bdbms – A Database Management System for Biological Data

Terhi Töyli
Department of Computer Science
University of Helsinki, Finland
terhi.toyli@helsinki.fi

***Abstract*—Current database systems lack several specific functionalities that are needed by biological databases. bdbms is an extensible prototype database management system for supporting biological data. bdbms extends the current database systems with: (1) Annotation and provenance management, (2) Local dependency tracking, (3) Update authorization, and (4) Non-traditional and novel access methods.**

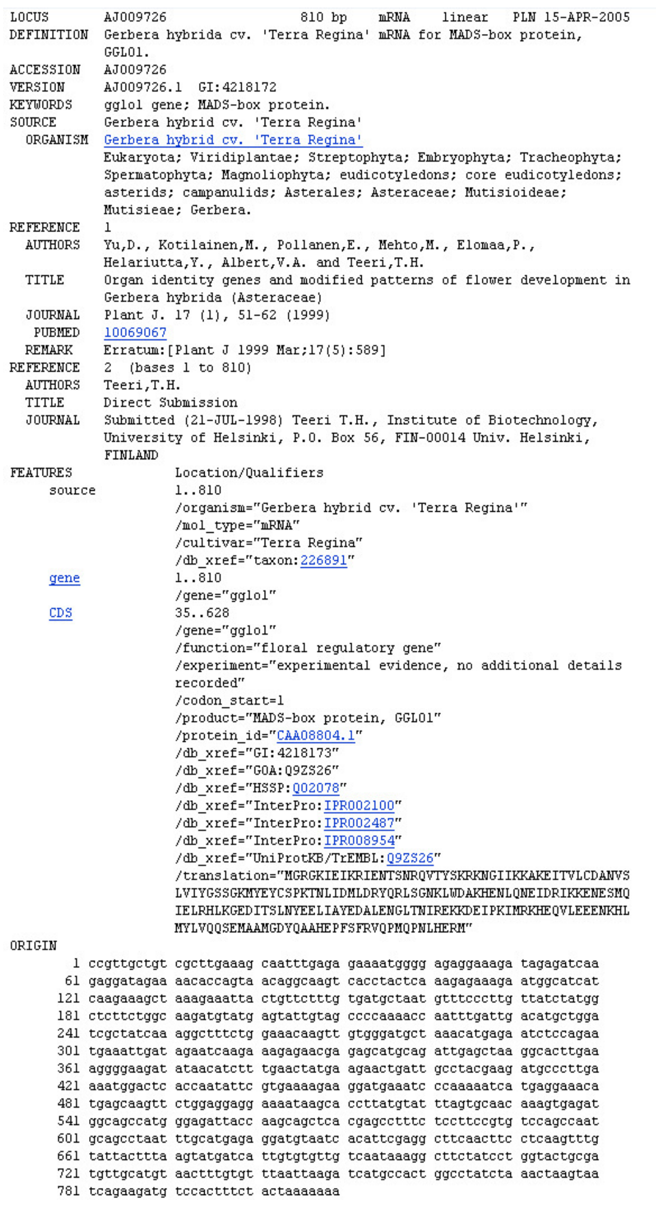
1. Introduction

Biological databases typically consist of raw data, metadata, sequences, annotations, and related data obtained from various sources (Fig. 1). These databases are used in life science research to deposit raw data, store interpretations of experiments and results of analysis processes, and search for matching structures and sequences. They are essential to biological experimentation and analysis [1].

Current database technology does not fulfill the specific requirements of biological databases, and that has become a serious handicap to scientific progress. In many cases, the data is stored in flat files or spreadsheets because the current database systems lack several functionalities that are needed by biological databases. Efficient support for sequences, annotations, and provenance are some of these functionalities that are needed to manage and process biological data properly. Consequently, many advantages and functionalities that current database systems offer are nullified and bypassed in biological databases.

