Hecataeus: A What-If Analysis Tool for Database Schema Evolution

George Papastefanatos* Fotini Anagnostou Yannis Vassiliou
National Technical University of Athens,
Dept. of Electr. and Comp. Eng.,
Athens, Greece
{ gpapas, fanag, yv}@dblab.ece.ntua.gr

Panos Vassiliadis
Univ. of Ioannina,
Dept. of Computer Science,
Ioannina, Greece
pvassil@cs.uoi.gr

Abstract

Databases are continuously evolving environments, where design constructs are added, removed or updated rather often. Small changes in the database configurations 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. HECATAEUS *is a tool, which represents the database schema along with its dependent workload, mainly queries and views, as a uniform directed graph. The tool enables the user to create hypothetical evolution events and examine their impact over the overall graph as well as to define rules so that both syntactical and semantic correctness of the affected workload is 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 or a more sophisticated reorganization in the overall database configuration. For instance, assume that an attribute has to be deleted from the underlying database schema. 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 workload – mainly 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 or the application developers. 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, queries and views are not designed to handle database evolution.

In this paper, a tool, named HECATAEUS, which allows the administrator/designer to execute what-if analysis scenarios and determine the impact of a potential change over a database graph, is presented. The tool allows the definition of hypothetical evolution events over a database graph 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.

This paper is organized as follows: In Section 2, the framework for adapting queries and views to database schema evolution changes is sketched. We present the architecture and main features of Hecataeus tool in section 3 and in section 4 we present the evaluation of the tool over a real-world application and provide some insights for future work.

2. A framework for handling database schema evolution

Our approach is to provide a mechanism for performing what-if analysis for potential changes of database configuration [1]. A graph model that uniformly models queries, views, relations and their significant properties (e.g., attributes, 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 its behavior in the presence of hypothetical changes occurring in the database schema. Rules that dictate the proper actions, when *additions*, *deletions or*

978-1-4244-2157-2/08/$25.00 © 2008 IEEE

modifications 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, proposes the readjustment of the graph to fit to the new semantics imposed by the change.

Example. Consider the simple example query `SELECT * FROM EMP`. Assume that provider relation `EMP` is extended with a new attribute `PHONE`. There are two possibilities:

- The `*` notation signifies the request for any attribute present in the schema of relation `EMP`. In this case, the `*` shortcut can be treated as "return all the attributes that `EMP` has, independently of which these attributes are". Then, the query must also retrieve the new attribute `PHONE`.
- The `*` notation acts as a macro for the particular attributes that the relation `EMP` originally had. In this case, the addition to relation `EMP` should not be further propagated to the query.

A naive solution to such modifications; e.g., addition of an attribute, would be that an impact prediction system must trace all queries and views that are potentially affected and ask the designer to decide upon which of them must be modified to incorporate the extra attribute. We can do better by extending the current modeling. For each element potentially affected by the addition, we annotate its respective graph construct (i.e., node, edges) with policies. According to the policy defined on each construct the respective action is taken to correct the query.

Therefore, for the example event of an attribute addition, the policies defined on the query and the actions taken according to each policy are:

- *Propagate attribute addition*. When an attribute is added to a relation appearing in the `FROM` clause of the query, this addition should be reflected to the `SELECT` clause of the query.
- *Block attribute addition*. The query is immune to the change: an addition to the relation is ignored. In our example, the second case is assumed, i.e., 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 or the administrator must handle the impact of the change manually; similarly to the way that currently happens in database systems.

3. HECATAEUS: Architecture and features

In the context of the aforementioned framework, we have prototypically implemented a tool, HECATAEUS, for graphical representation and what-if analysis of several evolution events over a database schema. A first version of the tool [2] enabled the user to represent SQL queries as directed graphs, while current version is enriched with evolution semantics and features for performing what-if analysis for evolution of database schemas.

The tool accepts as input files DDL definitions and workload files (expressed in SQL). Regarding graph representation, the tool creates and visualizes a graph that holds all the semantics of the nodes and edges of the aforementioned graph model. Moreover, the tool assists the user in several ways: apart from the zoom-in/zoom-out capability, HECATAEUS offers the user the ability to isolate and highlight a part of a graph, in order to facilitate the accurate and complete understanding of the whole system as well as to modify the graph by adding or removing graph constructs. Regarding evolution functionality, users can define evolution semantics on the graph constructs, such as potential events and policies for the reaction of affected graph constructs. Users can perform evolution scenarios on the graph by assigning hypothetical events on specific parts of the graph. The tool highlights the affected parts and proposes a suitable transformation according to the policies defined on the graph for the occurring events.

The tool architecture consists of the coordination of Hecataeus' four main components: the *Parser*, the *Evolution Manager*, the *Graph Viewer* and the *Catalog*.

