Paper #3016
Application Development with ODBC and IMAGE/SQL

Kriss Rant
Hewlett-Packard Company
Commercial Systems Division
Cupertino, CA
(408) 447-6653

INTRODUCTION

Recent support for Microsoft ODBC (Open Database Connectivity) in HP IMAGE/SQL and ALLBASE/SQL enables database access from many third-party Microsoft Windows applications and tools already available on the desktop such as Microsoft Excel and Lotus 1-2-3.

ODBC is an industry-standard API (Application Programming Interface) that enables PC applications and tools to access Relational Database Management Systems (RDBMS) such as IMAGE/SQL and ALLBASE/SQL in a client/server environment using SQL (Structured Query Language). SQL is the industry-standard language of relational databases.

Decision support tools such as Excel and Lotus 1-2-3 require no SQL knowledge and allow users direct access to corporate data for ad-hoc types of queries without any programming assistance. The user simply utilizes the application GUI (Graphical User Interface) to describe what data to retrieve and the application then creates the appropriate SQL statement to retrieve the data on the user's behalf. On the other hand, programmer's who wish to take advantage of the simplicity of the GUI to develop client/server applications using application development tools such as Microsoft Visual Basic, PowerSoft PowerBuilder, or Gupta SQLWindows require a basic knowledge of SQL.

This paper will describe SQL, and how it can be exploited to quickly develop client/server applications with IMAGE/SQL and ALLBASE/SQL. This objective will be accomplished by presenting simple solutions to complex business problems. Examples will be shown using Visual Basic. In addition, advanced application development SQL features like stored procedures and business rules will be emphasized.

IMAGE/SQL and ALLBASE/SQL

There has been a lot of discussion in the HP 3000 community about the difference between IMAGE/SQL and ALLBASE/SQL.

IMAGE/SQL is Hewlett-Packard's popular TurboIMAGE Database Management System (DBMS) recently enhanced with an SQL interface that enables relational access (read and write) to TurboIMAGE data on the HP 3000. It gives you the benefits of investment protection, meaning you can continue to run your existing TurboIMAGE applications unchanged, as well as access to many new applications and tools that are based on SQL. ALLBASE/SQL, on the other hand, is Hewlett-Packard's open, industry-standard Relational Database Management System (RDBMS) for the HP 3000 and HP 9000. It is based on ANSI SQL, and supports advanced relational features such as integrity constraints, BLOBS (Binary Large Objects), and stored procedures. Integrity constraints allow you to define data validity checks directly in the database rather than coding them in the application. As we move into the world of multimedia, the ability to store images and sound in the

database becomes more important. BLOB data types allow you to accomplish just that. Stored procedures which are available both with ALLBASE/SQL and IMAGE/SQL will be discussed later in this paper.

IMAGE/SQL is an evolutionary product that allows you to move to relational technology at your own pace. It is based on the ALLBASE/SQL SQL interface, which provides transparent data access to both relational databases. In fact, the full ALLBASE/SQL product, which includes ODBC, is included as a component of IMAGE/SQL; however, IMAGE/SQL customers are limited to storing 12 megabytes of data in a native ALLBASE/SQL database.

MANIPULATING DATA

A relational database is a logical collection of data arranged into tables, also known as relations. Tables are equivalent to datasets in TurboIMAGE. Tables are composed of rows (or tuples) and columns, which are equivalent to records and fields in TurboIMAGE.

A relational database is accessed using SQL. SQL is a powerful language that enables flexible access to your data. SQL allows you to retrieve data using a SELECT statement, and to modify data using INSERT, UPDATE, or DELETE statements. These general purpose SQL statements are referred to as the Data Manipulation Language (DML), and they form the basis for developing relational applications.

Let's explore each of these SQL statements in depth.

Selecting Data

The SELECT statement is the most basic SQL statement. The SELECT defines what data you wish to retrieve from the database. This is usually referred to as a query.

The SELECT statement occurs in many variations; however, in its simplest form it is composed of the following items:

1) Column names
2) Table names
3) WHERE clause

The WHERE clause is used to define the search condition. The search condition is used to determine which rows in the table are to be retrieved. If the WHERE clause is NOT present, then all rows will qualify for retrieval. The column names are used to define which column values to return, and the table names are used to define which tables the data is to be retrieved from. If there are two or more tables in the table list, then the query is referred to as a Join.

Let's look at an example using the PARTSDBE sample database. Suppose we wish to find all vendors who reside in California. The example in Figure 1 shows us how this can be accomplished.