Experimental technologies and semantics of the information content change rapidly and biological databases should evolve with them. One problem is a lack of absolute authority to verify the correctness of information in biological databases. All this makes it is difficult to foresee the kinds of additional information that may become necessary to attach to data in the database.

bdbms is an extensible prototype database engine for supporting and processing biological databases[1]. bdbms focuses on the four following functionalities: (1) Annotation and provenance management, (2) Local dependency tracking,



```
LOCUS       AJ009726                 810 bp    mRNA    linear   PLN 15-APR-2005
DEFINITION  Gerbera hybrida cv. 'Terra Regina' mRNA for MADS-box protein,
            GGLO1.
ACCESSION   AJ009726
VERSION     AJ009726.1  GI:4218172
KEYWORDS    gglo1 gene; MADS-box protein.
SOURCE      Gerbera hybrid cv. 'Terra Regina'
  ORGANISM  Gerbera hybrid cv. 'Terra Regina'
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; core eudicotyledons;
            asterids; campanulids; Asterales; Asteraceae; Mutisioideae;
            Mutisieae; Gerbera.
REFERENCE   1
  AUTHORS   Yu,D., Kotilainen,M., Pollanen,E., Mehto,M., Elomaa,P.,
            Helariutta,Y., Albert,V.A. and Teeri,T.H.
  TITLE     Organ identity genes and modified patterns of flower development in
            Gerbera hybrida (Asteraceae)
  JOURNAL   Plant J. 17 (1), 51-62 (1999)
   PUBMED   10069067
  REMARK    Erratum:[Plant J 1999 Mar;17(5):589]
REFERENCE   2  (bases 1 to 810)
  AUTHORS   Teeri,T.H.
  TITLE     Direct Submission
  JOURNAL   Submitted (21-JUL-1998) Teeri T.H., Institute of Biotechnology,
            University of Helsinki, P.O. Box 56, FIN-00014 Univ. Helsinki,
            FINLAND
FEATURES             Location/Qualifiers
     source          1..810
                     /organism="Gerbera hybrid cv. 'Terra Regina'"
                     /mol_type="mRNA"
                     /cultivar="Terra Regina"
                     /db_xref="taxon:226891"
     gene            1..810
                     /gene="gglo1"
     CDS             35..628
                     /gene="gglo1"
                     /function="floral regulatory gene"
                     /experiment="experimental evidence, no additional details
                     recorded"
                     /codon_start=1
                     /product="MADS-box protein, GGLO1"
                     /protein_id="CAA08804.1"
                     /db_xref="GI:4218173"
                     /db_xref="GOA:Q9ZS26"
                     /db_xref="HSSP:Q02078"
                     /db_xref="InterPro:IPR002100"
                     /db_xref="InterPro:IPR002487"
                     /db_xref="InterPro:IPR008954"
                     /db_xref="UniProtKB/TrEMBL:Q9ZS26"
                     /translation="MGRGKIEIKRIENTSNRQVTYSKRKNGIIKKAKEITVLCDANVS
                     LVIYGSSGKMYEYCSPKTNLIDMLDRYQRLSGNKLWDAKHENLQNEIDRIKKENESMQ
                     IELRHLKGEDITSLNYEELIAYEDALENGLTNIREKKDEIPKIMRKHEQVLEEENKHL
                     MYLVQQSEMAAMGDYQAAHEPFSFRVQPMQPNLHERM"
ORIGIN
        1 ccgttgctgt cgcttgaaag caatttgaga gaaaatgggg agaggaaaga tagagatcaa
       61 gaggatagaa aacaccagta acaggcaagt cacctactca aagagaaaga atggcatcat
      121 caagaaagct aaagaaatta ctgttctttg tgatgctaat gtttcccttg ttatctatgg
      181 ctcttctggc aagatgtatg agtattgtag ccccaaaacc aatttgattg acatgctgga
      241 tcgctatcaa aggctttctg gaaacaagtt gtgggatgct aaacatgaga atctccagaa
      301 tgaaattgat agaatcaaga aagagaacga gagcatgcag attgagctaa ggcacttgaa
      361 aggggaagat ataacatctt tgaactatga agaactgatt gcctacgaag atgcccttga
      421 aaatggactc accaatattc gtgaaaagaa ggatgaaatc ccaaaaatca tgaggaaaca
      481 tgagcaagtt ctggaggagg aaaataagca ccttatgtat ttagtgcaac aaagtgagat
      541 ggcagccatg ggagattacc aagcagctca cgagcctttc tccttccgtg tccagccaat
      601 gcagcctaat ttgcatgaga ggatgtaatc acattcgagg cttcaacttc ctcaagtttg
      661 tattacttta agtatgatca ttgtgtgttg tcaataaagg cttctatcct ggtactgcga
      721 tgttgcatgt aactttgtgt ttaattaaga tcatgccact ggcctatcta aactaagtaa
      781 tcagaagatg tccactttct actaaaaaaa
```

Fig. 1: An example of a database search from Entrez Nucleotide database [3]. The database is actually a collection of sequences from several sources including GenBank[4] and RefSeq[5].

