IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

All About Triggers

Steve Thomas
CA Technologies

Session code: E17

Thursday 8th November 2018 @ 09:40

Platform: Db2 for z/OS

Agenda

- Introduction and Common features
- Before Triggers
- After Triggers
- Instead of Triggers
- Advanced Triggers (Db2 12 for z/OS)
- Trigger Syntax Obfuscation (Db2 12 for z/OS)

Introduction - What is a Trigger?

- Industry standard Database construct
- Defines action to be taken whenever updating DML statement runs
  - INSERT, UPDATE and DELETE supported natively
  - MERGE handled via INSERT and UPDATE Triggers
  - TRUNCATE does not fire DELETE Triggers – see later
  - *Note:* There is no such thing as a SELECT or FETCH Trigger
- Can run Before, After or Instead of the original DML statement
  - BEFORE/AFTER Triggers are defined on Tables, INSTEAD OF on Views
- Operate at the Row or Statement level
  - Row level can be used for all Trigger types
  - Statement level is only supported for AFTER triggers

Why use a Trigger?

- Huge number of potential reasons - some replaced by new features
- Some typical examples:
  - Data Validation
  - Data Propagation
  - Embedding business rules in the DBMS to ensure they're followed
  - Simplify programming
  - Create an Audit Trail (although TEMPORAL data is now becoming more common)
  - Enforcing Referential Integrity
  - Making Complex views updateable
  - Improving Security
  - … and many more

Trigger Support in Db2 for z/OS

- Trigger support has been available for almost 20 years
  - First introduced in Version 6 in 1999 - BEFORE and AFTER Triggers
  - Version 9 in 2007 added INSTEAD OF Triggers
- Now in Db2 12 for z/OS (2016)
  - We get ADVANCED Triggers
  - Older styles are now called BASIC Triggers
    - Can still be created – essentially no change
    - But as we'll see Advanced Triggers provide much more functionality
  - Db2 12 also adds support for obfuscation of Trigger syntax

Common CREATE TRIGGER syntax

CREATE TRIGGER *trigger-name*	
NO CASCADE BEFORE/AFTER/INSTEAD OF	<- Trigger Type
INSERT/UPDATE OF *column-list*/DELETE ON *table/view*	<- Triggering Event
REFERENCING *transition-clauses*	<- Transition Variables
FOR EACH *ROW/STATEMENT*	<- Row or Statement Trigger
MODE DB2SQL	<- Omit for an Advanced Trigger
SECURED/NOT SECURED	<- Uses Row/Column access controls
[WHEN *search-condition*]	<- Optional Condition clause
[BEGIN ATOMIC]	<- Trigger has multiple statements
Triggered-Statement(s)	<- What Trigger actually does
[END];	<- Closes ATOMIC clause

When does DB2 execute Trigger Action?

- Mainly defined by whether it's a Row or Statement Trigger
  - But also subject to any Condition Clause – see next slide
- FOR EACH ROW
  - Trigger Action executed for each modified Row of subject table
  - If No Rows are modified then the Trigger will not execute
- FOR EACH STATEMENT
  - Trigger Action executed once per DML – will always be an AFTER Trigger
  - A Statement Trigger will always execute even if no data is modified
- ON UPDATE OF *column-list*
  - UPDATE only, column-List clause optional, each column can only be listed once
  - Trigger is fired if *ANY* of the columns listed have been updated

Trigger Condition

- Defined using the optional WHEN clause
- WHEN Clause evaluates to True, False or Unknown
  - The Trigger fires only if the Condition is TRUE
  - Useful to avoid any access to another table (e.g. if Child Count maintained by Parent)
- If there is no WHEN clause then the Trigger always fires
- Restrictions:
  - Cannot Specify a WHEN clause on an INSTEAD OF Trigger
  - System-Period Temporal if Trigger Bound with SYSTEMTIMESENSITIVE(YES)
  - Archive-enabled table if Trigger Bound with ARCHIVESENSITIVE(YES)
- See sample Trigger on Slide 20 for an example

Transition Variables and Tables

- How a Trigger refers to data in the Row or Table it acts on
  - Both the original data and any new values after updates have been applied
- Row Triggers usually use Transition Variables (values for one row)
  - Although AFTER and INSTEAD OF Row Triggers can use Transition Tables
- Statement Triggers always use Transition Tables (the whole table)
- Defined by REFERENCING clause when required
  - DELETE Triggers can refer to OLD data
  - INSERT Triggers can refer to NEW data
  - UPDATE Triggers can refer to both OLD and NEW data
- Any Transition variable used must be defined - REFERENCING clause
- XML is not supported – even in an Advanced Trigger

