The Emergence of the Object-SQL Database

A White Paper

Matisse Software Inc.

Table of Contents

1. The Shift to Object Development

The most important trend in software development in the past decade has been the shift to object development methods and tools, thanks in part to the emergence and adoption of Java. Developers have found that object-based development provides huge advantages in the rapid development of applications that model real-world problems. Object concepts such as inheritence, encapsulation, one-to-many relationships, and polymorphism provide a fast and natural way for developers to directly represent complex data and its inter-relationships.

With the acceptance of Java, the momentum for object-based development has accelerated with no signs of slowing. Recently, a report by IDC indicated that there are over 2.6 million professional developers worldwide using C/C++ as their primary language and over a million professional developers worldwide using Java as their primary language. This means over 37% of all developers worldwide are using these languages and the numbers continue to grow.

In order to capitalize on the shift to object-development, database vendors have tried to create database management systems that provide a good foundation for object applications, but have largely failed to meet the expectations of object developers, IT organizations, and corporate management. These databases fit into one of two categories: object databases (ODBMS) and relational databases (RDBMS) with object-relational mapping.

Object databases were a close fit, but...

Realizing the advantages of programming with objects, several database vendors introduced first generation object databases in the mid-90s. While directly designed and intended for objects, and thereby better suited for object development, the early object databases had numerous deficiencies that hurt ODBMSs chances of being deemed worthy of serious deployment.

Ironically, one of the deficiencies (persistent objects) of the first generation ODBMS systems wasn't initially seen as a deficiency, but instead hailed as the ideal solution for object developers. The first generation ODBMS bypassed the need for O-R (object-relational) mapping by operating as an extension of the object programming language. This allowed developers to create and store persistent objects in the database simply by declaring them in application code, without needing to use a separate database interface or language. This is why object databases were also referred to as persistent object stores.

Most of the ODBMS vendors used this transparent persistence because objects could be mapped directly to the database, but persistent objects came at a heavy price. The principal reasons that ODBMSs failed to become the de-facto foundation for object application development included the uncontrolled way with which developers could specify persistent objects, the inability of the programmer to specify which objects persist and the lack of rigor in database development. While the prototypes appeared quickly and worked well, production systems often suffered once the applications were deployed.

Another deficiency of these early ODBMSs was the lack of SQL support (or in the way SQL was implemented when it was provided). The ODBMSs without SQL failed to capitalize on the SQL expertise within most companies. This forced companies to absorb the additional time and cost of training developers on proprietary access languages. Those ODBMSs that did implement SQL on the client-side had network bandwidth problems that limited utility and created performance problems.

With client-side SQL implementation, almost all processing associated with a set of SQL statements occurred on the client. All classes and subsequent contents referenced by the SELECT statement had to be passed over the network from the server to the client, even if the result of the query was only one or two items of information. This created large volumes of network traffic for each SQL request; further impacting performance because extensive portions of the database needed to be locked against other write attempts. Contrast this with a server-side SQL implementation where the SQL statements are passed from the client to the server. The server processes the query and passes only filtered results to the client; no unnecessary information is sent across the network and the performance impact from locks is minimized. Overall, server-side SQL is considered the most effective implementation.

Early ODBMSs failed to provide the optimal database platform for object development. The poor performance, lack of SQL support, inadequate network bandwidth, and the proliferation of persistent objects caused developers to abandon the early object databases. Intent on providing another option, RDBMS vendors relabeled their products object-relational databases and added capabilities such as O-R mapping in order to support objects. These also failed to become the ideal database platform for object developers.

Relational databases weren't quite ready

In order to represent an object design in a relational database, all classes must be mapped to relational tables, a process known as object-relational mapping. With O-R mapping, each object must be represented by two-dimensional tables; there is no way to directly represent or store object concepts, such as inheritance or relationships between multiple objects.

O-R mapping also created a substantial performance problem because related data had to be split (deconstructed) into multiple relational tables. Each time objects were stored, they had to be deconstructed into multiple tables, and reconstructed when retrieved – adding as much as 30-40% to the code required for an application. Object reconstruction usually involved combining data from multiple tables using complex-relational joins and the complexity of these joins was extremely processing-intensive and ultimately slowed systems to a crawl.

While seeming to provide a solution that circumnavigated the object to relational problem, the O-R mapping process turned out to be notoriously complex and time-consuming. In an attempt to avoid the development overhead, object-relational database suppliers and third parties introduced object-relational mapping tools, a layer of software that translates between application objects and relational database tables. However, the same process of converting between objects and tables still occurred within the database, and so did attendant performance problems.