Helsinki, April 24, 2008
58308110 Seminar: Management of Biological Databases, spring 2008
Department of Computer Science
University of Helsinki, Finland

(3) Update authorization, and (4) Non-traditional and novel access methods.

bdbms treats annotations and provenance data as first-class objects [2]. The framework that is provided allows adding annotations/provenance at multiple granularities, archiving and restoring annotations, and querying the data based on annotation/provenance values.

bdbms introduces Annotation-SQL, or A-SQL for short, which is an extension to SQL. The extension supports the processing and querying of annotation and provenance information and allows them to be propagated with query answers with minimum user programming. The users are not expected to know how to use A-SQL, so a graphical interface will be provided with bdbms.

bdbms also proposes a systematic approach for tracking dependencies among database entries [1]. In practice, when a database item is modified, bdbms can track and mark any other database item that is affected by this modification. In biological databases this is very desirable, since e.g. modifying one DNA sequence may have effect on several other items, for instance on the protein sequence derived from the modified DNA sequence. Lack of this automatic dependency tracking raises concerns on the quality and the consistency of the data available in biological databases.

The GRANT/REVOKE access models supported by current database systems depend only on the identity of the user. bdbms extends this by proposing a *content-based* authorization, which is based on both the identity of the updater and the content of the updated data.

There is a need to integrate for example new types of index structures along with their supporting operators inside bdbms. These non-traditional and novel access methods provide access methods for supporting various types of biological data.

2. Architecture

The annotation manager, the dependency manager, and the authorization manager are the main components of bdbms [1]. A-SQL is bdbms's extended SQL and it supports annotation and authorization commands. The *annotation manager* is responsible for handling the annotations in an annotation storage space, the *dependency manager* is responsible for handling the dependencies and derivations among database items, and the *authorization manager* handles content-based authorizations and the standard authorizations.

The dependencies are stored in a *dependency storage space* and *index structures* are available in bdbms in support of the multidimensional and compressed data.

3. Annotation Management

Users' comments, experiences, related information that is not modeled by the database, and the provenance of the data are examples of the extra information that is linked to the database as annotations [1]. Annotations are important mean of communication and interaction between database users, and they are used to allow users to have a better understanding of the data. Annotations often answer the questions such as: How was the piece of data obtained? Why was something added or modified? Which experiments or analyses or programs were used? From which source was the piece of data obtained? Answering these questions is very important in assessing the value and credibility of the data.

```
CREATE ANNOTATION TABLE <ann_table_name>
ON <user_table_name>
```

```
DROP ANNOTATION TABLE <ann_table_name>
ON <user_table_name>
```

Fig. 2: The A-SQL commands CREATE and DROP [1]

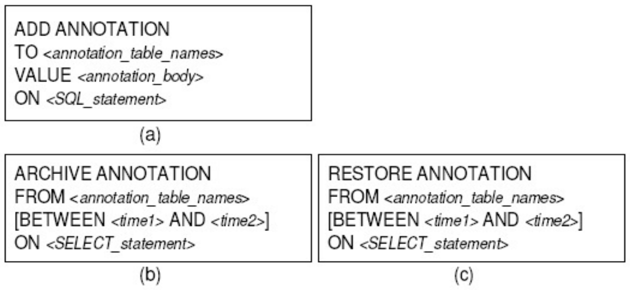
Most of the current database systems do not support annotations systematically, despite their importance. Most of the proposed techniques assume simple annotation schemes and focus on annotation propagation. However, for example mechanisms for annotation insertion, archival, and indexing are needed for efficient annotation management. bdbms addresses several of these requirements of annotation management.

