The GenAlg Project: Developing a New Integrating Data Model, Language, and Tool for Managing and Querying Genomic Information

Joachim Hammer and Markus Schneider
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL, 32611-6120
{jhammer,mschneid}@cise.ufl.edu

1. Introduction

Life scientists are faced with a rapidly increasing accumulation of data stored in large *genomic repositories* like EMBL, GenBank, SwissProt, and DDBJ, to name a few. The flood of genomic data, their high variety and heterogeneity in semantics, formats and access methods, their semi-structured nature, and the increasing complexity of biological applications and methods mean that many and very important challenges in biology are now challenges in computing.

Biological data management has so far focused mainly on integrating different repositories into federated databases or data warehouses. However, most of these approaches ignore an important aspect: the discrepancy between the *low-level treatment* of biological data as alphanumerical values and their conceptual *high-level* nature and usage in terms of genes, proteins, and nucleotide sequences limit the ways biologists can process this wealth of data effectively. A second but equally important problem is the resolution of *semantic differences* that exist among related or the same terms across different or even within the same genomic repository: Inconsistencies in naming, terminology, and nomenclature lead to an ambiguous concept overloading both at the structure level (e.g., different interpretations of terms such as *gene* or biological processes such as *splicing*) and the instance level (e.g., different names for the same gene). Before biological data integration can be performed successfully, new techniques for modeling, representing, and querying biological data must be developed.

The ongoing *GenAlg project* focuses on overcoming the enforced low-level treatment of biological data imposed by genomic repositories, the lacking expressiveness and limited functionality of current query languages, and the frequent lack of structure and semantics in biological data representations. Its concept differs substantially from other approaches in the literature and rests on the following two-step approach:

1. *Genomics Ontology*. Due to different goals and/or shortcomings of existing ontologies, this effort deals with the appropriate design of an ontology for genomics in order to resolve terminological, syntactic, and especially semantic differences, to disambiguate biological terms and processes, and to create a consensual communication and interface language between different research communities in genomics. Although freely usable, it is especially tailored to the needs of the Genomics Algebra.
2. *Genomics Algebra*. Based on the Genomics Ontology, this effort incorporates the conceptual design, implementation, and database integration of a new, formal data model, query language, and software tool for representing, storing, retrieving, querying, and manipulating genomic information. This *extensible* algebra provides a set of high-level *genomic data types* (*GDT*s) (e.g., *Genome*, *Gene*, *Nucleotide*) together with a comprehensive collection of appropriate *genomic operations* (e.g., *translate*, *transcribe*) for biological computation.

Section 2 discusses problems of genomic data management. In Section 3 we give an overview of our solution concepts and system architecture. Section 4 describes the Genomics Ontology. Section 5 introduces our concept of Genomics Algebra. In Section 6 we describe the integration of the Genomics Algebra with a DBMS. Section 7 draws some conclusions.

2. Problem Statement

We now examine the two biological data management problems mentioned in the introduction in detail. The first problem of *low data representation* refers to the *low-level* treatment of data in biological data sources, which is adopted in most integration efforts and which includes the problems of *low-level representation* and *low-level querying* of data. As an example for low-level representation, consider a GenBank entry. The GenBank Web interface shows a large amount of *textual* information introduced by keywords like LOCUS, DEFINITION, ACCESSION, and VERSION, etc. These textual representations often have a complicated, semi-structured format at a very low data level, and it is up to the biologists to parse these data themselves. This situation does not change even if the keywords become attributes in a relational schema. Result data are mostly only available as text files or are obtained through copy-and-paste, preventing the direct application of subsequent operations.

A consequence of the low-level representation of data is their low-level querying and analytical processing. For example, large-scale Expressed Sequence

Tag (EST) sequencing projects output thousands of uncharacterized DNA sequences. If the biologist wanted to predict the protein secondary structure for a subset of the EST sequences sharing a certain molecular function, they would need to perform four successive computational manipulations as illustrated in the following scenario. Let us assume that the molecular functions of the EST sequences have been obtained by joining the initial BLAST output with the annotated Swiss-Prot database. Let us further assume that the annotated EST data is stored in a relational table called SEQUENCE_FUNCTION where GO (Gene Ontology) terms representing various molecular functions of the gene products have been assigned to the EST sequences from the BLAST result. The relation SEQUENCE_FUNCTION has the following structure (primary key is underlined):

```
SEQUENCE_FUNCTION      (
     sequence-ID:integer,
     sequence:varchar(255),
     length:integer,
     GO_ID:varchar(16) );
```