Note, the PARTSDBE database, which is Hewlett-Packard's ALLBASE/SQL demonstration database, will be used for illustration purposes throughout this paper; however, similar examples can be created using any IMAGE/SQL database. The PARTSDBE can be created using a script called SQLSETUP.SAMPLEDB.SYS, which is included with IMAGE/SQL.

```
SELECT VENDORNUMBER, VENDORNAME, VENDORSTATE, VENDORREMARKS
        FROM PURCHDB.VENDORS
        WHERE VENDORSTATE = 'CA'
```

Figure 1. SELECT Example

The data is returned in a tabular format of rows and columns, and is called the query result. To return values for all of the columns the * symbol can be substituted for the column list in Figure 1 above.

Inserting Data

The INSERT statement is used to add rows to a table. The INSERT statement requires the following information:

1) Table name
2) Column names
3) VALUES clause

The VALUES clause is used to assign values to each of the column names in the list. There is a one-to-one correspondence between each of the values in the VALUES clause and each of the column names. For example, the first value in the VALUES clause is assigned to the first column name in the list of columns and the second value is assigned to the second column name and so on. If the list of column names is not present, then the values are assigned to the column names in the exact order that the columns are defined in the table. It is recommended that the column list always be used for ease of application maintenance.

Let's look at an example. Suppose we wish to start doing business with vendor A1 Disk Drives. The SQL statement in Figure 2 below will add the vendor to the Vendor table with a Vendor Number of 7777.

All column names that are not listed in the example are assigned default values. For a list of the IMAGE/SQL default data types, see Table 2-5 in the HP IMAGE/SQL Administration Guide. For ALLBASE/SQL tables, columns that do not have a value provided on INSERT are assigned NULL unless a default is defined for the column.

```
INSERT INTO PURCHDB.VENDORS
        (VENDORNUMBER, VENDORNAME, VENDORSTREET,
         VENDORCITY, VENDORSTATE, VENDORZIPCODE)
        VALUES (7777, 'A1 Disk Drives', '123 Silicon Way',
                'Orchard City', 'CA', '99999')
```

Figure 2. INSERT Example

Updating Data

The UPDATE statement is used to modify one or more columns in a table. The UPDATE statement is composed of the following items:

1) Table name
2) SET clause
3) WHERE clause

The WHERE clause is used to define the search condition. The search condition is used to determine which rows in the table are to be updated. If the WHERE clause is NOT present, then all rows will qualify for update. The SET clause is used to assign new values to specific columns for rows that are qualified by the WHERE clause.

For example, suppose we wish to assign real values to the columns that were assigned default values for A1 Disk Drives in Figure 2 above. The SQL statement in Figure 3 will update these columns.

```
UPDATE PURCHDB.VENDORS
        SET CONTACTNAME       = 'John Smith',
            PHONENUMBER       = '408 255 9999',
            VENDORREMARKS     = '10% discount on all products'
        WHERE VENDORNUMBER = 7777
```

Figure 3. UPDATE Example

Deleting Data

The DELETE statement is used to remove rows from a table. The following items compose the DELETE statement:

1) Table name
2) WHERE clause

The WHERE clause is used to qualify which rows are to be deleted. For example, suppose you wish to stop doing business with vendor 7777 due to the lack of demand for their components. Assuming that the vendor does not currently exist on any purchase orders, the SQL statement in Figure 4 will remove the vendor from the VENDOR table.

```
DELETE FROM PURCHDB.VENDORS
        WHERE VENDORNUMBER = 7777
```

Figure 4. DELETE Example

The WHERE clause is optional. If you wish to remove all vendors from the VENDOR table, simply omit the WHERE clause.

VISUAL BASIC, ODBC, and SQL

Let's take a look at how the SQL Data Manipulation Language can be used to access data in an IMAGE/SQL or ALLBASE/SQL database using a PC application development tool like Visual Basic.

Visual Basic supports the SQL Data Manipulation Language via three methods: Data Controls, SQL PassThrough, and ODBC API calls. The Data Controls, which utilize the Microsoft Jet Engine, provide the highest level of data access with minimal programming effort. The Jet Engine, which is built on top of the ODBC Interface, hides the complexity of the ODBC API. Jet is included with Visual Basic. The second method is SQL PassThrough. SQL PassThrough offers many of the benefits of Jet; however, the programmer has the option to eliminate the overhead associated with processing SQL statements by Jet and send them directly to the ODBC Interface. The programmer, who must create the SQL statements, has more control over the database access as well. The last method is the ODBC API calls. The ODBC API provides the lowest level of access and gives the programmer complete control over database access.

