A Survey of Schema Versioning Issues for Database Systems

John F. Roddick

Advanced Computing Research Centre
School of Computer and Information Science
University of South Australia
The Levels, SA 5095, Australia

roddick@unisa.edu.au

Abstract

Schema versioning is one of a number of related areas dealing with the same general problem - that of using multiple heterogeneous schemata for various database related tasks. In particular, schema versioning, and its weaker companion, schema evolution, deal with the need to retain current data and software system functionality in the face of changing database structure. Schema versioning and schema evolution offer a solution to the problem by enabling intelligent handling of any temporal mismatch between data and data structure. This survey discusses the modelling, architectural and query language issues relating to the support of evolving schemata in database systems. An indication of the future directions of schema versioning research are also given.

Keywords

Schema Evolution, Schema Versioning, Evolving Database Systems

1. Introduction

1.1. Background

Database and software systems are rarely stable following initial implementation. Although estimates differ, most agree that 50% or more of programmer effort arises as a result of system modifications after implementation (Lientz 1983) and facilitating those changes is complicated if large numbers of programs or large quantities of data are involved. Moreover, system modifications that result in changes to database structure are relatively frequent (Sjøberg 1992, 1993). As a result, modifying the database schema is a common, but often troublesome, occurrence in database administration. These are significant industrial concerns, both from the viewpoint of database system manufacturers and information system users. *Schema evolution*, and its stronger companion, *schema versioning*, have arisen in response to a need to retain data entered under schema definitions that have

since been amended. A more formal definition of schema evolution is given later but a loose definition can be given as the ability for a database schema to evolve without the loss of existing information. Schema evolution and schema versioning can also be considered to contribute to the solution of a number of identified deficiencies in current database systems research, for example, in the database support for legacy systems (Stonebraker, *et al.* 1993) and non-stop, industrial strength databases (Selinger 1993).

Interest in evolving database systems has predominantly resulted from research in two areas. Firstly, as a logical extension from work with temporal data modelling and temporal databases, and secondly from within the object-oriented paradigm, specifically with a view to producing architectures suitable for CAD and other Engineering domains. A recent (and growing) bibliography of schema evolution research (Roddick 1994) lists over 50 papers in the area of which only a handful are dated prior to 1987. This research has been in connection with both relational (Clifford and Croker 1987; Dadam and Teuhola 1987; McKenzie and Snodgrass 1990; Ariav 1991; Roddick 1991, 1992b; Ventrone and Heiler 1991) and object-oriented databases (Banerjee, *et al.* 1986, 1987a, 1987b; Skarra and Zdonik 1986, 1987; Zdonik 1986; Penney and Stein 1987; Chou and Kim 1988; Kim and Chou 1988; Bretl, *et al.* 1989; Kim, *et al.* 1989; Nguyen and Rieu 1989a, 1989b; Osborn 1989; Lerner and Habermann 1990).

1.2. Pragmatic considerations

Before looking in detail at the issues, since the demand for evolving schemata is being driven by the user community, this section discusses briefly some of the constraints or practical considerations for any proposed solution to the problems.

First, it is recognised that schema modifications need to be, to a greater or lesser extent, guided by the database administrator. However, it is desirable that schema modification should require the minimum level of intervention appropriate to the change being performed. Details of the implementation (such as decisions between strict or lazy data conversion, for example) are dependent more on the operational environment rather than the user's requirements and thus should not be the direct concern of database administrators. Not withstanding this, it is unlikely, and probably undesirable, that the database design process will ever be totally automated.

Second, the schema modification should be as symmetric as possible so that not only can existing data be viewed through new schema definitions but also so that data recorded later can be viewed under previous schemata. This would also promote a stability of operation by not requiring existing applications to be recompiled. In addition, changes should be as reversible as possible so that erroneous changes can be removed. This implies, for example, that data definition functions should operate losslessly.

Third, as far as possible, the modifications should be expressed in terms of definable algebraic operations on the schema and should allow for formal verification of the schema change within the system as a whole. This would provide for a known and complete mechanism for schema manipulation. Furthermore, rather than supplying a large numbers of schema change operators, changes on a larger scale might be made available through a composition of more elementary operations.

Finally, while many models require some level of transaction-time support for schema definitions (in order to identify the versions) it should not be a requirement to support time in the base architecture (although see the parallels discussed in §2.3). The advantages of doing so have been examined in depth elsewhere (see for example (Snodgrass and Ahn 1986; Roddick and Patrick 1992; Tansel, *et al.* 1993)) but it would defeat the purpose to enforce this as a prerequisite.

1.3. Outline of this paper

This paper surveys the field of schema versioning in database systems and is organised as follows. Section 2 provides a number of definitions including refinements to those presented by Jensen, *et al.* (1994). The associated areas of data and view integration and temporal data modelling are also discussed. Sections 3 to 6 discuss various issues in schema versioning outlining current proposals and prototypes. Section 3 looks at data modelling issues, section 4 looks at architectural issues while section 5 looks at query language considerations. Section 6 discusses a few issues which do not fit easily under the other headings. Finally section 7 draws these issues together and presents a number of areas for future research.

2. Handling heterogeneous schemata

This section gives some definitions and discusses the wider field of accommodating heterogeneous schemata in database systems. For completeness, this section also introduces the concepts of data and view integration and temporal data modelling and discusses their relationship with schema evolution.

2.1. Schema modification, evolution and versioning

The distinction between modification, evolution and versioning of database schemata has been, in some cases, confused and, where distinction has been made, there have been various usages of the terms. The following definitions, however, are implicitly consistent with the majority of the research dealing with schema evolution and versioning issues (Beech and Mahbod 1988; Narayanaswamy and Bapa Rao 1988; Bjornerstedt and Hulten 1989; Osborn 1989; Andany, Leonard and Palisser 1991; Ariav

1991; Monk and Sommerville 1992, 1993) and were included by the author in a recent glossary of temporal database concepts (Jensen, *et al.* 1994).

Definition - *Schema Modification*

> Schema Modification is accommodated when a database system allows changes to the schema definition of a populated database. **n**

Definition - *Schema Evolution*

> Schema Evolution is accommodated when a database system facilitates the modification of the database schema without loss of existing data. **n**

Definition - *Schema Versioning*

> Schema Versioning is accommodated when a database system allows the accessing of all data, both retrospectively and prospectively, through user definable version interfaces. **n**

A number of points should be made about these definitions:

i. In its simplest sense, schema evolution does not imply full historical support for the schema; only the ability to change the schema definition without loss of data. In practice, retention of past definitions will often be appropriate. In contrast, schema versioning, even in its simplest form, requires that a history of changes be maintained to enable the retention of past schema definitions.

ii. The significant difference between evolution and versioning is the ability for users to identify quiescent or stable points in the definition and label the definition in force at that time for later reference. Schema evolution does not require the ability to version data except in so far as each changed schema can be considered a new version. Nor does it require that the database system provide a viewing mechanism using past schema definitions.

iii. Schema changes will not necessarily result in a new version. Typically schema changes will be of a finer grain than the definable versions.

iv. Versions will tend to be labelled either by the (transaction) time of the schema change or by some user-defined method whereas schema evolution changes are referred to more often only by the time of change.

