FirstSQL/J ORDBMS, Ver. 2.0

The advantages of the relational database model, SQL, object-orientation, and the Java programming language are well known. FirstSQL, Inc. brings these powerful technologies together to provide a first class Java database development tool for the most important computing platforms anywhere. FirstSQL/J™ ORDBMS is written in 100% Java and delivers the power of objects integrated with a relational database and Intermediate Level SQL92. It is not only written in Java, this powerful language provides the basis for object capabilities, ensuring the widest availability of services and flexibility for database application developers.

FirstSQL/J Ver. 2.0 database technology combines yet another powerful technology that increases its performance an order of magnitude. With the optional In-memory architecture, FirstSQL/J becomes the fastest Java database available anywhere. From low power handheld computing devices to server platforms, the database performance gains provided with In-memory architecture are powerful.

Availability of Main Memory

Several important factors are driving the use of In-memory database architecture for a wide range of traditional database management and communication processing applications. With the growth of e-Business and internet and wireless infrastructures, the need for faster database access is becoming more important. To meet the increased need for performance, random access memory is becoming cheaper and form factors are shrinking. Even the amount of ram that can be addressed had has taken a huge leap forward.

64-bit Chip Architecture

Offering significant performance increases for both Java applications and In-memory databases, 64-bit microprocessors support direct access to potentially up to 16 terabytes of addressable random access memory. FirstSQL/J IMDB Server is optimized for the Intel® Itanium™ IA-64 platform making it the only Java database available to process potentially multi-terabyte size databases directly in memory. In addition to a large number of registers and hardware techniques such as prediction, parallelism, prefetch, and data speculation, Itanium provides significantly faster execution and optimization of both In-memory databases and Java applications.

FirstSQL/J In-memory Mode

FirstSQL/J Ver. 2.0 provides an optional “memory mode” of processing designed to execute entirely in memory with direct access to data and processes residing in ram. This represents a substantial performance increase over disk-based systems, even when they are entirely cached in ram. Additional features provided with the In-memory option include logging of persistent data to disk and live standby systems, to support replication for High Availability configurations and Hot On-line backup.

FirstSQL/J in memory mode is used, developed, and administered in the same way as the disk-based Enterprise Server version. The Administrative and monitoring tools have not changed and operate in the same fashion to allow remote, local, and zero administration.

Except for increased performance, the only difference you will notice is the creation of the log files and persistent .tab and .ndx files in the Server/db directory when starting and stopping the database.

The memory mode uses the FirstSQL/J standard table format for storage on disk. In memory mode, the engine keeps data and indexes in memory, using memory structures for access.

Processing is as follows:

1. At startup, the engine reads all catalog tables from disk into memory structures.
2. During processing – when a table is first referenced by a SQL statement, it is loaded into memory with associated indexes. All access is in memory from that point on.
3. Any changes to tables (in memory) are logged in a roll forward journal and written to disk.
4. With a controlled shutdown, the engine rewrites all changed tables & indexes from memory back to disk in FirstSQL/J file format.
5. If the engine is aborted without updating database data on disk, subsequent recovery uses the journal to roll forward all changes to the files (this file update at recovery-time occurs even when the ORDBMS is started in main memory mode).

The replication facility also uses a roll forward journal. In this case, the journal is sent via sockets to another (standby) live FirstSQL/J server. The standby server reads the journal stream and dynamically applies the changes to its private copy of the database. At termination, both databases will contain the same data. The replicated standby provides read-only access and can be used to balance query processing loads from clients and applications.

Comparison: Memory-Resident Data Access Vs. Disk-Based Data Access

The difference between a disk-based database and IMDB architecture is fundamentally different. Even when a disk-based DBMS has enough ram to cache all its data, it still processes it as though the data will be stored in disk format, for instance, in pages. Caching only allows it to avoid the overhead of disk I/O until it must write to disk. Memory resident database processing is significantly more efficient and quicker as a result because:

1. Disk-based DBMSs are designed and optimized to format data for disk storage and require conversion to/from the internal format the DBMS uses in operating on the data. There is some expense (in terms of CPU processing) in conversions.

