OS/2 EE Database Manager overview and technical highlights

by P. Y. Chang
W. W. Myre

Structured Query Language (SQL) has become an industry standard. It is supported by mainframe products. This paper describes the OS/2 EE Database Manager, which is based on the relational database model of E. F. Codd and on the SQL query language. A functional overview of the OS/2 EE Database Manager and OS/2 EE is provided; technology applied to different areas is highlighted.

With the increasing power of personal computer systems, users' database management requirements are growing at a rapid pace. The amount of data that can be stored and managed on a personal computer is rapidly increasing. The complexity of the data and the variety of applications on top of these data also grow by leaps and bounds. Users who once were satisfied with simple file-manager programs now need powerful database management tools to manage and control their data. Application developers also are requiring the full features of database management systems once available only on larger machines.

The OS/2 Extended Edition (OS/2 EE) Database Manager is included in the IBM OS/2™ Extended Edition to satisfy the database requirements of both end users and application developers. It is an IBM-developed, full-function, relational database manager supporting the IBM Structured Query Language (SQL). IBM technology and architecture in the OS/2 EE Database Manager provide high performance, single-user and multiuser concurrent access, robust data integrity, and data protection facilities.

The OS/2 EE Database Manager is based on the SQL query language and on the relational database model[1] invented by E. F. Codd at the IBM San Jose Research Center. The relational model has been widely accepted by the industry and by end users. The main advantage of the relational database model is its clear separation between the user perception and the internal implementation of data. In the personal computer environment, user friendliness is a paramount concern. The relational model has been designed to be easy to understand, while at the same time the Database Manager itself is free to implement efficient access mechanisms to optimize performance. In the OS/2 EE Database Manager, the actual storage and access methods are far more complex than the simple table structure of the data would indicate, but all the complexity is hidden from the users. Users of the OS/2 EE Database Manager see only the simple relational form of data; the Database Manager is responsible for providing good performance characteristics.

Structured Query Language (SQL),[2] originally developed in the System R project[3] at IBM San Jose Research, has also become a standard in the industry. SQL is considered simple to learn, yet powerful in

expressing sophisticated queries. A single statement in SQL can perform the same function as many lines of conventional code.

SQL is supported by two products in the IBM relational product family: IBM Database 2 (DB2)[4] and Structured Query Language/Data System (SQL/DS), both on IBM System/370. SQL is also included in the IBM Systems Application Architecture (SAA)[5,6] as the database interface component of the common programming interfaces. SAA provides a standard interface across the three major IBM computing environments: System/370 (TSO/E under MVS/XA, and CMS under VM), System/3X, and personal computers (OS/2). The OS/2 EE Database Manager supports the SAA Database Interface and is consistent with DB2 and SQL/DS.

The interactive end-user tool included in the OS/2 EE Database Manager, called the OS/2 EE Query Manager, provides data entry, edit, query, report, and customized application support. The OS/2 EE Query Manager supports the SAA query interface and is consistent with the Query Management Facilities (QMF) product on System/370, which provides query and report interfaces to DB2 and SQL/DS.

This paper describes the functions of the OS/2 EE Database Manager and highlights the technologies used in developing it. The paper by Watson[7] in this issue highlights the functions and user interfaces of the OS/2 EE Query Manager.

IBM relational technology

After the creation of the basic theory on the relational model, IBM developed several research prototypes based on the model. The best-known prototype developed in the San Jose Research Laboratory is called System R. The SQL language was invented as part of the System R research. Other technologies developed with System R include SQL statement processing techniques, query optimization and compilation, concurrency control, and locking and logging protocols. Most importantly, System R showed that the relational model can provide significant benefits and reasonable performance in real-life environments.

The System R prototype provided the basis for research in the Distributed Database Management Systems (DDBMS) area. The R* project[8] at the San Jose Research Laboratory was a test bed for many breakthroughs in DDBMS research. Global optimization, global naming, global data definition, and optimized two-phase commit are among the most significant results. This research showed that the relational model is also suitable for the distributed database environment.

After the R* project was finished in 1984, IBM Research continued the development of the relational database technology with a project called Starburst[9]. This project is continuing the work on DDBMS in the area of heterogeneous DDBMS interactions. New techniques for implementing the relational model and for building extendable database systems are among the current research topics.

The OS/2 EE Database Manager was designed specifically to run in the OS/2 environment.

The SQL/DS product, running in the VSE and VM environments, is a direct descendant of the System R project. Many pieces of System R were rewritten to make it a program product and to adapt it to the VSE and VM environments. New techniques in the areas of recovery were developed in SQL/DS. The first version of SQL/DS was released in 1981.

The DB2 product, running in the MVS environment, was designed specifically for MVS with large-system emphasis. It contains some of the new code written for SQL/DS, but most of the SQL function was rewritten to incorporate new technologies developed at San Jose Research and in the Santa Teresa Laboratory. New techniques in DB2 include logging and locking, recovery protocols, and buffer-management techniques. The first version of DB2 was released in 1985. The QMF product, released together with DB2, is the end-user facility for DB2 and SQL/DS. It permits users to submit queries interactively and to generate customized reports.

The OS/2 EE Database Manager benefited from all the technologies, research effort, and product expe-

riences mentioned above. It was designed specifically to run in the OS/2 environment. The code in the OS/2 EE Database Manager was newly written to utilize innovative technologies and to optimize for OS/2. Researchers at the IBM Almaden Research Center were involved in the original design of the OS/2 EE Database Manager and were held in close consultation throughout the development cycle. Developers of DB2, SQL/DS, and QMF were also consulted on a regular basis.

The significant benefits to the OS/2 EE Database Manager from this technology transfer can be found in the following areas:

Performance: Performance is the paramount design objective of the OS/2 EE Database Manager. Many new technologies have been applied in this area, e.g., optimization techniques, join algorithms, and buffer-

The functions of the OS/2 EE Query Manager can be used to create customized applications.

management techniques. New indexing and locking techniques also provide a high level of concurrency while maintaining high levels of performance.

