A Comparison of University POSTGRES and Miró

Michael A. Olson[1]

Miró Systems Inc.
2000 Powell St., Suite 1405
Emeryville, CA 94706
510-652-8000
e-mail contact: sales@miro.com

1. Introduction

In this paper we discuss the history of the POSTGRES project, and the differences between the POSTGRES system and the commercial database system produced by Miró Systems Inc.

2. History

In 1985, Dr. Michael Stonebraker and his group of researchers at the University of California at Berkeley were preparing to retire the public-domain database system called "University INGRES." Dr. Stonebraker's group had begun development on INGRES in the early 1970s. By 1985, the group had spun off a successful company[2] to commercialize the research software, and believed that they had addressed most of the interesting research issues that had confronted them ten years earlier.

In the aftermath of the INGRES project, Dr. Stonebraker was looking for a new direction in which to push database systems research. He believed that more powerful systems, combining the best features of both object-oriented programming systems and relational databases, would allow users to solve large, complicated problems more easily than any other technology then available. To test this conviction, he organized a research effort that was the successor to the INGRES project. This effort, called the POSTGRES project, would design and implement a new kind of database system.

From its beginning in late 1985, POSTGRES was designed to do things that ordinary relational systems could not. Most importantly, it was an *extensible* system. Since the uses to which POSTGRES would be put were impossible to predict, the system was designed to be easy

[1] Mike Olson received his Bachelors' and Masters of Science degrees from UC Berkeley in 1991 and 1992, respectively. He joined the POSTGRES research project under Dr. Michael Stonebraker at Berkeley in 1988. During his time with the POSTGRES group, Olson worked extensively on the database system's internals and designed and implemented large parts of the storage system. In 1993, he left the University to join Miró.

[2] Then called Relational Technology, Inc., later INGRES Corp., which was subsequently acquired by ASK Inc.

to extend. As new applications appeared, support for them could be added to the database system. Users could define new data types, functions, and operators to solve new problems. Object-oriented techniques like inheritance, encapsulation, and abstraction made it easy to make small modifications to existing code to get new behavior. Sophisticated users could even define new ways to store and fetch data structures efficiently from various storage devices, allowing POSTGRES to find data of interest quickly via user-defined indices.

POSTGRES provided a number of other novel features. Its "no-overwrite" storage manager allowed users to travel in time over their database, looking not only at its current state, but at the data it stored at any time in its history. A sophisticated rules system allowed POSTGRES to detect accesses to particular data values stored in the database, and to take user-specified actions in response to them. The internal "device manager" interface made it easy to add support for new kinds of storage devices, like optical disk and tape jukeboxes, so that user data could be spread across a number of very large data stores.

The fundamental goal of the POSTGRES project was to produce a database system suited for the computer systems and applications of the 1990s and beyond. As a research system, POSTGRES was remarkably successful. The software developed at UC Berkeley was first released to the public in 1988. By 1992, it was in use at hundreds of installations around the world for applications as diverse as jet engine testing, land-use planning, document storage and retrieval, satellite image capture, and climate modelling.

By the early 1990s, it was becoming clear that the original research mission of the Berkeley group had been largely satisfied. POSTGRES had been widely disseminated and was being used regularly for a large variety of new applications. Releases of the system were more often to fix bugs, tune performance, and enhance features than to provide users with fundamentally new services. The amount of research undertaken by the group was dwindling, and the time spent on software support and maintenance was growing.

In response to this situation, Miró Systems Inc. was formed in 1992 to commercialize the POSTGRES research software. The company was formed to extend, sell, and support the database system, leaving the University group free to move on to the next research project.

The founders of Miró included several senior members of the POSTGRES group, as well as veterans of relational database vendors. The combination gave the company the right mix of expertise — it could design and build database systems for the future, and it had experience in selling and supporting them to real-world customers who cared about performance, reliability, and security.

Since Miró was founded, it has made many improvements in the POSTGRES software. The most visible difference between the two systems is in their names; the name "postgres" has been entirely supplanted by "miro." Naturally, the differences between the commercial and University systems are more substantial than mere naming. Those differences are explored in detail below.

3. Query Language Support

Since its inception, POSTGRES has supported the POSTQUEL query language. POSTQUEL is a derivative of Quel, which was invented for University INGRES in the early 1970s. From the point of view of a group of researchers, POSTQUEL has the advantage that it is not subject to standardization. The University group had complete control over the language's syntax and semantics, and could change its definition at will to experiment with new features.

