IBM Database 2 overview

by D. J. Haderle
R. D. Jackson

IBM Database 2 (DB2) is a data base management system that supports the relational model of data. This paper presents the major features of DB2 and discusses its architecture and the relationship of DB2 with the host operating system. These principles are illustrated by an example.

IBM Database 2 (DB2) is a data base management system that is available in Multiple Virtual Systems/370 (MVS/370)[1] or Multiple Virtual Systems/Extended Architecture[2] (MVS/XA) which supports the relational model of data.[3-5] The user interface is a modified form of the Structured Query Language (SQL),[6,7] which was developed at the IBM San Jose Research Laboratory for use in System R,[8-10] a research prototype of the relational model.

Access to data is supported concurrently from Customer Information Control System/Operating System/Virtual Storage (CICS/OS/VS),[11] Information Management System/Virtual Storage (IMS/VS),[12] and Time Sharing Option (TSO) foreground, and TSO background. The Query Management Facility (QMF), a separate product, provides interactive users in the TSO environment with access to DB2 data and report generation facilities. Data Extract (DXT), another IBM product, permits users to extract data from Data Language/I (DL/I) data bases, Virtual Storage Access Method (VSAM) files, or physical sequential files, and to enter those data into DB2.

DB2, QMF, and DXT may be installed on an Extended Architecture (XA) or non-XA MVS system. In the XA environment, extended virtual storage addressing is utilized to support additional data buffers and working storage. This allows DB2 to support additional concurrent transactions and/or ad hoc query users.

The architecture of DB2 evolved from System R, IMS/VS, and new technology. The Relational Data System (RDS), an adaptation from System R, decomposes and manages the evaluation of each SQL request. The Data Base Manager (DBM) manages access to stored data and guarantees its consistency, and the Buffer Manager (BM) manages the virtual storage buffers and the transfer of data to and from virtual storage and Direct Access Storage Devices (DASD).

DB2 provides services to manage the connections from CICS/OS/VS, IMS/VS, and TSO environments. All work in DB2 is performed within the scope of a transaction. Transaction recovery is coordinated within DB2 by a set of protocols that guarantee that all DB2-managed resources changed by a transaction are either committed or rolled back, based on the disposition of the transaction. Log Management services provide for the recording of information to permit recovery. Lock Management services allow DB2 to determine which transactions can use common data base resources concurrently and queue those that cannot. The main data base management features of DB2 are shown in Figure 1.

This paper, which provides an overview of DB2, first describes the major features of DB2. The second section provides an overview of architectural considerations and of the product's relationship with the host operating system (MVS). Finally, we give a view of the implementation by chronicling the life of a data base request.

Figure 1 DB2 overview

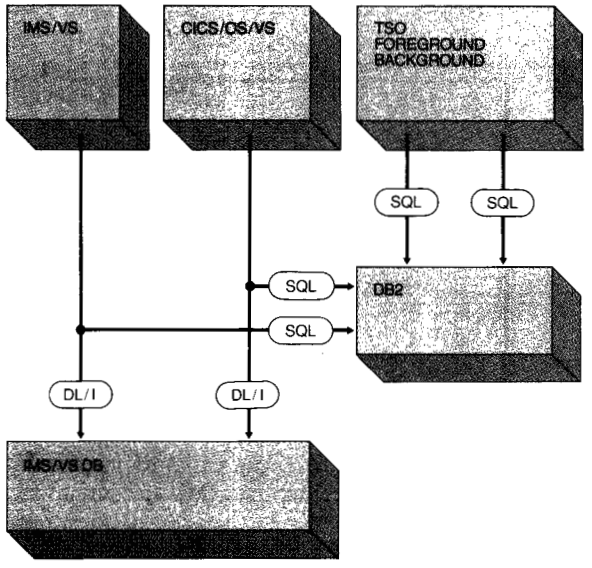


Figure 2 EMPLOYEE table

EMPNO	NAME	DEPTNO	PHONENO
000010	PANGBORN, HARRIET	M12	555-5320
000020	ALLISON, GAIL	M08	555-8198
000040	STEPHENSON, WALTER	D11	555-5200
000060	STEPHENSON, SUDY	M12	555-0413
000090	BLAKE, BILL	D11	555-0180

Figure 3 DEPARTMENT table

DEPTNO	DEPTNAME	MGRNO
M12	GRAPHICS CENTER	000160
M08	FACILITIES MAINTENANCE	000020
D11	PLANNING	000090

Overview of DB2 features

Relational data model. Relational data base concepts are well defined elsewhere,[2] but are presented here as they relate to DB2. Data are represented in the form of *tables* (see Figure 2). Each *row* (record) of a table consists of a set of *columns* (fields). All rows of a given table are of the same form. A *base table* defines a stored table (see Figure 3). Each base table identifies a single *table space*, which describes a set of Direct Access Storage Device (DASD) files to be used for the storage of rows of that table. More than one base table may identify the same table space.

All operations on data are performed using Structured Query Language (SQL). The language, which was initially developed at the IBM San Jose Research Laboratory,[2] has been modified to its present form for common usage by any IBM product. At present, SQL/Data Systems (SQL/DS) and DB2 support the SQL language. Statements in that language may be specified within COBOL, PL/I, FORTRAN, or Assembler application programs. An interactive subset of the language may be specified from a terminal via the Query Management Facility (QMF) for immediate execution. The single language permits the definition, authorization, and manipulation of data. Control statements (e.g., COMMIT) are specifiable when DB2 is managing the transaction. In instances where a transaction is managed by another connection (such as IMS or CICS), the application must request those services from the appropriate transaction manager.

The set of data to be retrieved is specified via a SELECT statement. In its simplest form the statement identifies every row of the named table for retrieval, as in the following example:

```
SELECT NAME, PHONENO
       FROM EMPLOYEE
```

The SELECT statement identifies the table (EMPLOYEE) and the columns of the table (NAME, PHONENO) to retrieve. A predicate may be specified as illustrated by the following example:

```
SELECT NAME, PHONENO
       FROM EMPLOYEE
       WHERE DEPTNO = 'D11'
```

The effect here is to cause a portion of the rows to be returned (that is, those rows with the value of D11 in the DEPTNO column). Multiple predicates connected by the Boolean operators OR, AND, and NOT may also be specified.

