Pulling the (DB2) Trigger
(Without shooting yourself in the foot)

Introduction

The following article is intended to describe a methodology to allow for the addition of event trigger programs to the production database and to gradually build up the business validation rules for the database without the risk of interrupting the production processes of the company.

"Modernization" became the buzzword of the year in 2015 for long-standing IBM i applications (what we like to call "heritage"). Many diverse forces may cause a company to look at modernizing its core systems, especially the homegrown variety, and the topic covers many aspects of the look and function of the software. While the user interface has garnered the bulk of the attention for much of this century, often bringing the database into a more current model (SQL-compliant, normalized, etc.) provides more compelling outcomes and an even greater return on the investment.

One key outside driver for database modernization often involves the addition of a third party-application that interfaces with, and more importantly writes data to, the heritage database. For example, the warehouse adds a new web-application that tracks inventory based on barcode readers or radio-frequency identification (RFID) tags. The order entry portion of the heritage system still needs to know the on-hand inventory available for fulfillment, so somehow the inventory data must move from the new system into the heritage database.

This introduces a new dilemma; suddenly, this "interloper" renders moot all the data validation performed in the heritage inventory tracking code, as it bypasses these routines completely. An ETL program between the new and existing databases could replicate these same validations, but this introduces redundant code maintenance (which likely compounds an already existent problem for the development staff). If the new system writes directly to the existing tables, its configuration parameters may offer less-than-ideal control over whether it writes values that conform to existing strictures, introducing a threat to the integrity and quality of the data in the heritage database.

Self-Awareness

A simple and elegant solution to the above conundrum presents itself from the handbook of database modernization; identify and extract the business validation and data integrity rules from the heritage application code into trigger programs associated with the various tables (files). This pushes those business rules into the database itself (where many experts argue they belong) and makes the tables more "self-aware." The table polices itself and rejects any non-conforming values via its associated trigger.

Many fine articles and publications on how to build triggers exist, so this article only skims the surface. For more information, we suggest starting with IBM's online documentation on Creating Trigger Programs at http://publib.boulder.ibm.com/html/as400/v4r5/ic2979/info/db2/rbafomstrzahftrb.htm.

In this case, add a before-trigger to each table to handle the validation of the fields within the record for inserts and updates. The code for the validation rules comes directly from the heritage application, distilling all existing instances of the validations to ensure completeness. If any of the columns (fields) contain invalid values, the trigger program generates one or more diagnostic messages describing the individual issue(s) and then an escape message that prevents the system from actually writing the record.

While this solution prevents the "interloper" application from writing bad data, this addition unfortunately often also negatively affects the heritage application. Any system over fifteen or so years old, especially if written in RPG or COBOL, likely contains duplicate versions of the above validations, as more than one interface within the code offered the opportunity to update a given table (such as updating the customer address through both customer maintenance and order entry).

Despite best efforts (or, in all honestly, often due to a lack of effort in less formal times) these replicated validations contain differences and even contradictions. The new trigger program (which hopefully contains accurate validation routines for the various columns or fields) suddenly starts throwing errors not only for bad data from the external application, but for portions of the heritage code as well. This scenario makes many IBM i shops uneasy about adding triggers to the heritage database, both out of concern for the disruption to normal business operations, and also because it exposes a "dirty little secret" regarding IT.

Document, but Don't Deny

To avoid the negative impact on the heritage code (and keep the "secret" within IT); evolve how the trigger program behaves over time depending on where the triggering event originates. Using the Retrieve Call Stack (QWVRCSTK) API, the trigger identifies the program initiating the table operation. Initially, the trigger rejects bad data from outside applications (like the new inventory system), but understands the domain of the legacy application, usually via a short list of one or more libraries. Instead of the trigger throwing errors for any programs within the heritage domain, it writes the error condition (including the offending program name) to a log file.

The log file should contain sufficient information for a developer to find and correct the issue in the heritage code; for example, it might include the following elements:

- The date and time of the error
- The job's user profile
- The name of the trigger program
- The name and schema (library) of the table
- The name and library of the initiating program
- The diagnostic message number indicating the issue
- The original record (before) image
- The new record (after) image

This allows for correction of the heritage programs, including fixing inconsistencies and errors. The update should also include the necessary handler(s) to allow the heritage code to interact gracefully with the trigger. We also suggest removing redundant versions of the validation code (now the domain of the trigger program). The log also allows for tuning the trigger program, in case the initial set of validation rules imported overlooked a condition found in another version.

