A Database Server for Next-Generation Scientific Data Management

Mohamed Y. Eltabakh [#1]

Advisors: Walid G. Aref [#2], Ahmed K. Elmagarmid [#3]

[#]*Computer Science Department, Purdue University*
West Lafayette, IN, USA
[1]meltabak@cs.purdue.edu
[2]aref@cs.purdue.edu
[3]ake@cs.purdue.edu

***Abstract*— The growth of scientific information and the increasing automation of data collection have made databases integral to many scientific disciplines including life sciences, physics, meteorology, earth and atmospheric sciences, and chemistry. These sciences pose new data management challenges to current database system technologies. The thesis work presented in this paper proposes a database server for next-generation scientific data management. The proposed sever realizes two core requirements in scientific databases, mainly, (1) Annotation management, and (2) Complex dependencies involving human actions. In the paper, we discuss the challenges involved in each of these requirements and present the key contributions and main results in each of the two fronts.**

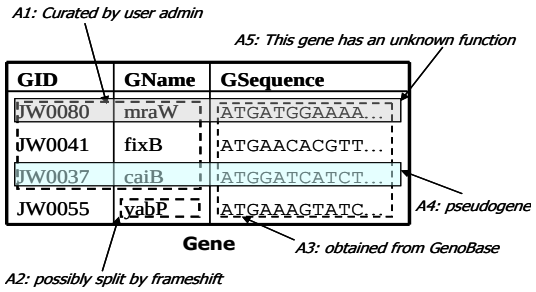


Fig. 1. Example of annotations

I. Introduction

Scientific databases are used at various stages of scientific experimentation and analysis, e.g., for depositing raw data, storing experiments and results of analysis processes, maintaining interpretations and comments about the data, tracking the provenance and history of data changes, and querying the existing data. As such, scientific databases represent the backbone of scientific discoveries. However, current database technologies have not kept pace with the proliferation and specific requirements of scientific databases [10], [22], [29]. In many cases, scientists tend to store their data in flat files or spreadsheets mainly because current database systems lack several functionalities that are needed by scientific applications, e.g., annotation and provenance management, history tracking of data changes, management of complex dependencies among the data items, and efficient access to a large pool of complex and non-traditional data types. The thesis work presented in this paper addresses two core requirements in scientific databases, mainly, (1) Annotation management, and (2) Complex dependencies involving human actions. In this section, we give a background on and discuss the challenges involved in each of the two requirements.

Annotation Management: Annotations play a key role in understanding and curating scientific databases. Annotations may represent comments, descriptions, warning or error messages, questions about any subset of the data, lineage information, among several others. Therefore, annotation management is a vital mechanism for sharing knowledge and building an interactive and collaborative environment among database users and scientists. Annotation management involves several challenges that include:

(1) ***Storing multi-granular annotations:*** Annotations can be large in size and attached to the data at various granularities, i.e., users may annotate an entire table, entire column, a subset of the tuples, a few cells, or a combination of these (See Figure 1). Because of this combinatorial nature of the annotations, efficient storage schemes are needed to avoid replicating the annotations.

(2) ***Adding annotations and defining behaviors:*** A key requirement in annotation management is to provide declarative mechanisms to annotate the data and to specify how annotations behave under different database operations. For example, users may want the newly inserted data to be annotated automatically if it satisfies certain criteria, or may want annotations to be deleted automatically when the base data get modified. Moreover, as time passes, annotations may become worthless or obsolete. Therefore, the DBMS needs to provide mechanisms to archive existing annotations. For example, annotation *A5* in Figure 1 may become obsolete when the function of the annotated gene is defined. Hence, users may want to stop propagating this annotation along with the query result.

(3) ***Querying and Propagating annotations seamlessly:*** Users want to propagate the annotations along with the query results and to query the data based on the annotation values seamlessly without complicating their queries. If annotation propagation and querying are delegated to end-users (or applications) without any database system support, then users' queries may become complex, user-unfriendly, and would

This research was partially supported by NSF Grant Numbers IIS 0916614 and IIS 0811954, and by Purdue Cyber Center.