As will be discussed in this paper, the accommodation of schema versioning presents various open problems relating to the update of data through historical schemata. The definition of schema versioning is therefore refined further by distinguishing between retrieval and update activity as follows:

Definition *- Partial Schema Versioning*

> Partial Schema Versioning is accommodated when a database system allows the viewing of all data, both retrospectively and prospectively, through user definable version interfaces. Data updates are allowable through reference to one designated (normally the current) schema definition only. **n**

Definition *- Full Schema Versioning*

> Full Schema Versioning is accommodated when a database system allows the viewing and update of all data, both retrospectively and prospectively, through user definable version interfaces. **n**

It should be noted that the term "evolutionary" is also used in relation to systems using a genetic or natural systems approach. An example is presented by van Bommel who presents a methodology which enables the development of structurally optimised data models (van Bommel 1993). The approach searches the solution space of possible internal representations of a conceptual model by random mutation. Although using an evolutionary approach, the accommodation of schema evolution of populated databases is not discussed.

2.2. Data and view integration

A closely associated research area is that of data and view integration which aims to facilitate the merging of schemata for update or viewing purposes. Research in this area has largely been directed at integrating heterogeneous systems but the applicability to the evolution of schemata is clear. Miller, Ioannidis and Ramakrishnan (1993) for example, investigate the concept of the *information capacity* of a schema to decide whether data or view integration would be lossless. A taxonomy is given in which the translation ability for two schemata is distinguished by their ability to retain information during update and retrieval. This aspect is also investigated by Orlowska and Ewald (1991, 1992; Ewald and Orlowska 1994) who view schema integration as a schema evolution process in which the integration of two or more schemata is effected by choosing one and applying the facts held in the others. Geller, *et al.* (1992) present a method which allows the integration of structurally similar but semantically dissimilar datasets by using a "dual" model which maintains separate representations for structure and semantics. Other work in this area includes Atzeni, *et al.* (1982) and Larson, Navathe and Elmasri (1989). To the knowledge of the author, schema and view integration of temporal database systems has not been investigated (see §2.3).

The four terms, schema evolution, data integration, view integration and schema versioning may be considered as flavours of the same general problem - that of using multiple heterogeneous schemata for various database related tasks. A taxonomy of research emphases for the various areas may be

promoted by examining the temporal relationships between the data format and the schemata for the tasks required which, for simplicity in Table 1 below, are data retrieval, data updating and effecting structural alterations. These are tabulated against the schema used for the retrieval etc. and the format through which the data was stored. In all cases a populated, tractable, online database is assumed. The primary schema in the case of schema evolution and schema versioning corresponds to the current schema while in the case of data and view integration it corresponds to the integrated schema. Similarly, the secondary schema corresponds to either the historical or local schemata or user views as appropriate. The pathological cases where the user-view equals the integrated view and when the local schema is the same as the integrated schema are shown parenthetically.

Schema used for function	Data held in the format corresponding to…	Data Retrieval	Data Update	Structural Alterations
Primary	Primary schema	SE, SV, DI, (VI)	SE, SV, DI, (VI)	SE, SV
	Secondary schema	SE, SV, DI	SE, SV, DI	
Secondary	Primary schema	SV, VI	FSV, VI	?
	Secondary schema	SV, VI, (DI)	FSV, VI, (DI)	

Key
SE = Schema Evolution,
SV = Schema Versioning (Partial or Full),
FSV = Full Schema Versioning,
DI = Data Integration,
VI = View Integration
? = Little published research

Table 1. Connection between areas of interest aims of schema handling research

Although very similar to data and view integration and although many aspects of this work are common, the accommodation of historical data and the ability to retain, and in some cases to influence, the evolutionary history of schemata enable and necessitate a different treatment of the problem.

2.3. Temporal database systems

As many of the concepts used in schema versioning are similar to those associated with temporal database systems, for completeness, a brief overview of temporal database systems is now given. Readers are, however, directed to other sources for a far more complete treatments of this area (Clifford and Ariav 1986; Snodgrass and Ahn 1986; Dean and McDermott 1987; Roddick and Patrick

1992; Tansel, *et al.* 1993). Only those aspects of temporal databases relevant to this survey are discussed here.

Temporal data models are concerned with the accommodation of the inherent temporal nature of the object world and the time-dependent recording of facts from a representation of this object world in a database system. Temporal data models are thus concerned with two orthogonal time dimensions, "real-world time", referred to in the temporal database literature as *valid-time*, and "database system time", referred to as *transaction-time*. Temporal database systems accommodating these are known as valid-time databases, transaction-time databases or *bitemporal* databases depending on their capacity to handle either of both of the two temporal dimensions. The proposed query languages defined to handle bitemporal data are augmented to allow a user to specify either or both of valid-time and transaction-time expressions for data retrieval or valid-time expressions for data update (transaction-time is a function of the time of update and cannot therefore be specified by the user). Furthermore, the time dimensions may be referenced in two ways; either by reference to a time continuum, known as *absolute* time referencing, or by reference to other events, commonly termed *relative* or *indirect* time referencing. Finally, the term *temporal density* refers to the manner by which data values are inferred at time points not explicitly recorded in the database. Various mechanisms have been proposed, including stepwise change, discrete event and continuous change. For the purposes of schema evolution, the stepwise change of database objects is usually assumed.

In common with schema versioning and schema evolution, temporal extensions have been defined for a number of database paradigms including relational, extended relational, object-oriented and deductive.

3. Data Modelling Issues

Miller *et al.* (1993) have shown that in order to update data stored under two different schemata using the opposite schemata, they must be equivalent, ie. all valid instances of some schema S_1 must be able to be stored under S_2 and vice-versa. Since it is not possible (generally) to foresee the future requirements of the database, and hence the changes required to the data structure, and since neither the active nor the historical schemata can be changed (in general it is only possible to create a new version which supersedes the old), this is too strong a condition to impose in many cases. Most papers dealing with the issue of modelling evolution in databases therefore adopt the weaker concept of *partial schema versioning* in which data stored under any historical schema may be viewed through any other schema but may only be updated through the current or active schema. Even this level of support often necessitates imposing some form of restriction on the schema modifications which can be achieved without database reorganisation and data coercion. This section discusses various data

modelling issues associated with evolving schemata. As would be expected, a number of the issues raised here are revisited in Section 5 when the characteristics of query language design are discussed.

