

A Freely Sharable Database Management System Designed for Use in Component-Based, Modular Genome Informatics Systems

(From a proposal to the Department of Energy)

July 14, 1994

Nathan Goodman and Steve Rozen

{nat,steve}@genome.wi.mit.edu
voice: 617 252 {1904,1923}, fax: 617 252 1902
Whitehead/MIT Center for Genome Research
Whitehead Institute for Biomedical Research
One Kendall Square, Building 300, Floor 5
Cambridge MA 02139
USA

A Specific Aims

The primary aim of this project is:

To produce and disseminate a freely sharable, domain-specific database management system (DBMS) suitable for use as a component of a genome informatics system.

Over the past three-and-a-half years we have developed and operated genome informatics systems for the genetic- and physical-mapping projects carried out at the Whitehead Institute/MIT Center for Genome Research (the Genome Center). Based on our experience and our analysis of other successful genome informatics systems, we have a concrete and pragmatic understanding of what genome informatics systems require from a DBMS.

More specifically, then, our aim is to construct a domain-specific DBMS tailored to the needs of genomics researchers and genome informatics systems by providing the following features, which we divide into five main groups:

Data Modeling Data modeling capabilities suitable to genomic data are critical. The proposed domain-specific DBMS will provide:

1. The modeling expressiveness of a “semantic” or “structural object-oriented” data model.
2. Built-in support for biological data types, such as biological sequences of arbitrary length, and specialized operations on these, such as reverse complementation of DNA sequences and translation of DNA to polypeptide sequences.

3. Built-in support for managing laboratory workflow and data.
4. An extensible internal architecture that allows programmers to add new built-in types and “user”-defined functions to the DBMS.
5. The ability to accommodate—both logically and administratively—schema evolution and partial data.

Sample Schemas, Data Sets, and Client Programs A “starter set” of schemas, sample data, and viewing and analysis tools will accompany the proposed domain-specific DBMS. These will include:

- a base of documented, modifiable schemas, including schemas from the Genome Center’s genetic- and physical-mapping projects and translated ACeDB and NCBI/ASN.1 schemas, together with sample data sets, and
- A base suite of viewing and analysis tools, including some adapted from those used by the Genome Center.

These schemas and tools will allow users to produce a first-cut genome informatics system in a matter of a few days or weeks.

Viewing, Updating, and Querying The proposed domain-specific DBMS will provide the following facilities (each suitable for particular situations) for viewing, updating, and querying its contents:

1. Automatically generated (from the schema) Mosaic/HTML forms for querying and updating the database.
2. Application program interfaces (APIs) for C, C++, perl, and Tcl/Tk.
3. An associative query language at least as powerful as SQL.
4. Data and schema browsers to provide a “point-and-click” interface to database contents.
5. Utilities that can load data in .ace and NCBI/ASN.1 data formats, and that can write query results in these formats.

Robustness and Performance The proposed domain-specific DBMS will be robust, will provide transactions (with archive logging) that support concurrent updates, and will provide single-user and concurrent multi-user performance that is competitive with commercial DBMSs.

Broad Availability Two features will contribute to achieving broad availability of the proposed domain-specific DBMS. The first is the ability to run on all major Unix platforms and on Macintoshes. The second, much more important feature, is *freely redistributable source code*. This means that software developers are allowed to modify the source code and redistribute the changed code without special permission of the original authors, under a license like the Free Software Foundation’s *copyleft* license or perl’s *artistic* license.

The availability of a DBMS with these features would free genome informaticists of many data-management tasks and allow them to focus their efforts on projects of more immediate benefit to genomic research.

B Background and Significance

B.1 Why We Need a Component-Oriented DBMS

We are fast approaching an age in which component-based genome informatics systems will be the norm. By “components” we mean significant subsystems, such as databases, user interfaces, analysis programs, programs to control laboratory instruments (e.g., robots), and programs to interpret image and other data collected by laboratory instruments. By “component-based systems” we mean systems constructed in a modular fashion from components that are designed to plug-and-play.

In the arena of federated systems, the component-based approach has clearly taken hold. Numerous data and compute servers are available on the Internet and the WorldWide Web (WWW) (Berners-Lee et al., 1992), and through the magic of Mosaic (NCSA, 1993) and the underlying HyperText Transfer Protocol (HTTP) (Internet Engineering Task Force) it is relatively straightforward to integrate these services in user interfaces and programs.

In the arena of local informatics, the component-based approach is not as well established, but the benefits are likely to be as great.

The component-based approach is an instance of the well-known, classic method of modular software design and shares its many benefits. In brief, modular design is based on the human problem-solving strategy of divide-and-conquer: we divide a complex system into a number of simpler components, and thereby reduce the initial complex problem into (i) a set of simpler sub-problems, plus (ii) the problem of integrating the sub-solutions into a complete solution. As in all areas of human problem solving, this strategy works to the extent that the sub-problems are genuinely simpler than the original problem, and the integration problem is also not too hard. The essence of good system design is finding a decomposition with these properties; this is a matter of skill and experience.

One example of a component-based local informatics system is the genome informatics system developed and operated by the Genome Center (Stein et al., 1994). We use this system to illustrate some specific features and benefits of component-based systems for local genome informatics.

- Ideally, components should be designed as independent software elements so they can be developed, tested, and maintained independently.
- Since they are independent, components can be written in the programming language that best suits the problem at hand. Within the Genome Center, we use (or have used) components written in C++, C, perl (Wall and Schwartz, 1990), Smalltalk (Goldberg and Robson, 1983), Lisp (Steele, 1990), Prolog (Clocksin and Mellish, 1987), and Tcl/Tk (Ousterhout, 1994).

