March 2001

# K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources

Susan B. Davidson
*University of Pennsylvania*, susan@cis.upenn.edu

Jonathan Crabtree
*University of Pennsylvania*

Brian P. Brunk
*University of Pennsylvania*, brunkb@pcbi.upenn.edu

Jonathan Schug
*University of Pennsylvania*

Val Tannen
*University of Pennsylvania*, val@cis.upenn.edu

*See next page for additional authors*



Postprint version. Published in *IBM Systems Journal*, Volume 40, Issue 2, March 2001, pages 512-531.
Publisher URL: http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=4628447&site=ehost-live

# K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources


## Abstract

The integration of heterogeneous data sources and software systems is a major issue in the biomed ical community and several approaches have been explored: linking databases, "on-the- fly" integration through views, and integration through warehousing. In this paper we report on our experiences with two systems that were developed at the University of Pennsylvania: an integration system called K2, which has primarily been used to provide views over multiple external data sources and software systems; and a data warehouse called GUS which downloads, cleans, integrates and annotates data from multiple external data sources. Although the view and warehouse approaches each have their advantages, there is no clear "winner". Therefore, users must consider how the data is to be used, what the performance guarantees must be, and how much programmer time and expertise is available to choose the best strategy for a particular application.



## Comments

Postprint version. Published in *IBM Systems Journal*, Volume 40, Issue 2, March 2001, pages 512-531. Publisher URL: http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=4628447&site=ehost-live



## Author(s)

Susan B. Davidson, Jonathan Crabtree, Brian P. Brunk, Jonathan Schug, Val Tannen, Chris Overton, and Christian J. Stoeckert

# K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources*

Susan B. Davidson, Jonathan Crabtree, Brian Brunk,
Jonathan Schug, Val Tannen, Chris Overton and Chris Stoeckert
Center for Bioinformatics
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
{susan,crabtree,brunkb,jschug,stoeckrt}@pcbi.upenn.edu, val@cis.upenn.edu
**To appear in the IBM Systems Journal.**

December 4, 2000

**Abstract**

The integration of heterogeneous data sources and software systems is a major issue in the biomedical community and several approaches have been explored: linking databases, "on-the-fly" integration through views, and integration through warehousing. In this paper we report on our experiences with two systems that were developed at the University of Pennsylvania: an integration system called K2, which has primarily been used to provide views over multiple external data sources and software systems; and a data warehouse called GUS which downloads, cleans, integrates and annotates data from multiple external data sources. Although the view and warehouse approaches each have their advantages, there is no clear "winner". Therefore, users must consider how the data is to be used, what the performance guarantees must be, and how much programmer time and expertise is available to choose the best strategy for a particular application.

## 1 Introduction

With the recent completion of a rough draft of the human genome, finished sequence for Drosophila, *C. elegans*, and yeast (among others), and numerous other sequencing projects in progress, a vast amount of genomic data has become available for further refinement and analysis. Moving past DNA sequences, researchers are interested in the corresponding protein sequences, their structure and function. Moving past sequences entirely, researchers wish to understand the "space" and "time" dimensions of genes, for example what genes are expressed in which tissues and during what stages of development. While these and other questions can only be answered exactly by direct experimentation, very often insights can be gained by accessing the tremendous amount of genomic information that is available online.

*This research was supported in part by DOE DE-FG02-94-ER-61923 Sub 1, NSF DBI99-75206, NSF IIS90-17444, ARO DAAG55-98-1-0031, and a grant from SmithKline Beecham.

As an example, suppose a researcher seeks to discover genes involved in a multi-genic neurological disorder such as bipolar schizophrenia. High-throughput analysis of gene expression patterns yields expression profiles of several tens of thousands of potentially involved genes. Analysis of the profiles reveals hundreds of genes that represent candidate genes for the disorder. Experimentally analyzing all these candidates is prohibitively expensive. The researcher must therefore prioritize the candidates to start the search with the most promising. To do so, he accesses databases such as GenBank [7], SWISS-PROT [5] and OMIM [49] to determine which of these genes are located in human chromosomal regions associated with schizophrenia by genetic mapping, or are located in human chromosomal regions syntenic to those in model organisms (e.g. mouse or rat) where related non-human neurological diseases have been mapped. Note that since the GenBank, EMBL [6], and DDBJ [67] databases exchange data regularly (forming a "consortium"), any one of these databases could have been used in the query above.

Unfortunately, although most genomic information researchers wish to access is available online, it does not all reside in one database and in one location. Rather, it is spread over multiple data sources using a variety of data models and data formats, and presenting a variety of languages and interfaces for data retrieval. Many of the data sources are not implemented using conventional database management systems (such as relational databases), but use formatted files with specialized GUIs and retrieval packages (e.g. SRS [27] and AceDB [68]). These formats have been adopted in preference to database management systems (DBMS) for several reasons. First, the data is complex and not easy to represent in a relational DBMS. Typical structures include sequential data (lists) and deeply nested record structures. This complexity would argue for the use of object-oriented database systems, but these have not met with success because of the constant need for database restructuring [30]. For example, as new experimental techniques are discovered, new data structures are needed to record details peculiar to that technique. Second, formatted files are easily accessed from languages such as Perl and C, and a number of useful software programs exist that work with these files. Third, the files and associated retrieval packages are available for a variety of platforms.

As an example of the type of genomic data that is available online, consider the SWISS-PROT entry shown in Figure 1. Each line begins with a two-character code, which indicates the type of data contained in the line. For example, each entry is identified by an accession number AC and is timestamped by up to three dates DT: the create date is mandatory, while the sequence update and annotation update dates only appear if the sequence or annotation has been modified since the entry was created. The sequence SQ (list of amino acids) appears at the end of the entry; the rest of the core data includes citation information (bibliographical references, lines beginning with R), taxonomic data OC (description of the biological source of the protein), and database references DR (explicit links to entries in other databases: EMBL (annotated nucleotide sequence database); HSSP (homology derived secondary structure of proteins); WORMPEP (predicted proteins from the *Caenorhabditis elegans* genome sequencing project); INTERPRO, PFAM, PRINTS, PROSITE (databases of protein families and domains) among other things. Annotation information, which is obtained from publications reporting new sequence data, review articles and external experts, is mainly found in the feature table FT, keyword lines KW, and comment lines CC (which do not appear in this example due to lack of space). Note that the bibliographical references are *nested* structures; there are two references, and the RP, RC, RA and RL fields are repeated for each reference. Similarly, the feature table can be thought of as a nested structure in which each line contains a start and end position (e.g. 14 to 21) and a type of feature (e.g. NP_BIND). The entry is designed to be easily read by a human being and structured enough to be machine parsed. However, several lines still contain a certain amount of structure that could be separated out during parsing. For example, the author list is a string, which could be parsed into a list of strings so as to be able to index into the individual authors. Similarly, the taxonomic data is a string spread over several lines that could again be parsed into a list.

The heterogeneity of the data sources, together with their frequently unconventional implementation, makes accessing genomic data across multiple data sources extremely difficult. Researchers – like the one in our example who is studying bipolar schizophrenia – are therefore faced with a dilemma: What software should they use to gain access to the data? How complicated is this software relative to their (frequently limited) in-

```
ID   EF1A_CAEEL     STANDARD;      PRT;   463 AA.
AC   P53013;
DT   01-OCT-1996 (Rel. 34, Created)
DT   01-OCT-1996 (Rel. 34, Last sequence update)
DT   15-DEC-1998 (Rel. 37, Last annotation update)
DE   ELONGATION FACTOR 1-ALPHA (EF-1-ALPHA).
GN   (EFT-3 OR F31E3.5) AND R03G5.1.
OS   Caenorhabditis elegans.
OC   Eukaryota; Metazoa; Nematoda; Chromadorea; Rhabditida; Rhabditoidea;
OC   Rhabditidae; Peloderinae; Caenorhabditis.
RN   [1]
RP   SEQUENCE FROM N.A. (EFT-3).
RC   STRAIN=BRISTOL N2;
RA   Favello A.;
RL   Submitted (NOV-1995) to the EMBL/GenBank/DDBJ databases.
RN   [2]
RP   SEQUENCE FROM N.A. (R03G5.1).
RC   STRAIN=BRISTOL N2;
RA   Waterston R.;
RL   Submitted (MAR-1996) to the EMBL/GenBank/DDBJ databases.
DR   EMBL; U51994; AAA96068.1; -.
DR   EMBL; U40935; AAA81688.1; -.
DR   HSSP; P07157; 1AIP.
DR   WORMPEP; F31E3.5; CE01270.
DR   WORMPEP; R03G5.1; CE01270.
DR   INTERPRO; IPR000795; -.
DR   PFAM; PF00009; GTP_EFTU; 1.
DR   PRINTS; PR00315; ELONGATNFCT.
DR   PROSITE; PS00301; EFACTOR_GTP; 1.
KW   Elongation factor; Protein biosynthesis; GTP-binding;
KW   Multigene family.
FT   NP_BIND      14     21       GTP (BY SIMILARITY).
FT   NP_BIND      91     95       GTP (BY SIMILARITY).
FT   NP_BIND     153    156       GTP (BY SIMILARITY).
SQ   SEQUENCE   463 AA;  50668 MW;  12544AF1F17E15B7 CRC64;
           MGKEKVHINI VVIGHVDSGK STTTGHLIYK CGGIDKRTIE KFEKEAQEMG KGSFKYAWVL
           DKLKAERERG ITIDIALWKF ETAKYYITII DAPGHRDFIK NMITGTSQAD CAVLVVACGT
           GEFEAGISKN GQTREHALLA QTLGVKQLIV ACNKMDSTEP PFSEARFTEI TNEVSGFIKK
           IGYNPKAVPF VPISGFNGDN MLEVSSNMPW FKGWAVERKE GNASGKTLLE ALDSIIPPQR
           PTDRPLRLPL QDVYKIGGIG TVPVGRVETG IIKPGMVVTF APQNVTTEVK SVEMHHESLP
           EAVPGDNVGF NVKNVSVKDI RRGSVCSDSK QDPAKEARTF HAQVIIMNHP GQISNGYTPV
           LDCHTAHIAC KFNELKEKVD RRTGKKVEDF PKFLKSGDAG IVELIPTKPL CVESFTDYAP
           LGRFAVRDMR QTVAVGVIKS VEKSDGSSGK VTKSAQKAAP KKK
```

