Santa Teresa
Laboratory
San Jose, CA

TECHNIQUES IN SQL APPLICATION DESIGN

by Nancy Wheeler

May 1986

TR 03.290

TR 03.290

TECHNIQUES IN SQL APPLICATION DESIGN

May, 1986

Nancy Wheeler

IBM
General Products Division
Santa Teresa Laboratory
San Jose, California

ABSTRACT

This report identifies common technical and design issues that arise in creating an application for an interactive environment using Structured Query Language (SQL). The topics include database design, locking, performance, error-handling and application portability. The target audience is programmers just beginning to use a relational database.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

INTRODUCTION

When programmers first begin thinking of writing applications that will use IBM relational databases, either Structures Query Language/Data System (SQL/DS) or IBM Database 2 (DB2), they first think about the programming questions. For example, how will the SQL statements be coded so that the correct data will be retrieved and updated? Once the data is retrieved, how will it be manipulated or displayed? How will the reports wanted by management be generated? These questions, all valid, need to be answered.

Another set of issues arise when designing a relational database application. These issues are not directly related to the data manipulation code, but rather have to do with the definition of the environment in which the code executes. Often these issues are ignored, or addressed after the fact. If, however, they are addressed early in the design of the application, the ease with which the application moves into efficient production can be increased.

This report is not intended to resolve all the issues nor answer all the questions. Its purpose is to identify the issues and give direction and references in order that informed decisions can be made.

The focus of this report is on design issues for SQL applications written in APL2. APL2 uses DYNAMIC SQL to execute its SQL statements. Dynamic SQL is used for interactive environments where the specific table and column names to be used in the queries are not known until execution time. STATIC SQL is used in compiled programs and requires that the table names and columns be known at compile time. Design issues for static SQL programs are sometimes different and not specifically discussed here.

DATABASE DESIGN

Before an SQL application can be written, the database configuration to be used must be designed. This design involves finding a place for the relational tables, creating the tables, indexing the tables intelligently, and deciding on a scheme for accessing the tables.

THE DATABASE

SQL/DS and DB2 each have different structures for their database storage allocation, although the two structures do parallel each other.

SQL/DS: In SQL/DS, a DATABASE is managed by a VM virtual machine. Some installations will have only one database defined, while others may have multiple databases. Since the databases are created by system programmers and are large-scale entities, the normal decision is choosing the database appropriate for the application. Only for very large applications would it be likely that an entire database was dedicated to an application. It should also be remembered that users must issue an EXEC, SQLINIT, to switch databases. That may be a consideration in designing the invocation process for your application.

Within a database, tables are created in DBSPACES. There are two types of DBSPACES, public and private. The basic difference between them is that anyone with the right authority level can create a table in a public DBSPACE, while only the owner of a private DBSPACE can create tables in it. Private DBSPACES also require more stringent locking policies. If there will be much concurrent access of the tables in your application, a public DBSPACE is probably best. Whether to create a separate DBSPACE for your application alone depends on its size and frequency of use.

Figure 1 on page 3 shows a pictorial representation of the SQL/DS structure.

