OPM*QS: The Object-Protocol Model Multidatabase Query System*

(Working Document)

I-Min A. Chen†

Anthony Kosky‡

Victor M. Markowitz§

Ernest Szeto¶

Data Management Research and Development Group
Information and Computing Sciences Division
Lawrence Berkeley National Laboratory
1 Cyclotron Road, Berkeley, CA 94720

January 1996

*Issued as Technical Report LBL–38181. This work is supported by the Office of Health and Environmental Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

†Author's e-mail address: *IAChen@lbl.gov*, phone: (510) 486-7264, fax: (510) 486-4004
‡Author's e-mail address: *Anthony_Kosky@lbl.gov*, phone: (510) 486-5471, fax: (510) 486-4004
§Author's e-mail address: *VMMarkowitz@lbl.gov*, phone: (510) 486-6835, fax: (510) 486-4004
¶Author's e-mail address: *E_Szeto@lbl.gov*, phone: (510) 486-7565, fax: (510) 486-4004

Abstract

We describe in this document the OPM multidatabase query system (OPM*QS) that allows querying multiple databases that have an OPM interface, using the OPM multidatabase query language (OPM*QL).

OPM is a data model that allows specifying database schemas in terms of objects and protocols (laboratory experiments). The OPM query language, OPM_QL, allows querying databases that have been developed using the OPM data management tools or have been retrofitted with an OPM interface. OPM*QL is an extension of OPM_QL that includes constructs for querying multiple, rather than single, databases.

Acknowledgements

We want to thank Chris Fields and Carol Harger from the National Center for Genome Resources, Santa Fe, and Ken Fasman and Stan Letovsky from the Johns Hopkins School of Medicine, Baltimore, for suggesting interesting queries across GSDB and GDB and for helping us understand the semantics of GSDB and GDB.

Contents

1 Introduction

Molecular biology data are scattered among multiple data repositories, including molecular biology databases (MBDs). Although containing related data, these repositories are often isolated and are characterized by various degrees of heterogeneity: they usually represent different views (schemas) of the molecular biology domain and are implemented using different database management systems (DBMSs). Comprehensive studies of biological data often involves examining data across heterogeneous databases.

Solutions currently promoted for querying data across heterogenous MBDs involve constructing MBD *federations* or *data warehouses*, such as the Genome Topographer (Cold Spring Harbor Laboratory) [7] and the Integrated Genomic Database (German Cancer Research Institute) [13]. These solutions entail constructing a *global* view of a collection of MBDs, where definitions of the component MBDs are expressed in a common language and discrepancies between these definitions are resolved before they are integrated into a global view. For data warehouses, data from MBDs must be also loaded into a central data repository. The main problem of MBD federations and data warehouses is the complexity of constructing global views. Data warehouses have also the additional problems of not being synchronized with evolving component MBDs and of potentially very large physical sizes.

Heterogeneous MBDs can be also connected via WWW hypertext links at the level of individual data items. Data retrieval in such systems is limited to selecting a starting data item within one MBD and then following hyperlinks between data items within or across MBDs. Numerous MBDs are currently providing such links (see [12]). In addition, systems such as SRS [8] extract explicit links from existing flat file MBDs and construct indexes for both direct and reverse links allowing fast access to these MBDs.

Querying heterogenous MBDs can be achieved without constructing MBD federations or data warehouses, by organizing MBDs in a loose *multidatabase* system. We have developed a multidatabase query strategy for MBDs implemented with relational DBMSs, in the context of the Object-Protocol Model (OPM) data management tools [4]. For MBDs that have not been developed using the OPM tools, OPM views of the MBDs are first constructed using an OPM retrofitting tool. Then, the OPM query translator [6] provides facilities for browsing and querying MBDs associated with such OPM views.

Our multidatabase query strategy is based on an MBD dictionary that contains information on MBDs, including their OPM views, DBMS implementation, and links to other MBDs. Multidatabase queries are expressed in the OPM multidatabase query language (OPM*QL). A multidatabase query translator processes OPM*QL queries expressed over MBDs associated with OPM views, by

1. decomposing these queries into subqueries for individual MBDs;

2. using the OPM query translator for processing the subqueries; and

3. assembling subquery results into multidatabase query results.

Our query strategy assumes that users understand the structure and semantics of the MBDs they query. In a related project, we plan to develop an MBD Schema Library containing comprehensive documentation on MBDs and with facilities that will assist users in expressing multidatabase queries.

The multidatabase query strategy outlined above is implemented as part of an OPM Multidatabase Query System (OPM*QS). OPM*QS is currently used for querying Genome DataBase (GDB) 6.0 and Genome Sequence Data Base (GSDB) 2.0.

OPM*QS follows a strategy similar to the *Kleisli* system developed at the University of Pennsylvania [3]. Kleisli provides query access to multiple heterogeneous data-sources using the query language CPL. Each data-source requires a driver which maps CPL queries into queries against that data-source, and then maps the resulting data into the nested-relational data model. The nested relational model is entirely value based, with no direct support for object identities or constraints, but supports arbitrary nesting of record and variant type constructors with collection types (sets, bags and lists). Kleisli has a programming language approach to querying multiple databases - Kleisli does not have a concept of database schemas, and when it accesses databases via functions it does not check whether these functions are compatible with the schemas of target databases. Furthermore, unlike OPM*QS, Kleisli requires users to understand the native database schemas (e.g., the Sybase definition of GSDB) and to find the relevant links between databases at this low level.

The rest of this document is organized as follows. Section 2 contains a brief overview of OPM. The OPM multidatabase query language is presented in Section 3. The OPM multidatabase query strategy is discussed in Section 4. Section 5 examines the existing links and overlaps between the OPM schemas of GDB 6.0 and GSDB 2.0. Section 6 discussed the processing of three typical multidatabase queries expressed over GDB 6.0 and GSDB 2.0.

2 The Object-Protocol Model and Query Language

In this section, we review the main constructs of the Object-Protocol Model (OPM) and the OPM query language (OPM_QL). OPM is an object data model whose non-versioned part is closely related to other semantic [9] and object data models [1]. The version constructs of OPM follow [14]. OPM is described in [4]. OPM_QL is described in [6].

2.1 The Object-Protocol Model

Objects in OPM are uniquely identified by object identifiers (oids), are qualified by attributes, and are classified into classes. A subset of the attributes associated with a class is specified as the external object identifier (`ID`). A class can be defined as a subclass of other (super) classes, where a subclass *inherits* the attributes of its superclasses.

