Mimer SQL
11.0
Technical Description

INTRODUCTION

Mimer SQL is a Relational Database Management System (RDBMS) developed by Mimer Information 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 embedded systems, to workstations, to enterprise 3-tier architecture systems and cloud solutions. It offers a unique level of scalability, including multi-processor support, and with its availability on Windows, OpenVMS, macOS, and Linux, it is perfectly suited for open environments where interoperability is important.

Mimer SQL has support for mobile systems such as Android. In Android, Mimer SQL runs seamlessly without applications being aware that the underlying database has been switched. Everything works faster and with a new level of security. It is also possible to integrate information from different parts of the phone in a secure and easy manner. Other popular enhancements include Pinyin support and sorting according to any language in the world.

The Mimer SQL product also runs in memory constrained environments such as embedded systems, IOT devices, etc. Because of the small footprint and unique architecture of the Mimer SQL product, unparalleled functionality is supported to applications in these environments.

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-2016 standard), an ADO.NET provider written completely in managed code, 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 also offers a native Mimer SQL client for those wishing to benefit fully from Mimer SQL's performance and stability. Mimer SQL is well suited for mission-critical multi-tier solutions, since its conformance to standards makes it compliant with many transaction processing middleware products. Mimer SQL support application servers based on both Java and .NET. The support includes distributed transactions that allow changes to span several Mimer SQL or other transaction-based systems. In the Microsoft environment the Microsoft distributed transaction coordinator (MSDTC) is used.

Mimer SQL also features unique support by allowing real-time access to relational data. The real-time access features guaranteed response times while allowing the real-time data to be accessed through an ordinary Mimer SQL Database 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 Mimer Information Technology.

All the products in the Mimer SQL family use the same code base. 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 mobile phone running Android, to a small laptop running Windows, and on to a large Linux or OpenVMS 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, or to an embedded device, without having to make any modification to the application code or database.

The following eight key features characterize Mimer SQL:

- Ease-of-use
- Performance
- Functionality
- Security
- 24 x 7 Operation
- Openness
- Small footprint
- Device awareness

... which when combined with Mimer Information Technology's aggressive pricing provides the customer with a significantly reduced total cost of ownership (TCO). By conforming to the international database standards Mimer Information 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 eight key areas:

Ease-of-use

- Automatic self-tuning, means a limited number of tuning parameters, and so removes the need for highly skilled staff or expert consultants.
- 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. Mimer SQL allows complete control over file sizes. Both a minimum and/or a maximum size may be specified. In addition, a so-called goal size may be specified. The system will automatically shrink the file to the goal size, or as close as possible, when the internal usage of the space in the databank allows.
- A high degree of isolation between applications. A completely stable and consistent view of the database is available to the application when executing transactions. Application code does not have to cater for situations where inconsistent data is returned by the database server.
- 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 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 and configured in a few minutes.
- Large objects are stored together with other data. No separate storage structures are needed.
- There are no restrictions on number of large object columns in a table.
- Advanced support for automatically upgrading the schema of a database between application versions. In a majority of the cases no separate upgrade program needs to be written.

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 of dictionary information for fast compilations and execution.
- Optimized protocols for large object handling. Allows large objects to be handled with a minimum of overhead.
- Caching techniques are specifically adapted for SMP environments. Provider a high degree of scalability even with many CPU cores.
- Use of highly efficient sort algorithms.
- Transaction management using Optimistic Concurrency Control, a method pioneered by Mimer Information 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 and scalability.
- Group commit transactions make optimal use of the available I/O-capacity by clustering several commit requests on a single I/O.
- Self-tuning transaction storage that adapts to the current transaction load and automatically grows and shrinks to achieve optimal performance.
- 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.
- Immediate restart allows application programs to access all data immediately after a system failure. This is regardless of the size of the database. If you do not have immediate restart activated, the system is still quick to restart unless you have very large databases. For these systems immediate restart is recommended.
- ADO.NET, JDBC, Embedded SQL, and ODBC implemented as native drivers integrated into the product, rather than as add-on layers.
- Optimized for open interfaces (e.g. bulk fetch and other optimizations in all interfaces).
- Automatic data compression reduces I/O load and saves space.
- When using read-only databanks the transaction handling takes advantage of the fact that the data is not changed to improve performance.
- Advanced indexing, single and multi-column indexes. Word, pinyin, coordinate, and location indexes. Support for collations with indexes.

Examples of SQL Functionality

- Inner and outer join, including natural, cross, left, right, and full outer join.
- With-clause, including recursive queries with cycle detection.
- Support for Exists, Any, Unique, and similar predicates.

Security

- The use of triggers, constraints, functions and stored procedures allows rules to be encapsulated in the database and enforces the rules for all applications using the database.
- Full set of integrity constraints 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.
- Secure protocols are used for logging in to the database. Passwords are never passed over network connection. Even if actual communication is recorded it cannot be reused at a later point as each login uses a unique communication id used by the authentication mechanisms.
- The communication between client and database server can be encrypted. All data passed between client and server is then protected from eavesdropping. Any tampering with the data is also detected. Each database session uses a different encryption key resulting in new data sequences, even if the same data happens to be passed.
- The blocks in the database use a checksum to ensure the low-level consistency of the data structures in the database.

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. In other products this has to be done manually using utilities such as Vacuum or run in the background causing disruption for data access.
- Shadowing facility that allows automatic failover in case of disk hardware failure.
- Extremely stable behavior in run-time environments, which removes the need to stop the database for manual intervention.
- Online backup, which allows a completely consistent backup to be taken without disturbing ongoing activities. The backup copies have no special

formatting, allowing them to be used by simply copying them into place. Also, very easy to look offline at the backup with query system or use for test.

- Immediate access to data after an uncontrolled system shutdown.

Openness

- Maximum conformance to SQL standards. Mimer SQL conforms to all features of SQL 2016 Core level. In addition, 145 features beyond core SQL are also supported.
- Support for SQL/PSM, standardized stored procedures.
- Compatible with all major web and client/server application development tools through the ADO.NET, JDBC, and ODBC interfaces.
- Support for Python standard database interface PEP 249. For details, see https://pypi.org/project/mimerpy/ and https://www.python.org/dev/peps/pep-0249/.
- Compliant with many middleware products for transaction processing, like Oracle Tuxedo and Microsoft COM+. On Microsoft platforms the MS DTC protocol is used to communicate with the transaction coordinator.
- Available on a wide range of platforms including Linux, Windows, macOS and OpenVMS. Examples of embedded platforms include VxWorks, QNX, and Greenhill Integrity.

Small footprint

- The Mimer SQL product is significantly smaller than competing products. This is due to the stringent design and engineering work done with the product.
- Mimer SQL can be used in memory constrained environments. Both the stack and heap usage are carefully controlled.
- The product automatically scales to the current environment. This means that it can support large, as well as small memories efficiently.
- The system automatically applies compression (more often) in environments with less memory.
- The database files can return space to the operating system for reuse automatically.
- The database cache can automatically grow and shrink within predefined limits determined by the user.[1]

Device awareness

- Databank files can be placed on removable devices, such as a memory card (flash), USB disk, or a CD. The system automatically detects when devices are added or removed.

