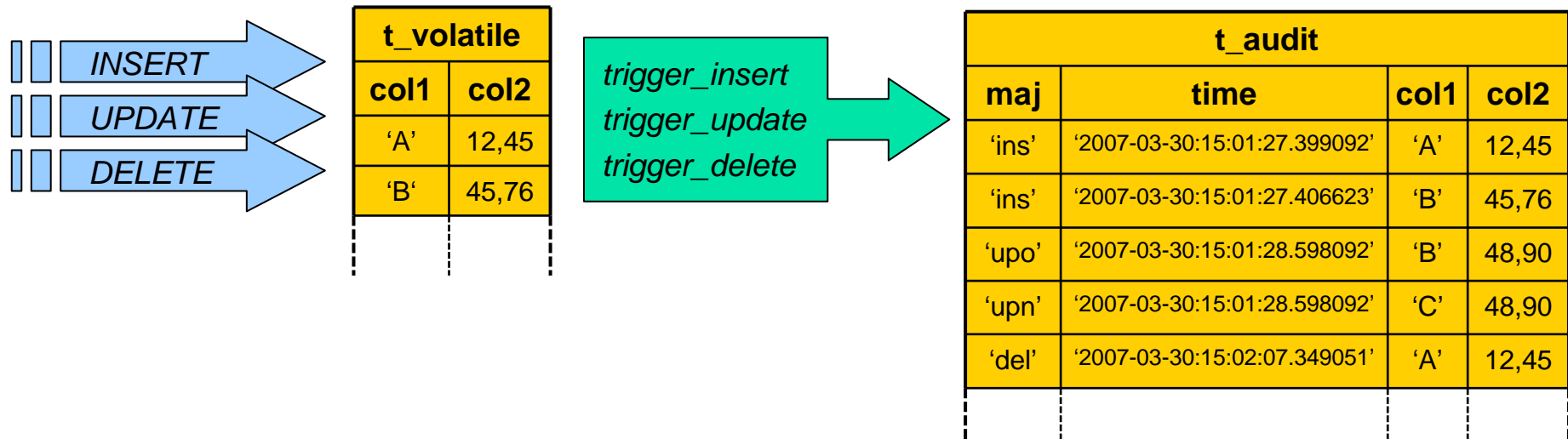
Auditer une table « volatile »

Objectif : prendre les statistiques à pleine charge sur une table dont la volumétrie fluctue en cours de traitement

- 1) Création de la table d'audit
- 2) Création des triggers d'audit
- 3) Analyse des mises à jour dans la table d'audit
- 4) Insertion des lignes dans la table volatile (ou son clone)
- 5) Exécution du RUNSTATS
- 6) Suppression des lignes

Trigger

Alimentation de la table d'audit



Trigger

Création de la table d'audit

```
CREATE VIEW v_audit AS
SELECT
  '      ' AS maj, -- valeurs : 'ins', 'upo', 'upn', 'del'
  CURRENT_TIMESTAMP AS time, -- timestamp de mise à jour
  t_volatile.* -- colonnes de la table volatile
FROM t_volatile;

CREATE TABLE t_audit LIKE v_audit;

DROP VIEW v_audit ;
```

Trigger

Capture des INSERT sur t_volatile

```
CREATE TRIGGER trigger_insert
AFTER INSERT ON t_volatile
REFERENCING NEW TABLE AS n
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
  INSERT INTO t_audit
    SELECT
      'ins',
      CURRENT TIMESTAMP,
      n.*
  FROM n;
END$
```

Trigger

Capture des UPDATE sur t_volatile

```
CREATE TRIGGER trigger_update
AFTER UPDATE ON t_volatile
REFERENCING NEW TABLE AS n
                OLD TABLE AS o
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
INSERT INTO t_audit
    SELECT 'upo', CURRENT TIMESTAMP, o.* FROM o;
INSERT INTO t_audit
    SELECT 'upn', CURRENT TIMESTAMP, n.* FROM n;
END$
```

Trigger

Capture des DELETE sur t_volatile

```
CREATE TRIGGER trigger_delete
AFTER DELETE ON t_volatile
REFERENCING OLD TABLE AS o
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
  INSERT INTO t_audit
    SELECT
      'del',
      CURRENT TIMESTAMP,
      o.*
    FROM o;
END$
```

Trigger

RUNSTATS de la table volatile

```
INSERT INTO t_volatile          -- si t_volatile est vide
SELECT col1, col2
FROM t_audit
WHERE type_maj = 'ins'
      AND time BETWEEN ... ; -- à analyser selon les cas
```

```
RUNSTATS t_volatile
```

```
DELETE * FROM t_volatile ;
```

```
DROP TABLE t_audit ;
```

```
DROP TRIGGERS
```

