Building a Laboratory Information System around a C++-Based Object-Oriented DBMS*

Nathan Goodman   Steve Rozen   Lincoln Stein

{nat,steve,lstein}@genome.wi.mit.edu
Whitehead Institute for Biomedical Research
One Kendall Square
Cambridge MA 02139
USA

Abstract

MapBase is a laboratory information system that has been supporting a high-throughput genome-mapping operation for the last three years. We chose to build MapBase around a C++-based OODBMS because, like CAD, CASE, and GIS applications, MapBase must be able to represent complex data and operations while providing fast response.

However, MapBase also turned out to share many characteristics of classical information systems: it provides a central repository of carefully administered, mission-critical data used by clients written in many languages and running on a variety of hardware. In addition, our laboratory emphasizes continuous process re-engineering, with the result that MapBase's schema must evolve rapidly in order to reflect the current experimental workflow.

We discuss how the technical characteristics of our OODBMS interacted with our requirements to form MapBase's current architec-

ture, and we analyze its strengths and weaknesses.

1 Introduction

A high-throughput laboratory project is characterized by large numbers of similar experiments organized into a production line. For example, the genome-mapping projects at the Whitehead Institute/MIT Center for Genome Research require millions of experiments to determine the location of short DNA sequences, called *markers*, on the chromosomes of an organism. As another example, [12] describes a high-throughput project to find partial DNA sequences of expressed genes, so-called *expressed sequence tags*. This project reads the DNA sequences of approximately 2000 expressed sequence tags per month, and hopes to increase this rate by an order of magnitude.

Such high-throughput laboratory projects require a laboratory information system to manage laboratory workflow and experimental results. MapBase is the database component of a laboratory information system that has been supporting the genome-mapping operations of Whitehead Institute/MIT Center for Genome Research (Genome Center). In this paper we focus on MapBase; please see [24] for a description of the information system as whole.

The Genome Center's genome-mapping efforts resemble a factory production line, in that a short piece of DNA, s, is found to be a marker by performing on it a sequence of steps, such as

- determining the DNA sequence of s,

- checking to see that the Genome Center has not already worked with s,

**Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994**

• determining and purchasing appropriate chemicals for further experiments on s,

and so forth. Some steps are performed by computer, and MapBase records the results from each step.

However, there are differences from a real factory production line. Some of the experimental steps require highly-trained scientists to interpret results. Many of the steps are inherently error-prone and sometimes must be re-done, which causes cycles in the workflow. Also, fewer than half of the potential markers that the Genome Center examines actually turn out to be suitable as markers.

In addition to managing experimental data, MapBase also stores analyzed results and makes them available to researchers in laboratories throughout the world.[1]

We began work on MapBase in January of 1991, and put it into production in September of that year. At the beginning of our work on MapBase, we could not formulate a complete set of requirements because no one had ever developed a database to support this kind of genome-mapping operation. However, after three years of developing and operating MapBase, we have formed a clear idea of MapBase's role in our genome-mapping production lines and of the kinds of support MapBase requires from a database management system (DBMS).

We divide MapBase's DBMS requirements into three main groups:

Modeling Expressiveness and Performance
MapBase needs to model complex objects such as DNA sequences, genome maps, and experimental workflow, while still providing acceptable response time. In this requirement MapBase resembles applications such as computer-aided design, computer-aided software engineering, and geographic information systems—applications that are collectively dubbed "CAx applications" in [20].

Integrative Role Like classical information systems, MapBase must provide a central repository of carefully administered, mission-critical data. This data helps integrate the operations of many programs and human activities. The requirement that MapBase fill an integrative role implies an additional need: multi-lingual access from disparate hardware. Section 3 discusses this need in more detail.

Schema Evolution The Genome Center is engaged in continual process re-engineering as it refines its

experimental protocols. Consequently, MapBase requires a schema change (i.e. a change to the set of C++ classes stored in the database) at least every two months. In addition, the Genome Center from time to time embarks on new projects. For example, until recently the Genome Center's major project involved producing a certain type of genome map called a *genetic-linkage map*. However, the Genome Center is now beginning to work on another type of map, called a *physical map*, the construction of which uses different experimental protocols and consequently different kinds of data.[2]

In the sequel we discuss these requirements in more detail, and show how they shaped MapBase's design.