3.1. Domain/type evolution

It is significant that while the evolution of a domain is one of the simplest modifications that can be made to a data model, it still represents a non-trivial amendment to the database schema[1]. Consider the trivial (but common) example suggested by Roddick (1992a) in which a salary relation holds the following fields:

Staff Id	Position Code	Salary
1	G55	$33,000
2	G56	$37,000
3	A05	$45,500
4	A09	$65,400
5	G51	$32,000

Figure 2. Example salary relation from Roddick (1992a)

Suppose that the existing position codes are to be replaced with new codes based on new incompatible domains, for example, a position code based entirely on a domain of four-digit integers. The database administrator is faced with the problems arising from the retention of the current data such as:

i. Is the position code attribute to be defined as (the hybrid) alphanumeric despite the new position codes being purely numeric?

ii. Is another attribute required to store the old codes, if so, for how long is this attribute retained? How is this old field related to the new one by the applications?

iii. What about position histories and retired employees for whom no new format position code may be allocated?

As another example, consider the reduction in size of the domain of a primary key which, in the worst case, may result in the illegal introduction of duplicate values.

Ventrone and Heiler (1991) discuss data interpretation problems caused by a change to a domain. They present a number of examples of changes to the semantics of a domain which may result in lost or misleading information. These examples can be grouped into two sets; those that are Object System generated, such as cardinality or granularity changes, and those that are Database Administration

---

1 The schema is considered to be the repository for knowledge about the (evolving) structure of the database. Under some paradigms this may be hard to isolate, nevertheless the abstract idea of an *all knowing* schema is used here to represent its equivalent in various paradigms.

generated, such as field recycling or data encoding changes. Their suggestions towards a proposed solution emphasise the importance of capturing the semantics of a domain and the identification of that semantic content within the metadata thus replacing the *problems of semantic heterogeneity by more tractable problems of syntactic heterogeneity* (Ventrone and Heiler 1991). There is a strong suggestion that the semantic capabilities of data dictionaries (ie. schemata) should be enhanced and tied more closely with the interpretation of values in the database.

The domain evolution problem is strongly associated with the expressiveness and the structure of the type system adopted by the database system. The SQL approach of using *character strings*, *exact numeric types* and *approximate numeric types* requires more database administrator intervention than one in which a C-like cast is available (such as in SQL-92) or when weak typing is adopted. The TSQL2 language design proposal for example, utilises the SQL-92 casting mechanism with some rules for data conversion and a fall-back position of the null or some other value when data are not convertible[2].

Domain evolution is a good example of the distinction between schema evolution and schema versioning. Under schema evolution, existing instances must be converted to the new format (the mechanisms for achieving this are discussed in Section 4.2) and thus existing applications are rendered incompatible. Versioning promotes program compatibility by leaving the existing definition in place (Zdonik 1990; Clamen 1992).

3.2. Relation/class evolution

Relation and class evolution include attribute and relation/class definition, redefinition and deletion and class lattice modification. Within the temporal database paradigm this may also include attribute and relation/class deactivation and reactivation. Suggestions in the literature indicate that modification of the database schema to accommodate changes at the relation or class level (and above) can be achieved in a number of ways. For instance, within the object-oriented paradigm a common method is to establish a set of invariants to ensure the semantic integrity of the schema and a set of rules or primitives for effecting the schema changes (Banerjee, *et al.* 1986; Bretl, *et al.* 1989; Lerner and Habermann 1990) while within the relational model a set of atomic operations is proposed which result in a consistent and, as far as possible, reversible database structure (Shneiderman and Thomas 1982).

It is important that the evolutionary history of the database's relational or class structure be traceable. The DDL statements must therefore be both logically complete, so that users need not resort

---

2 This exercise, known as the TSQL2 language design proposal, aimed to define a temporal extension to the SQL92 standard. Relevant papers relating to this effort include (Hsu, Jensen and Snodgrass 1992a, 1992b; Snodgrass 1992; Roddick and Snodgrass 1993, 1994; Snodgrass, *et al.* 1994) .

to DML support (to directly modify data), and clear, both in their specification and their action. In earlier work (Roddick 1993; Roddick, Craske and Richards 1993) a taxonomy for schema evolution is proposed which uses a transaction-time meta-relation approach; ie. previous schemata may be constructed through temporal rollback of the meta-relations.

Many schema change requirements will involve composite operations and thus a mechanism for schema level (DDL) commit and rollback functions are suggested which should be separate from the data level (DML) commit and rollback operations. In addition, the data level commit operations might function differently when schema level transactions are active. For example, since data updated to a revised schema may be inapplicable if the schema change is not itself committed, it may not be considered appropriate to allow data to be committed outside of the scope of the current session until the schema-level commit is issued. This allows for the definition *and* population of attributes to be completed as one molecular operation. As an example, the following sequence of database operations may be issued to test a program:

```
Add attribute(s) to relation;
Populate attribute(s);
Data level commit;
Run test programs;
If tests successful
   Schema-level-commit;
else
   Schema-level-rollback;
```

It should be noted that the issue of long-lived and nested transactions is the subject of much current research and any proposals in this area should not be isolated from this research. See also the discussions in Section 4.4 on the use of multiple concurrent versions.

3.3. Algebras supporting schema evolution

The relational model (Codd 1970) and its extension (Codd 1979) provide for a logically complete language with which to describe data transformations (Codd 1972) and a number of time-related algebras have been proposed to extend the static relational model (Clifford and Tansel 1985; Tansel 1986; Clifford and Croker 1987; McKenzie and Snodgrass 1987a, 1987c, 1990; Gadia 1988; Lorentzos and Johnson 1988; Sarda 1990; Tuzhilin and Clifford 1990). A survey of valid-time and bitemporal algebras is given in McKenzie and Snodgrass (1991) which includes an evaluation of twelve representative algebras against various criteria. Of these, only the proposals by McKenzie and Snodgrass (McKenzie and Snodgrass 1987b, 1990; McKenzie 1988) are extended to deal explicitly with schema evolution[3].

