Relational Database Integration in the IBM AS/400

S. Scholerman*, L. Miller*, J. Tenner#, S. Tomanek#, M. Zolliker#

Abstract

A great deal of research has been focused on the development of database machines. In parallel to this work some vendors have developed general purpose machines with database function built directly into the machine architecture. The IBM AS/400 is one of the principle examples of this approach. Designed with a strong object orientation and the basis functions of the relational database model integrated into it's architecture, the AS/400 has proved to be a commerical success. In the present work we look at the database component of the AS/400.

1. **Introduction**

A great deal has been written about the utilization of database machines in or for data processing applications [HuMP89,HMPE89]. However, at this point the market remains software based with only limited penetration by machines that satisfy the typical definition given for database machines.

* Department of Computer Science, Iowa State University, Ames, Iowa 50011

#International Business Machines Corporation, Rochester, Minnesota 55901

Paralleling the work on database machines has been the work of some manufacturers on integrating database functions into the architecture of general purpose computers. An important example of this approach is the IBM AS/400 product line (and its predecessor, the S/38).

The AS/400 family of computers is a line of general purpose, mid-range systems. The AS/400 family currently (V2R2 announced 2/92) includes 13 models ranging from the E02 to the E90. The smallest model, the E02, supports up to 24MB of main memory and up to 1,967MB of DASD capacity. The top of the line E90 machine is a multi-processor that supports three processors, up to 512MB of main memory and 123.4GB of DASD capacity. All of these models use the same operating system, OS/400, and support the same approach to databases so the remainder of this paper will generally apply to any AS/400 model.

The AS/400 line was introduced by IBM in June of 1988. The machine was depicted as a merger of the best features of its predecessors, the System/38 and the System/36. The underlying architecture of the machine is basically the S/38 architecture with some of the S/38 limits removed and additional enhancements included. The S/36 influence can be seen in the extensive use of menus and the communication capabilities.

One of the primary features that distinguishes the AS/400 from other computer systems is its integrated function. Functions that are typically implemented via separate system-software packages are integrated directly into the hardware and operating system software of the AS/400. This integration should not be confused with bundling. The problems that separate software products exhibit such as overlap of function, inconsistent interfaces and complex system management are not overcome by bundling. On the other hand, the integration of function that is used by the AS/400 overcomes these drawbacks and has the advantages of consistent user interfaces, reduction of system code duplication, increased efficiency of system software, simplification of application development and reduced user intervention and system management.

Several other features distinguish the AS/400 line of computers from other computer systems. First, the architecture of the machine is object-based and defines a high-level interface. A more detailed description of the AS/400 architecture can be found in Section 2. The AS/400 also has an integrated relational database management system. In fact, many of the system's design decisions were made with database in mind and its primary use is for data-intensive applications. The relational DBMS is discussed in more detail in Section 3.

2. **AS/400 Architectural Features**

This section briefly examines some of the architectural features that set the AS/400 apart from other computers in its class. Levy discusses the architecture of IBM's System/38 in [Levy84]; the majority of this information also applies to the AS/400. Additional information regarding the architecture of the S/38 can be found in IBM's technical documentation [Henr80]. Several reports developed by the ADM consulting firm [And90a, And90b] provide information on the AS/400 architecture and an AS/400 system overview can be found in [ScTa89].

The primary architectural features of the AS/400 include the layered organization of function, the high-level machine interface, the notion of a single-level storage, the object orientation of the machine, capability-based addressing and the tagged pointer architecture. Each of these concepts will be discussed briefly in the following paragraphs.

The layered organization of the function in the AS/400 consists of five layers. The bottom 3 layers, the hardware, the horizontal licensed internal code (HLIC) and the vertical licensed internal code (VLIC), implement the machine interface (MI). Above the MI is the operating system (OS/400) and the top layer consists of high level language compilers, utilities, and applications. Each layer provides a well-defined, consistent interface which makes the layers independent and allows implementation changes to lower layers without affecting higher levels. An application programmer is prohibited from using hardware and microcode instructions and is forced to program at a higher level. This has the net effect of making application development easier and faster. Figure 1 shows the AS/400 architecture.

Directly tied to the layered organization of function is the high-level machine interface (MI). This interface exists between the operating system (OS/400) and the vertical licensed internal code (VLIC). This interface is referred to as an opaque interface because it is the lowest interface that can be accessed by the layers above it. All of the implementation details below this interface are hidden and can be changed without affecting the user.