Miró, on the other hand, because of market demand and to adhere to key industry standards, supports SQL. SQL was originally derived from work done on the System R project at IBM. It has since undergone standardization by ANSI and ISO, and has become the dominant query language for database systems. Miró implements an ANSI-compatible version of SQL-89.

Miró's support for SQL has several practical implications for users. First, the language's syntax and semantics are well-documented. This means that it is easier to predict how the database system will interpret a query than was the case under POSTQUEL. Second, applications originally developed to use other SQL database systems are easier to convert to use Miró.

Users of POSTGRES who want to convert their applications to use Miró must reformulate their POSTQUEL queries in SQL. In general, this is straight-forward. Miró has also standardized the names of the tables storing the data dictionary, so POSTGRES users must adopt the new names when issuing queries against the system catalogs.

4. Library Interface

POSTGRES supported programmatic access to data stored in databases via the *libpq* library. This library allowed the user to write a program that sent POSTQUEL queries to the database server process and received data back.

Miró has changed the library interface. The new library, *libmiro*, supports a slightly different interface than did *libpq*. The major difference visible to users is that all of the *libpq* procedure names have been changed to names that begin with the characters "mi." In addition, error handling by the library has been cleaned up, and the protocol by which data is moved between the client and server processes has been revised.

One other important improvement made by Miró is that the *libmiro* interfaces are supported for dynamically-loaded user code. Both POSTGRES and Miró allow the user to write procedures that are declared to the database system, and that will be dynamically loaded and executed on demand. Under POSTGRES, the programmatic interface for these dynamically-loaded functions was completely different from that for client routines. Under Miró, the user may develop, test, and debug a procedure by running it in the client program, and may then migrate it to the server by declaring it to be dynamically loaded. No changes to the procedure are required. A uniform interface makes it easier for users to develop and test their software privately, and to release it for general use when it works correctly. This has an additional benefit that it allows the application developer to change where a given function is executed (on the client or the server) and to observe the impact on performance, without having to modify the function itself.

5. Support for User-Defined Functions

POSTGRES demonstrated that allowing users to extend the database system by writing dynamically-loaded functions was a simple and powerful way to support new applications. Under POSTGRES, however, developing dynamically-loaded functions was difficult. The internal interfaces were largely undocumented, so programmers were required to read the source code for the database server to figure out what routines they should call. In addition, debugging dynamically-loaded code was very hard. The routine was loaded into a process owned by the database administrator, so ordinary users could not stop the process while it was running and examine its state. Most debuggers were unable to deal with processes that did dynamic loading, so even if the user had permission to control the process, he lacked the tools to do so.

Miró has solved both of these problems. The interfaces that are available to programmers have been cleaned up and are completely documented. In addition, since the user's function

can now be run either in the client or the server, debugging is simple. The user writes his function and links it into the client program, which he owns. He can run the program using any debugger, starting and stopping it at will and examining its state. When he is satisfied that the function works correctly, the user simply recompiles his function and declares it to be dynamically loaded.

Conversion from POSTGRES to Miró requires less than no work, in this case. Under POSTGRES, most users who write dynamically-loaded code have evolved complicated strategies, usually involving *printf*, for debugging their software. Those strategies can be thrown away, and standard tools can replace them.

6. Error and Exception Handling

In addition to cleaning up error handling in the client library, Miró has cleaned up and standardized the way that errors are handled in the server. SQL error codes are correctly maintained, and exceptions are detected and repaired when possible. POSTGRES took an *ad hoc* approach to error management in the server, and had no consistent strategy for recovering from problems.

7. Security

Since it supports SQL, Miró provides the full set of SQL security controls, including `grant` and `revoke` commands for controlling user access to sensitive data, as well as providing the rules system introduced with POSTGRES. POSTGRES's support for security was limited to allowing users to declare rules that allowed or disallowed access to data stored in the database, and there were easy ways to circumvent the rule system and to overcome the security controls.

One important difference between POSTGRES and Miró is in how access controls are configured by default. In POSTGRES, any user can examine any system table. Miró requires users to be granted explicit permission to read system tables. This may mean that queries that work under POSTGRES will no longer work by default under Miró. If desired, the database administrator can grant the user permission to run the query.