---

3 The nomenclature outlined in (Jensen, *et al.* 1994) is adopted in this paper in which the term *bitemporal* refers to the accommodation of both valid-time (the time when the fact is true in the modelled world) and transaction-time (the time when it was physically recorded in the database). In a previous taxonomy (Snodgrass and Ahn 1985)

4. Architectural Issues

4.1. Schema conversion mechanisms

A number of suggestions have been proposed for the conversion of the schema at the physical level. Firstly, the complete schema can be converted to a new version as in Orion (Banerjee, *et al.* 1986; Kim, *et al.* 1989, 1990). This method, while being conceptually simple, prohibits the parallel schema versions required in some application environments. The approach of Skarra and Zdonik in Encore (Skarra and Zdonik 1986, 1987; Zdonik 1986) is to version at class level and thus permit parallel changes as long as they are in different classes. Secondly, Kim and Chou (1988), and later Andany, Leonard and Palisser (1991), present a system whereby views (or contexts) are constructed and schema evolution is achieved through view creation. This allows multiple concurrent versions of the schema. The relational equivalent can be considered as the creation of a completed or meta-schema. Thirdly, as in Charly (Palisser 1989), the objects can be made self descriptive thus making object and schema modification homogeneous. This method, while being potentially powerful, leads to other problems (for example, schema versioning by this method is difficult) and was later rejected by Andary *et al.* in favour of the Kim and Chou approach in a subsequent system.

4.2. Data conversion mechanisms

Currently, when a change to a schema is required any changes made by the Database Administrator are propagated to the data immediately (this parallels the concept of strict evaluation in functional programming languages). This results in the database being unavailable while the data are being modified and encourages a centralised schema change operation. This method is exemplified by the strict (or eager or early) conversion method adopted in GemStone (Penney and Stein 1987) in which a change to the schema results in an immediate propagation of that change to the data. This results in longer schema modification time but reduces subsequent data access time. An interesting extension to this approach is given by Lerner and Habermann (1990) where a data transformation table generator is used to produce routines to assist the Database Administrator with the data conversion.

As an alternative, Tan and Katayama (1989) propose a lazy mechanism for converting data in a database to the current version only when required. Given the assumptions that routines exist and may be run at any time to effect the conversion, the lazy evaluation method has the following advantages:

i. Changes to the schema can be made more rapidly,

---

the term *temporal* was used although this is now used to refer more widely to data models which support some aspect of time. Similarly the term *historical* has been replaced by *valid-time* and the term *rollback* has been replaced by the term *transaction-time.*

ii. Data are changed only when required and thus the identification of obsolete data is not required,

iii. The immediate withdrawal of a schema change operation is possible without effect. Furthermore, compensating schema changes may result in no physical data change at all[4].

The method proposed in ORION argues for a logical conversion only (Banerjee, *et al.* 1986, 1987b; Kim and Chou 1988). A system of screens is proposed which translates the attribute into the required format at data access time. No conversion is therefore required. This method has neither the schema modification overhead of GemStone, nor some of the data access overhead of the lazy conversion method of Tan and Katayama. However, the method leads towards a database of increasing complexity and therefore one needing to be rationalised at appropriate quiet points. Skarra and Zdonik introduce a method similar to that of ORION whereby identifiable versions are defined periodically to link all instances in a database (Skarra and Zdonik 1986, 1987; Zdonik 1986). A schema change in itself may not be sufficient to create a new version and individual instances which differ from the version interface are dealt with by an error handling routine. Thus access to data may involve instance variables being modified twice, the first time to ensure they adhere to the version interface and the second time for conversion to the required format.

4.3. Access right considerations

In object-oriented database systems where methods and attributes can be inherited from classes higher in the hierarchy, schema evolution changes can result in violations of access rights. Consider, for example, a change to an Employee class from which attributes are inherited to an Engineer class and for which the modifying user has no legitimate access. Any change to the definition of these inherited attributes can be considered to violate the access rights of the class. Moreover, in some systems ownership of a class does not imply ownership of all instances of that class. In GemStone (Bretl, *et al.* 1989) ownership of the class is considered sufficient authorisation to allow modification to all instances of that class and any subclasses that may inherit attributes.

4.4. Concurrency considerations and concurrent schemata

In a multi-user environment, it may be possible to modify the database schema while another user is currently accessing the database. This aspect is explored by Sockut and Iyer (1993). These problems can be overcome if schema versioning is accommodated but are significant if static schema evolution only is supported. This problem becomes more acute when two users are modifying related schema definitions at one time.

---

4 Consider for example the reversed business decision not to hold cents on financial values.

Multiple concurrent versions have been proposed by a number of researchers, most notably by Zdonik (1986) who proposes the concept of a *surface of consistency*. Under this approach two users may both read, modify and update the same schema version resulting in two, probably inconsistent versions but which are both consistent with the remaining definition in the database. In this way a surface of consistency is developed linking consistent parts of the database definition. New transactions are required to choose between the alternative surfaces of consistency when the transaction is started and a system of *conversational merging* is used to coalesce divergent paths. This aspect also relates to the meta-level transactions discussed in Section 3.2. If multiple versions are permitted multiple future schema version(s) could be tested and refined as alternatives to the current schema until implementation time when conversational merging would take place in the form of merging the requirements of the new versions and the imposition of translation functions from the current to the, now unified, new version.

5. Issues in query language support

With the possibility of multiple versions coexisting within a database system, the availability of data being retrievable through versions other than the current one becomes apparent. As discussed earlier, research into temporal databases (qv. work by Ben-Zvi (1982) and Snodgrass (1987)) has illustrated the difference in temporal perspective that users can adopt when viewing database data. By adding the facility of schema evolution users can view data by three independent mechanisms.

i. Valid-time (The time line pertaining to reality)

ii. Transaction-time (The time data was stored in the database)

iii. Schema-time (The time indicating the format of the data through reference to the schema active at that time)

The first two have been examined widely and will not be of concern here. The latter provides a mechanism which ensures that the data adheres to a specified version. The time the object model structure changed (the meta-database equivalent of valid time) can be considered as a fourth type of time. The sections below examine some of the issues of incorporating schema-time into a query language[5].

---

5 This was first published as part of an SQL extension presented in Roddick (1992b). Parts also appeared as commentaries in the current TSQL2 collaborative effort to accommodate temporal support within the SQL92 standard (ISO 1992; Roddick and Snodgrass 1993, 1994; Snodgrass, *et al.* 1994).