Attributes can be *simple* or consist of a *tuple* of simple attributes. For example, the address of a person can be modeled as a tuple attribute `address` consisting of simple attributes `street`, `city`, and `zip_code`. A simple attribute can have a single value, a set of values, or a list of values, and can be *primitive*, if it is associated with a system-provided data type, or *abstract*, if it takes values from object classes. For example, citations on genes can be modeled by associating class `Gene` with abstract attribute `citations` taking values from class `Citations`.

The attributes of an object class can be partitioned into *non-versioned* and *versioned* attributes. Non-versioned attributes represent stable object properties (e.g., social security number of a person), while versioned attributes represent evolving object properties of an object (e.g., the address of a person).

OPM supports the specification of derived attributes using derivation rules involving arithmetic expressions, aggregate functions, and attribute composition.

OPM allows specifying subclass-superclass relationships in a class hierarchy, and supports multiple inheritance in such hierarchies. A subclass is a specialization of its superclasses, and inherits all the attributes associated with its superclasses.

Protocol classes in OPM are used to model laboratory experiments. Given an input, a protocol instance (experiment) results in an output, where both input and output consist of objects. OPM supports the recursive specification (expansion) of protocols. Protocol expansion in OPM allows specifying a protocol in terms of alternative subprotocols, sequences of subprotocols, and optional protocols. A protocol class can be associated with regular as well as **input** and **output** attributes. Input and output attributes are used for specifying input and output connections between protocols.

OPM also supports two types of derived object classes: derived subclasses and derived superclasses. A derived subclass is defined as a subclass of another derived or non-derived object class with an optional derivation condition. A derived superclass is defined as a union of two or more derived or non-derived object classes.

2.2 The OPM Query Language

The OPM query language follows the SQL-like structure of query languages supported by object-oriented DBMSs such as O_2 [1]. An OPM query for a class O_i can involve local, inherited, derived and system attributes associated with O_i, and path expressions starting with these attributes. Although each OPM query is associated with a single target class, this limitation can be offset using abstract attributes (referencing other object or protocol classes) in select and condition statements. Furthermore, multi-target class queries can be constructed using derived classes and derived attributes. Other object-oriented query constructs have not been included in the current version of the OPM query language for the sake of simplicity and in order to avoid the complexity of implementing them on top of relational DBMSs.

An OPM query consists of a data manipulation statement that can be a `SELECT`, `INSERT`, `DELETE`, or `UPDATE` statement; `INSERT` `DELETE`, and `UPDATE` queries are not considered in this document. These statements can involve conditions consisting of and-or compositions of atomic comparisons of the form shown in figure 1.

An OPM `SELECT` query on target class O_i can involve local, inherited, derived and system attributes associated with O_i, as well as path expressions starting with these attributes. If class O_i is versioned (i.e., has versioned attributes), then the keywords `ALL VERSIONS` has to be specified in the `WHERE` statement in order to get *all* the versions of qualified instances; otherwise, only *default* versions are returned.

Each `SELECT` statement can have an optional condition statement and can contain an `ORDER BY` clause, which specifies an attribute whose values are used for sorting (in ascending

Atomic comparisons involving two attributes (or attribute compositions), A and B, of a class, O_i, can have one of the following forms:

1. (i) A IS NULL or (ii) A IS NOT NULL, where A is an attribute name or an attribute composition;
2. (i) $A\ \theta\ n$ or (ii) $A\ \theta\ B$, where $\theta \in \{ =, \neq, >, \geq, <, \leq, =$ ANY, $\neq$ ANY, $>$ ANY, $\geq$ ANY, $<$ ANY, $\leq$ ANY, $=$ ALL, $\neq$ ALL, $>$ ALL, $\geq$ ALL, $<$ ALL, $\leq$ ALL $\}$, n is a primitive value or a set of values, and A and B are attribute names or attribute compositions;
3. $A\ \theta\ c$, where $\theta \in \{$ MATCH, NOT MATCH $\}$, c is a character string, and A is an attribute name or an attribute composition;
4. (i) $A\ \theta\ s$ or (ii) $A\ \theta\ B$, where $\theta \in \{$ IN, NOT IN $\}$, A and B are attribute names or attribute compositions;
5. (i) $A\ \theta\ n$ or (ii) $A\ \theta\ B$, where $\theta \in \{$ CONTAINS, NOT CONTAINS $\}$, n is a primitive value or a set of values, and A and B are attribute names or attribute compositions. For example, consider class `Chromosome` with attribute `number` and class `Gene` with attribute `chromosome` taking values from class `Chromosome`. Then `Gene` can be associated with derived attribute `chromosome.number` consisting of the composition of attributes `chromosome` and `number`.

Figure 1: Atomic Comparisons in OPM Conditions

or descending order) the class instances returned by the query.

Consider class `Chromosome` with attribute `number`, and class `Gene` with attributes `name` and `chromosome`, where `chromosome` takes values from class `Chromosome`. The following OPM query expresses the selection of genes located on chromosome 13:

```
SELECT name
FROM   Gene
WHERE  chromosome.number = "13";
```

2.3 The OPM Data Management Tools

OPM is currently implemented on top of relational database management systems (DBMSs). In order to implement OPM on top of a relational DBMS, an OPM Schema Translator maps OPM schemas into DBMS-specific relational schema definitions and SQL procedures [5]. For existing relational databases, an OPM schema can be retrofitted on top of such databases using the OPM Schema Retrofitting tool. The OPM Schema Translator and the OPM Schema Retrofitting tool also generate a metadata file with information regarding the correspondence of OPM and DBMS elements.

The OPM Query Translator processes OPM queries and, based on the metadata file mentioned above, translates them into SQL queries [6].

3 The OPM Multi-Database Query Language

The Object Protocol Model multi-database query language (OPM*QL) supports ad hoc queries across multiple databases that have an OPM interface.

OPM*QL is intended to be similar to the single-database OPM query language, OPM_QL. However, unlike OPM_QL, OPM*QL supports only retrieval (i.e., "select") queries, and does not support inserting, deleting or updating multiple databases. OPM*QL also includes a number of extentions to OPM_QL necessary for dealing with multiple databases. These extensions include the ability to query multiple classes, possibly from distinct databases, constructs that allow navigation between the classes of multiple databases, and the ability to rename fields of a query in order to resolve potential naming conflicts between multiple databases.

