The BASIS DBMS – New 5.0 Features

By Jeff Ash

As the newest BBx® generation, BBj®, continues to grow and evolve with enhancements and new features, so does the database management subsystem. This part of the system provides a way to generate queries and data update statements more easily, and enables access to data from several third party reporting and data access applications, including Crystal Reports, Microsoft Access, Microsoft Excel, and dynamic Web pages.

This article describes several of the new features added to the BASIS Database Management System in BBj 5.0.

Binary Data Support (Including BLOBs)

Sometimes it is desirable or necessary to store binary data, such as photographs of employees with their employee information, in each record of a database. In the past, it was common practice to use fixed-length CHARACTER (type C) fields to store such binary data. This is an adequate solution if all the values are the same length, but this approach does not work well with third party ODBC or JDBC applications. If the values can vary in length, the developer could define the field as variable length (either terminated or padded). The problem with this approach is that the binary data can contain any byte values, and therefore, might contain the terminator or pad character. This means that the database engine has no way of knowing whether the terminator is really the terminator or is actually a valid binary byte value. Alternatively, the developer could define the field as variable-length, terminated by the end of the record. This approach works for values up to 32767 bytes, the limit for character fields, but it can only be used once per record (for the last field).

What Is A BLOB?

BLOB stands for **B**inary **L**arge **OB**ject, a common data type supported by most popular databases around the world. Now, programmers can use the BLOB data type instead of a CHARACTER field to store binary data or large character strings (over 32767 bytes). The BLOB type provides a means for developers to store binary data in a way that ODBC and JDBC applications can understand. The problem with storing binary data as character data is that applications outside of BBj cannot determine if the data is actually string or binary. Many applications handle strings and binary data in different ways so it is important to have the ability to tell the third party application what type of data is contained in the field. For example, Microsoft Access will not correctly display binary data unless it is stored as a BLOB, due to the way that it interprets character and binary data. In fact, it can often cause incorrect search results or empty record sets. Another benefit of the BLOB data type compared to CHARACTER data is that BLOB values can be as large as 2 GB (2^{31}-1 bytes).

BLOB Example

A good way to understand how BLOBs work is to look at a simple example. This example shows how to create a table with a BLOB field using SQL:

```
CREATE TABLE mytable
      (ID INTEGER PRIMARY KEY, my_blob LONG VARBINARY(2000))
```

Most third party applications see this field as a BLOB value, and can access it automatically through an ODBC or JDBC driver. BBj applications use a new template type to designate the type of data in the record for that field. Template type "O" (object) indicates variable-length binary data. For example, to access the records written into the table created in the example above, use the following code:

```
DIM MYREC$:"ID:I(4),MY_BLOB:O(2000)"
READ RECORD(mychan)MYREC$
MY_BIN_BUF$ = MYREC.MY_BLOB$
```

Important Notes Regarding the Use of BLOBs

Before including BLOBs in a database, keep in mind the following:

1. Do not use BLOBs as any part of a key or index. The file system will allow it, but the results are unpredictable. The native data format for BLOB values is not sortable, which means that iterating on a key with a BLOB value will not come back in any particular order.
2. Always use BLOB for storing binary data of any size if a third party ODBC or JDBC application will access the data.
3. Allow for length_of_BLOB + 4 bytes for each BLOB field in a record. A BLOB consists of a 4-byte length indicator plus the bytes.
4. Make the BLOB only as big as necessary. Most BBj file types, with the exception of JKEYED files, do not support true variable-length records. Each record in the file uses the maximum record length, even if the BLOB data only takes up a portion of the record.

GET TABLE INFO

It is often necessary or desirable for an application to acquire the string template that defines the records in a database table. In addition, it is sometimes useful to know the full path to the data file for a particular table. This is a need specific to BBj applications since these applications often utilize a combination of direct file access and SQL operations.

To make this task easy for the developer, the BBj SQL engine now provides the GET TABLE INFO command. The BBj SQL engine treats this command like any other SQL statement, but since it is not standard SQL, non-BASIS databases will not recognize it. This command returns a result set that contains the string template for the specified table and the full path to the table's data file. Using this method to locate a data file and template means that the developer does not need to include template definitions in the application itself, but can retrieve them at runtime. The benefit of their dynamic access

continued...

Jeff Ash
Software Engineer

to the template at run time is to greatly reduce the application maintenance effort associated with any template changes.

A single GET TABLE INFO statement returns one record. After executing a GET TABLE INFO statement, SQLTMPL(*sqlchannel*) returns this string template:

```
FILENAME:C(32767*=10),TEMPLATE:C(32767*=10)
```

A BBj application can use the GET TABLE INFO statement to retrieve a table's template and file information as shown in this example:

```
MYCHAN=SQLUNT
SQLOPEN(MYCHAN)"mydatabase"
SQLPREP(MYCHAN)"GET TABLE INFO FOR mytable"
SQLEXEC(MYCHAN)
DIM REC$:SQLTMPL(MYCHAN)
REC$=SQLFETCH(MYCHAN)
FILENAME$=REC.FILENAME$
TEMPLATE$=REC.TEMPLATE$
```

Analyze This!

For the BBj SQL engine to best optimize queries, it needs to know certain information about the records in the tables involved. BBj addresses this need with a feature called *database analysis*. Administrators perform this in the BBj Enterprise Manager when the database is first set up, and then whenever the structure of the data changes significantly.