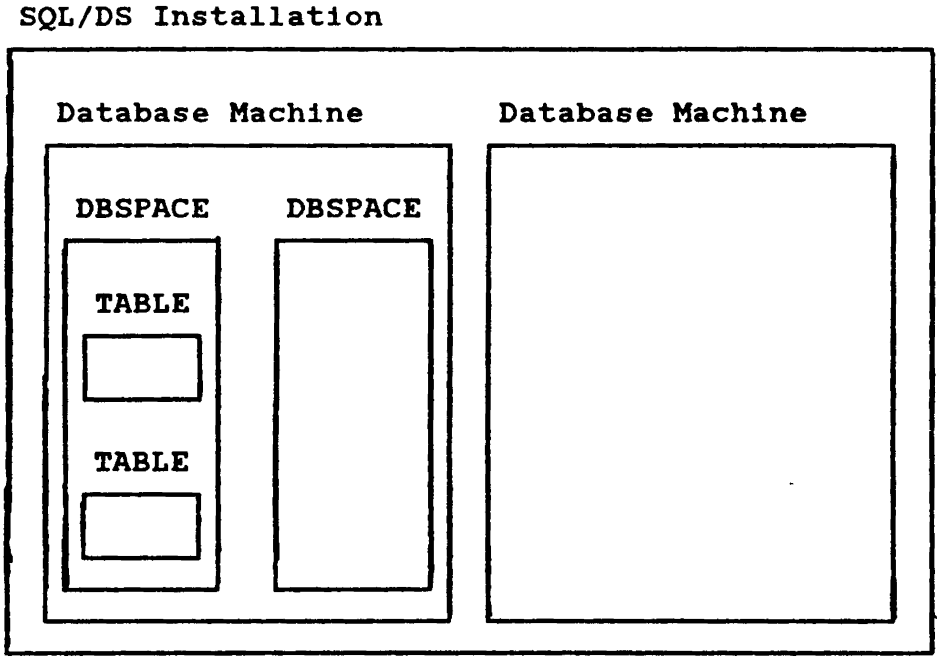

Figure 1. SQL/DS Database Structure

DB2: In DB2, a DATABASE is not a physical entity. It is a way to logically group tables together. If your application is large and has more that a few tables, it may be a good idea to group them together by defining a separate database for them. If not, you will need to be assigned a DATABASE to put them in.

The physical entity in which tables are created in DB2 is the TABLESPACE, which is analogous to the SQL/DS DBSPACE.

Figure 2 on page 4 shows a pictorial representation of the DB2 structure.

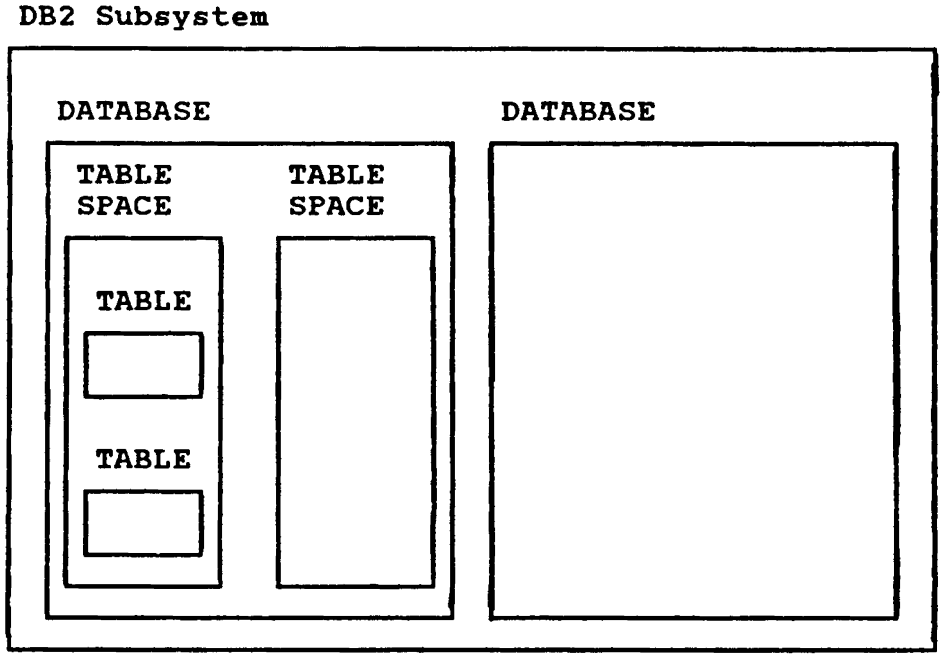

Figure 2. IBM Database 2 Database Structure

THE TABLE

Table design is a topic to which an entire paper could be devoted, and many papers have been written on design methodology for relational tables. Here we will be content to just describe to you some of the database-specific considerations of table design.

TABLE SIZE: There are limits on the width of table you can create.

In SQL/DS this limit is in the number of columns (255) you can define for a table, and in the maximum number of bytes (approximately 4070) that each row can occupy. Note, however, that the limit in bytes does NOT include LONG VARCHAR columns. They are stored differently in the database. More about LONG VARCHAR in a moment.

In DB2, you can declare up to 300 columns per table, and the maximum number of bytes per row is based on your page size (4K or 32K, depending on the installation). The DB2 byte limit does include LONG VARCHAR columns.

COLUMN TYPES: The SQL column data types supported by APL2 are summarized in Figure 3 on page 6. Most of the types are fairly straightforward. The LONG VARCHAR type, however, is handled quite differently in SQL/DS and in DB2.

