Solving the Data Warehousing Dilemma with BBj's Newest DBMS Features

By Nick Decker

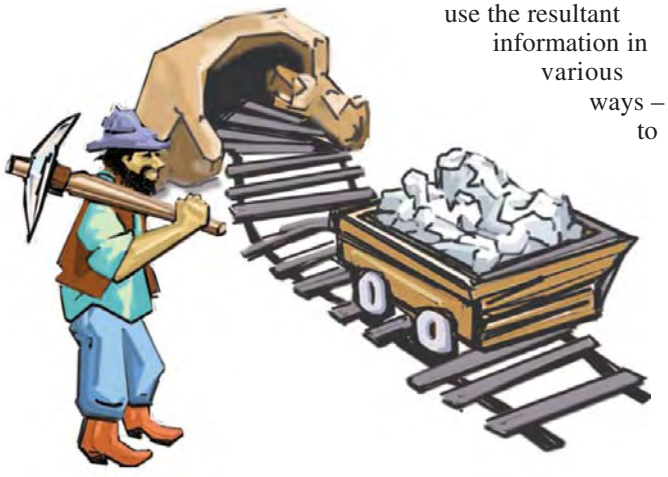
BASIS continues to add several groundbreaking features such as triggers and ESQL Tables to the BBj® database management system. Although some of these terms and concepts may not be new to many developers, the challenge always seems to be realizing the full potential of such features. Ironically, developers sometimes fail to exploit the most potent features fully because they are so versatile. Triggers fall squarely into this category – they are so incredibly useful that we continue to discover new ways to utilize them in working systems to solve age-old problems. One such problem that plagues many is the implementation of a standardized and efficient data warehousing system. A secondary problem is the choice of RDBMS to house the data. This article focuses on the nuances of data warehousing and how to go about solving the associated dilemmas.

Data Warehousing

Before digging into the details of applying triggers and ESQL's relational Tables, let us step back for a moment and look at the concept of data warehousing and its function. Simply put, a data warehouse is the main repository or storage of company data. This data is both current data used to maintain the normal day-to-day operations of the company as well as historical data accumulated over the life of the company. Both banks of data are critical to the life and function of the company. Without access to the current data, there is no way to process orders, invoice customers, or receive payments. Without access to the historical data, management would be unable to analyze past sales to determine profitability and whether the company is meeting its forecast goals.

Data Mining

In order to make use of the data at their disposal, companies perform "data mining." Similar to the miners digging deep into the earth in hopes of extracting a rare and valuable diamond, corporate analysts perform complex SQL queries and sift through mountains of data to retrieve valuable information. Analysts use the resultant information in various ways – to predict and forecast future sales, to see if recent business decisions such as whether a new advertising campaign has affected sales, and so on.



As necessary as data mining may be, it comes at a cost. The data retrieval and analysis usually places a heavy load on the system, sometimes slowing down daily operations to a crawl. Needless to say, this situation is impracticable and requires a solid solution. Companies have proven that data mining is indispensable, but how do they accommodate the load on the system? As with most complex problems, no simple solution is readily available. However, after a lengthy "tour of duty" in the trenches, engineers finally unearthed a standardized, multi-part structured solution to the data mining dilemma.

Solution – Part 1

The first step seems quite obvious – make an offline copy of the operational data system. At first blush, this seems to be a perfect solution. The data analysts can tie up the offline system running elaborate queries and the daily operation of the company will be completely unaffected. However, after implementing this step, engineers noted several outstanding problems. As usual, the solution is never this simple.

For starters, data retrieval from the offline system was still very slow, even though it was running on a separate system from operations. Analysts wrongly assumed that once they separated the historical data, their queries and data extraction would speed up tremendously. A closer look revealed the true nature of the slowness – non-normalized data. Most of the company's data was located in antiquated table formats. They found tables that did not

continued...

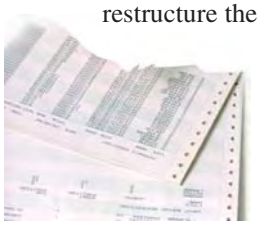
Nick Decker
Engineering Supervisor

make good use of indexes, formats that allowed for multiple data types in a single field, fields comprised of multiple data elements, and so on. Most modern-day database administrators would shudder at such a design, but the truth is that many administrators often ignored data normalization with little consequence – that is until SQL entered the picture.