The predicate may specify a SELECT from another table, thereby retrieving a value or a set of values for comparison, as in the following example:

```
SELECT NAME, PHONENO
       FROM EMPLOYEE
       WHERE DEPTNO =
       (SELECT DEPTNO
       FROM DEPARTMENT
       WHERE DEPTNAME = 'Planning')
```

In this SELECT example only those rows in the EMPLOYEE table that report to the planning department are returned.

Data may be retrieved from multiple tables with a single statement, as shown in the following example:

```
SELECT NAME, DEPTNAME
       FROM EMPLOYEE, DEPARTMENT
       WHERE EMPLOYEE.DEPTNO = DEPART-
       MENT.DEPTNO
       ORDER BY DEPTNAME
```

This example illustrates a *join* of rows from the two tables, EMPLOYEE and DEPARTMENT. A resultant row from the SELECT contains the NAME of an employee and the associated DEPTNAME of the department to which the employee belongs. The selected rows are returned in ascending sequence on the DEPTNAME values, as specified by the ORDER BY clause.

Functions are provided that operate on a set of data values. For example, the SUM function may be specified in a select clause [SELECT SUM(column name)] that identifies a column in the table. For the rows in the table identified by the SELECT predicate, the values in the specified column are added together and returned by the SUM function.

The INSERT statement places a single row in a table or identifies a set of rows in another table (using a SELECT) to copy into the target table. The DELETE and UPDATE statements delete and modify one or more rows, respectively. The DELETE and UPDATE statements may identify the set of rows using a SELECT clause or they may optionally operate on a single row after it has been retrieved.

In an application program, the SELECT statement is associated with a cursor. The application program OPENS the cursor and iteratively issues the FETCH statement, which returns one row for each FETCH request. In an interactive query environment (QMF), the user specifies the above SELECTS, and the interactive query application effects the retrieval using OPEN and FETCH.

Access to data. Rows are stored in DASD data sets. Data are written in 4K blocks, called *pages*, which correspond to the operating system page size. Data can be accessed by sequentially reading all the data in a table space, applying predicates, and returning only selected rows.

An index may give beneficial performance in processing a join and many other SQL query forms.

Sequential scanning techniques are inefficient for queries that retrieve only a few rows in a large table space. Users may define one or more indexes on a base table using SQL, thereby identifying a set of column names and ordering information. For example, CREATE INDEX DEPTIX ON DEPARTMENT (DEPTNO ASC) defines an index on the department table, ordering the values of the DEPTNO column in ascending sequence.

Each index resides in its own data set separate from the table data. For each row in the table, an entry exists in the index that contains the ordering column values from the row and the address of the data record. Each index is maintained current with the data, thus providing fast access for queries that are predicated by the index columns or request the data to be returned in an order that matches the index ordering. The use of an index may give beneficial performance in processing a join and many other SQL query forms.

In all cases, DB2 determines the method for evaluating a query and, thus, whether an index is to be used.

Data independence. If the data must be ordered, and a matching index does not exist, DB2 sorts the data. An index may be added that maintains the order desired by the query. In this case, DB2 accesses the data via the index, without having to sort the results.

Users cannot specify usage of an access path directly. Furthermore, DB2 can resolve any query using table space scans only. This allows the installation the flexibility of tuning the system without affecting application code, because the applications are insensitive to the existence of indexes.

Authorization. Authorization of users to data and applications is an integral capability of the DB2 system. The creator of a table may identify a set of users who may access the table for retrieval or update. One or more VIEWS may be defined on a table or set of tables. The VIEW statement that follows contains a SELECT statement referring to the EMPLOYEE table, naming a set of columns and containing a WHERE clause that determines which rows are observable via the VIEW.

```
DEFINE VIEW PLANNING_EMP AS
  SELECT EMPNO, NAME
         FROM EMPLOYEE
         WHERE DEPTNO = 'D11'
```

Inasmuch as the result of an SQL query is itself a table, the VIEW name may be used as a table name in any SQL statement. By granting other users access to data via a VIEW, rather than directly to the base table itself, users may restrict access to a subset of the rows and columns of the table. VIEWS may be created on other VIEWS.

The construction of an application includes encoding SQL statements in one or more programs.

The VIEW mechanism extends the notion of data independence, thereby permitting queries to be independent of the current stored table(s). If a base table is split into two tables, a VIEW that joins the two new tables may give the appearance of the original table. By using the original table name as the VIEW name, existing queries may be salvaged. Data are not changed, however, by using a VIEW that joins tables.

The DB2 catalog. The DB2 *catalog* is a set of tables that are created and maintained by DB2. These tables describe all the objects that DB2 manages. This description includes data definitions (e.g., tables and columns) and authorization data. Authorized users may query the catalog using SQL.

Application processing. The construction of an application includes encoding SQL statements in one or more programs. DB2 provides a process developed for System R that extracts the SQL statements from the application programs prior to their compilation and decomposes the statements into a form that is more efficient for processing at statement execution. This process is incorporated within language preprocessors that are provided as part of DB2, the DB2 SQL Precompiler, and the DB2 BIND Processor.

Each SQL application source program must be precompiled using the appropriate DB2 language preprocessor. The SQL source statements are extracted, placed in an Operating System (OS) file, and replaced in the host-language programs with control transfer statements that are understood by the host language. At statement execution, control is transferred to DB2, thereby providing the identification of the statement to be executed and the host-language variables and data locations (e.g., addresses into which data are to be selected) to use.

At precompilation, there is no communication with DB2. This permits users to precompile and compile SQL application programs on processors without dependence on the availability of the DB2 data base management subsystem.