5.1. Levels of support for schema evolution in query languages

Five approaches may be taken to the handling of changing database structure by database query languages.

i. It can be ignored. That is, it is assumed that the schema is immutable once set up or that queries using an old schema definition are illegal. This is the current assumption used by most database query languages. Note that this does not preclude the use of temporal database systems as long as all data adhere to the same schema definition.

ii. A restriction can be included that no schema changes may have taken place during the interval(s) used to satisfy the query.

iii. It can be accommodated fully allowing database queries to be asked using any or all of the three (or more) temporal dimensions (valid, transaction and schema-time)[6]. A query using all three dimensions would be "Find all employees who earned more than $40,000 in 1992, as recorded on January 1, 1993, and report the details using the format in use in March 1993". This approach would provide the maximum resilience for application programs.

iv. Schemata can be allowed to change and the *effective* schema definition is that of a schema containing all attributes that were defined for the relation at *all* points during the transaction-time intervals used to satisfy the query.

v. Schemata can be allowed to change and the *effective* schema definition is that of a completed schema containing the union of all attributes that were defined for the relation at *any* point during the transaction-time intervals used to satisfy the query, cf. (Clifford and Warren 1983; Roddick 1991).

The first two approaches are simple but overly restrictive as, in many practical situations, schemata do change (qv. (Sjøberg 1992, 1993)). The third approach is the most expressive while the latter two are simplifications on this approach. Option v. is an attractive alternative to option iii. for the following reasons:

i. It degrades to the static case in the case where schema evolution is not used for the relation.

ii. Query language semantics can intuitively be fashioned to degrade to conventional (static) database query languages.

iii. It enables all relevant attributes to be displayed but without intuitively irrelevant attributes also being displayed.

iv. It is conceptually simple (although not as simple as the static case).

v. It enables database schema to be free of inactive attributes while still providing a mechanism for retrieval of the historical data.

---

6 Some researchers have identified situations where more than three time dimensions are necessary.

5.2. Completed schemata

It is often necessary to obtain a complete listing of the data held in a database for such purposes as audit or backup. In their temporal modelling research, Clifford and Warren examine the concept of the *completed relation* as a relation containing a tuple for every key that has existed in that relation (Clifford and Warren 1983). This concept can be applied to the schema to create (in relational database terminology) a *completed schema* with relations containing the union of attributes that has ever been defined for them, each with the least general domain which can include all domain values. In cases where a general domain cannot be used it is necessary to duplicate the attribute with enough different domains to hold all necessary values.

This completed schema can be considered as an overarching version and can be used to extract data across versions. This facility is difficult to implement in some of the proposals presented (most notably by Palisser (1989)) due to the searching necessary to compile the completed schema.

5.3. Problems presented by null values

The three-valued null logic proposed by Zaniolo (1984) and Roth, Korth and Silberschatz (1989) presupposes a static schema. The introduction of attributes (or relations) undefined at a given point in time introduces new semantics to null values in the database. Indeed, Zaniolo's 3-valued null logic can be extended by a dimension as follows:

	Attribute is Defined		**Attribute is Not Defined**	
	Value is Known	**Value is Unknown**	**Value is Known**	**Value is Unknown**
Attribute is Applicable	value	ω_1 = UNK	ω_4	ω_5
Attribute is Inapplicable		ω_2 = DNE		ω_6
Applicability is Unknown	ω_3 = NI		ω_7	

Table 3 Null value interpretations

For example, for attributes *not currently* defined their value is dependent on the reason behind their non-existence. Intuitively, a student does not cease to have a marital status simply because the information is no longer collected. Thus in this case a *value unknown* null would be an appropriate interpretation for the missing data. However a change in the object system (such as the subdivision of

a University into Faculties) may result in additional attributes being introduced. In this case an *attribute inapplicable* would be appropriate for extant data[7].

5.4. Schema valid-time support

Many of the temporal models proposed in the literature suggest a duality of time lines with respect to stored data; the date the real world event occurred and the date the data about that event was stored in the database. This duality exists also in the definition of the schema; the date the schema change is to take effect and the date the schema modification is recorded in the (meta) database. McKenzie and Snodgrass propose an extension to the relational algebra to support schema transaction time but argue that schema valid time is not necessary *since it* (schema evolution) *defines how reality is modeled by the database* (McKenzie and Snodgrass 1990). It could be argued however that the same reasons advanced for the holding of valid-time data (rollback, auditing, etc.) can be advanced for accommodating schema valid time. The utility of adding this aspect to the query language is largely dependent on the user's auditing requirements for the information system and the system's capacity to hold such data.

5.5. Schema-time projection

The relation project operator involves the specification of a subset of the relation's attributes through which the base relation is viewed. Versioning requires that we determine the effective schema used for data retrieval, which can be considered as one of a number of possible views of the underlying data. The term *schema projection* is therefore used to describe this viewing operation, the two foremost aspects of which are the method of schema specification and the method of *effective* schema construction.

5.5.1. Implicit .v. explicit specification

The specification of the schema version required may either be implicit in the rest of the query or explicitly stated by the user. If implicit there are a number of ways in which the schema could be selected, including:

---

7 The problems presented by null values is becoming increasingly complex and can be associated with the problems of fuzzy data, unreliable data, unavailable data and granularity mismatch. For example, researchers looking at access authorisation, especially in an OODBMS context, have suggested a null indicating *Not Authorised* (Gal-Oz, Gudes and Fernandez 1993). Other researchers, including the author, have suggested nulls for *Temporally Inapplicable Attributes* (Roddick 1991) and for attribute values representing higher level concepts of the correct data. Table 3, for example, shows that Zaniolo's 3-valued null logic may be expanded to a 7-valued null logic with the current understanding of a null being a set $\Phi \equiv \{\omega_1 - \omega_7\}$. $\omega_4 - \omega_7$ have different semantics from $\omega_1 - \omega_3$. ω_4, for example, indicates that the database holds a known value in an applicable attribute but that the schema being used prohibits its display (although for structural rather than for security reasons). Much of this work, however, is outside the scope of this paper and is the subject of future work. See also Atzeni and De Antonellis (1993).

i. the current schema;
ii. a default schema (which may not be the current schema);
iii. a function of the (implicit or explicit) transaction-time of the query; or
iv. a function of the (implicit or explicit) valid-time of the query.

The first option is the simplest and, given the need to accommodate legacy applications, arguably the most attractive. The other two require temporal query language support in addition to schema versioning (which may be provided separately).

Alternatively, the user may explicitly state the version by either:
i. specifying by version date;
ii. specifying by version name (if applicable);
iii. specifying by reference to the transaction-time interval specified in the query;
iv. specifying by reference to valid-time interval specified in the query.

The first option allows for some measure of program stability by allowing embedded database queries within an application to specify *version as at* <*compile-time*>. As above, the latter two require temporal query language support.

5.5.2. Simple .v. constructed schemata

In addition to the aspect above, the explicit specification of schema-time may be given as either an event (the schema current at a specified time) or as an interval (some function of the schemata current during a specified period). The former results in the use of a schema definition as defined at some point in time. This mode of schema specification is termed *simple*. For simple schema specification it is sufficient to give an absolute time or the version name. For the latter a schema is constructed from those active versions during the specified (or implied) interval. The two obvious functions are the intersection and union (completion) of the attributes active for the relations during that period.