[1] Not available in all environments.

- When a device is not present the system acts as if the table is empty. From an application programming point of view this is very simple to handle and much error handling is avoided.
- When memory load is low the system can expand the database cache. When memory is scarce, the system release memory that may then be used by other applications.
- In applications where power consumption is important the fast startup of the Mimer SQL server can be used to allow a complete startup/shutdown of the CPU, Operating system, Mimer SQL server, and application for each set of requests.
- The small number of read and write operations of the Mimer SQL server make it ideal for flash memory cards where there is a limit to the number of times a block may be written.
- In-memory databases are supported. For small footprint environments this means the system can run on an embedded controller without disk or flash connected. The database server starts with a predefined schema in this case.

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. This database cache is automatically configured depending on the amount of memory available on the machine when the database definition is set up. The number of request threads can easily be determined after only a few days run-time experience.

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. In fact, many Mimer SQL systems run in environments where there is no supervision at all.

Many 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 mean 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 supports a powerful concept called instead-of triggers. This allows any view of arbitrary complexity to become updatable. Instead of directly updating the tables referenced by the view a trigger is activated which performs the appropriate updates of the underlying tables.

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 indexes 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 Mimer SQL limit for a single file is ~3 exabyte, so it is the file system limits that govern how much can be stored in a file.

The opposite is also true. Mimer SQL can release space to the file system, either if a goal size has been set on a databank or an Alter Databank statement directing the system to release space is executed.

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 to the online backup facilities, 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.

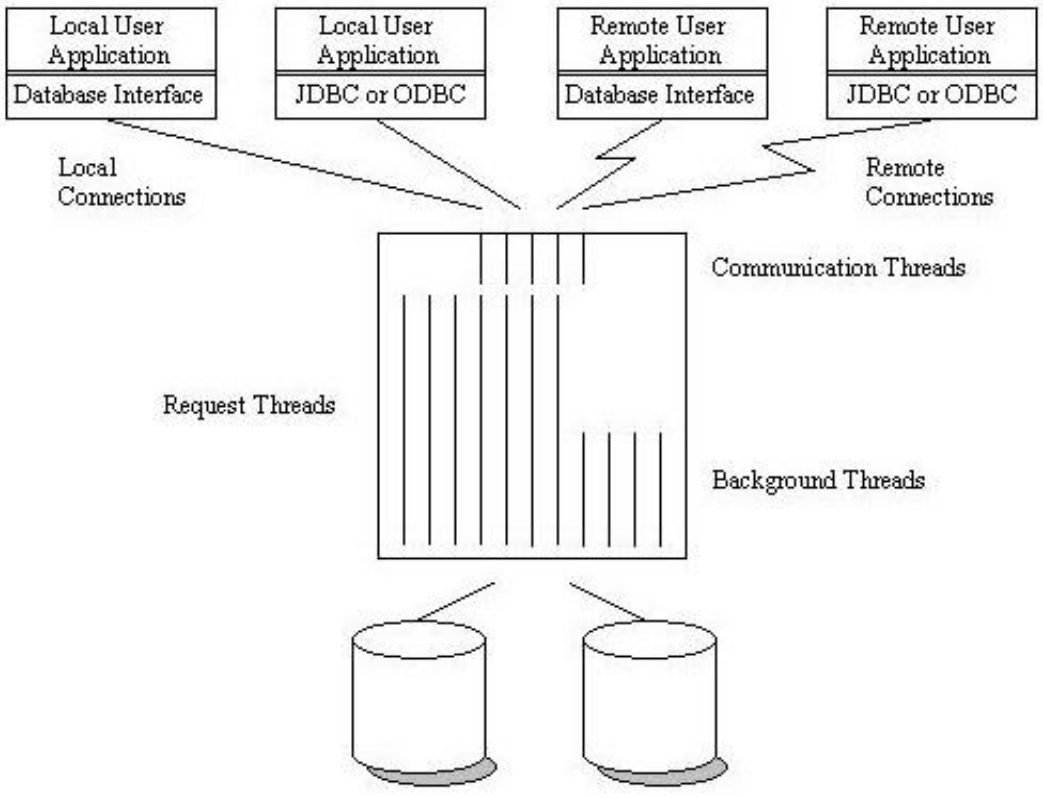
For an **embedded system** or a database on a mobile phone the system must not require any maintenance whatsoever. The fact that the Mimer SQL system has been designed and implemented with this in mind from the ground up makes it the ideal database system for these environments.

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** handle incoming requests to the database server. On some platforms a single communication thread is sufficient to handle all the connections in a scalable manner, while on others more than one may be required. On some platforms there are operating system primitives that allow the request threads to handle incoming requests directly, thereby eliminating the communication threads

layer altogether. 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. When a client connects to the server no additional processes or threads are needed. This allows the system to respond very quickly to new connection requests.

The **Request threads** perform the SQL operations requested by the applications. When the Database Server is requested to perform an 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. Also note that the request threads may serve any incoming request, there is no tie between request thread and a specific application. In case of many long request to the database server, the system automatically reschedules the operations to allow more requests to be served concurrently.

When an 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 an 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, logging, 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.

On most platforms 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. In many circumstances the I/O may actually complete before the result of the I/O is required by the system. Most I/O are both initiated and waited for by the background threads.

The system uses many different techniques to achieve a high throughput rate. For example, if the SQL optimizer determines that a sequence of pages will be accessed to perform the query in question, it will instruct the underlying access manager to prefetch pages. The rate at which this is done depends on how many pages are needed and in what manner they are accessed. For queries that are guaranteed to access many pages a higher prefetch rate is used than queries that only need a few pages at a time.

Server Architecture Benefits

The threads-based implementation of the Mimer SQL Database Server performs particularly well in Symmetrical Multi-Processing (SMP, multi-core CPUs, or hyperthreading) 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 Mimer SQL server accesses the database cache and other shared memory in a manner that is beneficial with regards to memory caches. This means that memory write operations are often local to a single cache line. This allows for a high degree of scalability when the number of processor cores grow.

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.

In embedded environments this results in lower usage of CPU and thus improves battery times.

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 system automatically takes the available memory into account when a database definition is set up.

In many environments the database cache can automatically expand and contract. Instead of specifying a fixed number of pages the user specifies a range where the Mimer SQL server starts with the minimum and allowed to grow to the maximum. The exact mechanism that controls shrinking and growing the cache is machine dependent.

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.

The size of the database cache is limited by the allowed size of the process. On a 32-bit system this may be limited by the size of the paging files or process specific quotas. In addition, only 4 gigabytes (and sometimes 2 GB) may be handled by a 32-bit address.

On a 64-bit hardware the database cache can currently support up to 327 terabytes of data. So, in practice, the size has to do with how much main memory should be used by the server rather than any limit in the system.

