

MIMER 8

Product Overview

INTRODUCTION	3
<i>Product Strategy</i>	<i>3</i>
<i>Technical Benefits Overview</i>	<i>4</i>
PERFORMANCE	6
<i>Database Server Architecture</i>	<i>6</i>
<i>Server Architecture Benefits</i>	<i>7</i>
<i>The Database Cache</i>	<i>7</i>
<i>Cleanup Handling</i>	<i>8</i>
<i>Stored Procedures</i>	<i>8</i>
<i>SQL Optimiser</i>	<i>9</i>
<i>Secondary Indices</i>	<i>10</i>
<i>Transaction Management</i>	<i>10</i>
<i>Storage Structures</i>	<i>12</i>
EASE-OF-USE	16
<i>Database Administration</i>	<i>16</i>
SECURITY	18
<i>Access Security</i>	<i>18</i>
<i>Backup and Recovery</i>	<i>20</i>
24 X 7 OPERATION	22
<i>Resilience</i>	<i>22</i>
<i>Product Quality</i>	<i>22</i>
OPENNESS	23
<i>SQL</i>	<i>23</i>
<i>ODBC</i>	<i>24</i>
<i>Web-based Database Applications</i>	<i>25</i>
<i>Java and ActiveX</i>	<i>26</i>
<i>Client/Server - Heterogeneous environments</i>	<i>27</i>
<i>Data Types</i>	<i>28</i>
<i>Platforms</i>	<i>29</i>

INTRODUCTION

MIMER is a Relational Database Management System (RDBMS) developed by Sysdeco Mimer AB in Uppsala, Sweden.

MIMER is a high performance database engine, targeted at mission-critical production environments. MIMER offers scalable performance, including multi-processor support, and with its availability on all major UNIX, Windows NT, Windows 95/98 and OpenVMS platforms, it is ideally suited for open environments where interoperability is important.

MIMER's run-time characteristics, such as high performance, ease-of-use, stability and self-tuning, makes it the perfect choice for software products with an embedded DBMS and for Internet/Intranet/Extranet applications where a maximum of database up-time and a minimum of database administration often is a requirement.

Through its 100% conformance to the SQL standards and a native implementation of Microsoft's ODBC (Open DataBase Connectivity) interface, a MIMER database can be accessed by a large number of different development tools. MIMER is also well suited for mission-critical multi-tier solutions, since its conformance to standards makes it compliant with many transaction processing middleware products based on CORBA or OTM techniques.

Product Strategy

The main trend in the computer industry in the last 20 years has been the increased interoperability between hardware and software components, resulting in heightened competition between vendors. One example of this is the different UNIX platforms now available, where because of a standardised external behaviour the competitive edge is now the internal functionality. In the RDBMS arena, what has now become evident is the separation between pure RDBMS functionality and the development tools (e.g. 4GLs), allowing customers to choose the software environment by selecting different components.

The MIMER product strategy recognises this interoperability between the RDBMS and the different development tools. For this reason, MIMER conforms to the existing RDBMS standards and *offers connections to all the major development and production environments*. As a result the performance and quality of both the database engine and the connections are of strategic importance to Sysdeco Mimer.

In contrast to most other RDBMS products, MIMER is exactly the same product on all platforms (no "light" version for PCs), from a small laptop running Windows 95/98, to a large UNIX Enterprise server. This makes it possible to develop an application with the database stored on your laptop, and then easily move the database to a larger server without having to change a single line of code.

MIMER is characterised by the following five key features:

- High performance
- Ease-of-use
- Security
- 24 x 7 Operation
- Openness

... which when combined with Sysdeco Mimer's competitive pricing provide the customer with an

attractive total economy, not just through the development phase but through the entire application life-cycle.

Technical Benefits Overview

MIMER is a mature and highly advanced Relational Database Management System, which has been available as a commercial product for more than 20 years. It offers a number of features, many unique to MIMER, which provide significant advantages in five key areas:-

High Performance

- Multi-threaded requester-server architecture based on threads, which makes it ideally suited to symmetric multiprocessor (SMP) environments as well as single processor machines.
- Stored procedures for optimal performance in client/server environments.
- Compiled SQL-queries and stored procedures are shared between MIMER-users, thus minimising the number of compilations performed.
- Automatic on-the-fly extension of database files, when required, without any costly manual formatting of new areas.
- Highly optimised B*-tree structure used for all tables, giving rapid access for keyed and sequential retrievals, and exceptionally high space utilisation with no fragmentation of free space.
- Automatic continual re-balancing guarantees perfectly balanced B*-tree structures with no noticeable delays for users, and no possibility of corrupt tree structures even following a hardware failure.
- Advanced SQL optimiser uses statistics to ensure queries use most efficient access routes.
- Group commit transactions make optimal use of the available I/O-capacity.
- Ability to perform Read-Only transactions for optimal performance when executing transactions which makes no updates to the database.
- Transaction management using Optimistic Concurrency Control, a method pioneered by Sysdeco Mimer, which overcomes many of the problems of conventional locking techniques such as deadlocks, and locks being retained by defunct connections, whilst offering superior performance.
- Since deadlocks cannot occur, there is no need for deadlock detection overhead.
- ODBC implemented as a native driver integrated into the product, rather than as an add-on layer.

Ease-of-use

- Automatic self-tuning, means a limited number of tuning parameters, and so removes the need for highly skilled staff.
- Fully automated complete recovery from system failures.
- Use of standard OS files for all database storage allows OS utilities to be used (e.g. for backups), and allows advantage to be taken of the latest storage technologies as they become available.
- Dynamic re-organisation of the database during run-time, thus avoiding any manual intervention for re-organising the database.
- Optimistic concurrency control guarantees that no deadlocks can occur.

Security

- Advanced security facilities offering fine-grain access control, and allow access using an application to be differentiated from ad-hoc access.
- System and object privileges controls access to database administration functions and objects in the database.
- Full set of integrity restraints to ensure logical database consistency.
- Backup and recovery functionality guarantees that a consistent and up-to-date database always can be restored after a disk crash.
- Audit trail utility that provides information about performed transactions in the system.

24 x 7 Operation

- Database structures always optimally organised and therefore never any requirement to carry out database reorganisations.
- Shadowing facility which allows backups to be taken on-the-fly, and automatic recovery following disk hardware failure.
- Extremely stable behaviour in run-time environments, which removes the need to stop the database for manual intervention.