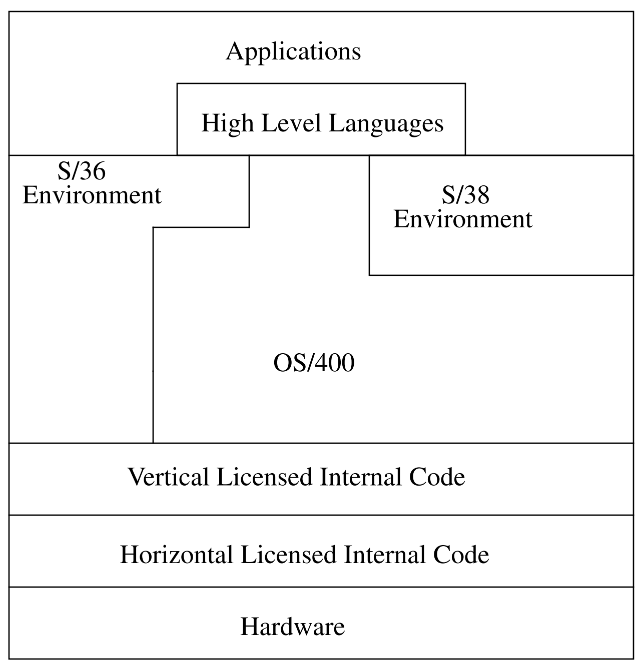
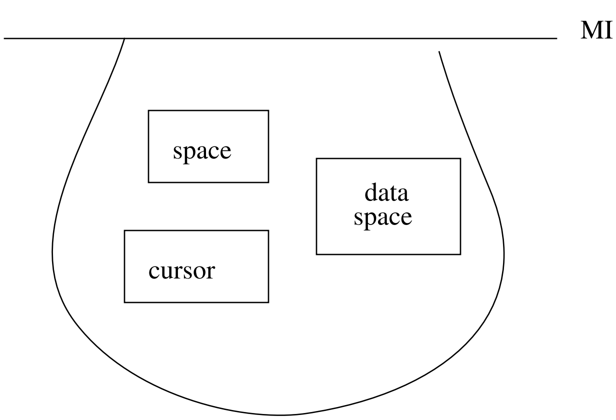


Figure 1. AS/400 Layered Architecture.

Secondary and main storage on the AS/400 is viewed as one large contiguous space. All objects reside in this single level store and can be accessed by any program that has the proper authority. All secondary and main storage is managed by a single storage management component that is part of the VLIC. At the MI level, one addressing method is used to reference both objects in main memory and objects on disk. Object sharing therefore becomes straight-forward because of this uniform addressability.

Everything that is stored on the AS/400 is stored as an object. The main difference between user objects and system objects is that system objects are used as the building blocks of user objects. The object orientation follows the abstract data type paradigm as opposed to the message passing paradigm. System objects and user objects are encapsulated. Figure 2 shows an example of a complex user object.

The interface part of an system object is defined by the MI instruction set. The instruction set contains some generic object operations like change object owner and grant authority. Since the information required to perform these operations is stored in the common object header, one operation definition can handle any object type. The instruction set also defines operations that are specific to certain object types like activate cursor and insert data space entry. These MI instructions that operate on objects, whether they are generic operations or type specific operations, can be thought of as “methods”. The code which implements all of these operations or methods is contained within the Vertical Licensed Internal Code.



A single membered, non-keyed user physical file consisting of three MI ojbects, one of type space, one of type cursor and one of type data space.

Figure 2. An example of a composite user object.

It is also possible to associate an unencapsulated space area with a system object. This space area is called an associated space. It is a byte accessible area that can be accessed by the machine user (the operating system) and is generally used to store control blocks or object definition information. All or part of the information in the associated space of a machine object is typically presented to the user upon the issuance of the appropriate display description command. A special type of object that contains no encapsulated part and only has an associated space is a space object. Whether the space is part of a space object or an associated space for another object, it is not used by the VLIC or the hardware. It is only used by and has meaning to the machine user (the operating system). The AS/400 utilizes capability-based addressing. On a capability-based system, each user and program has access to a list of capabilities which define those objects that the user or program is authorized to access. A capability can be thought of as a token that gives the possessor permission to access an object. On the AS/400, a capability is implemented as an authorized system pointer. The system pointer contains the address of the object as well as a set of usage rights. Initially, a system pointer is in an unresolved state. In this state, the pointer contains the name of the object, not the address. Upon first reference, the system searches for the object and, once found, replaces the name with the address. Subsequent references to the pointer will not require the search.