- Since they are independent, components can run on the platform that best suits the problem at hand. Within the Genome Center, most of our user interface programs run on Macintosh computers, which our users strongly prefer. Most other software runs on Unix workstations, because the Unix operating system supports concurrent, independent execution of multiple programs and is better suited to software development.
- Since the informatics system is built from modular elements, it is straightforward to incorporate software developed outside the Genome Center, or developed internally but for other purposes. The Genome Center uses the following such programs for essential purposes:
 - SQUIRREL (Dear, 1993) to identify and conceptually strip vector sequences from small-insert clones,
 - FASTA (Pearson and Lipman, 1988) to compare newly sequenced DNA markers against a library of known repetitive elements and other anomalous sequences,
 - BLAST (Altschul et al., 1990) to check whether random DNA coincidentally matches a known gene,
 - ted (Gleeson and Hillier, 1991) to edit trace data from DNA-sequencing machines,
 - PRIMER (Lincoln et al., 1991) to select PCR primer pairs, and
 - MAPMAKER (Lander et al., 1987) to construct genetic linkage maps.
- Modularity also makes it easy to replace existing components with new and improved ones. Sequence editing is an excellent example of this. The problem is to inspect the output of an ABI DNA-sequencing machine to determine regions of the sequence in which the quality is acceptable for selection of PCR primer pairs. The original solution, many years ago, required laboratory personnel to manually edit hardcopy ABI trace data, then transcribe their edits into a Macintosh Word document, and copy the document to a designated file directory *before* primers were selected. The next solution replaced the hardcopy editing with manual editing using ABI-supplied software; the user was still required to move the edited file to a designated file directory. Finally, we imported the ted trace editing program from the Washington University *C. elegans* sequencing project, reorganized the protocol workflow so that sequence inspection occurs *after* primer selection, and set up the software so that users have to inspect only the sequence in the immediate vicinity of the primers. Eventually, when someone invents a more automatic tool for this purpose, it will be easy for us to swap it in.
- In a similar vein, it is possible to radically change software technology used by some components within the system. For example, many of our user interfaces are implemented as “plug-in” modules for commercial Macintosh software, viz., Microsoft Mail and Excel. The decision to use this technology, which replaced Smalltalk interfaces, was made quite late in the project.

- Likewise, it is straightforward to embrace new technology as it becomes available. One example is the use of Mosaic/HTTP for forms-based queries originating both inside and outside the Genome Center, replacing a previous text-based interface for this purpose. As another example, having seen Searls's beautiful contig-display program (Searls, 1994) written in Tcl/Tk we have become enamored of Tcl/Tk as a graphics programming language and have begun using it to develop a number of new interactive graphical user interfaces. We expect to incorporate important new technology throughout the life of the project.
- Components can often be reused in other systems. This can be accomplished by design, if, when creating a component, one has the foresight (and energy) to make it general enough to solve a problem that goes beyond the immediate local needs. Or it can happen by luck, if a second system needs the exact same capabilities as the first. More typically in our experience, reuse happens through evolution in which a component originally designed for a narrow purpose is modified to solve a more general problem.

In addition to the above pragmatic benefits, there is also a compelling societal benefit to component-based system design. Almost all systems-oriented research and development in the genomics field is done in mega-projects funded through genome centers or through stand-alone community database projects. There are only a handful of small, "R01 scale" grants in this area, making it difficult for systems-oriented genome informaticists to function as independent scientists. (In contrast, there are a reasonable number of small scale grants on theoretical topics.) Genome informatics cannot flourish as a scientific discipline until it becomes possible for significant contributions to be made by individual, systems-oriented scientists. This cannot happen until the "unit of contribution" becomes the component, rather than the complete system. This in turn cannot happen until we learn how to build complete genome informatics systems out of independently developed components.

A closely related societal benefit is the potential to encourage community-based software development in this area. Software "products" such as Mosaic, perl, Tcl/Tk, T_EX (Knuth, 1989), and L^AT_EX (Lamport, 1986), GNU emacs (Free Software Foundation, 1994, Cameron and Rosenblatt, 1991), ACeDB (Durbin and Thierry-Mieg, 1991, Dunham et al., 1994), the Staden package of sequence tools (Staden and Dear, 1991, Charnock-Jones and Aparicio, 1994), and many others gain tremendous value from incremental improvements made by software developers throughout the community. For such community efforts to gel, it is essential that the community adopt a culture of free, and freely redistributable software .

A key impediment to widespread adoption of the component-based approach for local genome informatics systems is the absence of a DBMS suitable for use as a component in such systems. Database management is a large fraction of the local informatics job. As a practical matter, the strategy one adopts for database management often dictates the overall system architecture.

We do not mean to suggest that the lack of a suitable DBMS is the only obstacle. We see two other significant problem areas. One is the lack of an agreed-upon protocol or language for true graphical user interfaces that are portable and transportable. By "portable" we mean a language that produces equivalent graphical user interfaces on the major windowing systems, namely X-windows, Macintosh, and Microsoft Windows. By "transportable", we

mean a language that can be downloaded into a Mosaic-style client and executed in response to an HTTP-style action. Gazing into our crystal ball, we believe that Tcl/Tk, in particular the SafeTcl dialect of Tcl/Tk, will emerge in the near future to fill this role. The second problem we see is the lack of an agreed-upon protocol for exchanging genomic data types among components. We expect that both ACeDB's .ace format and NCBI's ASN.1 format (NCBI, 1993) will emerge as standards to solve this problem.

Nor do we mean to suggest that componentry is doomed without such a DBMS. Rather, our premise is that (i) the absence of a suitable DBMS diminishes the value of the component-based approach, and (ii) conversely, the availability of a high quality, suitable DBMS would greatly enhance the value and appeal of the component-based approach, and might well accelerate its acceptance.