As with the early ODBMSs, the stopgap measures provided by the relational vendors were unwieldy, added to the overall cost of deploying and maintaining applications, and failed yet again to provide the optimal database platform for object developers. Individually, RDBMSs and ODBMSs could not meet the needs of object developers. Some hybrid of the two was necessary to provide developers and IT with the exact capabilities required to facilitate their object application development and management.

The best of both worlds

The solution was an object-SQL database that combined the ease of object development and the ubiquity of SQL – the ultimate fusion of relational and object technology. Designed from the ground up to handle objects and to enable the rapid development of complex object applications that model and solve real world problems.

Uniquely positioned to provide the optimal foundation for object development and deployment, an object-SQL database would succeed where previous generations of databases failed. Developers would be able to create applications and services faster, cheaper and more effectively while simultaneously allowing IT to cost-effectively deploy and administer these applications. No such database existed, though…until Matisse.

As an object-SQL database Matisse provides support for object-to-object mapping, ANSI SQL, standards, UML, while offering high performance and zero administration. Matisse is the only database that has been able to bridge the gap between object developers and object management. Matisse has taken the best capabilities of object and relational databases and created The Object Developer's Database.

2. Matisse: The Object Developer's Database

Object development has created a need for a DBMS that provides the direct object management and UML support of an object database, yet ofers the benefits that have traditionally been delivered by relational databases: SQL access, performance, scalability and standards compliance. Matisse is the first database to truly bridge the object development and management gap and provide the capabilities that are needed by object developers and IT to overcome the deficiencies of early databases. Matisse was built from the ground up to be a high-performance database that integrates objects with SQL.

Direct object management

Matisse supports the full object model, including features like classes, inheritance and encapsulation. In terms of the management of objects, Matisse has brought together the best aspects of object and relational DBMSs by eliminating the need for object-relational mapping and allowing objects to be directly accessed. Removing the dependence upon object-relational mapping allows Matisse to speed object-based development by enabling developers to map objects directly to the database. This also means that since there is no dependence on object-relational mapping there are also no intervening layers of software, no translation between objects and relational tables, no joins, and no performance penalties associated with Matisse.

UML support

Matisse had traditionally used ODL (an object definition language championed by the Object Data Management Group) as the definition language for objects. ODL is now being replaced by the Unified Modeling Language (UML) as the favored language for modeling and defining objects. In order to ensure that object developers can utilize the benefits of UML, Matisse takes the UML output directly from tools like Rational Rose and converts it to Matisse database schema (bypassing ODL).

SQL access

Matisse- managed data can be defined, read, updated or deleted using industry-standard SQL. So developers can leverage their existing expertise with a variety of object languages including SQL (the standard for data access). Matisse provides a full, server-based, native SQL implementation that complies with the SQL 2 standard with object extensions and includes support for stored procedures and triggers.

Performance

The Matisse versioning engine assures data consistency while enabling applications to read data while being updated (or loaded). Matisse can out-perform relational databases in situations where large volumes of data are being loaded or updated at the same time as other applications are reading data. This means that Matisse can support transaction processing and data analysis applications simultaneously on a single system.

When an object is updated, the versioning engine creates a new version of the object in a new location, instead of updating the old version in place. Applications can read the previous version of the object while the new version is being created, and because all references are left intact, see a consistent view of the database at all times. Once the new version has been created, and the transaction is complete, subsequent read requests are directed to the new version. If the transaction fails, requests simply continue to be directed to the previous version; there is no need to roll back the database in order to achieve a consistent state.

As an additional benefit, applications can examine historical data at any time by exploiting the versioning engine architecture. The versioning engine provides access to previous versions of objects, enabling applications to explicitly access data from previous periods, or identify objects that have been changed, deleted or added. A financial analysis application could use this feature to track changes in a company's stock price over time, for instance.

Relational databases typically suffer performance problems when data is retrieved from multiple tables, due to the processing-intensive joins required to match related data between tables. But the performance of SQL-based applications is often substantially better with Matisse, than on relational databases, because Matisse's architecture eliminates the need for processing-intensive complex joins when accessing related data.

This is because Matisse uses pre-computed relationships (fast joins) to rapidly locate and access related data from multiple classes/tables. Inter-object references make fast joins possible with Matisse.

With Matisse, when a class is defined, its relationships with other classes are also defined. This results in a semantic network of relationships throughout the database. These relationships allow Matisse to directly access related data, a process that is transparent to the requesting application. The only visible result is greatly improved performance.

Scalability

Matisse is highly scalable because its server is implemented on top of kernel threads, and thus scales linearly on SMP (symmetric multi processing) architectures as new CPUs are added. Matisse APIs are thread-safe to capitalize on the multi-threading capabilities of the most recent operating systems. Matisse takes full advantage of its optimized cache and versioning architecture in order to maximize database performance with existing hardware.