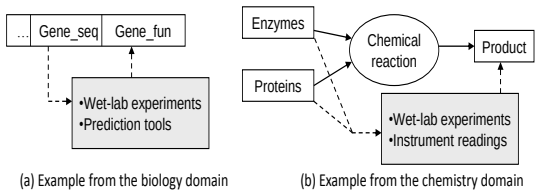


Fig. 2. Complex dependencies in scientific applications

require users to be aware of the internal representation of the annotations. Instead, the DBMS should provide annotation-aware query operators that facilitates propagation and querying the annotations along with the data items.

Complex Dependencies Involving Human Actions: Scientific databases are full of complex dependencies among the data items that may involve human actions, e.g., conducting a wet-lab experiments, collecting instrument readings, or taking manual measurements (See Figure 2). In traditional derived data that are stored inside the database, e.g., deriving age from the date-of-birth attribute, simple procedures internal to the database system can be coded and executed automatically to maintain the consistency of the data. In contrast, when the derivations involve human actions, these derivations cannot be coded within the database. Moreover human actions may take long time to prepare for and perform. As a result, a database update operation may render all dependent and derived data items invalid (inconsistent) for unbounded periods of time while the data still need to be made available for querying. For example (refer to Figure 2), gene functions are derived from gene (DNA) sequences. If a gene sequence is modified, the corresponding gene function, may become invalid. Similarly, we may store descriptions of chemical reactions, e.g., substrates, reaction parameters, and products. If any of the substrates in the reaction are modified, the products of the reaction become invalid until the involved wet-lab experiments are re-conducted to re-evaluate the reaction product.

Supporting dependencies that involve human actions (more broadly *real-world activities*) within the database engine include the following challenges:
(1) ***Defining dependencies based on activities:*** New mechanisms are needed to enable users to define real-world activities inside the database system and to express the dependencies among the data items on these activities.
(2) ***Tracking potentially invalid and outdated data items:*** When a database item is modified, the DBMS needs to keep track of all dependent and derived data items that become temporarily invalid. Moreover, the DBMS should keep track of the real-world activities that need to be performed to re-validate the data.
(3) ***Extended querying mechanisms:*** Extended semantics for query operators are needed to automatically alert users when the query results contain invalid data items, and to enable evaluating queries on either valid data only or both valid and potentially invalid data.
(4) ***Defining re-validation mechanisms:*** The DBMS needs to provide systematic mechanisms for re-validating and/or correcting the outdated data items.

II. Related Work

Annotation management has been the focus of recent research as a key requirement in supporting scientific databases [5], [22], [23], [28]. Annotation management in the context of relational databases has been addressed in previous works, e.g., [2], [7], [9], [21], [30], [8]. The two main systems are DBNotes [2], [9], [3], and MONDRIAN [20], [21]. DBNotes proposes an extension to SQL, termed *pSQL*, that extends the querying capabilities by adding a PROPAGATE clause to the SELECT statement that allows users to specify how to propagate the annotations along with the query answers. DBNotes proposes several propagation schemes that allow propagating the annotations on equivalent values resulted from operations such as union and equi-joins. MONDRIAN [20], [21] proposes an algebra, termed *color algebra*, that extends annotating single values to annotating multiple related values with the same annotation. Both systems allow users to add conditions to restrict the propagated annotations and to query the data based on the annotation values. Although current annotation management systems focus on querying and propagating the annotations along with the query results, there are several aspects of annotation management that have not been addressed adequately, e.g., efficient storage techniques, handling multi-granular annotations, processing different types of annotations with different behaviors, and mechanisms for adding and archiving annotations. The thesis work presented in this paper proposes several efficient techniques and algorithms that address these aspects.

In the context of modeling and tracking the dependencies among the database items, the concept of functional dependencies (FDs), e.g., [25], [31], cannot be used to keep track of the dependencies that involve real-world activities. The reason is that the derivations that involve real-world activities cannot be codded inside the database system, e.g., using database triggers, regardless of how well the database schema is designed. Other related works are in the areas of long-running transactions, e.g., [13], [26], *active databases*, e.g., [1], [12], and Provenance management, e.g., [2], [4], [6], [11], and uncertain and fuzzy database systems, e.g., [19], [32]. However, none of these systems can model or capture the dependencies among the data items that involve real-world activities. Hence, they neither keep track of the derived outdated data that requires re-evaluation and re-validation nor reflect the status of the invalid data items on the query results. In history-tracking and multi-version databases, e.g.,[24], an update operation creates new versions of the modified tuples and maintains the existing values as old versions. The update operation may still require executing some real-world activities to update other dependent data items. Hence, the most recent version of the database may still exposes potentially invalid and outdated data for querying. Some systems, e.g., the checkout\checkin model proposed in [27], query old versions of the data while the required changes are being performed on an offline version of the database. The drawbacks of this approach include violating the need for making users' updates available as early as possible, hiding possible data corrections for unbounded long delays, and resolving any consistency issues outside the DBMS, i.e., data conflicts are resolved at the checkin time using version-control systems outside the DBMS.