SQL ushered in a new era of data independence. Armed with third party reporting tools such as Crystal Reports, all of the company's data was fair game. Users, even those without any database experience, were suddenly performing ad-hoc queries left and right. Since the software application design did not optimize the underlying tables for data retrieval from generic reporting software, the queries would sometimes take forever to complete. This was a great disappointment as everyone depended on the speedy execution of the fixed, specialized queries written into the company's software. The solution to this glaring problem was clear – normalize the data.

Solution – Part 2

Since the offline database was separate from the company's operational dataset, normalizing the data was not an insurmountable task. Even though the offline database was to contain the same data, it did not require the same data organization. Developers were free to restructure the data in a normalized format without having any impact on the operations software. After completing the normalization, analysts were thrilled to see their queries execute orders of magnitude faster. The celebration ended quickly, though, as they discovered their next problem: the offline database was out of date. They made no attempt to synchronize the offline version of the data with the version that operations used, so by now, the offline version was missing thousands of records and was no longer a complete and current set of data.



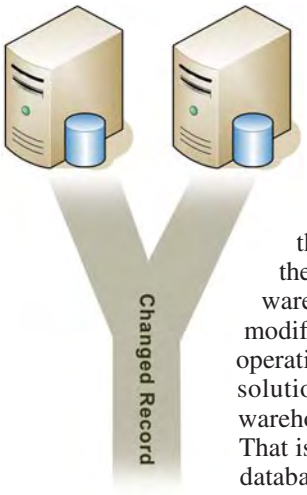
Solution – Part 3

Analysts easily recognized the next step in the solution – they had to keep the offline database in synch with the company's main dataset. They accomplished this by copying all of the data from operations to the data warehouse at regular intervals. However, this too proved to be problematic. Even though the data was *more* current than before, it still was not *actually* current. Once users got a taste of accessing data that was relatively current, it left them wanting more. A 24-hour lag on the operations database was surely an improvement, but soon everybody wanted real-time data and instant updates.

Additionally, the synchronization process turned out to be extraordinarily time consuming. As part of the synch process, nightly cron jobs kicked off programs that removed all of the records from the offline database. The next step involved painstakingly copying every single record from the operations database to the data warehouse. This process, while effective, was terribly inefficient because users do not modify a majority of the records in the database. Typically, they add new records and modify recent records, but most historical data remains unchanged on a day-to-day basis. The synch process never differentiated between modified and non-modified records, so it dutifully removed and recopied the same inactive data night after night. Not only did this take *a lot* of time, it ended up taking *most* of the time required to synch.

Solution – Part 4

Analysts solved this final hurdle in the data warehousing project by rethinking the synchronization process. Instead of blindly removing and copying every record in every file, they needed to synchronize in a more intelligent manner. The most effective way to perform the synch involved copying only the small percentage of records that changed instead of every individual record. Additionally, in order to satisfy the need for real-time updates, they would have to update the data warehouse immediately after a user modified or removed a record in the operations dataset. Simply put, the solution was to update the data warehouse on a transactional basis. That is, whenever the daily operations database changed, the same change should appear in the data warehouse.



Problem Solved!

This last tweaking to the system solved the problem for everybody. Operations continued unaffected by the offline data warehouse. Not only did developers solve their performance degradation problem by moving a copy of the data offline, but their legacy application did not require altering to accommodate new data structures and table layouts. Analysts were also pleased. They were able to quickly execute intricate queries and process an abundance of data on a live dataset.

Using BBj for Data Warehousing

Over the years, many BASIS developers have implemented their own data warehousing solution by keeping an offline copy of their corporate data. The typical solution closely follows our historical overview. The first step is to create a new relational and fully normalized database for offline storage of the data. The next step is to copy all of the operational data over to the normalized database on a nightly basis. The process, however, has always stopped there. The developers were stuck at the fourth part of the solution; they lacked real time updates to keep the data live and eliminate the costly bulk copies. With the advent of BBj's new database trigger capability, developers can now complete the final step to solve the data warehouse dilemma.

continued...