Openness

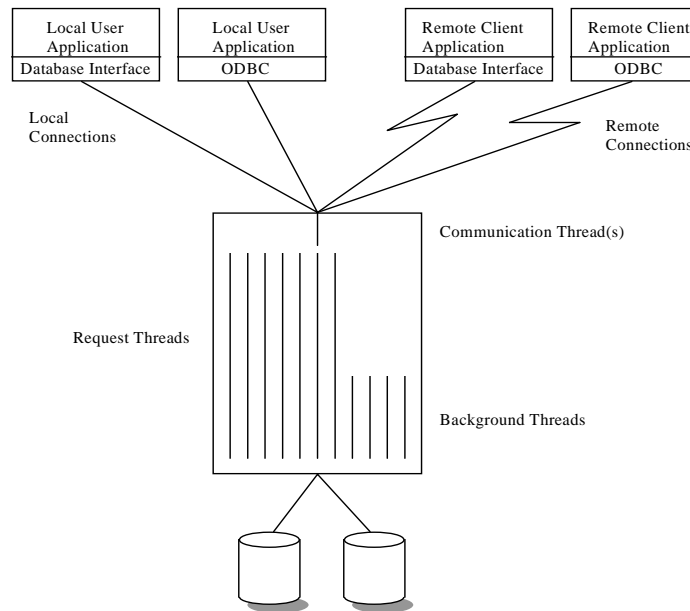
- Maximum conformance to SQL standards.
- The first DBMS on the market with support for ISO's standard for stored procedures, SQL/PSM.
- Compatible with all major web and client/server application development tools through the ODBC interface.
- Compliant with many middleware products for transaction processing, like Tuxedo and Microsoft Transaction Server.
- Available on a wide range of platforms including UNIX and Windows.

PERFORMANCE

Database Server Architecture

The MIMER DBMS is based on a client/server architecture. The Database Server executes in one single, multi-threaded process with multiple **Request and Background threads**. On some platforms, one or more **Communication threads** are used. The MIMER architecture is truly multi-threaded, with requests being dynamically allocated to the different Request threads. As threads scales very well over multiple CPUs, MIMER is particularly well suited for symmetric multiprocessor (SMP) environments. By the use of threads within the Database Server, optimal efficiency is achieved when context-switching in the Database Server.

The use of a separate Database Server Process guarantees that a program error in an application process cannot corrupt the shared data areas which control the database server.



MIMER Database Server Architecture

The **Communication thread(s)** are used on some of the MIMER platforms to handle parts of the communication between the applications and the database server. It is implementation defined whether Communication threads are used or not on a specific platform, and also if one or more Communication threads are used. On some platforms other mechanisms are used to handle the communication between the applications and the database server.

Both local and remote applications are handled directly by the Database Server. This means that in **Client/Server** environments, where MIMER executes in a distributed environment with the client and server on different machines, all remote clients connect directly to the Database Server. Thereby avoiding any additional overhead of network service processes being started, neither on the client nor on server machine.

The **Request threads** perform the SQL operations requested by the applications. When the Database Server is requested to perform a SQL operation it allocates one of its Request threads to

perform the task. When the SQL operation is complete the result is returned back to the application, and the Request thread that has performed the operation returns to a waiting state until it receives another server request. Since the SQL operations are evaluated entirely within the Database Server, inter-process communication is reduced to a minimum.

When a SQL query or a stored procedure is executed by a Request thread, the compiled version of the query or the procedure is stored within the Database Server. In this way the same, compiled version of the query or procedure can be used again by other applications. This leads to improved performance, since a SQL query or a stored procedure only need to be compiled once by the Database Server.

The **Background threads** perform database services such as transaction completion and database shadowing. These services are performed asynchronously in the background to the application processes, which means that the application process does not have to wait for the physical completion of a transaction or a shadow update, but can continue as soon as the transaction has been prepared and secured to disk.

I/O-operations are performed in parallel directly by the server and shadows threads using asynchronous I/O. Thereby any need for separate I/O-threads are avoided.

Server Architecture Benefits

The threads-based implementation of the MIMER Database Server performs particularly well in Symmetrical Multi-Processing (SMP) environments, since threads are very efficiently scheduled to the different CPUs by the operating system. As a result of this, this architecture allows the different processes to run concurrently on different processors, and so parallel execution is achieved.

The RISC architectures implement an instruction-level synchronisation technique (Load-linked/Store-conditional) which dramatically reduces the number of operating system calls required and therefore the number of context switches performed when there are several concurrently executing processes. This facility is used by MIMER in these environments.

These Load-linked/Store-conditional synchronisation techniques are derived from Optimistic Concurrency Control and have the same benefits of improved performance and freedom from deadlocks. The combination of these technologies therefore, delivers systems with excellent performance.

Due to its simple and well-designed architecture, MIMER offers a highly performant option for production environments. The performance benefits of MIMER provide better response times and an increased utilisation of computer resources.

The Database Cache

The **Database Cache** of MIMER (sometimes also called the Bufferpool) ensures that large parts of the database can be kept in main memory, reducing the number of disk I/O operations needed. The size of the Database Cache can easily be changed, making it possible to effectively utilise available main memory. The Database Cache is accessed only by the Request threads.

The Database Cache is partitioned, which means that each partition can be viewed as an independent memory area. Since there are in effect a number of Database Caches, tasks executing simulta-

neously on separate CPUs have a reduced risk of conflicting when accessing a partition and so inter-processor cache coherency traffic is reduced.

Cleanup Handling

To handle cleanup when users abort their applications without disconnecting from MIMER, the Database Server is configured to interact with the Operating System on the server machine. When a user application is interrupted without a proper disconnect, the Operating System will send a signal to the Database Server. When the Database Server receives this signal it performs a cleanup for that application. The MIMER Server performs the following at a cleanup:

1. First it cancels any database requests from the application that are currently being executed.
2. Then it closes all tables that the application held open.
3. Finally all transactions that were going on for the application are rolled back.

If a remote client is switched off without disconnecting from the Database Server, MIMER makes use of the `KEEPALIVE` functionality in TCP/IP and DECNET to be notified that a cleanup operation is required. In such cases it depends on the value of the `KEEPALIVE` time interval parameter on the server machine, as to how soon MIMER will be notified to perform the cleanup. Observe that since MIMER uses optimistic concurrency control, no locks are held during this time.

Stored Procedures

The most important new functionality in MIMER 8 is the support for PSM (Persistent Stored Modules), ISO's new standard for Stored Procedures (ISO/IEC 9075-4:1996(E), December 15, 1996). Now, applications can be developed using stored procedures without compromising the conformance to standards.

MIMER is the first RDBMS product with support for stored procedures according to the SQL/PSM standard. Most RDBMS vendors have supplied some kind of language for stored procedures for the last couple of years. But since no specified standard has existed in this area, the different vendors have had their own different variations.

The concept of stored procedures is very useful in a RDBMS. Using stored procedures allows application logic to be moved from the applications to the RDBMS. This implies a number of important benefits when developing database applications:

- Thin clients are achieved as a result of moving program logic from the applications to the database.
- Business rules can be stored in one place and does not have to be duplicated in all the applications.
- Performance is improved, since less communication with the application process is needed to perform the operations in the stored procedure.
- Giving the users execution rights to a set of stored procedures instead of giving them access rights to the database objects directly, eliminates the risk of accidental damage of data through interactive tools.

- DBMS access can be standardized by creating a set of stored procedures by which all database access is performed.

