THE INTELLIGENT DATABASE MACHINE™

PRODUCT DESCRIPTION

BRITTON LEE, INCORPORATED

IDM

INTELLIGENT DATABASE MACHINE PRODUCT DESCRIPTION

Preface/Table of Contents

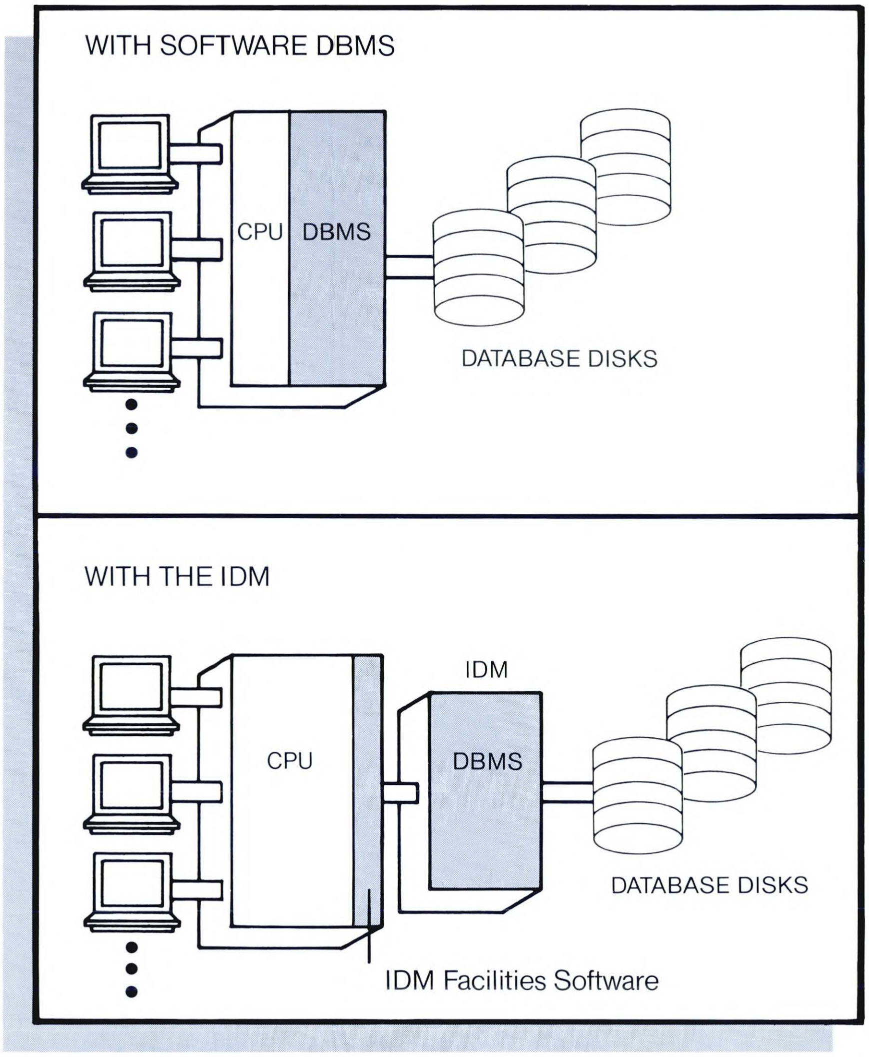
Incorporated in 1978, Britton Lee brought together the special talents of relational database software experts, engineers and software innovators to produce a high performance database computer which would function as an attached processor for any host or hosts. The product concept specified that this database computer would:

- Put non-data processing managers directly in touch with the vital information stored in their computers.
- Serve as an invaluable productivity tool for the DP professional who was overburdened with information management tasks.
- Provide the flexibility necessary to easily accommodate expansion in host hardware as well as changes in application software.

The Intelligent Database Machine (IDM), our back-end relational database management computer, is Britton Lee's solution to the information management challenge of the '80s.

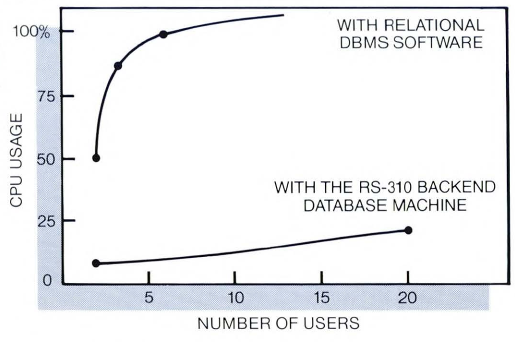
1. The Intelligent Database Machine

The Intelligent Database Machine (IDM) is a fully relational database management computing system. Implemented with specialized hardware to perform its relational DBMS functions, the IDM is situated between host and disks where it offloads the host of all burdensome DBMS tasks. Used with intelligent terminals, the IDM can function as a stand-alone database computer or as a back-end database processor serving one or more mini, micro, or mainframe CPUs.

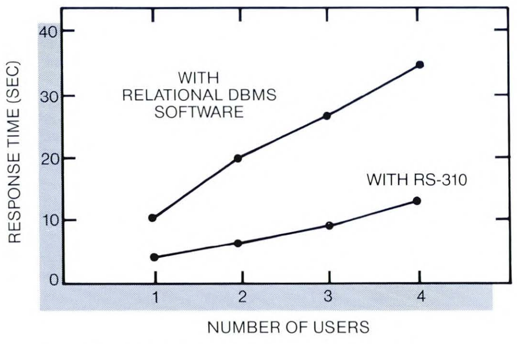


1.1 Advantages of a specialized back-end processor

Designed for high-speed execution of its relational DBMS software, the Intelligent Database Machine offers the user significant benefits:

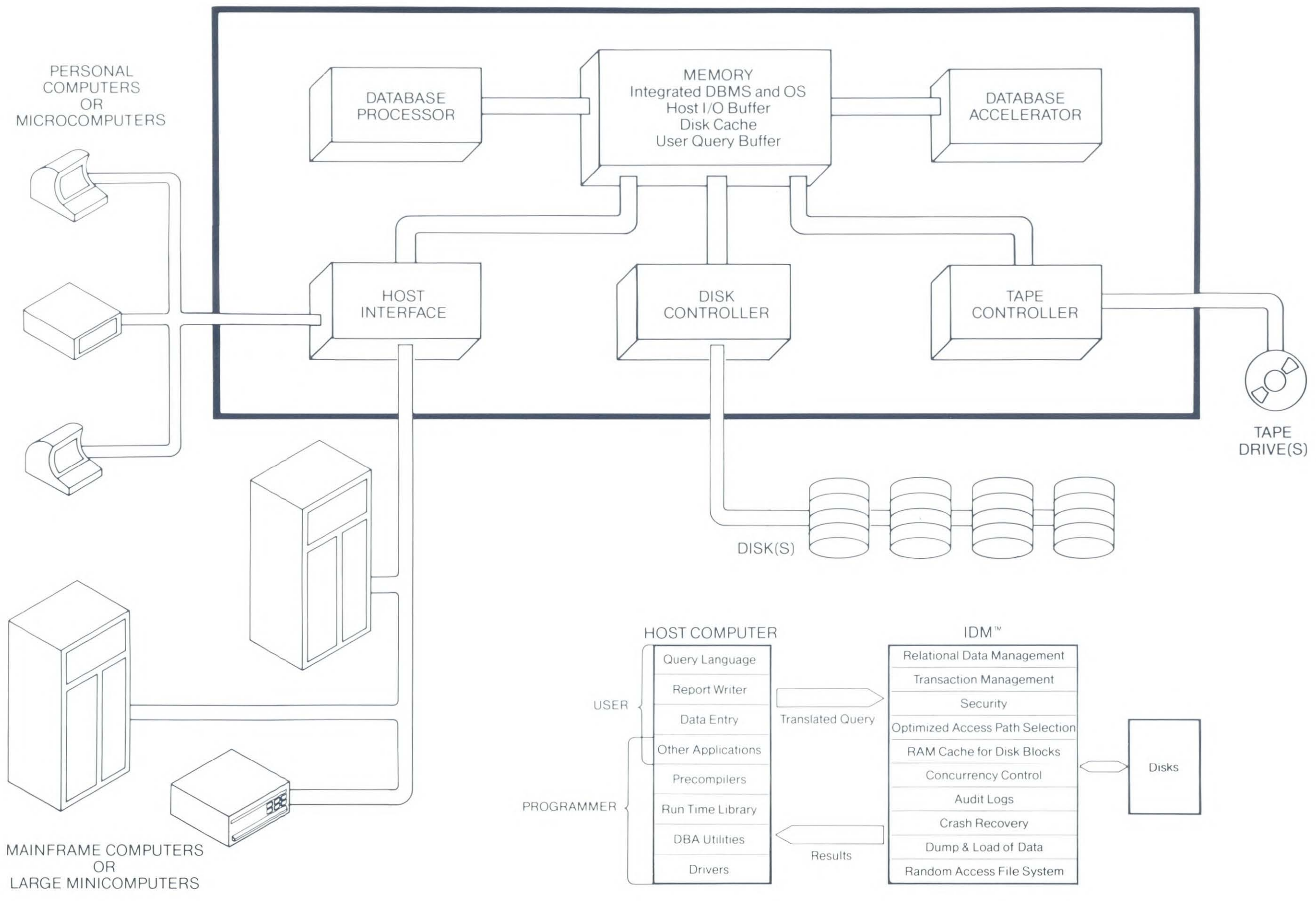


It offloads the host CPU of the resource-consuming database management tasks.