3.1 CREATE ANNOTATION TABLE and DROP ANNOTATION TABLE

All annotations are metadata, but they may have different importance, meaning, and creditability, and therefore require categorization and separation. For example, annotations representing the lineage of the data have different purpose and importance from the annotations representing users' comments, and they may also have an impact on the storage mechanism of the data.

bdbms provides a mechanism for categorizing annotations at the storage, query processing and annotation propagation levels [1]. At the storage level, the A-SQL command CREATE ANNOTATION TABLE (Fig. 2) allows user to design and categorize their annotations. This also facilitates annotation propagation. User relation may have multiple annotation tables attached to it, for example one that stores experiment information and another one that stores users' comments. The DROP ANNOTATION TABLE (Fig. 2) command is used for dropping an annotation table.

Several storage and indexing schemes are being investigated to efficiently store the annotations. For example, a single annotations of provenance can be attached to many database items, and adding the same information multiple times to different tables may cause the processing and storing to be very expensive. The annotations should be stored with respect to storage overhead, I/O cost to retrieve the annotations, and the query processing time. It is also considered to take into account whether the annotation is linked to multiple data items in different tables or just to very few specific cells.

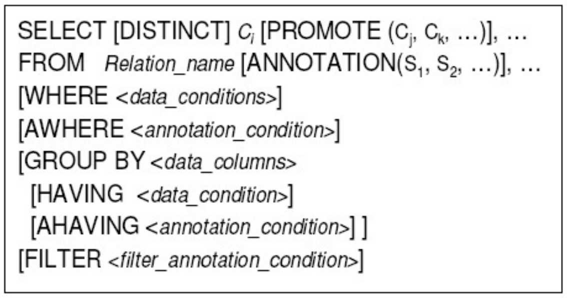


```
ADD ANNOTATION
TO <annotation_table_names>
VALUE <annotation_body>
ON <SQL_statement>
```
(a)

```
ARCHIVE ANNOTATION
FROM <annotation_table_names>
[BETWEEN <time1> AND <time2>]
ON <SELECT_statement>
```
(b)

```
RESTORE ANNOTATION
FROM <annotation_table_names>
[BETWEEN <time1> AND <time2>]
ON <SELECT_statement>
```
(c)

Fig. 3: The A-SQL commands ADD, ARCHIVE, and RESTORE [1]



```
SELECT [DISTINCT] Ci [PROMOTE (Cj, Ck, ...)], ...
FROM   Relation_name [ANNOTATION(S1, S2, ...)], ...
[WHERE <data_conditions>]
[AWHERE <annotation_condition>]
[GROUP BY <data_columns>
  [HAVING  <data_condition>]
  [AHAVING <annotation_condition>] ]
[FILTER <filter_annotation_condition>]
```

Fig. 4: The A-SQL SELECT command [1]

3.2 ADD ANNOTATION

Storing the annotations should be transparent from end-users. Current database systems do not provide a mechanism to facilitate annotating of the data and user has to know exactly how the annotations are stored in the tables before the user can make the UPDATE command. The system should provide new expressive commands as well as visualization tools that allow users to add their annotations seamlessly and graphically at various granularities.

bdbms proposes the A-SQL command ADD ANNOTATION for adding annotations (Fig. 3(a)). The *annotation_table_ names* specifies to which annotation table(s) the added annotation will be stored, the *annotation_body* specifies the annotation value to be added, and the output of the *SQL_statement* specifies the data to which the annotation is attached. The output of the *SQL_statement* can be at various granularities. It is planned to support XML-formatted annotations in bdbms, since the annotations may contain important information that users want to query. In case the annotation body is XML-formatted text, the users can structure their annotations and make use of XML querying capabilities. An example of ADD ANNOTATION command could be:

```
ADD ANNOTATION
TO this_table.geneAnnotation
VALUE '<Annotation>
          This is a gene.
        </Annotation>'
ON (Select T.geneID
     From this_table T);
```

This attaches the annotation to every *geneID* in the table *this_table* and stores the annotation in the annotation table *geneAnnotation*. The <*Annotation*> is the XML tag that encloses the annotation information.

The *SQL_statement* will be an INSERT, UPDATE or DELETE statement to allow users to link annotations to database operations. For example, the user can use the delete operation to store the deleted items to separate log table along with the annotation that specifies why the items have been deleted. The standard system recovery log cannot be used for this purpose because it does not support the users' need to structure their annotation freely.

It is planned to add a visualization tool to bdbms to allow users to annotate their data in a transparent way. Users' tables are displayed as grids or spreadsheets where users can select one or more cells to annotate. In Oracle, the integration of database tables with Excel spreadsheets is addressed, and it is planned to add that integration feature to bdbms.