2 Why a C++-based OODBMS?

It was MapBase's requirement to model complex data that led us to select an object-oriented database management system (OODBMS). Once we had decided on an OODBMS we considered GemStone [6], O_2 [16], ObjectStore [13, 20, 17], ONTOS [19], and VERSANT [28]. We decided on a C++-based OODBMS (that is, one of ObjectStore, ONTOS, or VERSANT) for the following reasons:

1. The expressiveness and performance requirements that MapBase displayed resembled those of the CAx application for which the C++-based OODBMSs were designed. We also knew that we would not be storing huge amounts of data—the Genome Center's largest instance of MapBase requires about 60 Megabytes of disk space—so we hoped to be able to keep much of the database in physical memory. We expected C++-based databases to perform well under these circumstances.

2. C++ was becoming the most widely used object-oriented language, which suggested that at least some of the C++-based OODBMSs would succeed in the marketplace.

3. We did not want to become locked into any particular vendor (especially in light of the fact that, in early 1991, object-oriented databases were very new). We had several C++-based OODBMSs to choose from, and believed that, if necessary, we

---
[1] For some of the biological results to which MapBase's use contributed please see [8].

[2] In its need to accommodate frequent changes in experimental protocols, MapBase differs from some other, well-known, molecular biology databases, such as GenBank [5], the PIR-International Protein Sequence Database [10], and many others, which act as central repositories of published results from numerous laboratories.

would be able to port MapBase between C++-based OODBMSs.

We chose ObjectStore because we judged it to be the most robust of the C++-based OODBMSs at the time the choice was made.

Although, as we discuss below, the facilities of ObjectStore do not ideally match MapBase's DBMS requirements, we continue to believe that our choice of a C++-based OODBMS was sound. Examples below illustrate the importance to MapBase of the modeling expressiveness of an OODBMS. In addition, experience suggests that for some applications relational databases cannot be tuned to deliver adequate performance; in one case with which we are familiar whole tables had to be moved out of the DBMS to achieve the necessary performance [21]. This is not to say that developing a laboratory information system around a relational DBMS is impossible; the expressed sequence tag project mentioned in section 1 built its laboratory information system, ESTDB (Expressed Sequence Tag Data Base), around the Sybase relational DBMS [15]. Such a choice involves trade-offs, however: well-defined design methodologies and strong support for the database's integrative role in exchange for less modeling expressiveness and less opportunity to tune performance by modifying data representations.

3 MapBase System Architecture

We initially hoped to use a system architecture similar to the one represented in figure 1, in which a number of ObjectStore clients executing on various computers in a network would interact concurrently with MapBase.[3] Some clients would provide data entry and interaction with lab personnel, others would perform automatic data analysis needed as part of the experimental protocol, and still others would disseminate analyzed results to biologists in other laboratories.

In this architecture, the ObjectStore server and its clients communicate by sending pages of binary data to each other.

Several considerations made the architecture of figure 1 impractical:

1. Although ObjectStore provides atomic transactions, it provides no roll-forward recovery procedure whereby a snapshot backup can be brought up to date by applying logs of transactions committed since the backup.[4] Consequently we had to provide our own logging.

---

[3] This diagram suppress some detail. In particular, in addition to the ObjectStore server, other ObjectStore processes manage paging files on machines running ObjectStore clients.

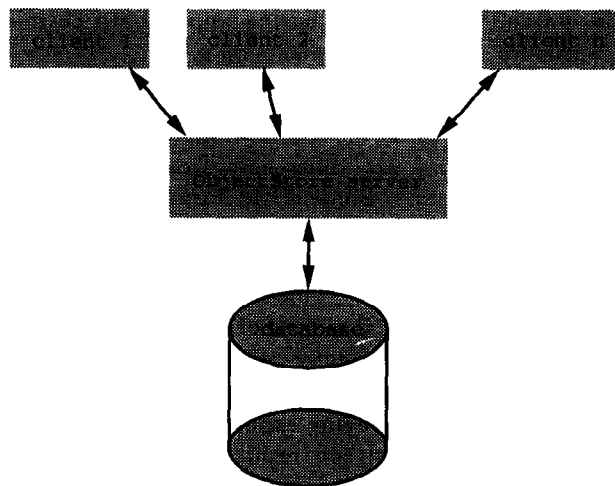[4] Roll-forward recovery is planned for ObjectStore Release 4 [18]

Figure 1: Initial System Architecture for MapBase.