B.2 ACeDB, SYBASE, and Genome Topographer

A great many DBMSs are used in the genome community, but only two can claim widespread acceptance. These are ACeDB and SYBASE (McGoveran and Date, 1992). Before embarking on a project to create yet another DBMS, it is imperative that we analyze the strengths that have led to the success of these systems, as well as their weaknesses. ACeDB is the outstanding example of software reuse and data sharing in our field. We believe that ACeDB owes its success to two primary characteristics:

- ACeDB is domain-specific, by which we mean it provides substantial built-in support for the needs of genome informatics systems, namely:
 - It comes with numerous example schemas (in ACeDB terminology, a set of “models”) that can be adapted to the needs of a particular genome informatics system. ACeDB implementations generally come with a dataset, making it easy for prospective users to experiment with the system.
 - ACeDB schemas are expressed in a *semantic data model* (Hull and King, 1987), also known as a *structural object-oriented* model (Dittrich, 1986). Database aficionados have debated the merits of semantic vs. relational data models since the early days of the database field. Proponents of semantic data models claim that semantic models are more intuitive than the relational model, and easier to use when creating schemas for complex databases. The relational camp replies that semantic models lack a solid theoretical foundation and as a result, query languages and query optimization for such models are on shaky ground. This is not the place to re-open this debate and we will meekly assert, as an empirical fact, that biologists who use of ACeDB find its schemas to be intuitive and easy to understand; users are able to understand how their data fits within the schema, and can modify schemas as necessary to suit their requirements.
 - ACeDB handles missing data in a flexible manner, which is very helpful when adapting an existing schema for a new dataset, and for schema evolution in general. ACeDB also provides tools for schema evolution.

- ACeDB provides a set of displays that give an intuitive, interactive graphical display of such genomics-specific objects as physical and genetic maps.
- The ACeDB community has encouraged software sharing by providing freely redistributable source code. For example, many ACeDB users have extended the preexisting ACeDB map displays. By using shared software, ACeDB has become a community effort; the entire ACeDB community benefits from the incremental contributions of each participant.

These two characteristics make it is easy for a laboratory to experiment with ACeDB. The main work involved is putting the laboratory's data into the required external .ace format, a task which is eased by the schemas that comes with the system, and by the system's graceful handling of missing data. By way of illustration, it took one of us two days to port the Genome Center's mouse genetic map dataset to ACeDB. (The port included DNA sequences for STS markers and PCR primers as well as genetic-map positions; it did not include detailed laboratory data which is the bulk of the data stored in our complete database.)

Despite ACeDB's impressive success, it is not a suitable for use as the DBMS component in a modular, component-based genome informatics systems. The main reasons are:

Weak support for database updates Only one person at a time can update the database. To perform an update, the user has to explicitly request "write access", which sets a write lock on the entire database. No one else can obtain write access until the first user explicitly relinquishes write access, or quits. Other users can read the database while updates are in progress, but they do not see updates performed since the time they began the ACeDB session. In other words, to see recent database updates, the user has to quit out of ACeDB, and re-launch the program. For this reason, ACeDB is not suitable for an environment requiring even modest levels of real-time or concurrent updates.

Weak support for database recovery ACeDB includes an elegant shadow-page scheme to prevent the database from being corrupted either by a "clean" crash of the ACeDB software or by most operating system crashes. But it does not provide "roll-forward" or "archive" logging to protect against a disk crash or major database corruption caused by a serious software bug. In such an event the database must be retrieved from the most recent disk backup or must be reloaded from the most recent .ace dump of the database; updates that occurred after the backup or dump are lost. This makes the system unsuitable for managing mission critical data that cannot be easily reconstructed from text files or other external sources.

No built-in support for managing workflow data Large-scale laboratories, such as the Genome Center, often have to track the history of operations performed on a given material, for example all analysis steps performed on a given STS marker. Although one could add a class to represent laboratory steps to an ACeDB schema, there is no built-in support for managing both the historical and static view of data discovered during experimental steps.

Not component-oriented There is no way to separate ACeDB's data management capabilities from its display and other capabilities. In other words, ACeDB is *monolithic*. ACeDB is a single, large program (about 100,000 lines of C code). The sheer bulk of the system makes it unwieldy to make changes, because each debug/test cycle requires a re-link of the entire program. Since it is a single program, a bug in any part of the system crashes the entire system, and a modification in one part of the system can introduce a bug elsewhere. This makes it difficult to use the system in an environment, such as the Genome Center, in which frequent and rapid software changes are essential.

No easy way to incorporate new technology Because ACeDB is monolithic it is difficult to extend the system with radical new technologies. We see no easy way to modify the ACeDB code to accommodate user interfaces implemented as Excel plug-ins or in Tcl/Tk. One could add these features on top of ACeDB, of course, but to do this systematically would require turning ACeDB into a kind of mega-component.

The use of SYBASE within the genome informatics community is quite different from that of ACeDB. SYBASE has been adopted by a number of different laboratories and groups, but there does not seem to be a SYBASE community in the same sense as there is an ACeDB community. As far as we can see, there is little sharing of schemas or other database software even among groups solving seemingly similar problems. For example, the human (Pearson, 1991), mouse (Nadeau et al., 1994), and Drosophila (FlyBase, 1994) community databases all use SYBASE, but, again, apparently use completely different schemas. As another example, the Lawrence Livermore, Los Alamos, and Utah genome centers use SYBASE for projects involving mapping and sequencing of human chromosomes, but apparently use completely different schemas. This observation is not intended as a criticism of these excellent projects—and indeed when one looks at each project in detail, important differences in requirements emerge. It is merely an empirical observation as to the manner in which SYBASE is used within the community.

Despite its wide acceptance, SYBASE is not suitable for use as the DBMS component in a modular, component-based genome informatics systems. The main reasons are:

No semantic data model and no example schemas The fact that biologists can understand and manipulate ACeDB schemas is strong evidence that a semantic data model is essential in this community. One way SYBASE could compensate for the lack of a semantic model would be to provide a comprehensive collection of example schemas. To our knowledge, no such collection has ever been assembled. Given that so many groups are using SYBASE, the fact that no one has done this suggests that SYBASE users do not believe this would be productive. A complementary approach would be to try to repair some of the deficiencies of the relational model vis-à-vis genome informatics systems by automatically translating from a more intuitive data model to the relational model. This is the approach taken in the Genera system (Letovsky and Berlyn, 1994) and in the Object-Protocol Model (OPM) (Chen and Markowitz, 1994).