Figure 1: Sample SWISS-PROT entry

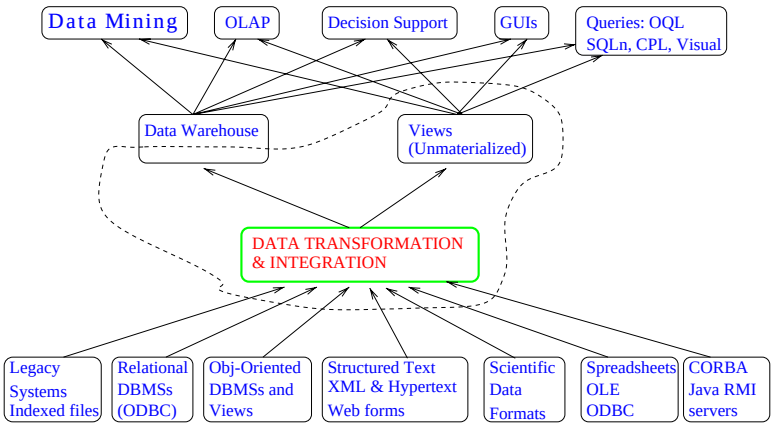


Figure 2: View and Warehouse Integration

house computer expertise? Should they leave the data where it is, or create their own specialized database? What are the tradeoffs in the approaches?

**Link-driven Federation versus Integration.** Over the past ten years, a variety of techniques have been developed within the genomic community to provide integrated access to multiple, heterogeneous data sources. Several *link driven federations* have been created, in which users start by extracting entries of interest and then hop to other related data sources via Web links that have been explicitly created by the developers of the system. SRS [27][1], LinkDB [28][2], and GeneCards [59][3] are examples of this approach. Systems to develop *view* integrations have also emerged within the community, such as the Kleisli/K2 system [13, 71] and OPM [19]. In this approach, the schemas of a collection of underlying data sources are merged to form a global schema in some common model (such as a relational, complex value or object-oriented model). Users query this global schema using a high level query language (such as SQL [1], OQL [58], or CPL [15]); the system then determines what portion of the global query can be answered by which underlying data source, ships local queries off to the underlying data sources, and combines the answers from the underlying data sources to produce an answer to the global query. These view integration systems can also be used to create an instantiation of the global schema, commonly referred to as a *warehouse*. In contrast to the view integration strategy, the global query can be answered using information in the warehouse rather than shipping off queries to the underlying data sources.

Figure 2 illustrates the connection between the view and warehouse integration strategies. At the bottom of the figure lie the multiple, heterogeneous data sources of interest. Above that sits a software layer which provides the ability to extract and integrate the underlying data sources. The dotted lines show that this integration layer can then be used for querying or as the basis for creating a data warehouse. In either the view or the warehouse strategy, a global query of the underlying data sources can be embedded in a variety of application programs such as a Web interface, shown at the top of the figure.

There are obviously tradeoffs between the link driven federation, view, and warehouse approaches [23]. The link driven federation approach is very useful for the non-expert user, since it relies almost entirely on a point and click interface. Most of the federation approaches also offer a limited retrieval language that can be quickly learned by non-technical users: For example, SRS allows users to specify logical combinations of regular expressions to index into fields of interest. There is also tremendous value in the linked connections. The approach is therefore very helpful for labs with little in-house computer expertise. However, the approach

[1] See `srs.ebi.ac.uk`.
[2] See `www.genome.ad.jp/dbget/`.
[3] See `nciarray.nci.nih.gov/cards/`.

does not scale well. When a new data source is added to the federation, connections between its entries and entries of all existing federation data sources must be added, commonly referred to as the "$N^2$" problem. Furthermore, it is often the case that if a user is interested in a join between two data sources in the federation, they must manually perform the join by clicking on each entry in the first data source and following all connections to the second data source.[4] In contrast, a join can be expressed in a single high-level query in the view or warehouse integration strategies. In general the query languages supporting view or warehouse integration approaches are much more powerful languages, and allow arbitrary restructuring of the retrieved data.

In the view or warehouse strategy, the user sees a global schema of the underlying data. To be useful, the schema should give the user the ability to connect various pieces of information. This can be done either by explicitly providing linking tables (as in the link-driven federation approach, but using some form of identifiers rather than hot links), or by providing software to compute matches between information (such as homology or similarity above some threshold for sequence entries). The schema itself can be thought of as a set of integration queries over the union of the schemas of the underlying data sources, perhaps with additional linking tables provided by the integrator.

In the remainder of this paper, we start by describing the K2/Kleisli integration system, which has been used to implement a view integration strategy in bioinformatics applications at the Penn Center for Bioinformatics (PCBI), SmithKline Beecham, and in the TAMBIS system at the University of Manchester [53]. We then discuss various practical issues associated with the warehousing approach before describing a particular warehouse called GUS (the Genomics Unified Schema). GUS forms the basis for several organism and tissue specific research projects at the University of Pennsylvania and collaborating institutions; in particular view applications have been developed in support of a *Plasmodium falciparum* database, a mouse and human gene index, and a database of genes expressed in the developing endocrine pancreas.

## 2 K2/Kleisli

K2 is the latest incarnation of a distributed query system that we have been developing over the past seven years at the University of Pennsylvania. K2 is based on many of the same principles that guided the design of Kleisli, its conceptual predecessor [13, 12, 24]. Like Kleisli, the K2 system uses a complex value model of data. This model is one in which the "collection" types, i.e., sets, lists and multisets (bags), may be arbitrarily nested along with record and variant (tagged union) types. Kleisli uses as its language the Collection Programming Language (CPL) [37], which was developed specifically for querying and transforming complex value data. Although equivalent in expressive power to SQL when restricted to querying and producing relational data, CPL uses a "comprehension"-style syntax [15] which is quite different in style from SQL. This departure from the de-facto query language standard has apparently made CPL less accessible to database professionals. Consequently the decision was made in K2 to support the more recent industry-standard query language OQL [16]. OQL uses the "select-from-where"-style syntax of SQL, but its semantics is that of comprehensions, just like CPL. K2 supports full OQL extended with variant (disjoint union) types.

The complex value data model of K2 also incorporates a new data type, that of "dictionaries". A dictionary is a function with an explicit *finite* definition domain. This allows the representation of object-oriented classes [16] as dictionaries whose domains model the class extents, i.e. sets of object identities. Dictionaries also allow a direct representation of Web-based data sources that provide information in answer to filling in field-structured query forms. K2 also differs from Kleisli in its implementation language; while Kleisli was written using Standard ML [50], K2 is implemented primarily in Java and makes use of several of the

[4] A counterexample to this is SRS, in which a linking operator is provided to retrieve linked entries to a set of entries.

```
simplified-pubmed-entry ::=
    ( abstract: <0: null, 1:string>,
      uid: <0: null, 1:long>,
      pmid: <0: null, 1:long>
      cit: (title:<0:unit, 1:{ <name:string, iso-jta:string, isbn:string, ...> } >,
            from: <journal: ..., book: ..., proc: ...>,
            authors: (names: <std: [ ... ], ml: [ string ], str: [ string ]>,
                      affil: ... )
      mesh: <0: null,
             1:{ (mp: bool, term: string,
                  qual: <0: null, 1:{ (mp: bool, subh: string) }>) }>,
      substance: <0: null,
                  1:{ (type: <nameonly: null, cas: null, ec: null>,
                              cit: <0: null, 1:string>,
                              name: string) }>, ... )
```

*() ::= record, [] ::= list, {} ::= set, <> ::= variant (tagged union),*
*<0: null, 1: string> ::= a variant that represents an optional string value*
*"..." ::= a portion of the type that has been omitted for brevity*

Figure 3: A sample K2 type that represents a simplified PubMed entry

standard protocols and APIs that are part of the "Java Platform"[5], including RMI[6] and JDBC[7].