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. Execution plans for routines are cached on the server and avoid the overhead 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.
- 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.
- The application can become less dependent on the schema structure. Typically, a stored procedure is used to store information that applies to many tables. A change in the underlying table structure only needs to be reflected in the stored procedure.

Support for stored routines within Mimer SQL is based on the ISO standard for SQL's Persistent Stored Modules. This language is a natural extension of SQL. This means that the data types available are the same as in SQL in general, and that variables have SQL null value handling built in. When comparing variables, collations may, of course, be used. For example, case insensitive comparisons can be made. This allows for seamless integration within the database server for this type of functionality.

An overview of the features supported include:

- Stored routines written entirely in SQL.
- Data manipulation statements.
- Domains and user defined types
- Procedural programming statements - RETURN, CASE, IF/THEN/ELSE, LOOP, LEAVE, WHILE, REPEAT, ITERATE, 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.
- FOR-loop for easy retrieval of data into the routine.
- 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.
- ATOMIC execution units where all statements succeed or fail as one unit.
- Support for recursive invocation of procedures and functions.
- Advanced exception-handling facilities.

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.

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, instead 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. Statement triggers are called with all rows modified by a single SQL modification statement. Row triggers are called once for each row modified. Statement triggers incorporate virtual tables that reflect the row image before and after the operation, as appropriate. Row triggers gets the changes to the current row through its parameters.

Tables can have multiple triggers and Mimer SQL supports multiple triggers for each modification event on a table. Triggers can be called recursively. This happens if a trigger in turn modifies the database, thus invoking the same or other triggers.

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. When a complex view is created the creator receives SELECT access only. When an INSTEAD OF trigger is created on the view the corresponding privilege is granted, with grant option, to the creator of the view (e.g. an INSTEAD OF INSERT trigger gives the creator INSERT privilege 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 indexes 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. The result of the optimization process can be viewed by using the Explain functionality. Explain is present both in DbVisualizer and bsql.

Many **advanced** SQL constructs are supported. For example:

- Scalar subqueries
- Views on views
- Union in views
- With hold
- Correlated subqueries
- and much more…

The use of primary keys is strongly recommended as they are highly efficient and the preferred access path for the optimizer.

The optimizer can handle both user written SQL as well as generated SQL efficiently. Typically, generated SQL contains constructions not present in hand-written SQL. This means that the optimizer rewrites the query into a generic format before doing the actual cost-based optimization. This has the effect that views are handled particularly well, as conditions outside the view are combined with the view definition whenever possible.

Many other operations are done such as:

- Or-optimization. This means that several indexes may be employed to solve the query quickly, eg. C1 = 2 or C2 = 5 may use index on C1 and another index on C2 to compute the result.
- Select item elimination.
- Join transformations such as converting outer join to inner join to allow more access paths to be used.
- Select in from list elimination.
- Constant evaluation of conditions.
- And much more…

Collations

The sorting capabilities of the Mimer SQL system are extremely advanced. So-called linguistic sorting is implemented in a very dynamic and efficient manner. This is both from a performance and footprint point of view.

The character repertoire is currently based on the Unicode 12.1 standard (see https://www.unicode.org/). This means that over 100 000 characters are supported.

Sorting and searching non-English text can cause a number of problems, a frequent one being how to handle accented letters, for example á, à and â.

The rules for sorting vary because the various natural languages sort words differently. There are occasions where the accented form of a letter is treated as a distinct letter for the purpose of comparison. For example, in Swedish, Å is a separate letter that is sorted after Z. In some languages, it is common to sort uppercase before lowercase, in other languages this is reversed; sometimes it is just a matter of personal preference.

A collation, also known as a collating sequence, is a database object containing a set of rules that determines how character strings are compared, searched and alphabetically sorted. The rules in the collation determine whether one character string is less than, equal to or greater than another. A collation also determines how case-sensitivity and accents are handled.

Mimer SQL has more than 140 collations built in and more collations published on the Mimer SQL developer site (https://developer.mimer.com/). The user can add their own collations in addition to ones that are predefined. This can be done with any existing collation as the base.

Collations can be used in a number of places:

- In a column definition in a CREATE or ALTER TABLE statement. The collation will henceforth be used as default in all column references.
- In a comparison expression. I.e. when comparing two string values it is possible to specify what type of collation to use.
- In an ORDER BY clause. This allows the application to specify the sort order of the rows returned for a query.
- In a secondary index definition. The index will then be ordered according to the specified collation. It is even possible to have several indexes on the same column that use different collations.

This means that the advanced collation logic is readily available for applications with no technical or performance complications whatsoever.

A special collation called CURRENT_COLLATION can be used to make applications independent on a specific collation. Instead, when the current collation is set, all places that refer to this collation are changed. This includes rebuilding secondary indexes using the current collation.

A special sort order exists to sort product codes, or similar, correctly. In this case the system sorts for example: A1, A2, A10, B1, B11 into the correct order. This is called numeric sorting.

Sequences

A SEQUENCE is a construction that returns unique (2-, 4-, or 8-byte) 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. A sequence can be used as a default value for a column or domain.

Sequences can be unique meaning that each value is only returned once. When the sequence is exhausted an error is given. If the sequence is not unique it restarts once it is exhausted.

The next value of a sequence is obtained by specifying NEXT_VALUE OF sequence-name. The current value, for the current user, is obtained through CURRENT_VALUE OF sequence-name.

Secondary Indexes

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. If a column is part of both the index columns and the primary key columns it is only stored once. The secondary index tables are fully maintained internally within Mimer SQL. The algorithm used for this guarantees that the indexes can never be out of step with their base table.

The SQL optimizer will automatically use secondary indexes 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.

Secondary indexes can be defined with different collations as described in the previous section.

Several types of special secondary indexes also exist:

- **Word index.** In a word index each word is extracted from the original column. This allows index use of the builtin functions BEGINS_WORD and MATCH_WORD.
- Pinyin index. In a pinyin index the pinyin text for the corresponding Chinese characters are stored. This allows index use for the builtin functions BEGINS_PINYIN and MATCH_PINYIN.

- Coordinate and location indexes. These allow indexed access to the builtin Coordinate and Location data types to be used. Functions such as INSIDE_RECTANGLE etc. are supported.

Index tables can be accessed directly in a select statement by selecting from the index name. However, they cannot be updated. This is only possible by changing the base table.

Word indexes are particularly useful when a user starts typing in a search box. It is possible to get all valid completions to the text that actually exist in the database by doings a distinct begins-query towards the index. These completions can be displayed in a drop-down box.

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. This method is today used within many distributed transaction handling systems, even for locking based databases, due to its superior conflict handling.

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 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 may also build 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 statement by the application.
- The **Commit** involves using the Read Set and the Transaction Cache to detect access conflicts with other transactions. If transaction serialization is **repeatable read** 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 application 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 statement and can resume its operations.
- During the **Apply** phase the changes are applied to the database, i.e. the databanks, the log, 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.

The above is true if the application is using serialization *repeatable read*. If the application chooses serialization level *read committed* however, some performance improvements are employed by the system. In this case no read set needs to be recorded. Instead, when the application reaches the **commit** phase, the system checks that no write operations in the current transaction overlaps any changes made by other transactions since the **buildup** phase started.

