IBM System i™

DB2 i5/OS Triggers

Presented by Jarek Miszczyk
DB2 for i5/OS Technology Team
IBM Rochester, MN USA

i want stress-free IT.
i want control.
i want an i.

What is a Trigger?

A program called when row(s) in a table are changed

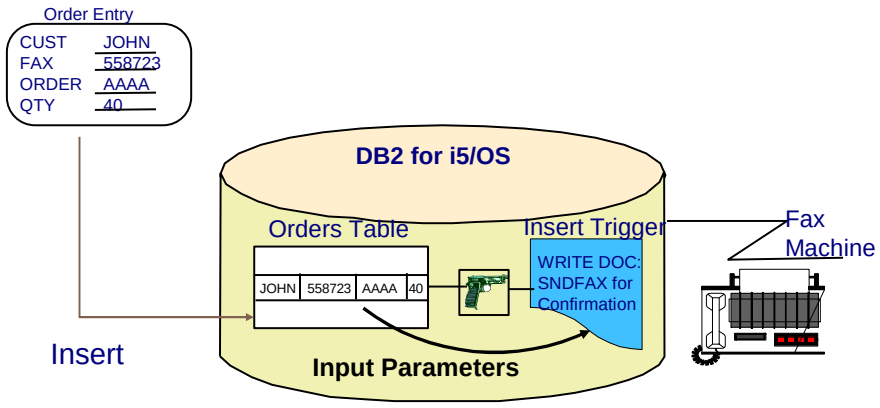
- Associated with a table (or view)
- Invoked automatically by DB2 before or after a (record) change to the table
- They become a property of the DB rather than an application responsibility
- Can be developed in any language of your choice
- Can interact with i5/OS resources

When do you need triggers?

- To consistently enforce complex business rules
- To interface with existing business routines
- To monitor critical tables
- In a client-server environment for performance

Move your business logic & processes into DB2

Triggers: An Example



- When a new order is inserted, a trigger starts
- Trigger gets information about order and customer
- A confirmation fax is automatically sent

Triggers types

Two types of triggers

- External triggers
  - Also known as ‘native’ or ‘system’ triggers
  - Two commands
    - ADDPFTRG
    - RMVPFTRG
  - External triggers can be added or removed from a file using the Database function in Operations Navigator
  - Row level trigger
- SQL Triggers
  - CREATE TRIGGER and DROP TRIGGER SQL statements
  - RMVPFTRG can also be used to remove trigger
  - Row, column and statement level triggers

i5/OS Support for Triggers

- A file can have up to 300 triggers
- Triggers for the same event are fired in the order created
- System-wide Trigger Catalog (SYSTRIGGERS)
  - Tracks both SQL and external triggers
- CHGPFTRG for Enabling & Disabling of Triggers
- No triggers allowed on catalog files or system tables

Triggers Concepts

- Base Table or View
  - Table object which the trigger is added to
  - Table containing LOB column only allowed for SQL triggers
- Trigger program
  - Program that is invoked at trigger fire time
    - User creates program for external trigger
  - Performs desired action
  - Must exist when trigger is added to the table
    - Created implicitly for SQL triggers
- Trigger "definition" is stored as a property of the table

Trigger events

Events for which triggers can be defined

- Record access
  - Insert
  - Update
  - Delete
  - Read
    - external triggers only
    - Avoid read altogether or use sparingly!
- Timing
  - Before
    - Trigger is fired before the record is modified
    - Valid for insert, update or delete
  - After
    - Valid for all record trigger types
  - Instead of
    - SQL trigger only
    - Specified against an SQL view rather than a table
    - Directs insert/update/delete to the trigger, no actual change to underlying table

External Triggers

Defining External Triggers

- Add or remove triggers by using CL commands:
  - ADDPFTRG

    ADDPFTRG FILE(MYFILE) TRGTIME(*BEFORE) TRGEVENT(*UPDATE) PGM(MYPROGRAM)
  - RMVPFTRG

    RMVPFTRG FILE(MYFILE) TRGTIME(*BEFORE) TRGEVENT(*UPDATE)