It performs DBMS functions at high speed because the hardware was specifically designed to execute its relational DBMS software.

The Britton Lee Intelligent Database Machine



It accommodates on-going changes in the host environment since it can be used with more than 100 CPUs of any type and with up to 16 SMD disks of most makes and capacities.

With the Intelligent Database Machine, the user has a centralized database resource that can be shared by a variety of users from many different hosts.

1.2 DBMS Functions Performed by the IDM

The Intelligent Database Machine incorporates all of the basic functions required of a complete database management system. These functions are summarized below.

- Transaction management facilities ensure data integrity.
- Protection features eliminate unauthorized access to information.
- Integrated Data Dictionary facilitates keeping track of what information is contained in the database.
- The non-procedural, high-level query capability makes it easy to access and update information.
- Automatic concurrency control permits multiple users to access and update data in the same database at the same time.
- Data administration facilities assist the DBA in optimizing throughput and in restructuring and expanding databases
- Crash/Recovery features include checkpointing and transaction journaling
- Data independence minimizes the effect on application programs when changes are made to the database
- Virtual tables permit different "views" of the same data
- Dynamic indexing allows creation or deletion of up to 251 indices per relation with up to 15 fields per index.
- The Stored Command facility provides an IDM macro definition capability so that repetitive DBMS functions can be preprocessed and stored in the IDM for high-speed execution.
- The Mirrored Disk option provides highly effective protection against catastrophic media failures and crashes by duplicating critical databases on a redundant set of disk drives.

1.3 Benefits of the Relational Data Model

The most logical way to represent data is with tables much like ledger sheets. Tables (or relations) consist of a variable number of rows (tuples) and a fixed number of columns (attributes). Both can be added or deleted as information grows and changes. A relational database is simply a collection of related tables. The example below illustrates two of many tables (relations) which might be in a Personnel database.

EMPLOYEES TABLE

LAST NAME	INITIALS	SOC_SEC_NUM	SEX	SALARY	START DATE	DEPT NAME
BROWN	TJ	776-30-4839	F	4000	070380	GARDEN
SMITH	RS	123-79-1122	M	1012	020175	SPORTS
JONES	AM	397-75-2628	M	2080	121678	TOYS
MILLER	RJ	463-55-1120	F	3010	101680	SPORTS
:						

DEPARTMENTS TABLE

DEPT NAME	MGR_SS_NUM	MAIL STOP	PHONE EXT.	QUOTA	YTD
TOYS	123-22-9750	11	570	300,000	198,000
SPORTS	463-55-1120	20	268	350,000	201,000
GARDEN	796-40-3724	77	853	500,000	314,000
:					

WHO IS THE SPORTS DEPARTMENT MANAGER?

Because relational data is viewed as rows and columns of information, databases are easy to implement and easy to use. The relational database structure allows the user to organize, reorganize and access data in a simple and direct manner. Implementation of the relational DBMS application can proceed quickly and without the complicated analysis associated with implementing pointer-based DBMS systems. Inevitable future changes in the database structure are easily accommodated by the IDM's relational database facilities.

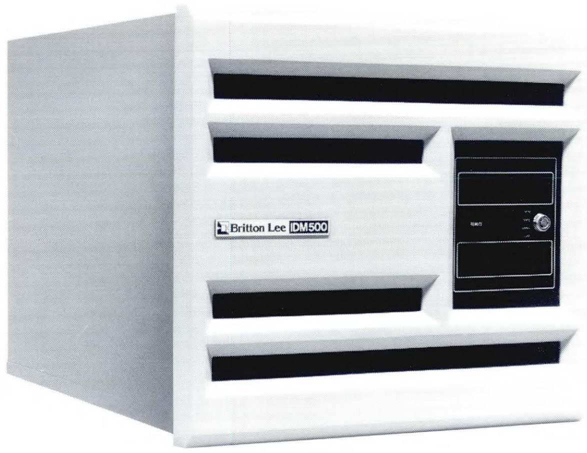
Users need not be highly skilled data processing professionals in order to extract information from the database. Note in the above illustration how simple it is to logically correlate the data in two different tables to find out the name of the Sports Department manager. The non-data processing end user can interact with his data through IDL, the IDM ad hoc query facility, or through SQL, the high-level query language developed by IBM. These languages simplify spontaneous inquiries into the database.

Since IDL and SQL are non-procedural languages, the user need only concern himself with what data he wants, never with how the computer should get it. Unlike procedural languages that force the user to understand the computer, nonprocedural languages force the computer to understand the user. The freedom from having to contend with the complexities of structural database details results in productivity increases and improved communication.

2. IDM Hardware

2.1 The IDM Accommodates Expansion

There are several configurations in the Britton Lee family of database management systems. They vary in performance and database capacity. All configurations are upwardly and downwardly compatible, and all machines are field upgradable.

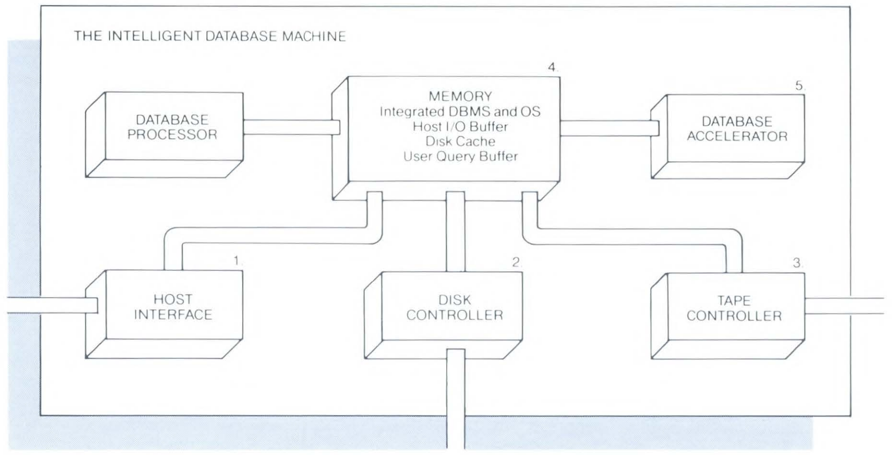


Our entry-level database system, the 500X, is designed for smaller applications. The 500X has 2 Megabytes of random access memory (RAM), 680 Megabytes of mirrored disk (two 340 Megabyte Winchester disk drives), a 500 Megabyte cartridge tape drive for data backup, and a high-performance tape controller. It is equipped with Britton Lee's 500/1 Database Processor and standard disk controller. Expansion slots allow the user to configure the 500X with RS-232, IEEE-488, IBM block multiplexer, or Ethernet local area network (LAN) interfaces.

The 500XL is Britton Lee's intermediate database solution. It comes standard with the company's new SMDE disk controller and 500/2 Processor, which offers the user twice the performance of the 500X. The 500XL comes with 1.03 Gigabytes of mirrored disk (two 515 Megabyte Winchester disk drives), a 500 Megabyte cartridge tape database backup unit, 3 Megabytes of main memory, and Britton Lee's high-performance tape controller.

The 500XLE is Britton Lee's ultimate database management solution. The standard XLE comes with 2 SMDE disk controllers and 500/4 Processor, which offers the user a 40 percent increase in performance over the 500XL. It comes standard with 2.06 Gigabytes of mirrored disks (four 515 Megabyte Winchester disk drives), a 500 Megabyte cartridge tape drive for information archiving, 4 Megabytes of main memory, and Britton Lee's higher-performance tape controller. Like the 500X and 500XL, the 500XLE offers the flexibility of any mix of serial I/O, IEEE-488, IBM block multiplexer or Ethernet LAN interfaces.

The IDM is organized around a central high-speed bus. The design is modular with each module dedicated to performing a specialized function. A module is a single PC board that plugs into the IDM bus. The base configuration includes the standard 7 card set. The boards include a host interface module, a database processor module, a memory timing and control module, 2 memory modules, a disk controller module and a high performance tape controller. The modular design of the IDM, with a function per board, contributes to the ease of maintenance by permitting problem diagnosis on a per board basis.



