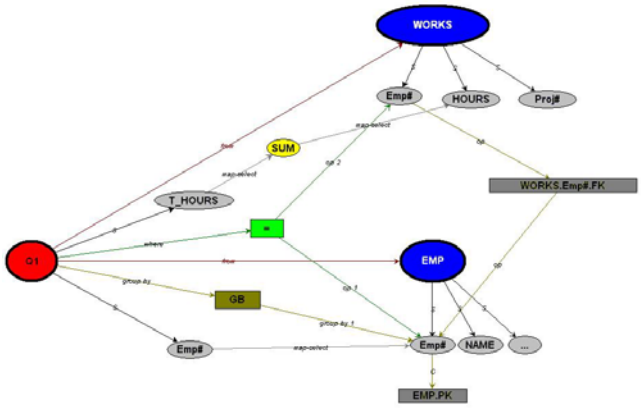
Management of the Evolution of Database-Centric Information Systems

Panos Vassiliadis
Univ. of Ioannina,
Dept. of Computer Science,
Ioannina, Hellas
pvassil@cs.uoi.gr,
http://www.cs.uoi.gr/~pvassil

George Papastefanatos, Yannis Vassiliou, Timos Sellis
National Technical Univ. of Athens,
Dept. of Electrical & Computer Engineering,
Athens, Hellas
{gpapas, yv, timos}@dblab.ece.ntua.gr,
http://www.dblab.ece.ntua.gr/

1 Introduction

Assume a database surrounded by a large variety of applications depending on it. For example, data entry or query forms are used by hundreds of users updating or querying information and complex workflows that operate in the enterprise frequently pose queries to the database. *What happens if we delete a popular attribute from a relation in the database?* Typically, all applications accessing this attribute will crash. Take for example, the case in Figure 1, where we use an graph-based sketch representation of two relations (WORKS and EMP) and a query Q1. Observe the attribute EMP.Emp#, which is the Primary Key (PK) of relation EMP. Its role is such that it participates as (1) a grouper in the group-by query Q1, (2) a part of join condition between relations EMP and WORKS, (3) a part of the result of Q1, while, at the same time, (4) it is also part of a foreign key in the database. Clearly, the impact of deleting this attribute is significantly higher for the structure of the database and its surrounding applications, than, e.g., attribute WORKS.Hours. At the same, if for some reason we would like to alter the primary key of relation EMP, this would incur even higher reconstruction costs of the database (both due to the presence of query Q1 and the foreign key among relations WORKS and EMP).




```
SELECT E.Emp#, SUM(HOURS)
       AS T_HOURS
FROM EMP E, WORKS W
WHERE E.Emp# = W.Emp#
GROUP BY E.Emp#
```

Fig. 1. A graph for a small part of a database and a query over it

The abovementioned example concerns one of the many possible facets of database evolution, specifically, *schema modification*. In the rest of this section, we will describe what we believe to be an interesting research agenda for the database community, in the larger context of database evolution.

1.1 Study of the fundamental laws of evolution

A fundamental problem in the area of database evolution is the lack of empirical studies. This has also resulted in the inability to be able to derive patterns, or laws of evolution for database systems. To our knowledge there has been exactly one experiment on the evolution of database systems [Sjøb93], which took place almost 15 years ago with duration of 18 months.

To our perception, the following research questions present an interesting research agenda on the topic:

- Can we collect test cases and observe them in order to come up with the **fundamental laws that govern database evolution**?
- Can we establish an **experimental protocol** for monitoring existing real-world databases and discover the way they evolve?
- Can we **collect such results and make them available to the research community** (obviously, without unveiling crucial information that the database owners would like to keep hidden)?

1.2 Principled description of the architecture of a database-centric information system

In [BHP00], the authors introduced model management, which has subsequently become an important topic of the research agenda of the database community. Till then, metadata management had received significant attention from the research community, but with no major practical results in industrial applications. We believe that model management is important for the entire environment of a database (i.e., it should also involve the surrounding applications, too) and not be restricted only to the internals of a database. Thus, the main goal we need to pursue is to **discover a commonly agreed formalism to express the internal structure of a database-centric environment, on the grounds of a well-founded theory**. The main questions that arise in this context are:

- Can we derive a **commonly agreed model** of the structural properties and dynamics of database-centric systems?
- How can we trace the full range of **interdependencies** in the components of a complex database-centric system?
- Can we provide a **scientific foundation** for the architecture of complex database-centric systems?