External Trigger Parameters

- **FILE** - Base table for trigger
  - File library - *LIBL, *CURLIB, name
  - File name - Name of base table for trigger
- **TRGTIME** - Trigger time
  - *BEFORE, *AFTER, *ALL (for RMVPFTRG only)
- **TRGEVENT** - Trigger event
  - *INSERT, *UPDATE, *DELETE, *READ, *ALL (for RMVPFTRG only)
  - Again, use *READ sparingly
- **PGM** - Trigger program to perform desired action
  - Program library - *LIBL, *CURLIB, name
  - Program name - Name of trigger program object
- **RPLTRG** - Replace trigger
  - *YES - Replace existing trigger
  - *NO - Do not replace existing trigger
- **ALWREPCHG** - Allow repeated changes to a row from within the trigger program
  - *YES - Repeated changes to a row ARE allowed
  - *NO - Repeated changes to a row are NOT allowed
- **TRGUPDCOND** - Trigger update condition. When is trigger fired on an update
  - *CHANGE – Do not fire trigger if no actual change to row's value
  - *ALWAYS - Always fire the trigger

External Trigger Program

- Can be written in any i5/OS high-level language (except Java)
- May or may not contain SQL
- Automatically called by DB2 and passed a set parameters
  - **Parameter 1:** Trigger Program Parameter List (Trigger Buffer)
    - variable length
    - contains info about the database change that is causing the trigger program to be fired
  - **Parameter 2:** Length of Trigger Program Parameter List
    - 4-byte binary value
  - **NOTE:** In general, this is an input-only parameter list. **BUT** it is possible for before insert, update triggers to change the new record portion of the parameter list

Trigger Buffer Layout

Seq	Data Type	Len	FieldDescription
1	Char	10	Physical file name
2	Char	10	Physical file Library
3	Char	10	Physical file member name
4	Char	1	Trigger event
5	Char	1	Trigger time
6	Char	1	Commit lock level
7	Char	3	Reserved
8	Binary	4	CCSID of data
9	Binary	4	Relative record number
10	Char	4	Reserved
11	Binary	4	Original record offset
12	Binary	4	Original record length
13	Binary	4	Original record null byte map offset
14	Binary	4	Original record null byte map length
15	Binary	4	New record offset
16	Binary	4	New record length
17	Binary	4	New record null byte map offset
18	Binary	4	New record null byte map length
19	Char	16	Reserved
20	Char	Var	Original record image
21	Char	Var	Original record null byte map
22	Char	Var	New record image
23	Char	Var	New record null byte map

Impact of ALWREPCHG on Trigger Program

- **ALWREPCHG(*NO)** - Prevent destructive data changes
  - CANNOT update record by updating the after image in the trigger buffer
  - CANNOT re-read and update record that caused the trigger to fire from within the trigger program itself
  - CANNOT update a record from another file more than once in a given trigger invocation
- **ALWREPCHG(*YES)** – allow data changes
  - CAN modify record by updating the new image in the trigger buffer
    - must be a BEFORE Insert or Update Trigger
    - Good for data corrections or transformations
  - CAN re-read and update record from within the trigger program itself
    - CAUTION - Recursive trigger program invocation!
    - Language must support recursion
    - Must be after Trigger, cannot update a record before it is written
  - CAN update a record from another file more than once in a given trigger invocation

ALWREPCHG(*YES) example

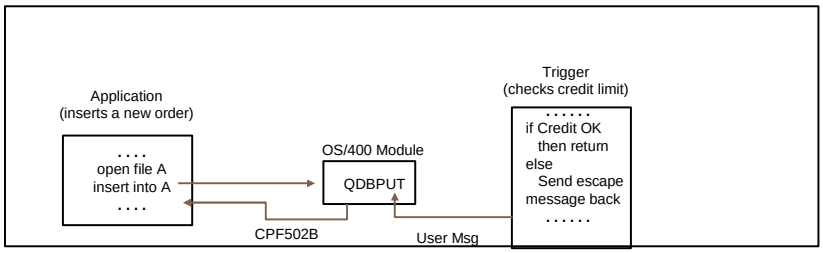
```
main(int argc, char* argv[])
{
long nOldValue; long nNewValue;
...
TrgBuffer = (Qdb_Trigger_Buffer_t *) argv[1];

NewRec=(cst_CUSTOMER_i_t * ) ((char *) TrgBuffer +
                          TrgBuffer->New_Record_Offset );
OldRec=(cst_CUSTOMER_i_t * ) ((char *) TrgBuffer +
                          TrgBuffer->Old_Record_Offset );
...
EXEC SQL DECLARE C1 CURSOR FOR
   SELECT nLastAutoNumber FROM MyLib/AUTONUMBER
     WHERE sTableName = 'MyTable';
EXEC SQL OPEN C1;
/* get the last value for this table */
EXEC SQL FETCH C1 INTO :nOldValue;

nNewValue = nOldValue + 1;
/* Update Trigger Buffer with new value */
NewRec->CUSTID = nNewValue;
...
/* record the new value in the autonumber table */
EXEC SQL UPDATE MyLib/AUTONUMBER
           SET nLastAutoNumber = :nNewValue
           WHERE CURRENT OF C1;
EXEC SQL CLOSE C1;
...
}
```