We also had to provide our own logging because the first version of ObjectStore that we used provided no support for schema evolution. Whenever we modified the storage layout of a C++ class we had to re-load the database. (The current version of ObjectStore supports schema evolution, but, because we had already developed our own procedures for schema evolution, we have not used ObjectStore schema-evolution facilities.)

2. We were unable to achieve multi-user performance on MapBase's workload that satisfied our requirements, probably due to concurrency hot spots in low-level storage allocation routines. (We understand that this problem will be fixed in the next release of ObjectStore.)

3. Most programs that we use cannot be written in C++ as ObjectStore clients, for a variety of reasons:

   • Some users need a particular interface. For example, most of the Genome Center's biologists and lab technicians are already familiar with Macintoshes and Microsoft Excel spreadsheets. Consequently, to minimize learning time and maximize user acceptance, the Genome Center uses Excel running on Macintoshes for much data entry. In fact, we initially provided special-purpose data-entry application programs. However, biologists and lab technicians preferred to first enter their lab data into spreadsheets, and then re-enter their results into MapBase only when absolutely necessary for their data to be analyzed by programs connected to MapBase. Another example is distribution of data to

724

researchers in other laboratories. E-mail is a least common denominator: virtually all researchers have access to it. Therefore we provide an E-mail server whereby researchers in other laboratories can retrieve data from MapBase.[5]

- Some programs were written at another laboratory or even run at remote sites. An example is BLAST, which is used to search for similarities between MapBase's marker sequences and sequences in the worldwide sequence database, GenBank [1, 5]. BLAST runs as a an Internet-accessible compute server on a National Center for Biotechnology Information computer. We use this server because it is fast and has access to extremely current versions of GenBank data.

- Some programs were written at the Genome Center, but must have an interface that allows them to be used at other laboratories, and not just with MapBase. An example is MAPMAKER, one of the programs the Genome Center uses to assemble markers into a genome map [14].

- Some programs are less demanding or are exploratory, and can be written in a language, such as LISP or Perl, that is terser than C++. Programs for "data dredging" or "data mining" [26] fall into this category. An example would be a program to analyze workflow loops due to reworked experimental steps.

In addition to these three main considerations, an ancillary consideration was the fact that, *in and of itself*, the architecture of figure 1 offers no application program interface at a *logical level*. C++ per se is not really a data modeling language, and it can be difficult for application programmers to understand how to use the data in MapBase. (In contrast, for users of relational DBMSs the logical schema, together with written documentation about database semantics, usually projects a relatively well-defined and succinct interface for application programs.) Perhaps what is required is better class design and better documentation, but those seeking to adopt the architecture of figure 1 should be careful to provide a clean, documented interface to database information. Use of C++ by itself will not necessarily coax developers in that direction as standard relational database design methodology would.
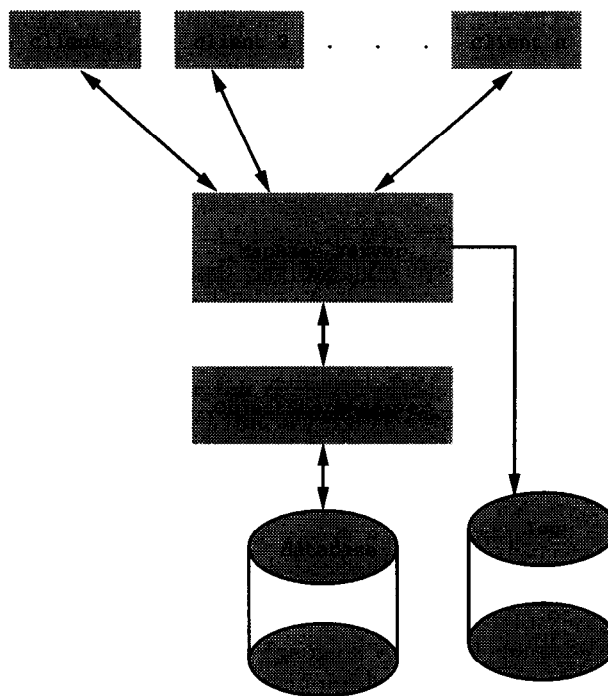

Figure 2: MapBase System Architecture.