Trigger



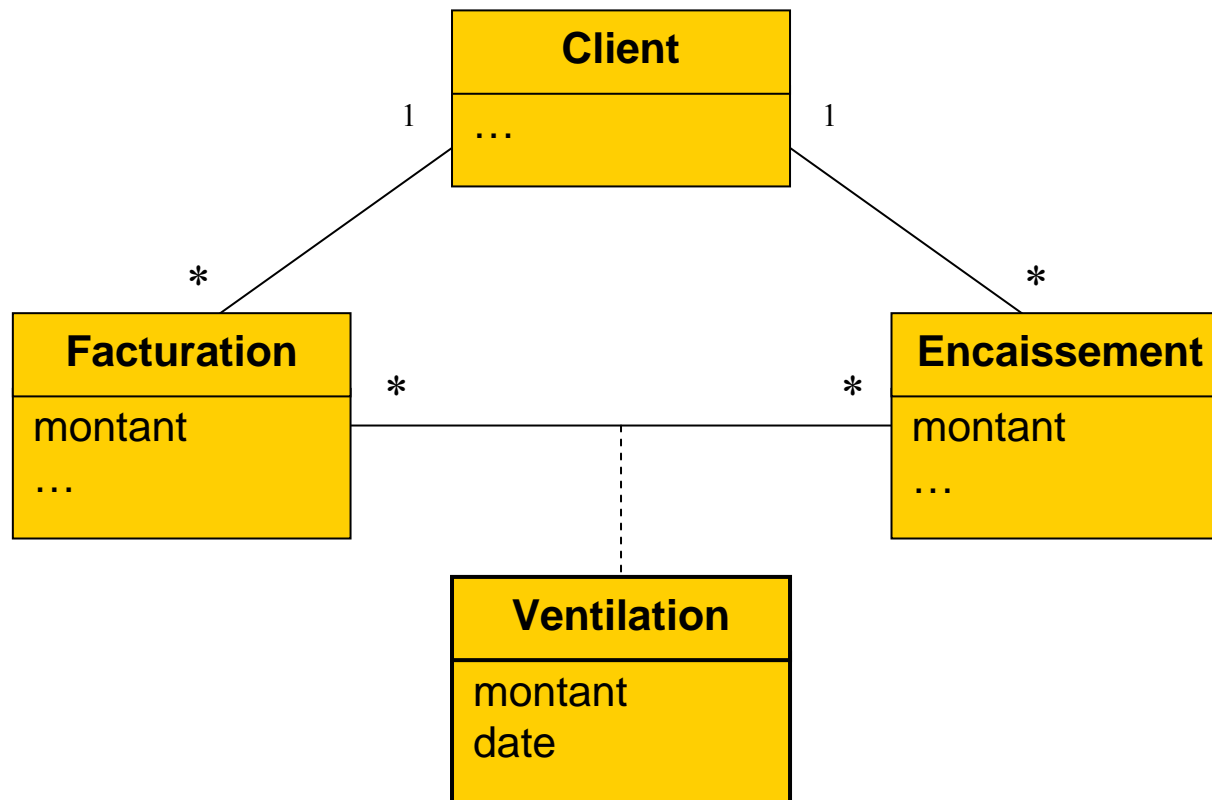
Autres utilisations de la table d'audit

- Faire une « log » sur une table donnée en rajoutant des special register (CURRENT SQLID, PACKAGESET ...) dans le but de faire un audit
- Propagation de données dans des cas spécifiques (tables d'historique pouvant être modifiées, ...), nécessite un mécanisme de synchronisation

Trigger

Cas pratique intégrité et agrégats

Modèle
logique



Trigger

Cas pratique intégrité et agrégats

facturation		
fact_id	cli_id	fact_mt
1	235	500,00
2	235	500,00

encaissement		
enc_id	cli_id	enc_mt
1	235	800,00

Ventilation			
fact_id	enc_id	vent_mt	vent_dt_creat
1	1	500,00	'2007-03-30'
2	1	300,00	'2007-03-30'

Facturation
<u>fact_id</u>
<u>cli_id</u>
fact_mt
...

Ventilation
<u>fact_id</u>
<u>enc_id</u>
vent_mt
vent_dt_creat

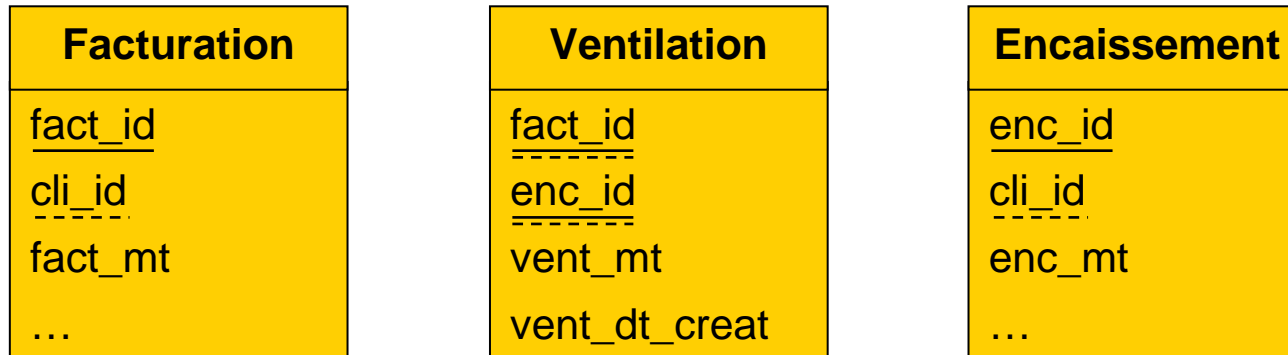
Encaissement
<u>enc_id</u>
<u>cli_id</u>
enc_mt
...