In SQL/DS, the LONG VARCHAR type is a distinctly different type from the VARCHAR type. VARCHAR columns can be declared with widths up to 254 characters. LONG VARCHAR columns have no size specification. Their size is fixed at 32K and there are many restrictions as to their use. LONG VARCHAR columns cannot be used in search predicates, no indices can be created using them, and you cannot use them to sort your table.

In DB2, the LONG VARCHAR type is really just the VARCHAR type with a default maximum width. A column declared as LONG VARCHAR is treated the same as one declared as VARCHAR. You can declare a VARCHAR column with any width up to the page size of your installation. If you use the LONG VARCHAR type to define a column, that page size will be used as the default width.

In either database system, you should be judicious in your use of LONG VARCHAR columns. Storage must be allocated in the APL2 SQL interface (AP 127) to retrieve data from tables, and often it is difficult to allocate enough to fetch more than a few rows with LONG VARCHAR columns. Because AP 127 does not know until after the fetch is complete how many bytes of real data are in the column, it must allocate enough for the maximum width. LONG VARCHAR should only be used in SQL/DS if the data will actually be wider than 254 characters most of the time. In DB2, since you can declare VARCHAR columns wider than 254 characters, it is better to make an accurate estimate of the size of the data, and use the VARCHAR declaration with that number. These techniques will help you minimize the amount of storage unnecessarily allocated.

SQL DATA TYPE	DEFINITION
SMALLINT	Halfword (15-bit) integer.
INTEGER	Fullword (31-bit) integer.
FLOAT	Double-Precision (8-byte) floating point.
DECIMAL m,n	Packed decimal, where m is the column width and n is the number of digits to the right of the decimal.
CHAR n	Fixed-length character, with column width specified by n, where n ≤ 254.
VARCHAR n	Variable-length character, with maximum column width specified by n. In SQL/DS, n ≤ 254. In DB2, n ≤ 32767.
LONG VARCHAR	Variable-length character. Maximum column width defaults according to database.

Figure 3. SQL Datatypes

VIEWS: If your tables are large, or if you want certain users or groups of users to access only part of the data they contain, you can create VIEWS of the tables. VIEWS are logical tables whose definitions are based on real tables; users can access them much as they would a real table. If you want to use the VIEWS to update the data, however, you should read carefully about the ramifications of doing so. Because VIEWS do not physically exist, the update rules for them are somewhat more restrictive than the rules for tables.

THE INDEX

Choosing appropriate indices for your tables is one of the most important design decisions you will make, because it can have a very great effect on the performance of your application. You can define indices on columns or combinations of columns, and indices can be unique (no duplicates are allowed in column(s) used to define the index) or nonunique.

If no indices are defined on a table, each row of the table must be searched every time you issue a query to find the rows that meet the conditions of the query. If there is an index on the column you specify in your search condition, the rows that meet the condition may be accessed directly, thereby speeding up the query.

Another effect of an index is that it can force uniqueness of data. If a UNIQUE INDEX is defined on a column or group of columns, the database will not allow duplicate data to be inserted into those columns.

The design question to be answered with indices is when to stop. How many indices is too many? If you define an index on every column in your table, you will have direct access to data whenever possible (there are some circumstances where an index cannot be used). The time required to update the table, however, will be increased. Every time a row is updated, each index must also be updated. The correct answer to the question is, of course, dependent on your application. Ideally, you will define indices on columns that will frequently be used in search conditions, and not define so many that updating the table is prohibitively slow.

AUTHORITY

When using Dynamic SQL, authority must be granted to users to access the tables after they have been created. Different levels of authority that can be granted (SELECT, UPDATE, INSERT, etc.); care should be taken that you do not grant more authority than is necessary to any user. There are many schemes for managing authority. We list a few here:

- Grant all authorities to all users. All users will then be able to access and update all data. This scheme is

not advised unless the integrity of the data is unimportant.

- Grant SELECT authority to all users and UPDATE/INSERT authority to a limited number of users. This scheme offers more integrity but limits usability. All database update requests must be routed to one of the few users who can do them.