Additional modules (boards) can be added to the chassis to accommodate more users, to increase disk storage capacity, or to optimize performance, making the entire line field upgradable.

The IDM 500 products can be expanded to include:

1. Up to 8 host interface modules that will support more than 100 mini, micro or mainframe hosts.
2. Up to 4 disk controllers to support up to 16 SMD or SMDE disks of any capacity
3. One tape controller to support 2 tape drives
4. Up to 6 megabytes of IDM memory
5. A database accelerator module that will increase performance by a factor of 2-10 times depending on the DBMS operation being performed

In addition, Britton Lee offers packaged systems complete with host interface and user software including an interactive query language, a report writer, data entry facilities, and application programmer facilities. At present, the IDM system products are designed for DEC VAX computers running VMS or UNIX, AT&T Series 3B computers running UNIX System V, Apollo computers running Aegis, IBM System 370 computers running VM/SP, and PCs supporting MS/PC-DOS.

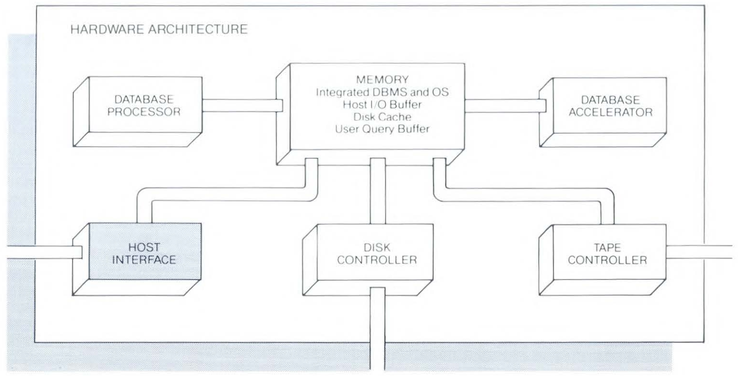
2.2 The IDM Supports a Multi-host Environment

The IDM and its host(s) communicate via the IDM's host interface module (board). The host interface module accepts commands from a host or hosts, checks to ensure that all information sent to and from the hosts has been correctly transmitted and received, coordinates retransmission in case of error, and notifies the Database Processor that it is transferring the request into IDM main memory for processing.

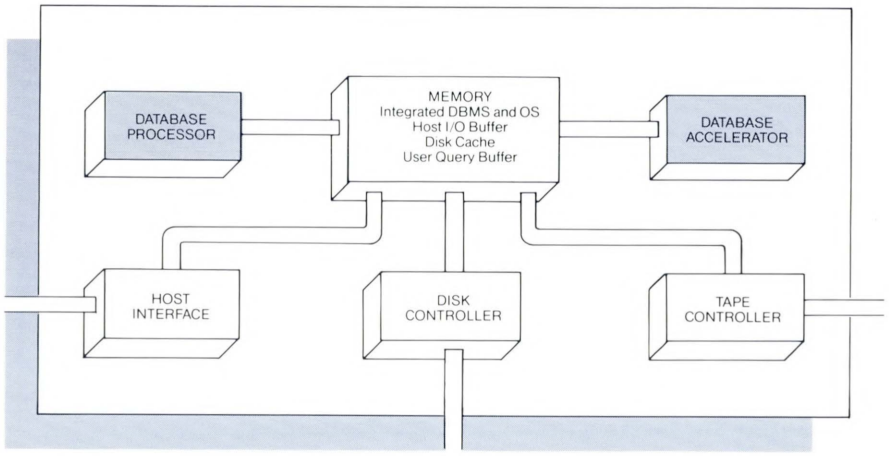
Host interface modules are available with up to 64 RS-232 ports, 4 IEEE 488 channels, 4 Ethernet LANs or 4 IBM block multiplexers.

The serial host interface modules have programmable baud rates of up to 19,200 baud.

Modem control can be enabled or disabled by the host. The parallel host interface module has a maximum data capacity of 170 Kbytes/second. In high transaction rate environments, separate parallel host interface modules can be assigned to each host. This increases parallelism—two hosts can communicate with the IDM simultaneously. In addition, the Ethernet host interface supporting the XNS (Xerox Network Standard) protocol and the IBM block multiplexer interfaces are also available. The Ethernet interface is the IEEE 802.3 specification using the XNS protocol. Britton Lee utilizes Sequence Packet Protocol (SPP) as the communications level in XNS.



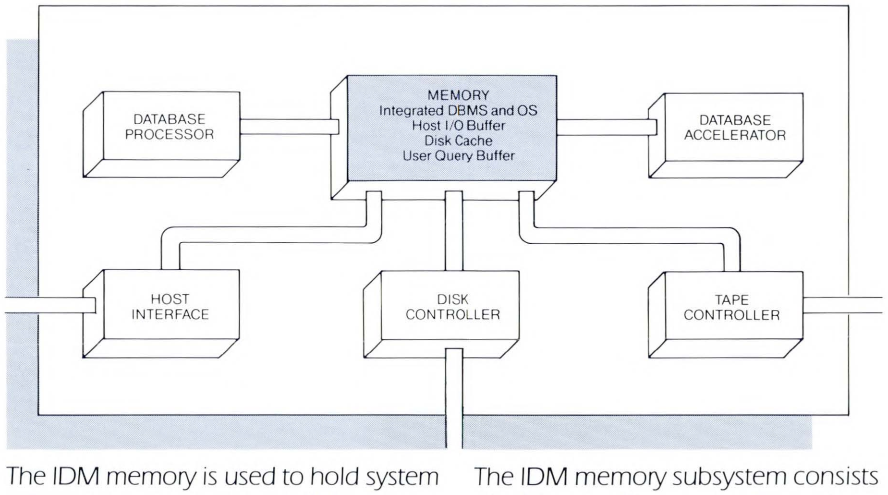
2.3 The IDM Includes Multiple Processors



The Database Processor (DBP) translates among different hosts' data types, manages all system resources and executes most of the software in the system. It evaluates the IDM configuration when the IDM is turned on and takes advantage of all available modules. A high performance DBP option provides 60% higher processing throughput than the standard DBP. In addition, the optional Database Accelerator can greatly enhance throughput of either DBP.

The Database Accelerator is a very high speed processor with an instruction set specifically designed to perform relational database functions. Addition of the Accelerator to the IDM can improve throughput for a factor of 2 to 10 depending on the database operation being performed. This 8 MIPS processor can search memory far more rapidly than the DBP. The Accelerator and the IDM system software are structured so that most of the time-consuming work is performed by the Accelerator under the direction of the Database Processor.

2.4 IDM Cache Memory Improves Throughput



The IDM memory is used to hold system information, its relational DBMS software, most frequently used database pages, user work space and stored commands. The amount of IDM memory can be expanded to optimize IDM throughput. The IDM can address up to 6 megabytes of dynamic RAM memory.

The IDM memory subsystem consists of a Memory Timing and Control module and at least one memory module. Memory modules are available in 1 megabyte increments. The Memory Timing and Control module (MTC) manages the main memory subsystem. It provides single bit error correction and double bit error detection. The MTC controls one or more memory storage boards. The board is pipelined to supply data at the processing speed of the Database Accelerator.

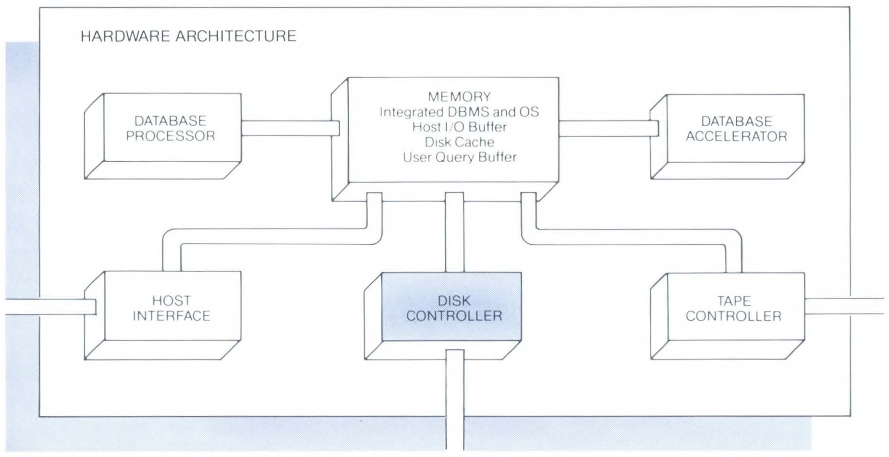
2.5 The IDM Uses Britton Lee SMD and SMDE Disk Controllers

The IDM Disk Controllers move data between external disks and the IDM main memory. They perform burst error detection and correction. The disk controller interface is industry standard SMD compatible. This allows disk drives made by many different vendors to be directly connected to the IDM.