The architecture of K2 is similar to that of a number of other view integration systems. K2 relies on a set of *data drivers*, each of which handles the low level details of communicating with a single class of underlying data sources (e.g. Sybase relational databases, Perl/shell scripts, the BLAST 2.x family of similarity search programs [4], etc.). A data driver accepts queries expressed in the query language of its underlying data source. It transmits each such query to the source for evaluation and then converts the query result into K2's internal complex value representation. For data sources that support it, this is done on a tuple-by-tuple or object-by-object basis analogous to the demand-driven tuple processing paradigm of relational databases. Data drivers are also responsible for providing K2 with data source metadata (i.e., types and schemas), which is used to type check queries.

Once a user's OQL query has been type checked, K2 must decompose it into subqueries that can be answered by the underlying data sources. Furthermore, it must rewrite the OQL query fragments, where necessary, into queries that the data sources can understand (since most will not support OQL directly.) Both of these tasks are handled by the system's *query optimization module*, which is an extensible rule-based optimizer. The K2 optimizer performs query "deforestation", i.e., the elimination of intermediate collection results. Further, it performs rewrites to group operations by data source so that the biggest subquery possible is sent to each data source. Cost-based optimization, which we are also investigating, is an alternative commonly used in commercial RDMSs. However, the distributed environment in which the system must run does not lend itself as well to accurate cost estimation. Note that any part of the query that the K2 optimizer is unable to ship to the underlying data sources will be executed by the K2 runtime system itself.

To illustrate how K2 is used, consider the following scenario in which GUS is queried in combination with NCBI's PubMed database. GUS will be discussed in greater detail in Section 4; among other things, it contains EST assemblies that represent genes.

[5] See `www.javasoft.com/j2se/`.
[6] See `www.javasoft.com/products/rmi-iiop/`.
[7] See `java.sun.com/products/jdbc/`.

By using the mapping and expression data integrated by GUS, a scientist has identified a set of EST assemblies that represent candidate genes for an inherited disorder. Some of these assemblies correspond to known genes and some do not. In order to gain further insight into the function of the "unknown" genes, the investigator wants to find publications that reference the known genes and that mention a specific protein or substance known to be affected by the disorder. Annotated bibliographic data of this kind is not part of GUS, but can be found in NCBI's PubMed database[8]; Figure 2 shows a simplified version of the K2 type that describes an entry in PubMed as provided by the NCBI Network Entrez service.

Network Entrez provides a C language API to PubMed and several other databases, including GenBank, and uses ASN.1 [39] to represent data and types. ASN.1 is a complex value data model much like that used by K2, and so the translation between the two is straightforward. A data driver for Network Entrez has been developed using its ASN.1/C API. The data driver appears in K2's OQL interface as a user-defined function that can be passed commands written using an ad-hoc syntax. For example, the following OQL statement retrieves all the substances (e.g., proteins, enzymes, etc.) associated with a single PubMed reference:

```
K2> entrez("-g 20296074 -d m -r Entrez-back.getmle.data.E.substance.E.name");
```

The result of executing the query would be echoed back as:

```
list("Heparin",
     "Complement 3d",
     "N-acetylheparin",
     "Glycoproteins",
     "Complement 9",
     "clusterin")

K2: optimized query in 0.0020 seconds.
K2: total elapsed time for request was 1.162 seconds.
```

In the preceding query, entrez is the OQL function that represents the Entrez data driver; "-g 20296074" specifies the Entrez ID of the PubMed reference; "-d m" specifies the PubMed/MEDLINE section of Entrez; and the "-r" flag gives an optional path expression that specifies which part(s) of the ASN.1 entry should be returned to K2. This syntax is difficult to remember, so we can use OQL's define directive to create a function that represents a very simple view on PubMed. In the following, "||" is OQL's string concatenation operator:

```
define get-medline-substances(pmid) as
    entrez("-g " || pmid || " -d m -r Entrez-back.getmle.data.E.substance.E.name");
```

Combining these functions with the data on genes in the GUS data warehouse, we can list the references and substances associated with an EST assembly (predicted gene) with the following OQL view functions. The first, GUS-transcript-seqs, takes as input a GUS EST assembly ID and returns the accession numbers of the individual sequences (both ESTs and mRNAs) that make up the assembly:

```
define GUS-transcript-seqs(rnaId) as
  select enaseq.source_id
  from GUS_RNASequence rs,
       GUS_NAFeature naf,
```

[8] See www.ncbi.nlm.nih.gov/entrez/query/static/overview.html.

```
      GUS_AssemblySequence aseq,
      GUS_ExternalNASequence enaseq
  where rs.rna_id = variant(1: rnaId)
  and rs.na_feature_id = naf.na_feature_id
  and naf.na_sequence_id = aseq.assembly_na_sequence_id
  and aseq.na_sequence_id = enaseq.na_sequence_id;
```

The second function, GUS-transcript-pubmed-refs, joins the relevant tables in GUS with PubMed, using the two functions that we have just defined. It calls get-medline-substances on each sequence in the EST assembly and returns a collection of records (the OQL "struct" clause), each of which contains a PubMed ID (pmid) and a list of substances. The function uses an additional mapping function, accn-to-ids, which takes as input the accession number of a sequence and returns the PubMed IDs of all the publications that reference that sequence. This function is implemented by a Perl script. Finally, note the use of OQL's "flatten" command to transform a nested collection (e.g., a set of sets) into a non-nested collection:

```
define GUS-transcript-pubmed-refs(rnaId) as
  select struct(pmid: pmid,
                substances: get-medline-substances(pmid))
  from flatten(select accn-to-ids("m " || accn)
               from GUS-transcript-seqs(rnaId) accn) pmid;
```

We can now call GUS-transcript-pubmed-refs on an assembly ID to get associated PubMed IDs and substances:

```
K2> GUS-transcript-pubmed-refs(101005);

bag((pmid: 9530155,
     substances: list("Neurotensin",
                      "Azacitidine",
                      "neuromedin N",
                      "Peptide Fragments")))
```

The preceding example can trivially be modified to retrieve MEDLINE abstracts, or to list only those EST assemblies linked to publications that also mention "neurotensin" (for example). More generally, we can incorporate data from any of the other sources for which we have data drivers. These include: metabolic and/or signaling pathway information from KEGG [41] and EcoCyc [42]; DNA sequence-related data from GenBank, GSDB [36], and dbEST [11]; organism-specific data from MGD [8] and GDB [54]; sequence similarity searches using BLAST [4]; and data from any of the databases indexed by SRS [27]. As in our example, the data sources may include scripts and application programs (e.g., BLAST), so long as an appropriate data driver is available.

Thus far we have not mentioned the object-oriented capabilities of K2. An interesting aspect of the system is that integrated views may be defined not only by OQL functions (as in our simple example), but also by user-defined object-oriented classes. A new language, K2MDL, lets users describe new classes by specifying how their extents and attributes are computed from the underlying data sources. View classes so defined can then be queried using OQL, and K2 will compose the OQL queries with the K2MDL views, producing multisource queries in its internal language; the resulting queries are then normalized and decomposed by the optimizer. In using K2 to define views, the user has a range of options. At the highest level of abstraction, he may define virtual classes that span several underlying databases. At the lowest, it is possible to use K2 to access the underlying data sources directly (as in our call to the "entrez" driver.) Either way, a user or K2 system administrator is free to provide integrated views only for selected parts of the underlying data sources, e.g. those that are best understood or most frequently used. Those parts of the underlying data sources that have yet to be integrated in some view may still be queried and joined directly.

K2MDL combines ODL (the Object Definition Language [58]) and OQL syntax. It defines the schema of a class in the view using ODL, and defines how the extent of the class and how the attributes of an object are computed using OQL. The approach is related to O2Views [26].

K2 is implemented as a multi-threaded server that can handle multiple client connections. Clients communicate with the server using either RMI-IIOP or an ad-hoc socket protocol. We have implemented a client that provides interactive command line access to the system (as shown in our examples), in addition to a set of client libraries that simplify accessing K2 from any web application that uses Java Servlets[9].

Additional information on the system and further examples of parameterized queries can be obtained from the K2 web site[10].

# 3 Issues in Warehousing

The view and warehouse strategies share the need for a common data model — relational, object-oriented, complex object, etc. — in which to represent the underlying data sources, as well as an appropriate query language —SQL, OQL [58], CPL [15] — in which to express the integrating queries. The difference between the approaches is whether there exists a separate physical copy of the integrated database, or whether the integration describes how to translate queries against a global view into queries against the underlying data sources [23]. In the warehouse approach, the integration queries are executed in advance to build a precomputed, physical copy of the integrated database. In the view approach, for each set of search parameters supplied by the user at a particular time, the integration queries are specialized to the given search parameters and executed to build the (hopefully smaller) relevant part of the integrated database as needed to satisfy the information request. This points to some obvious advantages of the view approach: it has a very low initial cost, very little maintenance cost, and the query result is always up-to-date.