POSTGRES may be configured to use the Kerberos library from Massachusetts Institute of Technology for user authentication and session management. By default, Kerberos is disabled. Since not all Miró users are likely to have Kerberos installed, Miró uses a more conventional password verification scheme to control access to the database system.

8. The Type System

POSTGRES was designed to provide an object-oriented interface to the data that it manages. However, due to some problems in the early implementation of the system, the type system did not support inheritance for built-in system types. There were, in effect, two different type systems in the University system. The abstract data type system, which was managed by dynamically loaded user-defined functions, did not support inheritance. The class system, built on top of tables stored in the database, did support inheritance. The result was that users had to choose between inheritance and simplicity (that is, not writing user-defined code) or fast execution and no inheritance.

Miró has remedied this situation. Both the abstract data type system and the class system support inheritance. Miró has extended POSTGRES' notion of inheritable types to include composite types (single values that are composed of records from the database), sets (collections of records or other data values), and arrays. Miró provides a richer set of built-in data types and

allows different types to be combined and compared more easily than did POSTGRES. Both POSTGRES and Miró support *overloading,* or different implementations of the same function name, depending on the number and types of arguments to the function.

Finally, Miró supports *late function binding*. This means that the function that is used to satisfy a user request is selected only when all of the arguments to the function are available. Since polymorphism means that different functions operating on different types may have the same name, it is critical to choose the correct function to run in response to a user query. This is only possible if the types of all arguments are known before the function is selected. POSTGRES selects the function to run once, when the query is parsed. This may lead to incorrect results, since the query parser may not have complete type information for the arguments to the function.

9. Support for Large Objects

Both POSTGRES and Miró support large objects as first-class values. This means that large data objects such as images, arrays, and maps may be stored in the database. These values are typed, may be passed to procedures, and operators may be defined that manipulate them. Conventional relational systems typically offer binary large objects, or "blobs," which are untyped, cannot be manipulated by user-defined functions or operators, and are much less interesting than those supported by Miró.

POSTGRES supports several different storage options for large objects, which allow the user to trade off security against services and speed of access. Miró has chosen to simplify the system by providing a single large object implementation. Like the more sophisticated POSTGRES large object implementations, Miró allows time travel on large objects. In addition, the access routines have been highly tuned to give good performance on reads and writes. Miró's large object implementation provides the same security controls for big data values as are provided for small ones. Users may be `grant`ed or `revoke`d access to the columns containing particular large objects.

POSTGRES used its large object implementations to provide a file system built on top of the database system. Users can load large objects into the database system, and can then mount the database system as a foreign file system and use standard programs on the large object data stored there. Miró has chosen not to provide this service. It was seldom exercised by POSTGRES users, and the protections supported by the file system were incompatible with the security controls that Miró provides for all of the data it manages.

10. Support for Multimedia Data

POSTGRES was intended from the beginning to support multimedia data management. Its support for large object storage, along with its extensibility, allowed users to define new types to manage images and to store them in the database.

Miró has improved support for multimedia data management in several ways. First, the faster large object implementation and some new interfaces allow multimedia data to be stored and fetched efficiently from various storage devices. Second, the improved type system makes defining and using multimedia objects much simpler. Finally, the Miró query optimizer allows the user to request *guaranteed delivery* of multimedia data.

Guaranteed delivery means that Miró agrees to deliver data to the client at a particular rate. If the rate is too fast, the system will reply that it is unable to satisfy the user's request. On the other hand, if it agrees to meet the request, Miró guarantees that it can deliver the data as fast as

it is required. This allows users to play movies or sound tracks, for example, at uniform speeds. In order for guaranteed delivery to be fully operational on a shared system, the underlying operating system must support non-preemptable resource reservation.

11. The Help System and Document Management

In Miró, large objects are used to store documents. All of the paper documentation shipped with the system is included as large objects in the database. Miró includes tools that can locate and display documentation for users that want to review it. This system is a general-purpose "help" facility, and allows users to search for information by subject, keyword, or phrase. Links among related documents are maintained.