Contraintes sur la table Ventilation

- facture et encaissement concernent le même client
 - ➔ **fact_id** et **enc_id** associés dans Ventilation ont le même **cli_id**

- conserver l'historique des mises à jour sur la table Ventilation
 - ➔ update interdit
 - ➔ sauvegarde des lignes supprimées dans une table historique

- ne pas ventiler plus que le montant d'une facture ou d'un encaissement
 - ➔ pour un **fact_id** donné : $\text{somme}(\text{vent_mt}) \leq \text{fact_mt}$
 - ➔ pour un **enc_id** donné : $\text{somme}(\text{vent_mt}) \leq \text{enc_mt}$

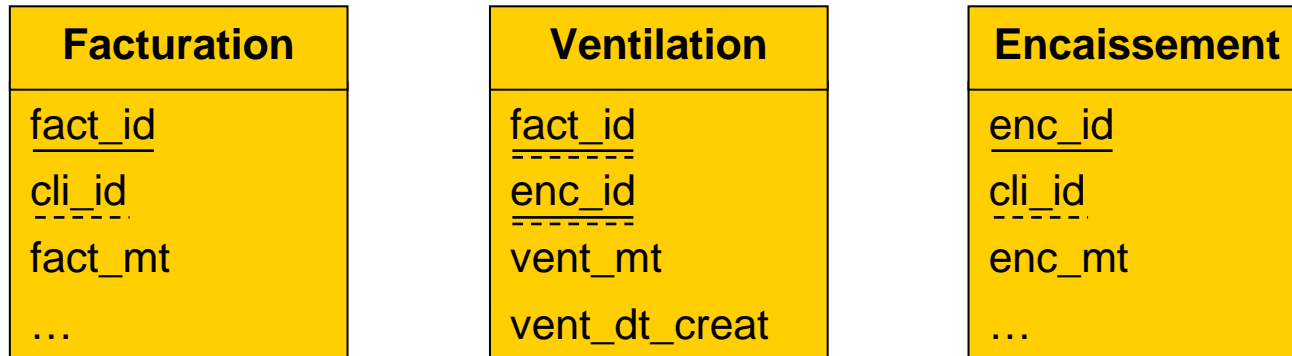


facture et encaissement doivent concerner le même client

```

CREATE TRIGGER trigger_1
NO CASCADE BEFORE INSERT ON ventilation
REFERENCING NEW AS n FOR EACH ROW MODE DB2SQL
WHEN (NOT EXISTS (SELECT 'X' FROM facturation f, encaissement e
                    WHERE e.enc_id = n.enc_id
                        AND f.fact_id = n.fact_id
                        AND e.cli_id = f.cli_id))
BEGIN ATOMIC
    SIGNAL SQLSTATE '00001'('facture.client <> encaissement.client'); END $

```



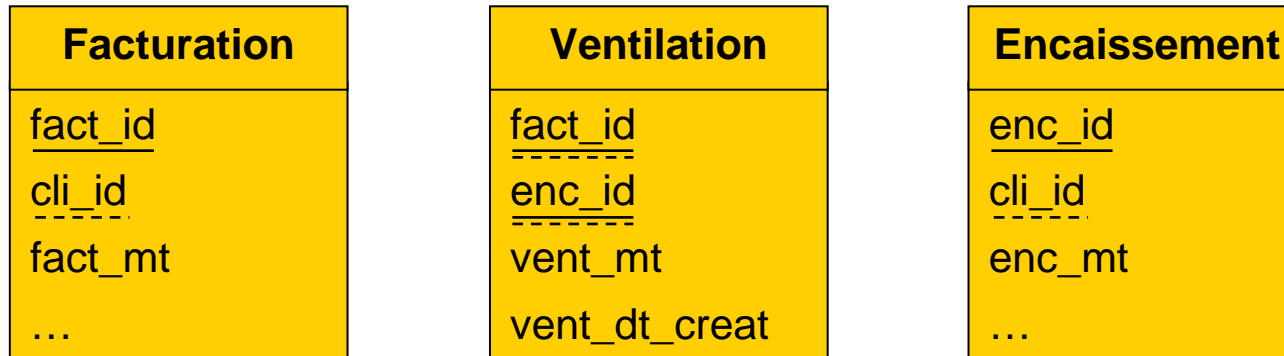
conserver l'historique des ventilations → update interdit



```

CREATE TRIGGER trigger_2
NO CASCADE BEFORE UPDATE ON ventilation
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    SIGNAL SQLSTATE '00002'('update interdit sur Ventilation');
END $

