

# Mimer SQL 8.2

## Technical Description

<b>Introduction.....</b>	<b>3</b>
<i>Product Strategy.....</i>	<i>3</i>
<i>Technical Benefits Overview .....</i>	<i>4</i>
<b>Ease-of-use.....</b>	<b>6</b>
<i>Database Administration .....</i>	<i>6</i>
<b>Performance .....</b>	<b>8</b>
<i>Database Server Architecture .....</i>	<i>8</i>
<i>Server Architecture Benefits.....</i>	<i>9</i>
<i>The Database Cache.....</i>	<i>10</i>
<i>Cleanup Handling.....</i>	<i>10</i>
<i>Stored Routines .....</i>	<i>10</i>
<i>Triggers .....</i>	<i>11</i>
<i>SQL Optimizer .....</i>	<i>12</i>
<i>Secondary Indices.....</i>	<i>12</i>
<i>Transaction Management.....</i>	<i>12</i>
<i>Storage Structures.....</i>	<i>14</i>
<b>Security .....</b>	<b>18</b>
<i>Integrity.....</i>	<i>18</i>
<i>Access Security.....</i>	<i>20</i>
<i>Backup and Recovery.....</i>	<i>21</i>
<b>24 x 7 Operation.....</b>	<b>23</b>
<i>Resilience .....</i>	<i>23</i>
<i>Product Quality.....</i>	<i>23</i>
<b>Openness .....</b>	<b>24</b>
<i>SQL .....</i>	<i>24</i>
<i>JDBC.....</i>	<i>26</i>
<i>ODBC.....</i>	<i>26</i>
<i>Web-based Database Applications.....</i>	<i>26</i>
<i>ActiveX.....</i>	<i>27</i>
<i>Client/Server - Heterogeneous environments.....</i>	<i>27</i>
<i>Data Types.....</i>	<i>29</i>
<i>Platforms.....</i>	<i>30</i>

## INTRODUCTION

Mimer SQL is a Relational Database Management System (RDBMS) developed by Upright Database Technology AB in Uppsala, Sweden.

Mimer SQL is a high performance database engine, providing an ideal data management solution across the full range of computing environments - from a workstation to enterprise 3-tier architecture systems. It offers a unique level of scalability, including multi-processor support, and with its availability on Windows, OpenVMS, Linux and major UNIX systems, it is perfectly suited for open environments where interoperability is important.

Mimer SQL's run-time characteristics, such as ease-of-use, high performance, stability and self-tuning, makes it the ideal choice for a wide variety of software products, including those that require an embedded DBMS and large-scale Internet and intranet client/server applications. With its simple installation, maintenance-free operations and no requirement for a database administrator it is a 'black box' without the maintenance, price, complexity or hardware requirements of other enterprise-class databases.

Through its 100% conformance to the SQL standards including pre-processors for Embedded SQL (according to the SQL-99 standard), a JDBC™ Type 4 driver and a native implementation of Microsoft's ODBC (Open Database Connectivity) interface, a Mimer SQL database can be accessed by a large number of different development tools. Mimer SQL 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. The Enterprise JavaBean component model is founded on CORBA technology; examples of CORBA application servers include BEA's WebLogic and Inprise's Application Server.

### ***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 standardized external behavior 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, allowing customers to choose the software environment by selecting different components.

The Mimer SQL product strategy recognizes this interoperability between the RDBMS and the different development tools. For this reason, Mimer SQL 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 Upright Database Technology.

In contrast to most other RDBMS products, Mimer SQL is exactly the same product on all platforms (no "light" version for PCs or Linux), from a small laptop running Windows 98/Me, to a large UNIX enterprise-class 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 make any modification to the application code or database.

The following five key features characterize Mimer SQL:

- Ease-of-use
- Performance
- Security
- 24 x 7 Operation
- Openness

... which when combined with Upright Database Technology's aggressive pricing provides the customer with a significantly reduced total cost of ownership (TCO). By conforming to the international database standards Upright Database Technology are protecting your development investment, ensuring portability and interoperability.

## **Technical Benefits Overview**

Mimer SQL is a mature and highly advanced Relational Database Management System, which is in use in mission-critical systems throughout the world. It offers a number of features, many unique to Mimer SQL, which provide significant advantages in five key areas:

### **Ease-of-use**

- Automatic self-tuning, means a limited number of tuning parameters, and so removes the need for highly skilled staff.
- Dynamic re-organization of the database during run-time, thus avoiding any manual intervention for re-organizing the database.
- Automatic on-the-fly extension of database files, when required, without any costly manual formatting of new areas.
- Fully automated complete recovery from system failures.
- Optimistic concurrency control guarantees that no deadlocks can occur.
- 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.
- Can be packaged with the applications to allow a single installation of both application and database software.
- Installed in a few minutes.
- Small footprint.

### **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.
- Advanced SQL optimizer uses statistics to ensure queries use the most efficient access routes. Includes use of data pre-fetch by the database server.
- Stored procedures and functions for optimal performance in network environments.
- Compiled SQL-queries and stored routines are cached and shared, minimizing the number of compilations performed, thereby improving performance where there is heavy use of dynamic SQL (e.g. JDBC and ODBC).
- Caching techniques are specifically adapted for SMP environments.
- Use of highly efficient sort algorithms.
- Transaction management using Optimistic Concurrency Control, a method pioneered by Upright Database Technology, which overcomes many of the problems of conventional locking techniques such as deadlocks, and locks being retained by defunct connections, whilst offering superior performance.

- Group commit transactions make optimal use of the available I/O-capacity.
- Ability to perform Read-Only transactions for optimal performance when executing transactions that do not update the database; allowing mixed workload environments such as data warehouse and OLTP applications.
- Highly optimized B\*-tree structure used for all tables, giving rapid access for keyed and sequential retrievals, and exceptionally high space utilization with no fragmentation of free space.
- Automatic continual re-balancing guarantees perfectly balanced B\*-tree structures without incurring delays for the users, and no possibility of corrupt tree structures even following a hardware failure.
- Fast restart, including integrity checks, following a system failure.
- JDBC and ODBC implemented as native drivers integrated into the product, rather than as add-on layers.
- Optimized for open interfaces (e.g. bulk fetch in ODBC).