An example of a simple OPM query over database `GSDB` is:

```
SELECT ID = GSDB:Gene, GSDBRef = GSDB:Gene.gdb_xref
FROM GSDB:Gene
WHERE GSDB:Gene.gdb_xref IS NOT NULL
AND   GSDB:Gene.name = "ACHE" ;
```

In the query above, the term "`GSDB:Gene`" refers to class `Gene` of database `GSDB`, while term "`GSDB:Gene.name`" refers to attribute `name` of class `Gene`. If a class name is unique among all the classes listed in the database directory, the database name can be omitted from a term, for example "`Gene`" can be used instead of "`GSDB:Gene`".

The structure of an OPM*QL query is defined below:

```
<query> ::= <select_stat> <from_stat> <where_stat> ';'
          | <select_stat> <from_stat> ';'
          ;
<select_stat> ::= SELECT <selected_terms>
                ;
<selected_terms> ::= <selected_terms> ',' <selected_term>
                   | <selected_term>
                   ;
<selected_term> ::= <class_spec> '.' <comp_term>
                  | <attr_id> '=' <class_spec> '.' <comp_term>
                  ;
<comp_term> ::= <composition element list> <last element>
              ;
```

```
<composition element list> ::= <null>
                             | <composition element list> <composition element>
                             ;
<last element> ::= <attr_id>
                 | <attr_id> '[' <class_id> ']'
                 | '!' <attr_id> '[' <class_id> ']'
                 ;
<composition element> ::= <attr_id> '.'
                        | <attr_id> '[' <class_id> ']'
                        | '!' <attr_id> '[' <class_id> ']'
                        ;
<from_stat> ::= FROM <class_specs>
              ;
<class_specs> ::= <class_specs> ',' <class_spec>
                | <class_spec>
                ;
<class_spec> ::= <db_id> ':' <class_id>
             | <class_id>
             ;
<where_stat> ::= WHERE <conditions>
               ;
```

The SELECT and FROM parts of a query are mandatory; the WHERE part of a query is optional. Each term (which can be an attribute name or a path expression) in the select statement can be renamed using the prefix "attr_id =". If this prefix is omitted then the attribute name of the term will be used. For example, the select statement

```
SELECT ID = GSDB:Gene.id, GSDBRef =  GSDB:Gene.gdb_xref
```

will return instances with attributes ID and GDBRef, while the select statement

```
SELECT GSDB:Gene.id, GSDB:Gene.gdb_xref
```

will return instances with attributes id and gdb_xref.

The WHERE statement is followed by a list of conditions separated by the keyword AND. The syntax for conditions is defined below:

```
<conditions> ::= <condition>
               | <conditions> AND <condition>
               ;
<condition> ::= <class_comp_term> IS NULL
              | <class_comp_term> IS NOT NULL
              | <class_comp_term> <comp op> <a primitive value>
```

```
                | <class_comp_term> <comp op> <set of values>
                | <class_comp_term> <comp op> <class_comp_term>
                | <class_comp_term> <match op> <string>
                | <class_comp_term> <in op> <set of values>
                | <class_comp_term> <in op> <class_comp_term>
                | <class_comp_term> <contains op> <a primitive value>
                | <class_comp_term> <contains op> <set of values>
                | <class_comp_term> <contains op> <class_comp_term>
                ;
<class_comp_term> ::= <class_spec> '.' <comp_term>
                      ;
<comp op> ::= '=' <any all> | '!=' <any all> | '>' <any all>
            | '>=' <any all> | '<' <any all> | '<=' <any all>
            ;
<any all> ::= <null> | ANY | ALL
            ;
<match op> ::= MATCH | NOT MATCH
             ;
<contains op> ::= CONTAINS | NOT CONTAINS
                ;
<in op> ::= IN | NOT IN
          ;
<set of values> ::= '{' <set of numbers> '}'
                  | '{' <set of strings> '}'
                  ;
<set of numbers> ::= <number>
                   | <set of numbers> ',' <number>
                   ;
<set of strings> ::= <string>
                   | <set of strings> ',' <string>
                   ;
```

(The ALL and ANY modifiers for comparison operators are not yet supported.)

Conditions can involve multiple classes, possibly from different databases. For example, the following query involves joining the Gene classes from the GSDB and HGD databases:

```
SELECT Name = GSDB:Gene.name, Reason = HGD:Gene.reason,
       Annotation = HGD:Gene.annotation
FROM GSDB:Gene, HGD:Gene
WHERE HGD:Gene.accessionID = GSDB:Gene.gdb_xref
AND   GSDB:Gene.name = "ACHE" ;
```

Note that the condition `HGD:Gene.accessionID = GSDB:Gene.gdb_xref` compares values arising from two different databases.

4 The OPM Multidatabase Query System

4.1 Multidatabase Query Strategy

The multidatabase query strategy we pursue involves the following main stages:

1. decompose OPM multidatabase queries into component OPM queries for each database in the multidatabase system, where the single-database OPM queries are evaluated using the existing OPM query translator [6];

2. retrieve and store locally the results of each single-database OPM query, and assemble the result of the multidatabase query from the results of the single-database queries.

More detailed description of OPM multidatabase query processing and illustrating examples will be given in section 5.

Currently, the multidatabase query assembly is carried out in main memory and therefore we assume that the query results do not exceed the available local main-memory. In the near future we plan to implement a simple, single user, local database management system for OPM that can be used for storing and combining the results of single database queries.

An alternative approach to evaluating multidatabase queries would be to evaluate sequentially the single-database queries, while using each query result to restrict the next query in the sequence. This approach could be more efficient for some queries and would require storing smaller intermediary query results. However, this approach is more difficult to implement (e.g., it involves a bi-directional flow of data between the OPM multidatabase query system and the component databases) and requires more cooperation from the component databases. Consequently, we decided against following this approach as our primary strategy, and will explore it as an alternative secondary strategy at a later stage.

Our multidatabase query strategy requires a local query processor capable of performing joins and evaluating conditions over complex nested data-structures, for assembling the multidatabase query result from the single-database query results. The current query processor is relatively simple, but can be easily extended: it supports joins, projections and simple condition testing, but does not yet support aggregate functions.

4.2 The Multi-Database Directory

A *database directory* is used to store the information needed for accessing each database in the multidatabase system. The database directory stores the name of each component database and a pointer to the metadata files employed by the OPM query translator for accessing the database. In addition, the database directory contains for each database the list of its classes, the address of the server, and information for accessing it remotely.

For each class appearing in a multidatabase query, the OPM multidatabase query language (OPM*QL) parser accesses the database directory to determine the database containing that class. If a class name appears in multiple databases, the class name should be qualified with a database name in the query.

The database directory also contains information about known links between classes in different databases. These links are used in forms-based query systems for constructing join expressions across databases.