The main advantage of the warehouse approach is that system performance tends to be much better (this will be illustrated in Section 5). First of all, query optimization can be performed locally, assuming that a good underlying database management system is used; second, inter-data source communication latency is eliminated. System reliability is also better since there are fewer dependencies on network connectivity; furthermore, the problem of determining whether the underlying data sources are available is avoided. (Data sources may go down, or be overloaded and temporarily unable to answer queries.) It is also much easier to enforce any inter-database constraints that have been determined in the integration [62, 70]. The most important advantage, however, is that the underlying data sources may contain many errors, and the only feasible way for the integrated database to have correct data is to keep a separate cleansed copy. Furthermore, the researcher may have additional information – or *annotations*– to add to the integrated information, which is either done by a human or is entered as a result of analysis packages guided by a human. The "added-value" of corrections and annotations stored with the integrated data gives a tremendous advantage.

However, a warehouse must be maintained as the underlying data sources change, and this raises a number of practical problems:

1. How can we detect that the underlying data sources have changed?
2. How can we automate the refresh process?
3. How can we track the origins or "provenance" of data?

**Detecting change in a data source.** Part of the problem of change detection is deciding whether a push or a pull technology should be used. In a *push* technology, users register queries with the underlying data

[9] See `www.javasoft.com/products/servlet/`.
[10] See `www.cbil.upenn.edu/K2`.

source and request explicit notification when a change occurs which matches the query; this is also known in the database literature as "continuous" (or "continual") queries [20, 47, 48, 46]. In a *pull* technology, the user periodically polls the underlying data source to see if a change of interest has occurred. Note that a push technology requires the underlying data source to be capable of processing some form of triggers, and to be willing and able to send such notification.

Genomic data sources are just beginning to offer push capabilities. For example, SWISS-PROT offers a service called "Swiss-Shop" which allows keyword-based and sequence/pattern-based requests [5]. When new sequence entries are entered which are relevant to the request, they are emailed to the requesting user. This occurs at weekly intervals as the new update files are generated. Most of the other major genomic data sources, however, do not yet offer push services.[11] Warehouse developers within the genomics community will therefore probably have to rely on pull technologies in many cases.

Another part of the problem of change detection is finding out exactly how the underlying data source has changed. In the context of genomic databases, this is complicated by the fact that updates are typically propagated in one of three ways:

1. Producing periodic new versions which can be downloaded by the user community;
2. Timestamping data entries so that users can infer what changes have occurred since they last accessed the data; and
3. Keeping a list of additions and corrections; each element of the list is a complete entry. The list of additions can be downloaded by the user community.

None of these methods precisely describes the minimal changes that have been made to the data.

As an example, suppose that a warehouse stores a portion of SWISS-PROT in a *normalized* relational database, which requires that all fields in a table be single-valued facts about the key of the table [69]. In the resulting relational schema, an entry is split over about fifteen tables. As pointed out in the introduction, the bibliographic reference field (`RN`) in an entry is not a single-valued fact about the key of an entry (`AC`) since there may be many references per entry. References must therefore be split off as a separate table. Furthermore, since a reference could be related to several different entries, it is not enough to include `AC` as a *foreign key* in the publication table referencing some tuple in the entry table; a separate table denoting a many-to-many relationship between publications and entries must be created, and the order in which the reference appears in the entry must be maintained in an attribute. The same reasoning can be applied to authors of a reference, keywords, features, and so on.

Now suppose that an update to a SWISS-PROT entry occurs. The warehouse maintainer can detect that an update has occurred since SWISS-PROT publishes a list of entries that have been modified since the last release. However, they do not say exactly how the entry has changed. The actual change may be very small; for example, adding an extra author to a reference consumes only a few characters of the new entry. If the entry is relevant to the warehouse (i.e. it is selected by the integration query), the addition of an author will only affect a few tables in the warehouse rather than all fifteen tables which represent SWISS-PROT.

Given an old and new entry, it is possible to use various DIFF algorithms to calculate minimal changes. For example, the "acediff" utility will do this for ACe databases. For data sources that export data in XML (and we believe that this will soon happen for many of the major data sources), algorithms for *ordered trees* [63, 66, 74, 75, 18] can be used.[12] However, it is not clear that the order of fields is important in the XML representation of a SWISSPROT or GenBank entry, nor is it easy to represent updates using positional information. In [45] we therefore advocate the use of a model in which value-based keys are used at every level of nesting, and in this case the DIFF algorithm becomes a simple, efficient top-down algorithm.

[11] Push capabilities may be more common in the private sector. For example, DoubleTwist (`www.doubletwist.com`) has software agents that notify users when relevant entries are added to their database(s).

[12] One such package is IBM's XMLTreeDiff, see `www.alphaWorks.ibm.com/formula/xmltreediff`.

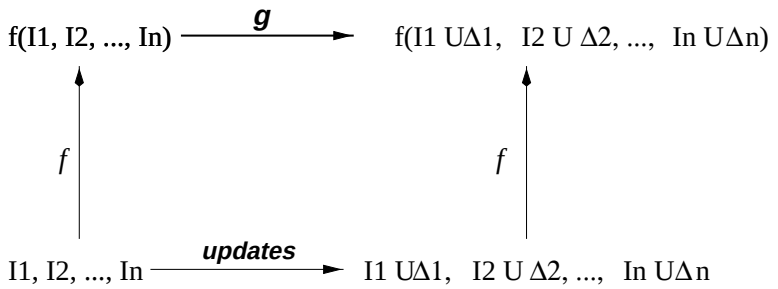


Figure 4: View maintenance problem

**Automating the refresh process.** To automate the refresh process, the portions of the warehouse defined by integrating queries must be updated (commonly called *view maintenance*) and any derived data or annotations based on the integrating query data recomputed.

The problem of view maintenance has received a lot of attention from the database community, and is illustrated in Figure 4. In this figure, $f$ represents an integration query that takes as input underlying data source instances $I_1, I_2, ...I_n$ producing the warehouse $f(I_1, I_2, ...I_n)$. The underlying data source instances are then updated, producing new underlying data source instances $I_1 \cup \Delta_1, I_2 \cup \Delta_2, ...I_n \cup \Delta_n$. Note that $\Delta_i$ may be a combination of insertions and deletions and that $\cup$ is used to denote the incorporation of the insertions and deletions into the data source instance $I_i$;[13] furthermore, it is common to represent the modification of a value by the deletion of the old value followed by the insertion of a new value. Thus the expression $I_i \cup \Delta_i$ represents all insertions, deletions as well as modifications that have been made to the i'th data source.

To produce the updated warehouse $f(I_1 \cup \Delta_1, I_2 \cup \Delta_2, ...I_n \cup \Delta_n)$, it is always possible to re-execute the integration query. However, this is very expensive so the problem is to find a query $g$ that takes as input the updates $\Delta_1, \Delta_2, ..., \Delta_n$, and possibly the original instances $I_1, I_2, ...I_n$ or the existing warehouse $f(I_1, I_2, ...I_n)$, and updates the warehouse to produce the new state. When $g$ can be written without requiring the original instances and only takes as input $\Delta_1, \Delta_2, ..., \Delta_n, f(I_1, I_2, ...I_n)$, the view is said to be *self-maintainable*.

For example, suppose that we have input (relational) data sources $R(A,B) = \{(a1, b1), (a2, b2)\}$ and $S(B,C) = \{(b1, c1), (b3, c3)\}$, and a view $V$ defined as $f(R,S) = R \bowtie S = \{(a1, b1, c1)\}$. Updates $\Delta_1 = \{(a3, b3)\}$ and $\Delta_2 = \{(b2, c2)\}$ occur to the base relations. Then $V$ can be updated by calculating $V \cup (\Delta_1 \bowtie S) \cup (R \bowtie \Delta_2) \cup (\Delta_1 \bowtie \Delta_2)$, which (assuming that $V$ is large and $\Delta_1$, $\Delta_2$ are small) is more efficient than recalculating the entire view. The view is not self-maintainable, however, since we need to access both $R$ and $S$ to calculate the changes to $V$. On the other hand, the following view $V'$ is self-maintainable: $f'(R) = \sigma_{A=a1}(R) = \{(a1, b1)\}$. When an update occurs (such as $\Delta'_1 = \{(a1, b3), (a3, b3)\}$), the updated view can be calculated by simply inserting the filtered update ({a1,b3)}) to the view. More complex view definitions can also be made self-maintainable by reasoning about functional dependencies and foreign key constraints, and storing auxiliary information at the warehouse (see [57, 61, 38] for details).

View maintenance has been extensively studied in the context of relational databases [64, 10, 9, 34, 17, 52, 55, 32, 56, 33] (see [35] for a survey), and less extensively studied in the context of object-oriented databases [29, 44], nested relational databases [43], models allowing multisets [31], and semistructured databases [65, 3, 76, 45]. However, the problem of recomputing corrections and annotations has not been studied.