### **Security**

- The use of triggers, constraints, functions and stored procedures allows rules to be encapsulated in the database and enforces the rules for all clients connecting to the database.
- Full set of integrity restraints to ensure logical database consistency.
- Advanced security facilities offering fine-grain access control. A role concept allows 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.
- 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 Operations**

- Maintenance-free operations mean no downtime.
- Database structures always optimally organized and therefore never any requirement to carry out database reorganizations.
- Continuous operation by allowing online backups, without any interruptions to the users.
- Shadowing facility that allows transparent recovery following disk hardware failure.
- Extremely stable behavior in run-time environments, which removes the need to stop the database for manual intervention.

### **Openness**

- Maximum conformance to SQL standards. First DBMS in the world with support for SQL-92 transitional level.
- Compatible with all major web and client/server application development tools through the JDBC and ODBC interfaces.
- Compliant with many middleware products for transaction processing, like Tuxedo, Microsoft Transaction Server and Inprise's Application Server.
- Available on a wide range of platforms including UNIX (including Linux), Windows and OpenVMS.

## EASE-OF-USE

### *Database Administration*

Mimer SQL eliminates many of the routine tasks associated with other database engines. Database administration with Mimer SQL is characterized by being greatly simplified and by wherever possible using Operating System facilities. Indeed in most cases normal operational control takes care of the database automatically thereby minimizing the amount of specialist knowledge required.

The number of tuning parameters in Mimer SQL is, intentionally, very limited. The two most important tuning parameters in Mimer SQL 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.

The simplicity by which the Mimer SQL system can be tuned removes the need for highly skilled RDBMS experts to supervise the operations, significantly reducing the maintenance costs for Mimer SQL-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. A Mimer SQL table is stored in a highly optimized balanced tree structure called a B\*-tree. These trees expand and contract dynamically as required, and the algorithms used during this process ensure that data is always stored in an optimal manner.

The use of B\*-trees means that there is never any need to reorganize a Mimer SQL database, or any benefit to be gained by so doing, since the B\*-tree structures are always kept perfectly 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 SQL 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 SQL's use of a careful replacement protocol during the dynamic table reorganizations 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.

In Mimer SQL, tables are created in databanks. A databank is a standard direct access file and contains an arbitrary number of tables. The use of bitmaps to control free space means that there is minimal initialization required when a databank is expanded. Mimer SQL 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 SQL 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. In addition, the databank files may be backed up using standard OS utilities.

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 SQL.

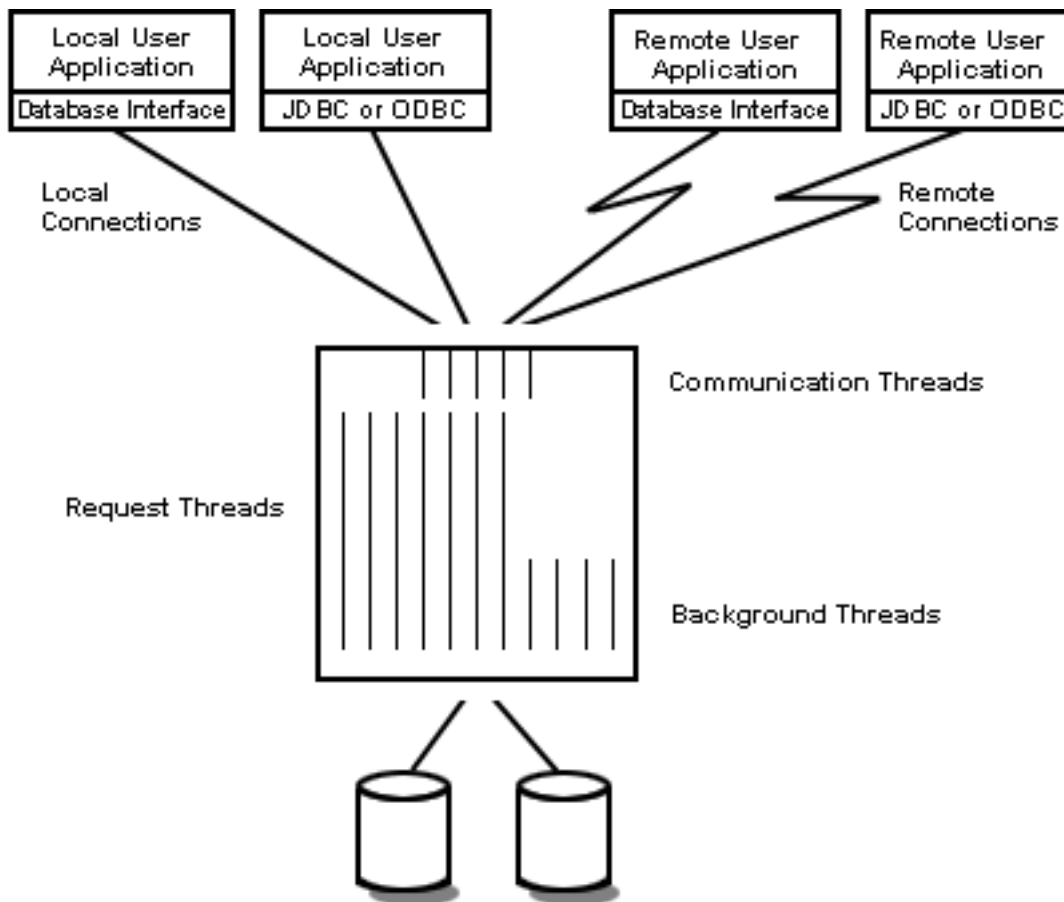
The Concurrency Control techniques utilized by Mimer SQL for transaction handling also eliminate the requirement for any DBA intervention to resolve deadlocks or other lock related problems such as those caused when a client fails.