The SQL/PSM standard does not include any specification for how a result set should be returned from a stored procedure. But since such functionality is of great importance, a solution for handling of result sets has been included in MIMER/PSM. A special type of procedures, so called result procedures are used to handle result sets. Result procedures make it possible to declare a cursor for a procedure. Such a cursor can be used just like a cursor for a SELECT statement, i.e. it is opened, rows are fetched, and finally the cursor is closed. The result procedure returns one row every time it is invoked by the FETCH statement, until there are no more rows to return. The rows can be returned from the result procedure either explicitly by a RETURN statement, or implicitly by using a direct SELECT statement.

Below follows an example of a stored procedure that enters the charge for lodging on a hotel guest's bill:

```

CREATE PROCEDURE ADD_LODGING (IN IN_RESERVATION INTEGER)
MODIFIES SQL DATA
BEGIN
  DECLARE P_PRICE, P_DAYS INTEGER;
  DECLARE P_CHECKIN DATE;
  --
  -- FIND PRICE OF ROOM
  --
  SELECT PRICE INTO P_PRICE
  FROM ROOM_PRICES, BOOK_GUEST
  WHERE BOOK_GUEST.RESERVATION = IN_RESERVATION
  AND ROOM_PRICES.ROOMTYPE = BOOK_GUEST.ROOMTYPE
  AND ROOM_PRICES.HOTELCODE = BOOK_GUEST.HOTELCODE
  AND FROM_DATE <= CURRENT_DATE
  AND TO_DATE >= CURRENT_DATE;
  --
  -- FIND LENGTH OF STAY
  --
  SELECT CAST((CHECKOUT-CHECKIN) DAY AS INTEGER), CHECKIN
  INTO P_DAYS, P_CHECKIN
  FROM BOOK_GUEST WHERE RESERVATION=IN_RESERVATION;

  BEGIN
  DECLARE P_COUNTER INTEGER DEFAULT 0;
  WHILE P_COUNTER < P_DAYS DO
    INSERT INTO BILL VALUES
      (IN_RESERVATION,
      CAST(P_CHECKIN+CAST(P_COUNTER AS INTERVAL DAY)
      AS TIMESTAMP),
      '100',
      P_PRICE);
    SET P_COUNTER = P_COUNTER+1;
  END WHILE;
  END;
END

```

Example of a stored procedure

SQL Optimiser

The SQL optimiser utilises table statistics to determine the method to be used to execute an SQL statement. This includes determining the order to access tables in and which indices to use, eliminat-

ing the need for the programmer to perform the optimisation. This is especially important for complex SQL queries, and queries using views. The optimisation is performed at run time, meaning that changes in large dynamic databases are automatically catered for.

Secondary Indices

Within MIMER a secondary index is implemented by creating an internal table invisible to the user. The same algorithms and structures are used for storing a secondary index as are used for an ordinary table. The primary key of the index table is the column(s) defined as the secondary index combined with the primary key of the base table. The secondary index tables are fully maintained internally within MIMER. The algorithm used for this guarantees that the indices can never be out of step with their base table.

Secondary indices will be automatically used by the SQL optimiser whenever there is a performance benefit.

Transaction Management

MIMER uses a method for transaction management called Optimistic Concurrency Control (OCC). OCC was innovated by the MIMER developers, though the first research paper on the method was published by Kung and Robinson (in 1981) who were also developing the method at the time. MIMER's pioneering work makes it the first RDBMS to use this method for transaction management. However, several Object Orientated Databases, which were more recently developed, have incorporated OCC within their designs to gain the performance advantages inherent within this technological approach.

Though optimistic methods were originally developed for transaction management the concept is equally applicable for more general problems of sharing resources and data. They have therefore been incorporated into several recently developed operating systems, and many of the newer hardware architectures provide instructions to support and simplify the implementation of these methods.

Optimistic Concurrency Control does not involve any locking of rows as such, and therefore cannot involve any deadlocks. Instead it works by dividing the transaction into phases.

- The **Build-up** phase commences the start of the transaction. When a transaction is started a consistent view of the database is frozen. This means that the application will see this consistent view of the database during the entire transaction. This is accomplished by the use of an internal **Transaction Cache**, which contains information about all ongoing transaction in the system. The application "sees" the database through the Transaction Cache. During the Build-up phase the system also builds up a **Read Set** documenting the accesses to the database, and a **Write Set** of changes to be made, but does not apply any of these changes to the database. The Build-up phase ends with the calling of the COMMIT command by the application.
- The **Prepare** (or the **Check**) phase, involves using the Read Set and the Transaction Cache to detect access conflicts with other transactions. A conflict occurs when another transaction alters data in a way which would alter the contents of the Read Set for the transaction that is checked. Other transactions that were committed during the checked

transaction's Build-up phase or during the Prepare phase can cause a conflict. If a transaction conflict is detected, the checked transaction is aborted and the Commit phase is not carried out. No rollback is necessary as no changes have been made to the database. An error code is returned to the application which can then take appropriate action. Often this will be to retry the transaction without the user being aware of the conflict.

- If no conflicts are detected during the Prepare phase, the **Commit** phase is performed. Now the operations in the **Write Set** for the transaction are moved to another structure, called the **Commit Set** which is to be secured on disk. All operations for one transaction are stored on the same page in the Commit Set. Before the operations in the Commit Set are secured on permanent storage, the system checks if there are any other committed transactions which can be stored on the same page in the Commit Set. After this all transactions stored on the Commit Set page are written to disk (to the transaction databank TRANSDB) in one single I/O operation. This behaviour is called a **Group Commit**, which means that several transactions are secured simultaneously. When the Commit Set has been secured on disk (in one I/O operation), the application is informed of the success of the COMMIT command and can resume its operations.
- During the **Apply** phase the changes are applied to the database, i.e. the databanks and the shadows are updated. This phase is carried out by the Background threads in the Database Server. Once this phase is finished the transaction is fully complete. If there is any kind of hardware failure which means that MIMER is unable to complete this phase, it is automatically restarted as soon as the cause of the failure is corrected.

Optimistic Concurrency Control (OCC) has a number of advantages including:

- Complicated locking overhead is completely eliminated.
- Deadlocks cannot occur, so the performance overheads of deadlock detection are avoided as well as the need for possible system administrator intervention to resolve them.
- Programming is simplified as transaction aborts only occur at the Commit command whereas deadlocks can occur at any point during a transaction. Also it is not necessary for the programmer to take any action to avoid the potentially catastrophic effects of deadlocks, such as carrying out database accesses in a particular order. This is particularly important as potential deadlock situations are rarely detected in testing, and are only discovered when systems go live.
- Data cannot be left inaccessible to other users as a result of a user taking a break or being excessively slow in responding to prompts. Locking systems leave locks set in these circumstances denying other users access to the data.
- Data cannot be left inaccessible as a result of client processes failing or losing their connections to the server.
- Delays caused by locking systems being overly cautious are avoided. This can arise as a result of larger than necessary lock granularity, but there are also several other circumstances when locking causes unnecessary delays even when using fine granularity locking.

Through the Group Commit concept, which is applied in MIMER, the number of I/Os needed to secure committed transactions to the disk are reduced to a minimum. The actual updates to the database are performed in the background, allowing the originating application to continue.