**Data provenance.** Data provenance addresses the problem of tracking the origins of a piece of data [21, 2, 51]. The data may be produced by an integrating query, where components of the data come from

[13] This is similar to using the expression "3+ (-1)" to denote "3-1".

different underlying data sources. Alternatively, the data may be derived from other data in the warehouse using various data mining algorithms. For example, suppose that one form of annotation in our warehouse is to assign function to sequences based on similarity (e.g. using BLAST searches). This annotation could then be transitively inherited by other sequences. If the original annotation is determined to be incorrect through experimentation, all subsequent annotations would also have to be undone. It is therefore important to track the origins of the annotation by keeping detailed information about what information the annotation was based on. This will be discussed in further detail in the next section.

Note that data provenance is related to the problem of recomputing annotations. In our example, if some sequence annotation is changed, then any subsequent annotations based on it will need to be redone. Knowing the provenance of data could be used to determine which annotations need to be recomputed.

# 4 Data Warehousing in GUS

To take advantage of the benefits of data cleansing and annotation that are available with data warehousing, we have developed a schema called the Genomics Unified Schema (GUS) to integrate and add value to data obtained from several major sequence databases. The databases which are included in GUS thus far are GenBank/EMBL/DDBJ, dbEST and SWISS-PROT, and contain annotated nucleotide (DNA, RNA) and amino acid (protein) sequences. GUS uses a relational data model with tables to hold the nucleotide and amino acid sequences along with associated annotation.

GUS uses the central dogma of biology (DNA $\longrightarrow$ RNA $\longrightarrow$ protein) as its organizational principle. Sequence-centric entries from the external databases are mirrored within GUS, and also transformed into gene-centric entities. Thus, GUS tables hold the conceptual entities that the sequences and their annotation ultimately represent (i.e., genes), the RNA derived from those genes, and the proteins derived from those RNAs. The incoming sequence annotation may be experimentally determined or predicted via a variety of algorithms, although they are all stored in GUS as features localized as spans (intervals) or points on the underlying sequence(s) (see the `FT` "fields" in Figure 1). During the transformation into a gene-centric database, data cleansing occurs to identify erroneous annotation and mis-identified sequences. Ontologies are used to structure the annotations, in particular those referring to organisms (see the `OC` field in Figure 1). Additional computational annotation is then generated based on the newly-integrated sequences (e.g., gene/protein function.) We are also just beginning the process of manual annotation and curation, which will become increasingly important as time goes by.

**Data provenance in GUS.** The ability to track where data came from is extremely important in GUS. In addition to the common data warehouse concerns of tracking the origins of data from external sources, we are also concerned with tracking the history of computationally and manually-derived data generated in the course of warehouse construction. Genome sequencing efforts, such as the Human Genome Project, have led to the availability of large amounts of nucleotide sequence for which there is little or no annotation that is experimentally verified. Instead, predictions of gene content, gene identity, and gene function are made based on a variety of computational approaches; many of these algorithms train on data sets which themselves contain predictions. Thus one prediction is often dependent upon earlier predictions, and errors can easily be spread and compounded. A similar situation exists for EST (expressed sequence tag) sequencing projects, which identify genes expressed in various types of cells and organisms. The genes represented by the ESTs are identified through computational analysis of individual or (as in the case of GUS) assemblies of ESTs. These predictions may be confirmed, altered, or discarded as new information becomes available in the form of experimental results, literature searches, or new sequence data. Thus when we annotate genomic sequences (for gene content) and genes (for gene identity and function) we must record enough information to allow both users and our professional annotators to evaluate whether the annotation is justified given the available evidence.

The information that GUS tracks for computationally-derived annotation is: 1) the algorithm used; 2) the algorithm implementation (software version); 3) the algorithm invocation (runtime information); and 4) the algorithm parameters (values passed to the program at runtime). A table for each of these *algorithm*-associated types of information is present in GUS. The algorithm tables are not only used for tracking annotation history, but also for tracking data integration. That is, the algorithm tables together with tables describing the *external* data sources are used to record the loading of data from external sources, allowing us to precisely track the source of each tuple, including which script was used to load it. In addition to the algorithm tables, an Evidence table is used to relate specific annotations to the facts they are based on. *Fact* tables hold the data generated when algorithms are run against GUS entries.

One example of a fact table is Similarity, which stores the results of similarity searches between any two sets of sequences in the database. These similarities could then become the evidence that allows us to predict the cellular role of a particular protein. Algorithm information associated with each tuple in the Similarity fact table includes the database index used in the search thereby providing the ability to identify which version of the sequence database was searched.

It should be noted that the algorithm and evidence tables are also used to track manual annotation. In this case, the `Evidence` table will point to the tuple in GUS on which the annotator based a decision to make an assignment (e.g. confidence in a prediction, or a controlled vocabulary term) or to change a relationship (e.g. merge genes, or split an EST assembly). The algorithm tables capture the annotation software used and (more importantly) who performed the annotation and when.

Finally, any updates to the database as a result of the annotation process are tracked in *version* tables that record not only the altered tuple, but the algorithm which caused that tuple to be altered, the database transaction, and time at which it was versioned. Thus, the history of any annotation – or more generally, of any tuple – in GUS can be retrieved. This complete annotation history is useful for reconstructing past database states, both for archival purposes and also to aid in identifying and rectifying potential problems with the automated annotation process. It also allows the system to respond more gracefully to user requests for entries that are no longer in the database; instead of simply saying "entry not found", the system can instead tell the user exactly when and why it was retired from active service.

Not surprisingly, the GUS schema is quite large (over 180 tables, most of which are versioned).[14] As mentioned in Section 3, the compact representation of a SWISS-PROT entry (shown in Figure 1) when translated to a relational model results in each entry being split over about fifteen tables; the same holds true for EMBL-format GenBank and dbEST entries. As a result, just mirroring the external databases in GUS takes about fifty tables. Various tables related to integration and annotation make up the remainder.

Since the schema is large and fairly unintuitive, a Perl object layer has been added on top of the relational implementation. Note that this is similar to the strategy used in OPM [19], in which users can view and query the database as if it were an object-oriented database while the actual implementation is relational.[15] For example, using the object layer a user can view an entry structured as it is in SWISS-PROT rather than as it is stored in relational form; thus the features of an entry can be listed with a simple Perl method invocation, which the object layer translates into a join over the requisite tables. The object layer is also crucial in tracking data provenance, as it transparently handles a number of versioning and book-keeping tasks without the need for direct user intervention.

**Update management.** GUS (the schema) has been instantiated to create a comprehensive public resource of human and mouse genome annotation[16]. Since new sequences that are relevant to this database appear daily in the external data sources, our goal is to complete a cycle of annotation with the most current information available every two to three months. Although ideally GUS should be completely up-to-date

[14] See `www.allgenes.org/cgi-bin/schemaBrowser.pl` for the complete schema.
[15] This approach was used by GDB (see `gizmo.lbl.gov/opm.html`).
[16] See `www.allgenes.org`.

with respect to the external data sources, it currently seems acceptable to provide a resource which lags by a few months: For example, SWISS-PROT is a highly curated version of the protein portion of GenBank, and is not completely up-to-date[17]; however it is extensively used due to the high quality of data and annotations.

To update the database, the latest versions of external databases are downloaded along with any subsequent updates and new entries (including both new entries and modifications of existing entries). Entries from the external databases are then classified as unmodified, new, or modified based on date and version information associated with entries. Unmodified entries can be ignored, and new entries simply inserted into GUS. Modified entries, which can be detected by examining the appropriate entry field (e.g., DT line in SWISS-PROT with "Last annotation update" or an increment in the version of a GenBank accession), are more problematic. Since the complete entry is given rather than the minimal updates, the actual differences must be identified during the entry loading process. Note that since we are mirroring relevant portions of the source databases (essentially "selecting" entries of interest), calculating how GUS should be updated merely entails filtering the updates according to our selection criterion rather than a more complex function (recall the discussion of automating the refresh process in Section 3).

To detect exactly what components of a modified entry have been changed, the object layer is used to implement a simple diff algorithm (recall the discussion of change detection in Section 3). Using the accession number of the entry (which is assumed to be a key for the entry), the object layer retrieves the top level tuple for the entry. It then navigates down the nested structure of the entry by traversing relationships. Note that each table representing a component of the entry uses an internal identifier, which is used to index into the next level of relationship tables. Only when a change in value is determined is a change actually made to the database. As noted earlier, version tables are used to track all changes.

The new and modified entries are then put through an annotation pipeline[18], which integrates protein and DNA sequences and their annotations, and transforms the sequence-based entries into gene and protein-based entries. Further annotation on the integrated sequences is then performed, including functional role prediction. Updates to the annotation produced by this pipeline need to be managed in the same way as the updates from external databases.