During this analysis, the BBj SQL engine determines the average number of distinct values for various numbers of segments of a particular key. At runtime, the SQL engine uses this information to determine which key to use for iteration. For example, assume there is a key on the LAST_NAME column in a table and one on the STATE column. Now assume that all people in the table live in Ne Mexico, Colorado, or Texas. Also, assume that most of the people not have the same last name. If the SQL engine knows that there a more distinct values in the LAST_NAME column than there are in the STATE column, it can conclude that searching on the LAST_NAME key is probably going to require that it read fewer records. If the SQL engine does not know that LAST_NAME is m distinct than STATE, it has no way of knowing which key is more efficient for searching.

Performing Analysis Programmatically

While administrators typically perform table and database analysis through the BBj Enterprise Manager, the developer can also do this programmatically. BBj 5.0 introduces two SQL statements to make t possible: ANALYZE TABLE and ANALYZE DATABASE.

Analyze a Single Table at a Time

The ANALYZE TABLE command analyzes a single table. The number of keys on a table and the number of records in the table determines how long this analysis process takes. A large number of keys and records may result in this analysis taking a very long time to complete. Here is an example of analyzing a table:

```
MYCHAN=SQLUNT
SQLOPEN(MYCHAN)"mydatabase"
SQLPREP(MYCHAN)"ANALYZE TABLE mytable"
SQLEXEC(MYCHAN)
```

Analyze the Entire Database

Use the ANALYZE DATABASE command to analyze all of the tables in the database. Here is an example of analyzing the entire database:

```
MYCHAN=SQLUNT
SQLOPEN(MYCHAN)"mydatabase"
SQLPREP(MYCHAN)"ANALYZE DATABASE"
SQLEXEC(MYCHAN)
```

When to Use ANALYZE

It is important to consider the differences between the two ANALYZE commands.

ANALYZE DATABASE analyzes all of the tables in the database at one time, and can be time consuming. This entire analysis has the potential to fail if an error occurs in one of the tables. If an error occurs, BBj returns an error message that names each of the problematic tables.

ANALYZE TABLE analyzes only one specified table. An application can run ANALYZE TABLE on each table, one at a time. After the analysis of each table, the application can check the status and provide the user with a progress indicator or individual error reports.

System Level File Locking

Previous versions of BBj always opened data files exclusively, meaning no other application could open the file as long as BBj held it open. As of BBj 5.x, the administrator has two options for configuring file locking on a BBj Services installation; "exclusive" or "shared" locking.

Configure file locking for exclusive or shared access

Exclusive Locking

Exclusive locking prevents all file access except by the BBj Services that opened the file. This means that no other application, not even another BBj Services, can open the file. Locking it in this way optimizes performance since BBj knows no one else can change the file. As a result, the server is more efficient when it reads records or opens additional handles on the same file – it knows exactly what has changed.

If two or more servers or BASIS products (i.e. PRO/5®
and BBj) need access to the same data file at the same time, then both parties must route their requests through one of two paths: a PRO/5 Data Server® or a BBj PRO/5 Data Server. This is not a difficult task, but it introduces additional configuration and maintenance issues. Follow the necessary steps to configure either BASIS Data Server.

continued...

PRO/5 Data Server

1. Install and setup a PRO/5 Data Server.
2. Change the path or prefixes in each environment to point to the PRO/5 Data Server instead of directly to the files.
3. Verify the configuration sets the PRO/5 Data Server to receive requests from all involved parties.

BBj PRO/5 Data Server

1. Setup and enable a BBj PRO/5 Data Server using the BBj Enterprise Manager.
2. Verify all PRO/5 client versions are 5.0 or higher.
3. Change the path or prefixes in each environment to point to the BBj PRO/5 Data Server instead of directly to the files.

Shared Locking

Shared locking eliminates the issues mentioned above. When using shared locking, BBj Services utilizes the PRO/5 method for locking files. This means that multiple installations of BBj in combination with multiple PRO/5 clients can access the same data files at the same time.

However, before changing your BBj Services to use shared locking, it is important to understand that it comes with a performance cost. The cost is that it takes more work for BBj Services to allow multiple sources to access files at the same time.

Each installation of BBj Services has its own global setting for shared or exclusive locking setting. It is not possible to use both shared and exclusive locking at the same time.

To enable shared locking:

1. Log in to BBj Services through the BBj Enterprise Manager.
2. Double-click on the "BBj Environment" node in the tree.
3. Select the check box labeled "Use Shared File Access."

Summary

The changes in BBj 5.0's BASIS Database Management System deliver two significant improvements: better interoperability with third party ODBC/JDBC applications, and easier, more flexible database administration. Using BLOBs, the developer can embed large binary objects right in the database for easy retrieval via READ RECORD operations or by using SQL from BBj or a third party ODBC/JDBC application. With an improved method for retrieving templates from tables in a database, applications can more dynamically adjust to changes made to record layouts. Now that applications can run a performance analysis programmatically on a database, it is possible to update this information automatically in an end user's database directly from within the program itself. Finally, the introduction of shared-locking in BBj allows single or multiple PRO/5 and BBj installations to share simultaneous access to the same data files, if necessary. All of these new features add another level of significant power and flexibility to the BBj product.