IBM Systems Application Architecture (SAA): Care was taken to make sure SAA benefits can be provided in the OS/2 EE Database Manager. The syntax and semantics of the database interface functions are designed to converge with those of other products, and the program product architectures are also designed to work together. For example, the consistent handling of host variables, precompile, bind, and query optimization simplifies the users' work in writing cross-system applications.

Robust data integrity support: Advanced transaction-management and logging techniques are used to preserve the integrity of data.

Distributed Database Management Systems (DDBMS) support: Though most of the DDBMS functions are not implemented in the first release, the product has been designed for growth into the DDBMS areas. Architectural similarities with DB2 and SQL/DS will simplify distribution among these heterogeneous products.

The rest of this paper provides a functional overview of the OS/2 EE Database Manager. Technologies applied in different areas are also highlighted.

System structure

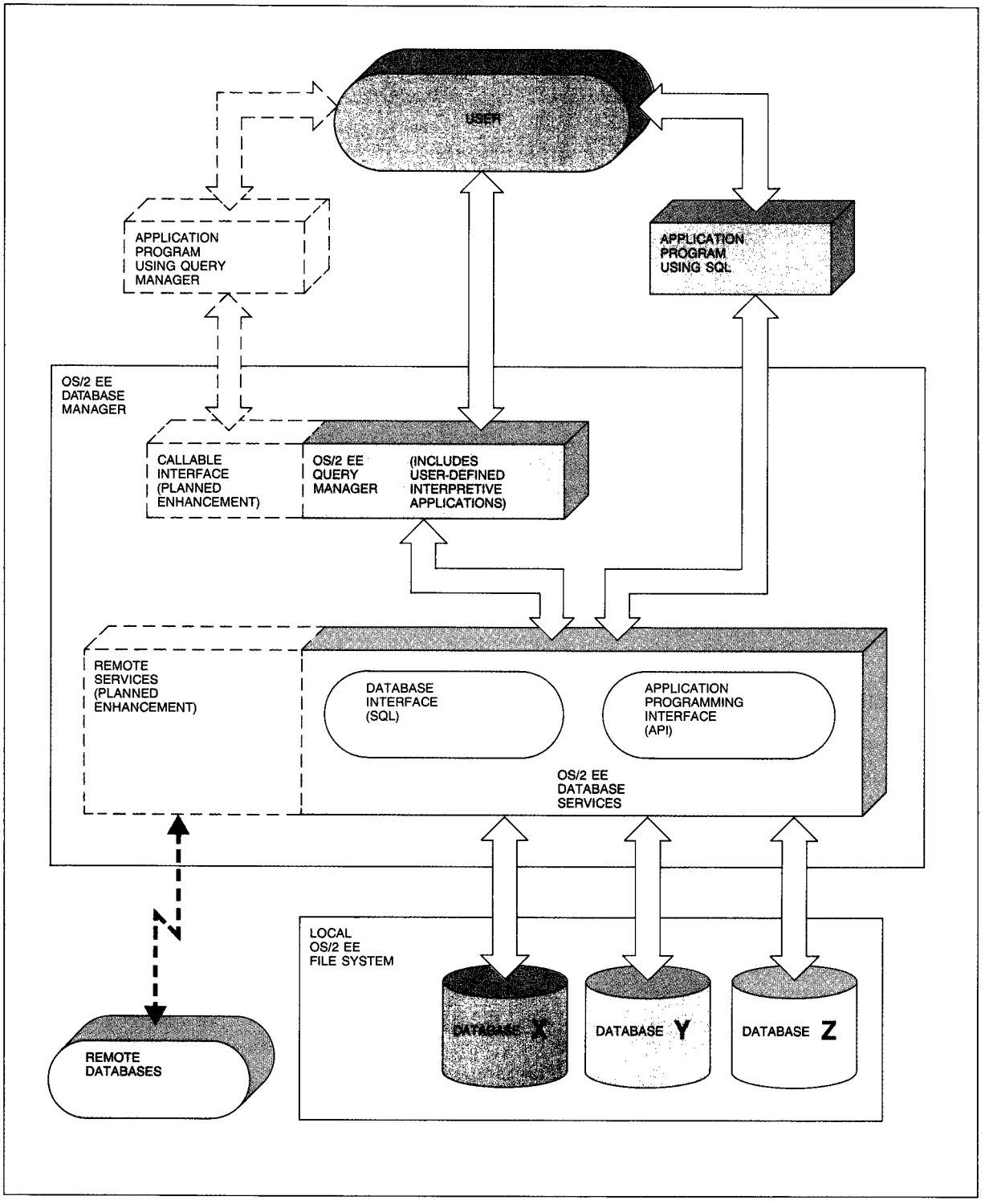
Figure 1 illustrates the OS/2 EE Database Manager functional structure. Note that some of the functions are marked with '*' as *planned enhancements*. Those functions are announced as part of a future release. They are not shipped in the first release of OS/2 EE.

Figure 1 shows three ways to use the OS/2 EE Database Manager functions:

- The OS/2 EE Query Manager interactive interface can be used to perform *ad hoc* queries, data entry, update, data definition, or reporting applications.
- The functions of the OS/2 EE Query Manager can be used to create customized applications with forms, menus, queries, reports, and procedures. The customized application can be executed repeatedly—the user of such applications does not even need to learn most OS/2 EE Query Manager functions or know that the OS/2 EE Query Manager is involved.
- Traditional programming languages, such as C, can be used. The OS/2 EE Database Manager functions can be accessed via the Database Interface and the OS/2 EE Database Manager Application Program Interface (API). To use the Database Interface functions, the programmers would need to issue SQL statements in their programs. The OS/2 EE Database Manager language-specific precompilers and binders are used to establish links (database requests) between user programs and the Database Services.