Two types of controllers are available for the IDM; one for SMD disks and one for Extended SMD (SMDE) disks. One controller manages from 1 to 4 disk drives. Each drive can have a different capacity. New drives can be added at any time. Drives used for the database must be dedicated to the IDM and can only be accessed through the IDM.

The IDM can have a maximum of 4 disk controllers. Thus it can accommodate up to 16 SMD or SMDE disks of any capacity. Both SMD and SMDE disks can be accommodated on the same IDM.

The IDM logic has been designed to manage up to 32 billion bytes of disk storage so that, when available, new SMDE disks with increased capacity can be easily accommodated by the IDM. Britton Lee has certified a number of disks for use with the IDM and guarantees their compatibility.



2.6 The IDM Hardware is Built for Data Integrity

The IDM tolerates a variety of common hardware error conditions including uncorrectable memory errors, host-IDM communication errors, disk read/write errors, and intermittent IDM bus errors. Each board, except the memory storage boards, contains built-in diagnostics and a self-test capability. The memory storage boards are tested from the Database Processor Board. Single bit errors can be detected and corrected and double bit errors can be detected. In most cases, bad memory can be mapped out of the system.

To insure bus integrity, parity is used throughout. The use of bus parity and bus operation retry provides stable and reliable interboard communication. The IDM disk controllers have built-in burst error correction and automatically retry and correct disk pages. Before writing to disk, the disk controller checks the status of the AC power. If normal, the write is committed. The IDM power supply is able to maintain DC power for the completion of the write in case of AC power failure.

Every IDM has two special RS-232 ports labeled "console" and "maintenance" which can be used for monitoring the system performance and for performing remote diagnostics. The IDM writes an error log and system status messages to the console port. The maintenance port is used to talk directly to the IDM from a remote location such as the Britton Lee service department. This significantly simplifies problem diagnosis and correction procedures.

The IDM reports errors through use of the FAULT light on the front panel and through a written log on the console. When the IDM discovers that it can not continue to execute, it will self-diagnose problems down to the board level and report which of its boards needs replacing.

Britton Lee offers several maintenance plans which include on-site or depot spare parts and the availability of trained service personnel.

3. Software Architecture

Since IDM products are fully programmed to perform relational DBMS functions, the user never programs the IDM itself. The user simply describes what information he wants i.e., the names of all employees in the Shoe Department. The user never requests information in terms of how or where to get it. The mini, micro, or mainframe host translates the request into IDM format and sends it to the IDM. These two host functions (translating and sending the request) are referred to as the host parser and host driver functions. After the IDM has processed the request, it buffers the results until the host is ready to receive them. The user does not intervene in the IDM's processing of the request in any way.

3.1 Host Interface Software for the IDM

Host driver software is required to handle the communication protocol with the IDM.

A variety of host-dependent/operating system-dependent driver modules are available from Britton Lee or our OEMs. Multi-user parallel drivers are available from Britton Lee for VAX/VMS, VAX/ULTRIX, VAX/UNIX and Apollo Aegis. XNS Ethernet Servers or drivers for VAX/VMS, AT&T Series 3B and IBM PC/DOS are also available. OEM products include drivers for many other CPUs and operating systems.

Host parser software is required to translate user commands to IDM internal format.

The following example illustrates two different query languages (SQL and IDL) being used to access the same information. Both queries are parsed (translated) by the host into the same IDM internal format before they are sent to the IDM for processing.

In order to get from the EMPLOYEES (E) relation, the names and salaries of all employees who work in the Sports Department, a user could enter either of the two commands below.

SQL Query:

```
select name, salary
from employees
where dept_name = "sports"
```

IDL Query (E refers to the EMPLOYEES relation):

```
retrieve (e.name,e.salary)
where
e.dept_name = "sports"
```

The preferred query language determines the choice of parser. A Britton Lee sales engineer can provide the latest information about parser software available from Britton Lee and our OEMs.

To simplify the task of describing the powerful relational database software facilities which are pre-programmed into the IDM products, the examples which follow use Britton Lee's IDL query language. However, it is not necessary that the user choose IDL as his query language or even that he use a query language. When activity within the databases from a particular host is consistent and predictable, an ad hoc query language is really not necessary. Hence parser software would not be needed. Instead, IDM stored commands could be used to standardize the procedure for accessing the data from that host.

3.2 The IDM Relational Database Software

Although the relational data model is relatively new, it has been acclaimed as the most flexible and elegant of the data models. Because it is based on a rigorous mathematical structure, its data can be easily described and accessed. Until there was hardware designed specifically to perform relational DBMS functions, however, it appeared that relational database systems would not be fast enough to be commercially viable. Relational DBMS systems implemented as software running on a host under control of a general purpose operating system were plagued with performance problems. The IDM satisfies the high performance criteria of DBMS systems while offering all the benefits of the relational data model.

3.2.1 The Relational Data Model

The IDM relational database management system organizes data into one or more independent databases. Each database is a collection of tables. Generally all of the tables in one database contain logically related information. For example, a Personnel database might include a table about employees, another about departments and a third about job openings. The user opens a database and then issues a set of commands to access and manipulate the data in the various tables.

A table contains rows and columns. Tables can also be thought of as files, rows as records, and columns as fields. The logical organization of a Personnel database with two tables, Employees and Departments, is illustrated below.

To relate information in one table with information in the other, we could use the DEPT__NAME column.

Additionally the IDM provides a random access file system. IDM random access files are a convenient way to store non-database information such as text which might relate to some tabular information in the database. Random access files can also be used to store executable programs for intelligent terminal hosts. IDM tables can be created and used as file directories and key-word indices into these random-access files.

The IDM users experience virtually no restriction on database capacity. The IDM can support up to 50 databases. A database can have up to 32,000 separate tables. Each table can hold up to 2 billion rows (records). Each row can be up to 2000 bytes long, and can have up to 250 columns. A column can have up to 255 characters.

EMPLOYEES TABLE

LAST__ NAME	INITIALS	SOC__SEC__NUM	SEX	SALARY	START__ DATE	DEPT__ NAME
BROWN	TJ	776-30-4839	F	4000	070380	GARDEN
SMITH	RS	123-79-1122	M	1012	020175	SPORTS
JONES	AM	397-75-2628	M	2080	121678	TOYS
MILLER	RJ	463-55-1120	F	3010	101680	SPORTS
:						

DEPARTMENTS TABLE

DEPT__ NAME	MGR__SS__NUM	MAIL__ STOP	PHC EXT
TOYS	123-22-9750	11	57
SPORTS	463-55-1120	20	26
GARDEN	796-40-3724	77	8
:			

3.2.2 The IDM's Relational DBMS Command Set

Some of the basic functionality supported by the IDM command set is itemized below:

- Create/destroy database
- Create/destroy relation
- Permit/deny access to data in a table
- Create/destroy indices on data
- Insert data into a table
- Modify or delete data in a table
- Select data from a table or tables
- Create/read/write/destroy random access files
- Create/destroy view
- Begin/end transaction
- Define stored command
- Audit changes to a table

Additional DBMS services which are automatically handled by the IDM are:

- Controlling concurrent use of the database
- Read/Write Locking of shared data
- Journaling (logging) of changes to the data
- Managing transactions for degree 3 consistency
- Optimizing throughput
- Checking and enforcing security
- Maintaining data dictionary information

The following examples are given in SQL (Structured Query Language). It is important to note the user need be concerned with only one language. SQL is both a Data Definition Language and a Data Manipulation Language.

CREATING THE DATABASE

Once a logical analysis of the data relationships has been performed, the DBA can begin to create the database and establish the tables.

To create the hypothetical PERSONNEL database previously illustrated in section 1.3 the user could enter the following command:

```
create database personnel
```

If appropriate, the user can specify in the CREATE DATABASE command, how much space should be allocated on individual disks for this database. Otherwise, the IDM will assign the first available space.

Initially the database is empty except for system relations that the IDM automatically maintains as part of its integrated data dictionary. The data dictionary tables will contain current information about the following:

- The name of each table in the database and a description of it
- Each column of a given table, its type, and a description of the column
- All database users and groups of users with their access privileges
- Any changes made to logged tables
- A catalog of all clustered and nonclustered indices associated with any tables

The IDM also maintains a special database which holds information about each of the databases, user-specified parameters, the IDM's environment, and the IDM relational DBMS software.

By using the ad hoc query language, the user can retrieve data from these system tables in exactly the same way that he would access data from his own tables. In addition, a set of predefined stored commands are supplied with Britton Lee host software to simplify access to these system tables.