- Grant SELECT authority to all users for interactive data access and write batch programs using STATIC SQL to do updates. Programs written in VS FORTRAN and IBM 370 Assembler Language can be called from APL2 using the Name Association Facility. Authority to update the batch programs is based on the authority to run the program rather than the authority to update the tables. This method restricts the updates as desired, but is less flexible since a new batch program must be written each time different type of update is desired.

- Define VIEWS for groups of users and allow the groups to update only their views. This technique may have restrictions based on the rules for updating VIEWS.

- Use a central server user ID, and route all requests through the server. The server is, then, the only ID that has authority to access the data. The server's programs can be written to check each user's requests for correctness, and can further refine the authority scheme by allowing users to update only certain columns of the tables. See "Multi-User SQL Applications in APL2", Dr. James A. Brown (IBM TR 03.247) for a complete discussion of this scheme.

- (SQL/DS only) Use the CONNECT command in your application to connect the user as the application user ID. In this scheme, only the application user ID has authority to access the tables. All users will be able to make updates while running the application, but will not be able to access the tables outside the application from their own user ID. The updates, therefore, are restricted to the type and format allowed by the application.

LOCKING

The types of locks obtained when accessing SQL tables, and the duration of those locks, can affect the perceived performance of an application. "Concurrency" is the term used to refer to the ability to have multiple users accessing data simultaneously. In general, with greater concurrency, there is less average waiting time for data access, and thus performance is perceived to be better.

The application can control some aspects of locking during execution, and others are determined at design time.

LOCK SIZE

The size of the locks obtained is determined at the time the DBSPACE or TABLESPACE is allocated. In DB2, a TABLESPACE can be locked in its entirety, or a page at a time. In SQL/DS, a public DPSPACE can be locked by DBSPACE, page, or row. A private DBSPACE is always locked in its entirety.

With TABLESPACE or DBSPACE locking, fewer locks are necessary but their scope is wider. Anyone else wishing to access the affected table or tables must wait until the current transaction is terminated before being granted that access.

With page locking, locks are obtained for the pages on which the rows accessed by the transaction reside. With row locking, locks are obtained only for the rows accessed by the transaction. If the transaction only accesses a few rows of the table, other users accessing different parts of the table may have a shorter wait. If the transaction accesses most of the rows of the table, however, the performance impact of obtaining many locks may be greater than that of the waits by other users. When deciding on a lock size, the types of transactions to be executed should be considered.

ISOLATION LEVEL

The duration of locks is controlled by the ISOLATION LEVEL attribute, The isolation level for APL2 is chosen during the BIND (DB2) or SQLPREP (SQL/DS) step of the APL2 installation.

There are two different isolation levels. With the REPEATABLE READ (RR) isolation level, locks are not released until the unit of work terminates with a COMMIT or ROLLBACK. With the CURSOR STABILITY (CS) isolation level, locks are released when the cursor moves off the area (DBSPACE, TABLESPACE, page, or row) locked if the data on the area has not been changed. If it has been changed, the locks are held until the unit of work terminates.

Repeatable read locking guarantees that if the same data is read twice during the same unit of work, the data will not have changed. Cursor stability locking allows more concurrency. Once an application is finished reading data and it moves on, other users can then access the data. However, CS also means that if the first application reads the data again, it could be different than it was before. Figure 4 shows an example where data integrity could be lost.

PROGRAM ACTION	DATABASE ACTION
P1 reads R1	R1 locked
P1 reads R2	R1 unlocked
P2 reads R1	R1 locked
P2 updates R1	
P2 commits	R1 unlocked
P1 updates R1 but R1 is different	R1 locked but its integrity is uncertain

Figure 4. Cursor Stability Locking

In general, applications that will do updates and depend on reading other data to determine which updates to make should not use CS locking. Applications that only read data can benefit from the added concurrency provided by CS locking.

In DB2, it is possible to BIND two plans for the same program. One approach in isolation level control is to BIND one APL2 plan with RR and one with CS, and execute with the RR plan only when doing updates.

EXPLICIT LOCK CONTROL

In addition to controlling locking with installation parameters, locking may be explicitly controlled with the SQL LOCK command. The SQL LOCK command will lock the entire TABLESPACE in which the table being accessed resides.

Since SQL LOCK commands cause the default locking to be overridden, another method for maximizing concurrency is to choose the CS isolation level with page or row locking, and use SQL LOCK commands when doing updates to insure data integrity.