Miró offers a general-purpose documentation management system as an add-on to the product. This system is built using *document trees*, an indexing structure that stores text and allows fast phrase-based or keyword-based lookup. Document trees offer a number of interesting services besides simple fast searching. For example, two document trees can be compared to compute the "similarity" of two documents. Similar documents tend to cover the same subject. This allows users to ask for all documents dealing with a particular subject, in order of relevance, or to look up all documents that are like some document in which they are interested.

12. The Rule System

POSTGRES included two different subsystems for storing and executing rules on data stored in a database. The user could define rules that would detect reads or updates of any value in the database, and that would take some action in response. The action could be any set of legal POSTQUEL commands, making both of the POSTGRES rule systems more general than the triggers or stored procedures supplied by conventional relational vendors.

The two rule systems in POSTGRES were called the *instance-level* rule system and the *query rewrite* rule system. The instance-level system worked by marking particular instances of values in the database with a special tag, and noticing the tag when the value was touched. The rewrite system used a more general strategy. It examined the queries submitted by users for any accesses to data protected by a rule. If such an access was detected, the query would be rewritten to include the actions defined by the rule.

The instance-level system, while simpler to implement, had a number of problems. Most notably, some rule actions would violate the transaction semantics that POSTGRES maintained.

Miró has simplified the code by eliminating the instance-level rule system. The more general rewrite system has been retained and cleaned up. In practice, POSTGRES users migrating their applications to Miró will probably not notice the change. Under POSTGRES, the rewrite rule system was the default, so users who defined rules used the rewrite system unless they specifically asked for instance-level rules. The Miró rewrite rule system supports all of the features previously supplied in POSTGRES by the instance-level system. The only conversion effort required is to reformulate the rules in SQL.

Finally, Miró correctly checks the argument types and return values for rules, making rule processing less error-prone. POSTGRES trusted the user to define all rules correctly. If a rule expected a value of one type, and was passed some other type, it could cause the server to crash. This is no longer the case under Miró.

13. Alerters

POSTGRES was designed to support *alerters*. Alerters are special pieces of code inserted in the database that watch for some event, and notify interested users when it occurs. For example, a stockbroker might like to be notified when the price of some stock drops by more than five percent. An initial implementation of alerters was shipped in version 4.0 of POSTGRES, but the code was never reliable, and was removed from subsequent releases.

Miró supports alerters. Users can register any number of alerters, and will be notified when any of the monitored events occurs. This substantially simplifies application development. Users of POSTGRES who wanted alerters were forced to write applications that continually scanned the database, looking for events of interest. This was an expensive and unreliable solution, since it could miss events that happened between scans. Using Miró, the user need simply declare an alerter, and is guaranteed to be notified if the desired event happens.

14. The Vacuum Cleaner

POSTGRES' no-overwrite storage manager leaves old data, as well as current data, in user tables. Periodically, the system must be *vacuumed* to collect the old data and move it to an archive, if the user has declared that old data should be saved. The POSTQUEL language includes a special command, *vacuum*, to do this.

The original design of POSTGRES called for the vacuum cleaner to do more work. Most importantly, it should collect statistics on the database, such as computing the number of distinct values stored in the columns of database tables and the maximum and minimum values in each column. These statistics can be used by the query optimizer to make user queries run faster, by selecting an execution plan tailored to the data stored in the database.

The POSTGRES vacuum cleaner never collected these statistics. Miró has rewritten the vacuum cleaner so that it does collect statistics, and has expanded the set of statistics that are stored for each table. Whenever the vacuum cleaner runs, it will compute and store information for every table in the database. This information is correctly used by the query optimizer. As a result, Miró can choose better execution strategies than those selected by POSTGRES.

15. Disaster Recovery

Both Miró and POSTGRES support *fast recovery*. Should the computer running the database server crash, the server can restart immediately on reboot. Unlike conventional database systems, neither Miró nor POSTGRES must process a log at recovery time to bring the database to a consistent state. The no-overwrite storage manager guarantees that all committed changes are visible to all users, and that no uncommited changes are.

POSTGRES ignores the recovery issues associated with media failure. If the disk that stores a POSTGRES database is destroyed by a power surge or natural disaster, the data on it is lost. POSTGRES has no mechanism for dumping data from a database to tape, or for restoring it after a disk crash.

Miró provides tools to back up and restore data, even in the face of a media failure. In addition, the system administrator may enable very high-level logging that allows Miró to reapply all of the changes made to databases since the last backup was made. This log need only be examined if the disk storing the database was lost, and the data was restored from tape. It is ignored after a simple system crash.