Regardless of the way in which the OS/2 EE Database Manager functions are used, all accesses to the database are performed by the Database Services component of the OS/2 EE Database Manager. This component, the "engine" of the OS/2 EE Database Manager, actually manages the data stored in the database and supports the SQL functions via the Database

Figure 1 OS/2 Database Manager structure



Interface. It generates optimized access plans and supports transactions, concurrency, storage, and buffer-management functions. It also handles utility functions such as import, export, backup, restore, reorg, and runstats via the Utilities API.

If the database being accessed is remote on a local-area network (LAN), Database Services calls the Remote Services component (a planned enhancement) to handle the remote access. The Remote Services component uses the SNA Advanced Program-to-Program Communication (APPC) protocol for remote database accesses. APPC is supported by the IBM OS/2 EE Communication Manager, another component included in the IBM OS/2 Extended Edition.

In the next sections, major functions provided in the Database Interface, the API, the precompiler and binder, the Database Services, and the Remote Services components are described.

Database Interface support

Data manipulation language. SELECT, UPDATE, INSERT, and DELETE are the four basic data manipulation language functions in SQL. SELECT is the most powerful SQL statement and is used for querying tables or views. Some examples of the supported SELECT statements are shown below. These statements assume two tables:

```
AUTHORS⟨AUTHORID,
   LASTNAME,FIRSTNAME,ROYALTIES⟩
BOOKS⟨TITLE, COPIES, TYPE, AUTHORID⟩
```

1. Ordering of output rows:
```
SELECT LASTNAME, FIRSTNAME
FROM AUTHORS
ORDER BY LASTNAME ASC, FIRSTNAME ASC
```
2. Built-in functions and expressions are supported:
```
SELECT AVG(ROYALTIES)*0.5 FROM AUTHORS
```
3. Get summary rows with GROUP BY and HAVING:
```
SELECT TYPE, SUM(COPIES)
FROM BOOKS
GROUP BY TYPE
HAVING TYPE = 'SF' OR TYPE = 'FI'
```
4. Joins of up to 15 tables:
```
SELECT LASTNAME, TITLE
FROM AUTHORS A, BOOKS B
WHERE A.AUTHORID = B.AUTHORID
  AND
  TYPE = 'SF'
  AND
  FIRSTNAME = 'JOHN'
```
5. Subqueries:
```
SELECT FIRSTNAME,LASTNAME
FROM AUTHORS
WHERE ROYALTIES >
  (SELECT AVG(ROYALTIES)
  FROM AUTHORS
  AND
  AUTHORID = ANY (SELECT AUTHORID
                  FROM BOOK
                  WHERE TYPE = 'SF'))
```
6. Correlated subqueries:
```
SELECT TITLE
FROM BOOKS X
WHERE COPIES > (SELECT AVG(COPIES)
                FROM BOOKS
                WHERE TYPE = X.TYPE)
```

UPDATE, INSERT, and DELETE statements can have WHERE clauses with search conditions or subqueries

SQL cursoring functions step through the set one row at a time.

much like the SELECT statement above. For example, the following UPDATE statement would give all AUTHORS who write SF books a royalty increase of 10 percent:

```
UPDATE AUTHORS SET ROYALTIES = ROYALTIES * 1.1
WHERE AUTHORID = ANY (SELECT AUTHORID
                      FROM BOOKS
                      WHERE TYPE = 'SF')
```

Programming interface. A SELECT statement would usually return more than one row of data, the number of rows returned ranging from a few to many. How can a program anticipate the amount of memory required to contain the data returned? SQL provides *cursoring* facilities to solve this problem.

Conceptually, a select statement returns a set of rows. The cursoring functions step through the set one row at a time. A program should first issue a DECLARE CURSOR for a SELECT statement. Then, after a cursor is OPENed, each FETCH statement retrieves one row from the answer set.

The FETCH statement also establishes a current cursor position. UPDATE and DELETE on CURRENT OF CURSOR can be specified to change or delete the row at the cursor position.

After a FETCH, the data are passed from the database to the program in *host variables*, which are variables declared in the host language as data areas for input to or output from the database. They are used directly in the embedded SQL statements. For example, the following embedded SQL statement (embedded in C) contains the two host variables :partno and :partprice.

```
EXEC SQL
SELECT pnumber, price INTO :partno, :partprice
FROM part
WHERE pnumber = 1234;
```

After the execution of this statement, the :partno and :partprice variables in C contain the values retrieved, respectively, from the pnumber and price columns of the part table.

More complicated interactions between the program and the database are achieved using a data area called SQLDA. SQLDA is an architected control block structure containing data and pointers to data. In the example above, the same results can be achieved by an "INTO SQLDA" clause instead of "INTO :partno, :partprice."After the statement is executed, SQLDA contains pointers to the values retrieved.

Another data structure, called the SQLCA, is used to return status information after each query. Perhaps the most important information in the SQLCA is the return code from a query which alerts the program to the success or failure of that query. If a failure occurs, the reason (as a numeric SQL code) is given. Additional error and diagnostic information may also be retrieved.

It is worth mentioning here that the host variable support and the SQLDA and SQLCA structures described above are consistent with DB2 and SQL/DS products. This is an important element of SAA—the programmer can interact with the IBM OS/2 EE Database Manager, DB2, and SQL/DS in the same fashion.

Static and dynamic SQL support. Each SQL statement must be *compiled* by the Database Services component before execution. The difference between static SQL and dynamic SQL is the time at which the compilation is done. For static SQL statements, the compilation is done at precompile or bind time. The compilation is done only once, no matter how many times the statements are executed. For dynamic SQL statements, the compilation is done at run time and must be repeated when the same statements are executed again. Static SQL is therefore more efficient and should be used when possible. However, if the program using SQL must issue arbitrary SQL queries, dynamic SQL is a necessity.