Sample Row Trigger using Transition Variables

```
CREATE TRIGGER T1
NO CASCADE BEFORE UPDATE OF SALARY ON EMP
REFERENCING OLD AS O NEW AS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
  SET N.BONUS = N.SALARY * 0.1;
  SET N.NEXT_REVIEW = CURRENT DATE + 1 YEARS;
  SET N.OLD_SALARY = O. SALARY;
END
```

So what happened?

```
UPDATE EMP
SET SALARY = 50000
WHERE EMPNO = '123456';
```

BEFORE

```
For Table   => THOST17.EMP
Browse Mode => F
SSID: D12A ----------FETCH STATUS: COMPLETE-
##.COLUMN NAME          NULL  DATA FOR ROW # 1
A1.EMPNO                      123456
A2.FIRSTNME                   STEVE
A3.LASTNAME                   THOMAS
A4.HIREDATE             N     2018-08-03
A5.JOB                  N     DEADLOSS
A6.EDLEVEL              N     5
A7.SEX                  N     M
A8.BIRTHDATE            N     1960-09-01
A9.SALARY               N     10000.00
B1.BONUS                N     2000.00
B2.OLD_SALARY           N     8000.00
B3.NEXT_REVIEW          N     2018-03-20
```

AFTER

```
For Table   => THOST17.EMP
Browse Mode => F
SSID: D12A ----------FETCH STATUS: COMPL
##.COLUMN NAME          NULL  DATA FOR ROW
A1.EMPNO                      123456
A2.FIRSTNME                   STEVE
A3.LASTNAME                   THOMAS
A4.HIREDATE             N     2018-08-03
A5.JOB                  N     DEADLOSS
A6.EDLEVEL              N     5
A7.SEX                  N     M
A8.BIRTHDATE            N     1960-09-01
A9.SALARY               N     50000.00
B1.BONUS                N     5000.00
B2.OLD_SALARY           N     10000.00
B3.NEXT_REVIEW          N     2019-03-19
```

What SQL can you use in a (Basic) Trigger?

	BEFORE	AFTER	INSTEAD OF
CALL	✔	✔	✔
DELETE (Searched)		✔	✔
fullselect	✔	✔	✔
INSERT		✔	✔
MERGE		✔	✔
REFRESH		✔	✔
SET *transition-variable*	✔		
SIGNAL	✔	✔	✔
TRUNCATE		✔	✔
UPDATE (Searched)		✔	✔
VALUES	✔	✔	✔

Notes on Permitted SQL Statements

- Note that no Complex logic is allowed
  - Not even a basic IF – THEN – ELSE construct (although Conditions can give you a little control)
  - If needed must use CALL instead but both performance and functional implications
- Setting or Changing data in the DML itself (SET) limited to BEFORE Triggers
- (Basic) Triggers have no access to program host variables, parameter markers, undeclared transition variables or Declared temporary tables
- Tables, Views and Procedures referred to in a Trigger must be local
- Before Triggers cannot refer to the triggering table in *fullselect*
- Business Time is not supported
- And more…

What Statements are missing?

- ALTER
- COMMENT
- CREATE
- DELETE
- DROP
- EXCHANGE
- GRANT
- LABEL
- LOCK TABLE
- MERGE
- REFRESH TABLE
- RENAME
- REVOKE
- TRUNCATE
- UPDATE (non-searched)

- Includes direct or indirect use of these in Called Procedures
- **Perhaps most critical SQL Procedure Language (SQL PL) not supported**

MERGE and TRUNCATE

- **MERGE** supported by a combination of INSERT and UPDATE Triggers
  - Be careful that both exist and are consistent
  - For example if one is defining new Columns such as the example on Slide 10
- **TRUNCATE** does not invoke DELETE TRIGGERs
  - IGNORE DELETE TRIGGERS is the default clause for TRUNCATE
    - This requires ALTER or higher Authority on the Table to execute the TRUNCATE
    - Means TRUNCATE cannot generally be used if a DELETE Trigger exists
  - Other option is RESTRICT WHEN DELETE TRIGGERS
    - This causes TRUNCATE to fail with SQLCODE -20356
  - A Mass DELETE with No WHERE clause will fire Triggers as normal
    - Generally prefer TRUNCATE but beware of losing DELETE Trigger functionality

Trigger Firing Sequence

- Multiple Triggers of the same type can fire on a single Statement
  - What defines the Order that this happens?
  - Could be significant, such as when a BEFORE Trigger changes a column value
- The answer is the TIMESTAMP they were created
  - The Oldest Trigger fires first, then the next and so on
- Represents a headache if a Trigger needs to change
  - Effectively there is no ALTER TRIGGER functionality for Basic Triggers
    - Only possible to change SECURED and NOT SECURED options
  - Drop all Triggers created after affected one and recreate them in correct Order
  - Significant Care required!
- Another problem that has been helped by Advanced Triggers