The ROLLBACK statement is supported but, because nothing is written to the actual database during the transaction Build-up phase, this involves only a re-initialisation of structures used by the transaction control system.

Another significant transaction feature in MIMER is the concept of **Read-Only transactions**, which can be used for transactions that only perform read operations to the database. When performing a Read-Only transaction, the application will always see a consistent view of the database. Since consistency is guaranteed during a Read-Only transaction no transaction check is needed and internal structures used to perform transaction checks (i.e. the Read Set) is not needed, and for this reason no Read Set is established for a Read-Only transaction. This has significant positive effects on performance for these transactions. This means that a Read-Only transaction always succeeds, unaffected of changes performed by other transactions. A Read-Only transaction also never disturbs any other transactions going on in the system.

Storage Structures

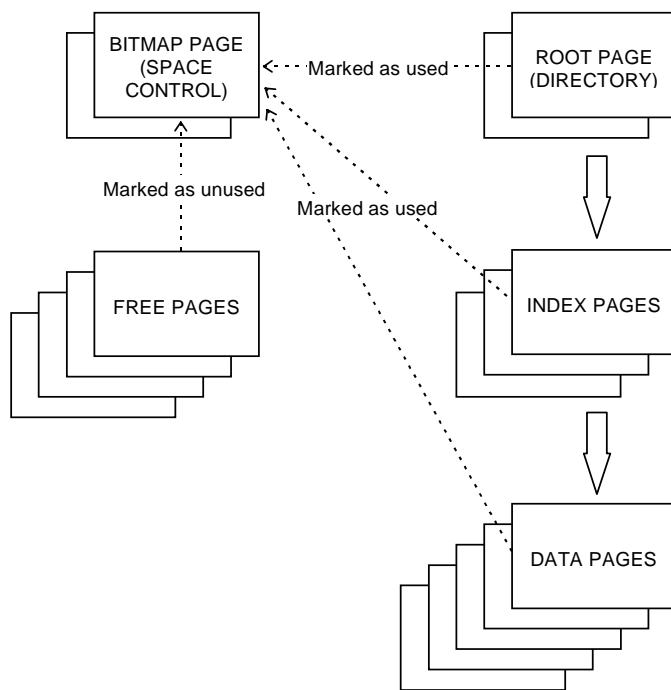
Fundamental to any database system is how the data is stored. In MIMER information is stored in **Databanks**. A databank is a standard Operating System direct access file and contains an arbitrary number of tables. A MIMER database may contain an arbitrary number of databanks at the discretion of the systems administrator.

Within a UNIX environment, databanks can also be set up as 'Raw Device' files which are often more efficient than ordinary direct access files.

MIMER performs data transfers between disk and the Bufferpool on a page basis (the page size used depends on the record size of the table, but will be either 2, 16 or 64 Kbytes). Each databank has a bitmap to indicate which pages are used and which are available. The bitmap can be regarded as a directory of space utilisation. If a table requires a new page, this page is marked as in use in the bitmap. If a page is 'released' (e.g. when deleting data), it is then marked as free.

The use of bitmaps allows the internal structure of the databank to change, without any requirement for a table to be allocated a fixed size in the databank.

When creating a new databank, MIMER automatically creates the bitmap and the root page. The root page is effectively a master directory of all the tables in the databank. If the initial root page is not large enough to contain the entire table directory then additional root pages are automatically created and linked to the initial page. In the same way, if the databank is large, additional bitmap pages are created. All other pages in a databank are either free or used to store index or data pages.



Internal Structure of a Databank

A MIMER table is stored in a highly optimised balanced tree structure called a B*-tree. This method offers fast access to all data for both indexed and sequential searches. By using only one access method for all types of data the MIMER DBMS has been highly optimised for this method, always providing an optimal performance.

The B*-tree consists of an index section and a data section. The rows of the table are stored in the data section (i.e. the 'leaf nodes' of the tree). The index section (the 'non-leaf nodes') is essentially a map to enable rapid physical location of the required page on the next level.

The top level index for a table is identified via the root page of the databank containing the table. The root page is the directory of the tables in the databank and contains page number references to the B*-tree structure.

Rows in a MIMER table are identified by the values of their primary keys, which comprises of one or more columns in the table. It is these values which are used in the index pages in the B*-tree.

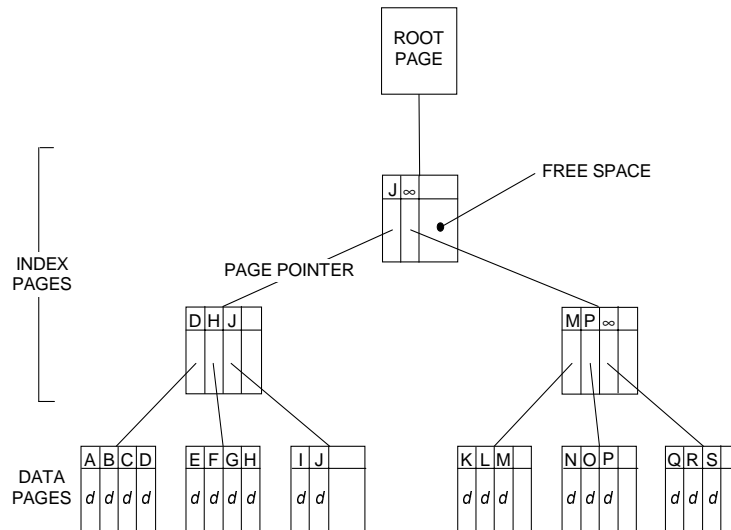


Table Structure (B*-tree)

Inside all pages including the data section, data is kept sorted according to the key values. This makes it possible to use a fast binary search algorithm to find a row, or to find the page where a row should be inserted. By holding the data section pages in sorted order the rows in are automatically clustered by the key.

When inserting new rows the tree grows, and when deleting rows the tree gets smaller. Sometimes this will mean that a page will become full and have to be split, or will be empty enough for the data in it to be able to be merged with adjacent pages and the page to be released. The splitting or merging is known as a reorganisation.

The algorithm used for reorganisation is a pre-emptive top-down scheme using a 'careful replacement' technique. If an insertion procedure encounters a full node in searching for the insertion position, this node is split and the node at the previous level is updated to reflect the split. The higher node cannot itself be full (otherwise it would have been split when the insertion procedure first encountered it), so the splitting effect does not propagate upwards through the tree. When the insertion search reaches the leaf node, the row can be inserted there and the operation is completed. A similar algorithm is used for deletions.

The 'careful replacement' protocol ensures that the permanent storage holds a consistent version of the B*-tree structure at all stages during the reorganisation. The split versions of the node are written to new pages taken from the free pages. When these writes to disk are complete, the node at the higher level is updated to reflect the change and written to disk. After this is completed, the old version of the node which required splitting is marked as free. Even if there is a machine failure during the execution of the reorganisation, the careful replacement protocol ensures that the disk version of the tree will never be inconsistent.

Reorganisations are performed automatically by MIMER, totally eliminating the need to perform manual reorganisations, whilst still ensuring that the tree is always kept perfectly balanced giving continuous optimal performance. The reorganisations are always small involving only the branch of the tree which is being traversed so there is no noticeable effect on response time for the user.