Updates to entries from external databases may of course impact the old integrated sequences and the validity of their annotation. Thus, in each annotation cycle the latest entries are used and the annotation (from computational analysis) is freshly generated, but the old integrated sequences and their annotation must also be kept. It is important to maintain stable identifiers for genes, RNAs, and proteins so that data analyses can be compared from different update cycles of GUS. Therefore, we compare the integrated sequences generated between update cycles to pass on identifiers whenever possible at the high level of genes, RNAs and proteins. Manual annotation tied to these identifiers can then be reapplied.

To ensure that a consistent version of the database is always seen by the public, our current working model is to lock down a "production" version of the database for public consumption while doing the updates in a "development" database. When the entries in the development database are complete, the new version is released to the public. This paradigm will need to change in the near future as we begin manual annotation of GUS entries, since entry selection for manual annotation is guided by user interest unlike the computational analyses which are driven by new or updated entries. As a result, "collisions" may occur between computational and manual annotations because they are on different schedules. Changes in the schema of external data sources sometimes occurs. Relational views are used to represent the annotation associated with the sequence entries providing us with the ability to easily incorporate certain kinds of schema changes, e.g., the addition or renaming of an attribute.

[17] A new version is released quarterly, but it is difficult to determine how much of a lag there is from when the sequences first appear in GenBank.

[18] By annotation pipeline we mean a series of computational analyses where the output of an analysis is used as input for the next analysis in the series. See for example [40].

**Early days experience.** After several initial revisions the GUS schema has stabilized, and current versions of SWISS-PROT (protein) and dbEST (EST) have been loaded; loading the newest version of GenBank is in progress. We are currently working on the integration protocols to fully integrate the DNA (GenBank) and protein (SWISS-PROT) entries. This data (GenBank and EST) has been used to create a gene index which attempts to assign mRNA sequences (both literal and those computed via assembly of ESTs) to single genes along with their genomic sequences and gene predictions. The database can be queried and viewed at `www.allgenes.org`.

A number of significant gains have been realized from building the GUS warehouse. First, since we structure the data using ontologies of biological terms on entry into the database, we can query the data in much more powerful ways than are possible in the individual source databases. Second, given this structure, new sequences representing mRNA molecules are much more readily (and reliably) identified for inclusion in our gene index. A third gain is the ease with which we can predict cellular roles and in particular track the evidence for those roles given the presence of proteins (SWISS-PROT and GenBank non redundant protein databases) in GUS.

We have also found the GUS warehouse to be an effective vehicle for delivering specialized databases. Prior to developing GUS, we were maintaining several different databases, using a variety of database management systems and data models, each of which was designed for a specific function now covered in GUS. These functions include genome annotation (GAIA), gene integration (EpoDB), and generation of a gene index (DoTS)[19]; in addition, we were becoming involved in maintaining organism-specific information (such as the mouse and human gene index, a *Plasmodium falciparum* database[20], and a database of genes expressed in the developing endocrine pancreas [21]. Each of these databases is now (or will become) a function- or organism-specific view of GUS.

The issue of update and annotation management has not been completely solved, and further research will continue to better integrate the annotation between cycles using workflow approaches. This becomes especially important in order to maintain personnel intensive manual annotation and will be aided by our ability to track the history of the annotations as described previously.

# 5 Performance

We state in Section 3 that performance is often an overriding concern in choosing a warehouse-based solution over a view-based one. Exactly how much faster a warehouse query will run compared to an equivalent query in a view integration system depends on many factors, including the nature of the query and the number and types of source databases that must be accessed. Table 1 presents some relevant performance data gathered in mid-1995 using an earlier version of the K2/Kleisli system discussed in Section 2. This example illustrates the performance gains that can be realized by using a relational warehouse (first row). It also demonstrates that the performance of a view system can vary widely depending on the join evaluation strategies available to the system (e.g., semijoin versus nested loop join, rows 2-3). The join evaluation strategies available depend in turn on the capabilities of the underlying data sources; the last version of the query, using GDB and Entrez (row 4), could not make use of a semijoin, because the Entrez data driver/system is non-relational and does not support the operation. Note also that several of the queries failed to complete in this case due to network timeouts (to which the Entrez driver is more susceptible), highlighting the point that quality-of-service must sometimes be considered in addition to performance.

By way of comparison, using a more recent data set the GUS system supports a similar query, namely "return all EST assemblies that can be localized to chromosome $c$ by radiation hybrid mapping." In our current

[19] For descriptions of these databases, see `www.cbil.upenn.edu`.
[20] See `www.plasmodb.org`.
[21] See `www.cbil.upenn.edu/EPConDB`

| Chromosome: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| GSDB LJ[a] (isql) | 20 | 20 | 18 | 19 | 19 | 23 | 21 | 20 | 17 |
| GDB-GSDB SJ[b] (Kleisli) | 147 | 106 | 215 | 150 | 97 | 93 | 138 | 75 | 75 |
| GDB-GSDB NL[c] (Kleisli) | 1771 | 1508 | 2135 | 1769 | 1298 | 3033 | 1531 | 1124 | 1131 |
| GDB-Entrez NL[c] (Kleisli) | -[d] | -[d] | -[d] | 1113 | 420 | 1943 | 342 | 558 | 848 |

*a.* LJ = Local Join ("warehouse" approach). *b.* SJ = Semijoin (relational sources only).
*c.* NL = Nested Loop iteration. *d.* Query failed to complete.

Table 1: Running times in seconds for several implementations of the following query: "retrieve the official HUGO (Human Gene Organization) names, accession numbers and amino acid sequences of all known human genes mapped to chromosome *c*." The query requires both mapping data (from GDB) and sequence data (from GSDB or Entrez/GenBank). At the time these experiments were conducted, GSDB contained identical copies of the relevant tables from GDB, allowing us to treat GSDB as a warehouse with respect to this type of query. This is shown in the first row of the table, where the Sybase command-line interface "isql" was used to run the requisite SQL queries against GSDB. The remaining three rows represent different view evaluation strategies using exactly two source databases and Kleisli as the view system. Note that only the data for the first 9 chromosomes are shown and that all times except those in the last row are averages of at least 5 repetitions.

development system–an Oracle 8i database running on a 4-processor Linux machine–this query takes on average 24 seconds. Note that this query translates to a 6-table join in which two of the tables each contain more than 3 million rows, whereas the original GDB-GSDB query involved substantially less data. For this and other frequently-used queries, we have created materialized views in GUS to improve their performance. Using the appropriate materialized view for this query reduces it to a 3-table join and allows it to run in less than a second, and we expect further performance improvements when the system is properly tuned for Oracle[22].

# 6 Conclusions

Both the K2/Kleisli view and GUS warehouse strategies have proven useful for genomic applications within the Center for Bioinformatics. Kleisli was used for some time to implement several web-based, parameterized queries that were written for specific user groups. Users could query views that integrated many important on-line data sources (such as GenBank, GSDB, dbEST, GDB, SRS-indexed databases, KEGG and EcoCyc) and application programs (such as BLAST) by supplying values for parameters; the data sources and application programs were then accessed on demand to provide answers to the parameterized queries. The set of available web-based queries grew as users mailed in requests, although few people actually wrote CPL queries themselves. K2 has now supplanted Kleisli on our web site, and the list of queries has been somewhat modified. K2/Kleisli has also been extremely successful within SmithKline Beecham, and has formed the basis of the TAMBIS system at the University of Manchester [53]. Thus K2/Kleisli (and other view strategies) seem most useful in a curiosity-driven/browsing type of environment where network delays and data-source inaccessibility can be tolerated.

GUS, on the other hand, has become vital as we have moved toward projects involving "production strength" support for function- and organism-specific applications, in particular those involving annotation. In a production strength system, we need much more control over the data to guarantee its correctness. Furthermore, the added annotation is tied to the state of the input data sources, and we have found it easier to mirror that state within the warehouse than to reconstruct it based on timestamps and version information.

[22] Our current production GUS database runs under Sybase 11.9.2 and we are in the process of adding support for Oracle.

It should be noted that K2 was not used to populate GUS. There are three main reasons for this: First, it was felt that although OQL is an excellent query language, it is not well-suited to the task of large-scale data restructuring. For this type of work it is better to rely either on a high-level declarative transformation language like TSL/WOL [22] (and, as an aside, on its ability to execute the resulting transformations efficiently) or to rely on a general-purpose programming/scripting language. Second, unlike SQL, OQL does not have an explicit insert/update syntax; in OQL updates must be performed by method calls that affect the state of the database as a "side effect." Third, the data is not highly restructured or integrated when initially mirrored in GUS; most of the integration is performed in the subsequent annotation pipeline. Overall, since the manner in which GUS was to be populated was well understood and relatively straightforward, it was most efficient to write Perl scripts to perform the task. K2 will, however, be used as we incorporate databases that we cannot or do not wish to warehouse locally, for example PubMed (as illustrated in Section 2) as well as a specialized database of mouse neuroanatomy that we plan to integrate in the near future.