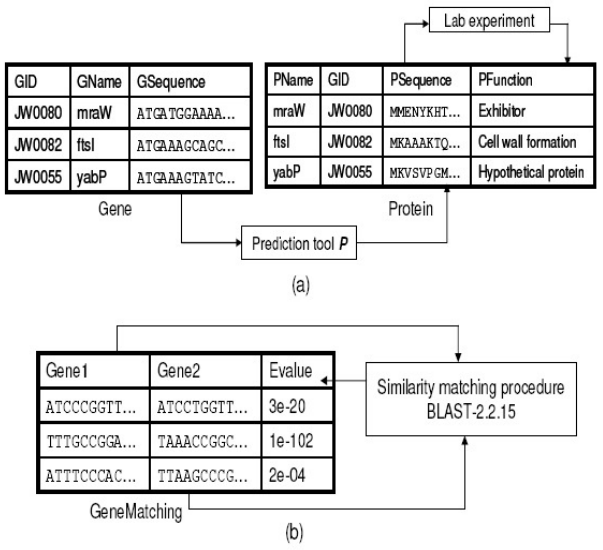
3.3 ARCHIVE ANNOTATION AND RESTORE ANNOTATION

Annotations may be needed to be deleted or archived when they become unnecessary or invalid. bdbms supports the archival of annotations instead of deleting them [1]. These old, archived annotations are not propagated along with query answers. Archiving isolates old and invalid annotations from the recent and valuable ones, but also reserves them for later use. The biological data is usually a little uncertain, and the old values may turn out to be the correct values. Archiving annotations gives users the possibility to restore the annotations later, if needed. Restored annotations are propagated normally along with query answers.

For archiving and restoring annotations, bdbms provides ARCHIVE ANNOTATION (Fig. 3(b)) and RESTORE ANNOTATION A-SQL commands (Fig. 3(c)), respectively. The *annotation_table_names* specifies which annotation tables will be archived/restored, and the output of the *SELECT_ statement* specifies the data on which the annotations will be archived/restored. The BETWEEN clause is optional and specifies a time range over which the annotations will be archived/restored. The time stamp is assigned to each annotation when it is first added to the database.

3.4 SELECT

Simplifying the users' queries is required to allow annotation propagation. The problem is that users view the annotations as metadata, whereas the DBMSs view annotations as normal data. In user's point of view, two entries of the same gene from different tables are identical if only their annotations differ. However, from the database view point, they are not identical because the annotations are viewed as normal attribute data. In order to overcome the mismatch in interpreting the annotations, users' queries may become complex. This can be avoided by extending the query operators.



GID	GName	GSequence
JW0080	mraW	ATGATGGAAAA...
JW0082	ftsI	ATGAAAGCAGC...
JW0055	yabP	ATGAAAGTATC...

PName	GID	PSequence	PFunction
mraW	JW0080	MMENYKHT...	Exhibitor
ftsI	JW0082	MKAAAKTQ...	Cell wall formation
yabP	JW0055	MKVSVPGM...	Hypothetical protein

Gene1	Gene2	Evalue
ATCCCGGTT...	ATCCTGGTT...	3e-20
TTTGCCGGA...	TAAACCGGC...	1e-102
ATTTCCCAC...	TTAAGCCCG...	2e-04

Fig. 5: Local dependency tracking [1]

bdbms' A-SQL command SELECT extends the standard SELECT command by introducing new operators and extending the semantics of the standard operators (Fig. 4). The new operators introduced are ANNOTATION, PROMOTE, AWHERE, AHAVING, and FILTER.

The ANNOTATION operator specifies the annotation table(s) which are to be considered in the query. Users do not have to know how or where annotations are stored, instead they can propagate their annotations transparently and specify only which annotations are of interest.

The PROMOTE operator specifies the annotations from one or more columns that users want to copy to a projected column.

The AWHERE and AHAVING clauses are applied over the annotations, but otherwise they are equivalent to the standard WHERE and HAVING clauses. The FILTER clause filters any annotation that does not satisfy *filter_annotation_ condition* from the data of the input relation.

The standard operators and the operators that group or combine, for example *projection, selection*, *group by*, *union*, and *intersect,* are also extended to process the annotations attached to the tuples.

