Managing Biological Data using bdbms

Mohamed Y. Eltabakh[1], Mourad Ouzzani[2*], Walid G. Aref[1**], Ahmed K. Elmagarmid[1,2***]
Yasin Laura-Silva[1], Muhammad U. Arshad[3], David Salt[4], Ivan Baxter[4]
[1]Dept. of Computer Science, [2]Cyber Center, [3]Dept. of Electrical and Computer Engineering ,[4]Dept. of Horticulture & Landscape Architecture
Purdue University, West Lafayette
{meltabak, mourad, aref, ake, ylaurasi}@cs.purdue.edu, {marshad@ecn.purdue.edu}, {dsalt, ibaxter}@purdue.edu

***Abstract*— We demonstrate *bdbms*, an extensible database engine for biological databases. bdbms started on the observation that database technology has not kept pace with the specific requirements of biological databases and that several needed key functionalities are not supported at the engine level. While bdbms aims at supporting several of these functionalities, this demo focuses on: (1) Annotation and provenance management including storage, indexing, querying, and propagation, (2) Local dependency tracking of dependencies and derivations among data items, and (3) Update authorization to support data curation. We demonstrate how bdbms enables biologists to manipulate their databases, annotations, and derivation information in a unified database system using the Purdue Ionomics Information Management System (PiiMS) as a case study.**

I. Introduction

Life sciences are a case in point where biological databases have become essential to keep track of various information about experimentation and analysis. However, considerable amounts of biological data are still stored in flat files and spreadsheets and do not use DBMSs. This is mainly due to current database systems lacking key functionalities needed for biological data like efficient and native support for annotations, provenance, and data dependencies. Furthermore, biological databases often rely on community-based curation, evolve with rapidly changing semantics, and lack absolute authority to verify the correctness of information. Thus, the characteristics of annotations that need to be attached to the base data cannot be completely foreseen at design time.

We are building bdbms, an extensible prototype database engine to support key functionalities needed by biological databases. These functionalities are implemented inside postgreSQL. In this demo, we focus on the following features: (1) annotation and provenance management, (2) local dependency tracking, and (3) update authorization. bdbms makes fundamental advances in the use of biological databases through new native and transparent support mechanisms at the database system level.

bdbms treats annotations as first class objects. bdbms allows adding annotations at multiple granularities, i.e., table, tuple, column, and cell levels, archiving and restoring annotations, and querying the data based on the annotation values. We extended SQL into *A-SQL* to support the processing and querying of annotations. A-SQL allows annotations to be seamlessly propagated with query answers with minimal user intervention. bdbms also includes a systematic approach for tracking dependencies among database items. When a database item is modified, bdbms tracks and annotates any other item that is affected by this modification and needs to be re-verified. This feature is particularly desirable in biological databases because many dependencies cannot be computed using coded functions, e.g., stored procedures and functions. Content-based authorization, i.e., the authorization is based not only on the identity of the user but also on the content of the data, is another feature that is integrated into bdbms.

We demonstrate the main features of bdbms using the Purdue Ionomics Information Management System (PiiMS) [1] (http://www.purdue.edu/dp/ionomics), a web-based system that collects and manages high throughput elemental profiling data and associated metadata on the experimental treatment, sample preparation, and instrument settings necessary to interpret results of mass spectrometry analyses in ionomics. PiiMS supports the entire process of planting, growing, harvesting, drying, and analyzing plants.

II. System Overview

We give an overview of the main components of bdbms that we will demonstrate. We briefly describe bdbms' functionalities along with the extended SQL language (A-SQL). More details can be found in [5], [6].

A. Annotation Management

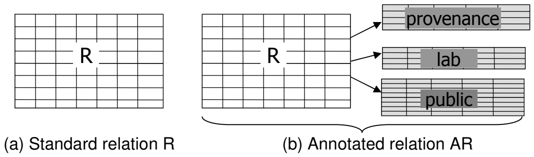
Biologists use annotations as an important mean to communicate and share information about the base data generated from experiments and analyses. Annotations may represent comments about the data inside the database, the source of the data, references to published literature, or the setup and running of experiments. Despite their importance, annotations are not systematically supported as first-class objects inside current DBMSs. Most existing work in annotation management focuses on the propagation of annotations along with the query answer, e.g., [2], [3], [4], [8]. A key distinction of bdbms is that it addresses several aspects of annotation management including:

Storing and organizing annotations – Effective organization and efficient storage of annotations are important features

*The work of Mourad Ouzzani was partly supported by Lilly Endowment

**Walid G. Aref acknowledges the support of the National Science Foundation under grant number IIS-0093116

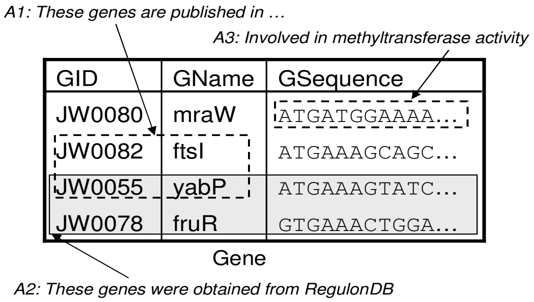
***The work of Ahmed Elmagarmid was partly supported by Lilly Endowment and NSF-ITR 0428168.



(a) Standard relation R

(b) Annotated relation AR

Fig. 1. Annotated relations


A1: These genes are published in …
A3: Involved in methyltransferase activity
Gene
A2: These genes were obtained from RegulonDB

GID	GName	GSequence
JW0080	mraW	ATGATGGAAAA...
JW0082	ftsI	ATGAAAGCAGC...
JW0055	yabP	ATGAAAGTATC...
JW0078	fruR	GTGAAACTGGA...

Fig. 2. Annotations at multiple granularities

for seamless handling of annotations. For example, distinguishing the annotations inserted by the public community from those inserted by the lab members. bdbms allows users to organize their annotations in logical entities called *annotation tables*. Each user relation in the database, representing the base data, may have one or more annotation tables attached to it (See Figure 1). Users can reference specific annotation table(s) at insertion and query times. To create an annotation table, we provide the following command:

```
CREATE ANNOTATION TABLE <ann_table_name>
ON <user_table_name>
SCHEME [Off-table—In-table];
```

To achieve good storage and retrieval performance, we provide three storage schemes to implement the annotation tables based on the granularity of the annotations, namely, *Off-table*, *In-table*, and *Hybrid* schemes.

Adding annotations at multiple granularities – Biologists may need to annotate a single or multiple data items, or entire rows or columns in a table. Figure 2 illustrates an example of a table annotated at different granularities, e.g., A1 annotates GID and GName in two rows. The command:

```
ADD ANNOTATION
TO <annotation_table_names>
VALUE <annotation_body>
PROPAGATION MODE [SINGLY — ALWAYS]
ON <select_statement>
```

allows users to easily add annotations at multiple granularities. The *annotation_table_names* specifies the annotation table(s) into which the annotation will be stored. The *annotation_body* allows annotations to be structured in XML and later queried using XPath, The *propagation mode* specifies the manner in which the annotation will be propagated, for example, SINGLY means that the annotation will not be propagated with the grouping, aggregation, or set operation while ALWAYS mean that the annotation will be always propagated. The output of the *select_statement* specifies the data to be annotated. For example, to add annotation *A1* (Figure 2), we execute the following command:

```
ADD ANNOTATION
TO Gene.GAnnotation
VALUE '< Comment >
          These genes are published in...
          < /Comment >'
ON (Select G.GID, GName
From Gene G
Where GID IN ('JW0082', 'JW0055'));
```

We do not expect most biologists to use A-SQL. Thus, we provide a graphical interface, to be demonstrated, that allows users to graphically select the cells they want to annotate and then insert the annotation value. The highlighted cells are mapped to one or more ADD ANNOTATION commands to be executed by the bdbms engine.

Archiving and restoring annotations – In the course of maintaining a biological database, some annotations may become invalid or unnecessary for the functioning of the database but may still be useful for historical reasons or some possible future usage. Thus, we allow archiving and restoring annotations without having to delete them. Archived annotations will not be propagated to end-users along with the query answers while restored annotations will be propagated as usual. Users can issue the commands:

```
ARCHIVE ANNOTATION
FROM <annotation_table_names>
[BETWEEN <time1> AND <Time2>]
ON <select_statement>
      and
RESTORE ANNOTATION
FROM <annotation_table_names>
[BETWEEN <time1> AND <Time2>]
ON <select_statement>
```

to archive and restore annotations, respectively. In the demonstrated GUI, users can first query annotations and then select which specific annotations to archive or restore.