To further improve space utilisation, MIMER compresses data before it is stored in the B*-tree structure. This also improves performance, since disk I/O is an expensive operation, and when the data pages are compressed more records can be stored in one page. In this way, more records are read from (or written to) disk in one I/O operation, which has positive effects on performance.

The B*-tree structure can handle very large quantities of data in relatively shallow trees. It also does not suffer from any fragmentation of free space within either data or index pages.

EASE-OF-USE

Database Administration

Database administration with MIMER is characterised by being greatly simplified and by wherever possible using OS system facilities. Indeed in most cases normal operational control takes care of the database automatically thereby minimising the amount of specialist knowledge required.

The number of tuning parameters in MIMER is, intentionally, very limited. The two most important tuning parameters in MIMER are the size of the Database Cache and the number of Request threads. Both these parameters can easily be determined after only a few days run-time experience. On some UNIX platforms, the use of 'Raw Device' databanks also significantly improves performance. To further improve performance, it is advisable to store the transaction databank (TRANSDB) on a disk separate from the user databanks.

The simplicity by which the MIMER system can be tuned removes the need for highly skilled RDBMS experts to supervise the operations, significantly reducing the maintenance costs for MIMER-based systems.

Most RDBMS require that the contents of the database have to be reloaded regularly to maintain performance, which will mean that the database is unavailable to the users and may require a high level of expertise to actually perform. As previously described, a MIMER database uses B*-trees to store all tables. These expand and contract dynamically as required, and the algorithms used during this process ensure that data is always stored in an optimal way.

The use of B*-trees means that there is never any need to reorganise a MIMER database, or any benefit to be gained by so doing, since the B*-tree structures are always kept well balanced. This feature is especially important as the database size grows. Consider, for example, the implications of a re-load of several Gigabytes of data in a production environment.

By implementing the relational model with separate B*-tree structures for each table and secondary index, MIMER also ensures that even when application enhancements mean that the database schema needs to be altered, this can be achieved by only altering those tables affected. Also the use of the relational model and the implementation of views means that existing application code need not be affected by underlying changes to the database schema as long as the data required by the application is still available.

MIMER's use of a careful replacement protocol during the dynamic table reorganisations and of the similar protocol used when updating tables with secondary indices ensures that these structures are always kept free of inconsistencies even in the event of a system failure. The use of these protocols means that there is no need for any repair utilities to correct such inconsistencies.

As previously discussed, MIMER databanks are standard operating system files. The use of bitmaps to control free space means that there is minimal initialisation required when a databank is expanded. MIMER databanks therefore expand dynamically when required as long as the operating system allows this, which usually means as long as free disk space is available.

The fact that MIMER databanks are standard OS files allows them to be moved using standard OS commands. Moving databank files may be necessary to make the best use of available disk space, or to balance disk loads.

The use of standard OS files also allows operating system facilities such as disk striping, solid state disk devices, RAID systems, and networked drives to be used where these are available without requiring any special facilities to be used within MIMER.

The Concurrency Control techniques utilised by MIMER for transaction handling also eliminate the requirement for any DBA intervention to resolve deadlocks or other lock related problems such as those caused when a client fails.

SECURITY

Access Security

Through the advanced security facilities of MIMER, the database can be protected from any unauthorised access. The fine-grained security system provided by MIMER enables data to be protected down to a single element (row/column) for each user accessing the system.

A unique feature of MIMER is the “role concept”, where the access rights for a user can be increased under password protection. The role concept allows MIMER’s security system to distinguish between users who are accessing the database from the controlled environment of an application, and users who are using ad-hoc tools. The role concept is provided by MIMER through the PROGRAM ident.

The following facilities are available for ensuring the integrity of a MIMER database:

- Four different types of authorised Idents
- Privilege system restricting the use of database administration functions
- Fine grained data access control system
- Stored procedures
- Views
- Domains
- Entity integrity (non-NULL primary keys)
- Referential integrity (foreign keys)
- Table integrity
- View integrity

Within MIMER an **Ident** is an authorised user of the system. It can also be a collective identity of a group of users sharing common privileges.

Four types of idents are supported:

- **USER** idents - authorised to log on to MIMER. User idents are generally associated with specific individuals authorised to use the system.
- **OS_USER** idents - a type of User ident with the same user name as in the operating system. If a user who has logged onto the operating system is also defined to MIMER as an OS_USER, then that user may log into MIMER without providing an additional user name and password.
- **PROGRAM** idents - may not log directly onto MIMER, but instead an ident who has already logged on may adopt the role of a Program ident by using the ENTER statement. To do this the ident requires Execute privilege on the Program ident and has to supply the password for the Program ident to the ENTER statement. Once a Program ident is entered, the privileges held by the Program ident apply. Program idents are generally associated with specific functions like running an application, rather than with physical individuals. This allows end users to carry out updates to the data in the controlled environment of an application, without being able to do the same using an interactive tool.
- **GROUP** idents - are collective identities for groups of idents. Any privileges granted to or revoked from a Group ident automatically apply to all members of the Group. Group idents provide a facility for organising the privilege structure in the database system.

When an Ident connects to MIMER in a client/server environment, the password for the ident is encrypted on the client side. This means that only encrypted passwords are sent over the network, to assure that no unauthorised users can get a hold on a password by tapping the network.

Each ident is given privileges within the system defining the operations that ident is allowed to perform. An ident receiving a privilege 'WITH GRANT OPTION' may pass the privilege on to another ident.

System privileges give the right to create global objects within the database. There are two system privileges:

- DATABANK - gives the right to create databanks
- IDENT - gives the right to create ids

Object privileges give rights over certain specified objects in the system. MIMER supports the following object privileges:

- TABLE - gives the right to create tables in a given databank
- EXECUTE - gives the right to execute a specified stored procedure, or to enter a given program ident
- MEMBER - grants membership in a specified group ident
- USAGE - gives the right to use a given domain

Object privileges are initially granted only to the creator of the object. Privileges may be revoked by their grantor.

Access privileges give rights of access to the contents of a specified table or view. There are five access privileges:

- SELECT - gives the right to read the table or view contents
- INSERT - gives the right to add new rows to the table or view
- DELETE - gives the right to remove rows from the table or view
- UPDATE - gives the right to update existing rows in the table or view
- REFERENCES - gives the right to use the primary key of the table as a foreign key from another table

Access privileges are initially granted only to the creator of the table or view. The privilege may be passed on to other ids with or without grant option

Stored procedures can be used so that users who require access to certain tables or views from an application, can gain access without having any explicit access rights to the actual objects. Instead the users can be granted the right to execute a stored procedure (with or without GRANT option) which accesses those objects. The user creating a stored procedure must, of course, have sufficient access rights to all objects involved in the procedure. Using stored procedures in this way allows you to create an environment where the users are forced to only perform database operations through a series of well-debugged procedures.