```

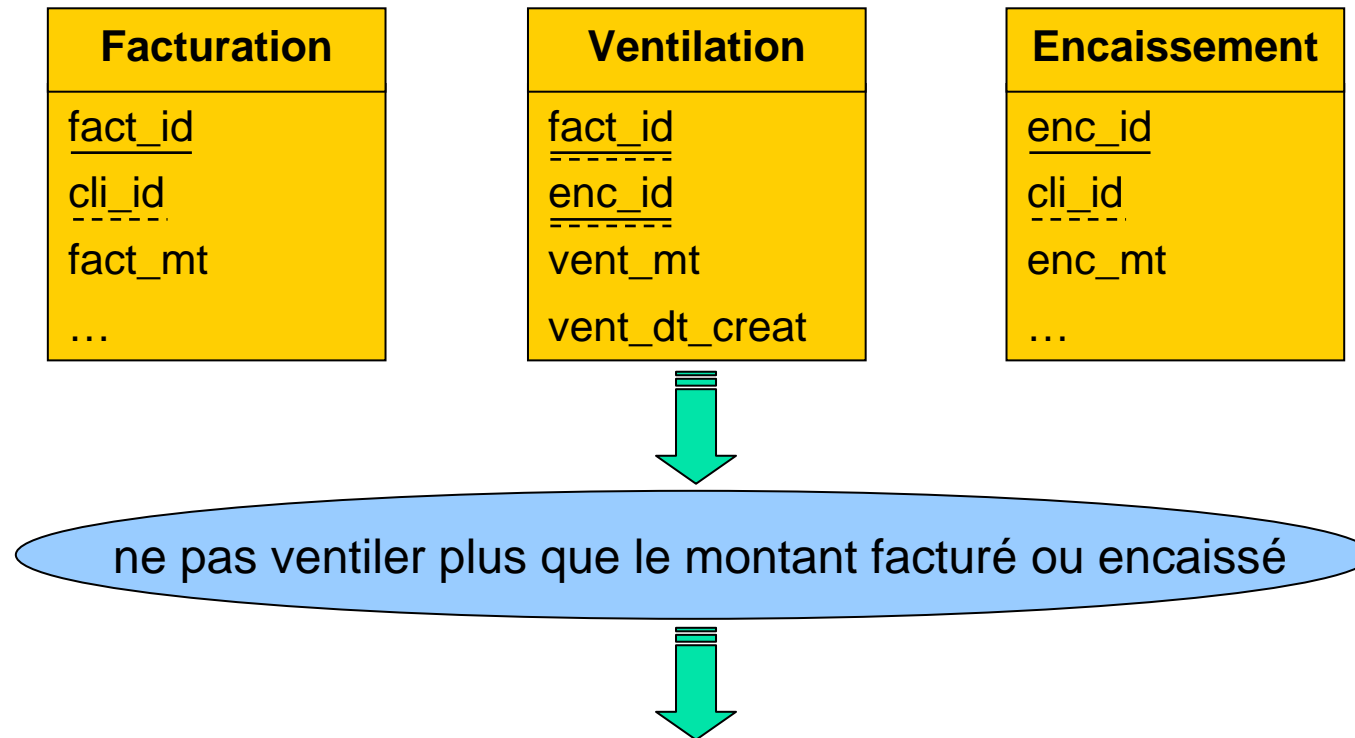


historisation des lignes supprimées

```

CREATE TRIGGER trigger_3
AFTER DELETE ON ventilation
REFERENCING OLD TABLE AS o
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
  INSERT INTO ventilation_hist
    SELECT o.*, CURRENT TIMESTAMP FROM o ;
END$

```



Dans ce cas, préférer les **attributs calculés** plutôt que les triggers de contrôle :

- dans Facturation : `fact_mt_vent` avec un CHECK (`fact_mt >= fact_mt_vent`)
- dans Encaissement : `enc_mt_vent` avec un CHECK (`enc_mt >= enc_mt_vent`)

On peut alimenter ces attributs avec des triggers

Trigger

Benchmark trigger

```
CREATE TABLE tab1  
( t1c1 CHAR(1),  
  t1c2 TIMESTAMP);
```

Table tab1	
t1c1	t1c2
' '	'2007-03-25:15:01:27.367395'
' '	'2007-03-25:15:01:27.399092'
' '	'2007-03-25:15:01:28.568636'
' '	'2007-03-25:15:01:29.768345'
	...

```
CREATE TABLE tab2  
( t2c1 CHAR(1),  
  t2c2 INTEGER);
```