Interfacing Triggers and Applications

- How can a trigger notify a "logical failure" to the application?
- Example: trigger checks customers' credit limit



- You must signal an escape message back to make the I/O fail using the QMHSNDPM API
- The originating change will fail
- Application will get a CPF502B notify message
- The users' escape message will be found in applications message queue as a diagnostic message

SQL Triggers

SQL Trigger Components

- SQL Trigger Components
  1) Base table (or view)
  2) Trigger name
  3) Trigger event
  4) Trigger time
  5) Trigger granularity
  6) Trigger mode
  7) Transition variables
  8) Transition tables
  9) Triggered action

SQL Trigger Components

1) Base table (or view)
   - Physical file or table which the trigger is added to
   - Instead Of trigger applies only to views
2) Trigger name
   - Provides unique trigger identification within a library
     - Can be long name
3) Trigger event
   - The condition that causes the trigger to fire
   - Insert of a new row
   - Update of existing row
     - Row update
     - Per column in row
   - Delete of an existing row
4) Trigger time
   - When trigger program is to be run
   - Before or after the trigger event

SQL Trigger Components…

5) Trigger Granularity

- Row level triggers
  - FOR EACH ROW
    - Triggered action executed for each row satisfying trigger condition
    - If trigger condition never satisfied, triggered action never executed
- Statement level triggers
  - FOR EACH STATEMENT
    - Triggered action executed only once per statement, regardless of the number of rows processed
    - If trigger condition never satisfied, triggered action still executed once at end of statement processing
  - Not valid with Before triggers or Trigger Mode DB2ROW
- Column level triggers
  - Columns listed as part of UPDATE trigger event
    - UPDATE OF column_name_1, column_name_2,...
  - Only an update of a listed column causes trigger to fire
    - Fires even if the before/after value of column is the same
  - If no columns listed, update to **any** column causes trigger to fire

SQL Trigger Components…

6) Trigger Mode

- MODE DB2ROW
  - Trigger fires after each row operation
  - Only valid with Row level triggers
  - Exclusive function of DB2 UDB for iSeries
    - Not available in other DB2 UDB implementations
- MODE DB2SQL
  - Trigger fires after all row operations are complete
  - If specified on a row level trigger, triggered action executed N times after all row operations completed
    - N = number of rows processed
  - Not as efficient as DB2ROW since each row is processed twice
    - Temp table created for AFTER trigger to house transition records
  - Only valid with After triggers

SQL Trigger Components…

7) Transition Variables
- aka Correlation Variables
- Provides access to before and after image of record
  - similar to before and after image buffers for external triggers
- Qualifies column names for the before and/or after row images
  - OLD ROW - Before image of row
    - Update and delete triggers
  - NEW ROW - After image of row
    - Update and insert triggers

```
...REFERENCING OLD ROW AS oldrow REFERENCING NEW ROW AS newrow…
...IF newrow.salary > oldrow.salary + 10000...
```

- Not valid with FOR EACH STATEMENT level triggers

SQL Trigger Components…

8) Transition Tables
- Temporary tables that contains the before and/or after images of all affected rows
  - OLD TABLE - Before image of all affected rows
  - NEW TABLE - After image of all affected rows

```
REFERENCING OLD TABLE AS oldtbl
...(SELECT COUNT(*) FROM oldtbl)...
```

- Provides function similar to before and after images in trigger buffer for external triggers
  - Except there are potentially multiple rows to deal with!
- A single SQL statement can process multiple rows
- Not valid with Before triggers or Trigger Mode of DB2ROW

SQL Trigger Components…

9) Triggered Action
- Analogous to trigger program in external triggers
- Three parts
  - SET OPTION
    - Specifies the options that will be used to create the trigger
  - WHEN
    - Search condition or execution criteria for Trigger Body
    - Specifies when the SQL statements in Trigger Body will be executed
    - Not allowed for INSTEAD OF trigger
  - SQL Trigger Body
    - Single SQL statement
    - Multiple SQL statements delineated with BEGIN and END

Speaking of trigger body

- Some of the statement types available for the trigger body:
  - DECLARE local variables
  - SET local variables
  - IF, CASE
  - WHILE, FOR, REPEAT, LOOP
  - DECLARE CONDITION
  - DECLARE HANDLER
  - SIGNAL, RESIGNAL
  - GET DIAGNOSTICS
  - Provides access to SQLCA-like information
  - CALL
  - Call external procedures to access HLL pgms or OS/400 APIs