After extracting each SQL statement from the application source program, the preprocessor parses each SQL statement to ensure syntactic validity. Optionally, names in the statement are checked for correct context usage. For example, table names are specified where table names should be specified. A function called DCLGEN is provided by DB2 to extract table definitions from the DB2 catalog and produce host-language declarations. These declarations can be included in the application program for documentation. They also provide the description of the data bases needed by the preprocessor to check referenced object names.

Prior to executing an application, the user must define a *plan*, using an on-line DB2 command called BIND that identifies all the SQL programs (by naming the OS files containing the extracted SQL statements) that may be invoked while processing the application. BIND saves each statement source string in the DB2 catalog. Each statement is checked for proper syntax. For nonmanipulative statements, the parsed statement representation is stored in the system data base for use at statement execution.

For a manipulative statement, all objects named in the statement are resolved to ensure that they are defined to DB2 and are used in the correct context (e.g., table name and column names in tables). If tables or columns have not yet been defined, the binding of the statement can be deferred until statement execution. If this option is chosen, however, the statement will be bound at each transaction execution that uses the statement. Also, the definer of the plan is checked for proper authority to execute the statement. Various methods for performing each statement are determined and the least-cost method is selected. This includes selecting access paths to the data, such as by way of an index. Executable code and structures are generated and stored in the system data base for use at statement execution, and each statement's dependencies on objects and authorization are registered in the DB2 catalog. If a dependent object is dropped or if authorization is revoked, the generated output of the plan definition becomes invalidated. A request for its execution causes the statements (which had previously been saved in the DB2 catalog) of the plan to be rebound automatically for execution.

This process, called *binding*, allows DB2 to minimize the cost of executing each statement by performing the operation just described only once, rather than each time an application is processed. SQL is not simply interpreted; it is parsed and acted upon. In some cases, code sequences are formed and saved for execution-time processing.

Query processing. DB2 and QMF query processing capabilities use a *dynamic SQL* facility provided within DB2. The SQL statement PREPARE passes the user's query (which is also an SQL statement) to DB2 at execution. The statement is prepared for execution using the previously described process, but omitting the registration of dependencies. The results of PREPARE are generated code and structures to be used when the prepared SQL statement is executed, and information describing data which must be supplied on statement execution. An example of the latter might be the location in which to place the selected data. SQL statements processed in this fashion are called dynamic SQL.

Data integrity. DB2 provides complete integrity of data bases with the following forms of recovery: *Unit of Recovery, emergency restart,* and *media recovery.*

Unit of Recovery. All DB2 data bases are recoverable. On the first request from an application that changes a data base, DB2 creates a *Unit of Recovery (UR).* The UR persists until the application commits, aborts, or terminates the process. If the application terminates the process, DB2 or the host transaction manager determines the disposition of the UR, i.e., whether to commit or abort the changes.

Until disposition of the Unit of Recovery, all changes are visible to the changing application only. Further, no changes are written to DASD until information has been recorded in a recovery log that permits DB2 to undo the changes. This is required because DB2 may write the data to DASD prior to the termination of the UR. Also, to guarantee successful commit and reconstruction of the data following a media failure, information has been recorded in a *recovery log* that permits DB2 to redo the changes. This information is placed on a *physical log* before DB2 agrees to commit the changes.

All data base operations can be performed while DB2 is operational.

Emergency restart. A catastrophic error can occur that causes DB2 to terminate abnormally. Catastrophic errors might be power or hardware failures. When DB2 is started, it recognizes its prior state from the recovery log, and thus it initiates recovery. All data bases are recovered up to the point of failure minus the activity from any Units of Recovery that had not been committed up to that point.

Media recovery. When an error related to the DB2 data base external storage media has occurred, the user can initiate media recovery procedures for each table space, file, or a portion of the table space for which errors have been detected. Generally, the user must back up table spaces on a periodic basis, using a DB2 utility service called *image copy.* The initial image copy copies all the data in the table space or portion of the table space to another data set, and registers in DB2 catalog tables the name of the data set to which the data were copied and the current

position in the recovery log. The user can submit subsequent image copy requests that copy all the data or only data that have changed since the prior image copy request. Copies of the changed data are called *incremental image copies*. These are also registered in the DB2 catalog. When recovery is required, the user invokes the RECOVER utility, which consults the catalog, restores the data to the latest complete image copy, adds subsequent incremental image copies, and applies recovery log changes from the point of the last image copy to the present.

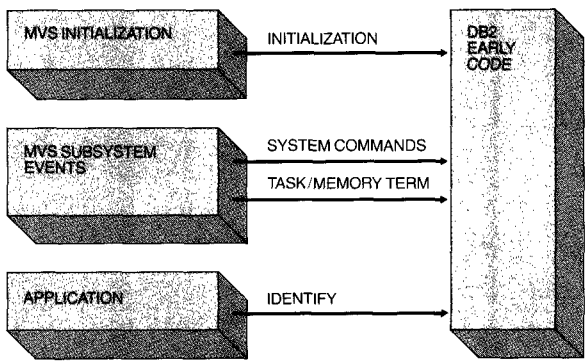
Availability. All data base operations can be performed while DB2 is operational. That is, no data base operation requires DB2 to be terminated. These include all of the traditional data base administration activities:

- Data base tables may be defined or deleted. Existing tables may be modified by adding new columns. (While these activities are being performed, applications may access unrelated tables.)
- New applications may be introduced to DB2, and old applications may be modified without interrupting other applications.
- Data backup retrieves the data from a table space and copies them to another data set. All backup may be performed concurrently with access and alteration of the data.
- Data recovery requires exclusive use of the table space. The image copies and the log are used to recreate the data. All other table spaces are accessible during recovery.
- Data reorganization may be performed to store the physical data in a specified sequence. This may provide better performance for queries that conform to the sequence. Reorganization requires exclusive use of the table space that contains the data being reorganized, but other processing of unrelated data may occur.

The recovery log, which is integral to the performance, availability, and integrity of data bases, consists of a set of data sets that reside on DASD. These data sets are called the *active log*. When a data set becomes full, it is archived to tape or mass storage. The data set is eligible for reuse when the archive process is through. Meanwhile, DB2 selects the next data set of the active log set for logging on-line data. Multiple levels of storage provide fast logging for applications, relegating data to less costly storage as it ages.