Execution Order with Triggers

1. BEFORE Triggers
2. VIEW with CHECK option
3. CHECK Constraints
4. Unique Index checks
5. Referential Constraint Checks
6. SQL or INSTEAD OF Trigger
7. AFTER Triggers

Trigger Packages for BASIC Triggers

- Each Trigger has an Associated Package created by CREATE TRIGGER
- No EXPLAIN is available at CREATE time
  - But you can REBIND TRIGGER PACKAGE using EXPLAIN YES
- REBIND also updates the Access Path
  - Limited set of Options can be changed such as ISOLATION and APPLCOMPAT
- Trigger Package is dropped when the Trigger is Deleted
  - Cannot use FREE PACKAGE
- How else do Trigger Packages differ from standard Packages?
  - Can only be Bound locally
  - Never become Inoperative but can become Invalid (e.g. if an Index is dropped)
- Once again Advanced Triggers operate differently - see later

BEFORE Triggers

- Usually used to change data that will be updated in the Statement
  - Or to provide additional values not in original SQL
- Sometimes used to impose standards or restrictions on changes made
  - E.g. to set limits for the pay rise in the Trigger – see next slide
- May also prevent execution of the Statement altogether
  - Useful for data validation beyond scope of a CHECK constraint
- Cannot contain any update style statements
  - Means a BEFORE Trigger never Cascades – hence the NO CASCADE BEFORE syntax
- Can have Before Trigger on Cloned object but not on a clone
  - The BEFORE trigger applies to both

Using a BEFORE Trigger to impose Thresholds

```
CREATE TRIGGER T2
NO CASCADE BEFORE UPDATE OF SALARY ON EMP
REFERENCING OLD AS O NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN (N.SALARY > (O.SALARY * 1.03))
BEGIN ATOMIC
   SIGNAL SQLSTATE '75001' ('Invalid Salary Increase – More than 3%');
END
```

Results

```
UPDATE EMP
SET SALARY = 60000
WHERE EMPNO = '123456';
```

BEFORE

```
For Table   => THOST17.EMP
Browse Mode => F
SSID: D12A ----------FETCH STATUS: COMPL
##.COLUMN NAME          NULL   DATA FOR ROW
A1.EMPNO                       123456
A2.FIRSTNME                    STEVE
A3.LASTNAME                    THOMAS
A4.HIREDATE             N      2018-08-03
A5.JOB                  N      DEADLOSS
A6.EDLEVEL              N      5
A7.SEX                  N      M
A8.BIRTHDATE            N      1960-09-01
A9.SALARY               N      50000.00
B1.BONUS                N      5000.00
B2.OLD_SALARY           N      10000.00
B3.NEXT_REVIEW          N      2019-03-19
```

RESULT

```
 UPDATE EMP
 SET SALARY = 60000
 WHERE EMPNO = '123456'
;
DSNT408I SQLCODE = -438, ERROR:  APPLICATION RAISED ERROR WITH
         DIAGNOSTIC TEXT: INVALID SALARY INCREASE - MORE THAN 3%
DSNT418I SQLSTATE   = 75001 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXRTYP SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 1  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'00000001'  X'00000000'  X'00000000'
         X'FFFFFFFF'  X'00000000'  X'00000000' SQL DIAGNOSTIC
         INFORMATION
BPA0012E: DB2 SQL/DDL ERROR HAS OCCURRED - ROLLBACK ISSUED.
```

AFTER Triggers

- AFTER Triggers are more often used to update other tables
  - Although you can still Signal an error if you wish to do so
- DB2 treats the Statement and all Triggers as a single operation
- Typical Uses include:
  - A basic replication tool, albeit synchronous as updates to target in same UOW
  - Maintaining Counts of Children on Parent Tables
    - Avoid access to the Child Table in a JOIN or need for COUNT(*) on Child rows
  - Automated Stock Control
    - Automatically re-order a part if the stock falls below a certain value
  - And many, many more
- The one type of Trigger that can be at the Statement Level

Cascading Triggers

- One big difference to a Before Trigger is they can Cascade
  - AFTER Trigger on T1 updates Table T2 in the Trigger Action
  - AFTER Trigger on T2 then fires and updates table T3
  - And so on…
- Limit of16 nested levels – similar to nested Stored Procedures
  - Exceeding this causes SQLCODE -724
- Some restrictions worth noting:
  - Table Modified by a Cascaded Trigger cannot be referred to again
  - A Table referenced higher up the tree cannot be updated by the Trigger