Stored procedures, triggers and referential integrity

Just as would be expected with any of the major relational databases, Matisse also provides SQL-stored procedures, triggers and referential integrity. Stored procedures are sequences of declarative and procedural SQL statements that execute on the server, and have become widely used to increase coding efficiency and performance. A stored procedure on the server can be used by multiple applications.

Triggers are stored procedures whose execution is initiated when a specific database operation is performed, or when an operation results in a predefined database condition. Unlike regular stored procedures, they cannot be called directly from an application. Triggers are widely used to implement constraints on database operations and maintain database integrity.

Maintaining referential integrity with relational databases creates additional work for developers. When data is deleted or moved, other tables that refer to the data must also be updated. For instance, if a customer is removed, the customer's orders must also be removed. With Matisse, maintenance of referential integrity is greatly simplified through relationships between objects. Matisse relationships are bi-directional. This ensures Matisse is automatically aware of referential integrity and if an object is deleted, all references to it are updated.

Standards compliance

Matisse object and SQL interfaces comply with established standards to ensure software compatibility and portability. ODL, Java and C++ access are ODMG and UML compliant. Matisse supports a full set of object features accessible through these standard languages including multiple inheritance, encapsulation, and polymorphism. Matisse also supports the JDBC/ODBC standard for SQL access from applications in Java and other languages. Matisse supports the EJB and J2EE standards.

Even if an organization's development is primarily focused on Java or other object languages, there is often a need to support existing SQL-based software. Products such as PowerBuilder and Crystal Reports provide a fast way to generate reports and applications. Many organizations use data-analysis applications that rely on SQL to extract database information for business decisions. Matisse’s standards-based SQL implementation means these applications can be used to access Matisse databases. Because the Matisse SQL implementation is native and server-based, these applications typically operate as effectively with Matisse as they do on relational databases.

3. Reducing the IT Burden

Object-to-object mapping, SQL, standards compliance, performance and scalability–these features represent the best-of-both-database-worlds and help eliminate the development complexity and performance road-blocks of previous databases. But the benefits of Matisse are more far-reaching than the developer community...IT organizations also face many issues with legacy database systems that make successful deployment and maintenance of object applications difficult.

Matisse was designed to require minimum administration once in operation; in many cases it will operate with zero administration, making the database ideal for embedded or remote systems where administrator intervention is either difficult or impossible. Designed to be always running, many of the tasks required to administer the leading relational databases are simply not required with Matisse. Simply setup, configure, and monitor.

In cases where ongoing administration is desired, all database management operations can be performed while the system is online and under load. Some of the zero-administration capabilities of Matisse include:

No log files

Traditional DBMS products require administrators to continuously monitor the growth of the journal or transaction log file (which contains a replica of every update performed in the system) and to run frequent backups to be able to truncate and reorganize the log. Matisse does not require a transaction log. The Matisse versioning engine guarantees database recovery while avoiding the overhead of transaction log operations and administration. This relieves organizations of the considerable burden of having skilled administrators define and manage log files.

Automatic tuning

Matisse automatically and continuously optimizes disk use without administrator intervention. As a new disk is made available, Matisse integrates it into its management along with all other available disks to ensure the best performance. In addition, Matisse automatically monitors the utilization of all disk resources and load-balances among them since each time the Matisse versioning engine writes a new version of an object; it selects the least-used disk as its location.

With Matisse, there is no need for administrators to allocate certain data to specific disk resources, or to reorganize data as disks fill up. Disk administration consists of setting parameters such as cache size and defining physical disks. Matisse's automatic tuning capability leads to a performance-optimized distribution of objects across all available disks.

Online parallel backup

Backups can be generated with Matisse while online and programmatically without manual intervention and without interrupting operational applications, even when the system is operating under a full workload. This is especially important for unattended or remote applications where a full-time DBA may not be available.

No scheduled downtime

Matisse backups require no blocking or downtime. The database schema can be changed while the database is online, so that even frequent design changes do not cause operational disruption.

Matisse's unique design allows nearly all-administrative functions to be performed while the system is up and under load and no matter whether data is being accessed in read or write mode. These functions include: managing users, updating the database configuration online by adding or deleting disks, and doing full or incremental backups.

24x365 reliability

Inherently reliable, Matisse also provides automatic disk mirroring and replication software mechanisms to provide fault tolerance and uninterrupted service in the event of disk crashes or system failures. Automatic disk mirroring also allows read access scalability without the addition of specialized hardware or software. The system automatically reconfigures itself, so no system administration is required to implement this capability in the case of a primary system failure.

Matisse technology has been used to control nuclear power plants and complex chemical manufacturing processes for over a decade. In these situations, the database is required to be available 24x365 with predictable response times and uninterrupted service even while configuration changes are being implemented.