No built-in support for biological types and operations or for managing workflow data Adding this to SYBASE would be even more difficult than adding it to

ACeDB.

Not free Licensing the DBMS may cost a few thousand dollars, making it difficult for new laboratories to try it out.

Genome Topographer (Cozza et al., 1994a) is another important system that is potentially relevant to this project. Our knowledge of this system is limited; we have had no direct experience with it over the past year, and we are unable to find any publications describing it except for the previously cited Cold Spring Harbor abstract. The best publicly available information on Genome Topographer seems to be its WWW page (Cozza et al., 1994b).

Based on the information available to us, it appears that the system is aiming to achieve a suite of capabilities similar to ACeDB's. The implementation, of course, is more modern: it is constructed in Smalltalk and makes use of the commercial GemStone object-oriented DBMS (Butterworth et al., 1991). We have every reason to expect that Genome Topographer will be successful for its targeted purpose.

Relative to the goals of this project, Genome Topographer appears not to be a suitable DBMS component for the following reasons:

Not component-oriented Same as ACeDB.

No built-in support for managing historical data Same as ACeDB.

Not free As with SYBASE, licensing fees must be paid, making it difficult for new laboratories to try it out.

To summarize, these systems, in their current form, cannot serve as the broadly applicable DBMS component that the community needs.

C Preliminary Results

The Informatics Core of the Genome Center has been operating genome informatics systems for mouse genetic mapping (Dietrich et al., 1993, Copeland et al., 1993, Miller et al., 1994) and human physical mapping for over three years. During that period we have become convinced of the utility of modular, plug-and-play, component-based genome informatics systems for the reasons discussed in section B.1 (*Why We Need a Component-Oriented DBMS*).

The first system we built, almost three-and-a-half years ago, was monolithic. The entire system was implemented as a single Unix process. The database, called MapBase (Goodman et al., 1994), was implemented in C++ on top of the ObjectStore C++-based object-oriented DBMS (Lamb et al., 1991). User interfaces were implemented in C using the Xt X-windows toolkit (Flanagan, 1992). There were no application programs yet implemented for this early system, but the plan was to implement such programs in C++ and link them into the monolithic process. A main feature of this architecture was that all code had direct, intimate access to the database by invoking C++ methods on database objects.

This architecture failed before it was even released! When attempting to integrate the database and user interface code for the first time, we discovered that the Xt library used

for the user interface was incompatible with the version of Unix required by ObjectStore. This forced us to give up the monolithic architecture. We had to split the database and user interface code into separate processes, run these processes on different machines, and devise a communication scheme for passing information between these processes over the network. This “screw up” was the first step down a path that has led us to our present modular, component-based system. Years later, we are convinced that this path was the right one, and that the component-based systems we are building now are easier to develop, operate, and modify as technology and user needs evolve.

MapBase was designed primarily to support the Genome Center’s mouse genetic mapping project and has done so for approximately 3 years. When the Genome Center started its human physical-mapping project, we attempted to extend MapBase for this purpose. This proved to be nettlesome due to several limitations in the MapBase design. Most importantly, the MapBase query language and processing model is organized around the assumption that users are primarily interested in one kind of reagent, namely STS markers. This assumption, though valid for genetic mapping, is incorrect for STS-content physical mapping in which users have great interest in two kinds of reagents—STS markers and YAC (yeast artificial chromosomes) clones. We felt it unwise to forcibly retrofit this capability into MapBase, and chose instead to design and implement a complete new DBMS component, called LabBase (Rozen et al., 1994).

LabBase is a general purpose, though domain-specific, DBMS designed for managing laboratory workflow and data. Like MapBase, this new system is implemented in C++ on top of the ObjectStore object-oriented DBMS.

LabBase is designed to store data for laboratory information systems. A laboratory information system records the experimental steps performed and experimental results obtained during the operation of a laboratory production line. Other examples of laboratory information systems are described in (Kerlavage et al., 1993, Clark et al., 1994). Laboratory information systems share much in common with classical information systems: in particular, the database component must provide a central repository of carefully administered, mission-critical data, and must integrate the operations of diverse software and human agents. Laboratory information systems typically involve numerous components in addition to the database component, as detailed in section B.1 (*Why We Need a Component-Oriented DBMS*).

A term commonly used in conjunction with the database component of laboratory information systems is “automated laboratory notebook”. The database is notebook-like because all experimental results are recorded in it, along with information about the experimental step itself, such as when it was performed and by whom. LabBase can be thought of as a generic automated laboratory notebook.

LabBase’s data model is based on the notions of *materials* and *steps*. All data about a material is entered in the context of a series of experimental steps. This series is called the material’s *step history*. Each step is a mapping from *tags* (which are essentially attribute names) to *values*. Each tag is associated with a unique type. Only values that conform to the type of a tag can be associated with it in the database.

Data can be viewed as a static attribute of a material by requesting the value associated with the most recent instance of a particular tag in the history of the material. Data can also

be viewed in historical context by requesting all steps associated with one or more materials. Queries about step histories are useful in refining experimental protocols. For example, one could pose a query to find out how often some expensive operation is being performed on erroneous data. This query would access step histories to look for correction steps that occur after the expensive operation in question. LabBase queries can freely intermingle both static and historical aspects.

The LabBase query language is a non-recursive logic-programming language (basically a non-recursive datalog), currently without rules. The language is intended for use as an application program interface and for occasional data-dredging queries posed by programmers. End-user queries are usually accommodated by forms-based interfaces so we do not require the language to be end-user-friendly. LabBase uses an easy-to-implement, bottom-up evaluation of queries, which seems adequate to our needs for the near future.

Materials and steps have *kinds*, and these kinds, together with tags, can be used as predicates when querying LabBase. Additional built-in predicates provide negation, disjunction, aggregates, boolean comparison operators, and so forth.