Table tab2	
t2c1	t2c2
'0'	0
'1'	0
'2'	1
'3'	0
'4'	0
'5'	2
'6'	1
'7'	0
'8'	0
'9'	0

Trigger

Benchmarks update timestamp

update sur tab1 → t1c2 alimenté à current timestamp par

SQL applicatif	trigger_1
<pre>UPDATE tab1 SET t1c1 = 'A' , t1c2 = CURRENT TIMESTAMP ;</pre>	<pre>UPDATE tab1 SET t1c1 = 'A' ; CREATE TRIGGER trigger_1 NO CASCADE BEFORE UPDATE ON tab1 REFERENCING NEW AS n FOR EACH ROW MODE DB2SQL BEGIN ATOMIC SET n.t1c2 = CURRENT TIMESTAMP; END \$</pre>

Trigger

Benchmarks update timestamp

		Cl.1 (sec.)	<i>t1c2</i> alimenté par		△ %
			applicatif	<i>trigger_1</i>	
un update d'une ligne	commit en fin de thread	Elaps.	N.S.	N.S.	N.S.
		CPU	0,0031	0,0037	+ 17%
un update de 10000 lignes		Elaps.	0,2457	0,5008	+ 104%
		CPU	0,1175	0,3630	+ 209%
10000 updates d'une ligne chacuns		Elaps.	0,6506	1,2988	+ 100%
		CPU	0,5312	1,1124	+ 109%
	un commit par update	Elaps.	18,5904	16,6137	- 11%
		CPU	1,8446	3,3029	+ 79%

△ % identiques si rollback

Trigger

Benchmarks attributs calculés

insert sur tab1 → compteur t2c2 incrémenté de 1 par

SQL applicatif	trigger_2
<pre>SELECT CURRENT TIMESTAMP INTO :timestp FROM SYSIBM.SYSDUMMY1; INSERT INTO tab1 (t1c2) VALUES(:timestp); UPDATE tab2 SET t2c2 = t2c2 + 1 WHERE t2c1 = SUBSTR(CHAR(:timestp), 26, 1);</pre>	<pre>SELECT CURRENT TIMESTAMP INTO :timestp FROM SYSIBM.SYSDUMMY1; INSERT INTO tab1 (t1c2) VALUES(:timestp); CREATE TRIGGER trigger_2 AFTER INSERT ON tab1 REFERENCING NEW AS N FOR EACH ROW MODE DB2SQL BEGIN ATOMIC UPDATE tab2 SET t2c2 = t2c2 + 1 WHERE t2c1 = SUBSTR(CHAR(n.t1c2), 26, 1); END \$</pre>

Trigger

Benchmarks attributs calculés

		Cl.1 (sec.)	<i>t2c2</i> mise à jour par		△ %
			applicatif	<i>trigger_2</i>	
un insert d'une ligne dans t1	commit en fin de thread	Elaps.	N.S.	N.S.	N.S.
		CPU	0,0032	0,0039	+ 22%
Elaps.		2,0198	4,2113	+ 109%	
CPU		1,7297	3,6581	+ 111%	
10000 inserts d'une ligne chacun dans t1	un commit par insert	Elaps.	21,9059	28,4629	+ 30%
		CPU	5,1778	7,5576	+ 46%

△ % identiques si rollback

Trigger

Benchmarks intégrité référentielle

insert sur tab1 → contrôle que l'unité de t1c2 existe une seule fois dans t2c1

SQL applicatif

```
SELECT CURRENT TIMESTAMP  
INTO :timestp FROM SYSIBM.SYSDUMMY1;  
  
INSERT INTO tab1 (t1c2) VALUES(:timestp);  
  
SELECT COUNT(*) INTO :cpt  
FROM tab2  
WHERE  
t2c1 = SUBSTR(CHAR(:timestp), 26, 1);  
  
IF cpt <> 1 THEN gestion_erreur_integrite;
```

trigger_3

```
SELECT CURRENT TIMESTAMP  
INTO :timestp FROM SYSIBM.SYSDUMMY1;  
  
INSERT INTO tab1 (t1c2) VALUES(:timestp);  
  
CREATE TRIGGER trigger_3  
NO CASCADE BEFORE INSERT ON tab1  
REFERENCING NEW AS N  
FOR EACH ROW MODE DB2SQL  
WHEN (1 <> (SELECT COUNT(*) FROM tab2  
WHERE t2c1 = SUBSTR(CHAR(N.t1c1), 26, 1)))  
BEGIN ATOMIC  
    SIGNAL SQLSTATE '99999'  
    ('erreur d'intégrité avec tab2'); END $
```

Trigger

Benchmarks intégrité référentielle

		Cl.1 (sec.)	<i>tab2</i> mise à jour par		Δ %
			applicatif	trigger_3	
un insert d'une ligne dans t1	commit en fin de thread	Elaps.	N.S.	N.S.	N.S.
		CPU	0,0040	0,0049	+ 23%
10000 inserts d'une ligne chacun dans t1	commit en fin de thread	Elaps.	2,4996	6,1910	+ 148%
		CPU	1,5090	4,1202	+ 173%
	un commit par insert	Elaps.	24,3961	31,9249	+ 31%
		CPU	4,6906	7,9970	+ 70%