16. Fast Data Copy

The POSTQUEL query language included a command that allowed users to copy data from ordinary files into tables in the database. POSTGRES included code to handle this bulk loading specially, in the interest of efficiency.

Miró has written an additional bulk loading tool, called *MiroCopy*. This is a facility that allows data to be converted into the correct types as it is entered into the databse, and that invokes the rules system on the data as it enters the system.

17. Front-End Tools

POSTGRES offered little in the way of tools for users of the database system. The *monitor* program accepted POSTQUEL queries from the user and submitted them to the server, and formatted and displayed the answers. Users who wanted other kinds of access had to write programs that used *libpq* to do what they wanted.

Miró offers a counterpart to *monitor*, called *msql*, which supports *ad hoc* queries against the database system. As noted above, Miró also supplies a library called *libmiro* for programmers who want to manage data in a database.

Miró also offers a sophisticated browser.

The browser consists of two parts, the Navigator and the Viewer. The Navigator allows a user to select query module icons with the mouse, connect them, and to execute the resulting query program. The data returned by the query program is passed to the Viewer, which renders the data on the screen and allows the user to pan and zoom over the results. Programmers can write query modules for the Navigator and locator and rendering functions for the Viewer to control what data is displayed, and where and how it is drawn.

18. Database Administration Tools

The POSTGRES project provided users with a free database system that was much more powerful than those available commercially. The programmers on the project concentrated on providing interesting features, and largely ignored tools that could make it easier to control or administer a running database server.

Miró has developed a suite of tools to make database administration easy. Tools are provided to dump and restore a database, start and stop the server, and install the database system on a new computer. In addition, Miró has added a number of *virtual tables* to the system. Virtual tables allow the database administrator to query the state of the database server. For example, the administrator can ask to see all the locks held by some user, or for information on all users currently using Miró. These tables can be queried like ordinary tables using SQL, and the system includes tools that issue the proper queries automatically.

Finally, Miró includes a license manager to control the number of users connected to the database server at one time.

19. Performance Improvements

The POSTGRES research group, while always interested in performance, was primarily committed to implementing and experimenting with new features. POSTGRES performed reasonably well on standard benchmarks, and did exceptionally well on some of the newer ones, such as Rick Cattell's OO1.

Miró has paid close attention to system performance. Depending on the particular benchmark used, Miró is between two and one hundred times the speed of University POSTGRES. As a commercial offering, Miró must compete against faster products in the marketplace. This means that the company is committed to speeding the system up even more in the future.

20. Engineering Improvements

POSTGRES was developed by a core team of programmers that averaged about ten members. As old programmers graduated, new ones joined the project. The primary goal of group members has always been to get advanced degrees and to leave the University for more remunerative employment in industry. As a result, stylistic and correctness controls on the software were not always rigorously enforced. The code was, in places, difficult to understand and hard to maintain.

Miró has made a serious effort to apply software engineering expertise to the system. Large modules have been rewritten, and every line of the system has been closely examined to be sure that it follows conventions on naming, use of data structures, and handling of errors. The result is a software system that is not only easier to maintain but also more reliable than the original University system was.

21. Completeness

The original design of POSTGRES specified a number of features that were never implemented. Usually, this was because these features were well-understood and supported by conventional database systems, and so were not of interest to the POSTGRES researchers.

Whenever possible, Miró has filled in these holes. For example, POSTGRES provides only table-level locking, but Miró has implemented finer-granularity locking thus promoting concurrency by allowing more users to share the same tables at the same time. Miró has also implemented genuinely recoverable indices, so that system crashes do not lead to corrupt indices in the database.

22. Testing and Support

Perhaps the most significant difference between the research and commercial systems is that the commercial system is thoroughly tested and backed by a committed support group. The programmers at the University, while always interested in helping users, were primarily committed to finishing their research and getting their degrees. As a result, user questions often went unanswered, and bugs were simply acknowledged, rather than fixed.

Since Miró plans to be a major player in the database market, and to remain one for a long time, it must provide reliable software and excellent support to its users. The company understands this, and has augmented its first-rate engineering department with a first-rate support group. This team has produced genuinely commercial quality software with all of the advanced features of the research system, and stands behind its product full-time.