Dynamic SQL is supported in the OS/2 EE Database Manager with PREPARE, DESCRIBE, EXECUTE, and EXECUTE IMMEDIATE statements.

Support of static SQL is a performance feature originally invented in System R. It is implemented in DB2, in SQL/DS, and also in the OS/2 EE Database Manager. For queries in application programs that execute repeatedly, the performance savings can be significant.

Dynamic SQL is supported in the OS/2 EE Database Manager with PREPARE, DESCRIBE, EXECUTE, and EXECUTE IMMEDIATE statements. For example, a program may have a string variable V to store SQL statements at run time. The program can be written as follows:

```
V = 'SELECT LASTNAME FROM AUTHORS';
EXEC SQL
PREPARE S1 INTO :NM FROM V;
```

Data Definition Language. SQL Data Definition Language functions supported in the OS/2 EE Database Manager are listed below:

- CREATE and DROP TABLE: These functions allow the user to add or remove a table from a database. When a new table is being added with CREATE, the table is given a name, and each column in the table is separately defined with a name and a data

type. Column data types supported include CHAR, VARCHAR, LONG VARCHAR, INTEGER, SMALLINT DECIMAL, FLOAT, DATE, TIME, and TIMESTAMP. A column can be defined as Nullable or not Nullable. A character column can be defined as FOR BIT DATA to indicate that it contains binary, not textual, data. A table can have up to 255 columns.

- ALTER TABLE: This function allows the user to add new columns to a table. New columns are added at the "right" of the table—the order of the original columns remains unchanged. In this way, access plans created before the table was altered are not invalidated.
- CREATE and DROP VIEW: Views can be defined on top of existing tables or views. When a table is dropped, views dependent on it are also dropped automatically.
- CREATE and DROP INDEX: Indexes can be created on frequently accessed columns to increase performance. Indexes are stored in B-Tree format with both efficient random access and update characteristics. Indexes can be used to improve performance for queries asking for particular values, for example in 'SELECT ... WHERE age = 35.' Indexes are also used to improve the performance of join operations and to avoid sort. Whether a particular index is used for a given query is controlled by the optimizer in the Database Services component. Indexes can be dropped if they are no longer useful.
- COMMENT on TABLE and COLUMN: Comments can be added to tables and columns.

The data types supported by the OS/2 EE Database Manager listed above are the same as those supported by DB2 and SQL/DS, with the exception of the double byte character set (DBCS) GRAPHICS types. The OS/2 EE Database Manager has internal provisions for adding the GRAPHICS data-type support in a Double-Byte Character Set (DBCS) release of the product (a planned enhancement).

LONG VARCHAR columns of up to 32 700 bytes are supported. Such long fields can be used to store large chunks of data such as graphics, image, or audio data.

Integrated system catalogs, much like those in DB2, are used to store the data definition and relationships in the database. Such catalogs are treated like normal tables, with the exception that only the OS/2 EE Database Manager code can update the catalogs. Users can use SQL DML statements to query the catalogs for information. System catalogs also contain information for the private use of the OS/2 EE Database Manager, such as access plans and database statistics.

Compared to DB2, the DDL functions for controlling the physical placement of data are simplified here. For example, the table space option in DB2 is not supported by the OS/2 EE Database Manager. While this reduces the ability of the user to fine-tune some performance characteristics, the need for a highly trained database administrator (DBA) is also reduced. End users can define the databases themselves with the simplified DDL commands.

Utility and environment API. The OS/2 EE Database Manager has functions that are not part of the SAA database interface. These functions are provided as callable application program interfaces (API) to the data manager. Application programmers can use the API to control the environment and to access the utility functions.

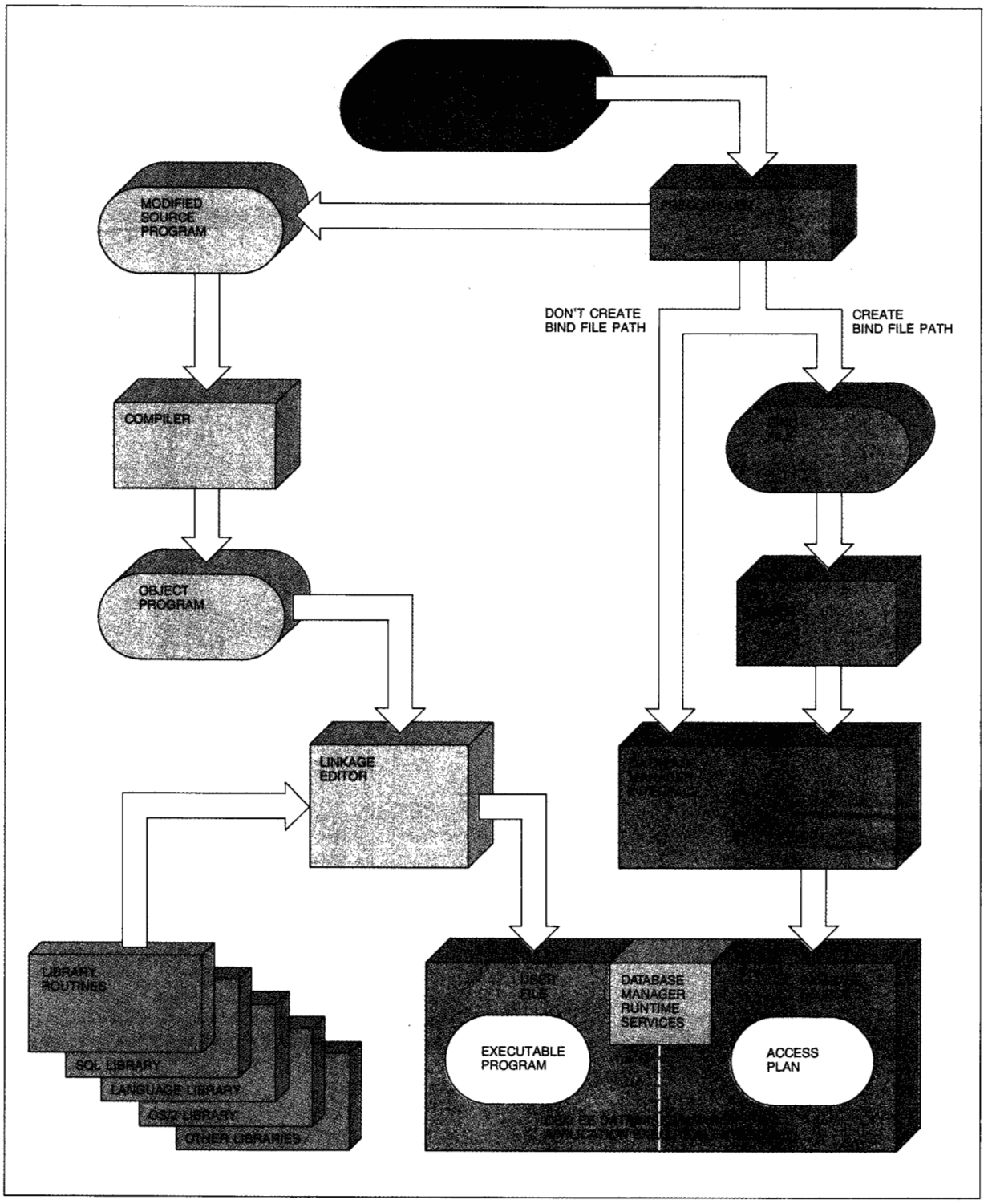
The environment API functions allow an application to create a database, drop a database, connect to (START USING) a database, and disconnect from (STOP USING) a database. Perhaps the most widely used environment API calls are START USING and STOP USING database functions. An OS/2 process must issue a START USING ⟨database name⟩ before any SQL commands can be issued.