UNITS OF RECOVERY

A unit of recovery (unit of work) ends when a ROLLBACK or COMMIT is issued by the application. At the time of the ROLLBACK or COMMIT, the changes made by the application are made permanent (COMMIT) or discarded (ROLLBACK), locks still held are released, and the state of the application is reset (queries are purged, and associated storage may be released).

Applications should issue COMMITS or ROLLBACKS as frequently as possible in order to free up data for other users, while maintaining the integrity of the data. If, for example, a transaction involves making two updates, and the two updates are dependent on each other, the COMMIT should not be issued until both updates are complete.

In compiled programming languages, an implicit COMMIT is issued for the program if it terminates normally. If it terminates abnormally, an implicit ROLLBACK is issued. In APL2, no implicit COMMITS are issued. Applications written in APL2 must issue COMMITS explicitly in order for their work to become a permanent part of the database. An implicit ROLLBACK (in SQL/DS, a ROLLBACK RELEASE) is issued when the AP 127 shared variable is retracted.

PERFORMANCE CONSIDERATIONS

One of the most frequently asked questions is, "How can I make my SQL application run faster?" Unfortunately, this question is also the most difficult one to answer. There are many factors that can affect the performance of an SQL application, some of which programmers can do something about, and some of which they cannot. The list of items here is not exhaustive, but each area mentioned is one in which careful design can make a difference.

INDICES

As stated previously, indexing can make a measurable difference in query performance.

CONCURRENCY

In general, the more concurrency your application can achieve, the better the perceived performance of the application. Even though the application is not using CPU time while it waits for locks, the user usually perceives the wait as a problem with the application. Techniques for achieving more concurrency are mentioned in the discussion of locking above.

QUERY OPTIMIZATION

Some SQL queries are very straightforward, and can only be written one way. When coding more complex queries, there may be several different ways to achieve the same result, each with different execution times. For example, the BETWEEN keyword is more efficient than using the equivalent mathematical expression (see Figure 5 on page 14), and a JOIN is less expensive than a correlated subquery. More techniques for query optimization are covered in the database application programmer's guides.

```
SELECT * FROM TAB WHERE NUM BETWEEN 1 AND 10

is faster than

SELECT * FROM TAB WHERE NUM >= 1 AND NUM <= 10

because the index is used more efficiently
```

Figure 5. Query Optimization

The SQL EXPLAIN command can be used to analyze query performance. EXPLAIN is executed with an SQL statement as its argument, and it places data about the query execution into an SQL table or tables defined for that purpose. Using EXPLAIN, a programmer can test variations of a query to see the differences in their performance.

BLOCKING

The SQL language allows only one row of data to be fetched from the result table at a time. There are two ways that the fetches can be blocked to save on execution time.

SQL/DS BLOCKING: In SQL/DS Release 3, the BLOCK option can be specified during the SQLPREP step of APL2 installation. When possible, SQL/DS will fetch a block of rows into a buffer, and do the individual fetches from that buffer. The access time to the buffer is less than that to the real data.

APL2 BLOCKING: When fetching data from SQL/DS or DB2 using the APL2 interface, it is possible to specify how many rows AP 127 should retrieve. AP 127 will fetch the desired number from the database and build one APL2 object to pass to the user. Using a larger blocking factor will cause the application to need fewer calls to AP 127 to retrieve the data it needs. If, for example, your result table has 1000 rows, a blocking factor of 20 will require 50 AP 127 calls to fetch the entire table. Change the blocking factor to 500 and only two calls to AP 127 will be necessary.

When deciding on the APL2 blocking factor, storage needs must also be considered. While it may seem smart to set blocking to a very high number to minimize the AP 127 calls, larger blocking factors result in larger storage requirements. If your tables have many thousands of rows, a compromise will probably be necessary.

CODING TECHNIQUES

There are some additional coding techniques that can make an application more efficient in its use of the SQL interface.