In addition to defining the commands and operators to support the features within bdbms, it is also needed to define for each A-SQL operator its algebraic definition, cost estimate function, and algebraic properties that can be used by the query optimizer to generate efficient query plans.

4. Provenance management

The provenance of data is very important in assessing the value and credibility of the data, since the biologists often interact and exchange data with each other. Biological data can also be queried by its provenance. Like annotations, the data provenance can be attached to the database at multiple granularities.

PName	GID	PSequence	PFunction
mraW	JW0080	MKENYKNM...	Exhibitor
ftsI	JW0082	MTATTKTQ...	Cell wall formation
yabP	JW0055	MKVSVPGM...	Hypothetical protein

Protein

PName	GID	PSeq.	PFun.
0	0	0	1
0	0	0	1
0	0	0	0

Protein-Bitmap

Fig. 6: Use of bitmap to mark outdated data [1]

In bdbms the provenance data is treated as a kind of annotations where all the requirements and functionalities that are available for annotations are also applicable to provenance data [1]. In addition, the provenance data has some special requirements and characteristics that need to be addressed.

Provenance data has usually well-defined structure, whereas annotations can be free text. The values of provenance data are drawn from a list of predefined values.

Provenance data is not usually allowed to be inserted or updated by end-users. It needs to be automatically inserted and maintained by the system and end-users can only retrieve or propagate this information. It is needed to provide an access control mechanism over the provenance data to restrict the annotation operations to certain users or programs as required.

5. Local Dependency Tracking

In many cases, the dependencies and derivations, which biological databases are full of, cannot be automatically computed using coded functions, e.g. prediction tools, lab experiments, or instruments may be needed to derive the data (Fig. 5).

In Fig. 5(a), protein sequences are derived from gene sequences using a prediction tool P, and the function of the protein is derived from the protein sequence using lab experiments. If a gene sequence is modified, then the protein sequences that depend on that gene have to be marked as *outdated* until their values are reverified. Also the function of the protein sequences marked as *outdated* must be marked *outdated* as well.

In Fig. 5(b) another type of dependency is presented. The value of data in the database depends on the procedure or program that generated the data. If the version of BLAST is modified or BLAST is replaced with another procedure, the *Evalue*s must be re-evaluated. If BLAST can be modeled as a database function, these values can be automatically evaluated. Otherwise the values are marked as *outdated*.

bdbms proposes to extend the concept of *Functional Dependencies* to *Procedural Dependencies* [1]. In *Procedural Dependencies* not only the dependency among the data is tracked, but also the type and characteristics of the dependency. The procedure on which the dependency is based, whether or not the procedure can be executed by the database, and whether or not the procedure is invertible are among the tracked characteristics. Procedural Dependencies are used in bdbms to allow users to model the dependencies among the database items. They are also used to detect conflicts and cycles among dependency rules, and to compute the closure of procedures.

Dependencies among the data can be stored either at the schema level or at the instance level. Schema-level dependencies can be modeled using foreign key constraints. Instance- level dependencies are more complex to model and can be modeled by dependency graphs.

Dependency graphs are used to figure out the items that have to be marked as *outdated* when a modification occurs in the database. The outdated items must be marked clearly that these items can be identified in any future reference. bdbms proposes to associate a bitmap with each table in the database. A bitmap cell set to 1 corresponds the outdated cell in the database, otherwise the bitmap cell set to 0. Fig. 6 shows in practice how the bitmap is modified when the sequences corresponding to genes JW0080 and JW0082 (Fig. 5(a)) are modified. Because Psequence is automatically updated by executing procedure P, the bits corresponding to Psequence are not set to 1. Data compression techniques can be used to effectively compress the bitmaps to reduce the storage overhead.

The items that need to be verified or re-evaluated should be able to be reported by database at all times. The outdated items involved in query answers should be propagated with an annotation that the query answer may not be correct. Detecting the outdated items at query execution time is a challenging problem, and a proposed solution is to consider the status of the database items as annotations attached to those items. The annotations will be automatically propagated along with query answers.

A mechanism for users to validate outdated items will be provided by bdbms. An outdated item may or may not need to be modified to become valid.