To allow for the possibility of recovery log media failure, DB2 provides dual logging. That is, the entire log (active and archive) may be duplicated. When access to one log copy encounters an I/O error, DB2 accesses the data on the other copy.

Figure 4 DB2 EARLY code resident in MVS



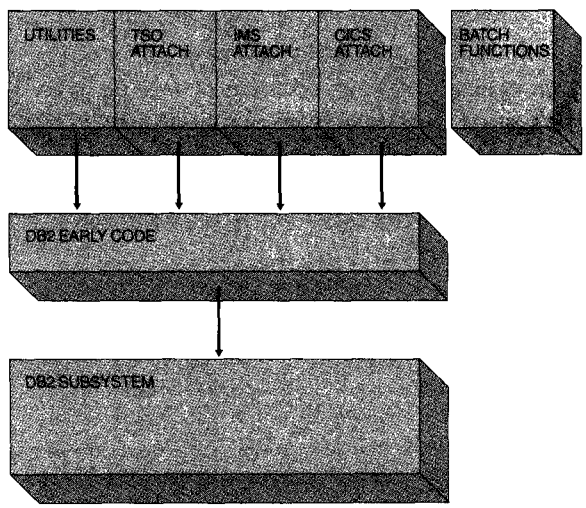
Architectural considerations

DB2 as an MVS subsystem. At MVS initialization, DB2 EARLY code resident in MVS is invoked, as shown in Figure 4. DB2 EARLY code establishes DB2 as an MVS subsystem, which makes DB2 eligible to receive notification of certain system events and provides an interface whereby applications may locate and communicate with DB2. The EARLY code receives all system commands by way of the MVS Subsystem Interface (SSI). The EARLY code can accept or reject each command, thereby permitting other subsystems to process them. DB2 commands are identified by a unique first character that is specifiable by the installation (e.g., -DISPLAY DATABASE). The overall architectural structure shown in Figure 5 is now presented functionally.

The EARLY code listens for a START DB2 command, which causes the initiation of the loading process and activating the code (called the DB2 subsystem code). This provides all of the data base management services. Two address spaces are created: (1) one space for system service components (those with no dependency on data bases), and (2) the other space for data base components. Successful initialization of the components is followed by restart processing, which disposes of any work that was in progress at the prior termination of DB2, thereby ensuring that data are consistent.

After the subsystem has been started, the EARLY code supporting command functions passes the inter-

Figure 5 DB2 structural overview



cepted commands directly into the command processing functions of the subsystem. During the period of time when the subsystem is being started, the EARLY code queues commands, thus making them available to the subsystem when it is ready to process them.

Before an application or another subsystem (such as IMS or CICS) can perform any data base activity, a line of communication must be established between it and the DB2 subsystem. DB2 code residing in the application address space (called *attachment code*) receives data base requests directly from the application. The attachment locates the DB2 subsystem via the Subsystem Interface (SSI) and establishes a line of communication by directing an IDENTIFY request to the EARLY code. When the DB2 subsystem is stopped, the EARLY code responds negatively to the request and optionally queues a notify message to be subsequently returned to the requestor after the subsystem is started. Once the subsystem has been started, IDENTIFY requests are passed to it, and connection processing is begun.

After a connection has been established, EARLY code monitors that application environment for terminating conditions by listening for task and memory termination events. If the application terminates before disconnecting and disposing of any outstanding unit of recovery established with DB2, the termination event handlers perform the necessary work.

The majority of the code and control structures that provide on-line access and management of data reside in one of the two subsystem address spaces. This packaging is the result of three major influences.

DB2 manages concurrent access to data by multiple applications. The reliability of DB2 does not depend on the reliability of any application that uses DB2 services. The isolation of vital control structures in address spaces apart from applications isolates DB2 itself from failures in those address spaces and from their availability.

Also, a lack of virtual storage in the non-MVS/XA environment prevents any new product from making large claims for storage common to all address spaces or storage in an application address space. The on-line DB2 code, without considering control structures, exceeds three megabytes, a quantity not generally available in non-MVS/XA application address spaces.

Finally, some work performed by the subsystem is not directly related to any application and may be scheduled and managed under tasks within the subsystem. For example, the log archive function is automated and managed within the subsystem. It is performed asychronously with respect to application work under a subsystem task.

The creation of two address spaces rather than one simply provides additional virtual storage, which is necessary in the non-MVS/XA environment.

Dispatching work. Multiple address space configurations demand an efficient mechanism for cross-address-space transfer of control and transfer of data. This is provided by MVS cross-memory services coupled with processor instructions that support multiple address spaces. Three distinct address spaces may be distinguished by control registers.

The first, designated as *home*, is the address space that contains the dispatching unit [i.e., Task Control Block (TCB) or Service Request Block (SRB)]. The dispatching of work honors the priority of this address space and the relative priority of the execution unit in it. The second address space, designated as *primary*, is used by the processor to fetch instructions. Unless specified, all data are also fetched and stored from this address space. The third address space, designated as *secondary*, is sometimes used by the processor to fetch and store data. If secondary addressing has been enabled or if certain instructions that explicitly specify the usage of secondary addressing are executed, the processor addresses data via the

secondary address space. In this case, one may execute a single instruction (i.e., MVCP or MVCS), which moves data between primary and secondary address spaces.

Using this technology, DB2 executes an application's request under that application's execution unit. When it is necessary to execute code in one of the DB2 address spaces, a Program Call (PC) instruction is issued that causes primary addressing (instruction fetch) to change to the appropriate DB2 address space. This retains priorities already established for applications in the system. Further, it avoids dispatching time to establish addressability to code in another address space. This process is illustrated in Figure 6.