Dynamic schema evolution

The application schema is language independent and stored in the database. Matisse's dynamic schema evolution capability allows for the addition or removal of classes or properties while the system is online, ideal for development environments in which the data model changes and simulation is frequently performed.

4. The Safe Business Solution

An important, but often overlooked element of application development and deployment, has to do with the availability of the database to meet the growing and changing needs of developers, IT and customers. It's critical that a database investment future-proof applications as changes occur, such as: developers' preferred development language changes, reduced administrative staff, different deployment platforms, new branch and remote offices or stores, and the addition of more computing resources (i.e. CPUs).

It takes a flexible product to meet all these scenarios. Matisse ensures the longevity of the investment in both object development languages and in the database management system. Matisse was designed to provide heterogeneous client/server architecture for high- performance operations on complex data types. The Matisse server, a general-purpose object manager, operates as a back-end server handling a repository of persistent objects. Client applications can connect to the server through the Web, a dedicated network, or local transport.

Matisse future-proofs object application development by providing an OPEN API for language bindings. As development languages continue to evolve and developer and corporate preferences shift, it is critical that the database powering object applications be open, while conforming to standards.

Language independent

Matisse is language independent, so that data created using one language can be accessed by applications using other scripting languages or object environments. New languages can be introduced as necessary or as industry standards and developers' preferences change. Applications in different languages can be mixed within the same system. This is useful when there is a change in the preferred development language at an organization. For example, new Java components could be added to an older C++ application.

Matisse provides support for most popular programming languages: SQL, Java, C++, Perl, Python, PHP, and C. Third party bindings (e.g. Eiffel) can also be added. Developers using native programming languages to manipulate persistent data can build and deploy applications quickly and easily, as there is no language learning curve. Matisse's support of virtually any language means organizations no longer have to worry about having the appropriate skill-set on hand; existing knowledge and skills can be leveraged, freeing organizations to focus on the development of the application components.

XML ready

Matisse is also a natural database for handling XML documents since XML hierarchical elements can be mapped directly to Matisse object structures. Matisse provides an object API to manipulate XML documents, as well as an XML utility, (referred to as mt_xml) that provides automatic loading and generation of XML documents through SQL queries. This utility maps XML documents to a pre-defined schema, performs batch loading into Matisse, or exporting from Matisse, and validates the structure of the XML documents against the Matisse schema.

Multimedia ready

The performance and consistency advantages of the Matisse database apply to multimedia information as well to other types of data. Streaming media can be managed within the database itself, rather than as separate multimedia files. Matisse allows applications to jump forward, backward, pause and restart at any point in a media stream. Because Matisse enables applications to update on disk only the portion of the data that has been changed, rather that rewriting the entire data set, performance when updating large volumes of multimedia information is enhanced.

Matisse includes powerful features for text analytics. Using Matisse's entry-point mechanism, text can be indexed in order to support rapid searches. Indexes can be built based on the occurrence of various text strings, or full-text indexing can be applied. Different strings can be ranked so that searches present data of the greatest value to the user.

5. Conclusion

Matisse has an extensive history with hundreds of man-years of development behind it and has been powering industrial and other mission-critical systems at over 150 sites in Europe for over a decade.

Matisse provides the following advantages for developers and IT professionals:

* The Matisse versioning engine ensures data consistency, recovery from transaction failure, and access to historical data without the need for log files; so there is no need for time-consuming log file administration.
* Matisse excels at "reading while writing" and can simultaneously load/update large amounts of data while supporting high query levels.
* The Matisse architecture automatically load-balances among available disk resources and in many applications Matisse can operate with zero administration, making the database suitable for distributed remote or embedded systems.
* Matisse is extremely flexible. Changes can be made to the schema at any time, even while the database is online.
* Matisse contains extensive features for storing, manipulating and indexing diverse data types such as streaming media and text.
* Matisse's reliability has been well proven being that the Matisse has been extensively used for 24x365 operation in applications such as nuclear fuel process control.
* Matisse is the fusion of relational and object technology. Matisse was designed from the ground up to handle objects and to enable the rapid development of object applications that model and solve real world problems.
* Matisse's support for object-object direct mapping, ANSI SQL, standards, the Unified Modeling Language (UML), high performance and zero administration means that Matisse has been able to cross the gap between object developers and object management.
* Matisse is uniquely positioned to provide the optimal foundation for object development and deployment. It allows developers and IT to create, deploy and administer applications and services faster, cheaper and more effectively.

With the success of Matisse in Europe, and the success of the Matisse-powered solutions, object developers familiar with previous generations of object and object-relational databases have found Matisse to be a fully SQL-compliant object database that exceeds the performance and scalability of relational databases.

Matisse Software Inc.
433 Airport Blvd, Suite 421
Burlingame, CA 94010
650-548-2581