These features make the AS/400 unique and provide the basis for the integrated database approach used in the design of the AS/400.

3. **AS/400 Integrated Relational DBMS**

3.1. Basic Notions

Although the AS/400 is not a database machine in the traditional sense, the design of the system was heavily influenced by database considerations to the point that a relational database management system (RDBMS) was integrated directly into the machine to overcome the drawbacks of a separate relational DBMS software package. Support for the DBMS can be found implemented in the hardware, the vertical licensed internal code (VLIC) and the operating system. Separate software packages (e.g.

SQL/400, Query/400, etc.) relating to the DBMS are available and can be added to the system but these packages primarily offer alternative interfaces to the DBMS. Each of these separate interface products supports a subset of the total database function that is implemented by the operating system, the VLIC and the hardware.

The base DBMS interface supported by the AS/400 operating system is usually called the “native” support. Most of the function that is supported by the DBMS can be accessed through the systems Control Language or CL. Since CL is part of the operating system, all users automatically have access to the relational DBMS as soon as the operating system is installed. CL commands can be used to create the two basic file types on the system, the physical file and the logical file.

The physical file is the file that holds the data records(Figure 2). The format of the record can be defined in several ways. The most powerful way is to define the record format with Data Description Specifications (DDS). When this method is used, the field level descriptions are known to the system. A file created in this way will have “externally described data”. It is also possible to only define the length of the record to the system with the field level descriptions contained within application programs. All of the records that are stored in a single physical file will have the same single record format. A physical file may have zero or more “members”. A physical file without a member contains no data and is a file description only. The members are what actually store the data records and they are the vehicles by which records are inserted into and retrieved from files. Multiple members for a single physical file can be used to subset the data that is stored in the file. For example, an accounts receivable physical file may have 12 members, one for each month of the year where each member contains the accounts receivable records for that month.

The logical file is used to define an alternate view of the data that is stored in physical files. A simple logical file is built over one physical file. The simple logical file can be used to rearrange or subset the fields of the physical file. It can also be used to map a single field to a different data type or length or it can be used to substring a field or concatenate two or more fields. A simple logical file may also contain selection specifications. A join logical file can be used to join the data records of up to 32 physical files. The join logical file supports either inner join or left partial outer join. All the functions that can be performed on the fields in a simple logical file can also be done in a join logical file. A third type of logical file is the multi-format logical file. This file type can also be used to combine the records of up to 32 physical files but it does not join records. It can be used to define a hierarchy between the records of the physical files. When the records are retrieved, they are returned in this hierarchical fashion.

A physical or logical file may be keyed or unkeyed. A keyed file is one that has one or more fields defined as key fields. The AS/400 currently supports up to 2000 bytes of key data per file. An access path is created and maintained by the system where this access path specifies the order of the records based on the values of the key fields and any other access path attributes that may have been specified by the user. The operating system tries to share access paths as often as possible. Sharing occurs when two or more logical files define compatible access paths over the same physical data.

An important concept on the AS/400 is that of externally described data. Any file that is created through the native approach using DDS is externally described. Tables and views that are created using SQL/400 are also externally described. An externally described file is characterised by the fact that all of the field level descriptions of the record format are known to the system. When a program is written, the field declarations can be automatically pulled into the program by the system when it is compiled. This prevents programs from using files that may have changed formats. The open of a file by a program will be disallowed if the program has been compiled against an incompatible version of the file.

The AS/400 supports multiple interfaces to data that resides on the system.As previously mentioned, the native system support can be accessed through CL. It is also possible to query data using the CL command, Open Query File (OPNQRYF) which is very powerful and, when issued, results in the open of a query member. Through this query member, the records meeting the query criteria can be retrieved. Once this member is opened, it is possible to override the file of a program to use this open member instead. This provides the user with an easy way to supply query results to a traditional file-oriented application.