Another set of APIs allows applications to obtain information regarding databases and to change the configuration parameters of a database, including RAM size of the buffer pool, RAM size of the lock data structure, and maximum number of transactions allowed. These are parameters the users can set to fine-tune the performance and resource-use characteristics of OS/2 EE Database Manager. For example, one can increase the size of the buffer pool to decrease the number of I/O operations.

Utility API functions allow program calls to IMPORT data generated by other programs into a table or EXPORT a table into one of the supported formats. Other utilities include BACKUP and RESTORE databases, REORG to reorganize the physical structure of a table, and RUNSTATS to update the statistics used by the performance optimizer.

Precompile and bind. The OS/2 EE Database Manager precompiler and binder are tools used by programmers who embed SQL statements in their application programs. Figure 2 illustrates how a source program with embedded SQL can be prepared for execution.

Figure 2 Prepare a program using the precompiler



As mentioned earlier, SQL statements are embedded in a C program with EXEC SQL at the beginning of the statement. Because this is not a "legal" C statement and would cause C compiler syntax errors, it is necessary to precompile the source code. The precompiler replaces the SQL statements with "legal" C statements—calls to the OS/2 EE Database Manager's Database Services functions. The file containing this new source program with the calls is called the *modified source* file; it can be compiled by the C language compiler to produce an executable object.

In addition to the modified source file, the precompiler also calls the Database Services component to analyze the SQL statements and generate *access plans* corresponding to the statements. Access plans are compiled, low-level representations of the SQL statements which contain optimized instructions on how to get the desired data. Access plans are stored in the database under the name of the program being precompiled. Each SQL statement in a program corresponds to a section in the access plan. When a program is executed, the function call replacing an SQL statement causes the corresponding section in the access plan to execute. This is how a program with embedded SQL can be prepared and executed.

Some changes in the database structure, such as the dropping of an index, may invalidate the access plans for precompiled programs. When an invalid access plan is retrieved for execution, the OS/2 EE Database Manager automatically regenerates the access plan (if possible) to suit the new situation. For example, the dropped index would no longer be used in the new access plan. Some changes may make regenerating the access plan impossible. For instance, if a table referenced by the program were dropped, an error would occur on execution.

In the precompile process described above, a program must be precompiled against a database before it can be executed against that database. It is impossible to ship an application program without shipping the source code because the program must be precompiled against the users' databases.

To solve this problem, the OS/2 EE Database Manager precompiler supports an optional *bind* step. The precompiler can optionally generate a *bind file* which contains all SQL statements in the source program and all the related information needed to generate access plans. An application vendor would ship the bind file together with the executable program to the customers, who would then install the program by running the OS/2 EE Database Manager binder process. The binder uses the bind files to cause the generation of access plans in the customers' databases.

Another advantage provided by the separate binder step is the capability for a program to access multiple databases. For example, the personnel records of a company could be segregated into several databases. A program with embedded SQL to generate a summary could be precompiled (and bound) against one of the databases, and then subsequently bound against all the others. The same program could then run against each of the databases, as long as a different START USING were first issued to the database used.

The OS/EE Database Manager supports concurrent access to the same database from different OS/2 processes.

The above discussions refer primarily to static SQL statements. Access plans are also generated for dynamic SQL statements, but these are just place holders that will cause the compilation of the SQL statement at execution time.

Database Services

Multiprocess and multiuser support. It is possible in IBM OS/2 EE to run several processes concurrently. The OS/2 EE Database Manager supports concurrent access to the same database from different OS/2 processes. Transaction commit and rollback features and locking mechanisms in the OS/2 EE Database Manager preserve consistency of data while multiple processes are reading or updating the database. Users can take advantage of this support by running several database applications concurrently. For example, a report program may be running in the background while a data entry function is running in the foreground; a third program updating the database from daily receipt may also be running.