Part of the database directory for GSDB-HGD is shown in Figure 2.

Information required for a inter-database link includes the names of the databases and classes involved in the link, and the information necessary to express the link. The inter-database links are still being developed and precise details of the information required has not yet been finalized.

4.3 Multidatabase Query Processing

An OPM multidatabase query is first partitioned into several single class OPM_QL queries. Conditions in a multidatabase query involving only a single class are generally incorporated in the component OPM_QL query for that class, while conditions involving multiple classes, possibly from different databases, are evaluated by the OPM*QL processor.

An OPM multidatabase query is processed as follows.

1. Given a multidatabase query, `SELECT` S `FROM` CS `WHERE` $Conds$;, for each class specification *DB:C* appearing in CS (where DB is a database name, and C is a class name), an OPM_QL query for C, $Q_{DB:C}$ is generated:

   `SELECT` $S_{DB:C}$ `FROM` C `WHERE` $Conds_{DB:C}$;

```
DATABASE GSDB                          DATABASE HGD
   DESCRIPTION: "GSDB 2.0"                DESCRIPTION: "GDB 6.0 HGD"
   SERVER: GSDB                           SERVER: gdb60
   USERID: (user id)                      USERID: (user id)
   PASSWORD: (password)                   PASSWORD: (password)
   USERDB: gsdb                           USERDB: hgd
   METADATAFILE gsdb.so                  METADATAFILE: hgd.so
   CLASSES: Feature                       CLASSES: Amplimer
            Gene                                   Chromosome
            Product                                CitationLink
            Reference                              DBObject
            Sequence                               Gene
            Source                                 GeneProduct
            ....                                   GenomicSegment
                                                   Map
                                                   MapElement
                                                   Organism
                                                   SequenceLink;

LINK GSDB-GDB-Genes
  DESCRIPTION: "External reference from GSDB.Gene to HGD.Gene"
  FROM: GSDB.Gene
  TO:   HGD.Gene
  CODE: HGD:Gene.accessionID = GSDB:Gene.gdb_xref;
```

Figure 2: Part of the Database Directory for GSDB-GDB HGD

where

(a) $S_{DB:C}$ consists of each attribute path Y such that *DB:C.Y* occurs either in S or in *Cond*.

(b) $Conds_{DB:C}$ consists of those conditions in *Conds* involving only constant terms and terms of the form *DB:C.Y* for some attribute path Y: that is $Conds_{DB:C}$ does not involve any references to other classes $DB':C'$ where $DB \neq DB'$ or $C \neq C'$.

The query $Q_{DB:C}$ is formed by first filtering the abstact syntax tree for the multidatabase query to remove any parts involving classes other than *DB:C*, and then translating the query back into OPM_QL, replacing any terms of the form *DB:C.Y* with the attribute path Y. If $Conds_{DB:C}$ is empty then the `WHERE` keyword is omitted.

2. Each query $Q_{DB:C}$ is submitted to the database *DB*, and the resulting set of tuples,

$V_{DB:C}$, is converted into the internal representation of the multidatabase query engine.

3. A set of conditions $Conds'$ is formed, consisting of those conditions in $Conds$ involving multiple distinct classes. The values $V_{DB:C}$ are then searched for sets of tuples satisfying the conditions, $Conds'$, and suitable sets of tuples are then projected, renamed and combined in accordance with the select statement S.

Example. Consider the following query expressed over the HGD (GDB) and GSDB databases:

```
SELECT Name = GSDB:Gene.name, Reason = HGD:Gene.reason,
       Annotation = HGD:Gene.annotation
FROM GSDB:Gene, HGD:Gene
WHERE HGD:Gene.accessionID = GSDB:Gene.gdb_xref
                                AND   GSDB:Gene.name = "ACHE" ;
```

First, the following two OPM_QL queries are generated:

Query $Q_{GSDB:Gene}$:

```
SELECT name, gdb_xref
FROM Gene
WHERE name = "ACHE";
```

Query $Q_{HGD:Gene}$:

```
SELECT reason, annotation, accessionID
FROM Gene;
```

For each object instance x returned by $Q_{GSDB:Gene}$, and each object instance y returned by $Q_{HGD:Gene}$, if `accessionID` of y is identical to `gdb_xref` of x, then return: `Name` $= x$.`name`, `Reason` $= y$.`reason`, `Annotation` $= y$.`annotation`. □

5 Constructing an OPM Multidatabase System: An Example

In this section we illustrate how OPM can be used for constructing an OPM multidatabase system consisting of heterogeneous databases. For this purpose, we use the Genome Database (GDB) and the Genome Sequence Database (GSDB). We briefly overview the relevant parts of GDB and GSDB, and discuss the links and overlaps between GDB and GSDB in the context of their OPM schemas.

5.1 GDB: The Genome Database

The Genome Database (GDB) is an archival database of genomic mapping data maintained at Johns Hopkins School of Medicine, Baltimore [10]. The new version of GDB, GDB 6.0,[1] is developed with the Sybase database management system (DBMS), using the OPM toolkit [4]. GDB 6.0 consists of a federation of three distinct databases: Human Genome Database (HGD), Citation Database and Registry Database.

GDB contains *accessioned* objects representing entities of interest. All accessioned objects in GDB belong to class `DBObject` and are classified in subclasses of `DBObject`, organized in a class hierarchy. The main subclasses of this class hierarchy contain objects representing genomic data, literature references, and information on people and organizations.

Biological function information is represented by objects of class `BiologicalObject`, whose subclasses include `Expression`, `GeneFamily`, `GeneProduct` and `Organism`.

Data for representing and displaying maps are represented by objects of class `MappingObject`, whose subclasses include `Map`, `GenomicSegment`, and `MapElement`. `Map` contains objects representing the information regarding a resolved map and its subclasses include `ContentContigMap`, `CytogeneticMap`, `IntegratedMap`, `LinkageMap`, and `RadiationHybridMap`.

`GenomicSegment` contains objects representing any named region or set of regions of a genome, and its subclasses include `Clone`, `CellLine`, `Chromosome`, `Contig`, and `CytogeneticMarker`.

Maps are constructed by placing genomic segments on maps. Objects of class `MapElement` represent location information for genomic segments on specific maps.

[1] http://wwwtest.gdb.org/gdb/

Literature references are represented by objects of class `Citations`, whose subclasses include `JournalArticle`, `ConferencePaper`, `PersonalCommunication`, and `Book`.

GDB provides links to external databases. Such links are represented by objects of class `ExternalLink`. Subclasses of `ExternalLink` include `Enzyme`, `Phenotype`, and `Sequence`.