2. In random processing, only a part of a page may be actually used (a row within the page) and needs conversion. The remainder can be left unconverted.

3. The page format contains additional 'positional' information which would have to be recreated otherwise and is needed for access paths.

There are several other, more technical reasons. However, the reason for caching data is to avoid disk access and take advantage of a ram access.

It is possible for DBMS caching of disk pages to be counterproductive. The operating system normally provides its own caching of disk information. This produces conflicts between the two caching algorithm leading to what is known as 'dueling caches', reducing the effectiveness of both caches.

However, when you don't have to worry about the format of data on disk, paging/caching/locating/buffering and conversion overhead go away. Access to a database in-memory is radically different, direct and more efficient. A database table can be stored as an array of records (rows) with each record being an array of column values. The column values are stored in the most convenient format for internal query plan execution and no conversion is required.

Note: A disk-based DBMS might use a high level cache mechanism that stored table records in the form described above, but it is unlikely. The elimination of conversion (and potentially access) expense would be offset by the extra expense of caching and related expenses. There also is no cost effective method of doing high level caching of standard access paths (indexing, hashing).

A disk based index/hash access path is structured to reduce disk accesses. This is not a consideration for memory-resident DBMSs, so completely different forms of indexing/hashing can be used. Compared to access using cached disk formatted pages, these techniques are extremely efficient with no caching mechanisms required at all.

Part of the art of being a database administrator is to organize data physically on the disk such that information that is often needed together is stored within the same disk block or page. Of course, the complexity of the DBMS doesn’t stop there. Indexes are required to provide the information needed to bring the correct block of information into the cache, but these indexes themselves reside on disk and must be cached for efficiency. Query Optimizers are needed to remove the complexity from the application designer and allow the DBMS to make its own informed decisions as to the best way to retrieve any particular piece of information. Transaction management must ensure that each separate user of an application sees data in a consistent state irrespective of updates by other users taking place concurrently. Separate transaction logs must be written to disk to provide a recovery mechanism in the event of a failure. DBMS logic must determine where the needed data is, on disk or in cache.

In short, the whole disk-based architecture has become very complex. Most of the processing that goes on in a server has nothing to do with executing business logic. Instead the processor has to manage and execute all of the data movement functions from disk to database cache, from database cache to application cache and from the application cache to the processor registers – and then all the way back again. In a distributed environment this is further complicated by the need to manage data movement between machines across various types of network.

Because so much memory is now available, the simplest means of exploiting this is to increase the size of the database cache to further reduce the amount of disk I/O needed. In many cases entire tables or even the entire database can become cache-resident. But because this cannot be taken for granted the DBMS still carries much of the processing overhead. Records are still blocked up as if they were to be stored on disk. Pages are created and converted. Indexes are still provided and must be maintained in buffers. Log records are still written. Data is still transferred to application local caches. The DBMS still has to work out whether the data being requested is currently cached or has to be retrieved from disk.

FirstSQL/J IMDB expects the entire database to reside in main memory and, as a result, huge simplifications are taken. These simplifications translate into greatly enhanced performance, simpler database design, and a more robust environment.

Conclusion

FirstSQL's innovative use of Java and In-memory architecture provide a performance potential that is beyond the scope of mainstream products. The end result is a very competent SQL ORDDMS product that is ready for mission-critical work anywhere database applications have high performance requirements.

The power and performance of In-memory database architecture is proven. Typical applications include:

- **Computationally intensive** – Transactional (handheld devices and servers), Analytic Modeling, Data Analysis, XML Manipulation, Personalization, Business Rules, Work Flow, Supply Chain Management, Learning Management Systems.
- **Heavy user load** - CRM, Call Management, e-Business, etc.
- **Strict response or throughput requirements** – Real-time Fault Tolerant Systems, Soft Switches, Intelligent Routers, Web Application and Mobility Servers, Process Control, Monitoring, SCADA, etc.

Some industry analysts believe the only significant advancement in database technology currently is the emergence of main memory database products and they predict in the near future a majority of database processing will utilize In-memory technology. FirstSQL/J is the only commercially available 100% Java IMDB and, as the first and most sophisticated 100% Java IMDB on the market, it can be positioned as the leader now and for years to come.