Parser is responsible for parsing the input files (i.e., DDL and workload definitions) and for sending each command to the database Catalog and then to the Evolution Manager.

The functionality of the *Catalog* is to maintain the schema of the relations as well as to validate the syntax of the workload parsed, before the Evolution Manager models them.

Evolution Manager component is responsible for representing the underlying database schema and the parsed queries in the proposed graph model. The Evolution Manager holds all the semantics of nodes and edges of the aforementioned graph model, assigning nodes and edges to their respective classes. It communicates with the catalog and the parser and constructs the node and edge objects for each class of nodes and edges (i.e., relation nodes, query nodes, etc.). It retains all evolution semantics for each graph construct (i.e., events, policies) and methods for performing evolution scenarios. It contains methods for

transforming the database graph from/to an XML format.

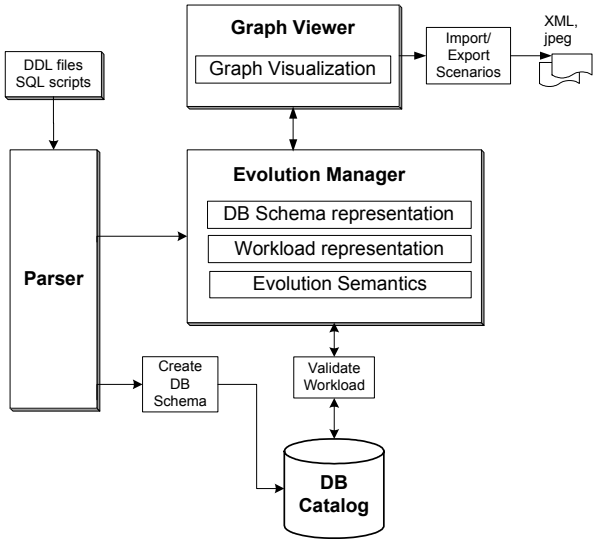

Figure 1: Hecataeus architecture

Lastly, *Graph Viewer* is responsible for the visualization of the graph and the interaction with the user. It communicates with the Evolution Manager, which holds all evolution semantics and methods. Graph Viewer offers distinct colorization for each set of nodes, edges according to their types and the way they are affected by evolution events, editing of the graph, such as addition, deletion and modification of nodes, edges and policies. It enables the user to raise evolution events, to detect affected nodes by each event and highlight appropriate transformations of the graph. Lastly, the user can import or export evolution scenarios to XML format and save scenarios to image formats (i.e., jpeg).

In Figure 1, we present the overall architecture of the proposed tool.

Hecataeus is implemented in Java. For the parser and the database engine, we have used HSQLDB, an open source SQL relational database engine written in Java [3], whereas for the graph visualization we have used the Java Universal Network/Graph Framework (JUNG), a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network [4].

4. Evaluation

We have evaluated the effectiveness of HECATAEUS tool via the reverse engineering of real-world evolution scenarios, extracted from an application of the Greek public sector. We extracted queries and views from within applications and stored procedures, monitored the changes that took place to the underlying relations over a period of 6 months and studied their impact to affected workload. In Figure2, a part of the examined system, comprising 2 relations, 1 view and 4 queries is visualized.

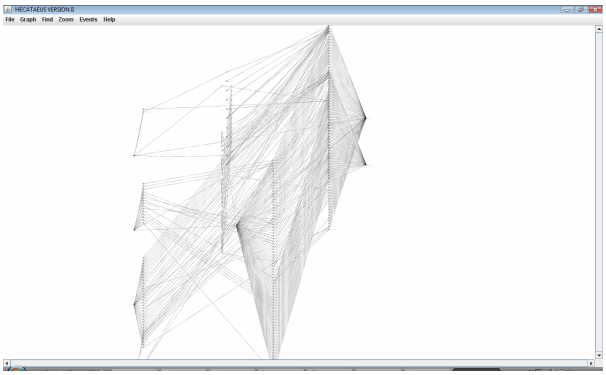

Figure 2: Graph Visualization with Hecataeus

HECATAEUS is an ongoing project and further development can be pursued in several directions. The integration of a simple extension of SQL with clauses concerning the evolution of important constructs is one of our most prominent goals.

5. References

[1] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou. "What-if Analysis for Data Warehouse Evolution." In 9th International Conference on Data Warehousing and Knowledge Discovery (DaWaK '07), Regensburg, Germany, 3-7 September, 2007, LNCS 4654, pp. 23–33, 2007.

[2] 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 for Systems Analysis and Design - EMMSAD '05 (in conjunction with CAISE'05) - Oporto, Portugal, June 2005.

[3] HSQL Database Engine: http://hsqldb.org

[4] Java Universal Network/Graph Framework (JUNG): http://jung.sourceforge.net/index.html