The most common method used for database access is SQL PassThrough. There is a lot of overhead associated with the Data Controls, and hence, performance issues. The ODBC API calls offer the highest level of performance at the expense of a complex programming interface.

Visual Basic supports SQL PassThrough via the CreateDynaset and ExecuteSQL Methods. The CreateDynaset Method is used to retrieve data via a SELECT statement created by a programmer. Visual Basic creates what is called a Dynaset to save the query result. Like a table, a Dynaset has rows (records) and columns (fields). The ExecuteSQL Method is only valid for SQL statements that do not return records. Examples of these statements are INSERT, UPDATE, and DELETE. ExecuteSQL returns the number of rows affected by the SQL statement.

Figure 5 below demonstrates how to use the CreateDynaset Method, and Figure 6 demonstrates how to use the ExecuteSQL Method. These examples were derived from a Visual Basic example called PARTSD.ZIP that was downloaded from CompuServe. The file is located in the HPSYS Forum under the MPE Library.

Note, the ODBC default for transaction control is SQL AutoCommit "ON", which means each SQL statement is automatically committed to the database. At the present time Visual Basic offers no method to turn this option off unless the ODBC API calls are utilized.

```
'This example uses an SQL statement to create a Dynaset of Vendors from California.

Dim MyDB As Database, MySet As Dynaset
Dim SQLStmt As String
Const DB_SQLPASSTHROUGH = 64      ' Set constant.

' Open PARTSDBE as an ODBC database.  PARTSDBE must be defined
' as a data source in the ODBC.INI file.
Set MyDB = OpenDatabase("", False, False, "ODBC; DSN=PARTSDBE; UID=")

' Prepare the SQL statement.
SQLStmt = "SELECT * FROM PURCHDB.VENDORS WHERE VENDORSTATE = 'CA'"

' Create the new Dynaset.
Set MySet = MyDB.CreateDynaset(SQLStmt, DB_SQLPASSTHROUGH)
```

Figure 5. CreateDynaset Method Example

```
'This example executes an action query that changes the Phone Number for Vendor 7777 in
'the Vendors table.

Dim MyDB As Database
Dim SQLStmt As String
Dim RecsChanged As Integer

' Open PARTSDBE as an ODBC database.
Set MyDB = OpenDatabase("", False, False, "ODBC; DSN=PARTSDBE; UID=")

' Prepare the SQL statement.
SQLStmt = "UPDATE PURCHDB.VENDORS SET PHONENUMBER = '408 255 8888'"
SQLStmt = SQLStmt + "WHERE VENDORNUMBER = 7777"

' Execute the SQL statement.
RecsChanged = MyDB.ExecuteSQL(SQLStmt)
```

Figure 6. ExecuteSQL Method Example

STORED PROCEDURES AND BUSINESS RULES

In addition to supporting a data manipulation language, IMAGE/SQL and ALLBASE/SQL allow you to create database objects such as stored procedures and business rules, which reduce the need for extensive application programming in addition to providing a means to improve application performance in a client/server environment by reducing network traffic.

Procedures define sequences of SQL statements that can be stored in an IMAGE/SQL database and invoked as a group directly by an application program or indirectly through rules. Rules allow you to

define relationships among tables by tying procedures to specific kinds of data manipulation on tables.

Creating Stored Procedures

Procedures can include many of the operations available within an application program. For example, you can use local variables, issue most SQL statements, create looping and control structures, test error conditions, print messages, and return data or status information to the calling application program. In addition, you can pass data to and from a procedure via parameters.

Procedures are normally created with ISQL (Interactive SQL) using the CREATE PROCEDURE SQL statement; however, they may be created with any application program that supports SQL as well.

Figure 7 demonstrates how to create a procedure using ISQL. The procedure defines the logic to add a new purchase order item to the database and update the respective total price and quantity for the order. It is assumed that the purchase order already exists and that the TOTALPRICE and TOTALQTY columns had been previously added to the ORDERS table.

```
CREATE PROCEDURE PURCHDB.ADDPOITEM
          (PONumber       INTEGER          NOT NULL,
           PartNumber     CHAR(16),
           Price          DECIMAL(10,2)  NOT NULL,
           Quantity       INTEGER          NOT NULL,
           DateRequired   DATE             NOT NULL)       AS
             BEGIN
              DECLARE Items INTEGER NOT NULL;

              SELECT COUNT(*) INTO :Items FROM PURCHDB.ORDERS
                     WHERE ORDERNUMBER = :PONumber;

              INSERT INTO PURCHDB.ORDERITEMS
                     (ORDERNUMBER, ITEMNUMBER, VENDPARTNUMBER,
                      PURCHASEPRICE, ORDERQTY, ITEMDUEDATE)
                     VALUES (:PONumber, :Items+1, PartNumber,
                             :Price, :Quantity, :DateRequired);

             UPDATE PURCHDB.ORDERS
                     SET TOTALPRICE = TOTALPRICE + :Price,
                         TOTALQTY   = TOTALQTY   + :Quantity
                     WHERE ORDERNUMBER = :PONumber;

            END;
```