A **View** is the result set from a predefined SELECT statement. A view may simply be a restriction of a single table, or may involve the joining of two or more tables (a so called 'join view'). A view can be used anywhere where a table may be used. Views with an implicit one-to-one relationship with an underlying table may be updated.

Views are a powerful tool for restricting user access to defined parts of the database, and complement the system of access privileges in maintaining database security. By defining restriction views (i.e. views based on one table but including only specified rows and/or columns in the table), access

privileges may be granted to subsets of table contents without affecting the physical database structure. Join views can be used to create the 'Universal Relations' which are used by many applications, from a normalised database.

Domains are used to define user-defined data types. By assigning the data type of a table column to be a domain when the table is created, the values which the column may hold are restricted to the set of values defined for the domain.

Entity integrity refers to the requirement that every row in a table must be uniquely identified and that no row in a table may be identified by NULL (i.e. an unknown value). According to Dr. E. F. Codd, entity integrity should be enforced in all true relational database systems. However, the SQL standard allows tables to be created without a primary key and in consequence MIMER conforms to this standard.

Referential integrity refers to the requirement that where appropriate the data entered in one table in the database must already be present in another table (e.g. a component row may not specify a manufacturer id if that manufacturer does not exist in the manufacturers table). MIMER supports referential integrity through the optional FOREIGN KEY clause of the CREATE TABLE statement.

Table integrity refers to the facility in MIMER of defining CHECK-clauses in table definitions, whereby the contents of one column is checked against a list of acceptable values or against the contents of one or more other columns in the same row. Data may only be entered into the table if the CHECK-constraint is not violated.

View integrity means that if a view is created WITH CHECK OPTION this indicates that any data inserted into the view will be checked for conformity with the definition of the view.

By utilising MIMER's advanced facilities for access control and security much coding in applications is avoided and all applications utilise a consistent set of controls.

Backup and Recovery

MIMER's backup and recovery procedures are designed to guarantee that a consistent and up-to-date database can be recreated following all types of system failure.

MIMER handles database consistency is handled on two levels: physical and logical.

Physical consistency means that the tables are readable by the MIMER database manager. The MIMER system guarantees physical consistency as long as the databank file is not physically damaged. If a databank file was not closed properly last time it was accessed, an internal consistency check is performed when the databank is opened the next time. If physical errors are detected the databank can be restored from a backup copy.

Logical consistency means that the tables contain correct data and no transactions are incomplete. MIMER's transaction handling ensures logical consistency. Details of all database accesses are saved in the TRANSDB databank during build-up and no changes are made to the database. The Commit command initiates the application of the changes which are carried out as if they were a single 'atomic' change. If a transaction is successfully committed then all operations in the transaction are applied. If the transaction is aborted due to a conflict, or a user request, none of the operations in the transaction are applied.

The databanks may be temporary logically inconsistent if MIMER is stopped (either deliberately or by a system failure) before all operations in a successfully committed transaction have been performed. When the system is restarted all uncompleted transactions are read from TRANSDB and are automatically applied to get the system into a consistent state again.

During the Commit Phase of a transaction the changes to a databank are written to the transaction log (the databank LOGDB). MIMER's logging system allows a backup copy of a databank to be rolled forward to recreate an up-to-date version of the databank following a file being physically damaged or other hardware faults making it inaccessible or corrupt.

As a complement to backup copies it is possible to archive the transaction log changes into an Incremental Backup file. This allows several backup copies of a databank of varying ages to be kept, along with the LOGDB entries for that databank in a sequence of Incremental Backup files. By using the appropriate sequence of Incremental Backups and the current LOGDB, it is possible to roll forward any of the backup copies of the databank to the current state. By using Incremental Backups you avoid LOGDB growing too large, and they can generally be produced more quickly than making a full backup copy of a databank.

As MIMER databanks are ordinary operating system files, the normal operating system facilities for backing up files can be used for them. Also incremental backups can be combined with host system backup copies. This means that the backup process can be incorporated into the normal site procedures for backup. It also allows the most efficient utilities to be used for this purpose.

The LOGDB file can also be used as an **audit trail**. The audit trail is accessed by a utility, where the following information about performed transactions in the system can be retrieved:

- The user who performed the transaction
- Type of transaction
- Date and time the transaction was carried out
- Row images before and after the transaction

24 x 7 OPERATION

The MIMER RDBMS is characterised by extremely high availability, with no requirement for manual database reorganisations or other maintenance operations that would cause down-time for the database applications.

The only other remaining reason for database down-time is taking backups. With MIMER's shadowing facility this, too, is eliminated. MIMER therefore ensures that the database remains operational 24 hours a day, 7 days a week without any disturbances to the production environment. In the event of a hardware failure (e.g. disk crash), a shadowed database will continue to be available, without any interruption for the application. MIMER is the database that *never* stops.

Resilience

Database Shadowing means that the database works with two, or more, copies of the database information at the same time. One copy - the master - is the 'normal' file from which data is retrieved. The other copy or copies are shadows. Alterations made to the master data are also made to the shadows, so that a shadow is always an up-to-date copy of the master. These alterations are done by applying the alteration separately to each shadow so that any physical corruption of the master are not transferred to the shadow.

In MIMER, shadowing is controlled at the databank (file) level. Any databank where transaction control is in force can be shadowed to as many copies as required. Important data can be protected against a disk crash without having to rely on frequent conventional backups.

Should there be a disk failure on the disk where master databanks reside, operations automatically continue towards the shadows. As there is no time-consuming process of restoring the database from backup files held on tape or other media the application can be kept running without any disturbance. The database manager can choose a suitable moment to transform the shadows to master databanks, and create new shadows.

MIMER's database shadowing allows conventional backups to be taken from the shadow 'on-the-fly', without stopping the database or interrupting work on the master databank. Any changes made to the master during the backup process are buffered in the system and are written, by the Background threads, to the shadow when it becomes available again.

During normal running the shadowing has no impact on response times. Changes made to the master databank are performed in the background to the shadows by the Background threads, without impacting the users.

Product Quality

MIMER is an extremely robust product, which is a necessity for 24 x 7 operation. MIMER is also a very mature product, which has been used for more than 20 years in heavy run-time environments all over the world. Sysdeco Mimer has also concentrated on ensuring that the design of the MIMER system is kept as simple as possible whilst still offering optimal performance.

The quality of the MIMER product is assured by the use of a documented and established development process which has been used for many years at Sysdeco Mimer. This development process assures that inspections of code and documentation, as well as execution of hundreds of thousands rows of automatic test suites, are performed before a new product is released.

OPENNESS

Sysdeco Mimer is committed to conforming to the relational database standards defined by organisations such as ANSI, ISO and X/Open. Sysdeco Mimer's policy means that MIMER has a unique openness towards independent development and middleware tools. With MIMER's native driver for Microsoft's ODBC interface a large number of web-based and Windows-based development tools can be used together with MIMER (e.g. Tango, Delphi, Visual Basic, Access, Centura and many more). MIMER is also compliant with many CORBA- and OTM-based middleware products.

The ODBC-interface within MIMER is implemented at the same level as the SQL-interface, providing excellent performance when ODBC is used with MIMER.