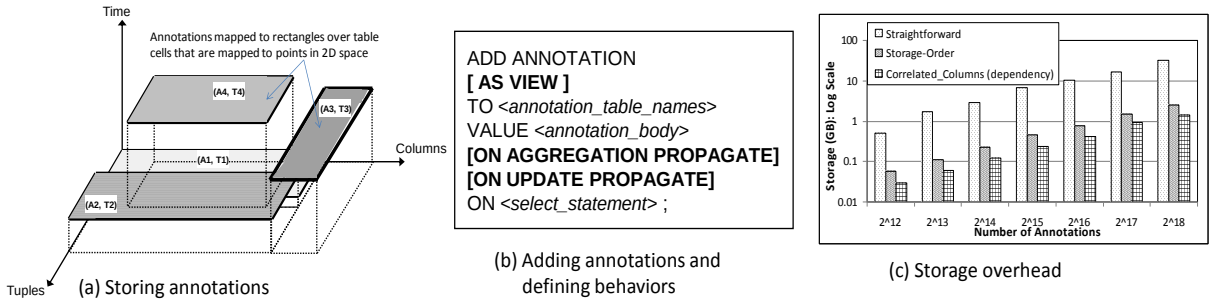


Fig. 3. Examples of the proposed annotation management techniques and sample performance results

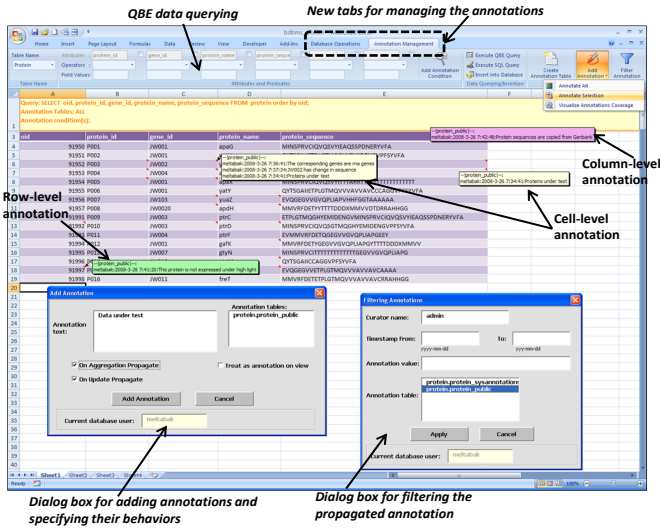


Fig. 4. Managing annotations via GUI

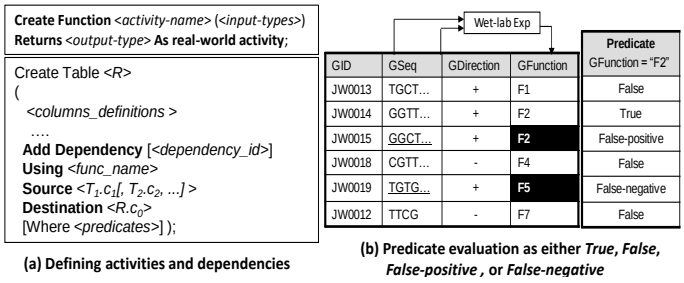


Fig. 5. Supporting real-world dependencies

III. Our Research Contributions

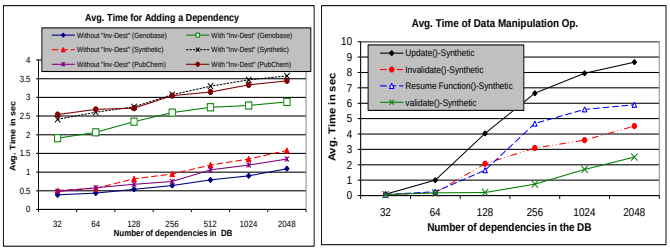
In this section, we present the key contributions in realizing the proposed prototype server [14], [15], [16], [17], [18].