As tables increase in size, they are expanded in multiples of 2K-byte pages. A page is a basic unit of allocation. The IDM user may control how much space a database or table will occupy on each disk and how it grows.

CREATING TABLES IN A DATABASE

Once created, the database must be opened by the user who will be issuing commands to that database.

```
open personnel
```

The result of an "OPEN" is that the user is assigned a process ID by the IDM so that all work done for that user can be identified uniquely as his.

To create tables in this opened database, the "CREATE TABLE" command should be used. If the DBA wishes to control disk space allocation, he can use options with the CREATE TABLE command to specify:

- That the table being created should be allocated an initial amount of space
- That it should have a space quota beyond which it is not allowed to grow (This quota can be increased at any time by the DBA.)

If all changes made to this table were to be logged, the LOGGING operation would also have to be specified as part of the "CREATE TABLE" command.

There are also commands to extend the size of the database and tables. These "extend" commands have the same options as the original CREATE command.

The result of a "CREATE TABLE" command is an empty table to which rows (records) can be appended. IDM system tables will be automatically updated to indicate the existence of the new table, its owner, its columns, and any other pertinent information about it (indices, protection, etc.)

In order to create the EMPLOYEES table in the PERSONNEL database as exemplified on page 11, the user would enter:

```
create table employees
(last__name char(20),
initials char(2),
soc. sec__num char(11),
sex fixed char(1),
salary fixed bcd(2),
start__date smallint,
dept__name char(15))
```

The EMPLOYEES table was created with the columns of LAST__NAME with maximum of 20 characters, INITIALS with a maximum of 2 characters, SOC__SEC__NUM with a maximum of 11 characters, SALARY which is a 2-byte BCD field, SEX with one character, START__DATE which is a 2-byte integer, and DEPARTMENT__NAME with a maximum of 15 characters.

Each column has a declared type which must be given at the time of the "CREATE TABLE" command. The IDM supports nine data types:

RANK	TYPE
1	char
2	binary
3	tinyint
4	smallint
5	integer
6	bcd
7	flt 4
8	flt 8
9	bcdflt

Conversion is supported to and from the datatypes: bcd, bcdflt, string, tinyint, smallint and integer. Both 4 and 8 byte floating point formats are treated essentially as fixed-length binary strings. A character field (column) can be represented with from 1 to 255 characters. These are automatically in compressed format unless "fixed" is specified as the leading element in the character type, i.e. "fixed char" gives uncompressed character type for fixed-length format.

EXAMPLE OF DATA TYPES

TYPE	DESCRIPTION
char(15)	A maximum 15 byte character field
fixed char(15)	A fixed length 15 byte character field
fixed bcd(7)	A fixed length bcd field with 7 decimal digits
bcdflt(7)	A floating point bcd number with 7 significant digits
tinyint	A 1-byte field between −128 and 127
smallint	A 2-byte field between −32768 and 32767
integer	A 4-byte field between −2,147,483,648 and 2,147,483,647

The "bcdflt" data type is a unique representation in the IDM which allows decimal arithmetic to be performed without the rounding problems of normal floating point arithmetic. For example, many hosts cannot store the decimal number 0.3 and 0.7 precisely. If these numbers are multiplied and if a decimal data type of at least "bcd(2)" has been specified, the IDM will provide the exact result, 0.21. With double precision floating point, the multiplication would have produced an approximate result of 0.2099.

The range of numbers represented for business/financial applications is realistically unrestricted. With the decimal format of the IDM, a company would have to have revenues exceeding a million trillion times the size of the U.S. gross national product with accuracy to the penny to exceed the numerical accuracy of the IDM.

The decimal data type also allows for scientific representation by supporting the use of exponents to achieve a range of numbers from 10** (−1024) to 10**1024. The decimal notation is stated in terms of a significant number of digits from 1 to 31. For example the multiplication 7.1 x 10**13 and 2 x 10**12 will provide the result 1.42 x 10**26 provided the field was defined as "bcdflt(3)."

PLACING DATA INTO TABLES

To place information into the relation, the user may take advantage of the IDM bulk load facility, FCOPY IN, or he can enter information a row (record) at a time using the SORT INTO command. Assume that the user wants to place a new employee row (record) into the EMPLOYEES table. He would enter:

```
insert into employees
      (last__name,
      initials,
      soc__sec__num,
      sex)
values ("Brown",
      "TJ",
      "776-30-4839",
      "F")
```

The row is stored in the table EMPLOYEES. The name, initials and social security fields have the values specified. Since the salary, start date, and department name are not specified, the IDM assigns to them the default values of zero for the numeric fields, and blank for the character fields.

The IDM's COPY IN facility provides a "bulk load" capability. It is useful for initially loading a database as well as to transfer data between the IDM and a host, between two IDMs or between different databases on the same IDM. A COPY OUT facility also exists which can be used to write all tables or selected tables in a database to the host or to an IDM file in a format readable by COPY IN.

SELECTING DATA FROM THE DATABASE

The relational data model provides the user with three basic operators: project, select and join. These operators can be combined with Boolean and comparison operators and aggregation for a precise description of the information desired. The simple SQL command, SELECT, can be used for all these operations.

SELECT

To retrieve data from the EMPLOYEES table, the SELECT command is used as illustrated below:

```
select last_name, salary from
employees
where dept_name = "sports"
order by last_name
```

This command will display the names and salaries of all employees who are in the Sports Department. The ORDER BY clause causes the results to be returned to the host in ascending order by LAST_NAME.

If the user had wanted to see all the names and salaries, he would have eliminated the WHERE clause from the above command.

If the user had wanted to see only the name and salary of the employees whose names started with an "SM" and ended with an "H," he would have entered:

```
select last_name, salary from
employees
where last_name = "sm*h"
```

The IDM supports 3 string manipulation functions: substring, concatenate and pattern matching. In the above pattern matching example, the '*' is a wildcard that means that any number of variable characters may appear in its place. Additional pattern matching features support a range of pattern matching requirements.

The items following the SELECT keyword comprise the target list. This list identifies the items to be displayed. The qualification, WHERE, defines which rows (records) are affected by the command.

Both the qualification and target list can refer to several relations. The following query "joins" the names and initials from the EMPLOYEES table with the mail stop and phone extensions in the DEPARTMENTS table based on the same DEPT_NAME.

EMPLOYEES TABLE

LAST_ NAME	INITIALS	SOC_SEC_NUM	SEX	SALARY	START_ DATE	DEPT_ NAME
BROWN	TJ	776-30-4839	F	4000	070380	GARDEN
SMITH	RS	123-79-1122	M	1012	020175	SPORTS
JONES	AM	397-75-2628	M	2080	121678	TOYS
MILLER	RJ	463-55-1120	F	3010	101680	SPORTS
:						

DEPARTMENTS TABLE

DEPT_ NAME	MGR_SS_NUM	MAIL_ STOP	PHONE_ EXT.	QUOTA	YTD
TOYS	123-22-9750	11	570	300,000	198,000
SPORTS	463-55-1120	20	268	350,000	201,000
GARDEN	796-40-3724	77	853	500,000	314,000
:					

```
select last_name, initials,
mail_stop, phone_ext from
employees, departments where
name = dept_name
```

The processing of multi-table queries is done completely by the IDM. Advanced optimization techniques are used to efficiently execute multi-table queries.

As illustrated the SQL language allows the use of comparison operators as follows:

OPERATOR	DESCRIPTION
=	compares two expressions for equality
>	compares two expressions for greater than
>=	compares two expressions for greater than or equal to
<	compares two expressions for less than
<=	compares two expressions for less than or equal to
!=	compares two expressions for NOT equal to.

Built-in functions which are completely executed by the IDM can also be specified in RETRIEVE statements. They include:

FUNCTION	DESCRIPTION
COUNT	counts the number of tuples containing the values satisfying some expression
AVG	calculates the average of some set of values satisfying some expression
SUM	calculates the sum of a set of attribute values satisfying some expression
MIN	calculates the minimum value of some set of values satisfying some expression
MAX	calculates the maximum value of some set of attribute values satisfying some expression
ANY	indicates if any records meet some qualification
ORDER	orders the response to a query to be sorted by the IDM and presented to the host in that order
IN	set inclusion

To select a count of all employees in the Sports Department, the user would enter:

```
select count (last_name) from
employees where dept_name =
"sports"
```

To select the average salary in the Sports Department, the user would enter:

```
select avg (salary) from employees
where dept_name = "sports"
```

Aggregate functions are different from simple aggregates in that they return a set of values instead of a single answer. For example, to select the department name and the average salary for every department, the user would enter:

```
select dept_name, avg (salary)
from employees group by dept_name
```

DEPT NAME	AVG SALARY
GARDEN	2500
SPORTS	2750
TOYS	1980
.	.
.	.
.	.

The difference in syntax is evidenced by the presence of the GROUP BY clause.

