Data Model Versioning and Database Evolution

Hassina Bounif
Swiss Federal Institute of Technology, Switzerland

INTRODUCTION

In the field of computer science, we are currently facing a major problem designing models that evolve over time. This holds in particular for the case of databases: Their data models need to evolve, but their evolution is difficult. User requirements are now changing much faster than before for several reasons, among them the changing perception of the real world and the development of new technologies. Databases show little flexibility in terms of supporting changes in the organization of their schemas and data. Database evolution approaches maintain current populated data and software application functionalities when changing database schema. Data model versioning is one of these chosen approaches used to resolve the evolution of conventional and nonconventional databases such as temporal and spatial-temporal databases. This article provides some background on database evolution and versioning technique fields. It presents the unresolved issues as well.

BACKGROUND

A major problem in computer science is how to elaborate designs that can evolve over time in order to avoid having to undergo redesign from scratch. Database designs are a typical example where evolution is both necessary and difficult to perform. Changes to database schema have to be enabled while maintaining the currently stored data and the software applications using the data in the database. One of the existing approaches used to resolve difficulties in database evolution is data model versioning. These techniques are the main focus of this article.

To avoid misinterpretation, we first recall the definition of the meaning of the basic concepts: schema versioning, schema evolution, schema modification, and schema integration. These concepts are all linked to the database evolution domain at different levels, but according to most of the research focusing on database evolution (Roddick, 1995), there is a consistent distinction between them.

- **Schema Versioning** means either creating from the initial maintained database schema or deriving from the current maintained version different schemas called versions, which allow the querying of all data associated with all the versions as well as the initial schema according to user or application preferences. Partial schema versioning allows data updates through references to the current schema, whereas full schema versioning allows viewing and update of all present data through user-definable version interfaces (Roddick, 1995).
- **Schema Evolution** means modifying a schema within a populated database without loss of existing data. When carrying out an evolution operation, there are two problems that should be considered: on one hand, the semantics of change, i.e., their effects on the schema; and, on the other hand, change propagation, which means the propagation of schema changes to the underlying existing instances (Peters & Özsu, 1997). Schema evolution is considered a particular case of schema versioning in which only the last version is retained (Roddick, 1995).
- **Schema Modification** means making changes in the schema of a populated database: The old schema and its corresponding data are replaced by the new schema and its new data. This may lead to loss of information (Roddick, 1995).
- **Schema Integration** means combining or adding schemas of existing or proposed databases with or to a global unified schema that merges their structural and functional properties. Schema integration occurs in two contexts: (1) view integration for one database schema and (2) database integration in distributed database management, where a global schema represents a virtual view of all database schemas taken from distributed database environments. This environment could be homogenous, i.e., dealing with schemas of databases having the same data model and identical DBMSs (database management systems), or heterogeneous, which deals with a variety of data models and DBMSs. This second environment leads to what is called federated database schemas. Some research works consider schema integration a particular case of schema evolution in which the integration of two or more schemas takes place by choosing one and applying the facts in the others (Roddick, 1995).

This article is organized as follows. The next section reviews some background related to database evolution and focuses on the versioning data model. In the Future Trends section, we present some emerging trends. The final section gives the conclusion and presents some research directions.

MAIN THRUST

Versioning Methods and Data Conversion Mechanisms

Various mechanisms are used to manage versioning/ evolution of databases according to the data model, the nature of the field of the applications, as well as the priorities adopted (Munch, 1995).

- **Revisions (Sequential):** consists of making each new version a modification of the most recent one. At the end, the versions sequentially form a single linked list called a *revision chain* (Munch, 1995). Each version in the chain represents an evolution of the previous one. This method is used for versioning schemas. This technique is not very effective because many copies of the database are created as schema versions. It is adopted by the Orion System (Kim & Chou, 1988), in which complete database schemas are created and versioned.
- **Variants (Parallel):** means changing the relationships for revision from one-to-one to many-to-one, so that many versions may have the same relationship with a common "ancestor" (Munch, 1995). A variant does not replace another, as a revision does, but is instead an alternative to the current version. This versioning method is adapted differently from one database to another depending on the unit (a schema, a type, or a view) to be versioned. For instance, the Encore System (Skarra & Zdonik, 1987) uses type versioning while the Frandole2 system (Andany, Leonard, & Palisser, 1991) uses view versioning. Revisions and variants are usually combined into a common structure named the *version graph*.
- **Merging:** consists of combining two or more variants into one (Munch, 1995).
- **Change Sets:** In this method, instead of having each object individually versioned, the user collects a set of changes to any number of objects and registers them as one unit of change to the database. This collection is termed a change set, or cset.
- **Attribution:** Used to distinguish versions via values of certain attributes, for example, a date or a status.

Other authors (Awais, 2003; Connolly & Begg, 2002) further categorize versions. A version can be transient, working, or released.

- **Transient versions:** A transient version is considered unstable and can be updated or deleted. It can also be created from scratch by checking out a released version from a public database or by deriving it from a working or transient version in a private database. In the latter case, the base transient version is promoted to a working version. Transient versions are stored in the creator's private workspace (Connolly & Begg, 2002).
- **Working versions:** A working version is considered stable and cannot be updated, but it can be deleted by its creator. It is stored in the creator's private workspace (Connolly & Begg, 2002).
- **Released versions:** A released version is considered stable and cannot be updated or deleted. It is stored in a public database by checking in a working version from a private database (Connolly & Begg, 2002).

With respect to data, there are three main data conversion mechanisms that are employed with these versioning methods: (1) Data is simply coerced into the new format; (2) Data is converted by lazy mechanisms when required; in this mechanism, there is no need to consider unneeded data, and schema changes take place rapidly when change occurs; and (3) Conversion interfaces are used to access data that never undergo physical conversion (Roddick, 1995).

Versioning and Data Models

The unit to be versioned must have (1) a unique and immutable identity and (2) an internal structure. These two criteria have some consequences on the possibilities of versioning within different data models. Object-oriented or ER-based data models clearly fulfill these two requirements (Munch, 1995). The adoption of an object-oriented data model is the most common choice cited in the literature (Grandi & Mandreoli, 2003) on schema evolution through schema versioning. The relational model has also been studied extensively. In fact, some problems exist. Tuples, for instance, are simply values which fulfill the requirements for non-atomicity, but they have no useful ID. A user-defined key cannot be used since the