## PERFORMANCE

### *Database Server Architecture*

The Mimer SQL DBMS is based on client/server architecture. The Database Server executes in one single, multi-threaded process with multiple **Request and Background threads**. On some platforms **Communication threads** are used. The Mimer SQL architecture is truly multi-threaded, with requests being dynamically allocated to the different Request threads. As threads scale very well over multiple CPUs, Mimer SQL 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 that control the database server. It also ensures that the application can only view data that has been formerly passed to the client side, which is extremely important from a data security point of view.



Mimer SQL Database Server Architecture

The **Communication threads** are used to handle parts of the communication between the applications and the database server. On some platforms other mechanisms are used to handle the communication between the applications and the database server. Whatever the mechanism, all communication with the database server is multi-threaded, allowing large numbers of simultaneous user requests.

Both local and remote applications are handled directly by the Database Server. This means that in **Client/Server** environments, where Mimer SQL 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, either on the client or on the 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 routine is executed by a Request thread, the compiled version of the query or the routine is stored within the Database Server. In this way the same, compiled version of the query or routine can be used again by other applications. This leads to improved performance, since a SQL query or a stored routine only need to be compiled once by the Database Server.

The **Background threads** perform database services including all database updates, online backup 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 request and background 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 SQL 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.

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

A very limited number of tuning parameters avoid the need for costly specialist knowledge.

## **The Database Cache**

The **Database Cache** of Mimer SQL (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 utilize available main memory.

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 simultaneously 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 SQL, 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 SQL Server performs the following at a cleanup:

- First it cancels any database requests from the application that are currently being executed.
- Then it closes all tables that the application held open.
- Finally all uncommitted transactions for that application are rolled back.

If a remote client is switched off without disconnecting from the Database Server, Mimer SQL makes use of the KEEPALIVE functionality in TCP/IP 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 SQL will be notified to perform the cleanup. Note that since Mimer SQL uses optimistic concurrency control, no locks are held during this time.

In competing products the problems caused by database locks are considerable. Connections between a client and server are lost for a large number of reasons, not just because the client machine is turned off. In a locking system, other users requiring the lock held by the abnormally terminating client will be blocked until the situation is resolved, in some cases this can require manual intervention by a DBA. Mimer SQL does not suffer from the problems caused by locks.

## **Stored Routines**

The concept of stored procedures and stored functions (collectively referred to as stored routines) is very useful in an RDBMS. SQL routines are stored in the database just like other schema objects (tables, domains, etc). Stored routines allow application logic to be moved from the applications to the RDBMS. Routines are cached on the server and avoid the overheads of transmitting and preparing frequently used SQL code. These features imply 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 server.
- Business rules can be stored in one place and do not have to be duplicated in all the applications.
- Performance is improved, since less communication between a client (e.g. an application

program) and the server is needed to perform the operations in the stored routine. This is particularly important in client/server environments, including the Internet and the World Wide Web.

- Giving the users execution rights to a set of stored routines instead of giving them access rights to the database objects directly eliminates the risk of accidental damage to data through interactive tools.
- Creating a set of stored procedures by which all database access is performed can standardize DBMS access.

Support for stored routines within Mimer SQL is based on the ISO standard for SQL's Persistent Stored Modules (SQL/PSM - ISO/IEC 9075-4:1996(E), December 15, 1996).

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 SQL/PSM. A special type of procedure, 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 the FETCH statement invokes it, 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.

An overview of the features supported include:

- Stored routines written entirely in SQL.
- Data manipulation statements.
- Procedural programming statements - RETURN, CASE, IF/THEN/ELSE, LOOP, LEAVE, WHILE, REPEAT, and compound statements.
- SQL variables and assignment statements for assigning values to SQL variables and to parameters of stored procedures.
- EXECUTE privilege determines which users can invoke given routines.
- CALL statement for invoking stored procedures.
- Functions may be invoked from within scalar expressions e.g. in SELECT and WHERE clauses in queries.
- Explicit parameter modes (IN, OUT, or IN OUT) for stored procedure parameters. An SQL function can only have input parameters but does return a value.
- Advanced exception-handling facilities.

Mimer SQL supports a pseudo data type called ROW. The ROW data type can be used in a compound statement and is defined by explicitly specifying a number of field-name/data-type pairs or by specifying a number of table columns from which the unqualified names and data types are inherited.

## **Triggers**

In addition to stored routines, Mimer SQL supports triggers. Triggers can be considered to be a special form of a stored procedure; they are not called directly by a user, they execute automatically when a data modification statement is used against a table. Triggers can be defined to execute either before or after rows are inserted into a table; when rows are deleted from a table; and when columns are updated in the rows of a table. Triggers incorporate virtual tables that reflect the row image before and after the operation, as appropriate.

Tables can have multiple triggers and Mimer SQL supports multiple triggers for each modification event on a table. Triggers can be called recursively, both indirect and direct.

Triggers are an extension to the concepts behind constraints, except that they can provide a greater flexibility because the trigger body allows a greater level of control. Their primary use is to enforce complex business rules or requirements. Triggers can be used to extend the integrity checking logic of constraints, defaults and rules; constraints and defaults should be used instead if they can encapsulate all the needed functionality. Unlike check constraints, triggers can reference columns in other tables. A trigger can reject the attempted modification that caused the trigger to execute.

Mimer SQL also supports an INSTEAD OF qualifier to the INSERT, DELETE and UPDATE triggers, which provides a mechanism to make any view updateable. In this case, it is expected that the trigger will modify one or more tables to simulate the data modification statement on the view.

