BASIS International Offers a Choice of Databases

By Nico Spence and Jeff Ash

BASIS provides a proven, robust, reliable, and scalable filesystem for the most demanding business applications. However, today's organizations use many different databases, requiring application developers to write database-independent applications. Furthermore, many application developers are considering the option of connecting to third-party databases to handle issues, such as data duplication, integration to other disparate systems, and to extend the functionality of their existing applications. BASIS's TechCon2003 demonstrated BASIS programs interacting with data from various back-end databases such as Oracle®, SQL Server®, and Access®.

BASIS provides the SQL language syntax and ODBC/JDBC for PRO/5®, Visual PRO/5® and BBj® to support database independence. With its Java interoperability, BBj provides a new way to achieve application database-independence. The BBj Filesystem Plugin gives the developer the option to create database independence with minimal code changes to existing applications. This article discusses these two alternative ways to make an application database-independent and what performance overhead, if any, is associated with such a change.

The first database alternative involves syntax changes to a BASIS program, incorporating SQL into an existing application. This is illustrated in the syntax example below:

Previous Syntax:

```
 DIM SALESREP$:"EMPLOYEE_ID:C(3*=10),NAME:C(30*=10),ADDRESS:C(30*=10),"+
 :
"ADDRESS2:C(30*=10),CITY:C(20*=10),STATE:C(2*=10),ZIP:C(10*=10)"+
 :              ",PHONE:(15*=10),COMM_TYPE:C(1*=10)"
 SALESREPCHAN = UNT
 OPEN (SALESREPCHAN) "EMPLOYEE"
 READ RECORD (SALESREPCHAN) SALESREP$
```

SQL Syntax:

```
 SALESREPCHAN = SQLUNT
 SQLOPEN (SALESREPCHAN) "EMPLOYEE"

 SQLPREP (SALESREPCHAN) "SELECT employee_id,name FROM employee
ORDER BY employee_id"
 SQLEXEC (SALESREPCHAN)

 DIM SALESREP$:SQLTMPL (SALESREPCHAN)

 SALESREP$ = SQLFETCH (SALESREPCHAN)
```

The back-end database could be a BASIS Filesystem or a third party relational database, such as Oracle, SQL Server, or MySQL. The BASIS program uses ODBC/JDBC connectivity to access the database.

Implementing this solution involves learning the new SQL syntax, normalizing the underlying database, and making code changes in the application software. After implementing this solution, the application becomes truly database-independent, giving developers and end users a choice of ODBC/JDBC-enabled databases.

The second database alternative, using the Filesystem Plugin, requires almost no code change to the existing application. The only change is to the OPEN verb, which makes this option a very simple retrofit to old code. Developers can normalize data structures without changing the existing application. The Filesystem Plugin is a white-box implementation, and BASIS provides the ability to re-route standard filesystem operations, such as READ RECORD to external Java code. This means that an experienced Java programmer must code the equivalent SQL syntax to emulate the BASIS syntax for filesystem operations.

The following example demonstrates this process using the Chile Company Database:

1. Insert a new entry in the **`config.bbx`** for the file type:

   ```
   ALIAS J1CUST chileco.plugin.CustomerOpenPlugin "jdbc:odbc:ChileCo"
   ```

2. Change the existing applications OPEN statement to point at the plugin:

   ```
   OPEN (customerChan) "J1CUST"
   ```

3. This executes the application programmer's customized white box Java code, which is equivalent to the OPEN verb:

   ```
   ...
   try
   (
       m_connection = DriverManager.getConnection(P_url);
   )
   catch (SQLException sql)
   (
      throw new FilesystemException(
         FilesystemException.NOF,
         "Unable to connect to database: " + sql.getMessage());
   )
   ...
   ```

4. Thereafter, the applications filesystem logic is re-routed to execute the applicable Java white-box code:

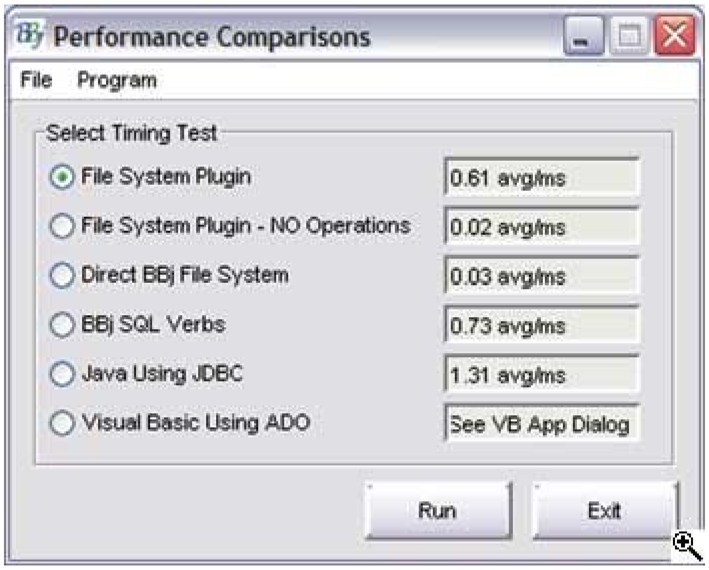
   ```
   ...
   private static final String BASE_SELECT =
   "SELECT c.cust_num, c.first_name, c.last_name, c.company, " +
      "c.address1, c.address2, c.zip_code, z.city, z.state, c.country, " +
      "c.phone1 FROM customer c, zip z WHERE c.zip_code = z.zip_code ";
   ...
   ```