Propagating annotations with query answers – To facilitate the propagation of annotations, we introduce new query operators and extend the semantics of the existing ones. The general form of A-SQL select queries is:

```
SELECT [DISTINCT] C_i, C_j, ..., [PROMOTE(C_k, ...)]
FROM Relation_name [ANNOTATION(S_1, S_2, ...)], ...
[WHERE <conditions>]
[GROUP BY <data_columns>
       [HAVING <condition>]
[FILTER <annotation_condition>]
```

For example, the ANNOTATION operator specifies which annotation table(s) to be considered in the query. The PROMOTE operator allows to keep annotations from columns not in the projection list to be propagated with the query answer. The FILTER operator drops the annotations that do not satisfy the filter condition.

For example, consider the following query:

```
SELECT GeneID, GeneName, PROMOTE(Sequence)
FROM GENE[ANNOTATION(Glab)]
WHERE GeneID IN ('JW0080', 'JW0055')
AND xpath_bool(Glab,'Root/Annotation/User ="ADMIN"')
FILTER xpath_bool(Glab,'Root/Annotation/TimeStamp >"Jan-01-2007"');
```

In this query, only annotations from *Glab* will be propagated. The query selects the GeneID and GeneName from base table GENE where (1) GeneID equals 'JW0080' or 'JW0055', and (2) the gene entry has annotations inserted

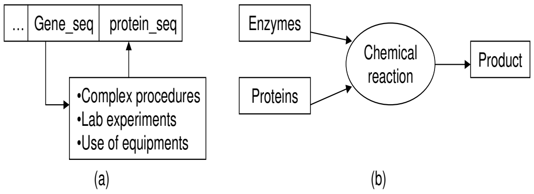


Fig. 3. Local dependency tracking

by user 'ADMIN' in Glab. For each output tuple, the query reports only the annotations inserted after 'Jan-01-2007'. The annotations on the Sequence column will be propagated with the query answer although that column is not projected.

B. Local Dependency Tracking

It is often the case that data in biological databases are dependent on or derived from other data. The challenge is that most of these derivations cannot be simply modeled using coded functions, e.g., stored procedures or database triggers. For example in Figure 3, protein sequences are derived from gene sequences using lab experiments and/or prediction tools that cannot be coded as a function. If a gene sequence is modified, the corresponding protein sequence(s) may become invalid. Similarly, we may store information about chemical reactions, e.g., substrates and reaction parameters. If any of these information is modified, then products of the reaction may become invalid. It is thus important to automatically track such dependencies and maintain the consistency of the data without burdening the users with extra checks.

bdbms enables the modeling of dependencies and derivations using the new concept of *Procedural Dependency*, an extension to *Functional Dependencies*. Procedural dependency allows to specify the dependency module or procedure and its characteristics, e.g., executable by the database or not, and invertible or not. For example, the following rule states that the protein_sequence depends on the gene_sequence through the lab experiment E that is neither executable by the database nor invertible:

$$Gene.Sequence\frac{Lab\ experiment\ E}{(non-exec,non-invert)}Protein.Sequence$$

Such rules allows bdbms to track which items can be automatically re-computed and which items need to be marked as *out-dated* whenever a change occurs in the database. As a result, bdbms provides two important functionalities: (1) reporting out-dated data that needs to be re-evaluated, and (2) annotating query answers to highlight any out-dated data that is part of the results.

C. Content-based Authorization

In current DBMSs, users get permission to execute certain operations based on their identity, i.e., grant/revoke access model [7]. Biological databases are usually a community-based and shared effort which may not fit with this model. For example, if only the lab administrator can modify the database, then (s)he becomes a bottleneck. Also, if all lab members can modify the data without revision, the credibility and authenticity of the data may be compromised.

bdbms provides a monitoring system, termed *Content-based Authorization*, where the authorization is based on the identity of the user as well as the content of the modified data. The database administrator can turn the content-based approval feature ON or OFF for a certain table using the two following commands:

```
START CONTENT APPROVAL
ON <table_name>
APPROVED BY <user/group>
      and
STOP CONTENT APPROVAL
ON <table_name>
```

The content-based approval mechanism maintains a log of all update operations, i.e., INSERT, UPDATE, and DELETE. All non-approved updates will be visible with an annotation mentioning they were not approved yet. The logs are then revised by authorized users to approve/disapprove the operations. If an operation is disapproved, bdbms executes an inverse operation that negates the effect of the original operation. This inverse operation is automatically generated and stored when the original operation is executed.