## **SQL Optimizer**

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

## **Secondary Indices**

Within Mimer SQL 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. Concatenating the columns defined as the secondary index with the primary key of the base table forms the primary key of the index table. The secondary index tables are fully maintained internally within Mimer SQL. The algorithm used for this guarantees that the indices can never be out of step with their base table.

The SQL optimizer will automatically use secondary indices whenever there is a performance benefit. If the secondary index contains all the information required by the query the optimizer will not perform a lookup in the base table.

## **Transaction Management**

Mimer SQL uses a method for transaction management called Optimistic Concurrency Control (OCC). Mimer SQL'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. The methods have 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.

- **Build-up** commences the start of the transaction. When a transaction is started a consistent view of the database is frozen based on the state after the last committed transaction. 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 transactions 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 **Commit** 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 that 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 this check phase can cause a conflict. If a transaction conflict is detected, the checked transaction is aborted. 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 the operations in the **Write Set** for the transaction are moved to another structure, called the **Commit Set** that is to be secured on disk. All operations for one transaction are stored on the same page in the Commit Set (if the transaction is not very large). Before the operations in the Commit Set are secured on permanent storage, the system checks if there are any other committed transactions that 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 behavior 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. The Background threads in the Database Server carry out this phase. Even though the changes are applied in the background, the transaction changes are visible to all applications through the Transaction Cache. Once this phase is finished the transaction is fully complete. If there is any kind of hardware failure that means that Mimer SQL is unable to complete this phase, it is automatically restarted as soon as the cause of the failure is corrected.

Most other DBMSs offer *pessimistic concurrency control*. This type of concurrency control protects a user’s reads and updates by acquiring locks on rows (or possibly database pages, depending on the implementation), this leads to applications becoming ‘contention bound’ with performance limited by other transactions. These locks may force other users to wait if they try to access the locked items. The user that ‘owns’ the locks will usually complete their work, committing the transaction and thereby freeing the locks so that the waiting users can compete to attempt to acquire the locks. Optimistic Concurrency Control (OCC) offers a number of

distinct advantages including:

- Complicated locking overhead is completely eliminated. Scalability is affected in locking systems as many simultaneous users cause locking graph traversal costs to escalate.
- 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.
- Removes the problems associated with the use of ad-hoc tools.

Through the Group Commit concept, which is applied in Mimer SQL, the number of I/Os needed to secure committed transactions to the disk is 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-initialization of structures used by the transaction control system.

Another significant transaction feature in Mimer SQL 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. For example, a complicated long-running query can execute in parallel with OLTP transactions.

## **Storage Structures**

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

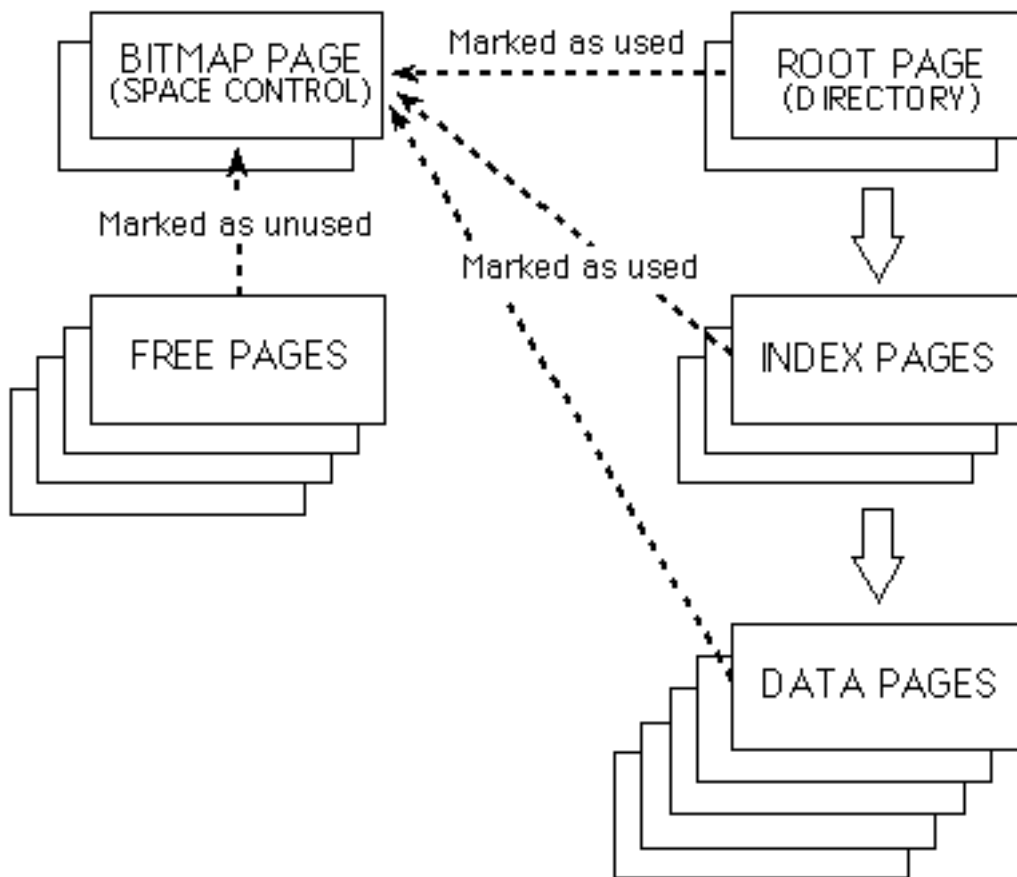
The use of standard OS files allows the databanks to be expanded dynamically when required; it also allows the use of Operating System facilities such as disk striping and RAID systems.

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 SQL 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 utilization. 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 SQL 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, as the databank grows, 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 SQL table is stored in a highly optimized 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 SQL DBMS has been highly optimized 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 SQL 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.