- *Step 1* requires the execution of an SQL query to select the proper subset of nucleotide sequences on which to perform the prediction analysis. The SQL query below produces a set of nucleotide sequences representing genes having GO ID 'GO:0003724', which represents the molecular function "RNA helicase".

```
Select sequence_id, sequence
From   SEQUENCE_FUNCTION
Where  GO_ID = 'GO:0003724';
```

- In *step 2*, each nucleotide sequence in the resulting set of ESTs is examined by a coding sequence (CDS) prediction algorithm to define the most probable protein-coding segment.
- In *step 3*, using the genetic code, each predicted CDS is translated into the corresponding amino acid sequence.
- In *step 4*, each derived amino acid sequence is input to an algorithm for secondary structure prediction. The result of each structure prediction is parsed and stored for viewing or further downstream analysis.

Since the intermediate results of the above steps are typically large, most biologists store the inputs and outputs of their analysis programs in a database, which in our scenario requires the existence of three tables, PREDICTED_CDS, TRANSLATION, and PREDICTED_2D_STRUCTURE for the storage of the CDS, amino acid sequences, and 2-D secondary structures respectively.

Even this simple example illustrates the complexity and the degree of database and computational expertise that is required of the biologist wishing to carry out the analysis. As we show in Sec. 3, the same analysis, when formulated using the Genomics Algebra, can be carried out in a single expression nested inside a simple SQL statement, involving only high-level biological terms and functions.

The second problem of *data semantics* and *semantic data integration* includes the problems of concept identification and resolution as well as concept overloading. *Concept identification and resolution* refers to the problem of identifying when data contained in different data sources refer to the same object and to subsequently reconcile such conflicting information. Addressing these issues starts by identifying which abstract concepts are represented in each source. Once shared information has been identified, conflicting information can be easily located.

Consider the example of two sources having different values for an attribute that is supposed to be the same. A pitfall that genomics adds to the reconciliation process is that there may not be a correct answer. Consider that a sequence representing the same gene should be identical in two different data sources. However, there may be legitimate differences between two sources and those two genes, and it is necessary to preserve these differences in the integrated view. *Concept overloading* addresses the problem that it is often difficult to determine whether or not two abstract concepts really have the same meaning and to figure out what to do if they do not. It arises at the structure level (e.g., different interpretations of the term *gene*) and the instance level (e.g., different names for the same gene). The number of distinct concepts used in genomics by different research communities and the use of the same name to refer to multiple variants makes overcoming these conflicts challenging. Our Genomics Ontology is aimed at resolving these semantic conflicts in order to be able to design uniquely defined types and operations for our Genomics Algebra.

Based on the above discussion, we have identified the following eight computer science related problems (C1-C8) which are addressed in this project:

C1. *Semantic ambiguity of genomics terms.* A precise definition of many biological terms is missing but indispensable for a computational treatment. A special problem is the use of the same term in different semantic contexts.

C2. *Missing standards for genomic data representation.* There is no commonly accepted way for representing genomic data as is evident in the many different formats and representations in use today.

C3. *Suitability of query languages.* SQL is tailored to answer questions about alphanumerical data but unsuited for biologists asking biological questions.

C4. *Limited functionality of genomic repositories.* The possible interactions of biologists with a genomic repository are limited to the functions available in the user interface of that repository. This implies a lack of flexibility and in the ability to ask new types of queries.

C5. *Inconsistency and incompatibility of data.* The existence of different genomic repositories with respect to the same kind of biological data leads to the question whether and where similar or overlapping repositories agree and disagree with one another.

C6. *Creation of new knowledge.* The nature of stored genomic data, e.g., in flat files, semi-structured records, makes it difficult to discover and create new biological knowledge. This is mainly the case because the extraction of relevant data from heterogeneous query results and the subsequent analysis has to be performed manually without much computational support.

C7. *Low-level treatment of data.* Genomic data representations and query results are more or less collections of textual strings and numerical values and are not expressed in biological terms such as genes, proteins, and nucleotide sequences. Operations on these high-level entities do not exist.

C8. *Integration of own evaluation functions.* The ability to evaluate data in genomic repositories with external methods is insufficient. It must be possible to create, integrate, and use user-defined functions with genomics repositories in an efficient manner.