Δ % identiques si rollback



Trigger sur DB2 z/OS



Administration

Trigger

CREATE TRIGGER package

- CREATE TRIGGER équivaut à un bind trigger package
- une occurrence dans SYSIBM.SYSPACKAGE avec la colonne TYPE = 'T'
- une occurrence dans SYSIBM.SYSTRIGGERS avec les propriétés et le code du trigger
- désignation du package : *nom_schema.nom_trigger* le schema est l'équivalent de la collection pour les extensions objets de DB2 (trigger, SP, UDF)

Trigger

Options de bind (CREATE TRIGGER)

ACTION(ADD)

ISOLATION(CS)

CURRENTDATA(YES)

NOREOPT(VARS) and NODEFER(PREPARE)

DBPROTOCOL(DRDA)

OWNER(authorization ID)

DEGREE(1)

QUERYOPT(1)

DYNAMICRULES(BIND)

PATH(path)

ENABLE(*)

RELEASE(COMMIT)

ENCODING(0)

SQLERROR(NOPACKAGE)

EXPLAIN(NO)

QUALIFIER(authorization ID)

FLAG(I)

VALIDATE(BIND)

Trigger



REBIND TRIGGER PACKAGE options

CURRENTDATA

EXPLAIN -- YES pour avoir le chemin dans la plan_table

FLAG

IMMEDWRITE

ISOLATION

RELEASE

Trigger



Trigger package invalidé ou supprimé

- trigger package invalidé si DROP d'une table, index ou view contenu dans le corps du trigger (rebind automatique à l'exécution)
- trigger package supprimé si :
 - DROP TRIGGER
 - DROP de la table qui le déclenche

Retrouver les informations du catalogue

- Pour lister **les triggers d'une table** dans l'ordre d'exécution :

```
SELECT DISTINCT SCHEMA, NAME, TRIGTIME, TRIGEVENT,  
    GRANULARITY, CREATEDTS  
FROM SYSIBM.SYSTRIGGERS  
WHERE TBNAME = 'name' AND TBOWNER = 'owner' ORDER BY CREATEDTS;
```

- Pour retrouver **le code d'un trigger** :

```
SELECT TEXT, SEQNO FROM SYSIBM.SYSTRIGGERS  
WHERE SCHEMA = 'schema' AND NAME = 'trigger' ORDER BY SEQNO;
```

- Pour déterminer **les triggers à rebinder** :

```
SELECT COLLID, NAME FROM SYSIBM.SYSPACKAGE  
WHERE TYPE = 'T' AND (VALID = 'N' OR OPERATIVE = 'N');
```

Trigger

Privilèges nécessaires

Pour créer un trigger il faut les autorisations suivantes :

- le privilège CREATEIN pour le schema ou tous les schema, ou SYSADM ou SYSCTRL
- les privilèges TRIGGER ou ALTER sur la table concernée, ou DBADM sur la database de cette table, ou SYSADM ou SYSCTRL
- le privilèges sur les actions exécutées par le trigger

Trigger

SQL terminator

Pour exécuter un ordre CREATE TRIGGER il faut changer le caractère de fin d'ordre SQL, le point virgule est déjà utilisé dans le corps du trigger.

- en SPUFI changer le paramètre SQL terminator
- en DSNTEP2 et DSNTIAD utiliser au choix :
 - le commentaire --SET TERMINATOR # dans le code SQL
 - le paramètre SQLTERM :
RUN PROGRAM(DSNTIAD) PLAN(*plan*) PARM('SQLTERM(#)') -

Trigger

Cascade de trigger

Les triggers peuvent se déclencher en cascade :

update T1 → trigger_1 → insert T2 → trigger_2 → ... → trigger_n

On dit qu'ils sont de niveaux différents.

DB2 supporte **16 niveaux** de triggers, ce qui évite les boucles.

On peut voir le niveau maximum atteint par les triggers dans l'accounting (IFCID 16) :

```
MISCELLANEOUS
-----
BYPASS COL          :          0.00
→ MAX SQL CASCAD LEVEL :          0.00
MAX STOR LOB VALUES :          0.00
```