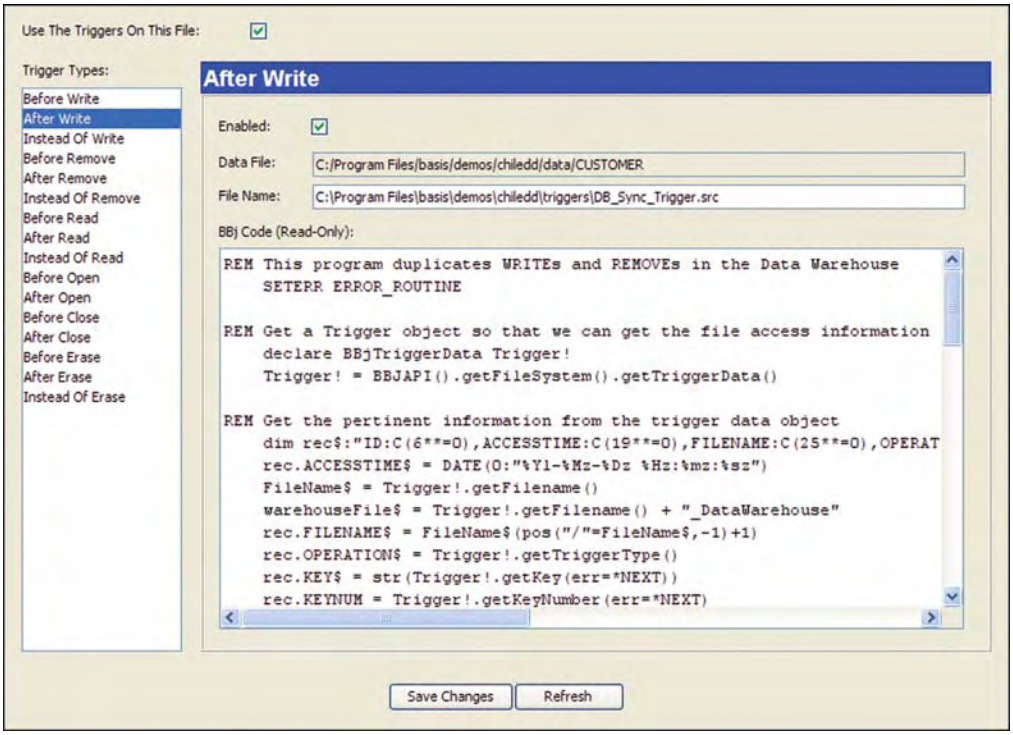


Figure 1. Trigger definition that synchronizes the data warehouse files in real time

Trigger Overview

Because triggers allow developers to execute a given program automatically whenever a particular file system activity occurs, triggers are perfect for keeping two databases in synch. As mentioned earlier, the goal is to update the data warehouse table the moment a user modifies or removes a record in the operations table. Triggers handle this with ease as the developer can configure the system to run a specialized synchronize program when one of these events occurs on the operational database. See **Figure 1.**

More specifically, the developer defines "After Write" and "After Remove" triggers to fire on an operations table. Now, whenever anyone modifies the table in any way, the respective trigger fires and executes the developer's synchronization code. This program instantiates a BBjTriggerData object and then retrieves pertinent information such as the name of the modified file, the type of file operation, the affected record, and so on. Armed with all of the essential details regarding the table access, the program would then make the corresponding change to the appropriate warehouse table; either making the same modification to the same record or removing the record as required.

Improving Data Access Time

Another new BBj database feature, ESQL Tables, further enhances the data warehousing solution. ESQL Tables offer many advantages, one of the most noteworthy is that they outperform SQL access to traditional **MKEYED** files. They also support standard SQL data types so on-the-fly data mapping for third party SQL reporting tools is a thing of the past. Additionally, they are field-based rather than record-based, so it is possible to read and write distinct portions of a data record without having to access the record in its entirety. Their support for variable-length records is also a tremendous bonus as it translates to enormous savings in disk space, record access, and reduced backup times. All of these features add up to a blazingly fast back-end database for the offline data warehouse, optimized for speed and space. It also provides fast and standardized access from an SQL-enabled application.

Summary

As the BASIS DBMS advances over time, new and exciting features such as triggers and relational ESQL Tables unearth numerous possibilities that were never available in the past. While a feature like database triggers may seem relatively simple at first glance, its straightforwardness can be deceptive. Triggers have dozens of potential use cases, from audit trails and access restriction to our most recent use case – real-time database synchronization.

By combining BBj's diverse triggers with its relational ESQL Tables, developers can implement an accelerated and powerful data warehousing solution. Triggers ensure that the various tables remain synchronized with one another and completely eliminate the need for bulk copies that tie the system up for hours overnight. Relational ESQL Tables ensure that analysts can sift through a plethora of information in the quickest way possible. By combining both of these new technologies, BASIS developers can finally solve the data warehousing dilemma!

Using Triggers to Maintain Database Integrity
www.basis.com/advantage/mag-v10n1/triggers.html

BASIS Puts Triggers to Work
www.basis.com/advantage/mag-v10n1/workingtriggers.html