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 or even entire tables, 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 statement whereas deadlocks can occur at any point during a transaction including the commit statement. 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.
- When performing a join operation in SQL the optimizer chooses which table to access first. In an optimistic based system this has no effect on the system, as the order data is accessed is irrelevant to the transaction handling. In a locking system however, this may cause deadlocks to occur. Note also that the optimizer may change the order tables are accessed when the amount of data grows. This may give the effect that deadlocks are introduced in a system that previously ran smoothly. In some circumstances, deadlocks in locking systems may take many seconds to detect due to timeouts in the lock handling.
- 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.
- It can also be noted that applications are written in the same manner for locking and optimistic systems. Both types of transaction handling prefer the use of short transactions. A programmer does not need to write code differently when switching between these two implementation types.

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. It also allows the system to gather changes from several transactions on the same page to be written at the same time, thus giving fewer I/O operations overall.

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 and involves no other users or I/O. In a locking system the actual data in the database must be restored to its original values, making the rollback a more costly operation.

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) are not needed, and for this reason no Read Set is established for a Read-Only transaction (regardless of isolation level). 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. Also, a Read-Only transaction never disturbs any other transactions going on in the system. For example, a complicated long-running query can execute in parallel with OLTP transactions.

It is also possible to set the data in a databank read-only. The transaction system takes advantage of the fact that the data will not change. I.e. no records are written to the read set for this part of the data accessed by an application. So, if data in a table, or a set of tables, is for example only updated once a month it may be beneficial to mark the data read/only between these infrequent changes. This also protects against inadvertent changes of the data in these databanks.

Mimer SQL also supports a mode called **delayed** transactions. In this case when a transaction is committed the transaction is not immediately written to disk. Instead, a timer is set. When the timer expires the transaction is secured on disk. While waiting for the timer other transactions may group commit with the waiting transaction. Should the page holding the transaction become full it will be written before the timer expires. Note that delayed transactions may not pass each other. The system always guarantees the order of transactions.

Delayed transactions are particularly useful for simple transactions. Consider an application doing inserts into a table and each insert is auto committed. Without delayed commits each insert will be written to disk as the transaction is committed. With delayed commit however, the application can group commit the inserts and will thus achieve a considerably better performance. When the application is done the last transaction is flushed as the timeout expires, which is typically after one millisecond. If the machine crashes when the application is running, the database will come back up with a consistent database where all transactions, up to the last millisecond, are all included.

Distributed Transactions

The Mimer SQL product supports the standardized XA protocol for distributed transaction handling. This protocol allows several database handlers, even of different brands, to cooperate with a distributed transaction coordinator. The coordinator is the entity that interacts with the application when transactions are started and committed or rolled back. When a transaction is committed the coordinator interacts with all the participants of the transaction. If all the participants are able to “prepare to commit” the transaction is committed. If any of the underlying transactions fail all transactions are rolled back. All these events, including recovery after a system failure, are covered by the XA protocol.

The functionality gained is that it is possible to perform transactions over more than one Mimer SQL database or database from another vendor that also supports the XA protocol.

Examples of products with this type of support are:

- Oracle Tuxedo
- Oracle WebLogic
- Microsoft Distributed Transaction Coordinator
- And others…

In modern application servers the application server handles the fact that an object participates in a transaction. There is typically not any application specific logic to achieve this.

Storage Structures

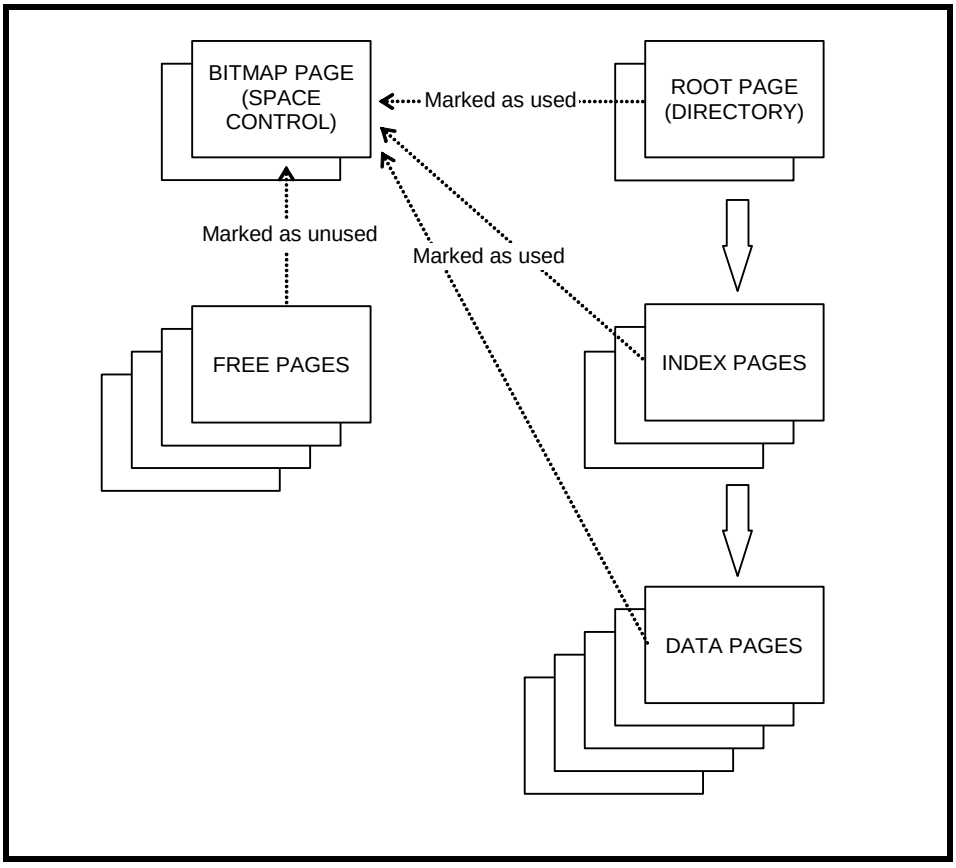
Fundamental to any database system is how the data is stored. In Mimer SQL information is stored in **Databanks**. A databank is a standard Operating System direct access file and contains an arbitrary number of tables. A Mimer SQL database may contain an arbitrary number of databanks at the discretion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

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

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 4, 32, or 128 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. The tree is suitable for both finding exact keys, but also ranges of keys. 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.