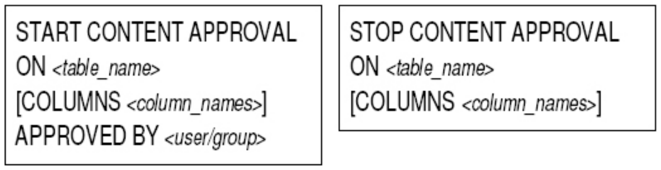
6. Update Authorization

Changes over the database should be subject to authorization and approval by authorized entities before the changes become permanent in the database, because they may have important consequences. In current database management systems the authorization is based on GRANT/ REVOKE access model. In this model a user is granted a permission to do certain operations based on only the identity of the user not on the content of the data being inserted or updated [1].

Biological databases may not fit with this model, as they are usually community based and shared effort. For example, if a lab administrator is the only user who has the right to update the database, then (s)he becomes a bottleneck in the process of populating the database, and if all the lab members can modify the data without revision, the credibility and authenticity of the data may be compromised [2].

bdbms introduces a monitoring system, termed *content-based approval*, that allows the database to systematically track the changes over the database. The content-based approval mechanism works with the existing GRANT/REVOKE mechanism.

The authorization is based on the identity of the user as well



```
START CONTENT APPROVAL
ON <table_name>
[COLUMNS <column_names>]
APPROVED BY <user/group>
```

```
STOP CONTENT APPROVAL
ON <table_name>
[COLUMNS <column_names>]
```

Fig. 7: Content-based approval [1]

as the content of the data being inserted or updated. The content-based approval feature can be turned ON or OFF for a certain table or columns by the database administrator (Fig. 7). The *table_name* specifies the user table on which the update operations will be monitored, and the optional COLUMNS clause specifies the columns to be monitored. The *user/group* specifies the user or group of users that can approve or disapprove the update operations.

The content-based approval mechanism maintains a log of all update operation that occur in the database [2] along with the identifier of the user who issued the operation and the issuing time. All non-approved updates will be visible with an annotation mentioning they were not approved yet. The logs are revised by authorized users to approve/disapprove the operations. If an operation is disapproved, bdbms executes an automatically generated and stored inverse operation that negates the effect of the original operation. The execution of the inverse statement may affect other elements in the database, so the Local Dependency Tracking feature tracks down and invalidates these elements [1].

7. Indexing and query processing

Non-traditional indexing techniques are proposed to be integrated inside bdbms to enable biological algorithms to operate efficiently on the database. bdbms focuses on two fronts: (1) Supporting multidimensional datasets via multidimensional indexing techniques (suitable for protein 3D structures and surface shape matching), and (2) Supporting compressed datasets via novel external-memory indexes that work over the compressed data without decompressing it (suitable for indexing large sequences) [1].

Compressing the data inside the database improves the system performance. The size of the data, the number of I/O operations to retrieve the data, and the buffer requirements are significantly reduced. In bdbms it is being investigated how biological data can be stored in compressed form and yet be able to operate on the compressed data without decompressing it.

In bdbms, an extensible indexing framework, termed SP-GiST, is used [1]. The SP-GiST framework is implemented inside PostgreSQL, and it broadens the class of supported indexes to include disk-based versions of space-partitioning trees. Space-partitioning trees are a family of access methods that index objects in a multi-dimensional space.

SP-GiST allows developers to instantiate a variety of index structures in an efficient way through pluggable modules and