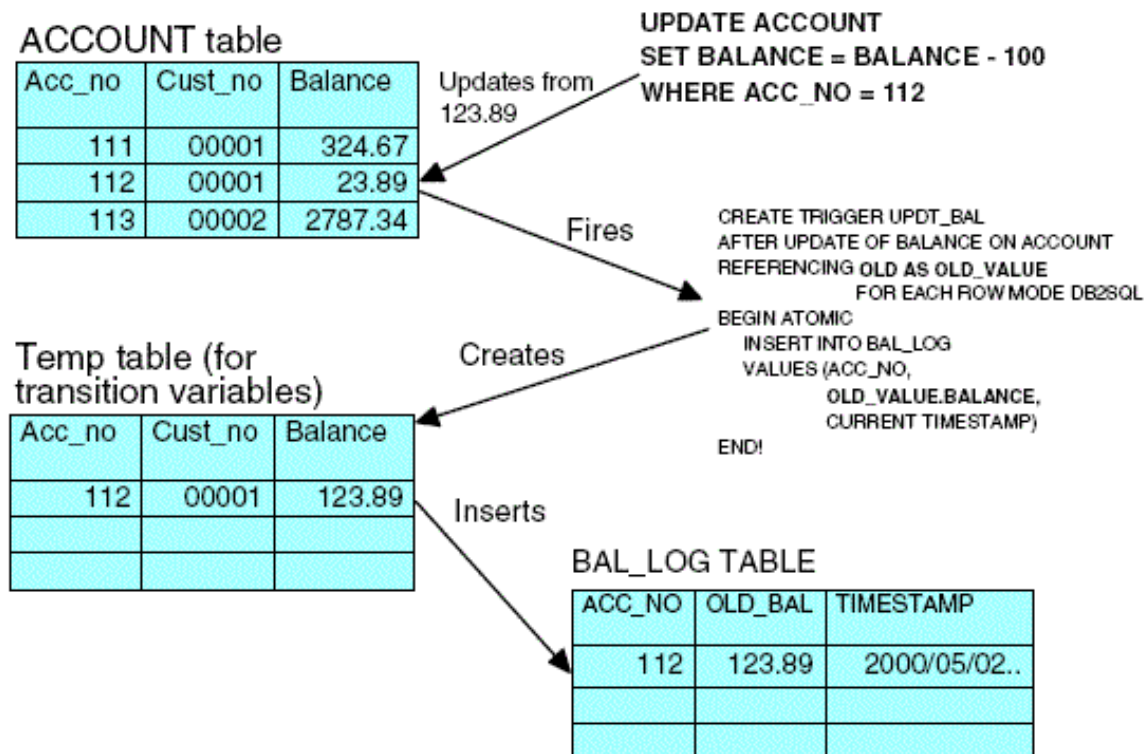
Ordre des actions

- Les triggers déclenchés par le même évènement (ex : AFTER INSERT ON tab1) sont exécutés dans l'ordre de création (CREATEDTS dans SYSTRIGGER).
- insert 10 lignes → *trigger_1* activé 10 fois → *trigger_2* activé 10 fois → ...
- Pour une mise à jour, DB2 effectue dans l'ordre :
 - 1 – détermine les lignes “candidates”
 - 2 – exécute les triggers before
 - 3 – effectue les mises à jour
 - 4 – applique les contraintes et les checks
 - 5 – exécute les triggers after
- Si un des triggers effectue aussi une mise à jour, DB2 répète les opérations de 1 à 5 dans l'ordre.
- Il est nécessaire de bien analyser l'ordre des actions.

Trigger

Workfile (table temporaire)

DB2 crée un workfile pour les triggers **AFTER** avec clause **REFERENCING** (variables de transition)



Trigger

Workfile (table temporaire)

- Workfile uniquement pour les trigger AFTER avec clause REFERENCING
- Workfile similaire au CREATED TEMPORARY TABLE (pas de logging, pas définie au catalogue DB2)
- 10 insert → 10 exécutions du trigger → 10 créations de workfiles
- Un workfile par clause REFERENCING (NEW et OLD) et un autre workfile si un tri est nécessaire dans la table temporaire
- Impacte les performances CPU et l'espace disque
- En V8 si la condition WHEN n'est pas satisfaite, pas de création de workfile
- Procédures stockées et UDF peuvent accéder aux tables temporaires
- Informations sur l'utilisation des tables temporaires dans les IFCID 16 et 17

Trigger

Performances et tuning

- Le coût d'un trigger est équivalent au coût d'un FETCH (hors exécution du code contenu dans le corps)
- Si le trigger n'est pas déclenché le coût est négligeable (ex : INSERT dans une table avec un trigger UPDATE)
- Une condition WHEN mal codée impacte les performances dans tous les cas
- Le trigger package et l'object descriptor OBD du trigger (appartient au DBD de la table) sont chargés dans l'EDM pool, en tenir compte pour définir sa taille.
- Il peut être efficace de faire un rebind trigger avec l'option RELEASE(DEALLOCATE), mais surveiller alors les ressources.
- N'utiliser les workfiles (REFERENCING) que si nécessaire, contrôler le nombre de workfile, les contentions, le nombre et la longueur des lignes
- Comparer le coût d'un trigger par rapport à du code inséré dans l'application
- Pour une même action si possible regrouper le code dans un seul trigger
- Ne pas utiliser les triggers en remplacement des contraintes ou pour la simple propagation de données dans DB2