A subset of the GDB 6.0 OPM schema is given in the Appendix.

5.2 GSDB: The Genome Sequence Database

Genome Sequence Database (GSDB) is an archival database of genome sequence data maintained at the National Center for Genome Resources, Santa Fe. The current version of GSDB, GSDB 2.0,[2] has also been developed with Sybase DBMS but without using the OPM toolkit. For GSDB 2.0, an OPM interface, including an OPM schema,[3] has been retrofitted on top of GSDB 2.0; this interface allows accessing GSDB using the OPM query tools [6]. OPM schemas have also been retrofitted for preliminary versions of the forthcoming GSDB 2.2.

GSDB 2.0 is structured around one main class of objects, `Entry`, whose objects represent DNA *sequences* and are identified by accession numbers. The actual sequences (strings) are represented by objects of another class, `Sequence`. GSDB 2.0 also contains objects representing various entities, including *genes* (class `Gene`), *products* (class `Products`), *sources* (class `Source`), and *references* (class `Reference`). These objects are commonly qualified as *features* (class `Feature`), that is, classes `Gene`, `Products`, `Source`, `Reference`, as well as other classes, are subclasses of `Feature`. The location of features on sequences is represented by *location* objects (class `Location`).

A subset of the GSDB 2.0 OPM schema is given in the Appendix.

5.3 Relationships between GDB and GSDB

We have examined potential links and overlaps between GDB 6.0 and GSDB 2.0 at the (abstract) level of OPM classes and attributes (rather than Sybase tables and columns). The result of our analysis is summarized below.

---

[2] http://www.ncgr.org/gsdb/gsdb.html

[3] See http://gizmo.lbl.gov/DM_TOOLS/OPM/opm_4.html.

Genes. Both GDB and GSDB have a `Gene` class. In GSDB, genes are considered to be a kind of `Feature`: not actually as a specialization, since the same gene can occur as several features (see attribute `Feature.genes`). Other information held on genes in GSDB is confined to the name of the gene and references to an external database where primary information on the gene can be found. External references to either GDB or MOUSEDB are represented by attributes `gdb_xref` and `mousedb_xref`, respectively.

In GDB, class `Gene` is a subclass of class *GenomicSegment*. Information on genes include why a genomic region is considered a gene (see tuple attribute `Gene.evidence`) and links to gene families the gene belongs to (see derived attribute `Gene.families`). In addition, genes are characterized by additional data regarding mapping information (see derived attribute `GenomicSegment.mapsOf`) and references to derived sequences (see derived attribute `GenomicSegment.sequences`). A gene in GDB can be referenced from an external database using its GDB accession number (represented by attribute `accessionID`).

Sequences. Sequences are primary objects in GSDB (represented by object class `Sequence`). Sequence data include the actual sequence (see attribute `sequence`), sequence length (see attribute `length`), and information on the source of the sequence. In addition, there are references to sequences from other GSDB classes. For example, a feature is associated with a particular point on a sequence.

Sequence information in GDB is represented by objects of class `SequenceLink`. These objects contain annotations linking primary GDB objects to external sequence databases such as GSDB, as well as information regarding the beginning and end points of sequences (see attributes `startPos` and `endPos`). A `SequenceLink` object associates a GDB `DBObject`, either a `GenomicSegment`, a `Variation` or a `GeneProduct`, (attribute `dBObject`), with an accession number for some external database (attributes `accessionID` and `externalDB` inherited from class `ExternalLink`).

Sources. The GSDB `Source` class contains information about the source of a sequence: the organism, species and so on, which chromosome the sequence is associated with, and the corresponding cell-type. In addition, the GSDB `Source` class contains references to external taxonomic databases (see attribute `taxonomy_xref`) and to GDB probes (see attribute `gdb_probe_xref`).

Similar data are contained in GDB in the `Organism` and `Chromosome` classes. Objects in class `Organism` represent links to an external taxonomic database. Objects of class `Chromosome` are characterized by mapping and organism information.

Products. Both GDB and GSDB contain classes representing products. In GDB products are limited to gene products, while in GSDB a product can be associated with any feature. In both GDB and GSDB, these classes seem primarily to serve as a way of referencing external databases, such as protein databases. In GDB class `GeneProduct` has two sub classes, `Protein` and `RNA`, meaning that a gene product can be either a protein or a piece of functional (non-messenger) RNA. It's not clear whether products provide an interesting cross-reference between GDB and GSDB: in particular they would not in general give rise to direct inter-database links. However, products can provide important links to other protein databases such as PDB, and hence indirect links between GDB and GSDB.

References/Citations. Both GSDB and GDB contain data representing references or citations. In GSDB, a `Reference` object is considered as a kind of (i.e., a specialization of) `Feature` object. References in GSDB are characterized by titles, publication status, lists of authors and editors (see attributes `title`, `pub_status`, `authors` and `editors`, respectively), and external references to the *Medline* bibliographic database (see attribute `medline_xref`).

In GDB citations are represented by objects of class `CitationLink` and are further classified in subclasses of `Citation` representing books, journals, articles and so on. (In GDB's HGD, the `CitationLink` contains only links to external databases of citations, namely GDB's *Citations* Database.)

6 Using the OPM Multidatabase Query System: An Example

In this section we illustrate how the OPM multidatabase query system can be used across the Genome Database (GDB) and the Genome Sequence Database (GSDB).

6.1 Typical Queries Expressed over GDB and GSDB

The following queries return results based on data in both the HGD database of GDB 6.0 and the GSDB 2.0 database. These queries were suggested by Chris Fields, National Center for Genome Resources, Santa Fe, and were specified with help provided by Ken Fasman and Stan Letovsky, Johns Hopkins School of Medicine, Baltimore and Carol Harger, National Center for Genome Resources.