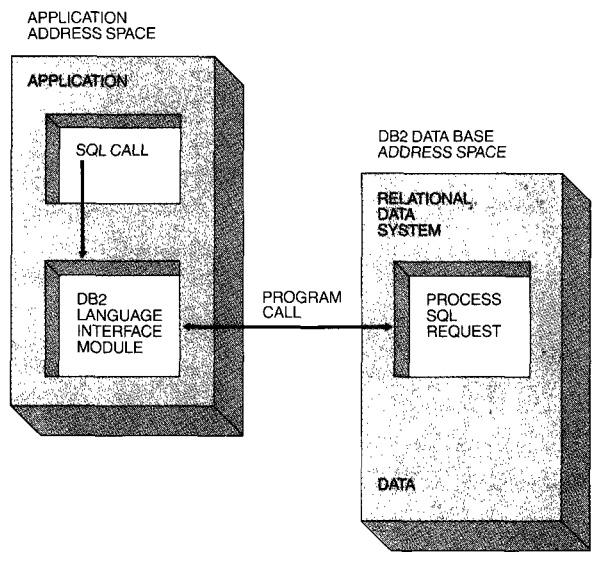
Some MVS functions (e.g., scheduler allocation) cannot operate within a cross-memory environment. That is, they require that home and primary address space be the same. When it is necessary to perform such a function, DB2 suspends work on the current execution unit and initiates processing under a task or SRB in a DB2 address space. This is called *execution unit switching.* Use of such functions is infrequent.

Isolation from errors. Certain vital processing must complete without failure to prevent making data or the entire subsystem unavailable. This includes the process of committing and backing out application changes. Should these processes fail, either the changed resources become inaccessible until they can be recovered, or, if the resources cannot be isolated, the subsystem terminates to prevent loss of data integrity. Termination is triggered so that the restart process, performed when DB2 is initialized, processes the recovery log and brings all data and units of recovery to a consistent state.

To isolate these operations from termination externally induced (e.g., job cancel) in certain host environments, DB2 switches to an execution unit in the system services address space to perform the function. Termination of the application task does not interrupt the processing of the execution unit performing the must-complete function.

Asynchronous activity. An application event or a time interval elapse can trigger the initiation of asynchronous work. To affect the scheduling, DB2 places a work element on a queue and notifies a task in the subsystem that work exists. Archiving the active recovery log is an example of this class of work.

Figure 6 Function invocation



Transaction management. DB2 data base services are available to applications in the supported host environments (i.e., TSO, IMS, and CICS) within the framework of a transaction. A transaction is a bounded set of actions (e.g., data base and data communications) that must succeed (commit) or fail (abort) as a unit. The transaction manager notes the start of a transaction by assigning it a unique identifier in the system so that all work subject to commitment can be related to that transaction. The transaction manager also acknowledges the end of the transaction, which can be signaled by an application request to commit or abort the work or by the termination of the application performing the work. When termination is signaled, the transaction manager coordinates commit and abort by informing all interested parties of the event so that they may properly dispose of all scheduled changes to their resources.

IMS and CICS provide transaction management. Each maintains its own log on which it records transaction-related events. DB2 maintains its own log on which it records transaction control information (e.g., the identifier IMS has assigned to a transaction that DB2 can query after failure to determine disposition) and data base changes (undo/redo). When the transaction terminates, IMS and CICS coordinate the commit or abort for transactions initiated in their environments among the DL/I data base manager, the DB2 transaction manager, and the data communication manager. DB2 is simply a participant

Figure 7 TSO application invocation

and has no knowledge of the other parties. The DB2 transaction manager notifies all interested DB2 components of the outcome and ensures their completion.

Because TSO provides no transaction management, DB2 assumes those responsibilities in a TSO environment.

Connection protocols. To establish and manage transactions, the DB2 subsystem provides a set of protocols that are implemented in the host attachment code transparent to applications. An IDENTIFY request is issued prior to any other request to establish a connection with DB2 and to identify the characteristics of the host environment. This specification determines DB2 behavior most notably in the areas of transaction management (coordinator versus participant role) and authorization administration.

When the host environment has been declared, transactions using DB2 services can begin. Hosts can support multiple concurrent transactions from the same address space and execution unit. To tie together all of the requests and resources associated with a single transaction, DB2 uses a set of constructs, collectively called a *thread.* At transaction initiation or on the first SQL statement, the attachment invokes DB2 to create a thread, naming the plan to be used by the transaction. The plan identifies the total set of capabilities (SQL statements) that the application may exercise. The DB2 subsystem creates the structures it needs to manage the transaction and returns the anchor address for the thread structures to the attachment which will provide it as a tag on all SQL requests associated with this transaction.

To demonstrate this in the TSO environment, Figure 7 gives a list of TSO commands entered to run two DB2 applications. The DSN command results in an IDENTIFY to DB2. The subsystem name (DB2PROD) is provided, since it is possible to have more than one DB2 subsystem (e.g., test and production) concurrently active in MVS. They cannot share data.

The RUN command names the application program (APPL1) to be given control. The RUN command handler saves the name of the plan to be used (ACCOUNT) and gives control to the application program. On the first SQL statement issued, the TSO attachment requests that the DB2 subsystem create a thread for the named plan. The TSO attachment then passes all SQL requests to the DB2 subsystem and associates them with the created thread.

When the application program (APPL1) terminates, the thread is terminated. The subsequent RUN command initiates another application (APPL2) with a different plan. The END command terminates the DB2 connection.

Concurrency rules. Applications can concurrently access and change (i.e., insert, update, delete) data in a table. When an application changes data in a table, that change is not visible to another application until transaction disposition, at which time the change occurs (commit) or does not occur (abort). Should a competing application attempt to access data in a changed and uncommitted state (called *dirty data*), that application waits until the disposition of those data has been made.

Further, when an application retrieves a row with the intention of updating it (a situation that can be expressed in SQL), no other application can change those data until another row has been selected. If both applications have retrieved the same row with the intention of updating it and both attempt to change the data, a deadlock results. This is detected, and one of the applications is selected for termination.[13]

Optionally, the second concurrency rule can be extended by an application to include all data that have been accessed. Change by another application of any data or index entry that has been used by this application is prevented until transaction disposition. At the expense of concurrency, this application may have the appearance of being the only application accessing the data.