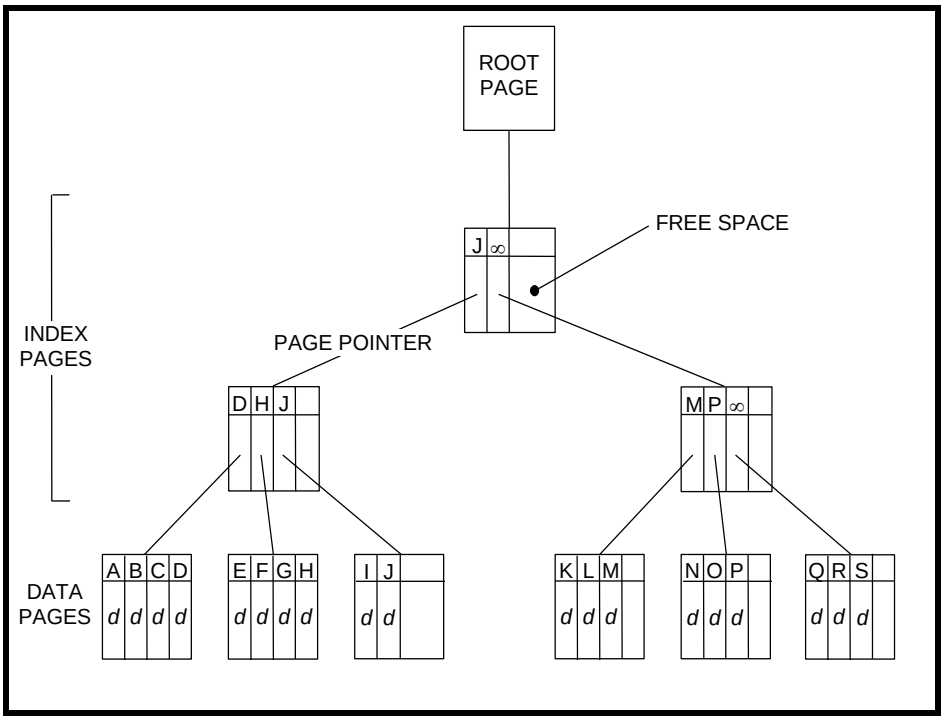


Table Structure (B*-tree)

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. In addition, these reorganizations are typically performed by the background threads, which means that applications never have to wait for them.

To further improve space utilization, Mimer SQL uses variable format records and may also compress 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 compression is done on entire records. This means that all types of data are compressed. So, for example, both fixed length character columns and variable length columns are compressed.

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. This means that no separate utility to clean up the b*-trees is needed as is common in other products.

To further improve performance, the system may pick different block sizes depending on the length of the key and of the non-key parts of the row. It is, for example, possible to use 4K pages for index pages and 32K pages for the data pages.

Large objects are stored together with “ordinary” B*-tree in a databank. Large objects are stored consecutively in a databank if possible. It is also possible to store large objects in chunks of 128, 64, or 4 kilobytes depending on the size of the large object in question. In the row there is a logical reference to the large object so that it can be located efficiently. In effect this means that clients explicitly request access to a large object when the application wants it. Other column types are sent in one chunk to the client whenever a row is requested. This mode of operation allows the row to be fetched and displayed quickly and associated large objects to be brought afterwards. This is very similar to what a web browser does when displaying a page, where the text is shown first and pictures are filled afterwards, one at a time. It also means that the Mimer SQL client software never caches the large object. It is read from the database server communication channel directly into the application buffer. If the large objects are sufficiently small, they are also passed together with the rest of the row data as this gives fewer round trips to the server.

Large objects are stored in whole blocks. If there is a small number of bytes left these are stored in an overflow table. This has the effect of both being very space and performance efficient.

If possible, the blocks are allocated contiguously as this advantageous when disks are used as storage medium.

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 to either the primary key, or one of the unique keys in the referenced table.

For example, you cannot insert a foreign key value into the referencing table 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. Finally RESTRICT, validates each operation in turn rather than at the end of the statement. An error is raised if the constraint is not valid.

Foreign key constraints can be declared as IMMEDIATE or DEFERRED. Immediate constraints are checked as data is modified by the application. With deferred constraint check the system validates that the foreign key constraints are valid at commit. I.e. it is possible to have inconsistencies during the transaction as long as these are resolved at the end of the transaction.

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 (alternate 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). A UNIQUE key differs from a PRIMARY KEY in that it is possible to allow null values in the columns. If a null is specified in one or more columns of the UNIQUE key no referential checking is performed until these values are known. Duplicate values are allowed if a null value is present. To take this one step further there are also unique indexes. A unique index may not be part of a foreign key. In a unique index null is considered a “value”, so multiple rows with the same set of values where one is null is not allowed.

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, functions and methods) 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. Of course, when changes need to be propagated to other tables, triggers are an indispensable mechanism. In Mimer SQL there are both statement triggers and row triggers. Statement triggers have access to all rows changed by a statement, for example all the rows updated by an update clause. Row triggers are called for each row modified, one at a time. The row trigger does not know what other rows are modified.

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 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. A view may contain any valid select statement such as INTERSECT, GROUP BY, or other SQL constructs. These views are not possible to perform insert, update, and delete on unless an INSTEAD OF trigger has been provided for the operation. When a, for example, INSERT is done on the view with an INSTEAD OF INSERT trigger, the trigger is provided with all the insert values and can perform the actions needed to modify the database. The exact semantics is decided by the author of the INSTEAD OF trigger. This provides full view updating.

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.

Mimer SQL differs from other database systems as the database administrator does not have access to all tables in the system. The administrator can still perform backups etc., but are not allowed to select data unless explicitly given access by the table creator.

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. Three 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. USER idents may log into the system using a password. It is also possible to associate one or more operating system users to an ident, in which case those operating users can connect to that USER directly without supplying a password. Only the creator of a user can setup operating system login for that user. It is possible to disallow password login, thus forcing operating system login for a specific ident.
- 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. A user may be member of several groups and in particular all users are member of the PUBLIC group. When a user is a member of several groups, he/she receives the combined privileges of the groups.

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 even on objects owned by other idents

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, collation, or a specified sequence

Object privileges are initially granted only to the creator of the object.

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.

When an instead-of trigger is created on a view the corresponding privilege is given to the user. Only the creator of a view may create an instead-of trigger.

Client/Server encryption

When communicating between database client and database server it is possible to encrypt the contents of the messages. This makes it possible to securely pass sensitive data over the network. It also ensures that messages are not manipulated by someone else as the messages are transferred over the network.

The encryption uses AES-GCM (Advanced Encryption Standard - Galois/Counter Mode). This is an authenticated encryption algorithm designed to provide both data authenticity (integrity) and confidentiality. This means that the data is protected from eavesdroppers, and that any tampering with the data is detected.

Each session will have its unique session key. The keys are not reused, so two encrypted sessions with the same content appear completely different.

It is very simple to enable encryption by setting a database server parameter called "NetworkEncryption". Three modes are supported:

- **None**. No network encryption is used. This is primarily used over non-public networks where the network infrastructure is protected.

- **AES-GCM, optional**. In this mode all clients are accepted. Newer clients with support for encryption use encryption, while older clients will communicate without encryption.

- **AES-GCM, required**. In this mode all network connections are encrypted. Clients that do not support encryption are rejected at login with error code -18531.

When logging into the database the system uses an algorithm called Secure Remote Protocol. With this protocol the actual password is never passed over the network (even when no encrypted). In addition, the actual information passed differs for each login. This protects against malicious users trying to gain access the system by eavesdropping and replaying the session.