An SQL product called SQL/400 is also available. The SQL product provides the user with an interactive SQL interface as well as the compile-time SQL routines needed to compile programs with embedded SQL statements. The runtime SQL routines needed to run programs with embedded SQL are provided as part of the operating system. This means that a user may purchase an application with embedded SQL and may run this application without purchasing the SQL/400 product. The SQL/400 product provides the user with the ability to create SQL collections (native library with a data dictionary), SQL tables (native physical files), SQL indexes (native simple keyed logical files) and SQL views (special logical file which cannot be created through the native interface).

Another query product supported is Query/400. This query product is typically used to create reports for existing data. The query definitions created with the product can be saved and later run by issuing a single command. Also supported is the Interactive Data Definition Utility (IDDU). The AS/400 IDDU is similar to the S/36 IDDU and it primarily provides the S/36 environment users with a menu-driven, interactive method for describing data. Externally described files can be created with IDDU or IDDU can be used to link a program described file to an IDDU-defined format description.

It is important to note that files created through any interface can typically be accessed through any of the other interfaces. For the AS/400 user, this fact is probably the most significant advantage of integration. This flexibility allows the user to see the data as files in one application and see the data as part of the relational model in another application. It eliminates the need to duplicate data for file and relational applications as is typically done on systems that utilize both a file system and a database management system. Data security and integrity are also strengthened since both native file access and relational access use the same security, journaling and commitment control functions that are implemented below the Machine Interface.

Another advantage of the integrated DBMS is the performance advantage it offers over traditional systems. As was previously mentioned, database function is implemented throughout the layers of the system including the microcode and the machine. This means that some database functions run more efficiently than they do on other systems. For example, several data conversion instructions are supported by the machine instruction set.

The AS/400 does have some disadvantages as well. The AS/400 tends to use more storage to store the same information than traditional file systems like VSAM. Data for a single object may be spread over several disk drives. The failure of a single disk drive can mean a complete reinstall of the system since it is impossible to reconstruct the objects on the system, unless some prevention feature has been used (e.g. mirroring, checksumming, alternate storage pools). This is not only a problem for file objects, it exists for all objects on the system.

The AS/400 has advantages in the way that the SQL catalogs have been implemented. On traditional systems, the catalogs are referenced whenever queries are compiled in order to retrieve definition information. Catalogs are also referenced during other operations like authority changes and table deletion. Therefore, these catalogs become a bottleneck. On the AS/400, definition information is stored with the objects so that references to relational data do not require references to the catalogs.

The SQL/400 product does suffer from a performance disadvantage verses native DBMS access due to the fact that the product is built on top of the native database management system and SQL queries must therefore execute a longer code path than native DBMS accesses.

3.2. Distributed Data

The integrated DBMS of the AS/400 system provides key advantages in the area of distributed relational database and distributed file access. Probably the greatest advantage is that the same objects can be accessed by either method. This combined with the integration of database functions with other system functions improves the function and usability of the OS/400 database.

Distributed file access is provided on the AS/400 system by Distributed Data Management (DDM) files. These files contain information used by the local system to locate remote systems and access remote files. File access operations are performed using the statements supported by the high-level language. For example, OPEN and UPDAT are used in RPG/400 applications while OPEN and REWRITE are used in COBOL/400. These statements are used to access remote files in the same way that they are used to access local files. Likewise, the same CL commands that are used to operate on a local file are used to operate on a remote file. For both high-level language access and CL access, when an application running on the local system performs operations against a DDM file, OS/400 communicates with the remote system using Distributed Data Management architected flows to have the operation performed against the remote file. Although access to remote files is not through SQL, the integrated DBMS makes it possible to access SQL tables and views on remote AS/400 systems using the high-level language file access statements and CL just as it does for local SQL tables and views.

Distributed relational database access is provided by using Distributed Relational Database Architecture (DRDA) to communicate with remote relational databases identified in a relational database directory. Each directory entry identifies a relational database and information on how to access it. Each AS/400 system is exactly one relational database. Therefore, for the relational database entries that refer to AS/400 systems, each entry refers to a unique system. Each relational database usually has several collections and/or libraries. Each library or collection usually contains several tables, views, indexes, etc. When an application wants to work with a specific table in a collection at a specific relational database, it directs it's SQL statements to the relational database by identifying its name on an SQL CONNECT statement. OS/400 determines if the relational database is remote or local by examining the relational database directory entry. When the relational database is remote, DRDA flows are used to communicate between the remote relational database and the local AS/400 system. At the remote relational database, an object called a SQL package is used to access the relational database. The SQL package is created at precompile time and contains information about the SQL statements and host variables to provide efficient processing of the statements. It also provides a means to have statements run under the profile of the application owner. Although access to remote relational databases is not through high-level languages and CL, the integrated DBMS makes it possible to access physical files and logical file on remote AS/400s using SQL just as SQL can be used against local files.