MIMER 8 is also the first RDBMS available, which has support for ISO's standard for stored procedures, **SQL/PSM**.

By using MIMER's standardised Embedded SQL interfaces for C, COBOL and FORTRAN, it is possible to develop portable 3GL applications.

SQL

Sysdeco Mimer's policy is to develop MIMER, as far as possible, in accordance with the established standards. The following SQL standards have been defined and accepted:

- ISO SQL1 (in some literature called SQL89)
- ISO SQL2 (in some literature called SQL92)
- ISO SQL Persistent Stored Modules (SQL/PSM)
- XPG3 SQL
- X/Open SQL 1992 (referred to here as X/Open-92)
- X/Open SQL 1995

Additionally, the Application Programming Interface ODBC defined by Microsoft, has been widely accepted as a de facto standard. While any one standard defines a workable version of SQL, there are few if any vendors who supply database managers defined entirely within the bounds of a single standard. MIMER combines many of the advantages of the different standards.

MIMER 8 conforms to the following standards:

- ISO SQL1
- ISO SQL2
- ISO SQL/PSM
- XPG3 SQL
- X/Open SQL 1992
- ODBC level 2 API, Extended SQL

The primary goal of the XPG3 and X/Open-92 standards is to allow applications to be moved between database systems from different vendors that are X/Open compliant.

The following table lists the lower limits that compliant systems should support (if a box is empty the standard does not specify a limit):

LIMIT	XPG3	X/Open-92	MIMER
Character string length	240	254	15000
Variable length character string	240	254	15000
DECIMAL precision in digits	15	15	45
FLOAT precision in digits	15	(47 bits****)	45
SMALLINT precision in digits	5	5	5
INTEGER precision in digits	10	10	10
REAL precision in digits		(21 bits****)	7
DOUBLE PRECISION precision in digits		(47 bits****)	15
Number of columns in a table	100	100	252
Number of columns in an index	6	6	32
Number of tables referenced in a statement	10	10	unlimited *
Number of cursors open simultaneously	10	10	unlimited **
Number of columns a single update statement can update		20	unlimited *
Number of nested subqueries		9	unlimited *
SQL statement length in bytes		4000	32767
Number of columns in a GROUP BY clause	6	***	unlimited *
Number of columns in an ORDER BY clause	6	***	unlimited *
Number of columns in a secondary index	6	6	32
Total length of a row	2000	2000	16000
Identifier length	8	18	18

* There is a limit on the total complexity of an SQL statement. The specified X/Open limits can easily be met by MIMER.

** The limit is only dependent on the amount of virtual memory available to the application.

*** There is a limit on the total complexity of the clause. The number of columns depends on the size of the columns.

**** This is in fact less than the MIMER limit.

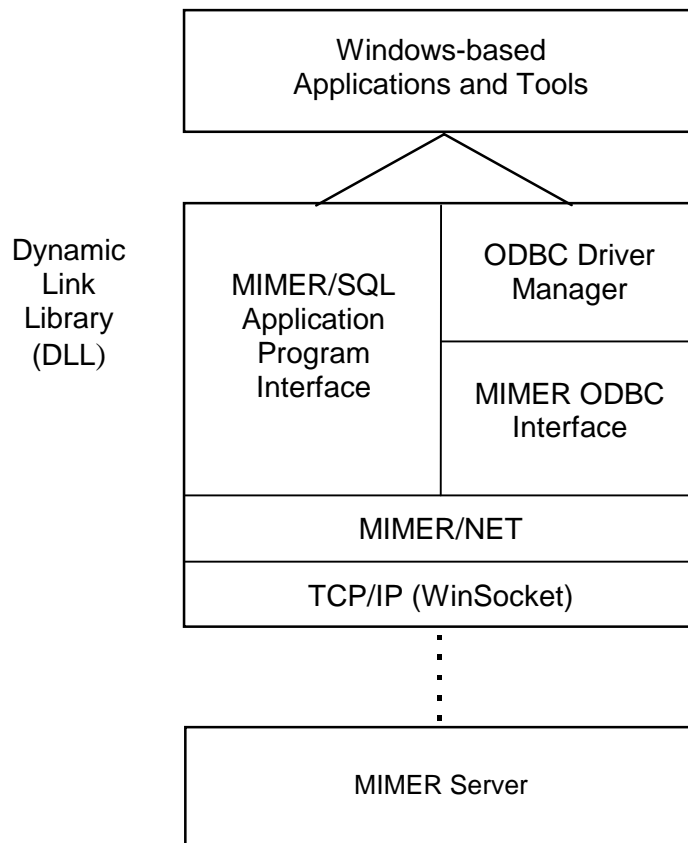
ODBC

MIMER ODBC is a very efficient implementation of Microsoft's Open DataBase Connectivity interface. MIMER ODBC enables Windows-based applications and tools to be connected to the MIMER Database Server. In the MIMER architecture the ODBC interface is placed at the same level as MIMER's standard SQL-interface, eliminating any overhead due to data conversion and additional layers. ODBC-specific features, such as asynchronous access, and block transfer of data which are not included in the traditional SQL interfaces are supported and fully implemented by MIMER ODBC. These additional features are particularly advantageous in a client/server environment and when using a GUI such as Windows.

MIMER ODBC is totally integrated into the Windows environment, because it:

- Is installed as a Windows DLL (Dynamic Link Library)
- Does not use any part of the DOS address space
- Is administered through Microsoft's ODBC Administrator
- Can be installed on a file-server, facilitating administration in large networks

MIMER ODBC includes the MIMER/NET client software for communication with MIMER servers.



Client/Server Architecture

Web-based Database Applications

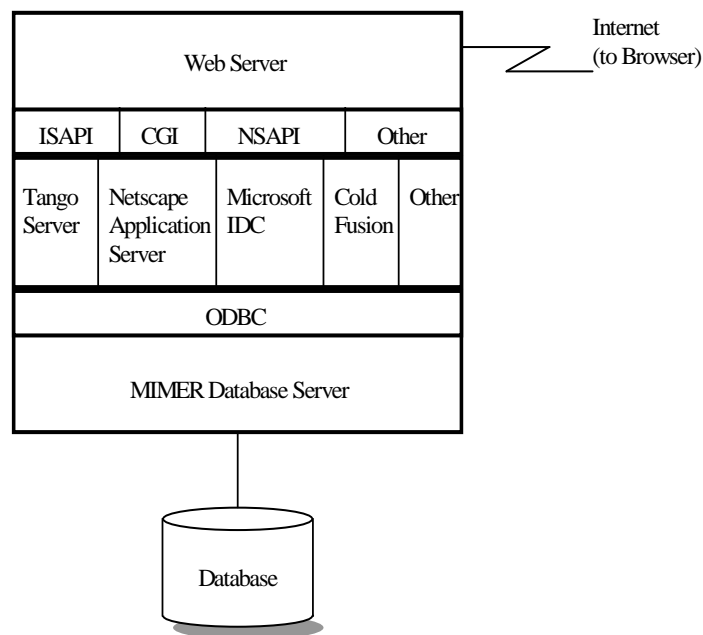
MIMER's stability in run-time environments and the built-in support for ODBC makes it ideal for building dynamic web-based database applications. Data can be extracted from a MIMER database and included in web applications. Web applications can also be used to implement user dialogues for database transactions.