Related to this ancillary consideration is the fact that in the architecture of figure 1 there is no "fire wall" between applications and the database. Thus an applications programmer could, for example, inadvertently store unreachable objects or dangling pointers in the database.

Because of these considerations we adopted the system architecture represented in figure 2. In this architecture, the only ObjectStore client is the MapBase server, which receives requests from its clients in the form of text queries posed in a special-purpose query language. The MapBase server also returns results to clients in textual form.[6]

A special-purpose query language is necessary because ObjectStore provides essentially no interactive query language.[7] We did not want to have to add a unique interface for each application that could not be tightly integrated with the MapBase server, so we constructed a query language.

The architecture depicted in figure 2 addresses all of the considerations enumerated above.

---

[5]ESTDB [12] also depends heavily on off-the-shelf software — in this case Macintosh HyperCard—for user interfaces.

[6]Queries and results are transmitted over TCP/IP sockets.

[7]The ObjectStore query expressions described in [20] (and referred to as ObjectStore DML in [17]) must be embedded in C++ and compiled. In the programmatic interface, a WHERE clause can be supplied at run time, but it can invoke only a restricted class of functions (containing only boolean comparison operators such as >). The WHERE clause cannot, in general, invoke the operations associated with abstract data types stored in the database.

1. The MapBase server writes all updates to a logical log, which can be re-read if roll-forward recovery is necessary. We have confidence in this scheme because

   - we reload the database from these logs when the set of C++ classes stored in the database changes, and

   - we have reloaded the database to correct coding errors and to recover from corruption in the stored data structures.

2. Multi-user performance is not an issue, because the MapBase server is the only ObjectStore client, and MapBase enjoys the benefits of ObjectStore's excellent single-user performance.

3. Programs written in any language can communicate with the MapBase server using the special-purpose query language.

In addition, the query language and its documentation provide a well-defined application program interface. It is worth noting that our experiences with MapBase strongly support two of propositions of the *Third Generation Database System Manifesto* [25]: Proposition 2.1—the importance of a high-level query language— and Proposition 3.1—the importance of multi-lingual access.

Like all designs, the one in figure 2 is a compromise, and there are costs associated with it:

- The MapBase server is fairly complex, because it must maintain connections with all clients, and buffer partial queries until they are complete. Furthermore, it has to implement a parser and interpreter for the special-purpose query language.

- Each client must parse the query results it receives from the MapBase server. However, we have Perl routines for parsing query results, and most of the Genome Center applications are connected to MapBase via Perl scripts. For example, when an Excel spreadsheet sends data to MapBase, it is first received by a Perl script which generates the appropriate update statements in the special-purpose query language. Similarly, the E-mail server is written in Perl.

- Clients written in C or C++ cannot benefit from the type information in the MapBase database. (However, in the current state of the art, Excel spreadsheets and E-mail readers could not use type information anyway.) More generally, much of the data-modeling expressiveness for which we originally chose an OODBMS cannot be transmitted to clients.

Lack of a general, interactive query language is probably the most severe limitation of MapBase's current architecture, because we have tended to make only those schema changes to which our special-purpose language can be easily adapted, with deleterious effect on MapBase's class design. The special-purpose query language depends heavily on the idea of markers and histories of experimental steps on single markers, so it has been hard to accommodate new laboratory protocols that do not involve markers or that involve steps producing results for more than one marker. Section 6 discusses how we hope to remove this limitation, as well as others.

4 MapBase's Data Model

The Genome Center runs several instances of Map-Base: two for mouse genetic mapping projects, one for a human physical mapping project, three for various other projects, and a few for testing. The data model and schema in each are basically the same. There are some C++ classes that are stored in the database, and others that are used only during the execution of the MapBase server. The mouse database stores approximately 100 different classes (not counting separately numerous parameterized collection classes), and the MapBase server uses approximately 40 additional classes.

Many of the classes stored in the database represent specialized scalar types: for example there is a class Centimorgan (derived from Float), which is used to represent distances between markers. Other classes represent more complex objects such as text strings and DNA sequences. Probably the most complex class in the mouse database is the one that represents assembled genome maps—the result of analyzing all the experimental data in the mouse database.