This multiprocess support is the foundation for the Remote Services component (planned enhancement) of the OS/2 EE Database Manager, which supports multiple workstations on a LAN. On the server workstation, multiple processes are created as "servers" to serve the "clients" on remote workstations. The OS/2 EE Database Manager manages concurrent accesses by the processes and ensures consistency of the database with transaction management techniques.

Multiprocess support is also used to support concurrent "demon" processes. For example, a "deadlock detector" demon in the OS/2 EE Database Manager wakes up periodically to detect deadlocks among concurrent transactions.

The OS/2 process synchronization and communication features are used extensively for multiprocess database access. Many control blocks and RAM buffers are shared and updated by different processes, with OS/2 semaphores being used to serialize changes to these shared data structures.

For the following discussions, the term *application process* is used generically to describe each process issuing database commands. In a multiuser LAN access environment, an application process may be a process created by OS/2 EE Database Manager Remote Services, or it may be a local process.

Transaction support. A transaction is a *unit of work* or *unit of recovery* that the user or application program considers to be atomic. All changes to the database in a transaction should either be committed or be rolled back; a database is in an inconsistent stage when a transaction is partially reflected in the database. An example of a user transaction might be the operation of moving some money from a checking account to a savings account. The debit to the checking account and the credit to the savings account should be done as one action; both become a permanent part of the database or neither does. This is a classic example of the role of transactions.

The OS/2 EE Database Manager provides full transaction support in much the same manner as DB2 or SQL/DS. Any reading or writing of the Database Manager database is done within a transaction. An application process that STARTS USING a database and proceeds to issue an SQL command automatically starts a transaction. This transaction can be explicitly ended by the SQL COMMIT or ROLLBACK command. The COMMIT command makes permanent all changes made within the transaction. The rollback command removes from the database all changes made by the transaction. The recovery log (explained below) is used to drive the ROLLBACK operation. A new SQL command after COMMIT or ROLLBACK automatically starts a new transaction.

Locks on data read or changed by a transaction are removed by a COMMIT, making the data available to other transactions. Locks are also removed on a ROLLBACK, and the data unlocked are the same as when the transaction started.

If for some reason an application process ends abnormally while in the midst of a transaction, the process' transaction is rolled back automatically by the OS/2 EE Database Manager. If an application ends normally without issuing a COMMIT or ROLLBACK, the transaction is committed automatically.

The consistency of a database is also protected in the event of a system crash by the recovery process. A recovery log file is maintained of the database changes which are either uncommitted or committed but not on disk, so that the recovery process can restore the database to a consistent state. The consistent state is defined as follows: At the time of the system failure, all transactions that had successfully committed or rolled back are restored, respectively, to that state; all transactions that were in flight (transactions that have made changes but not yet committed or rolled back) are rolled back. Thus, although some work may be lost, the recovery process restores the database to a state in which all changes to data made by committed transactions are reflected in the database.

To return the database to a consistent state after system failure, some of the changes reflected in the log must be undone, while some others must be redone. The OS/2 EE Database Manager keeps track of the status of each transaction and is capable of undoing and redoing the necessary operations. This recovery processing is very similar to that of DB2. A complete description of the transaction recovery process in DB2 can be found in Reference 10.

The recovery log is also used to implement internal save points during the execution of a transaction. The OS/2 EE Database Manager can recover changes to a transaction to a declared save point. There is a class of errors which are not serious enough to require a rollback of the whole transaction; with save points, the operations involved in the error can be backed out and either retried or circumvented by

the transaction. Save points can only be used by the OS/2 EE Database Manager; they are not available to an application.

The effect of the use of save points is made visible at the SQL interface as follows: A save point is declared before the execution of each SQL statement. For instance, if an UPDATE operation runs into an error and cannot continue after having already updated some records, the OS/2 EE Database Manager rolls back only the records changed by that UPDATE statement and returns an error to the calling program. Suppose the error concerns only the SQL statement and is not serious. If the calling program can discriminate between errors fatal to the transaction and errors involving only the SQL statement, it may be able to proceed with the transaction. Applications which use dynamic SQL, such as interactive applications, may find this feature particularly useful, as they have the greatest flexibility.

The OS/EE Database Manager locks logical data objects in a hierarchical manner.

Concurrency. As mentioned earlier, the OS/2 EE Database Manager allows multiple processes to access a database concurrently. The locking system used to maintain data integrity during concurrent processing is discussed in this section.

The locking system of the OS/2 EE Database Manager follows the general concept of locks in System R, SQL/DS, and DB2. The OS/2 EE Database Manager maintains the Repeatable Read (RR) level of data isolation. This means that within a process' transaction, all updates are locked so that other transactions may not access them—this is the heart of the concept of data integrity. In addition all records read are locked so that other transactions cannot change them—this is the essence of Repeatable Read versus other levels of data isolation. Repeating a SELECT query within a transaction provides identical data each time under Repeatable Read, except when the transaction itself updates the data. All data remain locked until the transaction is committed or rolled back.

The OS/2 EE Database Manager locks logical data objects, tables, and records in a hierarchical manner, in contrast to DB2, which locks physical data objects: table spaces and pages. Record-level locking provides finer granularity and better concurrency support.

The SQL command LOCK TABLE [Exclusive Share] ⟨table name⟩ is supported externally. This is the only explicit way in which one may directly lock data within a database. The table specified is locked until the application issues a COMMIT. If LOCK TABLE is not specified, implicit locks are applied automatically.

The level of locking done implicitly (and the level of concurrency that is externally detectable) depends on choices made on the optimization of the SQL DML. The OS/2 EE Database Manager attempts to maximize concurrency in a database by using intention locks at the table level and locking the records individually. For example, a transaction executing an UPDATE statement can lock the table as IX (Intention eXclusive) while locking each record as S (shared). This prevents other transactions from updating the records accessed by this transaction. However, it does not prevent read accesses until the actual record for update is found. Because an UPDATE statement with a WHERE clause typically reads many records before a record is updated, intention locking at record level can increase concurrency significantly.