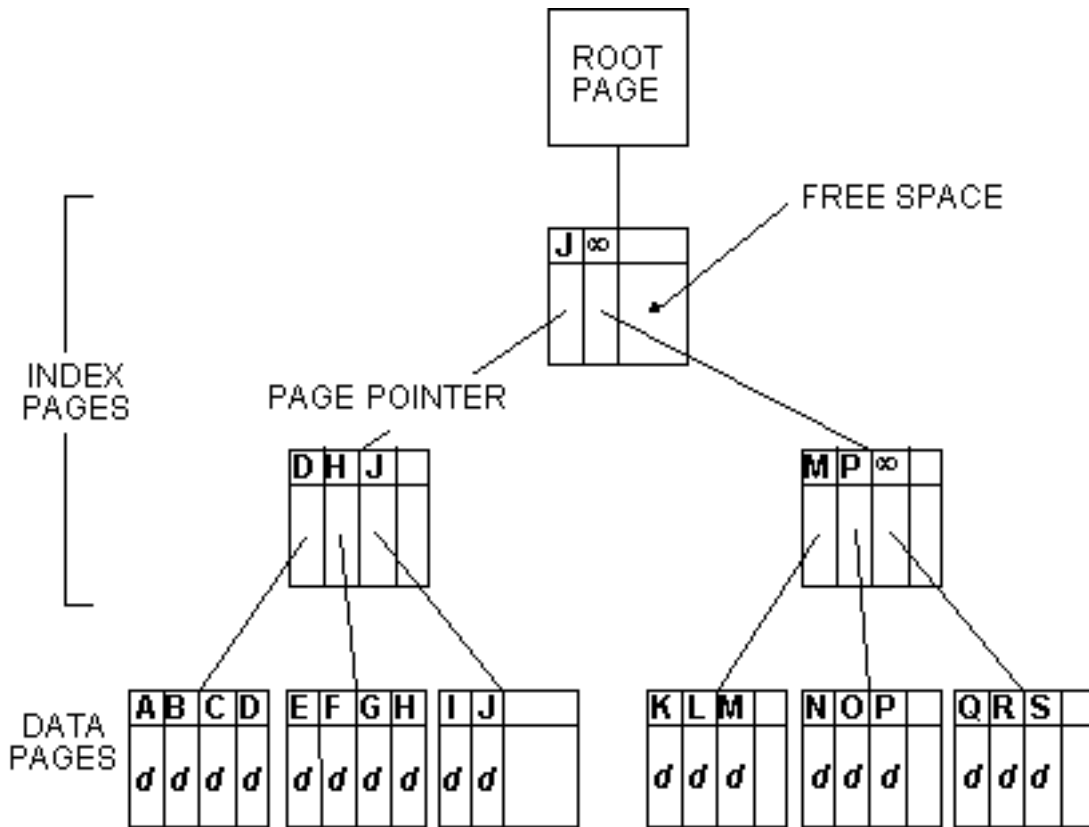


Table Structure (B\*-tree)

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 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 reorganization.

The algorithm used for reorganization 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 reorganization. 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 that required splitting is marked as free. Even if there is a machine failure during the execution of the reorganization, the careful replacement protocol ensures that the disk version of the tree will never be inconsistent.

Mimer SQL performs reorganizations automatically. This totally eliminates the need to perform manual reorganizations whilst still ensuring that the tree is always kept perfectly balanced giving continuous optimal performance. The reorganizations are always small involving only the branch of the tree that is being traversed so there is no noticeable effect on response time for the user.

To further improve space utilization, Mimer SQL 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.

As the tree is always perfectly balanced, the number of index levels to traverse is always the same for all the rows in a table. There are never any overflow chains, which are a common problem in competing products and can lead to a large number of I/Os to access a specific row.

## SECURITY

### *Integrity*

The Mimer SQL Server allows you to enforce constraints in the database, either for database integrity purposes or business-related rules. Through the use of declarative integrity constraints, database procedures, and database triggers, Mimer SQL provides a high degree of security and business rule enforcement.

Constraints define rules that enforce data integrity. Constraints are relatively simple to maintain but aren't suitable in situations where you need to enforce complex logic. With stored procedures, functions and database triggers you can enforce complex business rules at the server level, improving application performance, scalability and security, and reduce development costs. Database triggers are executed automatically when data manipulation statements are actioned, and can be used to enforce complex integrity rules in the server.

Data integrity is vital in a database; if the quality of the data is questionable, any information derived from that data must be suspect. The relational model defines four types of integrity that can be used to ensure that your data is consistent and correct:

- Domain integrity
- Entity integrity
- Referential integrity
- User integrity

**Domain integrity** is to do with the values that may be contained within a specific column. All columns have an implicit domain derived from their data type but Mimer SQL also supports the CREATE DOMAIN statement. A domain can be defined with a number of CHECK clauses and a DEFAULT value.

A domain definition can be used instead of a data type in a column definition. This has the advantage that the same definition of data type, check clauses and default value can be used in many column definitions and therefore those columns are guaranteed to have the same attributes.

If a DEFAULT value is not specified for a column definition (either explicitly or implicitly through the use of a DOMAIN) then NULL is used.

A column definition can specify NOT NULL. This indicates that the table column must contain a value for each row. By implication a column definition that does not specify NOT NULL do not have to contain an actual value. NULL is a condition (rather than a value) that represents at least one of three states of the data values: not applicable, applicable but not available, or applicability unknown. While this is not formally a domain constraint it is an important concept that is referred to in the other constraints.

**Entity integrity** ensures that each row in a table is uniquely identified. The concept is basic to database design. The primary key of a table is one or more columns that are used to uniquely identify each row of the table.

The primary key enforces entity integrity. A table can only have one primary key. The primary key constraint does not allow duplicate values and does not allow NULL.

**Referential integrity** is concerned with the maintenance of relationships between tables. A referential constraint defines a foreign key relationship between the referencing table and another table in the database (the ‘referenced table’). The foreign key is defined in the referencing table as a relationship either the primary key or one of the unique keys in the referenced table.