3.3. Role of VLIC

There are four areas of the VLIC that are concerned with database function. Each is a separate "component" of the VLIC; they are commit management, database management, independent index management and journal management. The commit component provides the capability to group a series of operations to make them appear as one operation. The database component provides the support to insert, retrieve, update and delete records from the database as well as define alternate views of data. The index component provides the access path support to insert and delete keys from a data space index and the support to find entries in the index. Finally, the journal component provides the support to record changes to data spaces and data space indexes in a journal. The journal is used for commitment control, for backup and recovery purposes and for audit trails. There are other components of the VLIC that indirectly support the database facility as well. Some of these include storage management, authorization management and recovery initialization.

In addition to providing the support for the database-related MI instructions, the database VLIC component supports many internal functions. Some examples include building keys, invalidating data space indexes, verifying mapping templates, unlocking data space entries and forcing recently inserted and modified entries.

All of the database functions described above operate on the machine objects that comprise a database file.A physical file consists of a space object which contains the file control block (FCB), and the objects that make up each member. Each member has one cursor object which contains the member control block (MCB) in the associated space of the cursor and all of the data mapping specifications in the functional object. A member has one data space per cursor where the data space contains a definition of the record format, the default record and the actual data records. Optionally, there is a data space index per cursor where the data space index holds the access path description and the access path. The logical file has similar makeup except that the logical member does not have a data space associated with it. It is built over the data spaces of physical members. Also, the cursor object of a logical file is typically much more complex than that of the physical file since is can map fields, join records, and specify selection.

The majority of the database function related to data retrieval (mapping, selection, join, grouping) and data insertion is implemented via the cursor machine object. This object type is one of the main reasons for the great flexibility of the AS/400. All files on the system, whether they are created using CL commands, created using SQL/

400 or created through some alternate interface like IDDU, have a cursor associated with them. The cursor is also the mechanism through which a query is defined. Since data retrieval through queries and files is implemented with the same object, it becomes easy to interchange relational and file access methods for the same data. Below the Machine Interface, all accesses are viewed in the same way. There is little distinction between relational and file accesses. The only real difference is the disposition of the cursors. Files use permanent cursors and queries create temporary cursors.

4. **Summary**

The AS/400 is a general purpose computer that has been designed for database applications. The functionality of the relational DBMS has been integrated throughout the system architecture. The resulting machine is one that does not precisely agree with the definition of a database machine, but incorporates many of the features attributed to database machines.

Acknowledgements

We would like to thank Gary Stroebel of IBM in Rochester, MN for hishelpful suggestions on this manuscript.

References

[And90a] D.H. Andrews et al., *The AS/400 Alternative,* 1990,ADM, Inc.

[And90b] D.H. Andrews et al., *Why Replace a Perfectly Good S/36 with as AS/400?*, 1990, ADM, Inc.

[Banc88] F. Bancilhon, "Object Oriented Database Systems," *Proceedings of the ACM Conference on the Principles of Database Systems,* 1988, pp. 152-162.

[Henr80] G.G. Henry et al., *IBM System/38 Technical Developments*, ISBN 0-933186-03-7, IBM Corporation, 1980.

[HuMP89] A. Hurson, L. Miller and S. Pakzad, *Tutorial: Parallel Architectures for Database Systems*, IEEE Computer Society Press, 1989.

[HMPE89] A. Hurson, L. Miller, S. Pakzad, M. Eich and B. Shirazi, "Parallel Architectures for Database Systems," *Advances in Computers*, Vol. 28, Academic Press, Inc., 1989.

[Levy84] H.M. Levy, *Capability-Based Computer Systems*, 1984, Digital Press, pp. 136-157.

[ScTa89] D.L. Schleicher and R.L. Taylor, "System Overview of the Application System/400," *IBM Systems Journal*, 1989, Vol.28, No. 3, pp. 360-375.