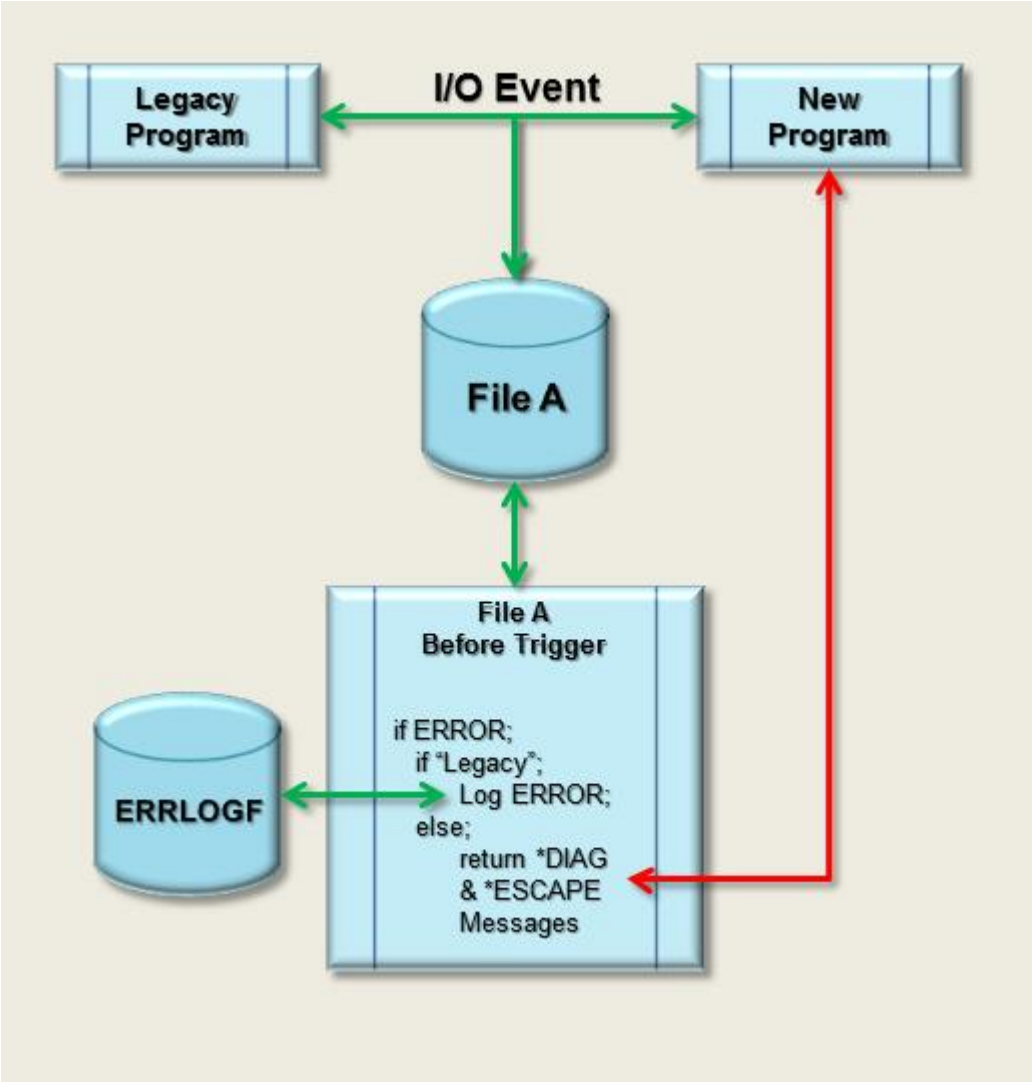
The trigger program may also include programmatic referential integrity checking to assess the status of the referential integrity between associated tables, allowing for necessary corrections and even possible identification of orphaned records. The trigger program can log any anomalies with sufficient detail for investigation and can even include code to repair error conditions in some instances. All of this occurs transparently during normal business operation with no impact to the status quo.

As the trigger logs the errors it finds, the development staff prioritizes the list of programs requiring updates and corrects them based on their importance and impact. They may even choose to leave low-impact programs alone until they require other maintenance.

As noted above, updates often include the removal of any now-redundant validation rules from the heritage code. This way the heritage application codebase grows smaller and now contains less redundant code. Validation and integrity checking generally accounts for as much as 60% to 80% of most heritage application code; not only do trigger programs protect data integrity more thoroughly, implementing them often makes maintenance simpler in the long run by consolidating the validation rules to a single instance.

As the development staff corrects the heritage code and enables it to handle the trigger errors, the trigger should begin returning errors to the corrected programs while continuing to log errors for the uncorrected programs. To avoid making changes to the trigger program for every corrected heritage program, add a "fixed" library to the front of the application library list (but not to the trigger program's list of libraries). This obviously assumes that the application does not use hard-coded library names in its program calls. Each corrected heritage program is moved into the "fixed" library and because that library does not appear in the application domain list in the trigger program, the trigger behaves the same as for an external program. If issues later arise, requiring additional code changes in the heritage program then move it back to the original library in the interim so that it does not disrupt business operations.

This process allows the trigger gradually to assume actual control for validations for the heritage application with minimal risk or impact. Likewise, once the trigger program stops logging any issues, the development staff should also convert any programmatic referential constraints into actual table constraints, thereby activating the full DB2 referential integrity functionality of the database.


Legacy Program
I/O Event
New Program
File A
File A
Before Trigger
if ERROR;
if "Legacy";
Log ERROR;
else;
return *DIAG
& *ESCAPE
Messages
ERRLOGF