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 on two levels: physical and logical.

Physical consistency means that the tables are readable by the Mimer SQL database server. 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. The consistency check is performed in the background to allow quick

access to the system. If an application accesses data that has not been checked yet the system performs a small local check of the pages accessed to ensure stable operation even in the event of a damaged database.

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 statement 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.

When an online backup is made a completely consistent snapshot of the system is taken. The backup files 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 the readlog command in the Batch SQL 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. The alteration is applied separately to each shadow so that any physical corruptions 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 operation the shadowing has no direct 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 30 years in heavy run-time environments all over the world. Mimer Information 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 Mimer Information Technology. Each Mimer version is subject to rigorous Quality Assurance checks before it is formally released.

Replication

Mimer SQL supports replication of data from a source database to a target database. Changes in one system are continuously propagated to the replicated system. Replication is selected on a table basis. To ensure consistency, the tables involved in replication must have a primary key.

If several operations on the same or several replicated tables are performed in a transaction, these are applied together in a transaction on the target system. Thus, transaction semantics are preserved on the target system.

The current implementation is a master-slave replication system. Conflicting changes in the target system are not applied. These operations are instead written to a log.

The current replication feature uses functionality in the Mimer SQL Experience.

MONITORING

There are a number of different ways to monitor a Mimer SQL database server. There exists a number of different tools to view information about logged in users, cache usage, and much more. Here follows a brief summary:

Tool	Description
Miminfo	This is a command line utility that is used to show information such as User information, Performance information, SQL POOL information, and Connection info. The tool can work both towards a live system as well as a dump taken from a server.
SQLMonitor	This utility shows how much resources are used by individual SQL statements executed on the server. It exists both as a command line utility and as a Windows application.
Performance Monitor	This is the built-in performance monitor on Windows. It can plot lots of information from either Mimer SQL database servers running on Windows as well as servers running on other platforms. The program can show both database server statistics as well as operating system. The correlation between the two is, of course, only of interest if you are actually using Windows for your database server. It is possible to both log to file and run interactively. It is possible to set alerts when certain conditions are met. The information displayed can also be retrieved by a custom, user-written, application using the ADO.NET interfaces.
Mimperf	This is a utility that shows various system information displayed at, by default, 10 second intervals. The utility can also write comma separated CSV files in a special format called T4. This format allows information from the Mimer SQL server to be correlated to Operating system information on the VMS platform.

OPENNESS

Mimer Information Technology is committed to conforming to the relational database standards defined by organizations such as ANSI, ISO and The Open Group. Mimer Information 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 3.5 driver and a 100% Pure Java JDBC type 4 driver and a 100% pure ADO.NET Data Provider. Through these interfaces a large number of web-based and Windows-based development tools can be used together with Mimer SQL (e.g. Microsoft Visual Studio, Eclipse, Embarcadero JBuilder and many more). Mimer SQL is also compliant with many CORBA- and Object Transaction Monitor-based middleware products.

The JDBC, ADO.NET, 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. An alternative to Embedded SQL is to use Module SQL. With Module SQL the SQL statements are stored outside the source file in separate Module SQL file. The SQL statements are invoked by calling function names defined in the Module SQL file. Also supported are the SQL standards for constructing SQL statements during program execution (Dynamic SQL), allowing ad-hoc querying of the database from applications.

Programming Language(s)	**Mimer SQL Client**
Java	JDBC
C#, VB.NET, and other .NET languages	ADO.NET or ADO.NET for compact framework
C, C++	ODBC, Embedded SQL, Module SQL, Mimer API or Micro API
Cobol, Fortran	Embedded SQL, Module SQL
Pascal	Module SQL
Python	MimerPy

MimerPy is a database adapter for the programming language Python, commonly used for research in Artificial Intelligence and Machine Learning. MimerPy implements the PEP 249 specification (see https://www.python.org/dev/peps/pep-0249/) and allows a Python programmer access to the powerful tools and advantages of the Mimer SQL database management system.

MimerPy is installed in the same way as all Python extensions by running the command ”pip3 install mimerpy” from the operating system command line.

Mimer SQL is also available from many other scripting languages such as Perl, PHP, and so on. Typically, these languages have a bridge to ODBC that is used.

The programming interfaces to Mimer SQL are multithreaded. 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 type of support is essential when programming GUI-based applications. Mimer SQL clients are thread safe. Typically, the clients run in parallel only when the threads use different connections.

SQL

Mimer Information Technology's policy is to develop Mimer SQL, as far as possible, in accordance with the established standards. The latest standard is called ISO/IEC 9075:2016 and is referred to as SQL-2016.

Mimer SQL support all features of Core SQL 2016 and 145 additional features in the standard.

For each SQL statement documented in the Mimer SQL reference manual, standard compliance for the statement in question is documented. This gives customers a clear picture what SQL to use to follow the SQL 2016 standard.

Mimer SQL also provides support on the web for testing standard compliance for individual SQL statements.

JDBC

JDBC is the de-facto standard for accessing relational database systems from the Java programming language. Mimer JDBC adapts to new versions of JDBC as they become available.

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 small (approximately 190KB 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.

For advanced interoperability with application servers Mimer SQL JDBC driver has three so-called standard extensions. They are: JNDI standard extension, Connection pooling, and distributed transactions. JNDI provides translation of names in an application server. Connection pooling allows many connect/disconnect operations to be processed efficiently. Finally, distributed transactions allow transactions to span several databases.

JDBC for small footprint environments

JDBC comes in two flavors for small footprint environments. The first driver runs under the Java 2 Platform Micro Edition called Connected Device Configuration (CDC). This environment is common on handheld devices supporting Java.

The second driver is for mobile phones running in a more constrained Java 2 Platform Micro Edition environment called Connected Limited Device Configuration (CLDC).

In both environments it is possible to use a Mimer JDBC client to access both database on the device and databases on other machines via a network, for example using GPRS or Bluetooth.

Mimer Information Technology was the first database vendor to give JDBC support in the CLDC environment.

ADO.NET

Mimer Information Technology has developed a fully managed ADO.NET provider for the .NET Framework. The interface exposes all the rich functionality of the underlying database server.

The fact that it is written in 100% managed code allows it to run under .NET Core that runs on both Windows and Linux.

The provider has been integrated with Visual Studio. The documentation is completely integrated and supports Intellisense. It is also possible to drop Mimer SQL specific ADO.NET object onto a Windows Form. When this occurs, a wizard is activated that guides the programmer through a sequence of steps to allow the ADO.NET object to interact with the database server. In the end, code is automatically generated, saving the programmer the work to do this explicitly.

In addition, there are many other wizards. More information about these can be found in the documentation for the Mimer SQL ADO.NET provider. The documentation is integrated into the Visual Studio environment. There is support for index lookup, search, and dynamic help.

ADO.NET for small footprint environments

The small footprint environment is called .NET Compact Framework. The corresponding Mimer SQL ADO.NET provider for this environment is called Mimer Compact Data Provider.