An example of a query that focuses on the most recent results relating to materials is the following, which finds the most recent DNA sequence associated with each STS marker:

```
marker(M),dna_sequence(M,S).
```

In evaluating this query, **M** is bound to each marker in the database, and, for each marker, *m*, **S** is bound to *m*'s most recently entered sequence. (More than one DNA sequence could be associated with *m* if its DNA sequence is re-read because experimental results suggest that the initial DNA sequence is incorrect.) The predicate **marker** is, in addition, a material kind, and the predicate **dna_sequence** is a tag. For any material kind, κ , $\kappa(X)$ is true of all materials, *X*, of kind κ . Similarly, for a tag, τ , and a material, μ , the predicate $\tau(\mu, V)$ is true for that value, *V*, that is the most recent value associated with τ in μ 's step history.

An example of a query that focuses on the history of steps associated with a material is the following, which finds all the DNA sequence steps (the steps that read the marker's DNA sequence) associated with STS marker D1118:

```
marker_id(M,'D1118'),all_steps(M,S),dna_sequence_step(S).
```

In this query, **M** becomes bound to the marker with id D1118, **S** is successively bound to each step in D1118's step history, and **dna_sequence_step** restricts the results of the query to steps that are of kind **dna_sequence_step**.

As a final example, the following query finds all STS markers and YACs that have been individually tested (i.e. "verified") against each other, and for which the marker was detected in the YAC.

```
marker(M),yac(Y),verify_yac_outcome(M,Y,Outcome),Outcome > 0.
```

Figure 1 represents the architecture of LabBase and its clients. The two processes labeled **lbserv** and **lback**, together with the ObjectStore database, constitute LabBase, and the various clients are to the left.

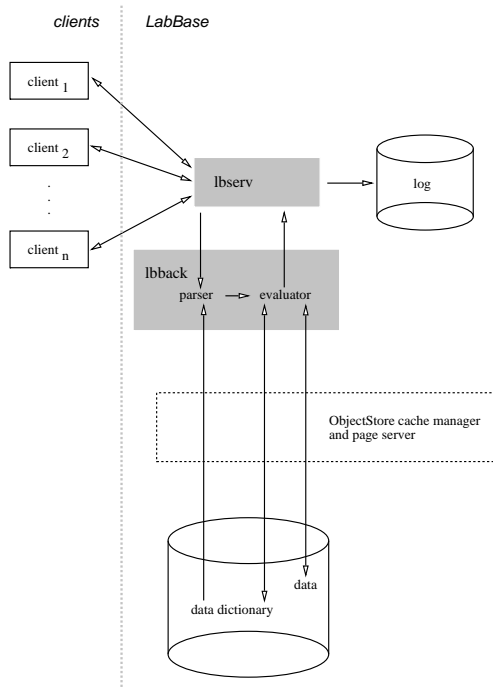


Figure 1: LabBase Architecture.

The `lbserv` process is written in perl, and manages connections from multiple clients. It can buffer partial queries from several clients. Once a complete query is available, `lbserv` forwards it to `lback`, which executes it. `lbserv` also logs all transactions to a logical archive log. (By “logical” we mean that the log contains statements in the LabBase query language rather than page images, and by “archive” we mean that the log can be used to reconstruct the state of the database in the event of a disk-media failure or a dirty software failure that corrupts the database.)

The `lback` process executes as an ObjectStore client written in C++ that implements the LabBase data definition facility and query language. Queries received from `lbserv` are first parsed (with the help of schema information stored in the data dictionary), and then executed by the evaluator. The evaluator (which requires information from the data dictionary) stores and retrieves database data, and can also update the data dictionary.

Many of the clients communicate with LabBase using a perl API library developed for the purpose. The library presents a set-at-a-time interface to perl clients, with all rows being stored in a single perl array. Each row can be transformed into an associative array (i.e. a hash table) implementing a partial map from variable identifier to binding.

In organizing the `lback` code we have taken care to make it easy to add new built-in predicates, and it is also possible to add new types that can be associated with tags.

As the first step in making LabBase independent of ObjectStore, we have rewritten LabBase’s step-storage routines to use the BSD Unix `db` library (Seltzer and Yigit, 1991). The result is a version of LabBase that stores about 90% of its data using the `db` library. LabBase performance in this version is similar to that of the ObjectStore-only version, and

its code size is about 5% larger than the ObjectStore-only version (which is about 8,500 lines of C++ code). We expect a version using only the db library to require about 20% more code than the ObjectStore version. We plan to make such a version available before the start of the proposed work, and to use the tp library (Seltzer and Olson, 1992) to provide transactions.

D Research Plan

D.1 Strategy

The key strategy in our research plan is to always have working code released to the public and to release enhancements frequently. The goal is to receive early feedback from users, and avoid the costly mistakes that often result when designs are not validated early in the crucible of actual use.

Our main development platform will be BSD/386 Unix running on Intel-based workstations. BSD/386 is a commercial port of the final version of Berkeley Unix, BSD 4.4 Lite, by Berkeley Software Design, Inc. (BSDI). It is as close to a “generic” Unix as one can get, and is POSIX-compliant. By developing software on this platform, we are better assured that it will port easily to all other major Unix platforms.

Before releasing software to the public, we will recompile and test the software on all Unix platforms available in the Genome Center, namely SunOS, Sun Solaris, DEC Ultrix, DEC Alpha/OSF, and Macintosh AUX. We will attempt to fix portability bugs with general solutions that work on all platforms, to avoid littering the code with platform-specific `#ifdefs`. We do not have access to HP and IBM RS/6000 Unix machines. If this becomes an issue, we will solicit short term use of such machines from colleagues, or from people who wish to use our software on these platforms.

We informally divide the software to be produced into two categories:

- **core** The core domain-specific DBMS, which provides a semantic data model, generic DBMS amenities (i.e. query language, API, concurrent writes, transactions, archive logging) as well as support for molecular biology data types and for workflow management.
- **periphery** Peripheral software and documentation, such as APIs, data translators (e.g. to and from .ace and NCBI/ASN.1 formats), viewing and analysis tools, translated schemas and sample data sets, and schema and operational documentation.

The terms “core” and “periphery” are meant to apply to the software organization, *not* to the importance of the categories to the utility of the proposed domain-specific DBMS.