The connection between the web application and the MIMER database can be made either directly from the web browser, or from the web server, or by using some kind of web application server middleware.

The most convenient way to connect your web application to a MIMER database is to use some web application server middleware. There is a variety of middleware products available which link one of the server interfaces and ODBC to provide document to database connections. MIMER has been utilised with a variety of these including:

- Tango Application Server
- Microsoft Internet Database Connector
- Netscape Application Server
- Cold Fusion

For connection at the web server either the (standard) Common Gateway Interface (CGI) or proprietary Server interfaces such as Microsoft's Internet Server API (ISAPI) or Netscape's Server API (NSAPI) may be used. These allow applications to be written using most programming languages. Commands embedded either in the web application, or in the request from the browser cause the applications to be called. The applications can read data which was returned with a request from the browser, and can write to a HTML page before it is sent to the browser. The applications can easily access a MIMER database using either embedded SQL or the ODBC interface.



Architecture for Web Server to MIMER database connections

Java and ActiveX

Connection to a web browser using MIMER's ODBC interface can be made using the Java DataBase Connectivity (JDBC) interface. JDBC to ODBC bridges are available from suppliers such as SUN, Microsoft and Intersolv. These allow MIMER database calls to be made by Java applications.

Alternatively, Microsoft have proposed that ActiveX controls can be embedded in web documents. As many ActiveX applications also support ODBC for database access, MIMER can also be connected in this way.

Client/Server - Heterogeneous environments

Access to remote databases in client/server environments is totally transparent within MIMER. An application, developed against a local database, can be directed to access a remote database ***without changing one single line of code***. In the application, databases are referred to by a logical name. These are mapped onto actual databases (either local or remote) by the MIMER system, using mappings set up by the database administrator.

By introducing a logical database concept, the physical location of the database is hidden from the user. When an application connects to a database by its logical name, the database location and communication protocol to be used are determined by the MIMER system.

MIMER supports heterogeneous environments, where UNIX, Open/VMS, Windows NT, Windows 95/98 computers can be freely mixed on a network. MIMER's use of a standardised format for data storage eliminates any need for conversion in the data transfers between servers and clients, making the communications very fast and efficient.

One single application can also access several different databases (local or remote) simultaneously.

When running in Client/Server mode MIMER uses either TCP/IP, Named Pipes or Pathworks (DECnet) protocols. MIMER Clients utilise the WinSocket 2.0 standard for TCP/IP communication.

Data Types

Explicit data type references are made in SQL statements for the creation of user defined domains and base tables and in the alteration of table definitions. The permissible data types and their allowable ranges within MIMER are:

Data type	Description	Range
CHARACTER(n)	Character string, fixed length n	$1 \leq n \leq 15000$
CHARACTER VARYING(n)	Variable length character string with max. length n	$1 \leq n \leq 15000$
INTEGER(p)	Integer numerical, precision p	$1 \leq p \leq 45$
SMALLINT	Integer numerical, precision 5	-32768 through 32767
INTEGER	Integer numerical, precision 10	-2,147,483,648 through 2,147,483,647
DECIMAL(p,s) *	Exact numerical, precision p, scale s	$1 \leq p \leq 45$ $0 \leq s \leq p$
FLOAT(p)	Approximate numerical, mantissa precision p	$1 \leq p \leq 45$ Zero or absolute value 10^{-999} to 10^{+999}
REAL	Approximate numerical, mantissa precision 7	Zero or absolute value 10^{-38} to 10^{+38}
FLOAT	Approximate numerical, mantissa precision 15	Zero or absolute value 10^{-38} to 10^{+38}
DOUBLE PRECISION	Approximate numerical, mantissa precision 15	Zero or absolute value 10^{-38} to 10^{+38}
DATETIME	Represents an absolute point in time, depending on sub-type.	Described below **
INTERVAL	Represents a period of time, depending on the type of interval.	Described below ***

* The keyword NUMERIC may be used instead of DECIMAL.

** DATETIME can be one of the sub-types DATE (YYYY-MM-DD), TIME (HH:MM:SS[.sF]) or TIMESTAMP (YYYY-MM-DD HH:MM:SS[.sF]).

*** There are basically two kinds of INTERVAL: YEAR-MONTH and DAY-TIME, and many variations of these.

All numeric data and intervals may be signed. The 45 digit numeric precision also extends to arithmetic, making MIMER ideally suited for applications where high numerical precision and accuracy are required.

Columns which contain an undefined value are assigned a NULL value. Depending on the context, this is represented in SQL statements either by the keyword NULL or by a host variable associated with an indicator variable.

MIMER supports the CAST function, which explicitly converts between data types.

Platforms

The MIMER RDBMS is available on all the major UNIX, Windows and Open VMS platforms. The following platforms are supported *:

Intel Windows NT

Intel Windows 95/98

Intel SCO

IBM RS6000/AIX

Bull/AIX

Hewlett-Packard HP-UX

SUN Solaris

Data General DG-UX

Alpha AXP/Digital UNIX

Alpha AXP/Windows NT

Alpha AXP/OpenVMS

Macintosh (client only)

* The list above includes both MIMER 7 and MIMER 8 platforms. Check with your local MIMER distributor for the current availability of MIMER V8 on your platform.

Modelling a world of information

Sysdeco Mimer is a member of Sysdeco ASA - a Norwegian company operating internationally. Sysdeco ASA has sales and support offices in the United Kingdom, Norway, Singapore and Sweden. Professional agents and distributors cover other major countries in Europe, North and South America and South East Asia.

Sysdeco Mimer AB

Box 1713
Kungsgatan 64
751 47 UPPSALA
Sweden
Tel: +46 18 18 50 00
Fax: +46 18 18 51 00

Sysdeco Technology Ltd

Beacontree Plaza
Gillette Way
Reading, RG2 0BS
United Kingdom
Tel: +44 118 921 06 58
Fax: +44 118 921 06 59

Sysdeco Technology AS

Oslo, Norway
Tel: +47 22 09 65 00
Fax: +47 22 09 65 01

Sysdeco Aurocomp Asia Pte Ltd

Singapore, Singapore
Tel: +65 293 4828
Fax: +65 293 7604

Everyware Development Inc

Toronto, Canada
Tel: +1 905 819 1173
Fax: +1 905 819 1172

Löwenfels Partner AG

Luzern, Schweiz
Tel: +41 4141 03923
Fax: +41 4141 08022

Mimer Hellas Ltd

Athens, Greece
Tel: +30 1 201 0909
Fax: +30 1 201 0911

Formula Open Soft

Dordrecht,
The Netherlands
Tel: +31 78 673 6341
Fax: +31 78 673 6528

O.S.I. de Guatemala

Guatemala City,
Guatemala
Tel: +502 332 6973
Fax: +502 332 0459