UPDATING INFORMATION

To change data, the IDM supports the UPDATE, DELETE FROM, and DROP commands.

UPDATE

The value for JONES' SALARY in the EMPLOYEES table can be changed through the UPDATE command as follows:

```
update employees
set salary = 5000 where
name = "jones"
```

This changes the salary of all people named JONES to 5000.

To increase only the salary of the specific Mr. A.M. Jones by 1000, the user could have entered the following:

```
update employees
set salary = salary + 1000 where
name = "jones" and initials = "a.m."
```

DELETE FROM

The delete command can be used to delete the JONES tuple (record) from the EMPLOYEES relation as follows:

```
delete from employees where name =
"jones" and initials = "a.m."
```

This command removes the tuple (record) for the employee named A.M. JONES from the EMPLOYEES relation.

Repetitive updates which are composed of several commands and require standard input can most efficiently be handled by using the IDM STORED COMMAND facility. Refer to the section on "Tuning the Database" for more information on STORED COMMANDS.

DROP

To remove the entire EMPLOYEES table, the command is:

```
drop employees
```

PERFORMING TRANSACTIONS

IDM transaction management maintains database consistency over a set of commands. Consistency means that if two or more transactions affect the same data, the results will appear as if only one transaction at a time had been operating on the data. The command SET AUTOCOMMIT OFF is issued. All following commands will be considered part of a single transaction until a COMMIT WORK command is issued. Transactions always appear either to have run to completion or never to have started. This degree 3 consistency is automatically guaranteed by the IDM.

Additionally when making changes to information, it is often advisable to evaluate the results of the transaction before committing the transaction changes to the database. The IDM permits the user to do just that. Once initiating transaction processing, the user stores, accesses, or modifies database information as desired. When he is satisfied with the results of his changes, he can commit the transaction with a COMMIT WORK command. Otherwise, he issues the ROLLBACK WORK statement which will cause the IDM to automatically back out all changes and restore that data to the value it was just prior to the beginning of the transaction.

In the preceding example the user gave a new salary of 5000 to all people named JONES. He could have avoided this mistake in the following manner:

```
set auto commit off
update (salary = 5000)
where last__name = "jones"
select (all)
```

(He would now see the results of his update).

```
rollback work
```

This series of commands would have allowed him to evaluate the results of his changes and rather than commit the transaction, he aborted it.

Many commands could have been included, and the entire group of commands would have been considered to be a single transaction which would be entirely performed or not performed at all. Even if there were a power failure in the middle of a transaction, when the IDM comes back on line, it would detect that the transaction had not been completely performed and would automatically roll-back the data to its previous state.

3.3 Tuning the IDM Database

After the Database Administrator becomes familiar with the access patterns of the database users, he can optimize the database for the typical types of access and updates. To do this he would use the IDM INDEX, VIEW, and STORED COMMAND facilities.

USING INDICES

Data in tables can be accessed quickly if pointers to the data exist; an index is a pointer to the data. There are two kinds of indices: clustered and non-clustered. A clustered index causes the data to be maintained in physical sort order on the clustered index field. This allows a single clustered index entry to point to a range of rows (records). The non-clustered index contains an entry for each row in the table. The command to create a clustered index is:

```
create clustered index on
employees (last__name)
```

This causes the IDM to sort the EMPLOYEES table on LAST__NAME, store in in that order, and to build a modified B*-tree index that contains location identifiers for a range of values of the LAST__NAME attribute in the EMPLOYEES table. When the EMPLOYEES table is accessed with the LAST__NAME specified in the WHERE clause, the index is searched first and the access is made directly to the disk location where that data resides.

There can only be one clustered index per relation. If the table is to be accessed by other columns as well, non-clustered indices may be created on these columns as follows:

```
create nonclustered index on
employees (start__date)
```

This command creates an index that has one entry for each row in the EMPLOYEES table. Each entry contains the value of START__DATE for the employee, and a pointer to the disk location that contains the row with that START__DATE.

At any time the access pattern to information changes, old indices can be destroyed and new ones created. To remove an index, the command is:

```
drop clustered index on
employees (last__name)
or
drop nonclustered index on
employees (start__date)
```

CREATING VIEWS

In a well structured relational DBMS, there should be almost no duplication of data. This insures consistent updating and simplifies maintenance. In the past, however, the same data items would often be found in several different files since each application program required its own record layouts. This need to duplicate data is eliminated by the IDM through its VIEW capability. As the database administrator tunes the IDM database, he will create VIEWS which conform to the record-layouts required by these application programs. Views are hypothetical tables which combine commonly related data from one or more different tables. Views can also be used to restrict access to sensitive data because the user need not be aware that he is only "viewing" data that has been materialized from underlying tables.

Assume that a group of users need to access the EMPLOYEES table but should not see the salary information. The IDM VIEW facility can be used to give them a hypothetical relation from which they can RETRIEVE information but which does not contain the SALARY field. A VIEW is materialized when the user retrieves information from it. A view is created as follows:

```
create view emp
as select last__name,
        initials,
        dept__name,
        start__date
from employees
```

The view EMP created above will contain the employee's name, his department, and his starting date but it will not contain salary information. Data can be retrieved from and protected in the VIEW as if it were a real relation. All changes to the underlying relation, EMPLOYEES, are automatically reflected in the view, EMP.

In the preceding example, the VIEW was a subset of a single table. A VIEW can also be used to define a hypothetical table which combines commonly related fields from several different relations. For example, a monthly business analysis might be a combination of fields from the two different relations, EMPLOYEES and DEPARTMENTS. Assume the monthly report lists the department names, managers' last names and initials and the quota and year-to-date (YTD) information about each department.

EMPLOYEES TABLE

LAST_ NAME	INITIALS	SOC_SEC_NUM	SEX	SALARY	START_ DATE	DEPT_ NAME
BROWN	TJ	776-30-4839	F	4000	070380	GARDEN
SMITH	RS	123-79-1122	M	1012	020175	SPORTS
JONES	AM	397-75-2628	M	2080	121678	TOYS
MILLER	RJ	463-55-1120	F	3010	101680	SPORTS
:						

DEPARTMENTS TABLE

DEPT_ NAME	MGR_SS_NUM	MAIL_ STOP	PHONE_ EXT.	QUOTA	YTD
TOYS	123-22-9750	11	570	300,000	198,000
SPORTS	463-55-1120	20	268	350,000	201,000
GARDEN	796-40-3724	77	853	500,000	314,000
:					

MONTHLY VIEW

LAST_ NAME	INITIALS	DEPT NAME	QUOTA	YTD	
WHITE	TR	GARDEN	300,000	198,000	
MILLER	RJ	SPORTS	350,000	201,000	
BURNS	SL	TOYS	500,000	314,000	
:					

To join fields from the two different tables EMPLOYEES and DEPARTMENTS in order to form one table, the user would enter:

```
create view monthly
as select last_name,
      initials,
      dept_name,
      quota,
      ytd
from departments
where mgr_ss_num =
soc_sec_num
```

Now the user has access to a virtual table listing all departments with their managers' names, quotas, and YTD performance. He can retrieve information from this view MONTHLY as if it were an actual table.

```
select * from monthly
where ytd > quota
```

In the ad hoc query environment, use of VIEWS simplifies accessing information from the database for the non-dp professional. Additionally, use of the VIEW capability makes programs more independent of the data (record layout) in the database and simplifies the programming task.

USING STORED COMMANDS

An efficient way to execute a repetitive group of dependent database commands is to store them as a group in the IDM. If several commands are always to be executed together, they can be sent to the IDM as a command group where they will be pre-processed and stored for later execution. The IDM does all the necessary pre-processing when the command group is stored. Let's define and store a command group in the IDM which will automatically give a 10% pay increase to an employee who is promoted to department manager and will also update the DEPARTMENTS table with the new manager's social security number. We will name this command group, PROMOTE, and when we invoke it, we will send two parameters: the first will be the social security number of the promoted employee and the second will be the name of the department he will manage.

```
store promote
update employees
set salary = salary + .10% salary
where soc__sec__num = $1
update departments
set mgr__ss__num = $1
where dept__name = $2
end store
```

The keywords STORE and END begin and end the command definition respectively. The $1 and $2 are the 'social security number' and 'department name' parameters whose values will be passed to the IDM when the stored command is invoked.

To invoke the PROMOTE command, the statement is:

```
start promote
("123-11-4795", "garden")
```

The keyword START invokes the stored command called PROMOTE. The two parameters will become the values of $1 and $2 respectively. In the EMPLOYEES table, the salary of the employee with social security number 123-11-4795 will be increased by 10% and his 'social security' number will be entered as the manager's number for the Garden department in the DEPARTMENTS table.