Query 1: Find the protein kinase genes on chromosome X. To identify protein kinase genes in GSDB it is necessary to first find protein kinase products in the GSDB Product class, and then to find Genes associated with the same Feature as the product. The corresponding Gene in the GSD Human Genome Database (HGD) can then be accessed by following the *gdb_xref* attribute from the GSDB Gene, if present, and equating it with the HGD *accessionID* attribute. (In fact some string reformatting is needed here because of current incompatibilities between the representations of GDB accession numbers in GDB and GSDB. These can be implemented using string reformatting functions built into the OPM multi-db query language, but will be ignored in the following examples.) In order to test whether a Gene occurs in chromosome X we can then follow the path in HGD from Gene to MapElement to Map to Chromosome.

```
SELECT HGD:Gene.displayName, HGD:Gene.accessionID
FROM   GSDB:Feature, HGD:Gene
WHERE  Feature.products.name MATCH "%kinase%"
AND    Feature.genes.gdb_xref = HGD:Gene.accessionID
AND    HGD:Gene.mapElements.map.chromosome.displayName = "X";
```

Query 2: Find sequenced regions on chromosome 17 with length greater than 100,000. Map Elements on chromosome 17 are selected from the HGD class MapElement using the path from MapElement to Map to Chromosome. Links from the

MapElements to GSDB Entries are found using the HGD SequenceLink class. From the GSDB Entry the corresponding sequence can be found and tested to see if its length is greater than 10,000.

```
SELECT Entry.accession_number, Entry.sequence.length
FROM   HGD:MapElement, HGD:SequenceLink, GSDB:Entry
WHERE  MapElement.map.chromosome = "17"
AND    SequenceLink.dBObject = MapElement.segment
AND    SequenceLink.externalDB.displayName = "GSDB"
AND    SequenceLink.accessionID = Entry.accession_number
AND    Entry.sequences.length > 10000;
```

Query 3: Find the sequences of ESTs mapped between 4q21.1 - 21.2. Currently this requires two queries: the first to find the coordinate range and the second to find ESTs with coordinates in that range and their sequences. Future extensions to the multi-database query system will allow this to be expressed as a single query.

The first part of the query finds the coordinates of the points q21.1 and q21.2 in the Cytogenetic Map of chromosome 4:

```
SELECT MapElement.coordinate, MapElement.point,
                                   MapElement.segment.displayName
FROM   HGD:MapElement
WHERE  MapElement.map.objectClass = "CytogeneticMap"
AND    MapElement.map.chromosome.displayName = "4"
AND    MapElement.segment.displayName IN {"q21.1", "q21.2"};
```

Next we can retrieve the expressed Amplimers occurring between these coordinates and lookup the corresponding sequence in GSDB.

```
SELECT Amplimer.displayName, Entry.accession_number,
       Entry.sequences.length
FROM   HGD:Amplimer, HGD:SequenceLink, GSDB:Entry
WHERE  Amplimer.isExpressed = "Yes"
AND    Amplimer.mapElements.map.chromosome.displayName = "4"
AND    Amplimer.mapElements.sortCoord >= START_COORD
AND    Amplimer.mapElements.sortCoord <= END_COORD
AND    SequenceLink.dbObject = Amplimer
AND    SequenceLink.externalDB.displayName = "GSDB"
AND    SequenceLink.accessionID = Entry.accession_number;
```

Where `START_COORD` and `END_COORD` are the values from the previous query.

6.2 Evaluating Queries Expressed over GDB and GSDB

Query 1: Find the protein kinase genes on chromosome X.

The first query generates the following OPM_AL query against HGD:

```
SELECT displayName, accessionID, mapElements.map.chromosome.displayName
FROM   Gene
WHERE  mapElements.map.chromosome.displayName = "X";
```

and the GSDB query:

```
SELECT products.name, genes.gdb_xref
FROM   Feature
WHERE  products.name MATCH "%kinase%";
```

leaving the condition `Feature.genes.gdb_xref = HGD:Gene.accessionID` to be tested by the local query engine.

Query 2: Find sequenced regions on chromosome 17 with length greater than 100,000.

The second query generates the OPM_QL queries for HGD:

```
SELECT map.chromosome,  segment
FROM   MapElement
WHERE  map.chromosome = "17";
```

and

```
SELECT dBObject,  externalDB.displayName, accessionID
FROM   SequenceLink
WHERE  externalDB.displayName = "GSDB"
```

and the following query for GSDB:

```
SELECT accession_number, sequences.length
FROM   Entry
WHERE  sequences.length > 10000;
```

Query 3: Find the sequences of ESTs mapped between 4q21.1 - 21.2.

The first part of the third query already only concerns a single class, `MapElement` of the HGD database, and may therefore be directly rewritten as an OPM_QL query. The second part of the query gives rise to the HGD queries:

```
SELECT displayName, isExpressed, mapElements.map.chromosome.displayName,
       mapElements.sortCoord
FROM   Amplimer
WHERE  isExpressed = "Yes"
AND    mapElements.map.chromosome.displayName = "4"
AND    mapElements.sortCoord >= START_COORD
AND    mapElements.sortCoord <= END_COORD;
```

and

```
SELECT dbObject, externalDB.displayName, accessionID
FROM   SequenceLink
WHERE  externalDB.displayName = "GSDB";
```

and the GSDB query:

```
SELECT accession_number, sequences.length
FROM   Entry;
```

7 Preliminary Conclusions

We have developed a multidatabase query system, OPM*QS, in the framework of the Object-Protocol Model (OPM). OPM*QS supports ad hoc queries over multiple heterogeneous databases via existing or retrofitted OPM interfaces. The current (first) version of OPM*QS has been developed between October 1995 and January 1996. This version of OPM*QS supports the expression of queries that combine (join) and manipulate data from multiple databases, where a database directory contains information on the OPM views (schemas) and remote access facilities of these databases. OPM*QS has been applied to queries over GSDB 2.0 and GDB 6.0.

The main purpose of the first version of OPM*QS was to demonstrate the feasibility of an OPM-based multidatabase approach to the problem of federating heterogenous molecular biology databases (MBDs). This version of OPM*QS also allowed us to gain experience in the area of constructing OPM-based multidatabase systems and of processing multidatabase queries. Some of this experience as well as our plans are discussed below.

7.1 Constructing a Multidatabase System

The OPM tools support construction of a multidatabase system consisting of databases that have been developed using OPM (such as GDB 6.0) or that have been retrofitted with an OPM view (such as GSDB 2.0). Currently the OPM retrofitting tool can be applied only to relational databases. We are, however, in the process of extending the OPM retrofitting tool for additional data models, such as ASN.1. This extension will broaden the range of databases that can be included in an OPM multidatabase system.

7.2 Multidatabase Query Processing Strategy

The multidatabase query processing strategy followed by the current version of OPM*QS is simple and general, but can be inefficient for certain types of queries. For example, a query that retrieves all the genes in class *Gene* of GSDB in order to find related genes in GDB via accession numbers, is often inefficient. A more efficient query strategy can put an order on the sub-queries and evaluate them in sequence, using the results of each sub-query to restrict the next sub-query in the sequence. However such a strategy would be considerably more difficult to implement, requiring statistics on sizes of individual classes and the selectivity of constraints in order to determine an optimal evaluation order. Although we consider examining such strategies in the future, in the short term we plan to increase the efficiency of multidatabase query processing by using *inter-database links*.