Row Level Trigger with Simple Trigger Body

```
CREATE TRIGGER audit_spending
AFTER UPDATE ON expenses
REFERENCING NEW ROW AS nw
FOR EACH ROW MODE DB2ROW
WHEN (nw.total_amount > 10000)
INSERT INTO travel_audit
VALUES(nw.empno, nw.deptno, nw.total_amount,
            nw.end_date);
```


Row Level Trigger with Complex Trigger Body

```
CREATE TRIGGER big_spenders
AFTER INSERT ON expenses
REFERENCING NEW ROW AS n
FOR EACH ROW
MODE DB2ROW
WHEN (n.totalamount > 10000)
BEGIN
  DECLARE emplname CHAR(30);
  SET emplname = (SELECT lname FROM employee
                  WHERE empid = n.empno);
  INSERT INTO travel_audit VALUES(n.empno, emplname,
            n.deptno, n.totalamount, n.enddate);
END
```

SQL trigger considerations

- Sql triggers will 'auto heal'
  - If pgm is lost, SQL triggers will be recreated
  - Makes save/restore less of a concern
- Column level trigger
  - What if the column is dropped (via ALTER)?
    - Cascade/restrict – prevent alter/trigger is dropped
  - What if column value is not really changed e.g. SET col1 = col1?
    - Trigger still fires
- INSTEAD OF trigger
  - Used in conjunction with views
  - Fires trigger but does not modify row in base table.
    - Intention is to notify trigger and let it decide how to handle the attempted change
- FOR EACH ROW + MODE DB2ROW matches native trigger firing

SQL trigger considerations…

- SET statement can be used by BEFORE Triggers to alter the new data
  - Generate missing values
  - Data cleansing
  - One of the *FEW* ways to change data in a Before trigger
    - INSERT, UPDATE, DELETE, etc not supported
    - New correlation variables can also be changed with SELECT INTO and as an output variable (OUT/INOUT) of a stored procedure
- SQL row trigger always fires on an update
  - Even if data did not change e.g. UPDATE… SET col1 = col1
  - Column level trigger, column must be target of update

Triggers vs. Constraints

Constraints

Database Constraints

- Referential Integrity Constraints - enforcement of business rules between related tables
  - EXAMPLE:

```
ALTER TABLE OrdHdr
  ADD CONSTRAINT OrderCustCst
  FOREIGN KEY (OrdCusNbr)
  REFERENCES customer(cusnbr)
 ON DELETE RESTRICT ON UPDATE RESTRICT;
```

- Check Constraints - enforcement of business values, valid domain
  - EXAMPLE:

```
ALTER TABLE Employee
 ADD CONSTRAINT SalaryChk
 CHECK(Salary<40000 AND Bonus<=Salary);
```

Triggers and Constraints

- Table can have both triggers and referential constraints
  - The referential constraint rule in effect will determine if the constraint is evaluated before or after the trigger program is run
- Restrictions for dependent tables
  - A **DELETE** trigger cannot co-exist with a referential constraint that has a delete rule of **CASCADE**
  - An **UPDATE** trigger cannot co-exist with a referential constraint that has a delete rule of **SET NULL or SET DEFAULT**
- Adding trigger vs. adding constraint
  - Adding a trigger does NOT validate existing records
  - Adding a constraint DOES validate existing records
- Constraint usually performs better than trigger when doing the same thing
  - Consider using a constraint when situation allows

Thank You

Additional Information

- DB2 System i home page – http://www.iseries.ibm.com/db2
- Newsgroups
  - USENET: comp.sys.ibm.as400.misc, comp.database.ibm-db2
  - iSeries Network (NEWS/400 Magazine) SQL & DB2 Forum – http://www.iseriesnetwork.com/Forums/main.cfm?CFApp=59
- Education Resources – Classroom & Online
  - http://www.iseries.ibm.com/db2/db2educ_m.htm
  - http://www.iseries.ibm.com/developer/education/ibo/index.html
- DB2 UDB for iSeries Publications
  - Online Manuals: http://www.iseries.ibm.com/db2/books.htm
  - Porting Help: http://www.iseries.ibm.com/developer/db2/porting.html
  - DB2 UDB for iSeries Redbooks (http://ibm.com/redbooks)
    - Stored Procedures & Triggers on DB2 UDB for iSeries (SG24-6503)
    - DB2 UDB for AS/400 Object Relational Support (SG24-5409)
    - SQL Query Engine
      - (http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg2456598.html)
  - SQL/400 Developer's Guide by Paul Conte & Mike Cravitz
    - http://iseriesnetwork.com/str/books/Uniquebook2.cfm?NextBook=183
- For additional questions or comments please contact
  - Software Service Provider
  - rchudb@us.ibm.com