5.6. Schema-time selection

Schema-time selection is used to access data based on the schema employed to store the data or the format the data currently adheres to. This may be necessary in some systems in order to access old format data. As discussed earlier, the ability to remove erroneous schema changes is linked to the mechanisms invoked when changes are required to the data. From the query language design viewpoint these are largely architectural considerations although some approaches result in data being held in heterogeneous formats, some of which may be excessively old. In these situations the query language must support the periodic “updating” of such data by providing selection predicates indicating schema format time. Note that the transaction-time values used in temporal query

languages may not necessarily be sufficient as they indicate only the date of the last modification *by a user*, rather than of a modification by the system for performance purposes.

5.7. Version naming

Versioning of schemata requires that a method of version naming be adopted. This can be any (or all) of a user-defined naming convention, a system-defined naming convention or system-defined time-stamping mechanism. This may be employed when a new schema is committed as follows:

i. A new version is created which can be referenced through either (a function of) the time of creation or with a user-defined name;

ii. A new version is created which can only be referenced through (a function of) the time of creation;

iii. A new version is not created, ie. there may be multiple evolutionary transactions between versions.

The naming of versions is most likely to occur at schema-level commit time for two reasons. Firstly, many query languages, including SQL, have no transaction start command and secondly, version naming is only required if the schema change is committed.

5.8. Casting of output attribute domains

While not directly necessary for schema evolution, a casting or user-defined conversion mechanism (such as in C and in the SQL92 language standard) may provide stability for applications which use embedded queries by allowing applications to coerce the data value to the required format. The conversion routines are intuitively simple and the null, or some other specified value, may be used in cases where conversion is not possible. In earlier work (Roddick 1993) a user-defined value (essentially a second null value) is proposed which can be used when a datum is incompatible with the target format.

6. Other related research issues

A number of research areas are related to the problems of database schema versioning/evolution. For example, work by Sjøberg (1992, 1993) investigates the quantification of schema changes within database systems in order to understand the ways in which schema changes are applied to actual systems both under development and in use. Such knowledge may be used to influence the architectural considerations for databases with schema evolution support; for instance in the choice of lazy or eager data conversion.

In temporal databases the concept of *vacuuming* allows for the physical deletion of temporal data in cases where the utility of holding the data is outweighed by the cost of doing so (Jensen 1993). Similar

consideration must be given to the retention of old schema definitions, especially in cases where no data exists adhering to either that version (physically) or referring, through its transaction-time values, to the period in which the definition was active. In (Roddick and Snodgrass 1994) two pragmatic positions are proposed:

i. All schema definitions which pre-date all data (both in format and in transaction-time values) are to be considered obsolete and should be deleted;
ii. Old schema definitions are considered valuable independent of whether data exists and may only be deleted through a special form of vacuuming.

For the sake of simplicity, the option adopted in (Roddick and Snodgrass 1994) was the former, however, both options are worth further research.

7. Further Research

This paper introduced the area of schema versioning and schema evolution in database systems including some of the associated areas such as query language support. Although various prototype systems have been developed and many of the ideas are starting to be incorporated into commercial systems, particularly object-oriented systems, significant research areas remain. In particular the following issues need to be addressed in more detail.

i. The ability to accommodate schema evolution within existing database systems. This includes accommodating schema evolution into existing query languages.
ii. The relationship of database schema evolution to the evolution of software systems.
iii. The pragmatic limitations of automated schema evolution. That is, to what extent should a database system assume that existing data can be accommodated under the revised structure and when should the DBA be required to direct changes?

A prototype database system Boswell is currently being constructed which aims to incorporate not only schema evolution (structure driven changes) but also transient inductively generated facts discovered through data mining (changes to which can be considered as update-driven schema evolution) (Roddick, Craske and Richards 1994). Some of the issues discussed here are thus one part of a larger project aimed at capturing and using the dynamic nature of database systems.

Some of the ideas contained here have also been proposed as background to a temporal extension to the SQL92 standard, TSQL2. The commentaries giving background discussions on this initiative are available on-line in the `tsql/doc` directory at `FTP.cs.arizona.edu` via anonymous FTP while further information on this initiative may be obtained from Prof. Richard Snodgrass at the University of Arizona.

Acknowledgments

I would like to acknowledge helpful support and comments from A/Prof. Tom Richards, La Trobe University, Dr Noel Craske, Monash University, Prof. Chris Marlin, Flinders University and Prof. Richard Snodgrass, University of Arizona. This work was done while on study leave at La Trobe University, Victoria, Australia and at the Flinders University of South Australia. The research was supported, in part, by a research grant from the University of South Australia.

I would also like to thank the anonymous referees for their useful comments on a previous version of this article.

References

Andany, J., Leonard, M. and Palisser, C. 1991. 'Management of schema evolution in databases'. In *Proc. 17th International Conference on Very Large Databases*, Barcelona, Spain. G.M. Lohman, A. Sernadas and R. Camps (eds.). Morgan Kaufmann, San Mateo, CA. 161-170.

Ariav, G. 1991. 'Temporally oriented data definitions: managing schema evolution in temporally oriented databases'. *Data Knowl. Eng.* **6**(6):451-467.

Atzeni, P., Ausiello, C., Batini, C. and Moscarini, M. 1982. 'Inclusion and equivalence between relational database schemata'. *Theoretical Computer Science.* **19**267-285.

Atzeni, P. and De Antonellis, V. 1993. *Relational database theory*. Benjamin/Cummings, Redwood City, SA.

Banerjee, J., Chou, H.-T., Garza, J.F., Kim, W., Woelk, D. and Ballou, N. 1987a. 'Data model issues for object-oriented applications'. *ACM Trans. Off. Inf. Syst.* **5**(1):3-26.

Banerjee, J., Chou, H.-T., Kim, H.J. and Korth, H.F. 1986. 'Schema evolution in object-oriented persistent databases'. In *Proc. 6th Advanced Database Symposium*, Tokyo. 23-31.

Banerjee, J., Chou, H.-T., Kim, H.J. and Korth, H.F. 1987b. 'Semantics and implementation of schema evolution in object-oriented databases'. *ACM SIGMOD conference, SIGMOD Record.* **16**(3):311-322.

Beech, D. and Mahbod, B. 1988. 'Generalised version control in an Object-oriented database'. In *Proc. 4th IEEE International Conference on Data Engineering*, Los Angeles, CA. IEEE Computer Society Press. 14-22.