Concurrent index access is supported by algorithms and data structures which allow high levels of concurrency. Key values and key ranges accessed are held by the accessing transaction until the transaction concludes.

Locks on logical objects are kept in memory with specialized data structures to optimize addition/deletion of locks. The amount of memory dedicated to locks can be specified in the configuration profile. If too many locks are obtained for records in a table, the locks are "escalated" to a table-level lock. This reduces the system resource required for locks.

Deadlocks can occur between OS/2 EE Database Manager transactions holding and requesting locks. These are detected and resolved by a background demon called the *deadlock detector* which is associated with

each database. Whenever a database is being used, the deadlock detector is periodically activated in the background. The deadlock detector looks at the locks in the system and determines whether a deadlock situation has occurred. The user can control the frequency of deadlock detector activity by changing a parameter in the database configuration file.

If there is a deadlock, the deadlock detector selects a "victim" which is a member of the deadlock cycle. The deadlock victim transaction is selected randomly, and the victim transaction is rolled back, allowing other transactions involved in the deadlock to proceed.

The optimizer contains the key technology enabling efficient query execution in the OS/2 EE Database Manager.

SQL compilation and optimization. As mentioned previously, static SQL statements are compiled at precompile or bind time, and dynamic SQL statements are compiled at run time. The compilation of SQL statements is much like the compilation of programming languages: The OS/2 EE Database Manager first parses the statements and performs syntax checks. Data definitions in the system catalogs are consulted for semantic verifications. Optimization and code-generation steps follow.

Instead of executable object code, the OS/2 EE Database Manager SQL compiler generates access plans containing low-level primitive operators. These operators are interpreted at run time by the OS/2 EE Database Manager to perform the actual SQL query functions.

Before the access plan of a query is generated, the OS/2 EE Database Manager optimizer first analyzes the query. Statistics kept in the system catalogs, such as the size of the table and the number of distinct values in an index, are consulted. Decisions on the actual *access path*, or strategy for accessing data and executing the query, are then made. Important choices made by the optimizer include the following: whether indexes should be used, in what order a join should be performed, what join algorithm should be used, and whether and when a sort is appropriate. The optimizer contains the key technology enabling efficient query execution in the OS/2 EE Database Manager.

Specialized sort algorithms are used in the OS/2 EE Database Manager for optimized performance in the relational database environment. The algorithm is very flexible in terms of RAM usages and temporary storage usages. The query optimizer has a great deal of freedom in controlling sort operations based on overall performance and resource utilization considerations.

Storage management. Conceptually, Database Services SQL compiler and optimizer functions are in a subcomponent similar to the Relational Data Services (RDS) component in System R. The physical data structures are managed by a subcomponent similar to the Relational Storage System (RSS) component in System R. However, the RSS interface in the OS/2 EE Database Manager is not a rigid one. For performance reasons, RDS and RSS in the OS/2 EE Database Manager are tightly integrated.

The OS/2 EE Database Manager uses the OS/2 file system to support a database. Generally, each table is stored in a single OS/2 file. In addition, all indexes for the same table are stored in a separate OS/2 file. Each of these files is internally divided into 4K pages, which is the standard I/O unit of the data manager. Space allocation is handled within each page. Long fields are handled separately to optimize the allocation of field sizes up to 32700 bytes in length. All long fields for a table are stored together in a single OS/2 file.

When a database is created, a special subdirectory for that database is formed, and all OS/2 files making up the database are stored in that subdirectory. Databases may not span a media partition—all of the OS/2 files making up a database must be on one drive letter in the special subdirectory created for that database. The database files are not meant to be used in any way except by the Database Manager, as any change to these files by other means could have catastrophic results on the integrity of the data.

The OS/2 EE Database Manager allocates a separate memory area, called the buffer pool, for each database in use. A buffer pool consists of multiples of 4K pages of space and is used by the OS/2 EE Database Manager to read from or write to the disk. The buffer pool is used as a *cache* area for the database data. In general, a Least Recently Used (LRU) algorithm is used for buffer replacement. The oldest, or least recently used, data page is replaced by a new page. Frequently used pages such as index nodes or "hot" data therefore tend to stay longer in the buffer pool, reducing the overall number of disk I/Os.

A write-ahead logging scheme is used in the OS/2 EE Database Manager. To ensure the integrity of data, the log is always written to the disk before the data page updates are written. This means that the data pages can stay in the buffer pool even after updates. Frequent accesses and updated data pages can therefore stay longer, again reducing the need for disk I/O.

Security. The OS/2 EE Database Manager supports password security at the database level. When a database is created, a database password can be assigned. (If password security is not desired, a NULL password indicates NO SECURITY.) On connection to a database (via the START USING command), a valid password is required before access is allowed. The password of a database can be changed via the ALTER DATABASE command.

National-language support. The OS/2 EE Database Manager provides National-Language Support (NLS) by isolating the end-user interactions (menus and messages) into separate files that can be translated. In addition, special consideration is given to functions that require special adaptation in different countries, such as date/time formats, collating sequences, and monocasing rules. In general, Database Manager NLS functions are built on top of the NLS functions in the base OS/2 operation system. For example, the collating tables and monocasing tables obtained through DOSGETCOLLATE and DOSCASEMAP are used by OS/2 EE Database Manager to support its sorting functions and monocasing functions. The SQL language itself is not translated into national languages. However, provisions are supported by the OS/2 EE Database Manager for identifiers in SQL (table names, view names, and field names) to be defined in national languages.

The OS/2 EE Database Manager is also enabled for Double-Byte Character Set (DBCS) language translation. DBCS enabling allows the support of a mixed one-byte and double-byte internal coded character set. IBM organizations in Asia can use this as a base for providing national-language support for selected Asian languages.