Locking. The implementation of the concurrency claims uses locks acquired and released by DB2 through the IMS Resource Lock Manager (IRLM). Through various specifications of control, the entire table space or only those pages accessed in the table space may be locked. In the case of the entire table space, any change to the table space requires its exclusive use, thus prohibiting concurrent access by

other applications. In the case of locking pages accessed in the table space, the concurrency rules apply to data pages.

Indexes are stored on 4K pages, which can be subdivided by the definer of the index into smaller units called *subpages*. If an entire table space is locked, the index is protected by that lock. If data pages are being locked, access and change of an index entry are protected by locking the index subpage on which that entry resides for the duration expressed by the concurrency claims.

Write ahead log. All changes subject to commitment use logging as an integral part of guaranteeing data integrity. When data are changed, log information is written that allows DB2 to make the change (redo) or reverse the change (undo). The data cannot be written until the log information is recorded on external media.

The transaction coordinator determines the disposition of the transaction.

After recording them, the data can be written. If the transaction aborts, the data changes can be reversed from the information in the log. Should the system or DB2 terminate before the transaction finishes, DB2 can redo committed changes and undo aborted changes from the log information when DB2 is restarted.

Transaction disposition. The transaction coordinator determines the disposition of the transaction and informs all participants of this decision. Environmental considerations (e.g., application abnormal termination) or application direction may lead to the initial decision to commit or abort. Since any participant may have similar environmental concerns (e.g., they abnormally terminated during a request) that preclude them from committing, the two-phase commit protocol is employed by all coordinators to determine transaction disposition.

If the coordinator's initial decision is to commit, all participants are notified of the intention to commit (Commit Phase 1). Commit Phase 1 is considered to be a vote collection and preparation phase. Each participant must declare whether to agree to commit. The effect of one negative vote is that nobody commits changes and the transaction is aborted. Each participant agreeing to continue with the commit process must guarantee to commit the changes, while still being capable of reversing the transaction's effect if a negative vote is cast by some other participant. DB2 provides this guarantee by ensuring that all log information is recorded on external media (not just in the virtual storage log buffer) before returning to the coordinator. Once a participant has returned a positive vote, the participant must await receipt of the total vote outcome relative to the transaction before continuing.

For a participant, the period of time between returning a positive vote and receiving the total vote outcome is called INDOUBT. Only the coordinator knows whether all votes have been received and whether an outcome has been determined. If DB2 terminates prior to casting a vote at Commit Phase 1, DB2 aborts the transaction at DB2 restart. If asked later by the coordinator for a Commit Phase 1 vote, DB2 will reply negatively. If DB2 terminates after casting a vote at Commit Phase 1 and prior to receiving and recording the outcome (INDOUBT), DB2 queries the coordinator at DB2 restart to determine the transaction disposition. If the coordinator is not present, DB2 locks all of the changed data to prevent access and resolve its disposition when the coordinator issues an IDENTIFY and provides the disposition of INDOUBT transactions.

The coordinator records the outcome and ensures that it is on external media before continuing. If the outcome is positive, the coordinator is responsible again for notifying the participants of the beginning of this phase. Commit Phase 2 is considered to be a must-complete notification and processing phase. The participants must prepare to make the new, committed form of the recoverable objects accessible to subsequent transactions and must forget the prior object form.

DB2 records the outcome (Commit Phase 1 or abort) on its own log to provide autonomy for restart and media recovery. That is, DB2 does not have to ask the coordinator for transaction disposition information, except for those INDOUBT. If the disposition is abort, the changes are reversed using logged infor-

mation. At the end of commit or abort, all data locks can be freed, because the data are in a clean state.

Authorization. Access to data and applications (plans) is controlled by the DB2 authorization mechanism, which uses the authorization identifier of the

Access to data and applications is controlled by the DB2 authorization mechanism.

requestor to determine whether access privileges are allowed. The authorization identifier may be supplied by IMS or CICS. Both environments assume responsibility for protecting the authorization identifiers and controlling their usage. In TSO, the authorization identifier may be supplied by an installation-provided exit, or it defaults to the TSO user identifier, usage of which is controlled by TSO.

The privilege of passing authorization identifiers is restricted to IMS and CICS. Each connecting environment tells DB2 its connection type through IDENTIFY. To prevent unauthorized IDENTIFY requests, an installation can control who can connect to DB2 and what connection type they are permitted to use through the Resource Access Control Facility (RACF).

Processing an SQL statement

The implementation of data base management in DB2 is demonstrated by sketching the processing for a data base request. In discussing this demonstration, we refer to the SQL flow shown in Figure 8.

Binding. Data base activity ranges from planned applications that are executed many million times over the same set of data to ad hoc queries that execute once. The DB2 bind process accommodates both environments. Application SQL statements are bound once. The product of the bind is saved and used for every application execution, thereby amortizing the cost of bind across the statement's repeated execution. At the same time, the bind process permits an application to construct and submit an SQL statement for immediate binding and execution. In short, all SQL statements are bound when the plan is defined, except for dynamic SQL statements, which are bound when submitted at execution time via the PREPARE statement.

In the bind process, all statements are parsed. As described previously, manipulative statements are bound further, thereby producing executable code and data structures tailored to the statement that is used at execution. Definition, authorization, and control statements use the parsed output interpretively at execution, rather than generated code. The expected savings that would result from any further reduction of these statement types do not warrant the cost and complexity of such a reduction.

DB2 invocation. The results of the plan creation are saved in a system data base for recall at application execution. When a thread is created by the attachment, a private version of the named plan is made available in the data base address space, in anticipation of the first SQL request.