During the course of the project we will add functionality to the core software. As functionality of the core increases, we will enhance the periphery to keep step. In addition, as the project advances, we will add offerings to the periphery.

Initially, we expect the DBMS to be used in the Genome Center, at other Whitehead Institute laboratories, and at a few other sites. As the functionality of the proposed DBMS

increases, we expect it to be adopted for use in more genome informatics systems, and expect to spend more time on user support and outreach, and on enhancing the system to meet functionality and performance requirements uncovered by actual use of the system. In the early stages of the project our relationships with other Whitehead laboratories will allow us to test the DBMS in a variety of laboratory settings.

At all points we will assess the technological landscape to take advantage of new, powerful tools, and to integrate the proposed DBMS with the new technology.

D.2 Feasibility

ACeDB, Genome Topographer, and LabBase are evidence of the feasibility of producing a domain-specific DBMS with a reasonable amount of effort. Feasibility is predicated on the judicious selection of DBMS facilities to be ignored, imported, and implemented:

- We will ignore all but the most simple query optimization. The domain-specific DBMS will push selects and decide whether or not to use indexes, but we hope to avoid the more challenging tasks of join-order optimization and dynamic reclustering (as seen, for example, in sort-merge join implementations). The assumptions underlying this expectation are that main-memory sizes will continue to increase as the sizes of genomic databases increase, with the result that appropriate static clustering can maintain most hot pages in the main-memory cache. If these assumptions prove incorrect, we would either
 1. draw on available techniques to address the problem in the DBMS, or
 2. provide a port of the domain-specific features to a generic, commercial DBMS—possibly *Illustra*¹ (Stonebraker, 1993)—for those large-scale genome informatics systems that require join optimization.
- We will import existing code for storage management, access structures (indexing), concurrency control and transactions. The code we propose to use is the BSD Unix db and tp libraries. The db library provides storage management and two kinds indexes (dynamic hashing and B-tree), as well as file scans. The tp library, which is designed to work with the db library, provides complete support for database transactions, including two-phase locking and log-based recovery. As discussed in *Preliminary Results*, we have already ported the most data-intensive parts of LabBase to the db library, demonstrating the feasibility of the approach.
- We will reuse the query parser, query evaluator, data dictionary, and archive logging already implemented for LabBase (see *Preliminary Results*). These implementations will constitute the initial implementations for the proposed domain-specific DBMS, though enhancements to the query evaluator and data dictionary will be needed as the project advances.

¹Formerly known as Montage and Miró, and growing out the Postgres project

D.3 Release Schedule

We plan to be able to release versions of the proposed DBMS according to the following schedule. The items in each release are divided into the “core” and “periphery” categories defined in section D.1 (*Strategy*).

Release 0—beginning of grant period This release will incorporate all work performed before the start of the funding period, and constitutes the foundation for the funded work.

- **core** The core DBMS for this release will consist of a freely redistributable version of LabBase (i.e., one that is independent of ObjectStore). Making LabBase independent of ObjectStore will require us to finish the port of LabBase to the db library discussed in *Preliminary Results*, and will require the integration of the tp library with the most recent version of the db library.

This release will have built-in support for biology data types and for workflow management and will support addition of new built-in types and operations coded in C++. This release will also have a general query language, transactions with archive logging and performance comparable to that achieved in our current ObjectStore-based implementation.

- **periphery** This release will include the existing set-at-a-time API for perl, user documentation available on the WWW, and a set of client programs oriented toward laboratory operations.

Release 0.5—end of first half year

- **core** In this release the DBMS will be augmented with a general semantic data model. This involves enhancements to the data dictionary, so that it can represent the set of attributes associated with particular object classes, and the is-a relationships among classes. The data dictionary will also be extended to represent integrity constraints such as constraints on attribute-value ranges and constraints on inter-object references (for example, a constraint that a particular reference be bidirectional). Some enhancements will also be made to the query evaluator, to make it aware of is-a relationships among classes.

The semantic data model will be similar to the “object” part of the Object Protocol Model. Our experience with LabBase convinces us that the proposed DBMS requires explicit set, list, and record constructors that can be composed without restriction, and we will include these in the data model. We will also include a NULL for missing data (the “don’t-know” NULL), since missing data plays such a large role in genome informatics systems. We might decide to represent the so-called “not-applicable” NULL with a user-specified distinguished value.

- **periphery** As a validation of the expressiveness of the data model we will translate the standard .ace and NCBI/ASN.1 (NCBI, 1993) schemas for the proposed DBMS. This

will involve careful study of the schemas and their semantics, and the creation of well-documented schemas for the proposed DBMS that correspond the original schemas.

This release will provide APIs for Tcl/Tk, C and C++, in addition to the perl API provided in Release 0.

A data and schema browser (one client) will be constructed. To construct the browser, it will be necessary to produce consistent conventions for displaying schemas and objects. It will be possible to move between views of data and views of the schema.

Release 1—end of first year

- **core** This release will consist primarily of performance enhancements and minor functionality enhancements motivated by user experience with Release 0.5. The choice of performance enhancements will be based on user experiences, together with profiling and analysis of disk activity. Possible technical means to achieve performance enhancement include reducing data storage requirements (to make more data available in main-memory cache), adding additional static clustering and indexing options, and simple query optimization such as pushing selects.

A functionality enhancement that we will consider at this point is provision for rules, which promise to significantly reduce the conceptual complexity of queries. Although the rules might be recursive, we will postpone enhancing query evaluation to make evaluation of recursive queries more efficient.

- **periphery** The release will provide documentation enhancements based on user feedback, and our increased understanding of user requirements in this area. We will work with users to help them set up systems employing the proposed DBMS, and thereby learn what aspects of the DBMS require enhancement.

This release will also include the Mosaic form generator. The Mosaic forms will be expressed in HyperText Markup Language (HTML) generated from the database schema and database data.