Implicit in the problems encountered in creating and maintaining view databases is the lack of standards and co-operation between the underlying data sources. These kinds of problems largely motivated the "standardization" approach based on the Common Object Request Broker Architecture (CORBA) proposed by the Object Management Group (OMG, see `www.omg.org`). The OMG is a consortium whose mission is to foster interoperability and portability for application integration via co-operative creation and promulgation of object-oriented standards. Underlying all OMG standards is CORBA, which describes how a network of component systems should behave in order to interoperate in a distributed environment. The Life Sciences Research Task Force (LSR[23]) has been formed within the OMG to address requirements and specifications of software components for life sciences using CORBA. Currently, the LSR has reached consensus on specifications for biomolecular sequence analyses and genome maps, and it is now up to individual data source owners or third parties to modify their data sources or to provide wrappers to their data sources so that they conform to these specifications.

However, we believe that the standardization of all genomic data sources is an unrealistic goal given their diversity, autonomy, and rapid schema changes. This is evidenced by the fact that interest in CORBA seems to have waned over the past year and to have been superceded by XML (see `www.w3.org.`). As a universal data exchange format, XML may well supplant existing formats such as EMBL and ASN.1 for biological data[24], and as such will simplify some of the lower-level driver technology that is part of K2/Kleisli and other view integration systems. There is an abundance of freely available parsers and other software for XML, and heavy industry backing of XML. The question is whether it will do more than function as an exchange format. It may, in fact, become a basis for view integration systems by using one of the query languages developed for semistructured data or XML[25, 60]. However, before it becomes a good basis for an integration system we believe that several things must happen:

1. Some kind of schema technology must be developed for XML. DTDs function as only a rough schema for XML. For example, there are no base types other than PCDATA (so the integer 198 cannot be distinguished from the string "198"), no ability to specify keys or other constraints on the schema, and the reliance on order makes representing tuples (in which the order of attributes is unimportant) tricky. The recent XMLSchema proposal [73] addresses many of these problems by providing a means for defining the structure, content and semantics of XML documents.

2. An agreement must be reached on the use of terms, or there must be a mechanism to map between terms. The discussions in this paper have sidestepped one of the most difficult parts of data and software integration: semantic integration. Semantic integration focuses on the development of shared ontologies between domains of users, and on the resolution of naming conflicts (synonyms and homonyms). In the TAMBIS project, although Kleisli was used for the low-level (syntactic) integration, a major effor of the project was to develop an ontology through which researchers could navigate to find information

[23] See `www.omg.org/homepages/lsr/mg.html`.
[24] See for example `www.ebi.ac.uk/microarray/MGED/`.

of interest. The view layer layer K2MDL in K2 aids in semantic integration by providing a means for mapping between concepts in different databases, and has proven extremely useful in the integration projects for SmithKline Beecham. For XML to be useful in data integration, either the usage of tag labels must be uniform across the community, or a semantic layer must be available.

3. A standard for XML storage must be adopted. Several storage techniques are currently being explored based on relational and object-oriented technologies; new technologies are also being considered. However, there is no agreement on what is best. Warehouse developers must currently therefore provide their own mapping layer to store the result of an integration query.

These issues are of current interest in the database research community, and within the near future we expect to see preliminary solutions.

Since it is likely that warehouses like GUS will continue to be developed, we believe that a number of practical steps should be taken by both the producers of primary data and the developers of warehouses:

1. *Develop "keyed" XML data interchange formats.* There are currently two ways of identifying a node in an XML tree: The first is to use ID attributes, which are globally unique PCDATA strings. The second is to use the ordering of subnodes to identify a position. For example, in a DOM [72] tree representation of an XML document the address of a node is given by the element position of each node on the path from the root to the given node. However, element positions change as new data is added; for example, inserting a new element after the third element changes the position of every element following it in the list. Since positions may change as updates occur, it is therefore desirable to use value-based rather than position-based identifiers. For example, in the relational model, tuples are identified by *keys*, that is, sets of attributes whose values uniquely identify a tuple and which cannot be modified. In a hierarchically structured model (as with K2/Kleisli or XML), the analog is to require that keys be specified at each level of nesting so that each node in the tree can be described by a unique path of keys from the root to that node. Note that this is different from IDs, which must be globally unique within a document rather than locally unique. XMLSchema [73] is addressing this, and several other proposals are also being made that may have some impact [14].

2. *Publish minimal changes.* Intuitively, rather than just publishing "Entry 90158697 has been modified" it would also be useful to know how it has been modified, such as "Feature X has been added to entry 90158697," where X is the value of the feature added. Given keyed hierarchical data, the changes to an entry can simply be represented as the set of paths that represent insertions, the set of paths that represent deletions, and the set of paths that represent modifications of values in the old entry. Since not everyone will want such detailed information, users should probably continue to have the choice of obtaining the newest version of a database or of obtaining the entries that have been modified.

3. *Keep track of where the data came from.* Since data in secondary databases is derived from primary sources, it is important to keep track of *where* it came from and *why* it is there. At a minimum, this implies that detailed information should be maintained about the version of the primary data source from which the data was extracted, the date on which the information was extracted, and information about the query that extracted the data. For data that was obtained through analysis packages based on data from primary sources, even more information should be maintained: the analysis performed, the input parameters, the name of the person performing the analysis, etc.

An additional problem that we have not mentioned but which has implications for creating warehouses is the ownership of data. Over the past ten years, data which were originally in the public domain have, for a variety of reasons, had increasingly restrictive licenses placed on their use. While databases such as GenBank and PubMed are still in the public domain, other databases such as SWISS-PROT, which is itself a valued-added secondary database, are placing restrictions on the use of their data. In the case of SWISS-PROT, the

restrictions are intended as a funding mechanism aimed at commercial uses rather than at non-profit uses[25]. However, if a non-profit user integrates part of SWISS-PROT into a specialized database which can then be used by commercial users, the non-profit user must provide a list of commercial users so that the licensing can be checked.[26] While such restrictions are quite reasonable given the high cost of producing high-quality data, they may present a significant barrier to instantiating those portions of a warehouse integration which draw data from primary sources with restrictive licenses.

**Acknowledgments.** K2 is the work of Jonathan Crabtree, Scott Harker and Val Tannen. GUS has been designed and developed by Brian Brunk, Jonathan Crabtree, Chris Overton, Jonathan Schug, Chris Stoeckert and the entire Computational Biology and Informatics Laboratory (CBIL). We are thankful to Peter Buneman and members of the Database and Bioinformatics group in the Department of Computer and Information Science, as well as the members of CBIL at the Center for Bioinformatics for their input on various aspects of this work.

# References

[1] *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann, 1993.

[2] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *ICDE*, pages 91–102, 1997.

[3] Serge Abiteboul, Jason Mc Hugh, Michael Rys, Vassilis Vassalos, and Janet Wiener. Incremental maintenance for materialized views over semistructured data. In *Int'l Conference on Very Large Databases (VLDB)*, pages 38–49, New York City, NY, August 1998.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[5] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Research*, 28:45–48, 2000.

[6] Wendy Baker, Alexandra van den Broek, Evelyn Camon, Pascal Hingamp, Peter Sterk, Guenter Stoesser, and Mary Ann Tuli. The EMBL Nucleotide Sequence Database. *Nucleic Acids Research*, 28(1):19–23, 2000.

[7] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, B.A. Rapp, and D. Wheeler. GenBank. *Nucleic Acids Research*, 28(1):15–18, 2000.

[8] Judith A. Blake, Janan T. Eppig, Joel E. Richardson, Muriel T. Davisson, and the Mouse Genome Database Group. The Mouse Genome Database (MGD): expanding genetic and genomic resources for the laboratory mouse. *Nucleic Acids Research*, 28(1):108–111, 2000.

[9] J.A. Blakeley, N. Coburn, and P.A. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, September 1989.

[10] J.A. Blakeley, P.-A. Larson, and F. Tomba. Efficiently updating materialized views. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 61–71, 1986.

[25] See `www.isb-sib.ch/announce`.

[26] The SWISS-PROT copyright notice also states for non-profit users: "There are no restrictions on its use by non-profit institutions as long as its content is in no way modified."

[11] M.S. Boguski, T.M. Lowe, and C.M. Tolstoshev. dbEST–database for "expressed sequence tags". *Nature Genetics*, 4(4):332–333, August 1993.

[12] P. Buneman, J. Crabtree, S.B. Davidson, C. Overton, V. Tannen, and L. Wong. BioKleisli. In S. Letovsky, editor, *Bioinformatics*. Kluwer Academic Publishers, 1998.

[13] P. Buneman, S.B. Davidson, K. Hart, C. Overton, and L. Wong. A data transformation system for biological data sources. In *Proceedings of VLDB*, pages 158–169, Sept 1995.

[14] Peter Buneman, Susan Davidson, Wenfei Fan, Carmem Hara, and WangChiew Tan. Keys for XML. Draft manuscript, 2000.

[15] Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, March 1994.

[16] R.G.G. Cattell, Douglas K. Barry, Dirk Bartels, Mark Berler, Jeff Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland, and Drew Wade. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.

