

Lava Distributed SQL

Tesseract Digital Information (Pty) Ltd

A New Database Concept

The conventional database is increasingly taking strain as a result of the increasing demands placed by more complex applications and greater data loads. In many installations, operability has become marginal, and often data archiving has become the only way to maintain a semblance of acceptable operation.

In several ways, the Lava System is a revision of the database model. Several mechanisms have been revised or reinvented.

- Table data interface uses previously unpublished interfaces in the Windows kernel to achieve unprecedented data rates
- Indexing is achieved using a completely new technique, allowing vastly improved seek times
- SQL performance is improved through application of expert system techniques to optimize execution
- Relational processing is vastly improved through the use of a new approach to primary key access
- Overall system performance is improved by multiples through use of data distribution to the client

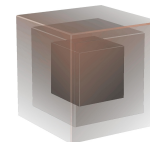
As a result of these and a number of more minor revisions, the Lava system can handle data and client loads which would be impossible with a conventional database.

The Approach

The Lava database presents a new concept in data storage and retrieval. Every facet of the database was conceived as a response to the problems relating to conventional databases.

Initial design meetings clearly illustrated the requirement for a unique and revised approach to the design of the database. After careful analysis of the problem domain and potential design techniques, it was clear that more than just the development of a database had to be taken into account.

Many of the traditional concepts in software design and development were reviewed and in most cases reinvented. Obvious shortcomings in conventional programming approaches resulted in a completely revised design and implementation technique. Problems associated with existing compilers resulted in the design of a proprietary compiler with several unconventional capabilities. Omissions in commercial linkers resulted in the design and implementation of a proprietary and multifunctional linker. Discrepancies in project management as presented by current programming tools resulted in the design and implementation of a new concept in software project management, targeted specifically at



large projects with multiple programmers.

Once basic tools of an acceptable standard were available, the principles of databases were revised without preconception. Methods of data storage and retrieval, and most especially indexing techniques, were approached in a new and highly innovative way, resulting in a system design which is a departure from the 30-year old model still used by almost all software houses.

Database Innovation

With almost no exceptions, all current databases are created according to the same formula :

- Ability to store data in row / table form
- Indexing ability on table column data
- SQL parsing and processing
- Simple network communication protocol to thin client

This formula has been in use since the early 1980's, and with reasonable success. It does not, however, have to be the only formula - or even the best.

Similarly, a standard formula has applied to database-centric client application development :

- The client stores no data locally
- All data is fetched directly from the database server, every time
- Updates to data are immediately transmitted to the database server
- Most if not all data access is achieved through SQL

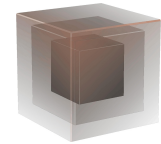
Again, this is the way things have been done for decades, and few have complained. Yet this model is incredibly inflexible and in most cases not particularly efficient.

This classical construction of a SQL database and related client applications places a clear separation between the client application and the database, making the application design and implementation very difficult.

A more useful and efficient implementation would have the client application link directly to the database, allowing a wide range of access methods (including SQL, row-level interfaces and even array access) as well as flexible, efficient access to all data. Amongst many benefits would be the possibility of using the database as a means of storing, manipulating and retrieving application temporary data, which can become quite significant and rather complex in the more demanding applications.

The complication with this method is that only one client can execute at any one time, since the client is now mounted exclusively to the database - precluding other applications from doing this simultaneously. For some applications this might be acceptable, but as a general approach it is fatally flawed.

In order to support this exclusive connection between application and database as well as the requirement for multiple simultaneous users, a new database model is required. The client interface to the server has to become a database in its own right, and the server database has

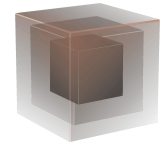


to be replicated to this client database. In addition, the client database has to be able to communicate any changes in data to the server, and vice versa. Lastly, the entire database server has to be very efficiently coded, and must place relatively low demands on system resources in order that the average workstation is capable of running the application. A Distributed SQL database is the result.