The representation of complex objects with complex, user-defined operations is a notable strength of C++-based OODBMSs such as ObjectStore. With the full power of the C++ class system we were able to define classes that store DNA sequences of arbitrary length, and that provide operations that would not be available in today's relational DBMSs. For example, MapBase represents DNA sequences as strings of the letters A, C, G, T, representing the *nucleotide bases* adenine, cytosine, guanine, and thymine, with the letter N used to indicate that the base at that position could not be experimentally determined.[8] For all practical purposes, DNA sequences can be arbitrarily long, making it important to avoid length limits in their database representation. The longest DNA se-

---

[8] For a high-level introduction to molecular genetics, see [2], which references more detailed treatments.

quence in MapBase contains over 3000 bases, several times the maximum length we expected when we began development.

An example of a user-defined operation implemented in MapBase is *reverse complementation*. To obtain the reverse complement of a DNA sequence, one reverses its order, and then substitutes the nucleotides according to the Watson-Crick base-pairing rules: G↔C and T↔A. For example, the reverse complement of GATTCCGGG is CCCGGAATC. Reverse complementation is an important operation because, when DNA molecules occur in their usual double-stranded form, each strand is the reverse complement of the other.

An additional, important advantage of a C++-based OODBMS is the ability to use custom representations to tune performance. Consider the following example. A great deal of MapBase's storage space is devoted to storing DNA sequences, and MapBase's performance depends on keeping as much of the data in main memory as possible. If we needed to economize on space we could store DNA sequences using only 3 bits to encode each base rather than the 8 bits we use in the current implementation. If we have done our job right in designing the DNA sequence class, its clients would be oblivious to this change of representation.

5 Workflow Data and Schema Evolution

As discussed above, one of MapBase's key responsibilities is to track the experimental steps performed on each potential marker. In so doing, MapBase records the user that entered each experimental result and the approximate time the step was performed, because sometimes experiments must be re-done. Normally, only the most recent experimental result is of interest. About a quarter of the classes in the mouse database are derived from a single base class, Process_Step. Each instance of one of these Process_Step classes records the results of a particular experimental step, using step-specific attributes whose values are usually instances of a MapBase scalar class (e.g. Centimorgan) or a string or DNA sequence class. In the C++ implementation, each marker is associated with an array of pointers to objects each of whose class is derived from Process_Step. This implementation allows MapBase to quickly find the most recent experimental results associated with particular markers using a linear search of each marker's experimental steps in reverse chronological order.

It is the recording of experimental steps that leads to MapBase's need for frequent schema evolution, be-

cause of two characteristics of the Genome Center's mapping operations:

1. continual process re-engineering and

2. occasional deployment of new processes.

Following are examples of the kinds of changes MapBase must accommodate in response to process re-engineering:

1. In one of the Genome Center mapping operations, we determine the location of the markers on a map based on the length of the marker in 46 individual mice. The process of determining the length of a potential marker in those mice is called "genotyping". We recently added a new error-checking procedure when a lab technician enters genotyping data: when the lab technician cannot resolve an apparent genotyping error (which is detected by a "sanity-checking" routine in the MapBase server), then the associated marker must be reviewed by a scientist. The scientist then either

   (a) changes the genotyping information,

   (b) certifies that the genotyping is in fact correct,

   (c) requests a new genotyping experiment, or

   (d) declares the marker to be useless.

   This new checking step required the addition of a new, special-case query to the MapBase server.

2. The first step in trying to find markers is to construct a "small-insert library"—a large set of small DNA fragments (i.e. no longer than a few thousand bases), each of which can be checked to see if it could serve as a marker. In an initial analysis we concluded that if we found a potential marker in the small-insert library, then there would be only about a 0.1% chance that we had previously examined that fragment as a potential marker. However, because of an unanticipated experimental artifact, we found that the actual chance of a potential marker already having been examined can be as high as 40%. When we discovered this we very quickly had to add a duplicate-checking workflow step, because our mapping project was essentially stalled until we could screen for duplicates. This new step was implemented as a class derived from Process_Step.

Workflow refinements, like the first example, are a regular occurrence. And because the Genome Center is using new technology we must compensate for occasional, inevitable missteps such as the problem with duplicates in the second example.

6 Discussion

MapBase constitutes a successful application of OODBMS technology to an information system with challenging requirements. However, we would like to improve on the current design of MapBase by providing

1. a more general query language, and

2. more convenient support for schema evolution.