In our research, we refer to the main construct that keeps the enriched metadata of a database-centric environment as the *Architecture Graph* of the system [AG06]. So far, we have had some preliminary results in the construction of blueprints for data warehouse environments and in the management of the evolution of a database, by exploiting the Architecture Graph [PKVV05, PSTS05, PaVV06]. We refer to the latter issue in the following section.

1.3 Principled response to evolutionary events

Mostly all the work of the research community on database evolution has focused on conceptual models and object-oriented databases [Rod+00], without any treatment of the significantly more difficult problem of managing a regular relational database which is surrounded by a large number of applications.

The main problem that we have to deal with is:

Given a set of user requirements on the structure, content and future availability of a certain part of data stored in a database, how do we handle events that affect the above properties in order to satisfy all user requirements?

This broad research topic raises the following questions:

- Given a certain event, how do we **forecast its impact** as this is propagated **throughout the whole database**, via module interdependencies?
- How do we handle **conflicts**? E.g., what happens if the administrator needs to delete a certain attribute, while a user has explicitly banned any such action?
- How do we keep **versions** of the database consistent to user views?
- How easily can we **express user requirements** (since the data entry for metadata is always the biggest problem in metadata management)?
- How do we treat evolution (and addition of information in particular), in the **absence of user requirements**?
- How can we perform all the above with minimal effort, for existing systems?

Viewed from another point of view that concerns the **automation of the system's reaction to changes**, the question that arises concerns our ability to derive (semi) automatic mechanisms for the self-monitoring, impact prediction, auto-regulation, and self-repairing of complex information systems?

1.4 Metrics

Given a model that describes the possible evolution of a database, how good is a certain schema that a designer produces? Is design *A* better than design *B*? Evaluating the design of a database for its evolution in the future, given a (probably vague) pattern, or prediction, is a hard research problem.

There is a huge amount of literature devoted in the evaluation of software artifacts [Fent94, FeNe02]. The main idea for the state-of-the-art in design metrics is the adoption of *measure families*, like size, complexity, coupling and cohesion for graph-theoretic system representations. The definition of these measure families is generic, in the sense, that depending on the underlying context, one can define his own measures that fit within one of the aforementioned categories. In order to be able to claim fitness within one of the aforementioned categories, there is a specific list of properties that the proposed measure must fulfill. Still, in the context of evolution, the following questions arise:

- What are the "right" **measures for the quality of the design of a database**, given estimations on its workload and evolution?
- What are the "right" **families of such measures**?
- Is there an underlying **well-founded theory** that supports the above results?

1.5 Design Patterns

Design patterns constitute a principled way of teaching, designing and documenting software systems [GHJV95]. Moreover, design patterns allow us to evaluate the quality of a design by measuring the compliance of a logical schema to a set of underlying patterns. Given a well-founded theory of patterns, the less deviations a schema has from the theory, the less is the risk of **maintenance problems**, since the amount of necessary improvisations the designer makes is reduced. With particular emphasis to the aspect of maintenance (from the software engineering point of view), we believe the following research questions are particularly important:

- Can we come up with a **well-founded theory for design patterns** to guide both database and application designers?

- Can we eliminate **maintenance traps** that occur due to ad-hoc, or unavoidably complex solutions?
- Can we design principled methods for **testing database designs** for the abovementioned purposes?
- Can we devise a **documentation method** that makes the administrator's/developer's life easier as user requirements, data, systems, and databases evolve?

2 Adapting queries and views to database evolution

So far we have had preliminary results in the management of database evolution and specifically in the adaptation of queries and views to events that alter the underlying database schema [PKVV05, PaVV06]. In this section, we briefly discuss the main results of our approach so far in an informal fashion. The interested reader is referred to [PaVV06] for a detailed presentation of our approach and a discussion of related work.