3. GenAlg Prototype System

Figure 1 depicts a conceptual overview of the GenAlg prototype system consisting of *Genomics Algebra* and underlying *DBMS*. The *Genomics Algebra* is a self-contained, high-level, and extensible *type system for genomic data* together with a comprehensive *set of operations*. For example, in the EST sequencing scenario described in Sec. 2, useful data types include `NUCLEOTIDE_SEQUENCE`, `PROTEIN_SEQUENCE`, and `2D_STRUCTURE` with appropriate attributes and methods. Furthermore, our genomics algebra may include biological operations on objects of these data types such as

predictCDS:
`NUCLEOTIDE_SEQUENCE` → `NUCLEOTIDE_SEQUENCE`

translate:
`NUCLEOTIDE_SEQUENCE` → `PROTEIN_SEQUENCE`

2D_STRUCTURE:
`PROTEIN_SEQUENCE` → `2D_STRUCTURE`

returning the possible coding sequence for a nucleotide sequence, translating a nucleotide sequence into a protein sequence, and predicting the possible secondary dimension (2D) structure for a protein sequence respectively. In terms of the computer science challenges outlined in Sec. 2, the Genomics Algebra represents a solution to C2, C3, C4, C7, and C8.

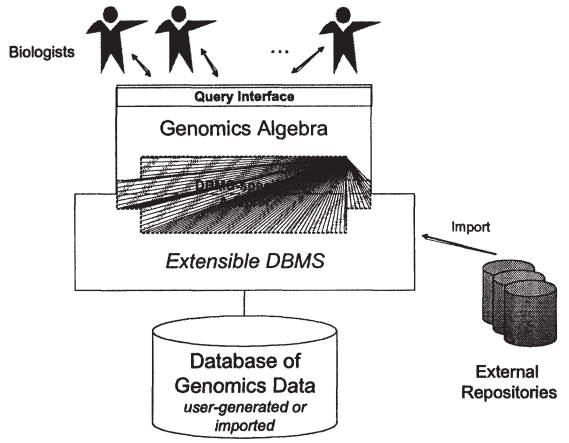

Figure 1: Conceptual architecture of the GenAlg system depicting the integration of Genomics Algebra with a DBMS.

The Genomics Algebra is derived and developed with the aid of a *Genomics Ontology* (not shown), which is a "controlled vocabulary for the description of the molecular functions, biological processes and cellular components of gene products". In the context of GenAlg, the Genomics Ontology expresses the terminological, syntactic, and semantic descriptions that are needed to disambiguate the biological terms and operations that make up the Genomics Algebra. Since one could regard the Genomics Ontology as a conceptual schema describing a subset of the genomics data, it could also serve as an aid to integrating related terms and concepts from different repositories. This will be important when using the Genomics Algebra to manage data from multiple genomics repositories. The Genomics Ontology addresses problems C1 and C5.

Although the Genomics Algebra can be implemented as a stand-alone, self-contained system for manipulating genomics data, it develops its full power only when *integrated with a database management system* (DBMS): by integrating the two, the Genomics Algebra serves as a sophisticated query system for the DBMS which in turn provides the persistent storage for the inputs and outputs of the Genomics Algebra. The underlying *Database of Genomics Data* may contain user-generated data or data imported from external repositories (shown on the lower right). Note that in order to access external data, it must be imported and merged with the existing data in the database. As mentioned above, an alternative approach would be to leave the external data in its existing repository and establish a separate connection using an adapter. Biologists access the genomics data through GenAlg's query interface which is an extension of the underlying query language of the DBMS enriched by the operations of the

Genomics Algebra. Details regarding the concepts described above as well as the benefits of the Genomics Algebra to the biological community are in [1, 2].

The integration of the Genomics Algebra with a DBMS is achieved via a *DBMS-specific adapter*, which encapsulates the knowledge about the DBMS interface and underlying (legacy) data schema and provides independence. As a result, in order to connect the Genomics Algebra to another data source, only the DBMS-specific adapter code must be changed and not the implementation of the Genomics Algebra itself.