Remote Services (planned enhancement)

The Remote Services component of the OS/2 EE Database Manager supports the distributed database functions on a LAN. It permits applications to issue remote SQL requests against remote databases on a LAN. All SQL functions can be executed against a remote database as if the database were in the local workstation.

Before a database can be accessed through Remote Services, the name and location of the database must first be cataloged. The OS/2 EE Database Manager provides *catalog database* and *catalog node* function calls to achieve these functions.

When a START USING database request is accepted by the OS/2 EE Database Manager Remote Services, the Remote Services requestor first performs the routing function by consulting the catalogs. If the database is remote, the requestor then passes the calls through the IBM OS/2 EE Communication Manager (APPC protocol) to the Remote Services server where the database resides. The Remote Services server then issues requests to the database. Any returned data or return codes from the database are passed back to the requestor. The requestor returns the information to the application in the same manner as the OS/2 EE Database Manager would. Therefore, the application does not need to know whether the database being accessed is local or remote. No changes to the application code are necessary to run on a local OS/2 EE Database Manager or a remote one.

This design makes it possible to write a single application to run on different system configurations. There is complete flexibility on how the LAN should be configured in terms of database usage. It is possible to put a database on a machine with more DASD and RAM resources as a server, and run applications from smaller machines. It is also possible to distribute the databases to different machines and allow cross-accesses when necessary.

The robust nature of APPC protocol permits the OS/2 EE Database Manager to guarantee database consistency across the LAN. If a remote site fails, the transaction for that application is rolled back and the

database is not harmed, because the Remote Services server is aware of the requestor status at all times through the APPC protocol.

Remote Services uses a client/server model to redirect the SQL-level database request using APPC. A complex SQL query can be sent once over the LAN. All query processing happens on the server, and the answer of the query is passed back. Compared to an alternative approach of redirecting file I/O through the OS/2 EE LAN Requestor (available in Version 1.1 of IBM OS/2 EE), the Remote Services approach tends to reduce the LAN traffic significantly. Remote Services does not use the LAN requestor nor require its installation.

In terms of future growth, the design adopted in the OS/2 EE Database Manager provides a good foundation for future extensions in distributed database functions. When the request is at the database request level, the Database Manager retains control of how to process the query. An example might be a query that joins two tables from two databases on different nodes. With database requests, the Database Manager can decide where the join operation should be performed and what data should be sent where. The potential benefit of such global query optimization is clearly high.

Conclusions

The OS/2 EE Database Manager, which was developed entirely within IBM, employs many IBM unique relational technologies. It was developed in parallel with the IBM OS/2 operating system and has been optimized for the OS/2 environment. The OS/2 EE Database Manager provides high performance, full function, and robust data integrity support. It is consistent in function with and similar in architecture to DB2, SQL/DS, and QMF, so that tightly integrated distributed database support among the IBM relational family of products will be easier to add in the future.

The OS/2 EE Database Manager, together with other functions in the OS/2 Extended Edition, provides a high-function platform for users and application developers alike. It also provides a strong foundation for future growth.

Cited references

1. E. F. Codd, "Relational database: A practical foundation for productivity," *Communications of the ACM* **25**, No. 2, 109–117 (February 1982).
2. D. D. Chamberlin et al., "SEQUEL 2: A unified approach to data definition, manipulation, and control," *IBM Journal of Research and Development* **20**, No. 6, 560–575 (November 1976).
3. M. M. Astrahan et al., "System R: Relational approach to database management," *ACM Transactions on Database Systems* **1**, No. 2, 97–137 (June 1976).
4. S. Kahn, "An overview of three relational data base products," *IBM Systems Journal* **23**, No. 2, 100–111 (1984).
5. *IBM Systems Application Architecture: An Overview*, GC26-4341-0, IBM Corporation; available through IBM branch offices.
6. *IBM Systems Application Architecture: Common Programming Interface, Database Reference*, SC26-4348-0, IBM Corporation; available through IBM branch offices.
7. S. L. Watson, "OS/2 Query Manager overview and prompted interface," *IBM Systems Journal* **27**, No. 2, 119–133 (1988, this issue).
8. B. Lindsay et al., "R*: A distributed database manager," *ACM Transactions on Computing Systems* **2**, No. 1, 24–38 (February 1984).
9. B. Lindsay et al., "A database management extension architecture," *Proceedings of ACM SIGMOD 1987*, San Francisco (May 1987).
10. R. A. Crus, "Data recovery in IBM Database 2," *IBM Systems Journal* **23**, No. 2, 178–188 (1984).

Philip Y. Chang *IBM Entry Systems Division, Austin Laboratory, 11400 Burnet Road, Austin, Texas 78758.* Dr. Chang is a senior programmer in the Austin Laboratory and a lead architect of the OS/2 EE Database Manager project. He joined IBM in 1979 as a lead designer for Displaywriter software, and managed several projects related to Displaywriter follow-on work and the Displaywrite software on PC. Dr. Chang joined the OS/2 EE Database Manager project at its inception in 1984 and has held both technical and management positions within the project. Before joining IBM, Dr. Chang taught in the Computer Science Department at the University of Texas at Austin between 1976 and 1979. He holds a B.S. degree in electrical engineering from the National Taiwan University and a Ph.D degree in computer science from the University of Utah.

William W. Myre *IBM Entry Systems Division, Austin Laboratory, 11400 Burnet Road, Austin, Texas 78758.* Mr. Myre is a technical planner for the OS/2 EE Database Manager project. He joined IBM in 1981 as a programmer on the IBM 5520 office system in the document distribution area. In 1984, he became part of the initial design team of the OS/2 EE Database Manager and did significant portions of the locking, transaction, and recovery subsystems design. Mr. Myre received a M.A. degree in computer science from the University of Texas at Austin in 1980 and a B.S. degree in computer science and mathematics from Vanderbilt University in 1977.

Reprint Order No. G321-5313.