Distributed Architecture

After a month of research into existing database design approaches, it was very clear that the desired results would only be obtained using a fairly major departure from existing techniques. Several items resulted from initial prototyping tests :

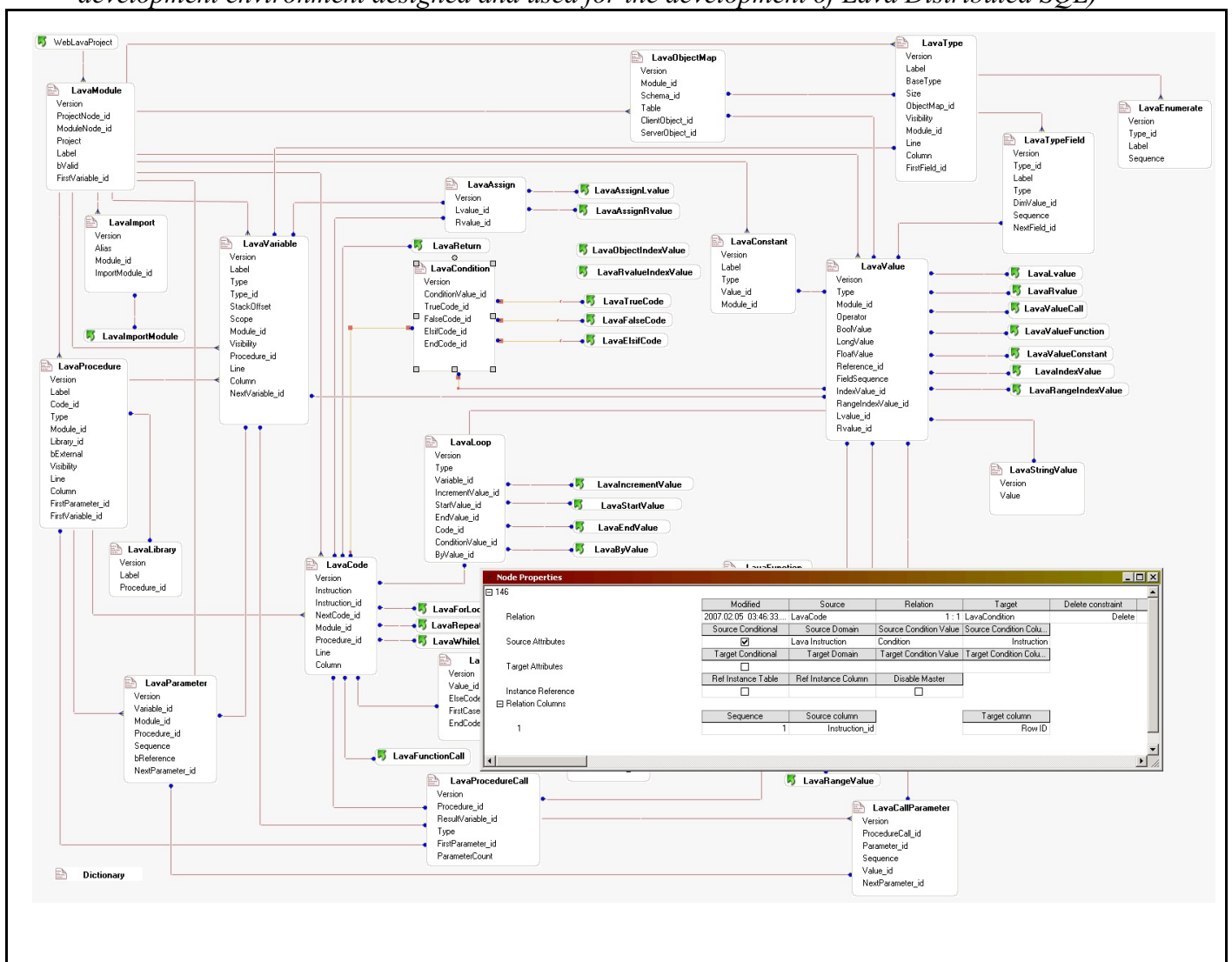
- Existing indexing techniques were unworkable. A new approach to indexing had to be invented which allowed faster (including on-the-fly) construction and seeking.
- Support of SQL was obviously required, but various conceptual requirements (most notably within the core itself) motivated for the addition of a comprehensive row-level interface, allowing greater flexibility and better performance in application as well as kernel construction. Careful design allowed the kernel row-level facilities to be exported unchanged to the application programmer interface.
- Beyond conventional row-level access, many application requirements showed the advantage of array access directly into data tables. This became central to the table interface design.
- Although the commonly used string-based relationships between tables would be supported, a new relational method based on direct access to the primary key would be provided, allowing vastly improved performance.
- To provide for subsequent addition of Time-Domain techniques, date-time stamping of each row would be mandatory and implemented automatically by the database kernel.
- Despite the possibility of other approaches, size and efficiency constraints imposed by the requirements of the distributed client motivated for a database kernel that booted from only standard tables - in other words, instead of having a “custom” kernel table set with dedicated boot software, the database kernel would be built entirely on standard tables - exactly the same as user tables.
- The concept of a distributed client was central to the design of the kernel. This implied, amongst others, the design of a two-way communication protocol for commands and data. Additional considerations (such as the completion port mechanism selected for the socket layer) dictated the use of an asynchronous threaded finite state protocol recognizer, with efficiency and performance an absolute priority.
- Experience with several stored-procedure languages pointed to the requirement of a programming language which not only supported stored procedures and triggers, but could also be used for batch processing (as well as later diversification into full-fledged OLTP). A flexible and very nearly general-purpose language specification resulted.
- Distribution considerations pointed to the requirement for an interpreted programming language which stored its byte-code in a conventional database schema, allowing automatic distribution of the executable to all clients through the standard data distribution mechanism. Adoption of this goal dictated the architecture of the language runtime.