There are several ways to implement the adapter. For example, the Genomics Algebra types and operations, which are represented as a collection of Abstract Data Types (ADTs), can be implemented using the type system and query language of the DBMS: in the case of an extensible, object-relational DBMS, ADTs can be represented using the *user-defined data type* (UDT) mechanism, and the Genomics Algebra operations are linked as *external functions* (e.g., in C, C^{++}, Java). Once implemented, the adapter is registered with the database management system at which point the UDTs and external functions become add-ons to the type system of the underlying database and can be used in SQL statements just like any of the built-in types and functions. Besides the design and implementation of the UDTs and external functions inside the adapter, a interesting research challenge will be to investigate to which degree it is possible to optimize their implementations, for example, by using as much as possible the DBMS-specific atomic data types for the implementation of the UDTs in order to take advantage of the query optimization techniques of the DBMS.

All major database vendors support UDTs and external functions and provide mechanisms to package them up for easy installation (e.g., cartridges, extenders, datablades). Encapsulating the DBMS-specific code in an adapter makes our Genomics Algebra completely independent of the software that is used to provide persistence; the Genomics Algebra can be tightly integrated with any DBMS (relational, object-oriented), as long as the DBMS is extensible.

The genomics data is stored using the containers provided by the underlying DBMS (e.g., relations, objects). An interesting strategy in the case of a Relational DBMS is to store data as binary large objects (BLOBs). In this approach, each ADT is localized inside a single BLOB rather then spread across one or more relations, which typically requires expensive joins when loading an ADT into memory. The integration of GenAlg with a DBMS addresses problem C6 since it provides biologists with a unified repository for their genomics data, which is accessible through a domain-specific interface language. Biologists are thus saved from tedious workflow activities during which they are forced to transform data from one format to another using programming languages or general-purpose query languages.

Using the sample data types and operations one could rephrase the 4-step iterative discovery process for the structure prediction example in Sec. 2 in a *single SQL expression*:

```
Select
    predict2DStructure(translate(predictCDS(n))
From
    nucleotide_sequence n
Where
    n.getMolecularFunction() = 'RNA-helicase';
```

In this example we are tacitly assuming that our Genomics Algebra has been integrated with an object-relational database management system which supports the creation of complex data types and functions which are part of the Genomics Algebra and which can be accessed using SQL-99, for example. However, the implementation and storage of the biological functions and the data types on which they operate is *hidden to the user* and *immaterial to the formulation of the query*.

This example demonstrates two points: (1) The entire structure prediction experiment, which had to be carried out in three computational steps plus one SQL query (not to mention the storing of intermediary results) can be achieved with one expression nested inside a simple SQL query. (2) Unlike current data representations, which model genomics data using atomic types such as string or integer and require knowledge about this representation in order to express queries, our approach provides biologists with a high-level representation and query interface that preserve biological concepts such as `gene` or `nucleotide sequence` and hide the low-level storage details, which depend on the implementation of the Genomics Algebra.

4. Genomics Ontology

The term ontology can be defined as a description of concepts and relationships that exist among the concepts for a particular domain of knowledge. It provides a consensual communication and interface language between people seeking a shared understanding of field-specific or interdisciplinary knowledge. For the biological sciences community, the idea and the use of the term *ontology* is relatively new. Some domain-knowledge specific and context dependent ontologies have been proposed like *EcoCyc*, *RiboWeb*, *Gene Ontology*, *Ontology for Molecular Biology* (*OMB*), *RiboWeb*, and *TAMBIS Ontology* (*TaO*). They all have different goals (e.g., database schema definition, annotation, communication, ontology-based search and query formulation), different formal frameworks (e.g., frame-based knowledge representation language, description logic), and different expressiveness. Without going into detail, although all ontologies incorporate

features that we could use for our purposes, none of them has them together. In particular, our Genomics Ontology is much more domain-neutral and context independent, and aims at covering a broader scope.

Our Genomics Ontology fulfills two important goals. First, it identifies the essential *object classes* (*data types*, *kinds*, *data categories*) in genomics and the logical connections among them. Further, it characterizes their properties by attributes. For that purpose, we learn from OMB and TaO pursuing similar goals. But as a specification method, we advocate an *Enhanced Entity-Relationship* (*EER*) model. Entity-relationship (ER) models are well known for the conceptual design of databases and due to their graphical representation relatively easy to understand, also for biologists. Figure 2 shows our current approach, which we call *Gene Onion* due to its shape and the paths ("onion skins") that can be followed from the top element to the bottom element. Object classes correspond to entity sets (e.g., *Gene*, *ORF*, *Intron*) that are shown without attributes here for space reasons. Logical connections, which are indicated by straight and dashed edges, refer to different variants of the two relationship sets *is composed of* and *is subtype of*. The enhancements of the ER model refer to special constructs, e.g., for representing *sequences* or *alternatives* (see legend).