Figure 7. Create Procedure Example

Invoking Procedures

Procedures are invoked directly using the EXECUTE PROCEDURE SQL statement via a PC client application program such as Visual Basic or a host-based SQL program written in a language such as COBOL or indirectly via business rules.

Creating Rules

Rules allow you to define generalized constraints or conditions on rows of a table by invoking procedures whenever specified data manipulation statements are performed on a table. The rule fires, that is, invokes a procedure each time the specified data manipulation statement (INSERT, UPDATE, or DELETE) is performed and the rule's search condition is satisfied. This permits data processing to be carried out directly by the IMAGE/SQL database, resulting in less application coding and more efficient use of resources.

Rules are normally created with ISQL using the CREATE RULE SQL statement; however, they may be created with any application program that supports SQL as well. There is no statement to invoke a rule since they are invoked indirectly.

The CREATE RULE statement identifies a table, types of data manipulation statements, a firing condition, and a procedure to be executed whenever the condition evaluates to TRUE.

Figure 8 illustrates a rule and procedure example. The procedure UPDATE_PO_TOTALS "fires" or executes whenever a purchase order line item is added or deleted or whenever the PURCHASEPRICE or ORDERQTY columns on an existing line item is updated. The procedure UPDATE_PO_TOTALS computes new values for the TOTALPRICE and TOTALQTY columns for the purchase order. As illustrated in Figure 6 above Visual Basic can easily invoke this rule by issuing INSERT, DELETE, or UPDATE SQL statements via the ExecuteSQL Method.

Note, procedures invoked by rules can include data manipulation statements that invoke rules that trigger the execution of other procedures. IMAGE/SQL and ALLBASE/SQL support a maximum rule chain of 20.

```
CREATE RULE PURCHDB.UPDATE_PO_TOTALS
        AFTER INSERT, DELETE, UPDATE (PURCHASEPRICE, ORDERQTY)
        ON PURCHDB.ORDERITEMS
        EXECUTE PROCEDURE PURCHDB.UPDATE_PO_TOTALS (ORDERNUMBER);


CREATE PROCEDURE PURCHDB.UPDATE_PO_TOTALS
        (PONumber INTEGER NOT NULL)     AS
          BEGIN
            DECLARE TotalPrice DECIMAL (15, 2) NOT NULL;
            DECLARE TotalQty    INTEGER          NOT NULL;

            SELECT SUM (PURCHASEPRICE) INTO :TotalPrice,
                        SUM (ORDERQTY)        INTO :TotalQty FROM
                        PURCHDB.ORDERITEMS
                        WHERE ORDERNUMBER = :PONumber;

            UPDATE PURCHDB.ORDERS
                        SET TOTALPRICE = :TotalPrice,
                              TOTALQTY    = :TotalQty
                        WHERE ORDERNUMBER = :PONumber;

          END;
```

Figure 8. Rule and Procedure Example

Transaction Handling Considerations

There are differences related to transaction management when a procedure is executed directly or indirectly via a business rule. When a procedure is executed directly via the EXECUTE PROCEDURE SQL statement, it is up to the programmer to determine where the transaction boundaries should occur. The programmer has two choices. The transaction handling can be done either within the procedure or outside the procedure. On the other hand, if a procedure is invoked via a rule, SQL treats the data manipulation statement that invoked the rule and the stored procedure that is subsequently executed as an atomic unit. Therefore, BEGIN WORK, COMMIT WORK, and ROLLBACK WORK statements encountered within a procedure invoked by a rule will result in an error condition and all previously executed statements tied to this rule will be undone.

CONCLUSION

As you can see, there are many features supported with the industry-standard SQL interface that make it easy to quickly develop new client/server applications using ODBC. IMAGE/SQL allows you to take advantage of these features, while protecting your investment in your existing TurboIMAGE applications.

The SQL Data Manipulation Language enables flexible access to your corporate data, while stored procedures reduce the need for extensive application programming in addition to providing a means to improve application performance in a client/server environment by reducing network traffic.

Armed with this information you should now be ready to take the next step to evolving to client/server computing by initiating a client/server application development pilot with IMAGE/SQL!