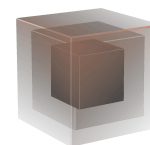


The combination of these and numerous other (smaller) requirements resulted in a dramatically different approach to database kernel design. Very careful functional abstraction combined with recursive refinement of both the core kernel functions and the interfaces to these resulted in an exported application interface of approximately 220 functions in the kernel, 30 functions in the SQL layer, and a runtime environment supporting nearly 300 exported functions independent of the database kernel. On completion of development and testing, this amounted to approximately 200,000 lines of carefully crafted source code in 159 modules. This relatively small code set combined with the generation efficiency of the compiler translates into approximately 4.3MB of executable code

The following subset of the LavaStream (stored procedure language) bytecode schema illustrates a few of the principles adopted.

(The Entity Relationship Diagram presented below is captured directly from the software development environment designed and used for the development of Lava Distributed SQL)





A few items of interest in the diagram :

- The presence of a “Version” column in all the permanent tables
- Relationships based on the automatic Primary Key (part of the Version structure)
- A relationship (cascade delete) conditional to a value in a nominated column¹

Legacy Mode Operation

Despite the obvious motivation for distributed mode, it is clear from a number of perspectives that no database can completely forego the traditional access methods. The most obvious reasons for this include migration from other databases and operation of third-party languages (such as PHP).

For this reason, the Lava database includes all the expected mechanisms as used in conventional database applications :

- SQL support

The support of SQL within the database was an obvious requirement, but the implementation for distributed database applications resulted in several optimization and execution techniques which would not be valid in a conventional client-server topology. Pragmas and clauses were added to provide for legacy execution where this was required.

- Relational Processing

In a distributed environment, the Primary Server does not actually process relational dependencies - these are all handled by the distributed client in order to reduce server load. Conditional execution was added to allow for relational processing on the server during client-server operation.

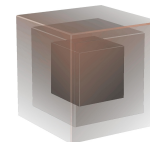
- Stored Procedures

Stored procedures and stored functions were a mandatory inclusion in the database, and no special processing or exceptions were required for client-server operation.