The stored command feature is particularly powerful. Once the command group has been parsed and sent to the IDM for storage, the host never has to parse the group of commands again. The host simply sends the command name to the IDM with the appropriate parameters. This reduces the amount of information that must be transmitted to the IDM. It also significantly reduces the time it takes for the IDM to execute the command set since the IDM has preprocessed the command. A side benefit of using stored commands is that application programs running on the host are smaller, more efficient and easier to maintain.

DATA VALIDATION

It is often important to insure that data being entered into a table satisfies integrity or data validation constraints. Data validation can be performed in several ways with the IDM. Stored commands, unique indices and the "select into" functions, can all be used. For example, to insure that updates to the SALARY field in the EMPLOYEES table only occur with appropriate data, the following STORED COMMAND could be used:

```
store upsal
update employees
set salary = salary + $2
end store
```

To invoke this stored command, the user would enter:

```
start upsal ("123-44-4886", 150)
```

The result of this execution would be to give a $150 raise to the employee with social security number 123-44-4886. If the second parameter, 150, had been a negative number, the employee table would not have been updated.

Assume that we have just read an EMPLOYEES database from a tape supplied from another division. We suspect that there may be employee duplications, and that extraneous data may have been stored in the marital status field of some record. Only "S" for single and "M" for married should be allowed. Also, we want to insure that all salaries are non-negative. To ensure the integrity of the data, we could enter the following commands:

```
select distinct into newemp *
from oldtemp
where marital__status = *
"[MS]" and sal > 0
```

In the target list, the * indicates that all the columns of each qualified row should be retrieved. * eliminates the need to name each column separately in the target list.

The SELECT DISTINCT INTO automatically deletes duplicate rows and the qualifier (WHERE) forces a pattern matching to ensure that only M or S will exist in the MARITAL__STATUS field of NEWEMP. Also, the SALARY field must be greater than zero. Erroneous data are ignored and not placed into the NEWEMP table if the qualifications are not met.

Additionally, the DISTINCT option can be specified with the CREATE INDEX command to ensure uniqueness on a key field. For example, employee social security numbers can be guaranteed to be unique by defining a DISTINCT index on SOC__SEC__NUM as follows:

```
create distinct clustered
index on employees
(soc__sec__num)
```

RESTRUCTURING TABLES

As we have seen, the VIEW feature of the IDM lets users and application programs access "hypothetical" relations which contain rows (records) which are consistent with their record requirements. For example, a COBOL program needing NAME, SALARY and HOURS__WORKED need not know that the "materialized" rows it receives from the IDM come from a view which the IDM forms by combining fields from several different relations. Thus views can be seen as a passive way of restructuring the database.

Should you need to actually change the width of a field or add a column to a table, this can also be done. The following example illustrates the EMPLOYEES table before and after using the SELECT INTO command to do a significant amount of restructuring.

- To combine the LAST__ NAME and INITIALS fields into a new field called NAME
- To change the name of the SEX column to TYPE
- To expand the DEPT__NAME field to 20 characters in length and call it DEPT
- To add a new column called "status"
- To eliminate the SOC__ SEC__ NUM field

```
select into new__emp
name = concat
  (last__name, initials)
  type = sex,
  salary,
  start__date,
  dept = substring (1,20,dept__name),
  status = " "
```

After the SELECT INTO is complete, the old EMPLOYEES table could be destroyed and NEW__EMP could be renamed to EMPLOYEES. Alternatively, a CREATE command followed by an INSERT could be used to take advantage of certain options associated only with the CREATE command.

EMPLOYEES TABLE

LAST__ NAME	INITIALS	SOC__SEC__NUM	SEX	SALARY	START__ DATE	DEPT__ NAME
BROWN	TJ	776-30-4839	F	4000	070380	GARDEN
SMITH	RS	123-79-1122	M	1012	020175	SPORTS
JONES	AM	397-75-2628	M	2080	121678	TOYS
MILLER	RJ	463-55-1120	F	3010	101680	SPORTS

NAME	TYPE	SALARY	START__ DATE	DEPT	STATUS
BROWN TJ	F	4000	070380	GARDEN	
SMITH RS	M	1012	020175	SPORTS	
JONES AM	M	2080	121678	TOYS	
MILLER RJ	F	3010	101680	SPORTS	

SECURING THE DATABASE

It is common that some information in the database is sensitive and should only be available to a particular class of user. In addition to using views and stored commands to control access as discussed in the previous section, the DBA can directly protect data from unauthorized access through the use of two protection commands, GRANT and REVOKE. These two protection commands are followed by keyword options which specify what is being permitted or denied and to whom. Activities which can be granted or revoked include read, write, execute, create index, create database and create relation.

REVOKE

Users are denied access to a table based upon their user identifier. The command to deny changing the SALARY field in the EMPLOYEES table to JOHN is:

```
revoke write of employees
(salary) to john
```

The user identified by JOHN is denied permission to change the SALARY field in the EMPLOYEES table.

If the user had wanted to deny both READ and WRITE (referred to as ALL) on SALARY to JOHN, he would have entered:

```
revoke all on employees
(salary) from john
```

Now John could neither read nor write to the SALARY field of the EMPLOYEES relation. If the user wanted to deny ALL (both READ and WRITE) to everyone, he would have entered:

```
revoke all on employees from all
```

PERMIT

To permit READ access to all fields of the EMPLOYEES table just to John, the user would enter:

```
grant read of employees to john
```

To permit a group of clerks to read some fields of EMPLOYEES, the user would enter:

```
grant read of employees
(last__name, dept__name) to clerks
```

In the above examples, let's assume that ALL was revoked to ALL before the GRANT commands were issued. The result of the above GRANT commands is that the entire EMPLOYEES table can be read by the user identified as JOHN, but the CLERKS group can only access the LAST__NAME and DEPT__NAME fields.

For both REVOKE and GRANT, an entire table or separate fields of a table can be specified. Similarly, the use of STORED COMMANDS, which perform restricted functions, can be controlled as follows:

```
revoke start of promote from john
```

This prohibits JOHN from executing the stored command, PROMOTE.

If the owner of a table protects any field of information in his table, this protection is extended to other users' VIEWS or STORED COMMANDS which try to use that protected field. Users or groups of users can also be denied the privilege of creating tables or databases, creating indices or executing stored commands.

4. IDM Back-up and Recovery

4.1 The IDM Transaction Log

The IDM provides facilities to protect stored data from loss as well as unauthorized access. There are two ways data could be potentially lost: hard failures (such as disk head crashes) and soft failures (such as power failure). The IDM protects data in one of two ways depending on the type of failure. Both ways involve use of the IDM transaction log.

Each command to the IDM is a transaction unless it is one of a group of commands terminated by the COMMIT WORK command. Then the entire group of commands is treated as a single transaction. Transactions always appear either to have run to completion or never to have started. To maintain database consistency, the IDM must keep track of all changes to the database. Then the IDM can guarantee consistency by using the before and after images of the changed data to undo any partially completed transactions and to insure that all finished transactions are completely reflected in the database. While the IDM understands that the COMMIT WORK command signals that the transaction should be committed, the IDM also understands that the ROLLBACK WORK command means that all changes to the data should be backed out. A user (or a program) can send a ROLLBACK command instead of a COMMIT command to cause the IDM to use its transaction log to automatically backout all changes made to the database since the SET AUTOCOMMIT OFF command was received.

Every time an event such as a change to a relation takes place, enough change information is automatically recorded in the IDM transaction log to both reconstruct the change and/or to back it out. The IDM transaction log is written from IDM memory to disk before the transaction is committed from IDM memory to the database. A transaction 'done' token is written to the transaction log after the actual transaction is committed to the disk database. These transaction logs are an essential part of the IDM's data protection scheme.

4.2 Soft Crash/Recovery

When the IDM detects a soft crash condition such as a failure condition due to loss of power, network failure, or IDM hardware problems, it is capable of maintaining database consistency.

When the IDM is rebooted after a soft crash, a recovery program is run against each database. This program maintains data integrity by looking into the IDM transaction logs to determine which transactions were only partially complete. These are automatically backed out. This process is speeded by the use of check points noted in the transaction log. A checkpoint forces all data blocks which have been modified to be written out to disk. Check point intervals are set by the DBA. The recovery program can proceed from the log's most recent checkpoint rather than from the beginning of the transaction log.

EXAMPLE OF RECOVERY

Suppose our transaction is an UPDATE that modifies several rows (records) stored in several different disk blocks. Each disk block will be brought into memory and modified; but, for efficiency, these blocks will not be immediately written back to disk. The IDM writes data blocks back to disk during periodic checkpoints or when it must make room in its memory for more data. These modifications are also being automatically recorded in IDM transaction logs. As each row (record) is changed, both the original data and the new values are written into IDM transaction log disk buffers. The IDM guarantees that these transaction logs will be written out before an actual transaction is committed. The log of a change is always written out to disk ahead of the actual data.