For example, you cannot insert into the referencing table a foreign key value that does not exist in the referenced table. This rule of referential integrity ensures that a non-NULL value of a foreign key must be within the *domain* of the related primary or unique key.

Rules can be defined in references that specify the action to be taken on the affected rows of the referencing table when a DELETE operation is performed on the referenced table. These rules can define one of the following triggered actions: CASCADE, the affected rows are also deleted; SET NULL, the appropriate foreign key columns are set to NULL; SET DEFAULT, the appropriate foreign key columns are set to their default value; and NO ACTION, which raises an error because the referential constraint would be violated.

**User integrity** covers all other forms of integrity constraints that are not covered by the other three. In Mimer SQL **table integrity** refers to the facility of defining CHECK clauses in table definitions whereby the contents of a column is verified against a list of acceptable values. Alternatively, it can check the relationship between values in more than one column in a row. In addition, UNIQUE constraints (alternative keys) can be defined on a table. Like the primary key, a unique key is composed of one or more table columns and a key value uniquely identifies a row in the table (i.e. there are no duplicates). **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 view-defining conditions.

User integrity also covers user-defined business rules, regulations, policies and procedures. Stored routines and triggers are usually used to enforce business integrity.

**Stored routines** (procedures and functions) 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 routine (with or without GRANT option) that accesses those objects. The user creating a stored routine must, of course, have sufficient access rights to all objects involved in the routine. Using stored routines 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 routines.

A **trigger** is a special type of stored procedure that is called automatically whenever a specific operation is performed on a table. Triggers are most useful when the features supported by constraints cannot meet the functional needs of the application.

The **View** mechanism is a very powerful concept. It is the result set from a predefined SELECT statement, a so-called derived table. 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 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 normalized database.

Views with an implicit one-to-one relationship with an underlying table are directly updateable, provided the appropriate access privileges are held. An INSTEAD OF trigger provides full view updating. The trigger body contains SQL procedural code, allowing the developer to define the operations to be performed on the tables that under-pin the view.

## Access Security

Through the advanced security facilities of Mimer SQL, the database can be protected from any unauthorized access. Database privileges authorize users to perform certain SQL operations, such as insert, update, or delete, on selected database objects. The extremely flexible security system provided by Mimer SQL enables data to be protected down to a single element (row/column); allowing you to precisely enforce database security policies, ensuring users have only the privileges they need.

A unique feature of Mimer SQL is the “role concept”, where the access rights for a user can be increased under password protection. The role concept allows Mimer SQL’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. Mimer SQL provides the role concept through the PROGRAM ident.

By utilizing Mimer SQL’s advanced facilities for access control and security much coding in applications is avoided and all applications utilize a consistent set of controls.

Within Mimer SQL an **Ident** is an authorized 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 - authorized to log on to Mimer SQL. User idents are generally associated with specific individuals authorized 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 SQL as an OS\_USER, then that user may log into Mimer SQL without providing an additional user name and password. This is sometimes referred to as **integrated login**.
- PROGRAM idents - may not log directly onto Mimer SQL, but instead an ident who has already logged on may adopt the role of a Program ident by using the ENTER statement. Typically, a user is given EXECUTE privilege on a Program Ident and the ENTER statement is performed by the application code. 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. The use of Program Idents can significantly reduce the burden of security management.
- 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 organizing the privilege structure in the database system.

When an Ident connects to Mimer SQL 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 unauthorized 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:

- BACKUP - gives the right to perform databank backup and restore operations
- DATABANK - gives the right to create databanks
- IDENT - gives the right to create idents
- SCHEMA - give the right to create schema
- SHADOW - gives the right to create and manage databank shadows
- STATISTICS - gives the right to execute the UPDATE STATISTICS statement

**Object privileges** give rights over certain specified objects in the system. Mimer SQL 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 routine, or to enter a given program ident
- MEMBER - grants membership in a specified group ident
- USAGE - gives the right to use a given domain or a specified sequence

Object privileges are initially granted only to the creator of the object. Their grantor may revoke privileges.

**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 or unique 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 idents with or without grant option

## ***Backup and Recovery***

Mimer SQL'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 SQL handles database consistency is handled on two levels: physical and logical.

**Physical consistency** means that the tables are readable by the Mimer SQL database manager. The Mimer SQL 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 SQL'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 that 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 SQL 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 SQL'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.

The Mimer SQL system allows an online backup (i.e. a 'full backup') of a databank to be taken, whilst maintaining full access to all the data. The backup is a complete copy of the databank file and may be used as the basis for a databank recovery operation.

As a complement to backup copies it is possible to archive the transaction log changes, forming the equivalent of an incremental backup of all logged databanks. This allows several backup copies of a databank of varying ages to be kept, along with a sequence of incremental LOGDB backup files. By using the appropriate sequence of LOGDB backups and the current LOGDB, it is possible to roll forward any of the backup copies of the databank to an up to date state.

As Mimer SQL databanks are ordinary Operating System files, the normal OS 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 SQL RDBMS is characterized by extremely high availability, with no requirement for manual database reorganizations or other maintenance operations that would cause downtime for the database applications.

Mimer SQL therefore ensures that the database remains operational 24 hours a day, 7 days a week without any disturbances to the production environment. Mimer SQL provides an online backup facility so that the database can be backed up while it is running and without interrupting transaction processing, even during heavy OLTP usage. In the event of a hardware failure (e.g. disk crash), a shadowed database will continue to be available, without any interruption to the application. Mimer SQL 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. Applying the alteration separately to each shadow so that any physical corruption of the master are not transferred to the shadow.

In Mimer SQL, shadowing is controlled at the databank 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.

During normal running the shadowing has no impact on response times. The Background threads perform changes made to the master databank and shadows, without impacting the users.

