Materialized Hypertext View Maintenance

Giuseppe Sindoni
ISTAT - National Institute of Statistics, Italy

INTRODUCTION

A hypertext view is a hypertext containing data from an underlying database. The *materialization* of such hypertexts, that is, the actual storage of their pages in the site server, is often a valid option[1]. Suitable auxiliary data structures and algorithms must be designed to guarantee consistency between the structures and contents of each heterogeneous component where base data is stored and those of the derived hypertext view.

This topic covers the maintenance features required by the derived hypertext to enforce consistency between page content and database status (Sindoni, 1998). Specifically, the general problem of maintaining hypertexts after changes in the base data and how to incrementally and automatically maintain the hypertext view are discussed and a solution using a Definition Language for Web page generation and an algorithm and auxilary data structure for automatic and incremental hypertext view maintenance is presented.

BACKGROUND

Some additional maintenance features are required by a materialized hypertext to enforce consistency between page contents and the current database status. In fact, every time a transaction is issued on the database, its updates must be efficiently and effectively extended to the derived hypertext. In particular, *(i)* updates must be *incremental*, that is, only the hypertext pages dependent on database changes must be updated and *(ii)* all database updates must propagate to the hypertext.

The principle of incremental maintenance has been previously explored by several authors in the context of materialized database views (Blakeley et al., 1986; Gupta et al., 2001; Paraboschi et al., 2003; Vista, 1998; Zhuge et al., 1995). Paraboschi et al. (2003) give a useful overview of the materialized view maintenance problem in the context of multidimensional databases. Blakeley et al. (1986) propose a method in which all database updates are first filtered to remove those that cannot possibly affect the view. For the remaining updates, they apply a differential algorithm to re-evaluate the view expression. This exploits the knowledge provided by both the view definition expression and the database update operations. Gupta et al. (2001) consider a variant of the view maintenance problem: to keep a materialized view up-to-date when the view definition itself changes. They try to "adapt" the view in response to changes in the view definition. Vista (1998) reports on the integration of view maintenance policies into a database query optimizer. She presents the design, implementation and use of a query optimizer responsible for the generation of both maintenance expressions to be used for view maintenance and execution plans. Zhuge et al. (1995) show that decoupling of the base data (at the sources) from the view definition and view maintenance machinery (at the warehouse) can lead the warehouse to compute incorrect views. They introduce an algorithm that eliminates the anomalies.

Fernandez et al. (2000), Sindoni (1998) and Labrinidis & Roussopoulos (2000) have brought these principles to the Web hypertext field. Fernandez et al. (2000) provide a declarative query language for hypertext view specification and a template language for specification of its HTML representation. Sindoni (1998) deals with the maintenance issues required by a derived hypertext to enforce consistency between page content and database state. Hypertext views are defined as nested oid-based views over the set of base relations. A specific logical model is used to describe the structure of the hypertext and a nested relational algebra extended with an oid invention operator is proposed, which allows views and view updates to be defined. Labrinidis & Roussopoulos (2000) analytically and quantitatively compare three materialization policies (inside the DBMS, at the web server and virtual). Their results indicate that materialization at the Web server is a more scalable solution and can facilitate an order of magnitude more users than the other two policies, even under high update workloads.

The orthogonal problem of deferring maintenance operations, thus allowing the definition of different policies, has been studied by Bunker et al. (2001), who provide an overview of the view maintenance subsystem of a commercial data warehouse system. They describe optimizations and discuss how the system's focus on star schemas and data warehousing influences the maintenance subsystem.

MAIN THRUST

With respect to incremental view maintenance, the heterogeneity caused by the semistructured nature of views, different data models and formats between base data and derived views makes the materialized hypertext view different to the database view context and introduces some new issues.

Hypertexts are normally modeled by object-like models, because of their nested and network-like structure. Thus with respect to relational materialized views, where base tables and views are modeled using the same logical model, maintaining a hypertext view derived from relational base tables involves the additional challenge of taking into account for the materialized views a different data model to that of base tables.

In addition each page is physically stored as a marked-up text file, possibly on a remote server. Direct access to single values on the page is thus not permitted. Whenever a page needs to be updated, it must therefore be completely regenerated from the new database status. Furthermore, consistency between pages must be preserved, which is an operation analogous to the one of preserving consistency between nested objects.

The problem of dynamically maintaining consistency between base data and derived hypertext is the *hypertext view maintenance problem.* It has been addressed in the framework of the STRUDEL project (Fernandez et al., 2000) as the problem of *incremental view updates for semistructured data*, by Sindoni (1998) and by Labrinidis & Roussopoulos (2000).

There are a number of related issues:

- different maintenance policies should be allowed (*immediate* or *deferred*);
- this implies the design of auxiliary data structures to keep track of database updates, but their management overloads the system and they must therefore be as light as possible;
- finally, due to the particular network structure of a hypertext, consistency must be maintained not only between the single pages and the database, but also between page links.

To deal with such issues, a manipulation language for derived hypertexts; an auxiliary data structure for *(i)* representing the dependencies between the database and hypertext and *(ii)* logging database updates; and an algorithm for automatic hypertext incremental maintenance can be introduced. For example, hypertext views and view updates can be defined using a logical model and an algebra (Sindoni, 1998).

An auxiliary data structure allows information on the dependencies between database tables and hypertext pages to be maintained. It may be based on the concept of *view dependency graph*, for the maintenance of the hypertext class described by the logical model. A view dependency graph stores information about the base tables, which are used in the hypertext view definitions.

Finally, incremental page maintenance can be performed by a maintenance algorithm that takes as its input a set of changes on the database and produces a minimal set of update instructions for hypertext pages. The algorithm can be used whenever hypertext maintenance is required.

A Manipulation Language for Materialized Hypertexts

Once a derived hypertext has been designed with a logical model and an algebra has been used to define its materialization as a view on base tables, a manipulation language is of course needed to populate the site with *page scheme instances*[2] and maintain them when database tables are updated. The language is based on invocations of algebra expressions.

The languages used for page creation and maintenance may be very simple, such as that composed of only two instructions: GENERATE and REMOVE. They allow manipulation of hypertext pages and can refer to the whole hypertext, to all instances of a page scheme, or to pages that satisfy a condition.

The GENERATE statement has the following general syntax:

```
GENERATE    ALL | <PAGESCHEME>
[WHERE      <CONDITION>]
```

Its semantics essentially create the proper set of pages, taking data from the base tables as specified by the page scheme definitions. The ALL keyword allows generation of all instances of each page scheme. The REMOVE statement has a similar syntax and allows the specified sets of pages to be removed from the hypertext.

Incremental Maintenance of Materialized Hypertexts

Whenever new data are inserted in the database, the status of all affected tables changes and the hypertexts whose content derives from those tables no longer reflect the database's current status. These pages must therefore be updated in line with the changes. An extension to the system is thus needed to incrementally enforce consistency between database and hypertext.

The simple "brute force" approach to the problem would simply regenerate the whole hypertext from the new