This release will also support reading and writing data in one of .ace and NCBI/ASN.1 formats, and translation of an existing data set in one of the formats. Additional data sets will be released from genome informatics systems at the Genome Center.

Release 1.5—end of 1.5 years

- **core** This release will see support for explicit protocol schemas based on the ideas of the Object Protocol Model. This release will also record input/output relationships among steps. (Steps correspond approximately to protocol objects in OPM.) This release will require
 - enhancements to the data dictionary to record protocol schemas and relationships among protocol schemas and step kinds,

- enhancements to the storage representation of steps to record dependencies among particular steps or to record steps as part of particular protocol instances, and
 - enhancements to the query evaluator to enable queries that involve facts about input/output relationships, about protocols and protocol instances, and about membership of step instances in protocol instances.
- **periphery** This release will support the ability to read and write *both* .ace and NCBI/ASN.1 formats, and additional data sets. Documented sample protocol schemas will also be developed in the course of validating and testing the protocol-modeling capabilities of the core, and will be made available to users. Documentation will be augmented to discuss the new protocol-modeling functionality introduced into the core.

Release 2—end of year 2

- **core** This release will consist primarily of performance and minor functionality enhancements motivated by user experience with Release 1.5. The new code for queries dealing protocols, protocol instances, and the relationships among step and protocol instances will be profiled and optimized. The new kinds of information stored in the DBMS may require enhancements to indexing and clustering options. With the addition of protocols to the system, it may be necessary to enable the query evaluator to perform *semi-naive* evaluation of recursive queries (see (Ullman, 1988) for a discussion).
- **periphery** To keep pace with enhancements to the core software, the data and schema browser and the Mosaic forms generator will be augmented to support protocol schemas and instances the new information about steps.

Releases 3-4

- **core** Later releases will include triggers and alerters. These will allow checking of integrity constraints not directly captured in the data model, and will allow the launching of external actions—for example, the sending of an order for oligonucleotides—based on changes to the database state. If required, additional query optimization would be implemented at this stage.

If operation under MachTen or AUX/AIX (Unix versions for the Macintosh, which also run native Mac-OS applications) is not satisfactory to the user community, this release will also contain a port to the native Mac-OS. This port would be accomplished by porting the Unix db and tp libraries to the Mac-OS. Since the Mac-OS is a single-user system, only the recovery portion of the libtp library is needed.

If certain genome informatics systems require the performance or operational characteristics of a commercial DBMS, we would port the domain-specific DBMS to one or more commercial DBMSs. The most likely candidate at this time is Illustra, whose hybrid object-relational model provides much of the modeling expressiveness of a semantic data model. By the time we reach this point in the project, however, additional SQL3 (SQL3, 1993) implementations might be available, and would also be candidates. This

port will require building a schema translator, reimplementing of built-in support for biology data types and workflow management, and adaptation of the APIs.

- **periphery** As the project continues, we expect to spend more time on maintenance, enhancements required by users, outreach, training, and documentation. In some cases we will act as consultant to users of the DBMS, to help them develop appropriate schemas for their genome informatics system, and to use existing clients to streamline their data entry and reporting. In some cases we will assist in adapting viewing and analysis tools to work with the DBMS, provided the authors are willing to redistribute these tools to other users of the DBMS.

If, by this time, standard relational schemas for genome informatics systems have emerged, we will provide translators to and from these schemas.

Bibliography

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

Berners-Lee, T., Cailliau, R., Groff, J.-F., and Pollermann, B. (1992). World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy*, 2(1):52–58. Also available at URL <http://info.cern.ch/hypertext/WWW/Bibliography/Papers.html>.

Butterworth, P., Otis, A., and Stein, J. (1991). The GemStone object database management system. *Communications of the ACM*, 34(10):65–77.

Cameron, D. and Rosenblatt, B. (1991). *GNU Emacs*. O'Reilly & Associates, Inc.

Charnock-Jones, D. S. and Aparicio, S. A. J. R. (1994). DNA sequence assembly on Unix systems. In Bishop, M. J., editor, *Guide to Human Genome Computing*, pages 191–214. Academic Press.

Chen, I.-M. A. and Markowitz, V. M. (1994). The Object-Protocol Model, version 2.3. Technical Report LBL-32738, Lawrence Berkeley Laboratory, 1 Cyclotron Road, Berkeley, CA, 94720, USA. This document and others on OPM available at URL ftp://gizmo.lbl.gov/pub/DM_TOOLS/OPM/opm.html.

Clark, S. P., Evans, G. A., and Garner, H. R. (1994). Informatics and automation used in physical mapping of the genome. In Smith, D. W., editor, *Biocomputing Informatics and Genome Projects*, pages 13–49. Academic Press, Inc.

Clocksin, W. F. and Mellish, C. S. (1987). *Programming in Prolog*. Springer-Verlag.

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

Cozza, S., Reed, E. C., Salit, J., Chang, W., and Marr, T. (1994a). Genome Topographer: A next generation genome database system. In *Genome Mapping & Sequencing*. Cold Spring Harbor Laboratory, Cold Spring Harbor, New York.

Cozza, S., Salit, J., Reed, E. C., Chang, W., and Marr, T. (1994b). URL <http://www.cshl.org/gt/index-actual.html>.

Dear, S. (1993). Author's e-mail address: sd@mrc-lmb.cam.ac.uk, or postal address: MRC Laboratory of Molecular Biology, Hills Road, Cambridge CB2 2QH, U.K.

Dietrich, W., Miller, J., Katz, H., Joyce, D., Steen, R., Lincoln, S., Daly, M., Reeve, M. P., Weaver, A., Anagnostopoulos, P., Goodman, N., Dracopoli, N., and Lander, E. S. (1993). SSLP genetic map of the mouse (*Mus musculus*). In O'Brien, S., editor, *Genetic Maps*. Cold Spring Harbor Laboratory Press, sixth edition.