Inter-database links are known connections between heterogeneous databases that are recorded in the database directory together with the metadata on the individual databases. An example of such a link is the link between the `Gene` class in GSDB and the `Gene` class in GDB, represented by attribute `gdb_xref` of class `Gene` in GSDB; this attribute contains GDB accession numbers and thus indirectly points to GDB *Gene* objects. Following such a link allows the system to retrieve individual objects from a remote database, determined by the starting object of the link, rather than having to retrieve entire classes: in a sense following inter-database links in a query determines a pre-defined evaluation strategy and order for the query, rather than leaving the system to choose between multiple query evaluation plans.

From the perspective of a user constructing OPM*QL queries, inter-database links should appear to be much the same as regular OPM abstract attributes (intra-database links), except that the result of following such a link will be an object in another database rather than an object in a different class of the same database. Thus the database directory would associate an attribute name with each inter-database link, augmenting the attribute names already present in the OPM definition of a class, which could then be used to include the inter-database link in attribute paths in a query. It should be noted that such links do not subsume the general multidatabase joins already implemented, but rather complement them: using a combination of multidatabase joins, inter-database links, and other locally performed data manipulations, it should be possible to express very general and efficient multidatabase queries.

7.3 Assumptions and Problems

Multidatabase querying is based on the assumption that each database in the multidatabase system provides access for ad hoc queries. In addition, constructing a multidatabase system requires detailed documentation of the schemas and semantics of each component database, including information on links to other databases and semantic heterogeneity problems.

A variety of problems were encountered in performing multidatabase queries with OPM*QS, including semantic heterogeneity problems that had to be resolved. For example accession numbers are represented in different formats in different databases: a GDB accession number "`GDB:118746`" is represented as "`G00-118-746`" in GSDB 2.0, while a GSDB accession number "`L05367`" is represented in GDB as "`GSDB:L05367`". OPM*QL supports some general string manipulation and comparison operators in order to resolve such problems. It is hoped that such incompatibilities will normally be hidden in the code for inter-database links, and will therefore not concern users performing multidatabase queries. However merely identifying such incompatibilities, and determining how to resolve them, is a difficult problem, in part because of the lack of detailed documentation for individual database schemas.

7.4 Technological Alternatives

There are commercial distributed-join software tools that allow querying multiple relational databases, such as the Sybase *Enterprise CONNECT* family of products. It should be noted that such tools do not help constructing multidatabase systems: one still needs to understand the component databases in their relational representation, their semantics and links. The OPM tools support higher level representations of databases, using object-oriented constructs better suited for representing biological data, which makes this task easier. In addition the use of a database directory, with representations of individual OPM schemas and inter-database links, simplifies the task of navigating multiple databases.

A distributed-join tool could underly the processing of OPM multidatabase queries. In this case, an OPM*QL query will first be translated into multidatabase SQL queries. The multidatabase SQL queries would be processed by the distributed-join tool and then the returned query results would be further converted into the OPM data format. This strategy is different from our current strategy of translating an OPM*QL query into single class OPM-QL queries, and then into single database SQL queries.

Although we plan to examine this alternative query processing strategy in terms of cost and performance, we are aware of several problems of this alternative. First, a distributed-join tool can be used only for a set of data sources supported by the tool: usually major commercial database systems or widely used standards, but not the more specialized data-sources frequently used for molecular biology databases (e.g., ASN.1, ACeDB). For data sources (e.g., ASN.1) that are not supported by such a tool, additional programming would be still required. Furthermore, such a tool is restricted to the constructs of a standard relational query language (e.g.,SQL). Such query languages have been found to be overly restrictive and difficult to use for the queries commonly asked of a molecular biology database: for example, an SQL query against the relational schema for GSDB 2.0 will in general reference many more tables, and be considerably more complex, than an equivalent OPM query against the OPM schema retrofitted to GSDB.

Thus, the OPM*QS query processor is based on a more powerful nested relational algebra which supports directly operations on nested sets and complex data-structures. Furthermore, using a distributed-join tool for processing OPM*QL queries will make the performance of OPM*QS dependent on this tool. With our current query processing approach, we hope to be able to experiment with query optimization strategies and achieve better query performance.

References

[1] Bancilhon, F., Delobel. C., and Kanellakis, P., Building an Object-Oriented Database System: The Story of O_2, Morgan Kaufmann Publishers, Inc., 1992.

[2] Bright, M.W., Hurson, A.R., and Pakzad, H., A Taxonomy and Current Issues in Multidatabase Systems, *IEEE Computer*, **25**, 3, March 1992, pp. 50-60.

[3] Buneman, P., Davidson, S., Hart, K., Overton, C., and Wong, L., A Data Transformation System for Biological Data Sources, in *Proceedings of the 21st International Conference on Very Large Data Bases*, 1995, pp. 158-169.

[4] Chen, I.A., and Markowitz, V.M., An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools, *Information Systems*, Vol 20, No 5 (July 1995), pp. 393-418. Additional documentation is available at *http://gizmo.lbl.gov/opm.html*.

[5] Chen, I.A., and Markowitz, V.M., OPM Schema Translator 4.0, Reference Manual, Lawrence Berkeley National Laboratory Technical Report LBL-35582 (revised), 1995.

[6] Chen, I.A., Markowitz, V.M., and Szeto, E., The OPM Query Translator, Lawrence Berkeley National Laboratory Technical Report LBL-33706, 1995. Available together with additional documentation at *http://gizmo.lbl.gov/opm.html*.

[7] Cozza, S., Reed, E. C., Salit, J., Chang, W., Marr, T., Genome Topographer: A Next Generation Genome Database System (Abstract), presented at the meeting on Genome Mapping and Sequencing, ColdSpring Harbor Laboratory, 1994.

[8] Etzold, T., and Argos, P., SRS, An Indexing and Retrieval Tools for Flat File Data Libraries,. *Computer Applications of Biosciences*, 9, 1993, pp. 49-57.

[9] Hammer, M., McLeod, D. Database Description with SDM: A Semantic Database Model. *ACM Trans. on Database Systems* **6**, 3, (September 1981), 351–386.

[10] Fasman, K.H., Letovsky, S.I., Li, P., and Cottingham, B.W. (1996) Improvements to the GDB Human Genome Data Base. Nucleic Acids Research, Vol. 24, Database Supplement (in press). See also `http://wwwtest.gdb.org/gdb/about.html`.