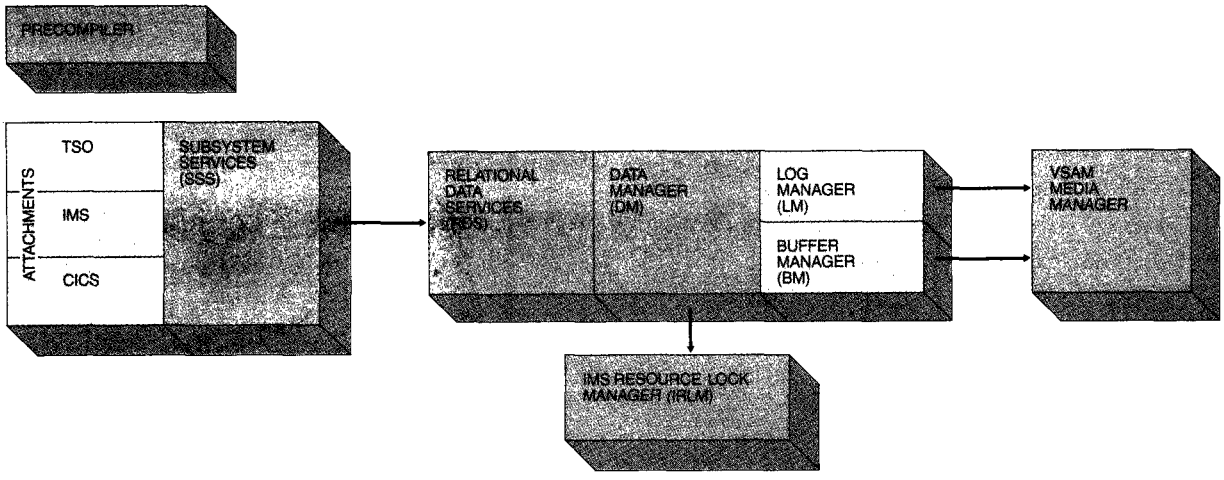
Having created or located the correct thread, the attachment passes control to the Subsystem Services (SSS) via a Program Call (PC), thus causing a switch to DB2 state (supervisor) and key.

Subsystem Services keeps a record of each application currently executing code in the DB2 address spaces. If the DB2 subsystem is to be stopped, SSS prohibits new arrivals and waits for all work dependent on the DB2 address spaces and control structures to cease before permitting the stop processing to occur. This provides an orderly reporting to the application of the event. SSS then verifies that the application is authorized to use the thread; i.e., a thread is not named for another application. Finally, the request is routed to the Relational Data Services (RDS) component of DB2.

RDS provides a small layer of code, called the SQL Program Request Handler (SQL PRH), that operates in the application address space and effects a primary address space switch to the DB2 data base address space where the rest of the RDS function operates. MVS continues to use the application address space (home) for dispatching priorities. The processor uses the DB2 data base address space for fetching instructions and general data references.

On entry in the data base address space, RDS moves program variable values required by DB2 (e.g., a program variable may be used in an SQL predicate)

Figure 8 SQL flow



from the application address space to the DB2 data base address space. This isolates all other code in other components from addressing mode considerations when using the data.

Data definitional structures. To help understand the flow from this point we divert to a presentation of the data definitional structure and responsibilities. Data description is layered, which strongly separates an application's view of data from the actual data on storage media. Data are stored in VSAM data sets, the descriptions of which reside external to DB2 in a VSAM catalog. The DB2 table space provides the logical description of the space (e.g., size of the pages) for mapping these attributes to the *physical repository* (a list of data set names). This separation permits media changes (e.g., changing device types) without disturbing logical descriptions.

To minimize the burden of data set maintenance on users and data base administrators, DB2 defines and deletes data sets if desired. One or more groups of DASD volumes, called *storage groups*, may be defined to DB2. When a table space or index is defined, the user may identify a storage group for DB2 to use to acquire space when defining or extending the definition of data sets for the table space or index. The user need not invoke Access Method Services (AMS) to create data sets for the physical repository.

Stored data representation *(internal)* is separated from user and application perception *(external)*. The definition of a base table generates both descriptions and identifies the data repository by naming a table space. Stored data contain control information and field values. The control information includes a table identifier that precedes each stored record: for example, to distinguish records from different tables when more than one table is defined in the same table space.

Field values may differ in form from the external value. Numeric data values are always converted to forms that neutralize the data, making nonarithmetic comparisons valid for a single data type (i.e., packed decimal, floating point, and binary). This conversion minimizes data search time by eliminating data type sensitivity in the search function.

External perception of data is in the form of a table, i.e., a set of data values of specified data types. Data may be materialized from a single stored table or multiple stored tables. Application data types may differ from stored data types. The SQL SELECT statement coupled with the target data types into which the data are to be selected is the external view of data.

The separation of descriptions provides an architecture that is responsive to change at each level, thereby minimizing disturbance to other levels. This separation is also reflected in DB2 component structures.

Relational data services. Relational Data Services (RDS) is responsible for materializing the external views of data from stored data. For each manipulative SQL statement, a procedure is developed for deriving the external data. The access path, scan or

index, and method for evaluating queries are chosen. The generated code reflects these choices and invokes the Data Manager (DM) to retrieve or alter the stored data.

Evaluations of joins, user-specified ORDER BY clauses, and other features require data to be ordered. If no index is available that meets the criteria for ordering, RDS extracts and orders the data. RDS uses its own sort function, which is tailored to the DB2 environment (cross memory), and uses space suballocated from a common system table space that is dedicated to sorting for intermediate storage. The function supports concurrent sorts from multiple transactions.

Management of stored data. The Data Manager manages all stored data by providing access to data and by using a sequential scan or index, as specified by its invoker. The DB2 catalog and other system table spaces are organized for hashed access and can contain records from multiple tables that are related by links. The Data Manager provides hash and link access to accommodate these formats.

The Data Manager provides a set of operators to alter (i.e., insert, delete, modify) data one record at a time. When data are altered, all indexes and links are automatically updated. The Data Manager also guarantees all data consistency claims by locking data, using the IRLM. Changes to data are guaranteed by logging information using the Log Manager (LM). The information enables DB2 to recreate committed changes and reverse aborted changes.

The Data Manager accesses data by identifying the desired page within the table space or index to the Buffer Manager (BM). If the data are to be changed, the Buffer Manager is informed, so that it knows that it is to write the page when all updates have finished.