- Dittrich, K. R. (1986). Object-oriented database systems: The notion and the issues. In *Proc. Int'l Workshop on Object-Oriented Database Systems*, pages 2–4.
- Dunham, I., Durbin, R., Thierry-Mieg, J., and Bently, D. R. (1994). Physical mapping projects and ACEDB. In Bishop, M. J., editor, *A Guide to Human Genome Computing*, pages 111–158. Academic Press.
- Durbin, R. and Thierry-Mieg, J. (1991). A *C. elegans* database. Documentation, code and data available from anonymous ftp servers at `lirmm.lirmm.fr`, `cele.mrc-lmb.cam.ac.uk` and `ncbi.nlm.nih.gov`.
- Flanagan, D., editor (1992). *X Toolkit Intrinsics Reference Manual*, volume 5. O'Reilly & Associates, Inc., 3 edition.
- FlyBase (1994). About FlyBase. Available at URL `ftp://fly.bio.indiana.edu/flybase/About-flybase`.
- Free Software Foundation (1994). Gnu emacs. Code available in directory `ftp://prep.ai.mit.edu/pub/gnu`.
- Gleeson, T. and Hillier, L. (1991). A trace display and editing program for data from fluorescence based sequencing machines. *Nucleic Acids Research*, 19:6481–6483.
- Goldberg, A. and Robson, D. (1983). *Smalltalk-80: The Language and its Implementation*. Addison-Wesley Publishing Company.
- Goodman, N., Rozen, S., and Stein, L. (1994). Building a laboratory information system around a C++-based object-oriented dbms. In *Proceedings of the 20th International Conference on Very Large Data Bases*. Available at `ftp://genome.wi.mit.edu/pub/steve/Y1994/building.ps.Z`.
- Hull, R. and King, R. (1987). Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19:201–260.
- Internet Engineering Task Force. The http protocol. Internet Draft available at `http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html`.
- Steele, G. L., Jr. (1990). *Common LISP The Language, Second Edition*. Digital Press.
- Kerlavage, A. R., Adams, M. D., Kelly, J. C., et al. (1993). Analysis and management of data from high-throughput expressed sequence tag projects. In N.Mudge, T., Milutinovic, V., and Hunter, L., editors, *Proceedings of the 26th Annual Hawaii International Conference on System Sciences*, volume 1, pages 585–594. IEEE Computer Society Press.
- Knuth, D. E. (1989). *The T_EXbook*. Addison-Wesley Publishing Company, 15 edition.
- Lamb, C., Landis, G., Orenstein, J., and Weinreb, D. (1991). The ObjectStore database system. *Communications of the ACM*, 34(10):50–63.

- Lamport, L. (1986). *L^AT_EX: A Document Preparation System*. Addison-Wesley Publishing Company.
- Lander, E. S., Green, P., Abrahamson, J., Barlow, A., Daly, M. J., Lincoln, S. E., and Newberg, L. (1987). MAPMAKER: an interactive computer package for constructing genetic linkage maps. *Genomics*, 1(1):174–181.
- Letovsky, S. I. and Berlyn, M. B. (1994). Issues in the development of complex scientific databases. In Hunter, L., editor, *Proceedings of the 27th Annual Hawaii International Conference on System Sciences*, volume V, pages 5–14. IEEE Computer Society Press.
- Lincoln, S. E., Daly, M. J., and Lander, E. S. (1991). PRIMER: a computer program for automatically selecting PCR primers. Whitehead Institute for Biomedical Research.
- McGoveran, D. and Date, C. J. (1992). *A Guide to SYBASE and SQL Server*. Addison-Wesley.
- Miller, J., Dietrich, W., Steen, R., Joyce, D., Merchant, M., Wessel, M., Damron, D., Nahf, R., Stein, L., Dredge, R., Marquis, A., Daly, M., Reeve, M. P., Goodman, N., Lord, C., Montague, C., Prins, J.-B., Todd, J., and Lander, E. S. (1994). SSLP genetic linkage map of the mouse. In Lyon, M. F. and Searle, A. G., editors, *Genetic Variants and Strains of the Laboratory Mouse*. Oxford University Press, third edition.
- Nadeau, J. H., Davisson, M. T., Eppig, J. T., Manly, K., Mobratten, L. E., et al. (1994). Mouse genome informatics project. In *Genome Mapping & Sequencing*. Cold Spring Harbor Laboratory, Cold Spring Harbor, New York.
- NCBI (1993). *NCBI Software Development ToolKit, Version 1.8*. Available as `ftp://ncbi.nlm.nih.gov/toolbox/ncbi_tools/newdoc/sdk.doc.Z`.
- NCSA (1993). NCSA Mosaic documentation. Available at URL `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/mosaic-docs.html`.
- Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company.
- Pearson, P. L. (1991). The Genome Data Base (GDB)—a human gene mapping repository. *Nucleic Acids Research*, 19:2237–2239.
- Pearson, W. B. and Lipman, D. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Science*, 85(8):2444–2448.
- Rozen, S., Stein, L., and Goodman, N. (1994). Constructing a domain-specific DBMS using a persistent object system. In *Sixth International Workshop on Persistent Object Systems*. In press.
- Searls, D. B. (1994). Personal communication.
- Seltzer, M. and Olson, M. (1992). LIBTP: Portable, modular transactions for UNIX. In *USENIX Winter 1992 Technical Conference*.

Seltzer, M. and Yigit, O. (1991). A new hashing package for UNIX. In *USENIX Winter 1991 Technical Conference*.

SQL3 (1993). ISO and ANSI SQL3 working draft.

Staden, R. and Dear, S. (1991). A sequence assembly and editing program for efficient management of large projects. *Nucleic Acids Research*, 19:3907–3911.

Stein, L., Marquis, A., Dredge, E., Reeve, M. P., Daly, M., Rozen, S., and Goodman, N. (1994). Splicing UNIX into a genome mapping laboratory. In *USENIX Summer 1994 Technical Conference*, pages 221–229.

Stonebraker, M. (1993). Object-relational data base systems. Distributed with Montage (now called Illustra) sales literature.

Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press.

Wall, L. and Schwartz, R. L. (1990). *Programming perl*. O'Reilly & Associates, Inc.