[11] Goto, S., Akiyama, Y., and Kanehisa, M., LinkDB: A Database of Cross Links Between Molecular Biology Databases, In [12].

[12] Abstracts of the Second Meeting on Interconnection of Molecular Biology Databases. Available at `http://www-genome.wi.mit.edu/informatics/abstracts.html`.

[13] Ritter, O., The Integrated Genomic Database, in *Computational Methods in Genome Research*, S. Suhai (ed), Plenum, 1994, pp. 57-73.

[14] Sciore, E. Versioning and Configuration Management in an Object-Oriented Data Model. *VLDB Journal* **3** (1994), 77–106.

A Part of GDB and GSDB OPM Schemas

A.1 Part of the GDB OPM Schema

```
OBJECT CLASS Amplimer isa* GenomicSegment
   ATTRIBUTE isExpressed: [1,1] YesNoUnknown_UnkDict
   ATTRIBUTE sequence: list-of [0,] VARCHAR(255)

OBJECT CLASS Chromosome isa* GenomicSegment
   ATTRIBUTE cellularCompartment: [1,1] CompartmentDict

OBJECT CLASS CitationLink isa* ExternalLink
   ATTRIBUTE dBObjects: set-of [1,] DBObject

OBJECT CLASS DBObjects
   ID: accessionID
   ATTRIBUTE accessionID: [1,1] VARCHAR(50)
   ATTRIBUTE displayName: [1,1] VARCHAR(255)
   ATTRIBUTE objectClass: [1,1] VARCHAR(30)
   ATTRIBUTE searchName: [1,1] VARCHAR(255)

OBJECT CLASS ExternalLink
   ATTRIBUTE displayName: [1,1] VARCHAR(255)
   ATTRIBUTE searchName: [1,1] VARCHAR(255)
   ATTRIBUTE objectClass: [1,1] VARCHAR(255)
   ATTRIBUTE externalDB: [1,1] ExternalDB
   ATTRIBUTE accessionID: [1,1] VARCHAR(255)
   ATTRIBUTE externalVersion: [0,1] VARCHAR(20)

OBJECT CLASS Gene isa* GenomicSegment
   ATTRIBUTE evidence (reason, annotation): set-of [1,]
                ([1,1] GeneEvidenceDict, [1,1] VARCHAR(255))
   ATTRIBUTE families
             DERIVATION: ! genes [GeneFamily]

OBJECT CLASS GeneProduct isa* BiologicalObject
   ATTRIBUTE sequences
             DERIVATION: ! dBObject [SequenceLink]
   ATTRIBUTE genes (gene, subunit, nCopies): set-of [0,]
             ([1,1] Gene, [0,1] VARCHAR(20), [0,1] SMALLINT)

OBJECT CLASS GenomicSegment isa* MappingObject
   ATTRIBUTE mapsOf
             DERIVATION: ! mapOf [Map]
   ATTRIBUTE sequences
             DERIVATION: ! dBObject [SequenceLink]

OBJECT CLASS Map isa* MappingObject
   ATTRIBUTE chromosome: [1,1] Chromosome
```

```
OBJECT CLASS MapElement isa* MappingObject
   ATTRIBUTE segment: [1,1] GenomicSegment
   ATTRIBUTE map: [1,1] Map

OBJECT CLASS Organism isa* BiologicalObject
   ATTRIBUTE taxonomicDescriptions (externalDB, externalAccessionID,
             externalVersion): set-of [0,]
             ([0,1] ExternalDB, [0,1] VARCHAR(255), [0,1] VARCHAR(20))

OBJECT CLASS SequenceLink isa* ExternalLink
   ATTRIBUTE dBObject: [1,1] GenomicSegment or Variation or GeneProduct
   ATTRIBUTE startPos: [0,1] INTEGER
   ATTRIBUTE endPos: [0,1] INTEGER
```

A.2 Part of the GSDB OPM Schema

```
OBJECT CLASS Feature
   ID: id
   ATTRIBUTE id: [1,1] table_id
   ATTRIBUTE type: [0,1] VARCHAR(30)
   ATTRIBUTE genes: set-of [1,] Gene
   ATTRIBUTE products: set-of [0,] Product

OBJECT CLASS Gene
   ID: id
   ATTRIBUTE id: [1,1] table_id
   ATTRIBUTE gdb_xref: [0,1] VARCHAR(20)
   ATTRIBUTE mousedb_xref: [0,1] VARCHAR(20)
   ATTRIBUTE name: [0,1] VARCHAR(80)
   ATTRIBUTE oldgsdb_xref: [0,1] INTEGER

OBJECT CLASS Product
   ID: id
   ATTRIBUTE id: [1,1] table_id
   ATTRIBUTE ec_xref: [0,1] VARCHAR(20)
   ATTRIBUTE name: [0,1] VARCHAR(200)
   ATTRIBUTE oldgsdb_xref: [0,1] INTEGER

OBJECT CLASS Reference isa Feature
   ATTRIBUTE feature_id: [1,1] Feature
   ATTRIBUTE medline_xref: [0,1] table_id
   ATTRIBUTE pub_status: [0,1] type_spec
   ATTRIBUTE title: [0,1] VARCHAR(255)
   ATTRIBUTE authors (au_last_name, au_first_name, au_initials,
                      au_suffix, au_summary): list-of [0,]
                     ([1,1] VARCHAR(40),[0,1] VARCHAR(20), (0,1) VARCHAR(10),
                      [0,1] VARCHAR(10), [0,1] VARCHAR(255))
```

```
   ATTRIBUTE editors (ed_last_name, ed_first_name, ed_initials,
                      ed_suffix, ed_summary): list-of [0,]
                     ([1,1] VARCHAR(40),[0,1] VARCHAR(20), (0,1) VARCHAR(10),
                      [0,1] VARCHAR(10), [0,1] VARCHAR(255))

OBJECT CLASS Sequence
   ID: id
   ATTRIBUTE id: [1,1] table_id
   ATTRIBUTE sequence: [0,1] TEXT
   ATTRIBUTE length: [0,1] INTEGER

OBJECT CLASS Source isa Feature
   ATTRIBUTE feature_id: [1,1] Feature
   ATTRIBUTE taxonomy_xref: [0,1] table_id
   ATTRIBUTE scientific_name: [0,1] VARCHAR(80)
   ATTRIBUTE common_name: [0,1] VARCHAR(80)
   ATTRIBUTE taxonomy: [0,1] VARCHAR(255)
   ATTRIBUTE gdb_probe_xref: [0,1] table_id
```