The main mechanism towards handling schema evolution is the annotation of the constructs of a graph that represents the internal structure of the system with operators that handle schema evolution. Therefore, we first introduce a graph modeling technique that uniformly covers relational tables, views, database constraints and SQL queries as first class citizens. The proposed technique provides an overall picture not only for the actual database schema but also for the architecture of a database system as a whole, since queries are incorporated in the model. Moreover, we distinguish the following essential components, which are included in our model: relations, conditions (covering both database constraints and query conditions), queries and views. The proposed modeling technique represents all the aforementioned database parts as a directed graph with the aforementioned entities being represented as nodes and edges covering different semantics of their interrelationships (e.g., part-of, value mapping edges, etc).
We, then, formulate a set of rules that allow the identification of the impact of changes to database relations, attributes and constraints and propose a semi-automated way to respond to these changes. The *impact* of the changes involves the software built around the database, mainly queries, stored procedures, triggers etc., which are affected in two ways: (a) *syntactically*, meaning that it is possible that the execution of the code will produce a compilation/execution failure and (b) *semantically*, meaning that a change in the database can affect the semantics of the software built around it. We abstract software modules where SQL is embedded within a host language and treat every such module as a set of SQL queries. The *rules* that we propose are annotations of the graph that determine the policy to be followed in the case of an event that modifies the graph. The combination of events and annotations determines the **action** to be followed for the handling of the potential change, i.e., the adaptation of the query to the change.
The space of potential **events** is quite simple and comprises the space of *hypothetical actions* (*addition*/*deletion*) over specific database graph constructs (*relations*, *attributes* and *conditions*). For each of the above events, the administrator annotates the appropriate graph constructs (i.e., nodes and edges) with **policies** that dictate the way they will regulate the change. Two kinds of policies are defined: (a) *propagate* the change, meaning that the graph must be reshaped to adjust to the new semantics incurred by the event and (b) *block* the change, meaning that we want to retain the old semantics of the graph and the hypothetical event must be blocked or, at least, constrained, through some rewriting that preserves the old semantics [NiLR98, VeMP04].
In order to give a flavor of our approach, we start with the simplest example of an SPJ query, specifically the query `SELECT * FROM EMP`. Assume now that the designer extends

the relation EMP with a new attribute PHONE. When an attribute is added to a relation of the underlying schema, we need to identify the queries to which the addition must be reflected and propagated. Both the current database systems and the state of the art in research do not react to this change, but rather, they let the designer/administrator propagate the change to any queries he thinks they should be modified to include the extra attribute. Eventually, the designer/administrator is obliged to rewrite the queries, which are to be modified, by adding appropriately the extra attribute to their syntax. This treatment is mainly due to the fact that (a) the addition of an attribute does not *syntactically* affect the involved queries (i.e., the existing queries can still be executed without any problem) and (b) up to now, we do not have any mechanism to tell the system that once an attribute is added to a relation, it must also be added to certain queries that access this particular relation.

Based on these, in the presence of an addition of an attribute, an impact prediction system must trace all queries and views that are potentially affected and ask the designer/administrator to decide upon which of them must be modified to incorporate the extra attribute. Extending the current modeling, for each element potentially affected by the addition, we annotate its respective graph construct (i.e., nodes, edges) with the aforementioned policies. According to the policy defined on each construct the respective *action* is taken to adjust the query to the change. Therefore, for the event of attribute addition, the policies defined on the query and actions taken according to each policy are:

- *Propagate attribute addition*. In this case, when an attribute is added to a relation appearing in the FROM clause of the query, this addition must be reflected to the SELECT clause of the query.
- *Block attribute addition*. In this case, the addition to the relation must be ignored and the query is immune to the change. The SELECT * clause must be rewritten to SELECT $A_1, \ldots, A_n$ without the newly added attribute.
- *Prompt*. In this case (default, for reasons of backwards compatibility) the designer/administrator must handle the impact of the change manually, like what happens now in database systems.