We discussed above the importance of the MapBase query language in integrating programs written in many languages and running on a variety of hardware, and we also discussed some of its limitations. We actively sought a better query language [11]. However, promising object query languages such as CQL++ [9] and OQL [7] do not yet have interpreted implementations.[9] We also considered rewriting MapBase for VERSANT, which does offer an interpreted query language [27] that is almost identical to OQL[C++] (the language used in the OpenOODB project) [3, 4]. SQL3 [23] also shows promise as a solution to MapBase's query and data-modeling requirements. SQL3 permits extension through user-defined classes ("abstract data types" in SQL3 terminology) with inheritance and object identity, and through user-defined functions coded in another programming language. Presumably SQL3 databases will inherit from their relational parents an emphasis on query languages, application program interfaces, and industrial-strength data administration facilities. We expect, in the future, to consider SQL3 databases for laboratory information systems.

After examining options for providing a more general query language for MapBase we concluded that, because of the substantial experience the Genome Center has accumulated in developing and maintaining ObjectStore databases, we would use ObjectStore to implement a successor to MapBase—one with a much more general query language. We are currently deploying this successor, which we call LabBase [22]. LabBase was to some extent inspired by CAx applications. Just as the user of, for example, a CAD database, does not usually create C++ classes to represent a new type of gasket or pose queries about it, we would like the user of a laboratory information system to be able to

represent a new kind of experimental step and pose queries about it without having to create C++ classes.

LabBase's query language—non-recursive datalog with aggregates—is more general than MapBase's. LabBase also provides a data dictionary in which users can define new kinds of experimental steps and new step attributes without modifying the C++-code in which LabBase is written. LabBase supports a data model that generalizes MapBase's: the central concepts are *material* (a generalization of MapBase's "marker") and *experimental step*. Various predicates in the LabBase query language allow queries over all materials of a particular kind, and provide access to either the most recent steps or to all the steps associated with particular materials. In MapBase, schema evolution requires modification to the MapBase server (entailing re-compilation and re-linking), and sometimes modification of the roll-forward logs (either manually or by means of some kind of script). In LabBase, most schema changes (those not requiring a new built-in primitive type) can be accomplished by adding new kinds of materials, experimental steps, or step attributes to the data dictionary. Finally, a sufficiently knowledgeable user could add new primitive types and operations to a LabBase server by linking in appropriate C++ class and function definitions. Therefore we hope we have retained some of the advantages of C++ in terms of modeling expressiveness and user-defined representations.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol. (England)*, 215(3):403–410, Oct. 1990.

[2] P. Berg and M. Singer. *Dealing with Genes*. University Science Books, 1992.

[3] J. A. Blakeley. OQL[C++]: Extending C++ with an object query capability. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press/Addison-Wesley, 1994.

[4] J. A. Blakeley, W. J. McKenna, and G. Graefe. Experiences building the Open OODB query optimzier. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD Int'l. Conf. on Mgmt. of Data*, 287–296, June 1993.

[5] C. Burks, M. Cassidy, M. J. Cinkosky, K. E. Cumella, P. Gilna, J. E.-D. Hayden, G. M. Keen, T. A. Kelley, M. Kelly, D. Krsitofferson, and

---

[9]There are two reasons why MapBase requires interpreted query execution rather than run-time compilation and loading because (i) it takes too long to compile even a short .C file (e.g. around 25 seconds on a Sparc 10, for a 73-line file plus headers), which is unacceptable in terms of MapBase's response requirements, and (ii) an error in a dynamically loaded .o file can crash the MapBase server. (It is unacceptable that a coding error in a single, ad hoc query make the database unavailable to all clients during recovery.)

J. Ryals. GenBank. *Nucleic Acids Research*, 2221–2225, 1991.

[6] P. Butterworth, A. Otis, and J. Stein. The GemStone object database management system. *CACM*, 34(10):65–77, Oct. 1991.

[7] R. Cattell, T. Atwood, J. Duhl, G. Ferran, M. Loomis, and D. Wade. *The Oject Database Standard: ODMG-93*. Morgan Kaufman, 1994.

[8] N. G. Copeland, N. A. Jenkins, D. J. Gilbert, J. T. Eppig, L. J. Maltais, J. C. Miller, W. F. Dietrich, A. Weaver, S. E. Lincoln, R. G. Steen, L. D. Stein, J. H. Nadeau, and E. S. Lander. A genetic linkage map of the mouse: Current applications and future prospects. *Science*, 262:57–66, Oct. 1993.