- ODBC drivers

To support access to the database from third-party applications such as report writers, as well as applications newly ported to the Lava database with no existing support for the Lava API, a comprehensive level 3 ODBC driver was implemented.

¹ The displayed relationship allows a more natural mapping between logical schema and physical schema - the conditional column value (in this case the *Instruction* column of the *LavaCode* table) allows a conditional relation between *LavaCode* and (in this case) *LavaCondition*, for condition instructions only.



In addition, to provide for ease of data import from other databases, an ODBC interface was included in the stored procedure language (LavaStream). This allows comprehensive access to alternative databases and high-speed import of any amount of data through to hundreds of millions of rows from all databases providing a reliable ODBC driver.

- Database API

A custom API was envisaged from the conceptual phase - special effort was made to ensure that the API is completely unchanged whether accessing a distributed client or an exclusively mounted primary server.

Major Database Functionality

Although describing the majority of the functionality integrated in the Lava database, the following list is certainly not comprehensive, and is presented in no specific order.

Lava SQL

The SQL implementation in the Lava database is fairly comprehensive, and includes features permitting advanced queries to be constructed :

- Correlated Subqueries
Subqueries may be used in any of the major divisions of SQL statements, with alias-based correlation used to link subqueries to the main statement
- Stored Functions
Any LavaStream function with an appropriate return type may be used in either the column or object list of a SQL select statement. Given the advanced nature of the LavaStream language, this allows results of arbitrary complexity to be constructed.

SQL Optimization

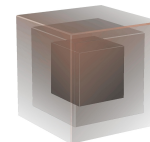
Manual execution plans for SQL statements may be entered, but this is usually not necessary. The Lava SQL engine includes a highly sophisticated optimizer which will under almost all circumstances arrive at an acceptable execution plan.

Views

Views may be constructed from any valid SQL select statement. View regeneration is limited to the absolute minimum and occurs only on demand after a data modification has necessitated this.

Transaction Frames

Both global and local transaction frames are supported, with either global or local



rollback or commit.

The database also fully supports autocommit mode, which may be selected or deselected as desired by the programmer.

Stored Procedures

The stored procedure language in the Lava database is known as LavaStream, and consists of an extremely powerful general purpose programming language, which is fully procedural to the extent of supporting recursion.

The LavaStream language is tightly integrated with the Lava database, to the point where data tables are accessed exclusively as arrays.

Language facilities include direct Windows text file access, output and analysis of XML, comprehensive mathematical functions, full access to ODBC-enabled databases, and much more.

LavaStream compiles to bytecode to allow extremely fast execution, and the resultant executable image is stored in distributable form in the Lava database. All distributed clients are automatically provided with a new bytecode image when a release build is performed on a given project or module.

Stored Procedure Programming

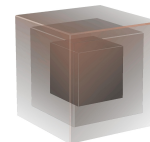
The Lava system is integrated with the Blueprint development environment, conceived and designed with the database itself. This environment provides facilities for project management of large-scale LavaStream projects, includes syntax-aware editing, compile and make facilities. In addition, a symbolic debugger is integrated into the editing environment, allowing extensive and detailed testing of LavaStream stored procedures and batch programs with visibility on all variables and row data.

Procedure execution

Due to the scope of the LavaStream language, it is typically used not only for stored procedures but also for batch programs of arbitrary complexity, used either to pre-process data for a report writer or to import and merge data from external sources. In order to allow this form of execution, a fully distributed client environment is provided which allows selection of a nominated module and procedure, entry of required parameters, and monitoring of execution.

Triggers

The LavaStream language is also used where triggers are required on data tables. Procedures of arbitrary complexity may be linked as triggers to a specific table.



Distribution

In order to allow the maximum flexibility especially with very large databases, data tables may be defined as fully distributed, distribution on demand or non-distributed.

Fully distributed data tables are available in their entirety in the client database, and may be accessed as if the client is a single-user database. This mode of distribution is suitable (depending on network constraints) to tables up to a million rows.