Like its bigger cousin for the .NET Framework, it is written as 100% managed code, making it extremely easy to package and deploy along with applications. Only one assembly is used together with the application with no other dependencies. The provider runs on many flavors of Pocket PC, Windows Mobile, Smartphone, and other compatible environments.

ODBC/CLI

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 3.8 specification. The driver supports applications written for earlier versions of ODBC in the manner defined in the ODBC 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 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 as it eliminates network communication between client and server.

Depending on the environment the Mimer SQL ODBC driver may be used with an ODBC Driver Manager or not. On platforms where the driver manager is not available the application is linked directly to the Mimer ODBC driver rather than being loaded at runtime by the driver manager. In these situations, Mimer ODBC acts like the standard Call Level Interface (CLI). In these environments the mimcli.h file is used instead of the ordinary ODBC include files.

ODBC for small footprint environments

Mimer SQL ODBC has been adapted to small footprint environments. Eliminating functionality that can be achieved in several different ways has done this. Also, only the wide character (Unicode) interfaces are supported as is common in operating system routines on Windows CE.

A special header file (called minodbc.h) is used which reflects the scaled down functionality.

This means that you can use ODBC in environments such as Windows Mobile where this interface may not always be available.

MimerAPI

Like most database products Mimer SQL has its own native database application programming interface (API). The MimerAPI is used with programming languages C and C++.

MimerAPI has important features for writing applications with optimum performance. The API has support for both bulk fetch and bulk insert/update/delete/call. This allows the application to minimize the communication with the database server. This can significantly improve performance.

MimerAPI supports all features and data types in the Mimer SQL database server. It is the preferred way of accessing Mimer SQL.

DbVisualizer

Bundled with the Mimer distributions is a tool called DbVisualizer from Dbvis Software. The tool shows a graphical representation of all objects in a Mimer SQL database. It provides a desktop environment for statement execution.

Of special interest are the menus associated with each object that invokes dialogs to perform SQL operations. The dialogs can display the SQL used to perform the selected action. A great tool to learn SQL and for statements that are used more seldom.

The tool has a graphical representation for the execution path of an SQL statement. This is called an explain plan. The explain plan allows you to evaluate the way SQL queries are written. It also helps in evaluating suitable indexes etc.

Web-based Database Application and Enterprise Applications

Mimer SQL's stability in run-time environments and the built-in support for JDBC, ADO.NET and ODBC makes it ideal for building dynamic web-based database applications and Enterprise Applications. Data can be accessed from a Mimer SQL database and included in web applications using standard techniques such as J2EE, PHP, Perl, and ASP.NET. The same techniques can be used in rich client applications using Web Services or the native client/server protocol in .NET and J2EE.

To accomplish the above some form of middleware application server or web server scripting plugin like PHP and Perl can be used. The most common application servers include J2EE servers like JBoss, Bea Weblogic, IBM Websphere, and Oracle OC4J and non J2EE servers like Microsoft .NET, Coldfusion, Zope, and many others.

The use of J2EE or .NET makes it easy to build transactional applications with support for distributed transactions and other advanced features. Often some kind of Object Relational mapping framework (OR) like EJB CMP, Hibernate, or NHibernate is used. This way the developer can focus on the application logic and leave the database operations to the framework.

Standard interfaces like ODBC, JDBC, and ADO.NET are used to connect Mimer SQL to the application servers.

Mimer SQL is used with a variety of these including:

- Apache Tomcat
- Oracle Weblogic
- ASP.NET
- JBoss
- PHP
- Perl
- OracleTuxedo
- Coldfusion

To build Web and Enterprise applications standard development tools like Visual Studio, DbSchema, Eclipse, and many others can be used.

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 Linux, Open/VMS, Windows, macOS, QNX, VxWorks, Greenhill Integrity etc. 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, Named Pipes or similar protocols.

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:

Data type	Description	Range
CHARACTER(n)	Character string, fixed length n.	$1 \leq n \leq 15000$
CHARACTER VARYING(n) or VARCHAR(n)	Variable length character string, maximum length n.	$1 \leq n \leq 15000$
CHARACTER LARGE OBJECT(n[K|M|G]) or CLOB(n[K|M|G])	Variable length character string measured in characters.	$1 \leq n \leq$ 9,223,372,036,854,775,807
NATIONAL CHARACTER(n) or NCHAR(n)	National character string, fixed length n.	$1 \leq n \leq 5000$
NATIONAL CHARACTER VARYING(n) or NVARCHAR(n)	Variable length, national character string, maximum length n.	$1 \leq n \leq 5000$
NATIONAL CHARACTER LARGE OBJECT(n[K|M|G]) or NCLOB(n[K|M|G])	Variable length national character string measured in characters.	$1 \leq n \leq$ 3,074,457,345,618,258,602
BINARY(n)	Fixed length binary string, maximum length n.	$1 \leq n \leq 15000$
BINARY VARYING(n) or VARBINARY(n)	Variable length binary string, maximum length n.	$1 \leq n \leq 15000$
BINARY LARGE OBJECT(n[K|M|G]) or BLOB(n[K|M|G])	Variable length binary string measured in octets.	$1 \leq n \leq$ 9,223,372,036,854,775,807
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
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.	$1 \leq p \leq 45$ $0 \leq s \leq p$
NUMERIC(p,s)	Exact numerical, precision p, scale s.	$1 \leq p \leq 45$ $0 \leq s \leq p$

Data type	Description	Range
FLOAT(p)	Approximate numerical, mantissa precision p.	$1 \leq p \leq 45$ Zero or absolute value 10^{-999} to 10^{+999}
REAL	Floating point value with 24-bit binary mantissa.	Zero or absolute value from 1.40129846^{-45} to 3.40282347^{+38} Corresponds to single precision float. The IEEE floating point standard supports the special values -Inf (negative infinity), +Inf (positive infinity), NaN (Not a Number) and -0.0 (Negative zero). None of these values are permitted in a Mimer REAL column. The value -0.0 will be converted to 0.0.
DOUBLE PRECISION, FLOAT	Floating point value with 53-bit binary mantissa.	Zero or absolute value from 4.9406564584124654-324 to 1.7976931348623157+308 Corresponds to double precision float. The IEEE floating point standard supports the special values -Inf (negative infinity), +Inf (positive infinity), NaN (Not a Number) and -0.0 (Negative zero). None of these values are permitted in a DOUBLE PRECISION column. The value -0.0 will be converted to 0.0.
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 on the type of interval.	Described below.
BOOLEAN	Used to represent truth-values.	True and False
BUILTIN.GIS_LONGITUDE	A geographic coordinate that specifies East-West position on the earth.	A value between -180.000 and +180.000.
BUILTIN.GIS_LATITUDE	A geographic coordinate that specifies North-South position on the Earth.	A value between -90.000 and +90.000.
BUILTIN.GIS_LOCATION	A combination of longitude and latitude to specify a location on the earth	Limited by restriction on GIS_LONGITUDE and GIS_LATITUDE.
BUILTIN.GIS_COORDINATE	A combination of an X and a Y value in a coordinate system.	Each part, X and Y, is an Integer value.
BUILTIN.UUID	A 16-byte universally unique value.	Stored as a BINARY(16).