### **Product Quality**

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

The quality of the Mimer SQL product is assured by the use of a documented and established development process that has been used for many years at Upright Database Technology. Each Mimer SQL version is subject to rigorous Quality Assurance checks before it is formally released.

## OPENNESS

Upright Database Technology is committed to conforming to the relational database standards defined by organizations such as ANSI, ISO and The Open Group. Upright Database Technology's policy means that Mimer SQL has a unique openness towards independent development and middleware tools. In turn this ensures application portability and interoperability, protecting your development investment.

Mimer SQL provides a high-performance native ODBC 3.0 driver and a 100% Pure Java JDBC Level 1, Type 4 driver. Through these interfaces a large number of web-based and Windows-based development tools can be used together with Mimer SQL (e.g. Tango, Delphi, Visual Basic, Access, Centura and many more). Mimer SQL is also compliant with many CORBA- and Object Transaction Monitor-based middleware products (e.g. BEA's WebLogic and Inprise's Application Server).

The JDBC and ODBC interfaces are tightly integrated into the database engine, providing excellent performance.

By using Mimer SQL's standardized Embedded SQL interfaces for C, COBOL and FORTRAN, it is possible to develop portable 3GL applications. Also supported are the SQL standards for constructing SQL statements during program execution (Dynamic SQL), allowing ad-hoc querying of the database from applications.

The programming interfaces to Mimer SQL are multithreaded. A multithreaded application is an alternative to making asynchronous calls to perform multiple calls in parallel. A thread can make a synchronous call, and other threads can be processed while the first thread is blocked waiting for the response to its call. This model is more efficient than making asynchronous calls because it eliminates overheads such as network traffic and repeated calls testing whether a function is still executing.

## SQL

Upright Database Technology's policy is to develop Mimer SQL, as far as possible, in accordance with the established standards. The following SQL standards are accepted as current:

- X/Open SQL 1995 (referred to here as X/Open-95)
- ISO/IEC 9075:1992 (referred to here as SQL-92)
- SQL/PSM (the section of the SQL-92 standard covering Persistent Stored Modules)
- ISO/IEC 9075:1999 (referred to here as SQL-99)

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 SQL combines many of the advantages of the different standards.

Mimer SQL 8 conforms to the following SQL standards:

- X/Open-95
- SQL-92 (entry and transitional levels)
- SQL/PSM

Triggers and Persistent Stored Modules are implemented according to the SQL-99 specification.

The primary goal of the X/Open-95 standard 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	Mimer SQL	X/Open-95
Character string length	15000	254
Variable length character string	15000	254
BINARY string length	15000	254
Variable BINARY string length	15000	254
DECIMAL precision in digits	45	15
FLOAT precision	45 digits	47 bits
SMALLINT precision in digits	5	5
INTEGER precision in digits	10	10
BIGINT precision in digits	19	
REAL precision	7 digits	21 bits
DOUBLE PRECISION precision	15 digits	47 bits
SQLCODE precision in digits	10	9
Fractional precision, in decimal digits, of TIME seconds component	9	6
Fractional precision, in decimal digits, of TIMESTAMP seconds component	9	6
Fractional precision, in decimal digits, of INTERVAL seconds component	9	6
Leading precision, in decimal digits, for INTERVAL data types	7-12	7
Fractional precision for INTERVAL data types that have a SECOND field	9	6
Number of columns in a table	252	100
Number of columns in an index	32	6
Number of tables referenced in a statement	Unlimited *)	10
Number of cursors open simultaneously	Unlimited **)	10
Number of columns a single update statement can update	Unlimited *)	20
Number of nested sub queries	15	9
SQL statement length in bytes	32767	4000
Total length of a row, the sum of · Two bytes for column identification · Lengths of all character fields · Precision of all numeric fields	16000	2000
Identifier length	128	18

\*) There is a limit on the total complexity of an SQL statement. Mimer SQL can easily meet the specified X/Open-95 limits.

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

## **JDBC**

JDBC is the de-facto standard for accessing relational database systems from the Java programming language.

The Mimer SQL JDBC Driver is a Type 4 - Native Protocol All-Java Driver. The Type 4 architecture uses a message protocol that is specific to Mimer SQL; as this means that there is no need for any intervening processes or translation, this architecture is extremely efficient.

A Type 4 driver is written completely in Java so that it can run on any client that supports the Java Virtual Machine (JVM). This makes the Mimer SQL JDBC Driver ideally suitable for both Internet and intranet applications and for use in application servers. An applet using the JDBC driver will run on any client browser.

The driver is extremely small (approximately 100KB in JAR format) in size, so that it can be simply incorporated into an applet that can be downloaded over the Internet. No other Mimer software is required to be installed on the Java client, eliminating any need for configuration management on the client side.

## **ODBC**

Mimer SQL ODBC is a very efficient implementation of Microsoft's Open Database Connectivity interface. The Mimer SQL ODBC driver complies with the Microsoft ODBC 3.0 specification. The driver supports applications written for the ODBC 2.5 or earlier versions of the ODBC function in the manner defined in the ODBC 3.0 specification.

Mimer SQL ODBC enables GUI-based applications and tools to be connected to the Mimer SQL Database Server. In the Mimer SQL architecture the ODBC interface is placed at the same level as Mimer SQL'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 that are not included in the traditional SQL interfaces are supported and fully implemented by Mimer SQL ODBC. These additional features are particularly advantageous in a network environment and when using a GUI such as Windows.

## **Web-based Database Applications**

Mimer SQL's stability in run-time environments and the built-in support for JDBC and ODBC makes it ideal for building dynamic web-based database applications. Data can be extracted from a Mimer SQL 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 SQL 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 SQL database is to use some web application server middleware. There are a variety of middleware products available which link one of the server interfaces and JDBC or ODBC to provide document to database connections. Mimer SQL has been utilized with a variety of these including:

- JBuilder
- Visual Café
- Star Office
- Tango Application Server
- Microsoft Internet Database Connector
- Netscape Application Server
- Cold Fusion
- PHP
- Perl

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 that 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 SQL database using Embedded SQL, JDBC or ODBC.