On-demand distribution allows the administrator to configure very large tables to perform no default distribution to the client. Data is distributed as required - in other words, when a particular row is accessed on the client, it is requested from the upstream server and is distributed to the client immediately. Under certain circumstances, lookahead estimation is performed and rows deemed to be required soon are also distributed at this time. Once distributed, this portion of the data table behaves as for a fully distributed table.

Where circumstances require, specific distribution constraints may be configured differently for individual users - in this way, the default distribution for a particular schema may be different for users connecting on the local area network or those connecting through the Internet.

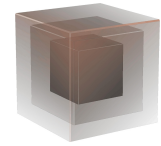
Security

Connection to the database requires a valid username and password, which may be of arbitrary complexity within the constraints of ASCII text.

All communication to and from client applications - whether distributed or not - is via an encrypted and packetized protocol.

As is always the case with client-server application communication, typical request / response communication is based on complete data sets - in other words, a typical request would be a well-formed SQL select statement, and the response would be the complete data set conforming to the request. Despite encryption, this mode of communication does allow for data extraction by unauthorized observers, especially where such communication is across the Internet.

By contrast, with distributed clients the request / response communication is in almost all cases related to the underlying data elements and not complete data sets. For example, an update to distributed data (originating either from the client or from the upstream server) will consist of a proprietary data array listing specific rows of data from (typically) different data tables, presented in a completely arbitrary arrangement and of course also encrypted. The resulting communication stream is essentially indecipherable to all but the Lava database system.



ODBC Driver

A comprehensive ODBC driver is supplied to permit access to the Lava database from applications such as report writers and programming languages such as PHP. This interface also allows existing programs, designed and written for conventional databases, to be superficially ported to a Lava database without extensive redesign.

Primary Server

The Primary Server is the central and permanent data repository in a Lava system. All data, including project data and LavaStream source code created in the Blueprint development environment, are stored by the Primary Server.

Satellite Servers

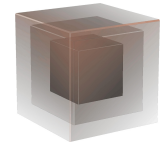
The Lava Satellite Server is a fully distributed server, providing connection facilities to client applications. The Satellite Server provides two important extensions to the Lava System.

Firstly, Satellite Servers may be used in an environment where the sheer number of clients would be overwhelming for a single server. In this case, several Satellite Servers may be connected to the Primary Server, and in turn each Satellite Server may be connected to a number of clients. This allows the total supportable client load to be increased substantially over a single server system.

Secondly, Satellite Servers may be used to provide enhanced performance to remote sites. Where the bandwidth to a remote site (or network latency) is not adequate to allow acceptable response to client applications, a Satellite Server may be installed at the remote site. The Satellite Server is connected to the Primary Server at the central site, and local users at the remote site connect to the Satellite Server. This provides vastly improved performance and response to each remote user, while significantly reducing data traffic to the central site due to the Satellite Server supplying the majority of the data requested especially by new connections.

Server Event Scheduling

Due to the fully distributed nature of the LavaStream executable image, it is possible to schedule the execution of a LavaStream procedure on any active Lava Server (Primary or Satellite) at a predefined time. Scheduling may also be specified as periodic. This allows repeated tasks (such as data import and merge) to be performed remotely and automatically wherever a Lava Server is installed and connected.



Standby Server

The Standby Server is simply a hot standby replication server. All data is continuously synchronized with the Primary Server.

In contrast with some replication servers, the Lava Standby Server does not have to be mounted at exactly the same time as the Primary Server. Due to the inherent distribution capability of the Lava system, the Standby Server may be mounted at any time after the Primary Server, and synchronization will automatically complete as soon as bandwidth and data volume constraints allow.

Server Monitor

The Monitor is a small service application which may be started on any machine hosting a Lava Server. The Monitor allows independent monitoring of Server operation and status. This status information may be viewed from the Lava Administrator, but is also used to ensure server continuity. The monitor is able to detect server malfunction - such as the dispatcher queue no longer processing commands - and is able to stop and restart the server service. In such cases, clients will automatically reconnect to the remounted server, which should be fully functional once more.