Ben–Zvi, J. 1982. 'The time relational model'. Ph.D. thesis, University of California, Los Angeles.

Bjornerstedt, A. and Hulten, C. 1989. 'Version control in an object-oriented architecture'. In *Object-Oriented Concepts, Databases and Applications*. W. Kim and F. Lochovsky (eds.), Addison-Wesley / ACM Press, New York. 451-485.

Bretl, R., Maier, D., Otis, A., Penney, J., Schuchardt, B., Stein, J., Williams, E.H. and Williams, M. 1989. 'The GemStone data management system'. In *Object-oriented Concepts, Databases and Applications*. W. Kim and F. Lochovsky (eds.), ACM Press, New York. 283-308.

Chou, H. and Kim, W. 1988. 'Versions and change notification in an object-oriented database system'. In *Proc. 25th ACM/IEEE Design Automation Conference*,

Clamen, S.W. 1992. 'Class evolution and instance adaptation'. Technical Report CMU-CS-92-133. Carnegie Mellon University, Pittsburge, PA.

Clifford, J. and Ariav, G. 1986. 'Temporal data management: models and systems'. In *New Directions for Database Systems Ch. 12*. ABLEX Publishing Co., Norwood, N.J. 168-185.

Clifford, J. and Croker, A. 1987. 'The historical relational data model (HRDM) and algebra based on lifespans'. In *Proc. 3rd IEEE International Conference on Data Engineering*, Los Angeles, CA. IEEE Computer Society Press. 528-537.

Clifford, J. and Tansel, A.U. 1985. 'On an algebra for historical relational databases: two views'. *SIGMOD Rec.* **14**(4):247-265.

Clifford, J. and Warren, D.S. 1983. 'Formal semantics for time in databases'. *ACM Trans. Database Syst.* **8**(2):214–254.

Codd, E.F. 1970. 'A relational model for large shared data banks'. *Commun. ACM*. **13**(6):377-387.

Codd, E.F. 1972. 'Relational completeness of data base sublanguages'. In *Data Base Systems*. Courant Computer Symposia Series, Vol. 6. Prentice-Hall, Englewood Cliffs. 65-98.

Codd, E.F. 1979. 'Extending the database relational model to capture more meaning'. *ACM Trans. Database Syst.* **4**(4):397-434. An early version of this work was presented at the 2nd Australian Computer Science Conference in Hobart, TAS.

Dadam, P. and Teuhola, J. 1987. 'Managing schema versions in a time-versioned non-first-normal-form relational database'. Technical Report 87.01.001. IBM Heidelberg Scientific Center, Germany. Also published in *Proc. Datenbanksysteme in Buro, Technik und Wissenschaft*. Darmstadt, West Germany, Springer-Verlag, 161-179, in German.

Dean, T.L. and McDermott, D.V. 1987. 'Temporal database management'. *Artif. Intell.* **32**(1):1-55.

Ewald, C.A. and Orlowska, M.E. 1994. 'A theoretical approach to the understanding of relational schema evolution'. Technical Report 287. Key Centre for Software Technology, University of Queensland.

Gadia, S.K. 1988. 'A homogeneous relational model and query languages for temporal databases'. *ACM Trans. Database Syst.* **13**(4):418-448.

Gal-Oz, N., Gudes, E. and Fernandez, E.B. 1993. 'A model of methods access authorisation in object-oriented databases'. In *Proc. 19th International Conference on Very Large Databases*, Dublin, Ireland. R. Agrawal, S. Baker and D. Bell (eds.). Morgan Kaufmann, Palo Alto, CA. 52-61.

Geller, J., Perl, Y., Neuhold, E. and Sheth, A. 1992. 'Structural schema integration with full and partial correspondence using the dual model'. *Inf. Syst.* **17**(6):443-464.

Hsu, S., Jensen, C.S. and Snodgrass, R. 1992a. 'Valid-time projection in TSQL2'. TempIS Document No 30. University of Arizona. Also available as a TSQL2 commentary, TSQL2 language design committee.

Hsu, S., Jensen, C.S. and Snodgrass, R. 1992b. 'Valid-time selection in TSQL2'. TempIS Document No 30. University of Arizona. Also available as a TSQL2 commentary, TSQL2 language design committee.

ISO 1992. 'Information processing systems - database language SQL with integrity enhancement'. ISO Standard 9075:1992. International Standards Organisation. Also available as Australian Standard AS 3968-1991 and American Standard ANSI X3.135:1992.

Jensen, C., *et al.* 1994. 'A consensus glossary of temporal database concepts'. *SIGMOD Rec.* **23**(1):52-64. Also Technical Report R93-2035, Department of Mathematics and Computer Science, Aalborg University, Denmark, November, 1993.

Jensen, C.S. 1993. 'Vacuuming in TSQL2'. TSQL2 Commentary TSQL2 language design committee.

Kim, W., Ballou, N., Chou, H.-T., Garza, J.F. and Woelk, D. 1989. 'Features of the Orion object-oriented database system'. In *Object-oriented Concepts, Databases and Applications*. W. Kim and F. Lochovsky (eds.), ACM Press, New York. 251-282.

Kim, W. and Chou, H.-T. 1988. 'Versions of schema for object-oriented databases'. In *Proc. 14th International Conference on Very Large Databases*, Los Angeles, CA. F. Bancilhon and D.J. DeWitt (eds.). Morgan Kaufmann, Palo Alto, CA. 148-159.

Kim, W., Garza, J.F., Ballou, N. and Woelk, D. 1990. 'Architecture of the ORION next-generation database system'. *IEEE Trans. Knowl. and Data Eng.* **2**(1):109-124.

Larson, J., Navathe, S.B. and Elmasri, R. 1989. 'A theory of attribute equivalence in databases with application to schema integration'. *IEEE Trans. Softw. Engng.* **15**(4):449-463.

Lerner, B.S. and Habermann, A.N. 1990. 'Beyond schema evolution to database reorganisation'. *SIGPLAN Not.* **25**(10):67-76.

Lientz, B.P. 1983. 'Issues in software maintenance'. *ACM Comput. Surv.* **15**(3):271-278.

Lorentzos, N.A. and Johnson, R.G. 1988. 'TRA: a model for a temporal relational algebra'. In *Proc. IFIP TC 8/WG 81 Working Conference on Temporal Aspects in Information Systems*, Sophia-Antipolis, France. C. Rolland, F. Bodart and M. Leonard (eds.). Elsevier Science Publ. (North-Holland) Amsterdam. 95-108.

McKenzie, L.E. 1988. 'An algebraic language for query and update of temporal databases'. Ph.D. thesis, University of North Carolina.