A. Annotation Management

We proposed a framework for efficient management of large volumes of multi-granular annotations [14], [18]. The framework includes: (1) New storage schemes that allow efficient and compact representation of multi-granular annotations. The proposed storage schemes map the table cells in the database to points in a two-dimensional space, and map the annotations to rectangles over these points as illustrated in Figure 3(a). In this representation, a single annotation is represented by the minimum number of *maximum bounding rectangles (MBR)* that can cover the annotated table cells. For example, although annotation *A2* in Figure 1 is attached to six table cells, it can be represented by one rectangle that covers these table cells. (2) Declarative mechanisms to support adding, archiving, and querying various types of annotations (e.g., see the adding mechanism in Figure 3(b)). We define three types of annotations: *snapshot*, *view*, and *join* annotations that inherit different behaviors. For example, in contrast to *snapshot annotations* that apply to a data instance, *view annotations* automatically annotate newly inserted data if they satisfy certain conditions, whereas *join annotations* are attached to data items across multiple relations and propagate only when these data items appear together in the query results. (3) Query processing and optimization techniques to efficiently support querying and propagating the annotations along with the query results. (4) New constructs such as *ON UPDATE PROPAGATE* and *ON AGGREGATION PROPAGATE* that allow users to control how annotations behave under different database operations (Refer to Figure 3(b)). All the proposed constructs and functionalities are realized inside PostgreSQL. Since most scientists prefer to use graphical interfaces over using direct SQL commands, we provided an easy to use and intuitive GUI using Excel sheets that facilitate performing the proposed functionalities and visualizing the annotations as illustrated in Figure 4. A sample performance analysis is depicted in Figure 3(c) in which the proposed storage schemes achieves more than an order of magnitude reduction in storage compared to the straightforward scheme in which annotations are replicated and stored with each individual cell.

B. Supporting Dependencies Involving Real-world Activities

We proposed a prototype database server for supporting dependencies that involve real-world activities while maintaining the consistency of the derived data under update and query operations [15]. The prototype server includes the following features: (1) New mechanisms and constructs that enable users to register real-world activities in the database system and to express the dependencies among the data items on these activities (See Figure 5(a)). The real-world activities are mapped to functions inside the database of type *real-world activity*. The dependencies that involve these functions are called *real-world dependencies*. (2) Mechanisms to keep track of the potentially invalid data items and reflecting their status in the query results. Referring to the example in Figure 5(b),



(a) Avg. Time for adding a new dependency (b) Avg. Time to perform a manipulation op.

Fig. 6. Performance results for adding new dependencies and manipulating the database items.

the gene function values are inferred from both the gene sequence and direction values through a wet-lab experiment. When a gene sequence value is modified (e.g., the underlined values in *GSeq* column), the corresponding gene function values are automatically marked as *potentially invalid* (e.g., the dark-marked values in *GFunction* column) until the wet-lab experiment is re-conducted. (3) New semantics for query operator that enable evaluating queries on either valid data only (no false-positives), or both valid and potentially-invalid data (include false-positives and possibly false-negatives). We introduce the notion of *false-positive* and *false-negative* query evaluation as illustrated in Figure 5(b). For example, if a tuple satisfies the query predicate based on a potentially-invalid value, then that tuple evaluates to *false-positive* in contrast to *true*(e.g., the 3^{rd} tuple in Figure 5(b)). Similarly, if a tuple disqualifies the query predicate based on a potentially-invalid value, then that tuple evaluates to *false-negative* in contrast to *false* (e.g., the 5^{th} tuple in Figure 5(b)). (4) New mechanisms for invalidating and revalidating the data items, and for keeping track of the real-world activities that need to be performed in order to re-evaluate and re-validate the outdated data items. In Figure 6, we present sample performance results that show the overhead involved in adding new dependencies to the database and in manipulating the database items, e.g., updating, invalidating, and re-validating the data. The results demonstrate the scalability of the the proposed system under a large number of dependencies and shows the practicality and feasibility in its realization.

IV. Conclusions and Future Work

In this dissertation, we proposed a database server for next-generation scientific data management. We addressed two core requirements which are: management of large-volume multi-granular annotations, and management of data dependencies that involve human actions. We realized both features via extensions to PostgreSQL. The experimental studies and performance analysis show the feasibility and practicality of the proposed features and their performance gains over other existing techniques and algorithms. Our plans for future work include: (1) management of scientific workflows, (2) annotation and provenance management in XML databases, (3) management of data conflicts in scientific databases, and (4) design of peta-scale data centers over cloud and massively parallel computing architectures.