Trigger

Accounting

Classes 1 et 2

Classes 7 et 8

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)		TOTAL
ELAPSED TIME	3:09.56694	3:09.56328		
NONNESTED	2:29.43810	2:29.43444		
STORED PROC	0.000000	0.000000		
UDF	0.000000	0.000000		
TRIGGER	40.128841	40.128841	TRIGGERS	
CPU TIME	1:22.40848	1:21.12886	STMT TRIGGER	0
AGENT	1:22.40848	1:21.12886	ROW TRIGGER	2314745
NONNESTED	46.563780	45.284165	SQL ERROR	0
STORED PROC	0.000000	0.000000		
UDF	0.000000	0.000000		
TRIGGER	35.844696	35.844696		
PAR.TASKS	0.000000	0.000000		
SERVICE UNITS				
	CLASS 1	CLASS 2		
CPU	692143	681396		
AGENT	692143	681396		
NONNESTED	391086	380339		
STORED PROC	0	0		
UDF	0	0		
TRIGGER	301057	301057		
PAR.TASKS	0	0		
TRIGG_1				
	VALUE			
TYPE	PACKAGE			
SUCC AUTH CHECK	NO			
LOCATION	FMP9121			
COLLECTION ID	SVEIL		...	
PROGRAM NAME	TRIGG_1			
CONSISTENCY TOKEN	1809E2A505A2948A			
ACTIVITY TYPE	TRIGGER			
SCHEMA NAME	C00			
ACTIVITY NAME	TRIGG_1			

Trigger



Evolutions à tester

- Refresh des MQT
- Raise_error
- Case
- Instead of trigger (V9)



Trigger sur DB2 z/OS



Annexes

Trigger

UPDATE CASCADE

Exemple : modification d'identifiant dans la table centre_teletrans
➔ report de la modification dans la table param_teletrans :

```
CREATE TRIGGER trigger_1
AFTER UPDATE OF tele_id ON centre_teletrans
REFERENCING OLD AS o
                NEW AS n
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    UPDATE param_teletrans
        SET tele_id = n.tele_id
        WHERE tele_id = o. tele_id;
END #
```

Trigger

PENDANT DELETE

Exemple : suppression de la dernière pièce composant un moteur
➔ suppression du moteur correspondant dans la table moteur

```
CREATE TRIGGER trigger_1
AFTER DELETE ON piece
REFERENCING OLD AS o
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
DELETE FROM moteur m
WHERE m.moteur_id = o.moteur_id
      AND NOT EXISTS (SELECT 'X' FROM piece p
                      WHERE p.moteur_id = o.moteur_id);
END #
```

Trigger

Unicité sur des colonnes longues

- En V7 pas d'index unique si plus de 255 caractères
- En V8 pas d'index unique si plus de 2000 caractères si l'index n'est pas partitionné, sinon 255 maximum
- L'unicité peut être assurée par un trigger

Trigger

Unicité sur des colonnes longues

```
CREATE TABLE tab1 (col1 VARCHAR(500)) ;

CREATE TRIGGER trigger1
NO CASCADE BEFORE INSERT ON tab1
REFERENCING NEW AS n
FOR EACH ROW MODE DB2SQL
WHEN (EXISTS
  (SELECT 'x' FROM tab1
   WHERE SUBSTR(tab1.col1, 1, 255) = SUBSTR(n.col1, 1, 255)
        AND SUBSTR(tab1.col1, 256, 244)= SUBSTR(n.col1, 256, 244)))
BEGIN ATOMIC
  SIGNAL SQLSTATE '99999'('Duplicate key on tab1'); END#
```

(en V7 les varchar ne sont pas autorisés dans un WHEN, d'où le SUBSTR)

Trigger

Limiter les mises à jour par plage horaire

Exemple : interdire les INSERT entre minuit et 9h

```
CREATE TRIGGER trigger1
NO CASCADE BEFORE INSERT ON tab1
FOR EACH ROW MODE DB2SQL
WHEN (TIME(CURRENT TIMESTAMP) <'09:00:00')
  SIGNAL SQLSTATE '70003' ('INSERT interdit avant 9h');
END$
```

Restrictions diverses

- Les triggers sur PLAN_TABLE, DSN_STATEMNT_TABLE ne se déclenchent pas si c'est DB2 qui insère les lignes, notamment par EXPLAIN
- Une requête faisant appel à une table sur laquelle est défini un trigger est classée en cost category B dans la DSN_STATEMENT_TABLE
- Pas de rename sur une table avec trigger
- Dans le cas d'un mass delete sur un TS segmenté, s'il existe un delete trigger sur la table, DB2 écrit les lignes dans la log comme TS non segmenté
- Le CREATE d'un trigger invalide les packages contenant un ordre identique au déclencheur (REBIND automatique à l'exécution)
- Pas de host variables ou des parameter markers dans le corps d'un trigger
- Pas de LIKE dans une WHERE clause avec une variable de transition