USING HOST VARIABLES: If the same query can be used more than once in a unit of work by using a host variable for selection criteria instead of a literal string, the total cost decreases, because the cost of the prepare can be split among all the calls rather than repeated for each call. Figure 6 on page 15 shows an example.

```
Prepare the query

DELETE FROM TAB WHERE NUM = ?

or

DELETE * FROM TAB WHERE NUM = :1 (in APL2)

and execute it several times, rather than executing

DELETE FROM TAB WHERE NUM = 1
DELETE FROM TAB WHERE NUM = 52
DELETE FROM TAB WHERE NUM = 99
```

Figure 6. Using Host Variables

USING THE LOWEST LEVEL OF THE INTERFACE: When several levels of access to SQL are available, choose the lowest level to avoid unnecessary overhead. In APL2, access to AP 127, the SQL interface, can be achieved using an end-user workspace function (SQL), or with application programmer functions (PREP, OPEN, etc.). The SQL function must parse the statement, choose the correct application programmer func-

tions, call them , and check for errors. While it is an excellent tool for ad hoc queries and testing, a production application will have better control and performance if it uses the direct access and adds its own customized error-checking.

OUTPUT FORMAT: Sometimes several different output formats are available. In APL2, the result data can be retrieved in two formats, MATRIX and VECTOR. The VECTOR format requires less space and therefore may improve the performance of the fetch. The MATRIX format is a more flexible format for manipulation in APL2. If the application can use the VECTOR object to do its computations, VECTOR may be the format of choice, and the larger the result tables are, the greater the savings will be. Figure 7 on page 16 shows the APL2 DISPLAY of the two formats.

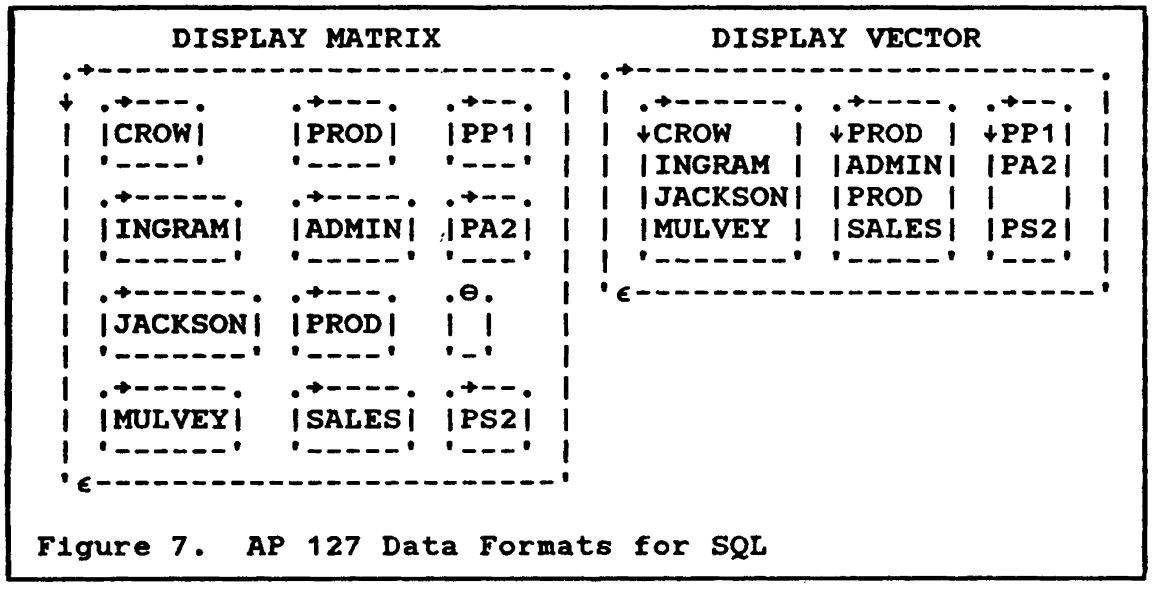

Figure 7. AP 127 Data Formats for SQL

ERROR-HANDLING

Error-handling, of course, is a part of every application design. In an SQL application, there are two levels of errors. There are errors from the programming environment (in APL2, AP 127) and errors from the database (SQL/DS or DB2). Error analysis can be tricky in SQL applications, especially if the application end users will enter SQL statements themselves. Determining whether the error was caused by the application or by the user can be difficult.

The APL2 interface, AP 127, provides the application with a numeric return code vector indicating the source and type of the error and the error code itself. Using this return code vector, several facilities are available to aid in error-handling.

THE SQLCA

The SQLCA is an SQL control block which provides error information. The fields of most interest to the application programmer are the SQLCODE (the error code), SQLERR (message tokens) and SQLWARN (warning indicators).