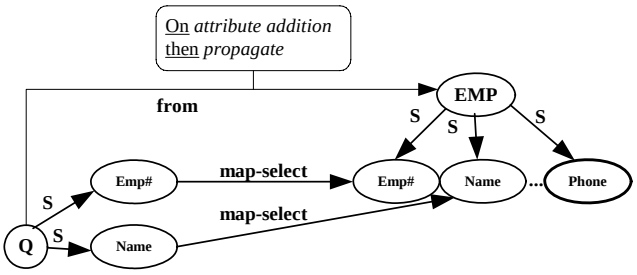


Fig. 2: Propagating addition of attribute PHONE to the schema of the query

The graph of the query SELECT * FROM EMP is shown in Figure 2. The annotation of the FROM edge as *propagating addition* indicates that the addition of PHONE node to relation EMP will be propagated to the query and the new attribute is included in the SELECT clause of the query. If a FROM edge is not tagged with this additional information, then a *default case* is assumed and the designer/administrator is prompted to decide.

Different policies capturing the same event can be defined on different elements of the graph --e.g., a relation node is annotated for propagating a deletion of an attribute to all queries accessing this attribute, whereas a specific query is annotated to block this change. As these policies may not always align towards the same goal, a general guideline for handling policy conflicts is proposed, which follows the rule: policies defined on

query graph structures are stronger than policies defined on view graph structures which in turn prevail on policies defined on relation graph structures. According to the prevailing policy the proper action is taken.
The cases of additions and deletions of other graph constructs are handled in a similar fashion. Moreover, to alleviate the designer from the burden of manually annotating all graph constructs, a simple extension of SQL with clauses concerning the evolution of important constructs is also proposed.

3 References

[ArGr06] Architecture Graphs for Databases. Project description available at http://www.cs.uoi.gr/~pvassil/projects/architecture_graph

[BHP00] Bernstein P.A., Halevy, A.Y., Pottinger, R.: A Vision of Management of Complex Models. SIGMOD Record 29(4): 55-63, 2000.

[FeNe02] N.E. Fenton, M. Neil: Software metrics: roadmap. ICSE - Future of SE Track 2000: 357-370.

[Fent94] N. Fenton. Software Measurement: A Necessary Scientific Basis. In IEEE Trans. on Software Engineering, 20(3), March 1994.

[GHJV95] E. Gamma, R. Helm, R. Johnson and J. Vlissides. "*Design Patterns: Elements of Reusable Objects- Oriented Software*". Professional Comptuting Series. Addison-Wedley, Reading, Ma, 1995.

[NiLR98] A. Nica, A. J. Lee, E. A. Rundensteiner. The CSV algorithm for view synchronization in evolvable large-scale information systems. In Proc. of International Conference on Extending Database Technology (EDBT '98). Lectures notes in computer science, Springer, p.359-373. Valencia, Spain, Mar 1998.

[PaVV06] G. Papastefanatos, P. Vassiliadis, Y. Vassiliou. Adaptive Query Formulation to Handle Database Evolution. Forum of the 18th Conference on Advanced Information Systems Engineering (CAiSE 2006), Luxembourg June 5-9, 2006 (short paper). Long version available at http://www.dblab.ece.ntua.gr/~gpapas/Publications/AdaptiveQueryEvolution-Extended.pdf

[PKVV05] G. Papastefanatos, K. Kyzirakos, P. Vassiliadis, Y. Vassiliou. Hecataeus: A Framework for Representing SQL Constructs as Graphs. In 10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'2005)

[Rod+00] J.F. Roddick et al. Evolution and Change in Data Management - Issues and Directions. SIGMOD Record 29(1), p.21-25 (2000)

[Sjøb93] Sjøberg, D.: "Quantifying Schema Evolution". Information and Software Technology, Vol. 35, No. 1, pp. 35-44, 1993.

[VSTS05] P. Vassiliadis, A. Simitsis, M. Terrovitis, S. Skiadopoulos. Blueprints for ETL workflows. In Proc. 24th International Conference on Conceptual Modeling (ER 2005), pp. 385-400, 24-28 October 2005, Klagenfurt, Austria

[VeMP04] Y. Velegrakis, R.J. Miller, L. Popa. Preserving mapping consistency under schema changes. VLDB J. 13(3), pp. 274-293, 2004.