Database Administration

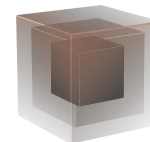
All important database administration tasks are supported through a simple but comprehensive graphical user interface in the Lava Administrator. Tasks ranging from creation of a new user through killing an undesired session to performing database backup or restore are all supported with ease of use the primary goal.

Query interface

In order to test SQL statements or perform routine data maintenance or queries on the database, the query interface provides a multiple document interface with result data grids for issuing select statements, as well as an ODBC link which allows data tables to be imported directly from foreign databases. Simple administrative tasks are supported through SQL, while table-specific backup and restore operations can be undertaken using a custom interface.

Backup and Restore

The backup and restore wizards allow selection of schemas for backup or data sets for restore with the minimum of complication. Both backup and restore speed is extremely fast - on a moderate specification server it is possible to restore a database of over 200 million rows of data in approximately 20 minutes.



Slice Backup and Restore

Unique to the Lava database, this facility allows the ultimate in flexibility when porting data. A slice backup allows the programmer to nominate a single row in any data table as the starting point for the backup. The system will then generate the backup from all related rows in all related tables, recursing downward as far as further relations require. The restore will allow all this data to be written to another database, with the additional (and necessary) twist that each row is placed in the first available slot in the target table. In order to ensure consistency of relations, each foreign key - where necessary - is modified to reflect the revised placement.

An example of such an operation would be to do a slice backup of a particular invoice. This would back up the invoice, all details for the invoice, and all attributes (stock, customer, supplier...). On restoring the invoice to a second database, all attributes to the invoice would be consistent after restore.

Blueprint Development Environment

The design environment integrated with the database provides a multitude of functions ranging from entity relation design through coding and testing of LavaStream procedures. It is probably sufficient qualification to state that the entire Lava project was designed and implemented using the Blueprint environment.

Database Design

Formal and comprehensive entity relation design systems are fairly scarce these days. The Blueprint environment provides one of the most powerful schema design mechanisms available.

In addition to a fully hierarchical schema design system providing for comprehensive table and relation design, the design database is fully accessible through LavaStream. Two comprehensive worked examples are provided to illustrate how to export a dictionary design to text based database definitions - one of which is for C or C++ table definitions and interfaces.

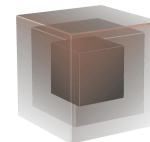
Data Import

In order to support the import of foreign data, several facilities are provided.

Firstly, foreign database table definitions may be directly and automatically imported into a Blueprint data dictionary.

Secondly, a LavaStream import generation template provides for the automatic generation of LavaStream source code which will import the foreign table data into the newly defined data tables.

Should any modifications to the import mechanism be required, these may be made



either at the template level or to the resulting code.

Once the import code is correct, arbitrary amounts of data may be automatically imported from any ODBC source at very high speed - data rates of 3,000 rows per second may be sustained indefinitely from most sources.

Functional Comparison

Although the Lava system is very different from all current commercial databases, there is some basis for comparison in various domains. The following provides a few points of reference.

Sybase

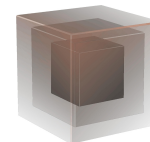
Sybase is a competent and highly functional database. There are a few issues worth mentioning :

1. Administration of a distributed Sybase database network is not simple. In comparison, administering a Lava system is trivial.
2. Required bandwidth is high. In order to operate distributed servers, Sybase requires bandwidth in the multi-megabit range. Lava is capable of operating distributed servers across bandwidth as low as 250 kbit/s
3. Data throughput is not as high as the equivalent Lava distributed database. Both retrieval and insertion rates are approximately 10 times lower.
4. There is no true distributed client available. Sybase do provide distributed servers, but no true distributed client.