Second, we will add *biological transformations* operating on entity sets and producing entity sets.

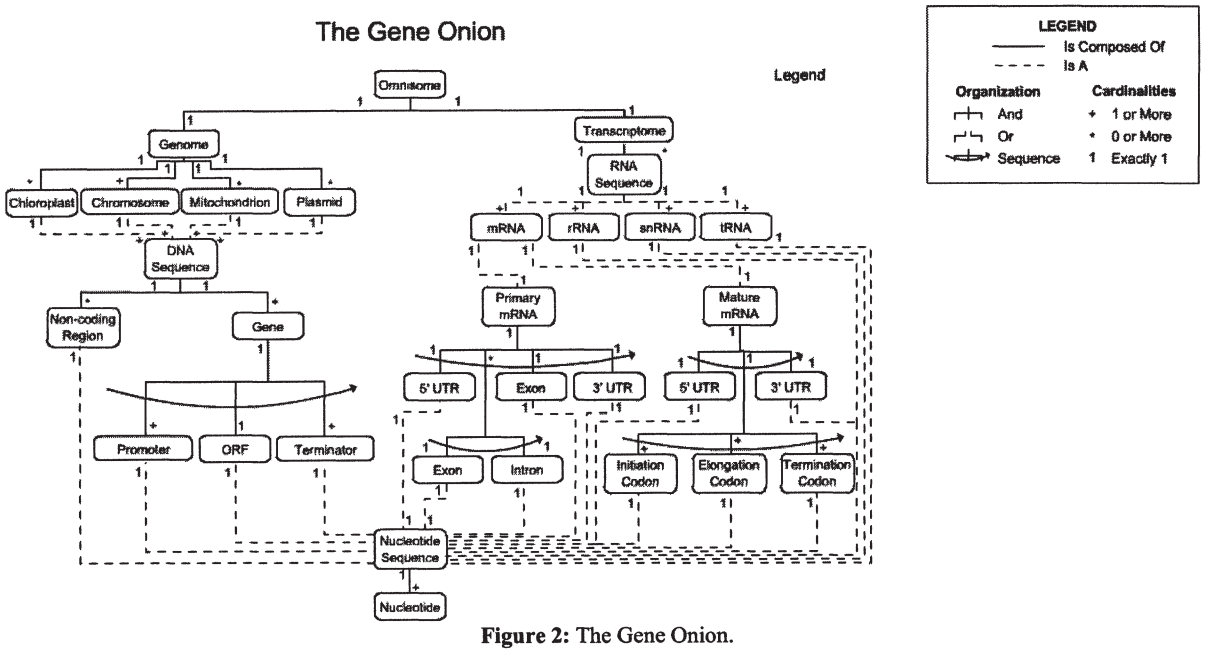

Figure 2: The Gene Onion.

In contrast to all aforementioned ontologies we attach great importance to a formal definition of all entity sets, relationship sets, and attributes. The central idea is to use mathematics, and here especially set theory and statistics, as a specification and modeling tool. This assures a clear definition of all concepts and an unambiguous communication basis between biologists of the same or different communities as well as between biologists and computer scientists for an adequate and unique design of our Genomics Algebra.

5. Genomics Algebra

Our *Genomics Algebra* is a domain-specific, many-sorted algebra incorporating a type system for biological data and serves as a communication basis and query interface for biologists. In particular, it incorporates a high-level biological terminology and is not based on the low-level concepts provided by database technology. To our knowledge, no such algebra currently exists in the field of bioinformatics.