[9] S. Dar, N. H. Gehani, and H. V. Jagadish. A SQL for a C++ based object-oriented DBMS. In *Proc. of the Int'l. Conf. on Extending Database Technology*, Mar. 1992.

[10] D. G. George. *PIR-International Protein Sequence Database (PSD): The Protein Sequence Component Version CO2_6.2*. National Biomedical Research Foundation, 3900 Reservoir Road, NW, Washington DC 20007, USA, May 1993.

[11] N. Goodman, S. Rozen, and L. Stein. Requirements for a deductive query language in the MapBase genome-mapping database. In R. Ramakrishnan, editor, *Proc. of the Workshop on Programming with Logic Databases In Conjunction with ILPS, Vancouver, B.C.*, 18–32, Oct. 1993. Available as Tech. Report #1183, Computer Sciences Department, University of Wisconsin, Madison WI 53706, USA.

[12] A. R. Kerlavage, M. D. Adams, J. C. Kelly, M. Dubnick, J. Powell, P. Shanmugam, J. C. Venter, and C. Fields. Analysis and management of data from high-throughput expressed sequence tag projects. In T. N.Mudge, V. Milutinovic, and L. Hunter, editors, *Proc. of the 26th Annual Hawaii Int'l. Conf. on System Sciences*, vol. 1, 585–594. IEEE Computer Society Press, Jan. 1993.

[13] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore database system. *CACM*, 34(10):50–63, Oct. 1991.

[14] E. S. Lander, P. Green, J. Abrahamson, A. Barlow, M. J. Daly, S. E. Lincoln, and L. Newberg. MAPMAKER: an interactive computer package

for constructing genetic linkage maps. *Genomics*, 1(1):174–181, Oct. 1987.

[15] D. McGoveran and C. J. Date. *A Guide to Sybase and SQL Server*. Addison-Wesley, 1992.

[16] O. Deux et al. The O_2 system. *CACM*, 34(10):34–48, Oct. 1991.

[17] Object Design, Inc., 25 Burlington Mall Rd., Burlington MA 01803-4194, USA. Manual set for ObjectStore Release 3.0 for UNIX Systems, Dec. 1993.

[18] P. O'Brien. R3 & R4 product directions overview, Mar. 1994. Talk presented at ObjectStore Northeast users group meeting.

[19] ONTOS, Inc., Three Burlington Woods, Burlington MA 01803, USA. *ONTOS DB 2.2 Developer's Guide*, Feb. 1992.

[20] J. Orenstein, S. Haradhvala, B. Margulies, and D. Sakahara. Query processing in the ObjectStore database system. In M. Stonebraker, editor, *Proc. of the 1992 ACM SIGMOD Int'l. Conf. on Mgmt. of Data*, 403–412, June 1992.

[21] S. Rozen and D. Shasha. Using a relational system on Wall Street: The good, the bad, the ugly, and the ideal. *CACM*, 32(8):988–994, Aug. 1989.

[22] S. Rozen, L. Stein, and N. Goodman. Constructing a domain-specific DBMS using a persistent object system. In *Sixth Int'l. Workshop on Persisten Object Systems*, Sept. 1994.

[23] ISO and ANSI SQL3 working draft, Feb. 1993.

[24] L. Stein, A. Marquis, E. Dredge, M. P. Reeve, M. Daly, S. Rozen, and N. Goodman. Splicing UNIX into a genome mapping laboratory. In *USENIX*, 1994.

[25] The Committee for Advanced DBMS Function. Third-generation database system manifesto. *SIGMOD Record*, 19(3):31–44, Sept. 1990.

[26] S. Tsur. Data dredging. *Data Engineering*, 13(4), Dec. 1990.

[27] Versant Object Technology Corp., 4500 Bohannon Dr., Menlo Park CA 94025, USA. *VERSANT C++ Application ToolSet Version 2.0, VERSANT Object SQL*, June 1993.

[28] Versant Object Technology Corp., 4500 Bohannon Dr., Menlo Park CA 94025, USA. *VERSANT Object Database Management System Release 2 System Manual*, July 1993. Part 1003-0793.