## **ActiveX**

ActiveX Data Objects (ADO) is a language-neutral object that is the basis for Microsoft's Universal Data Access strategy.

The Open Database Connectivity (ODBC) specification provides data access to, primarily, SQL-based databases. OLE DB is Microsoft's next generation following on from ODBC. Microsoft's plans for OLE DB go way beyond just providing access to relational databases. OLE DB providers will include spreadsheets, documents, graphics, e-mail, ...

ADO is a high-level interface to OLE DB. OLE DB includes a data provider that allows access to ODBC data sources; so ADO can use OLE DB's ODBC data provider to access a Mimer SQL data source in a very simple and straightforward manner.

Online applications can be built by using ADO on the server to deliver dynamic content through the World Wide Web. Through this mechanism, any client platform supporting a modern Web browser can access these applications, allowing users of Windows PCs, Macs, UNIX workstations and web-devices to simply connect to the application.

## **Client/Server - Heterogeneous environments**

Access to remote databases in client/server environments is totally transparent within Mimer SQL. 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 SQL 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 SQL system.

Mimer SQL supports heterogeneous environments, where UNIX, Open/VMS, Windows NT/2000, Windows 98/Me computers can be freely mixed on a network. Mimer SQL's use of a standardized 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 SQL uses TCP/IP or Named Pipes protocols. Mimer SQL Clients utilize 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 SQL are:

<b>Data type</b>	<b>Description</b>	<b>Range</b>
CHARACTER(n)	Character string, fixed length n.	$1 < n < 15000$
CHARACTER VARYING(n) or VARCHAR(n)	Variable length character string, maximum length n.	$1 < n < 15000$
BINARY(n)	Fixed length binary string, maximum length n.	$1 < n < 15000$
BINARY VARYING(n) or VARBINARY(n)	Variable length binary string, maximum length n.	$1 < n < 15000$
INTEGER(p)	Integer numerical, precision p.	$1 < p < 45$
SMALLINT	Integer numerical, precision 5.	-32768 through 32767
INTEGER	Integer numerical, precision 10.	-2,147,483,648 through 2,147,483,647
BIGINT	Integer numerical, precision 19.	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807
DECIMAL(p,s)	Exact numerical, precision p, scale s.	$1 < p < 45$ $0 < s < p$
NUMERIC(p,s)	Exact numerical, precision p, scale s.	$1 < p < 45$ $0 < s < p$
FLOAT(p)	Approximate numerical, mantissa precision p.	$1 < p < 45$ 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 $10^{+38}$
DATE TIME TIMESTAMP	Composed of a number of integer fields, represents an absolute point in time, depending on sub-type.	Described below.
INTERVAL	Composed of a number of integer fields, represents a period of time, depending	Described below.

In SQL, a temporal value is either a datetime (i.e. a date, a clock time, or a timestamp) or an interval (i.e. a period of time). They consist of a contiguous subset of one or more of the fields: YEAR, MONTH, DAY, HOUR, MINUTE and SECOND. Temporal values follow the usual rules of the Gregorian calendar and the 24-hour clock.

Intervals are either year-month intervals (periods of time involving years and/or months) or day-time intervals (periods of time involving days and/or hours and/or minutes and/or seconds and/or fractions of a second).

Mimer SQL supports the OVERLAPS predicate that compares either a pair of datetimes, or a datetime and an interval, to determine whether the two chronological periods overlap in time.

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

A SEQUENCE is a construct that returns integer values regardless of concurrent access to the database system; this eliminates application contention when obtaining a unique numeric key value, a common requirement in transaction processing applications. It is also possible to retrieve the previous value returned to the application. One use for a SEQUENCE is as the default value for a column or domain.

Columns that 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 SQL supports the CAST function, which explicitly converts between data types.

## **Platforms**

Mimer SQL is available for a wide range of hardware and Operating Systems. Hardware platform and OS decisions should be based on the requirements of the application; Mimer SQL allows the developer to create the solution that the business requires.

- Intel Windows NT/2000
- Intel Windows 98/Me
- Intel Linux (Red Hat, SuSE, and many more)
- Intel SCO
- IBM RS6000/AIX
- Bull/AIX
- Hewlett-Packard HP-UX
- SUN Solaris
- Alpha AXP/Tru64 UNIX
- Alpha AXP/Windows NT
- Alpha AXP/OpenVMS
- (Data General DG-UX)

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



**Upright Database Technology AB**  
Box 1713  
Kungsgatan 64  
SE-751 47 UPPSALA  
Sweden  
Tel: +46 18 780 92 00  
Fax: +46 18 780 92 40

**Upright Database Technology Ltd**  
PO Box 5102  
Daventry  
NN11 6ZA  
United Kingdom  
Tel: +44 1327 300050  
Fax: +44 1327 300131

**Formula Open Soft  
Ltd**  
Dordrecht  
The Netherlands  
Tel: +31 78 673 63 41  
Fax: +31 78 673 65 29

**Sysdeco Technology AS**  
Oslo  
Norway  
Tel: +47 22 09 65 00  
Fax: +47 22 09 65 01

**StarVision Information Technology Pte**  
Singapore  
Tel: +65 293 48 58  
Fax: +65 293 76 04

**RelTech Software**  
Merlischachen  
Switzerland  
Tel: +41 41 852 02 95  
Fax: +41 41 852 02 96

**Mimer Hellas Ltd**  
Athens  
Greece  
Tel: +30 1 201 09 09  
Fax: +30 1 201 09 11

**Mimer USA**  
Long Lake MN  
USA  
Tel: +1 952 473-8562  
Fax: +1 952 473-2788

**O.S.I. de Guatemala**  
Guatemala City  
Guatemala  
Tel: +502 332 69 73  
Fax: +502 333 04 59