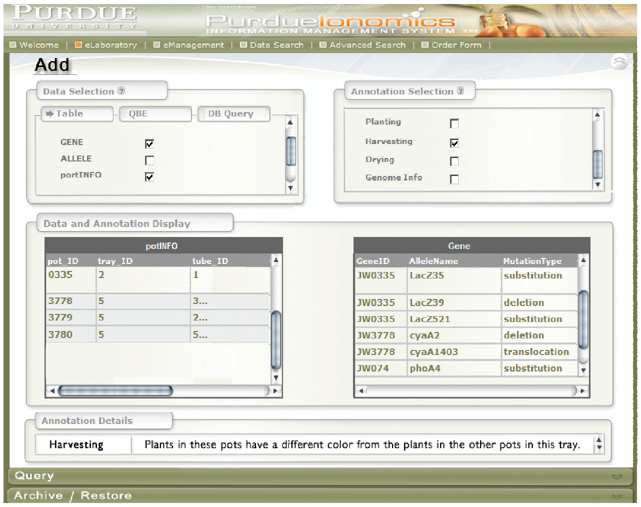
III. Demonstration Scenarios

We demonstrate the functionalities of bdbms in the context of PiiMS [1] by replacing its underlying database engine by bdbms. PiiMS helps understand how plants take up, transport and store their nutrient and toxic elements, collectively known as the ionome. PiiMS main functionalities include: (i) collecting and managing elemental profiling data and associated metadata on the experimental treatment, sample preparation, and instrument settings, (ii) supporting the entire process of planting, growing, harvesting, drying, and analyzing of plants; (iii) providing integrated workflow control and analysis to facilitate high-throughput data acquisition, along with integrated tools for data search and visualization for hypothesis development.

We demonstrate bdbms through a web-based GUI that allows to add/archive/restore annotations and query/browse base data and annotations, define/track procedural dependencies, and define tables subject to content-based authorization and track/approve/disapprove updates.

Scientists in the ionomics lab while harvesting plants from different pots noticed that some plants had their leaves with a different color from the rest of the plants. Thus, they decided to record this information in the database as an annotation to the pot table *potINFO*. This information may be helpful in explaining the concentration of the elements being tracked. Using the GUI, we demonstrate how scientists can add new annotations, i.e., highlighting certain cells in the query results and attaching an annotation to them (Figure 4). We also show how users can query the annotations.

PiiMS allows to perform several statistical analyses based on the data being provided by the ICP-MS mass spectrometer instrument. Results of these analyses are stored in different tables. An example is the % differences in concentrations for each element which are stored in table *percentDIFF*.


PURDUE UNIVERSITY
PurdueIonomics INFORMATION MANAGEMENT SYSTEM
Welcome | eLaboratory | eManagement | Data Search | Advanced Search | Order Form |
Add
Data Selection
Table
QBE
DB Query
Annotation Selection
Data and Annotation Display
Annotation Details
Query
Archive / Restore

GENE	☑
ALLELE	□
portINFO	☑

Planting	□
Harvesting	☑
Drying	□
Genome Info	□

potINFO

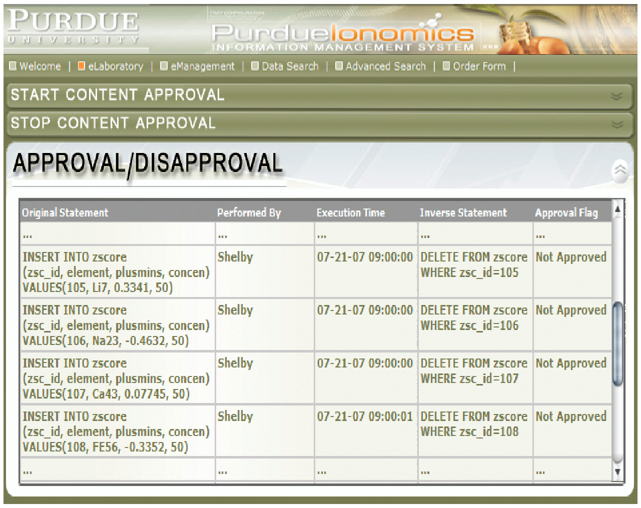
pot_ID	tray_ID	tube_ID
0335	2	1
3778	5	3...
3779	5	2...
3780	5	5...