Sample AFTER Triggers to maintain Parent Counts

```
CREATE TRIGGER INSEMP
AFTER INSERT ON EMP
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
UPDATE DEPT D
        SET STAFF = STAFF + 1
        WHERE D.DEPTNO = N.WORKDEPT;
END
```

```
CREATE TRIGGER DELEMP
AFTER DELETE ON EMP
REFERENCING OLD AS O
FOR EACH ROW MODE DB2SQL
UPDATE DEPT D
        SET STAFF = STAFF - 1
        WHERE D.DEPTNO = O.WORKDEPT;
END
```

- Simplistic example – can easily get more sophisticated than this
- E.g. Inserting new Departments if they did not exist

Another, from DB2 Manuals, restocking via a UDF

```
CREATE TRIGGER REORDER
AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
REFERENCING NEW AS NROW
FOR EACH ROW MODE DB2SQL
WHEN (NROW.ON_HAND < 0.10 * NROW.MAX_STOCKED)
BEGIN ATOMIC
VALUES(ISSUE_SHIP_REQUEST(NROW.MAX_STOCKED - NROW.ON_HAND, NROW.PARTNO));
END
```

INSTEAD OF Triggers

- Unlike other Trigger types these operate exclusively on Views
- Often used to allow Update DML Statements on Read only Views
- But be careful what you ask for!
  - Unless carefully defined they can cause huge confusion or worse
  - See Employee sample that follows for an example of what I mean
- Can also use INSTEAD OF Trigger for Security purposes
  - Create a View which restricts access to sensitive data but is updateable
- Only 1 INSTEAD OF Trigger for each Operation is allowed per View
- Not allowed to use WHEN clause on an INSTEAD OF Trigger
  - Although a lot of Views would be non-updateable anyway if Trigger did not fire

INSTEAD OF Triggers

- Huge list of exclusions on Target View including (not exclusive):
  - View defined using WITH CASCADED CHECK
  - Underlying data contains different Encoding Schemes or CCSIDs
  - An underlying Column is based on any of the following data types:
    - LOB, XML, ROWID, Identity Column, Security Label, Row Change Timestamp, Row-Begin or Row-End Column, Transaction Start ID Column
  - Contains Field Procedures
  - All underlying tables are Catalog Tables
  - All underlying tables are Created Global Temporary Tables
  - All underlying tables are Clone tables
  - View has other dependent Views

Sample INSTEAD OF Trigger for weather tables

```
CREATE TABLE WEATHER
 (CITY VARCHAR(25),
  TEMPF DECIMAL(5,2));

CREATE VIEW CELSIUS_WEATHER
(CITY, TEMPC) AS
   SELECT CITY, (TEMPF-32)/1.8
   FROM WEATHER;
```

```
CREATE TRIGGER CW_INSERT
INSTEAD OF INSERT ON
CELSIUS_WEATHER
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
INSERT INTO WEATHER VALUES
(N.CITY, (1.8*N.TEMPC)+32);
END
```

DDL for VEMPDEPT and INSTEAD OF TRIGGER

```
CREATE VIEW VEMPDEPT
    ( DEPTNO,DEPTNAME ,EMPNO, FIRST,LAST,
      WORKDEPT, SALARY, COMM) AS
SELECT ALL DEPTNO , DEPTNAME , EMPNO , FIRSTNME,
LASTNAME , WORKDEPT , SALARY, COMM
FROM DEPT RIGHT OUTER JOIN EMP
ON WORKDEPT = DEPTNO
```

- **Does this look OK?**
- **Both Statements execute RC=0**

```
CREATE TRIGGER TRIGVIEW
INSTEAD OF UPDATE ON VEMPDEPT
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
   UPDATE EMP E
   SET E.SALARY = N.SALARY
   WHERE E.EMPNO  = N.EMPNO;
END
```


29

Update Statement and Result from EMP

```
For Table    => THOST17.EMP                          > Row number=> 1 OF 1
Browse Mode => C                                       Max Char   => 070
SSID: D12A ----------FETCH STATUS: COMPLETE------------------------- THOST17
      EMPNO  FIRSTNME       LASTNAME            N:SALARY       N:COMM
      000010 CHRISTINE      HAAS                N     52750.00 N      4220.00
```

```
UPDATE VEMPDEPT
SET SALARY = 10000,
    COMM   = 5000
WHERE EMPNO = '000010
```

```
For Table    => THOST17.EMP                          > Row number=> 1 OF 1
Browse Mode => C                                       Max Char   => 070
SSID: D12A ----------FETCH STATUS: COMPLETE------------------------- THOST17
      EMPNO  FIRSTNME       LASTNAME            N:SALARY       N:COMM
      000010 CHRISTINE      HAAS                N     10000.00 N      4220.00
******************************** BOTTOM OF DATA ********************************
```

