Schema Versioning

Zouhaier Brahmia
University of Sfax, Tunisia

Fabio Grandi
University of Bologna, Italy

Barbara Oliboni
University of Verona, Italy

Rafik Bouaziz
University of Sfax, Tunisia

INTRODUCTION

Persistent information and data-centric systems use databases to store data. The contents of a database must adhere to a formal structure that is fixed in advance, and is called the schema of the database (Date, 2003). In those systems, not only data changes are obvious tasks done almost every day but also schema changes are unavoidable, in order to reflect a change in the real world or in the user's requirements, to correct mistakes in the initial design, to migrate to a new platform or to allow the expansion of the application scope over time. Two main problems have to be considered when dealing with any schema change: *semantics of change* (i.e., the effects of the change on the schema itself) and *change propagation* (i.e., the effects of the change on the underlying data). Resolving the former guarantees schema consistency, while resolving the latter guarantees consistency of data with respect to the changed schema.

In the literature, s*chema evolution* and *schema versioning* (Roddick, 1995; Jensen et al., 1998) are the two techniques that were proposed to support schema changes in a DBMS, without loss of extant data and with continued support of legacy applications. After applying schema changes, schema evolution keeps only the current schema version and retains the data which are adapted to such a schema. On the other hand, each time schema changes are applied, schema versioning creates a new schema version, while preserving old schema versions and their corresponding data. With schema versioning, data access through any schema version is supported, which avoids applications developed with past schemata to become obsolete.

Schema versioning has been widely investigated, both in the context of traditional and temporal database research. Several models, languages and approaches, dealing with schema versioning, have been proposed during the two last decades. However, to the best of our knowledge, limited support of schema changes and no support of schema versioning is provided by commercial database management systems (DBMS). Therefore, diligent database designers and administrators have to work hard to solve the problem of evolving a database schema in an ad hoc manner. Besides, with the growing interest in XML adopted as a data modeling language and storage format, research work has also recently done on the problems of schema versioning in XML databases.

The main goal of this article is (1) to present the different research proposals dealing with schema versioning, and (2) to discuss the support provided by available DBMSs to manage schema versioning. In particular, the next section gives some basic definitions related to the considered subject. In "Current Research in Schema Versioning," we present the different research proposals on schema versioning. "DBMS Support for Schema Versioning" surveys the support of schema versioning in existing DBMSs. Finally, future work directions and conclusion are provided.

DOI: 10.4018/978-1-4666-5888-2.ch754

BACKGROUND

The schema versioning technique allows changes to the database schema with continued support of previous schemata and their corresponding data, which are retained without any change. The newly created schema version is (usually) used to accommodate new data insertions, modifications and deletions. This technique neither leads to loss of information nor to obsolescence of existing applications, as they can still work with old schema versions.

In a schema versioning environment, two related issues are involved: (1) compatibility between schema versions and applications and (2) data access facilities.

As far as the first issue is concerned, we distinguish between backward compatibility and forward compatibility. Backward compatibility means that data defined under the previous schema version can be processed by an application designed for the new schema version, whereas forward compatibility means that data introduced under a new schema version can be processed by an application designed for a previous schema version.

As for data access facilities, we distinguish between transparent data access facilities and explicit data access facilities. Transparent data access facilities do not explicitly consider schema versions for accessing data: a "completed schema" compatible with all the different schema versions (i.e., similar to a "Global-As-View" integration schema) is used for accessing data defined under any schema version in a transparent way. On the contrary, explicit data access must refer to versions. Thus, existing database query languages (e.g., SQL) have to be extended by new schema selection features (e.g., based on schema version number, schema version creation time, schema version interval) to allow users to access data defined under the desired schema version(s).

Furthermore, the most used solutions for performing schema versioning are: *sequential revisions* and *parallel variants* (Munch, 1993). With the former, each new schema version is created by modifying the last schema version; we always have only one current schema version and a set of past schema versions. With the latter, a new schema version (or a variant) does not replace another, but becomes an alternative to the modified version instead; thus, multiple current schema versions could exist, which derive from a common previous schema version.

On the other hand, the storage of schema versions could be achieved by one of the two following solutions: either *versioning by difference* or *versioning by copy* (Loomis, 1995). In versioning by difference, the new schema version contains only the changed parts of the initial schema. In versioning by copy, the new schema version contains both the changed parts and the parts that have not changed of the initial schema.

Although it is the most powerful and flexible technique for managing schema changes, schema versioning is not supported by any commercial DBMS yet.

In order to illustrate the effects of schema versioning, let us assume that we have a relational database that contains only an AUTHOR relation with the attributes ID (primary key), NAME, PHONE, and COUNTRY. The first state of this database is as shown in Figure 1.

The catalogues store information on the schema S1 of the AUTHOR relation. The table AUTHOR contains two tuples for two authors. Then consider the following schema changes:

```
ALTER TABLE AUTHOR
DROP COLUMN PHONE;
ALTER TABLE AUTHOR
ADD COLUMN EMAIL CHAR(30);
```

Figure 1.

S1

AUTHOR (ID, NAME, PHONE, COUNTRY)

ID	NAME	PHONE	COUNTRY
1	Aicha	11223344	Tunisia
2	Cristiana	55667788	Italy