Gene

GeneID	AlleleName	MutationType
JW0335	LacZ35	substitution
JW0335	LacZ39	deletion
JW0335	LacZ521	substitution
JW3778	cyaA2	deletion
JW3778	cyaA1403	translocation
JW074	phoA4	substitution

Harvesting	Plants in these pots have a different color from the plants in the other pots in this tray.

Fig. 4. Adding Annotations


PURDUE UNIVERSITY
PurdueIonomics INFORMATION MANAGEMENT SYSTEM
Welcome | eLaboratory | eManagement | Data Search | Advanced Search | Order Form |
START CONTENT APPROVAL
STOP CONTENT APPROVAL
APPROVAL/DISAPPROVAL

Original Statement	Performed By	Execution Time	Inverse Statement	Approval Flag
...	...	...	...	...
INSERT INTO zscore (zsc_id, element, plusmins, concen) VALUES(105, Li7, 0.3341, 50)	Shelby	07-21-07 09:00:00	DELETE FROM zscore WHERE zsc_id=105	Not Approved
INSERT INTO zscore (zsc_id, element, plusmins, concen) VALUES(106, Na23, -0.4632, 50)	Shelby	07-21-07 09:00:00	DELETE FROM zscore WHERE zsc_id=106	Not Approved
INSERT INTO zscore (zsc_id, element, plusmins, concen) VALUES(107, Ca43, 0.07745, 50)	Shelby	07-21-07 09:00:00	DELETE FROM zscore WHERE zsc_id=107	Not Approved
INSERT INTO zscore (zsc_id, element, plusmins, concen) VALUES(108, FE56, -0.3352, 50)	Shelby	07-21-07 09:00:01	DELETE FROM zscore WHERE zsc_id=108	Not Approved
...	...	...	...	...

Fig. 5. Approving/Disapproving Updates

The settings of the ICP-MS spectrometer depends on several information from the line catalog table *lineCAT* (a line defines the seed being planted). If any of the information in *lineCAT* is changed, the information in *percentDIFF* may become invalid and another run of the mass spectrometer may be required. We demonstrate how the corresponding rule(s) are defined and how data is tracked.

In the analysis stage, data from the instrument is uploaded into PiiMS. This important stage consists of several steps where different parameters set by the lab technician are used to perform the analysis including the generation of z-score values and % difference in concentrations for each element. Any lab technician can perform this stage and upload the data to PiiMS. This data will be then readily available for other users. However, due to its importance and the risk of errors in setting the different parameters of the analysis this data is flagged as not validated yet. This data stays flagged until an authorized user validates or invalidates these updates. In the demo, we demonstrate the logging mechanism of the database operations of non-authorized users and how authorized users can approve/disapprove the operations (Figure 5).

References

[1] I. Baxter, M. Ouzzani, S. Orcun, B. Kennedy, S. S. Jandhyala, and D. E. Salt. Purdue Ionomics Information Management System. An Integrated Functional Genomics Platform. *Plant Physiol.*, 143(2), 2007.

[2] D. Bhagwat, L. Chiticariu, W. Tan, and G. Vijayvargiya. An annotation management system for relational databases. pages 900–911, 2004.

[3] P. Buneman, A. P. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGOMD*, 2006.

[4] P. Buneman, S. Khanna, and W.-C. Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002.

[5] M. Eltabakh, M. Ouzzani, and W. Aref. bdbms: A database management system for biological data. In *CIDR*, pages 196–206, 2007.

[6] M. Eltabakh, M. Ouzzani, W. Aref, A. Elmagarmid, and Y. Lura-silva. Supporting annotated relations. *Purdue University, Technical Report, CSD TR07-025*, 2007.

[7] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems (TODS)*, 1(3):242–255, 1976.

[8] W.-C. Tan. Containment of relational queries with annotation propagation. In *DBPL*, 2003.