Advanced Triggers

- Added in Db2 12 for z/OS at Function Level M500
- Remove many of the restrictions imposed by Basic Triggers
  - Define, Reference and Assign SQL Variables including Global Variables
  - Include most SQL statements including Dynamic SQL with Literal Concentration
  - Include SQL PL – allows much more comprehensive logic
  - Explicitly assign options including BIND options such as EXPLAIN
  - SQL Comments
  - Version Support and changing Trigger Options or Regeneration of Trigger Body
  - Transition variables can be Nullable
  - Unhandled Warnings are returned to the Application Program
  - Changes to Transition Variables are returned to the invoking application
  - DEBUG support
- According to the Documentation there is “some” overhead – still tbc!

Creating an Advanced Trigger

- Create by leaving MODE DB2SQL out of the CREATE TRIGGER Syntax
- Advanced CREATE TRIGGER supports OR REPLACE option
  - Creates a New Trigger or replaces existing one in one Statement
  - Can create a new Version of a Trigger or Replace an existing Version
  - There is also essentially complete ALTER TRIGGER support
  - These changes avoid the Trigger Firing Sequence issue referred to earlier
- Advanced CREATE TRIGGER can set most PACKAGE BIND options
  - Not all are the same Syntax as a regular BIND or REBIND
  - For example WITH EXPLAIN or WITHOUT EXPLAIN
- Advanced Trigger Packages are regarded almost as Regular Packages
  - No Longer need to use REBIND TRIGGER PACKAGE Statement
  - Regular REBIND PACKAGE can (must) be used

An Advanced Trigger borrowed from IDUG Paper

```
CREATE TRIGGER myafter2
AFTER INSERT ON t_source
REFERENCING NEW ROW AS n
FOR EACH ROW
BEGIN
DECLARE multiplier INTEGER CONSTANT 10; DECLARE parm2 INTEGER;
DECLARE parm3 INTEGER;
SET parm2 = n.data1;
SET parm3 = n.data2;
CALL mysp1(multiplier, parm2, parm3);
INSERT INTO t_target VALUES (n.keycol, parm2, parm3, n.data3);
END
```

This would not be possible with a Basic Trigger for several reasons including:

- Declaring Variables
- Using the result variables from mysp1

Trigger Text Obfuscation

- Provides Protection of Intellectual Copyright if you distribute Applications
  - Trigger Text is rendered unreadable when viewed by users and in Catalog
  - Also supported for Functions and Procedures
- You cannot ALTER an Obfuscated Trigger
  - But you can always Drop and Recreate it
  - Still checking whether this causes Trigger firing sequence issues again
- To me (a non-IBMer) this looks to be a first step not the finished article
  - Strong encryption not used – *"not meant to be used for security contexts"*
  - Worse still the non-obfuscated SQL Statements are openly visible in SYSPACKSTMT!
  - But any Program Logic in the Trigger Text is secured

How do you do this?

```
SELECT WRAP(' CREATE TRIGGER T1
NO CASCADE BEFORE UPDATE OF SALARY ON EMP
REFERENCING OLD AS O NEW AS N
FOR EACH ROW
BEGIN ATOMIC
  SET N.BONUS = N.SALARY * 0.1;
  SET N.NEXT_REVIEW = CURRENT DATE + 1 YEARS;
  SET N.OLD_SALARY = O. SALARY;
END')
FROM SYSIBM.SYSDUMMY1
```

Produces this result:

```
CREATE TRIGGER T1 WRAPPED DSN12015
| ablGWmdiWmtyTmduTmJqTmtaUmtGUnteU
mZKWmtqWidaWmdaWmdaXmdyWncaGica
| GK6ot_81NzyodncdrRIJFp_tBjpJeIwg_dTKN
HcdtHPSaNCpmqBKH2pMwExkRTJW
| Zr:dJd0_gSbehW:4Xx1UGPGnDxvmJfa5ZAG
Or_1sfFiyaPrkOXzt5UMTmsASfyJR
| ksbPfM2dlATbq:0RW
```

DDL runs on a DB2 12 Subsystem at M500 - the Trigger is created correctly

All About Triggers

Steve Thomas
CA Technologies

Session code: E17

Thursday 8th November 2018 @ 09:40

Platform: Db2 for z/OS