In addition, the developer can rewrite the plugin to manage more than one "file." For example, rewriting the plugin above to handle the CUSTOMER and EMPLOYEE information makes the setup of the application much cleaner because there are fewer plugins to manage. When a plug-in manages more than one "file," it relies on information from the MODE= portion of the OPEN call. The example above would change to:

```
OPEN(customerChan, mode="FILE=CUSTOMER")"J1CUST"
```

Or

```
OPEN(customerChan, mode="FILE=EMPLOYEE")"J1CUST"
```



Performance Comparisons

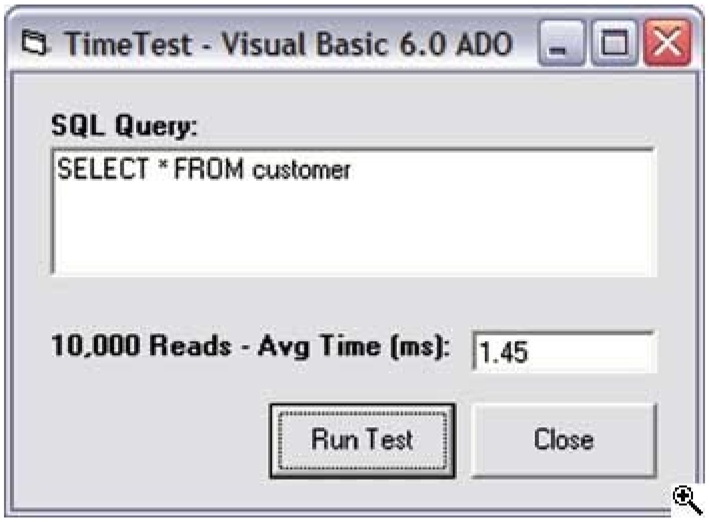
File Program

Select Timing Test

(●) File System Plugin	0.61 avg/ms
( ) File System Plugin - NO Operations	0.02 avg/ms
( ) Direct BBj File System	0.03 avg/ms
( ) BBj SQL Verbs	0.73 avg/ms
( ) Java Using JDBC	1.31 avg/ms
( ) Visual Basic Using ADO	See VB App Dialog

Run Exit

Figure 1. Performance comparison results.



TimeTest - Visual Basic 6.0 ADO

SQL Query:

SELECT * FROM customer

10,000 Reads - Avg Time (ms): 1.45

Run Test Close

Figure 2. Visual BASIC time performance test.

The plugin receives a **`java.util.Map`** of name/value pairs passed in the MODE= string, and then the plugin determines how to use the information. The list of name/value pairs has no limit.

The Q2-3 Advantage CD includes this example. While this example does not attempt to show the normalization process without changing the existing application, it does demonstrate that the BASIS application code has a different logical view of the customer data than the relational database. The plugin has combined two relational database tables using a common field to present the logical view that the application expects.

Performance Issues

The enclosed CD contains performance comparisons of the data access options discussed in this article. The CD also contains a number of performance tests. **Figure 1** and **Figure 2** illustrate results of these performance tests, carried out on an internal BASIS notebook, with 10,000 iterations per method.

As one would expect, the native BASIS Filesystem delivers the fastest performance. The second option, "Filesystem Plugin - NO Operations" demonstrates that the

Filesystem Plugin itself does not add any significant overhead. The comparison of the white-box Java Code performance to other data access methods demonstrates the insignificant overhead of using the Filesystem Plugin. BBj SQL engine access is comparable to the BBj Filesystem Plugin. In all cases, file access performance is mainly dependent on database design and implementation.

In conclusion, one can continue to use the BASIS Filesystem for the most efficient data processing. Alternatively, developers can implement SQL syntax in the BASIS application and access the BASIS Filesystem or any third-party database with ODBC/JDBC access. Finally, with hardly any change to existing BASIS programs, one may implement the BBj Filesystem Plugin and write Java code to implement access to third-party databases.

Click HERE for the "BASIS International Offers a Choice of Databases" source code.

With BASIS - the Choice of Database is Yours!