[17] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *17th Int'l Conference on Very Large Data Bases (VLDB)*, pages 577–589, Barcelona, Spain, September 1991. Morgan Kaufmann.

[18] S. Chawathe and H. Garcia. Meaningful Change Detection in Structured Data. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 26–37, May 1997.

[19] I.-Min A. Chen and Victor M. Markowitz. An overview of the object-protocol model (OPM) and OPM data management tools. *Information Systems*, 20(5):393–418, 1995.

[20] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *SIGMOD*, pages 379–390, Houston, TX, May 2000.

[21] Y. Cui, J. Widom, and J. Wiener. Tracing the lineage of view data in a data warehousing environment. *TODS*, June 2000. (To appear).

[22] S.B. Davidson and A. Kosky. WOL: A language for database transformations and constraints. In *Proceedings of the International Conference of Data Engineering*, pages 55–65, April 1997.

[23] S.B. Davidson, C. Overton, and P. Buneman. Challenges in integrating biological data sources. *Journal of Computational Biology*, 2(4):557–572, Winter 1995.

[24] Susan Davidson, Christian Overton, Val Tannen, and Limsoon Wong. Biokleisli: A digital library for biomedical researchers. *Journal of Digital Libraries*, 1(1):36–53, November 1996.

[25] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. In *Proceedings of the International World Wide Web Conference (WWW8)*, Toronto, 1999.

[26] C. Souza dos Santos, S.Abiteboul, and C. Delobel. Virtual schemas and bases. In *Extending Database Technology*, 1994.

[27] Thure Etzold and Patrick Argos. SRS: An indexing and retrieval tool for flat file data libraries. *Computer Applications of Biosciences*, 9:49–57, 1993.

[28] W. Fujibuchi, S. Goto, H. Migimatsu, I. Uchiyama, A. Ogiwara, Y. Akiyama, and M. Kanehisa. DBGET/LinkDB: an integrated database retrieval system. In *Pacific Symp. Biocomputing*, pages 683–694, 1998.

[29] D. Gluche, T. Grust, C. Mainberger, and M. H. Scholl. Incremental updates for materialized OQL views. *Lecture Notes in Computer Science (LNCS)*, pages 52–66, December 1997.

[30] N. Goodman, S. Rozen, and L. Stein. Requirements for a deductive query language in the MapBase genome-mapping database. In *Proceedings of Workshop on Programming with Logic Databases, Vancouver, BC*, October 1993.

[31] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *ACM SIGMOD Conference*, pages 328–339, San Jose, California, May 1995.

[32] T. Griffin, L. Libkin, and H. Tricket. An improved algorithm for the incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):508–511, 1997.

[33] A. Gupta, V. Harinarayan, and D. Quass. Generalized projections: A powerful approach to aggregation. In *Proceedings of International Conferencce on Very Large Databases (VLDB)*, pages 358–369, Zurich, Switzerland, September 1995.

[34] A. Gupta, I. S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *ACM SIGMOD Conference*, pages 157–166, Washington, DC, May 1993.

[35] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, June 1995.

[36] C. Harger, G. Chen, A. Farmer, W. Huang, J. Inman, D. Kiphart, F. Schilkey, M.P. Skupski, and J. Weller. The Genome Sequence DataBase. *Nucleic Acids Research*, 28(1):31–32, 2000.

[37] Kyle Hart, Limsoon Wong, Chris Overton, and Peter Buneman. Using a query language to integrate biological data. In *Abstracts of 1st Meeting on the Interconnection of Molecular Biology Databases*, Stanford, August 1994.

[38] N. Huyn. Multiple-view self-maintenance in data warehousing environments. In *Intl. Conference on Very Large Databases (VLDB)*, pages 26–35, Athens, Greece, 1997.

[39] ISO. *Standard 8824. Information Processing Systems. Open Systems Interconnection. Specification of Abstraction Syntax Notation One (ASN.1)*, 1987.

[40] LC Bailey Jr., S Fischer, J Schug, J Crabtree, M Gibson, and GC Overton. Gaia: Framework annotation of genomic sequence. *Genome Research*, 8(3):234–250, 1998.

[41] M. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 28(1):29–34, 2000.

[42] Peter D. Karp, Monica Riley, Milton Saier, Ian T. Paulsen, Suzanne M. Paley, and Alida Pellegrini-Toole. The EcoCyc and MetaCyc databases. *Nucleic Acids Research*, 28(1):56–59, 2000.

[43] A. Kawaguchi, D.F. Lieuwen, I.S. Mumick, and K.A. Ross. Implementing incremental view maintenance in nested data models. In *Proceedings of International Workshop on Database Programming Languages*, pages 202–221, Estes Park, Colorado, August 1997.

[44] H.A. Kuno and E.A. Rundensteiner. Incremental maintenance of materialized object-oriented views in multiview: Strategies and performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), 1998.

[45] H. Liefke and S. Davidson. View Maintenance for Hierarchical Semistructured Data. In *DaWaK'00*, London, England, September 2000.

[46] L. Liu, C. Pu, R. Barga, and T. Zhou. Continuous queries over append-only databases. In *ACM SIGMOD Conference*, pages 321–330, 1992.

[47] L. Liu, C. Pu, R. Barga, and T. Zhou. Differential evaluation of continual queries. In *ICDCS*, pages 458–465, 1996.

[48] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *TKDE*, 11(4):610–628, 1999.

[49] V.A. McKusick. *Mendelian Inheritance in Man. Catalogs of Human Genes and Genetic Disorders.* Johns Hopkins University Press, 12th edition, 1998.

[50] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, 1990.

[51] P. A. Bernstein and T. Bergstraesser. Meta-Data Support for Data Transformations Using Microsoft Repository. In *IEEE Data Engineering Bulletin*, pages 9–14, 1999.

[52] R. Paige. Applications of finite differencing to database integrity constrol and query/transaction optimization. In *Proceedings of Advances in Database Theory*, pages 170–209, New York, 1984.

[53] N.W. Paton, R. Stevens, P.G. Baker, C.A. Goble, S. Bechhofer, and A. Brass. Query processing in the TAMBIS bioinformatics source integration system. In *Proc. 11th Int. Conf. on Scientific and Statistical Databases*, pages 138–147. IEEE Press, 1999.

[54] P. Pearson, N. Matheson, N Flescher, and R. J. Robbins. The GDB Human Genome Data Base Anno 1992. *Nucleic Acids Research*, 20:2201–2206, 1992.

[55] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):337–341, September 1991.

[56] D. Quass. Maintenance expressions for views with aggregation. In *Workshop on Materialized Views: Techniques and Applications*, pages 110–118, Montreal, Canada, June 1996.

[57] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Conference on Parallel and Distributed Information Systems (PDIS)*, pages 158–169, Miami Beach, USA, December 1996.

[58] R. G. G. Cattell et al., editor. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann, San Mateo, California, 1997.

[59] M. Rebhan, V. Chalifa-Caspi, J. Prilusky, and D. Lancet. GeneCards: encyclopedia for genes, proteins and diseases. Technical report, Weizmann Institute of Science, Bioinformatics Unit and Genome Center, Rehovot, Israel, 1997.

[60] J. Robie, J. Lapp, and D. Schach. XML query language (XQL). In *W3C Query Languages Workshop (QL'98)*, Boston, December 1998.

[61] K.A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 447–458, Montreal, Canada, June 1996.

[62] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer*, December 1991.

[63] S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[64] O. Shmueli and A. Itai. Maintenance of views. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 240–255, Boston, June 1984.

[65] D. Suciu. Query decomposition and view maintenance for query languages for unstructured data. In *Proceedings of International Conference on Very Large Databases (VLDB)*, pages 227–238, Bombay, India, September 1995.

[66] K. C. Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery*, 26(3):422–433, 1979.

[67] Yoshio Tateno, Satoru Miyazaki, Motonori Ota, Hideaki Sugawara, and Takashi Gojobori. DNA Data Bank of Japan (DDBJ) in collaboration with mass sequencing teams. *Nucleic Acids Research*, 28(1):24–26, 2000.

[68] Jean Thierry-Mieg and Richard Durbin. ACeDB — A C. elegans Database: Syntactic definitions for the ACeDB data base manager, 1992.

[69] Jeffrey D. Ullman. *Principles of Database and Knowledgebase Systems I.* Computer Science Press, Rockville, MD 20850, 1989.

[70] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.

[71] L. Wong. Kleisli, a functional query system. *J. Functional Programming*, 10(1):19–56, 2000.

[72] World Wide Web Consortium (W3C). *Document Object Model (DOM) Level 1 Specification*, 1998. `http://www.w3.org/TR/WD-DOM-971009`.

[73] World Wide Web Consortium (W3C). *XML Schema Part 0: Primer*, 2000. `http://www.w3.org/TR/xmlschema-0/` .

[74] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[75] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42:133–139, 1992.

[76] Y. Zhuge and H. Garcia-Molina. Graph structured views and their incremental maintenance. In *14th Int'l Conference on Data Engineering (ICDE)*, pages 116–125, Orlando, Florida, February 1998.