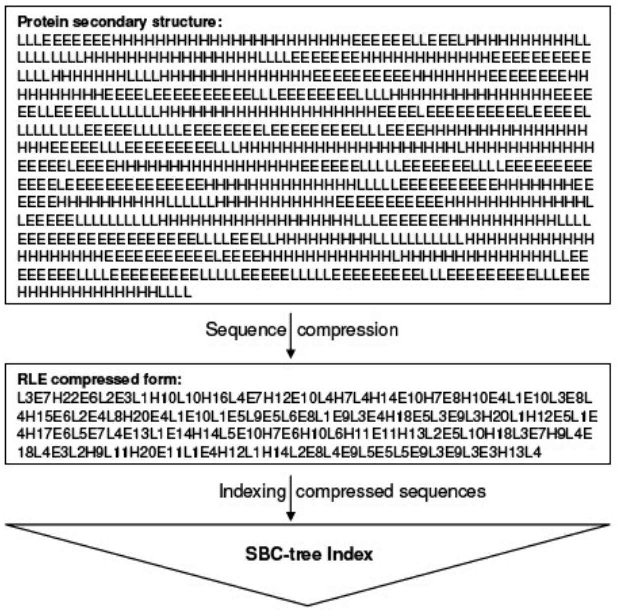

Protein secondary structure:
Sequence compression
RLE compressed form:
L3E7H22E6L2E3L1H10L10H16L4E7H12E10L4H7L4H14E10H7E8H10E4L1E10L3E8L
4H15E6L2E4L8H20E4L1E10L1E5L9E5L6E8L1E9L3E4H18E5L3E9L3H20L1H12E5L1E
4H17E6L5E7L4E13L1E14H14L5E10H7E6H10L6H11E11H13L2E5L10H18L3E7H9L4E
18L4E3L2H9L11H20E11L1E4H12L1H14L2E8L4E9L5E5L5E9L3E9L3E3H13L4
Indexing compressed sequences
SBC-tree Index

Fig. 8: Indexing and querying RLE-compressed sequences [1]

without modifying the database engine. In bdbms, several advanced search operations, e.g., k-nearest-neighbor search, regular expression match search, and substring searching, were implemented.

Biological databases consist of large amounts of sequence data, which need to be stored, indexed and searched efficiently. Compressing sequences improves the system performance as it reduces the size of the data significantly. New techniques for compressing biological sequences and operating over the compressed data without decompressing it is investigated in bdbms [1] .

In bdbms, the process of Run-Length-Encoded sequences are investigated. RLE is a compression technique that replaces the consecutive repeats of a character C by one occurrence of C followed by C's frequency. Fig. 8 illustrates how protein secondary structures are stored in bdbms. SBC-tree (String B-tree for Compressed sequences) is an index structure for indexing and searching RLE-compressed sequences of arbitrary length.

First the sequence is compressed using RLE, and then an SBC-tree index is built over the compressed sequences. Queries will use the index to retrieve the desired data without decompression.

In bdbms, the following challenges regarding the processing of compressed data are planned to be addressed:

Full integration of the SBC-tree index: To fully integrate the SBC-tree index inside bdbms it is planned to address several query processing and optimization issues including: (1) supporting subsequence matching, and (2) providing accurate cost functions for estimating the cost of the index [1]. The supported operations of the SBC-tree index is planned to be extended to include subsequence matching, which is an important operation as it is used in many algorithms such as sequence alignment algorithms.

Processing various formats of compressed data: bdbms supports indexing and querying RLE-compressed sequence data [1]. In the case of sequences where characters have long repeats in tandem, RLE is effective. Other compression techniques can be more effective in compressing the other kinds of data. The plan is to investigate indexing and querying other formats of compressed data in addition to RLE-compressed sequences to efficiently support these data inside bdbms.

8. Discussion

Building a database resource for the E. coli model organism and a protein structure database project are the two applications that have been driving the bdbms project [1]. bdbms is currently being prototyped using PostgreSQL. The A-SQL language and content-based authorization model are currently under development in PostgreSQL. The SP-GiST and SBC-tree access methods are already integrated inside PostgreSQL.

It is clear that the current database systems are not optimal for storing biological data. Biology is not that exact science, and the experiments and results need to be explained and commented to make them more understandable and comparable.

My own knowledge on biology and biological databases is not yet comprehensive enough so that I could analyze the necessity of being able to treat the annotations as first-class objects and make queries based on them, but I strongly agree that annotations are very essential in biological databases.

References

[1] M. Y. Eltabakh, M.Ouzzani, W. G. Aref, *bdbms - A Database Management System for Biological Data*. CIDR 2007: 196-206.

[2] M. Y. Eltabakh, M.Ouzzani, W. G. Aref, A. K. Elmagarmid, Y. Laura-Silva, M. U. Arshad, D. Salt, I. Baxter, *Managing Biological Data Using bdbms*. ICDE 2008.

[3] The Entrez Nucleotide database, 2008. http://www.ncbi.nlm.nih.gov/sites/entrez?db=nuccore

[4] GenBank® http://www.ncbi.nlm.nih.gov/Genbank/index.html

[5] Refseq http://www.ncbi.nlm.nih.gov/RefSeq/