

Adaptive Query Formulation to Handle Database Evolution

George Papastefanatos¹, Panos Vassiliadis², Yannis Vassiliou¹

¹ National Technical University of Athens,
Dept. of Electrical and Computer Eng.,
Athens, Hellas
{gpapas, yv}@dbnet.ece.ntua.gr

² University of Ioannina,
Dept. of Computer Science,
Ioannina, Hellas
pvassil@cs.uoi.gr

Abstract. Databases are continuously evolving environments, where design constructs are added, removed or updated rather often. Research has extensively dealt with the problem of database evolution. Nevertheless, problems arise with existing queries and applications, mainly due to the fact that, in most cases, their role as integral parts of the environment is not given the proper attention. Furthermore, the queries are not designed to handle database evolution. In this paper, we introduce a graph-based model that uniformly captures relations, views, constraints and queries. For several cases of database evolution we present rules so that both syntactical and semantic correctness of queries are retained.

1. Introduction

In typical organizational Information Systems, the designer/administrator is frequently faced with the necessity to predict the impact of a small change in the overall configuration. For instance, assume that an attribute has to be deleted from the underlying database. A small change like this might impact a large number of applications and data stores around the system: queries and data entry forms can be invalidated, application programs might crash (resulting in the overall failure of more complex workflows), and several pages in the corporate Web server may become invisible (i.e., they cannot be generated any more). Syntactic as well as semantic adaptation of queries and views to changes occurring in the database schema is a time-consuming task, treated in most of the cases manually by the administrators.

To deal with the aforementioned issues, *our approach is to provide a mechanism for performing what-if analysis for potential changes of database configurations.* A graph model that uniformly models queries, views, relations and their significant properties (e.g., conditions) is introduced. Apart from the simple task of capturing the semantics of a database system, the graph model allows us to predict the impact of a change over the system. Furthermore, we provide a framework for annotating the database graph with policies concerning the behavior in the presence of hypothetical changes occurring in the database schema. Rules that dictate the proper actions, when *additions* or *deletions* are performed to *relations*, *attributes* and *conditions* (all treated as first-class citizens of the model) are provided. Specifically, assuming that a graph construct is annotated with a policy for a particular event (e.g., a relation node is tuned to deny deletions of its attributes), the proposed framework (a) performs the identification of the affected part of the graph and, (b) if the policy is appropriate,

automates the readjustment of the graph to fit the new semantics imposed by the change.

This paper is organized as follows. In Section 2, the framework for adapting queries and views to schema evolution changes is sketched and in Section 3 we conclude our results and provide insights for future work. Due to strict space limitations, we refer the reader to the long version of this paper [PaVV05] for an extensive discussion of the issues raised in this paper.

2. Adapting queries and views to database evolution

The main mechanism towards handling schema evolution is the annotation of the constructs of the database graph (i.e., nodes and edges) with operators that handle schema evolution [PaVV05]. 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 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 an 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 case 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 remarks, 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 A1, ..., An` 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, as it happens now in database systems.

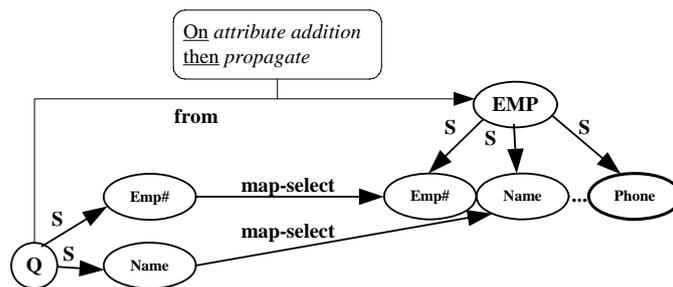


Fig. 1: Propagating addition of attribute `PHONE` to the schema of the query

The graph of the query `SELECT * FROM EMP` is shown in Figure 1. The annotation of the `FROM` edge as *propagating addition* indicates that the addition of `PHONE` node 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.

To alleviate the designer from the burden of manually annotating all graph constructs, a graph representation tool [PKVV05] and a simple extension of SQL with clauses concerning the evolution of important constructs is proposed.

3. Conclusions

In this paper, an automated mechanism that allows a designer to execute what-if analysis scenarios and determine the impact of a potential change over a database graph is proposed. The framework allows the insertion and deletion of relations, attributes and query conditions and equips the designer with the possibility of annotating the graph with policies that either accept, or block such potential events. The impact and the possible reshaping of the graph are automatically determined in the proposed framework, based on a set of rules provided by the administrator.

Research can be pursued in several directions. For example, transactions of events can be thought of as combinations of modifications to the database structure (e.g., combined modifications of primary and foreign keys). Also, visualization techniques can be discussed to further automate the evolution of the database.

References

- [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.
- [PaVV05] G. Papastefanatos, P. Vassiliadis, Y. Vassiliou. Adaptive Query Formulation To Handle Database Evolution (Extended Version). Working Draft November 2005, www.dbnet.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 Proceedings of 10th International Workshop on Exploring Modeling Methods for Systems Analysis and Design - EMMSAD '05 (in conjunction with CAISE'05)
- [VeMP04] Y. Velegrakis, R.J. Miller, L. Popa. Preserving mapping consistency under schema changes. VLDB J. 13(3), pp. 274-293, 2004.