References

[1] A. Aiken, J. Widom, and J. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. In *SIGMOD*, pages 59–68, 1992.

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

[3] D. Bhagwat, L. Chiticariu, W. Tan, and G. Vijayvargiya. An annotation management system for relational databases. *VLDB Journal*, 14(5), 2005.

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

[5] P. Buneman, J. Cheney, and W.-C. Tan. Curated databases. In *PODS*, pages 1–12, 2008.

[6] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. *Lecture Notes in Computer Science*, 1973:316–333, 2001.

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

[8] A. P. Chapman, H. V. Jagadish, and P. Ramanan. Efficient provenance storage. In *SIGMOD*, pages 993–1006, 2008.

[9] L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. Dbnotes: a post-it system for relational databases based on provenance. In *SIGMOD*, pages 942–944, 2005.

[10] P. Cudre-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R. Simakov, E. Soroush, P. Velikhov, D. L. Wang, M. Balazinska, J. Becla, D. DeWitt, B. Heath, D. Maier, S. Madden, M. Stonebraker, and S. Zdonik. A demonstration of scidb: A science-oriented dbms. In *VLDB*, 2009.

[11] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *VLDB*, pages 471–480, 2001.

[12] U. Dayal. Active database management systems. *SIGMOD Rec.*, 18(3):150–169, 1989.

[13] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. *SIGMOD Rec.*, 19(2):204–214, 1990.

[14] M. Y. Eltabakh, W. G. Aref, A. K. Elmagarmid, M. Ouzzani, and Y. N. Silva. Supporting annotations on relations. In *EDBT*, pages 379–390, 2009.

[15] M. Y. Eltabakh, W. G. Aref, A. K. Elmagarmid, Y. N. Silva, and M. Ouzzani. Supporting real-world activities in database management systems, short paper. In *ICDE*, 2010.

[16] M. Y. Eltabakh, W.-K. Hon, R. Shah, W. G. Aref, and J. S. Vitter. The sbc-tree: an index for run-length compressed sequences. In *EDBT*, pages 523–534, 2008.

[17] M. Y. Eltabakh, M. Ouzzani, and W. G. Aref. bdbms - a database management system for biological data. In *CIDR*, pages 196–206, 2007.

[18] M. Y. Eltabakh, M. Ouzzani, W. G. Aref, A. K. Elmagarmid, Y. N. Silva, M. U. Arshad, D. Salt, and I. Baxter. Managing biological data using bdbms. In *ICDE*, pages 1600–1603, 2008.

[19] J. Galindo, A. Urrutia, and M. Piattini. Fuzzy databases: Modeling, design, and implementation. *Idea Group Publishing*, 2006.

[20] F. GEERTS and J. V. D. BUSSCHE. Relational completeness of query languages for annotated databases. *DBPL*, 4797:127–137, 2007.

[21] F. Geerts, A. Kementsietsidis, and D. Milano. Mondrian: Annotating and querying databases through colors and blocks. In *ICDE*, page 82, 2006.

[22] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber. Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41, 2005.

[23] H. V. Jagadish and F. Olken. Database management for life sciences research. *SIGMOD Record*, 33(2):15–20, 2004.

[24] D. Lomet, R. Barga, M. Mokbel, and G. Shegalov. Transaction time support inside a database engine. In *ICDE*, pages 35–46, 2006.

[25] D. Maier. Theory of relational databases. In *Comp. Sci. Press*, 1983.

[26] H. Molina and K. Salem. Sagas. *SIGMOD Rec.*, 16(3):249–259, 1987.

[27] M. Ranft, S. Rehm, and K. Dittrich. How to share work on shared objects in design databases. In *ICDE*, pages 575–583, 1990.

[28] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[29] M. Stonebraker, J. Becla, D. J. DeWitt, K.-T. Lim, D. Maier, O. Ratzesberger, and S. B. Zdonik. Requirements for science data bases and scidb. In *CIDR Perspectives*, 2009.

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

[31] J. Ullman. Principles of database and knowledge-base systems. volume 1, 1988.

[32] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. *CIDR*, pages 262–276, 2005.