MySQL

MySQL is a competent SQL database with good functionality. However, there are a few issues where functionality is lacking or compromised :

1. The database does not support table aliases (synonyms). This causes several problems in large system design, and is a significant inconvenience to the programmer.
2. Several strange operational issues are present in the database - for example, the database automatically updates the first date-time column declared in any table. This leads to designers adding “dummy” columns to get around undesired functions such as these.
3. Although stored procedures are supported, the language is restricted and editing and debugging facilities are poor or unsupported - this makes large or complex project development very difficult.
4. Although a programming API is presented, it is restricted to fewer functions than are required for true ease of programming, causing development time to be longer than necessary. The alternative of using SQL for interfacing falls back to the conventional client application model which is far from ideal.



5. Large and complex SQL statements execute far slower than is the norm for a first-class SQL database. Although queries on single large tables are executed reasonably fast, multiple-table queries typically yield longer execution times than are desirable.
6. Several important functional issues are poorly supported - an example would be sequences, which are not supported in a conventional manner and have to be simulated through creation of a redundant table with an auto-incrementing ID.
8. Backup and restore of data is supported, but operate exclusively to text files (essentially SQL scripts) and it is possible to create backups for the database which cannot be restored successfully due to character set clashes. Both backups and restores have very poor performance.
9. The future of MySQL is highly uncertain. Since the purchase of Sun by Oracle (and in fact even before that) all the technically competent and experienced programmers originally involved in MySQL development have left the company. Development of the product is therefore put on hold, while even required maintenance is not guaranteed.

In terms of the above issues, the Lava database is a complete solution. All of the above issues are fully addressed, providing the designer with a fully workable solution for system development. A future growth and development path is mapped out for years to come.

SQL Server

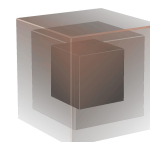
Microsoft's SQL server is a highly functional database but again there are some issues which yield development problems :

1. Server performance is compromised. To support a reasonable number of users in an average application, a very powerful server is required. Once user numbers or application complexity raises to high levels, SQL server typically cannot cope with demand, and as a result is not sufficiently scalable for very large or complex applications.
2. Large objects / file objects are supported, but performance is compromised. Where large numbers of files need to be inserted into or extracted from the database at frequent intervals, this will yield performance problems.
3. Licencing is expensive in terms of performance limitations, especially for large implementations and high user counts.
4. Database administration, although less of an issue than with Oracle, still forms an additional cost and complication factor for the end-user.

Both the performance and administration issues are significantly better in a Lava database. In fact, performance can under some circumstances be 20 times higher or more if distributed functionality is utilized.

Oracle

Of the listed databases, Oracle - which is also the most expensive- is the most capable, delivers the best performance and is also the most scalable. There are still two issues which argue against using Oracle :



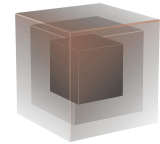
1. A database administrator is mandatory, even for the first installation of the database. Every operation, including backup and restore, requires highly specialized knowledge and even competent users will not be able to manage the database without professional assistance. Oracle administrators are very expensive, and form a significant factor to total operating cost as well as being an operating risk due to the highly complex nature of some administrative tasks.
2. Licence cost is very significant. Even small installations have significant cost attached, and large installations quickly become very expensive indeed.

These issues are irrelevant in the Lava database. Licencing costs is low by comparison and administration is almost nonexistent where the end user is concerned. Mandatory administration tasks may be easily undertaken by any competent technical person, without highly specialized training being required.

Database Performance

The following specifications are based on a low cost server (3 Ghz processor, 4Gb ram, 15,000 rpm uncached drive, Windows Server 2008)

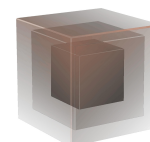
Benchmark	Performance	Comments
Typical insertion rate	20,000 rows per second	Based on a 1,000 byte row size, table having 3 indexes
Typical extraction rate	30,000 rows per second	Sequential extraction
Typical seek time	10 microsecond	Seek time excluding row extraction
Typical read time	25 microsecond	Row extraction after seek
LavaStream row processing rate	4,000 rows per second	Based on 100 lines of LavaStream code per loop. Note that 3GL performance would be closer to the insertion and extraction rates quoted above.
Index creation	12 seconds	1,000,000 rows of data
Index creation	30 minutes	100,000,000 rows of data
Query result sort	30 seconds	2,000,000 rows in result set
Mount time, Primary Server	1 minute	Time taken until server accepts connects, database containing 200,000,000 rows of data. A further 5 minutes



		of diagnostic execution (typically 3% cpu activity) until server becomes dormant.
Data distribution to client	7 seconds	100 Mb of data across a 100 Mbit/s LAN
Total client mount / connect / distribute time	11 seconds	100 Mb distributed data across a 100 Mbit/s LAN