Buffer management. The Buffer Manager manages pools of virtual storage (buffers) in the data base address space that holds data and index pages. The Buffer Manager directs the VSAM Media Manager to transfer data and index pages between media storage and virtual storage. The goal is to minimize the time that any request must wait for data while meeting an installation-specified goal for the time it takes to recover from a subsystem or system failure.

Multiple threads can reference the same page in the buffer pool concurrently. The Data Manager serializes updates by locking data to be altered, thus preventing access to it by any other threads. Since index pages are serialized at the subpage level, multiple transactions may be updating the same index page concurrently. The Buffer Manager tracks this activity and writes a page only when it is in a consistent state (i.e., all changes are committed or backed out).

In general, page writing is asynchronous with respect to applications.

Pages are retained in the buffer pool even though no outstanding claims exist for their usage. Unused pages are aged out of the pool as the demand for pages exceeds the unassigned buffers. The buffer pool is always searched for a requested page before accessing data from stored media.

In general, page writing is asynchronous with respect to applications. At commit time, the DB2 log is forced, which ensures that the data can be recreated at restart time if there is a problem that terminates DB2. Pages are written primarily on the basis of demand for buffers that exceeds the available supply. Using the Write-Ahead-Log protocol, the Buffer Manager ensures that the log records that guarantee data consistency have been externalized prior to writing a buffer.

Installations specify checkpoint intervals to balance subsystem recovery time with the performance of the subsystem. With no other guidelines, the Buffer Manager delays writing a highly accessed page until DB2 termination. If the subsystem fails, the page is reconstructed from the log during restart. The amount of log processing required is a key determinant of the time it takes to make the subsystem receptive to new work. Checkpoint, which is scheduled on the basis of installation-specified log activity limits, signals to DB2 resource managers the desire to schedule and complete work in progress. This re-

duces the amount of log processing requried at restart. The Buffer Manager participates in checkpoints by attempting to initiate writes for committed changes. Those buffers that cannot be written because they are in an inconsistent state are marked to be written the next time they reach a consistent state.

Concluding remarks

DB2 provides relational data base management in the MVS environment. Provision for data independence acknowledges the inevitability of change while maintaining a consistent interface to programs and users not requiring those changes. This protects the programming investment and improves usability. The design of DB2 anticipates growth and changes to DB2 itself. The linkage mechanism, differentiation of responsibility in data base definitional constructs, and other principles of the design make DB2 receptive to change for tomorrow while offering function for today.

Cited references

1. W. W. Chiu and W. M. Chow, "A performance model of MVS," *IBM Systems Journal* **17,** No. 4, 444–462 (1978).
2. A. Padegs, "System/370 Extended Architecture: Design considerations," *IBM Journal of Research and Development* **27,** No. 3, 198–205 (May 1983).
3. E. F. Codd, "Relational database: A practical foundation for productivity," *Communications of the ACM* **25,** No. 2, 109–117 (February 1982).
4. C. J. Date, *An Introduction to Database Systems, Third Edition*, Addison-Wesley Publishing Company, Reading, MA (1981).
5. G. Sandberg, "A primer on relational data base concepts," *IBM Systems Journal* **20,** No. 1, 23–40 (1981).
6. D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, "SEQUEL 2: A unified approach to data definition, manipulation, and control," *IBM Journal of Research and Development* **20,** No. 6, 560–575 (November 1976).
7. *IBM DATABASE 2, Introduction to SQL*, GC26-4082, IBM Corporation; available through IBM branch offices.
8. M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson, "System R: A relational approach to data base management," *ACM Transactions on Database Systems* **1,** No. 2, 97–137 (June 1976).
9. M. W. Blasgen, M. M. Astrahan, D. D. Chamberlin, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J. W. Mehl, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, I. L. Traiger, B. W. Wade, and R. A. Yost, "System R: An architectural overview," *IBM Systems Journal* **20,** No. 1, 44–62 (1981).
10. D. D. Chamberlin et al., "A history and evaluation of System R," *Communications of the ACM* **14,** No. 10, 632–646 (October 1981).
11. D. J. Eade, P. Homan, and J. H. Jones, "CICS/VS and its role in Systems Network Architecture," *IBM Systems Journal* **16,** No. 3, 258–286 (1977).
12. W. C. McGee, "The Information Management System IMS/VS," *IBM Systems Journal* **16,** No. 2, 84–168 (1977).
13. J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger, *Granularity of Locks and Degrees of Consistency in a Shared Data Base*, North-Holland Publishing Co., New York (1976).

Reprint Order No. G321-5213.

Donald J. Haderle *IBM General Products Division, Santa Teresa Laboratory, P. O. Box 50020, San Jose, California 95150.* Mr. Haderle joined IBM in 1968 as a programmer to develop operating systems for use in process control. After work on the 1800 Multiprogramming Executive (MPX) and a prototype for support of point of sales, Mr. Haderle shifted his efforts to data access methods and data base management systems. In 1972, he was the technical leader for the development of common data access method services for OS/VS2 MVS. In 1977 he joined the team to develop the Database 2 (DB2) data base management system. Mr. Haderle, currently a senior programmer at the Santa Teresa Laboratory, is continuing work on the DB2 product. He received a B.A. in economics from the University of California, Berkeley, in 1967.

Robert D. Jackson *IBM General Products Division, Santa Teresa Laboratory, P. O. Box 50020, San Jose, California 95150.* Mr. Jackson is a senior programmer at the Santa Teresa Laboratory. He joined the IBM Data Processing Division in 1966 after receiving a B.S. in mathematics from San Jose State University. During his IBM career, Mr. Jackson has been associated with systems design and development, working in the I/O supervisor area of OS/VS2 MVS and later concentrating on Advanced Data Systems. He joined the DB2 project at its inception and has held both technical and management positions within the project. During 1983, Mr. Jackson was on assignment to the Installation Support Center, Greenford, England, where he participated in introducing the DB2 product to customer accounts. Mr. Jackson is currently working in Advanced Data Systems at the Santa Teresa Laboratory.