The database application programmer's guides contain detailed information on the structure of the SQLCA and the meaning of its fields.

In APL2, the SQLCA can be obtained using the AP 127 function MESSAGE.

MESSAGE TEXT RETRIEVAL

DB2 MESSAGE TEXT: DB2 provides an assembler routine, DSNTIAR, which accepts the SQLCA as its parameter and returns formatted message text. The MESSAGE function of AP 127 calls this routine automatically and returns the formatted text along with the SQLCA.

SQL/DS MESSAGE TEXT: SQL/DS provides HELP text for all SQLCODES and other SQL keywords in SQL tables shipped with the SQL/DS product. These tables can be accessed by the application using SQL SELECT statements.

AP 127 MESSAGE TEXT: If the return code vector indicated an AP 127 error rather than an SQL error, the MESSAGE function will return to the application the formatted text of the AP 127 message.

WARNING CONDITIONS

In addition to error conditions, SQL may also return warnings to the application programmer in the SQLCA. Warning conditions are identified by a positive SQLCODE (errors give negative SQLCODES) or by the character "W" in one of the SQLWARN fields. When a warning is returned, a condition that may or may not be indicative of a problem exists. For example, a DELETE was issued, but no rows in the table met the criteria specified in the DELETE.

APPLICATION PORTABILITY

There are two IBM relational database systems. IBM Database 2 (DB2) runs in the MVS environment. Strucured Query Language/Data System (SQL/DS) runs in the VM environment. In both databases, the SQL language is used for data access. There are, however, differences that an application programmer must be aware of.

DATABASE CONFIGURATIONS

The mechanisms used to set up the environment are different in the two databases. As we noted above, DB2 has the TABLESPACE, SQL/DS has the DBSPACE, and the two systems have a different concept of the DATABASE. Authorization privileges are only partially compatible, and user access to the system is achieved differently in DB2 and SQL/DS. As a result of these differences, the installation and setup procedures for an application must be different for the two systems.

CATALOG TABLES

Catalog tables are tables installed with the database system that keep track of the status of the system. For example, one catalog table keeps data on each table created. While the SQL/DS and DB2 catalogs contain similar information, their naming conventions and structures are different.

If your application uses information from the catalog tables, it is necessary to have separate routines to access the tables in the two systems. Another alternative is to define VIEWS of the system tables your application uses during the installation of the application. The creation of the views would be different in the two environments, but the code that accessed the views could then be the same.

SQL INCOMPATIBILITIES

There are not a great number of SQL language incompatibilities between SQL/DS and DB2 other than the configuration differences already mentioned. Some SQL extensions exist in SQL/DS, however, that are not supported in DB2. One is the CONNECT command, a command that allows connection to the database with a user ID other than the VM user ID. The set of commands referred to as EXTENDED DYNAMIC SQL is also not supported in DB2. Extended Dynamic SQL allows an interactive program to prepare queries and save them in the database to be executed at a later time.

NOTE: APL2 supports CONNECT (an error code is returned if it is used in the DB2 environment), but it does not support Extended Dynamic SQL.

A complete list of the SQL incompatibilities between SQL/DS and DB2 can be found in the Development Guide for Relational Applications.

ERROR CODES

In general, given a specific error condition, both DB2 and SQL/DS will return an error code. The number of the error code, however, is usually not the same. An application that is to be portable should not depend on any specific SQLCODES, but rather have general error-handling routines.

CONCLUSION

This report has discussed various topics in SQL application design, at a very high level, in order to help programmers become more aware of the issues they will encounter. More detail is available on all these topics and other topics of interest to the SQL application programmer. The bibliography lists the references that were used in preparing this paper. In addition, the SQL/DS, DB2 and APL2 products each have a complete library of publications available for further research.

BIBLIOGRAPHY

1. SQL/Data System Application Programming for CMS (GH24-5068)

2. SQL/Data System Planning and Administration for VM (SH24-5043)

3. IBM Database 2 Application Programming Guide for TSO (SC26-4081)

4. IBM Database 2 Reference (SC26-4078)

5. Development Guide for Relational Applications (SC26-4130)

6. APL2 Programming: Using Structured Query Language (SH20-9217)

7. Multi-User SQL Applications in APL2 (TR 03.247)