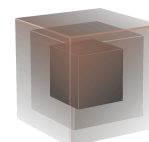
Database absolute maxima

Specification	Limit	Comments
Database size	Only limited by total disk space available	Due to total capacity of currently available drives (multiple terabyte drive arrays can now be configured quite easily) Lava does not support drive spanning.
Table rows	Maximum of 2,000,000,000 per table	Note that even with a moderate row size (say 250 bytes) such a table would occupy half a terabyte of disk space
Row size	Maximum of 15,000 bytes per table	Note that this excludes variable column sizes - limit only applies to the sum of the fixed length columns
Columns per table	limited only by maximum row size	Practically a limit of approximately 1,500 columns or so would result from the total row size constraint
Variable length columns	Any number per table row	
Variable length size	Maximum of 100 MB per entry	This limit is strictly per variable column item - does not apply to a table row
Blob size	Maximum of 2 GB per blob	



Supported Operating Systems

Operating System	Verification level	Stability and Performance	Comments
XP	Moderate verification	Reasonable stability, client performance very good with adequate memory	Only viable for clients with moderate distribution size
Vista	Moderate verification	Good stability, client performance very good with adequate memory. Server performance good.	May be used for clients or small server systems with small client load
Server 2003 R2	Thorough verification on 32-bit, client and server. Moderate verification on 64-bit	Good server stability except with high client loads - socket layer collapses. Performance good.	Only viable as a server for small numbers of clients and moderately sized databases due to limitations in the network layer
Server 2008	Rigorous verification both 32 and 64-bit, client and server	Very good server stability. Performance excellent.	Very suitable for servers with large databases or large numbers of clients
Windows 7	Rigorous verification, client and server	Very good client and server stability. Client performance excellent. Server performance very good.	Very suitable for clients, including large distributions. Suitable for small to medium-sized databases in a server role, where client count is not large.
Server 2008 Hyper-V	Guest operating system : Server 2008 or Server 2008 R2. Rigorous verification, client and server	Very good stability - some performance degradation as compared with R2. Performance very good provided hardware adequate.	Suitable for large databases and large numbers of clients provided underlying hardware is adequate for total VM load
Server 2008 R2	Rigorous verification, client and server	Very good stability under all conditions. Performance excellent.	Highly suitable for large through huge installations



Minimal Client Configuration

The following specification (typically for a notebook or very small workstation) would be the minimal configuration able to support a distributed client.

Operating system : Vista or XP, Windows 7 preferred
Memory : 1 Gb
Free disk space : As per distributed size
Processor : 2 GHz single core
Network : 250 Kbit/s sustained

Server Configuration

Suitable server configuration will depend on database size and concurrent client load. For a moderately large database (20Gb or more) with reasonably high client load (50 or more distributed clients) and a moderately demanding application requirement (200 effective transactions or more per second) the following configuration would provide a good starting point.

Operating system : At least Server 2008, Server 2008 R2 preferred
Memory : DDR3, at least 4 Gb. 8Gb preferred
Disk subsystem : 15,000 rpm drive or drive array - either 3Gb/s SATA or SAS
Processor : Quad-core, at least 3 GHz - 3.4 GHz or faster preferred
Network : Suitable to client response requirement

For further information, see the Lava Distributed SQL Website www.lava-sql.com

Tesseract Digital Information (Pty) Ltd

P O Box 9, Irene, Pretoria 0062, South Africa

Fax : +27 997 1082

e-mail : support@qubik.net

Website : www.qubik.net