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.

Mimer SQL also support distinct types. Distinct types are formed from a predefined data type. Distinct types are strongly typed, which means that it is only possible to compare values of the same type. When comparing a predefined data type and a distinct user-defined type a type cast must be used.

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. A sequence can be either a SMALLINT, INTEGER, or BIGINT (16-, 32-, and 64-bits integer respectively).

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.

Mimer SQL embedded

Mimer SQL can be customized for a particular environment. It is possible to customize both functionality and footprint. Using this capability, a number of editions has been created of Mimer SQL. Please note that the editions use the same source code, so that the database engine is the same in all cases.

All editions of Mimer SQL support a rich set of functionalities such as:

- Stored procedures
- Triggers, including instead-of triggers
- Referential integrity
- Primary keys
- Check constraints
- Unique constraints
- Views
- All data types

Mimer SQL Mobile is a database server for high-end mobile phones and PDAs. It features a multi-threaded database server.

Its smaller cousin the Mimer SQL Micro server is a single threaded server. This means that the server handles the database request from one application at a time.

The smallest edition of Mimer SQL is the Mimer SQL Nano server. It can run with as little as 30k RAM.

The following table summarizes the main differences between the different editions of Mimer SQL.

	Mimer SQL Nano	**Mimer SQL Micro**	**Mimer SQL Mobile**	**Mimer SQL Enterprise**
Record length	450	3000	3000/16000	32000
Block sizes	4k	4k and 32k	4k, 32k, and optionally 128k	4k, 32k, 128k
Thread-model	Single	Single	Multi	Multi
Sequences	Optional	Yes	Yes	Yes
Collations		Optional	Yes	Yes
Floating point functions				Yes
Dynamic SQL				Yes

As can be seen the main difference is the width of a record in a table. The reason for restricting the record length is to conserve memory. Depending on the block sizes used different length records are supported. Note, however, that the record length does not

include large object columns (BLOB, CLOB, NCLOB). These are supported in all editions of Mimer SQL (a large object column occupies 21 bytes in the record).

Collations use very little RAM, but do use some read-only memory to support the different languages.

By eliminating floating-point functions, the server can run in environments that does not support floating-point arithmetic.

Mimer SQL Real-time edition

Mimer SQL Embedded, Real-Time Edition, is a scalable and zero maintenance database management system for embedded systems on standard platforms as well as on tailor-made appliances and mobile devices. By combining predictable hard real-time database access with the non-real-time world, it enables integrated data management solutions impossible to implement with other DBMS products on the market.

Safe data sharing at run-time

Mimer SQL Embedded, Real-Time Edition, allows both real-time tasks and non-real-time data processing to independently access real-time data without risking unpredictable transaction aborts or blocking. All access to real-time data from the non-real-time world is made through standard SQL.

Deterministic hard real-time database access

Mimer SQL Real-Time Edition, uses efficient in-memory database pointers to guarantee fast and deterministic hard real-time database access. Examples of suitable uses include data management of real-time sensor/actuator data, event logging, diagnostics, producer/consumer, and data streaming applications.

Small footprint

The enterprise version of Mimer SQL has a footprint of only a few MBytes. The footprint of a tailor-made version of Mimer SQL Real-Time Edition, can be reduced to 500 KB or even less, still with support for full SQL and concurrent database access. I.e. the real-time functionality can be combined with any of the editions described in the previous section, such as Mimer SQL Experience, Mimer SQL Mobile, Mimer SQL Micro, and Mimer SQL Nano.

Shorter time-to-market

By combining real-time and non-real-time data in one single Mimer SQL database, there is no need for application specific data transfer and duplication solutions. This, together with the full support for the ISO/ANSI SQL standard, will reduce the development time for database applications significantly.

Real-time functionality

Currently the following data types are supported from the real-time interfaces:

- Integer (16, 32, and 64 bit)
- Character string
- Float (single and double precision)
- Timestamp

The real-time data can be saved persistently by flushing the data to disk. These flush operations do not interfere with the real-time access. The data that is saved is in a consistent state, i.e. the data saved is between two real-time updates of the data.

The following table shows the real-time operations supported:

Operation	**Description**
Single value database pointers	Read/Write individual database values. One database pointer per value.
Multicolumn database pointers	Atomic Read/Write of multiple column values of a database record. One database pointer per record.
Multirow database pointers	Read/Write of database value in multiple database records of a table. Writes are performed in sequential order. Reads are performed in sequential, backwards or arbitrary order. Can be combined with multicolumn functionality. One database pointer per table.
Database pointer flushing	Flushing of any database pointer to persistent storage. Will not block real-time operations.

Mimer SQL In-memory edition

Mimer SQL In-memory is a database where all data is stored in main memory. When an In-memory server is started its initial state is given by the databank files belonging to the database. This can be an empty database or a database with tables, view, users etc.

All operations done toward the server is kept in the database cache. This allows the In-memory version to operate extremely fast as no data is written to disk.

It is still possible to use transactions as usual. In fact, almost all operations that are applicable (>99%) of a Mimer SQL server is supported in the In-memory server. An application that runs towards an ordinary Mimer SQL experience server can run, unchanged, against an in-memory server.

Any objects created are only present in the database cache. For example, the create databank statement that ordinarily creates a file in the file system, only works toward memory. The same goes for tables and any other objects created.

Saving the state of the server

When an in-memory server is stopped the data in the data is not saved. It is reset to the state it had before last startup.

It is, however, possible to save the state of the server by taking an online backup of the database. This backup can be used in the same or another in-memory server or in an ordinary Mimer SQL server.

The In-memory server is ideal for testing purposes and/or loading of lots of data. The load program runs quickly and then an online backup is taken. The size limit of an In-memory database is the maximum size of the database cache. This is currently 327 terabytes.

Mimload

Mimer SQL provides a flexible method for loading information to and from databases using the LOAD and UNLOAD commands and the MIMLOAD program.

The MIMLOAD program is used directly from the operating system command line prompt. Using the STDERR, STDOUT and STDIN options in the LOAD/UNLOAD syntax, command line file redirection for input, output and logging is enabled. Both database definitions and data are handled.

The LOAD command copies definitions and/or data from one or more files. The UNLOAD command generates data and/or definitions and places the result in a file.

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.

- HP Integrity/OpenVMS
- Linux (Red Hat, SuSE, Ubuntu and many more)
- Microsoft Windows
- macOS
- Windriver VxWorks
- ENEA OSE
- Embedded Linux
- QNX
- Greenhill Integrity

Please check the Mimer SQL developer site for an up-to-date list!

https://www.mimer.com

https://developer.mimer.com

Mimer Information Technology AB
Kungsgatan 64
SE-753 41 UPPSALA
Sweden
Tel: +46 (0)18-780 92 00
Email: info@mimer.se

Version 11.0 Dated 18 May 2021