Appendix:

General Trigger Considerations

Triggers and Commitment Control

- Trigger failure results depend on commitment control state

Application Program	Trigger Program	Originating Database Change	Trigger Database Changes
COMMIT = YES	COMMIT = YES	ROLLBACK	ROLLBACK
COMMIT = YES	COMMIT = NO	ROLLBACK	NO ROLLBACK
COMMIT = NO	COMMIT = YES	ROLLBACK if BEFORE Trigger	ROLLBACK if Activ Group Ends
COMMIT = NO	COMMIT = NO	ROLLBACK if BEFORE Trigger	NO ROLLBACK

- Using commitment control gives you protection against failures
- Triggers and applications *should* share commitment definition and isolation level when possible
- Triggers with SQL:
  - SET TRANSACTION sets the program lock level at run time
- Native triggers:
  - Open tables with or without commitment control at run time

Commitment Control: Recommendations

- Use commitment control both in triggers and in applications
- Create ILE triggers with ACTGRP (*CALLER)
  - They will always share the application commitment definition
- If triggers need a separate commitment definition:
  - COMMIT and ROLLBACK allowed in triggers
  - Always commit or rollback before exiting trigger
- In SQL triggers, set the same lock level as the application
- Remember: non-database changes will not be rolled back
  - Examples: Sending a fax or a message, updating a data area, writing in the spool...

Triggers and Journaling

- Information in Journal Entries to reflect "trigger related" changes

```
                    Display Journal Entry Details

Journal.............:   QSQJRN              Library.............:   ORDENTL
Sequence............:   137

Code................:   R  - Operation on specific record
Type................:   UB - Update, before-image

Object..............:   CUSTOMER           Library.............:   ORDENTL
Member..............:   CUSTOMER           Flag................:   1
Date................:   08/20/94           Time................:   21:57:05
Count/RRN...........:   1                  Program.............:   QRNXIO

Job.................:   008314/CHILANTI/MICKI
User profile........:   CHILANTI           Ref Constraint.....:   No
Commit cycle ID.....:   136                Trigger.............:   Yes
```

- Apply Journal Entry will not fire triggers
- Triggers changes should be journaled to the application journal
- Remember: non-database activity is not tracked by journals
  - Use the Send Journal Entry command or API to insert a user defined journal entry
  - May facilitate the recovery process by providing an audit trail

Activating Triggers

- Triggers execute under the job that activates them

```
                Request   Program or
Opt               Level   Procedure      Library     Statement   Instruction
                          QCMD           QSYS                        0351
                          QUICMENU       QSYS                        00C1
                      1   QUIMDRV        QSYS                        0455
                      2   QUIMGFLW       QSYS                        0483
                      3   QUICMD         QSYS                        03E4
                          QUOCPP         QPDA                        0541
                          QUOMAIN        QPDA                        0FDD
                      4   QUOCMD         QSYS                        0176
                          T4249CINS       ORDENTLIB    136           00D9
                          QSQROUTE       QSYS                        02F0
                          QSQINS         QSYS                        01C0
                          QDBPUT         QSYS                        0193
                          T4249RADT      ORDENTLIB    .GET           012D
```

- Triggers and application will share:
  - The library list
  - The QTEMP library

Finding Trigger Info for a Table

- Trigger information can be displayed with Display File Description Command (DSPFD)
  - DSPFD has a TYPE parameter
  - DSPFD FILE(DEPT-MSTR) TYPE(*TRG)
- Operations Navigator will also return trigger information by just right-clicking on the database table or physical file

Triggers and Object Management

- When you save a table, trigger definitions are saved as well
  - However, Trigger programs are NOT saved with the table
  - Easiest just to create the trigger programs in the same library as the table
- If you move, rename or delete a Trigger Pgm, the table description will not be updated
  - Remove and add the trigger definition again
- The Copy Library command will update triggers information in each table description
- Security: Triggers can be made to access objects users may not be authorized to:
  - Create triggers with USRPRF (*OWNER)

Performance Techniques

- Triggers invocations are similar to the overhead of an external call - be careful not to overuse
- Try to avoid full open for files (external triggers)
  - Use "soft exit" (RETRN in RPG, GOBACK in COBOL, return in C)
  - Avoid closing files as much as you can
  - Use the SHARE (*YES) in nested triggers
- Always avoid compiling with ACTGRP (*NEW)
- Handle exceptions in triggers running in their own Activation Group
- Code SQL triggers so that the system chooses a reusable ODP

Trademarks and Disclaimers