From an abstract, conceptual point of view, the sorts of our algebra are derived from the entity sets of our Genomics Ontology. We call them *genomic data types* (*GDTs*). *Genomic operations* operate on GDTs and produce a GDT as a result. The assignment of sets to GDTs has already been performed when formally defining the semantics of the genomic entity sets in our ontology. For each operation, we need a function with corresponding domains and a codomain. It has to be formally defined how a function manipulates the domains and produces the codomain. The collection of sets for GDTs and functions for the genomic operations forms our Genomics Algebra. To illustrate the concept, we take some GDTs from Figure 2 and formulate a possible part of our algebra:

sorts

Gene, *PrimarymRNA*, *mRNA*, *Protein*

ops

transcribe:	*Gene*	→ *PrimarymRNA*
splice:	*PrimarymRNA*	→ *mRNA*
translate:	*mRNA*	→ *Protein*

This "mini algebra" contains four GDTs for genes, primary mRNA, messenger RNA, and protein as well as three operators *transcribe*, which for a given gene returns its primary transcript, *splice*, which for a given primary transcript identifies its messenger RNA, and *translate*, which for a given messenger RNA determines the corresponding protein. Hence, the high-level nomenclature of our Genomics Ontology is directly reflected in our algebra. The algebra allows us to (at least) syntactically combine different operations by (function) composition. For instance, given a gene g, we can syntactically construct the term *translate*(*splice*(*transcribe*(g))), which yields the protein determined by g.

Finding a "complete" set of GDTs and genomic operations (what does "completeness" mean in this context?) is impossible, since new biological applications can induce new data types or new operations for already existing data types. Therefore, we pursue an *extensible* approach, i.e., if necessary, the Genomics Ontology and Genomics Algebra can be extended by new types and operations. The idea is to identify new, powerful, and fundamental genomic operations that nobody has considered so far.

From an implementation perspective, the Genomics Algebra is an extensible, self-contained software package providing an implementation for a set of genomic data types and operations for biological computation. This requires sophisticated data structures for the GDTs and efficient algorithms for the genomic operations. The algebra is principally independent of a database system and can be used as a software library by a stand-alone application program. However, the algebra develops its full expressiveness and usability only if it is designed as a collection of ADTs and integrated into the query language of a DBMS (Figure 1). ADTs encapsulate their implementation so that it is hidden from the user or another software component like the DBMS. From a modeling perspective, the DBMS data model and the application-specific algebra or type system are *separated*. This enables the developer to focus on the application-specific aspects embedded in the algebra.

Data structure and algorithm design have to satisfy some constraints. A first aspect is that algorithms for different operations processing the same kind of data usually prefer different internal data representations in order to be as efficient as possible. In contrast to traditional work on algorithms, the focus is here not on finding the most efficient algorithm for each single problem (operation) together with a corresponding sophisticated data structure, but rather on considering the Genomics Algebra as a whole and on reconciling the various requirements posed by different algorithms within a single data structure for each genomic data type. Otherwise, the consequence would be enormous conversion costs between different data structures in main memory for the same data type. A second aspect is that the implementation is intended for use in a database system. Consequently, representations for genomic data types should not employ pointer data structures in main memory but should be embedded into compact storage areas which can be efficiently transferred between main memory and disk. This avoids unnecessary and high costs for packing main memory data and unpacking external data.

6. Conclusions

We have described the *GenAlg* project developing the Genomics Algebra as a new data model, language, and tool for representing, storing, retrieving, querying, and manipulating genomic information. Our integrated Genomics Algebra in conjunction with a DBMS will cause a fundamental change in the way biologists analyze genomics data and will allow them to pose questions using biological terms instead of low-level programs. Biologists should, and indeed want to invest their time being biologists, not computer scientists. Biologists are freed of the responsibility of *managing* their data and can concentrate on what they can do best, namely *analyze* the data. Due to the integration of genomic data types as *abstract data types* into databases and query languages, the Genomics Algebra can be embedded, for example, in a relational, object-relational, or object-oriented DBMS equipped with appropriate extensibility mechanisms throughout the whole system architecture. For example, in the case of a relational DBMS, our GDTs and their operations can be implemented as user-defined data types (UDTs) and functions (UDFs) inside a DBMS-specific adapter.

We have defined an initial set of genomics data types and are in the process of identifying the corresponding operations and their semantics so that we can design suitable algorithms for their implementation. Our next goal is the implementation of the Genomics Algebra and its integration into a relational DBMS.

References

[1] J. Hammer and M. Schneider, "Genomics algebra: A new, integrating data model, language, and tool for processing and querying genomic information," First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, 2003.

[2] J. Hammer and M. Schneider, "Going back to our database roots for managing genomic data," *OMICS-A Journal of Integrative Biology, Mary Ann Liebert, Inc.*, vol. 7, pp. 117-119, 2003.