In the event of a soft crash, the IDM recovery routine may find transactions in one of four states:

1. The transaction may have been completed and the modified data written out to disk before the crash. Then everything is all right.
2. The transaction may not have been completed, and the data was never written out. Again everything is all right.
3. The transaction was completed, but the database changes were not written out. In this case, the IDM will find a "done" token in the transaction log so it will update the modified database data from information stored in the log.
4. The transaction might not have been completed, but some of the database changes were written out to disk. In this case, the IDM will not find a "done" token in the transaction log. Thus, it will use the before-images of the partially completed transaction log to restore the data to its original state.

In all instances, the transaction is either complete or not applied at all.

4.3 Hard Crash/Recovery

The soft crash recovery procedure requires that the transaction logs are intact. This may not be the case in the event of a hard disk crash. To protect against this, users should follow IDM backup and recovery procedures.

Unlike most systems which do backup and recovery by physical disk device, the IDM performs backup and recovery by database. This allows the IDM to guarantee that the backup of each database is consistent.

The IDM crash/recovery scheme requires occasional backup of the complete database and frequent backups of the transaction logs. The transaction log dump is analogous to the incremental dump used in many operating systems. If the database ever needs to be restored the transaction logs can be reloaded and applied against the complete dump of the database via the ROLLFORWARD utility.

BACKUP PROCEDURES

As previously discussed, the transaction log is updated whenever a change is made to a logged table (as specified at CREATE time). The procedure for handling the possibility of catastrophic loss of data is to:

1. Dump the entire database occasionally.
2. Dump the transaction log frequently.
3. In case of loss of data, load the last database dump.
4. Load and roll the transaction logs forward.

The success of the above procedure requires logging of all tables important to the application.

To backup the PERSONNEL database and its transaction log to the host, the DBA would call the DUMP utility:

```
idmdump personnel working
/transaction = personnel.log
/database = personnel.db
```

However, if the user prefers to maintain his backups under IDM control, he could create a special database for this purpose. Assume a database called BACKUP has been created to hold his transaction dumps. Then to take a backup of the PERSONNEL database to IDM tape and its transaction log to an IDM file, the DBA would enter:

```
idmdump personnel backup
/transaction = aug_chg%ifile
/database = %itape
```

The database PERSONNEL is written to IDM tape and the transaction log is written to the new log called AUG_CHG in the BACKUP database.

Both the database dump and the incremental dump can be sent to one of three places: back to the host where it is normally stored on disk or mag tape, to a file in another database in the IDM, or to the IDM tape drive.

Users should normally do a full database dump shortly after their initial database load or creation. Additional full database dumps should be performed at regular intervals.

Between full database dumps, users should do periodic dumps of the transaction log. The frequency of these dumps depends on how dynamic the databases are, and how valuable the data is. Should a database ever need to be restored, the DBA need only LOAD the most recent full dump and then LOAD and ROLLFORWARD all subsequent transaction log dumps. Dumps of transaction logs and databases can occur while users are actively retrieving and updating the database. After a database dump or transaction log dump, the IDM automatically truncates the transaction log and begins logging anew.

Since only logged objects are backed up by a transaction log dump, IDM users need to know which objects are logged. Users can check the IDM system table called RELATION to see whether or not an object is logged. The "rels" stored command will also report logging status. Tables can be designated as logged or not logged when they are created, or logging may be turned on or off with the ALTER command. The IDM recovery scheme assumes that only logged objects are important.

RECOVERY PROCEDURES

Assume that we are dealing with a multi-disk, multi-database configuration. Assume also that an entire disk is rendered unusable by a disk crash. First, the damaged pack must be replaced. Then the new pack is formatted by the IDM. Any database that partially resided on the crashed disk will be offline and must be restored. The other databases will be online and available while the DBA follows the IDM recovery procedure.

First he recreates the destroyed databases. Then he loads the most recent full database dump. In our example, a copy of the PERSONNEL database was stored on tape. The following commands will load this copy into the newly created PERSONNEL database.

```
idmload personnel
working % itape
```

Then he loads and rollsforward any subsequent dumps of the transaction logs.

```
idmrollf personnel backup aug__chg
```

AUG__CHG was the copy of the transaction log in the BACKUP database. Our example loads the contents of AUG__CHG into the AUG relation created in the new PERSONNEL database. The ROLLFORWARD command applies these changes to the restored PERSONNEL database. A date and time option can be included with the ROLLFORWARD command if only transactions committed by the specified date and time are to be reapplied.

As soon as this is completed, the database is ready to be used. Any updates to the database that were made after the last transaction log dump and before the crash will have been lost. The DBA should define procedures to provide for re-entry of data unavoidably lost between the last transaction log backup and a crash.

APPENDIX 1

The IDM products represent a first in the evolution of database management systems. Our back-end relational DBMS processors, installed in a network or behind heterogeneous hosts, relieve the other units of the burdensome DBMS tasks. Because the IDM products are implemented in specialized hardware, they provide an attractive price/performance ratio when compared with software DBMS products running on general-purpose CPUs.

IDM SYSTEMS products that combine an IDM with host-resident interface software are designed for DEC VAX computers running VMS or UNIX, AT&T Series 3B computers running UNIX, Apollo workstations running Aegis, IBM System 370 computers running VM/SP, and PCs supporting MS/PC-DOS.

Installation service, hardware and software warranty and maintenance are all offered by Britton Lee in support of its IDM products.

IDM CAPACITIES

SPECIFICATION	BASE CONFIGURATION		EXPANDABLE TO
IDM Memory		1 Mbyte	6 Mbyte
Disk Storage	supports up to	4 SMD drives	16 SMD or SMDE drives
Host Interface	supports up to	8 Hosts	100+ Hosts
Tape		not included	support 2 transports
Database Accelerator		not included	1 Accelerator
Number of databases		50 +	50 +
Tables per database		32,000 +	32,000 +
Columns per table		250	250
Rows per table		2 billion +	2 billion +
Indices per table		251	251
Columns per index		15	15
Index type		B*tree	B*tree
Number of Users		128 + +	400 + +

+ Depends on available disk storage
+ + Depends on available memory

APPENDIX 2

SUMMARY OF QUERY COMMANDS

SQL/IDL COMMAND	DESCRIPTION
Rollback work	Causes transaction to be aborted
Insert	Adds tuples to a relation
Connect	Used to add information in the description catalogue
Audit	Creates audit report from transaction log
Set autocommit off/ commit work	Marks beginning and end of multiple IDL commands to be considered one transaction
Create	Used to create databases, relations, IDM random access files, indices, and views
Store	Defines a stored command
Delete	Used to remove tuples from a relation
Revoke	Denies access to information in a relation by user or group
Drop	Used to eliminate databases, relations, files, views, stored commands, and indices
Start	Executes a stored command
Alter	Increases or decreases space allocation for a relation or database
Open	Opens a database or IDM file for activity
Grant	Permits access to a relation or attribute(s) by user or group
Range	Associates a variable name with a relation or view
Update	Replaces one or more attributes in zero or more tuples of a relation
Select	Retrieves data from a relation and sends it to the host or puts it 'into' a new relation

APPENDIX 3

SUMMARY OF DATA DICTIONARY RELATIONS

TABLE NAME	DESCRIPTION
Relation	The Relation table is a catalog of all objects in the data-base: tables, views, files and stored commands
Attribute	Lists the attributes of each relation and their data definitions
Indices	Catalog of indices that exist in a database, the relation being indexed, and the attributes included in the index
Protect	Contains protection information including type of access, for which attributes of what object (relation, view, stored command, or file) for which user or group
Query	Used by IDM to hold precompiled queries and views
Crossref	Catalog of dependencies among relations, views and stored commands
Batch	Temporary transaction's logging relation for transaction management
Users	Mapping of user and group names to IDM user id
Blockalloc	Catalog of disk blocks showing relations assigned and free space
Disk__Usage	Shows relation and database allocation
Databases	Catalog of databases on the system
Disks	Lists of disks known to the system
Lock	Used by IDM for 'read' and 'write' lock protection
Configure	Contains information about I/O interfaces, checkpoint intervals, and monitor intervals
Dbinstat	Contains information about users currently "signed on".
Transact	Permanent transaction log/audit trail
Host__Users	Mapping of host identification number to IDM user id
Descriptions	Stores comments about relations and attributes
Monitor	System monitor, shows CPU usage, memory use, and I/O activity
Devmonitor	System monitor, shows I/O activity for each device
Account	Accounting data

Britton
Lee, Inc.

Britton Lee, Inc.

14600 Winchester Boulevard
Los Gatos, California 95030
(408) 378-7000
Telex: 172-585