McKenzie, L.E. and Snodgrass, R.T. 1987a. 'Extending the relational algebra to support transaction time'. *SIGMOD Rec.* **16**(3):467-478.

McKenzie, L.E. and Snodgrass, R.T. 1987b. 'Scheme evolution and the relational algebra'. Technical Report 87-003. Department of Computer Science, University of North Carolina, Chapel Hill, NC.

McKenzie, L.E. and Snodgrass, R.T. 1987c. 'Supporting valid time: an historical algebra and evaluation'. Technical Report TR87-008. Computer Science Department, University of North Carolina, Chapel Hill.

McKenzie, L.E. and Snodgrass, R.T. 1990. 'Schema evolution and the relational algebra'. *Inf. Syst.* **15**(2):207-232.

McKenzie, L.E. and Snodgrass, R.T. 1991. 'Evaluation of relational algebras incorporating the time dimension in databases'. *ACM Comput. Surv.* **23**(4):501-543.

Miller, R.J., Ioannidis, Y.E. and Ramakrishnan, R. 1993. 'The use of information capacity in schema integration and translation'. In *Proc. 19th International Conference on Very Large Databases*, Dublin, Ireland. R. Agrawal, S. Baker and D. Bell (eds.). Morgan Kaufmann, Palo Alto, CA. 120-133.

Monk, S.R. and Sommerville, I. 1992. 'A model for versioning of classes in object-oriented databases'. In *Proc. 10th British National Conference on Databases*, Aberdeen. P.M.D. Gray and R.J. Lucas (eds.). Springer-Verlag. 42-58.

Monk, S.R. and Sommerville, I. 1993. 'Schema evolution in OODBs using class versioning'. *SIGMOD Rec.* **22**(3):16-22.

Narayanaswamy, K. and Bapa Rao, K.V. 1988. 'An incremental mechanism for schema evolution in engineering domains'. In *Proc. 4th IEEE International Conference on Data Engineering*, Los Angeles, CA. IEEE Computer Society Press. 294-301.

Nguyen, G.T. and Rieu, D. 1989a. 'Schema change propagation in object-oriented databases'. In *Proc. IFIP 11th World Computer Conference*, San Francisco, CA. G.X. Ritter (ed.) North-Holland. 815-820.

Nguyen, G.T. and Rieu, D. 1989b. 'Schema evolution in object-oriented database systems'. *Data Knowl. Eng.* **4**(1):43-67.

Orlowska, M.E. and Ewald, C.A. 1991. 'Meta-level updates: the evolution of fact-based schemata'. Technical Report 211. Key Centre for Software Technology, Department of Computer Science, University of Queensland.

Orlowska, M.E. and Ewald, C.A. 1992. 'Schema evolution - the design and integration of fact-based schemata'. In *Research and Practical Issues in Databases, Proc. 3rd Australian Database Conference*. B. Srinivasan and J. Zeleznikow (eds.), World Scientific, La Trobe University. 306-320.

Osborn, S.L. 1989. 'The role of polymorphism in schema evolution in an object-oriented database'. *IEEE Trans. Knowl. and Data Eng.* **1**(3):310-317.

Palisser, C. 1989. 'Charly, un gestionnaire de versions pour la CAO en architecture'. Doctoral thesis, Aix-Marseilles.

Penney, D.J. and Stein, J. 1987. 'Class modification in the GemStone object-oriented DBMS'. *SIGPLAN Not. (Proc. OOPSLA '87).* **22**(12):111-117.

Roddick, J.F. 1991. 'Dynamically changing schemas within database models'. *Aust. Comput. J.* **23**(3):105-109.

Roddick, J.F. 1992a. 'Schema evolution in database systems - an annotated bibliography'. *SIGMOD Rec.* **21**(4):35-40. An updated version of the bibliography may be obtained from the author.

Roddick, J.F. 1992b. 'SQL/SE - a query language extension for databases supporting schema evolution'. *SIGMOD Rec.* **21**(3):10-16.

Roddick, J.F. 1993. 'Implementing schema evolution in relational database systems: an approach based on historical schemata'. Technical Report 10/93. Department of Computer Science and Computer Engineering, La Trobe University.

Roddick, J.F. 1994. 'Schema evolution in database systems - an updated bibliography'. Technical Report CIS-94-012. School of Computer and Information Science, University of South Australia. An earlier version appeared in *SIGMOD Rec.* vol. 21, no. 4, pp. 35-40. 1992.

Roddick, J.F., Craske, N.G. and Richards, T.J. 1993. 'A taxonomy for schema versioning based on the relational and entity relationship models'. In *Proc. 12th International Conference on Entity-Relationship Approach*, Dallas, Texas. R. Elmasri (ed.) 139-150. Also appears in Lecture Notes in Computer Science, Springer-Verlag, 143-154.

Roddick, J.F., Craske, N.G. and Richards, T.J. 1994. 'Handling discovered structure in database systems'. *IEEE Trans. Knowl. and Data Eng.* Accepted for Publication.

Roddick, J.F. and Patrick, J.D. 1992. 'Temporal semantics in information systems - a survey'. *Inf. Syst.* **17**(3):249-267.

Roddick, J.F. and Snodgrass, R.T. 1993. 'Transaction-time support in TSQL2'. TSQL2 Commentary TSQL2 language design committee.

Roddick, J.F. and Snodgrass, R.T. 1994. 'Schema versioning support in TSQL2'. TSQL2 Commentary TSQL2 language design committee.

Roth, M.A., Korth, H.F. and Silberschatz, A. 1989. 'Null values in nested relational databases'. *Acta Inf.* **26**(7):615-642.

Sarda, N.L. 1990. 'Algebra and query language for a historical data model'. *Comput. J.* **33**(1):11-18.

Selinger, P.G. 1993. 'Predictions and challenges for database systems in the year 2000'. In *Proc. 19th International Conference on Very Large Databases*, Dublin, Ireland. R. Agrawal, S. Baker and D. Bell (eds.). Morgan Kaufmann, Palo Alto, CA. 667-675.

Shneiderman, B. and Thomas, G. 1982. 'An architecture for automatic relational database system conversion'. *ACM Trans. Database Syst.* **7**(2):235-257.

Sjøberg, D. 1992. 'Measuring schema evolution'. Technical Report FIDE/92/36. Department of Computer Science, University of Glasgow.

Sjøberg, D. 1993. 'Quantifying schema evolution'. *Inf. Softw. Technol.* **35**(1):35-44.

Skarra, A.H. and Zdonik, S.B. 1986. 'The management of changing types in an object-oriented database'. *SIGPLAN Not. (Proc. OOPSLA '86).* **21**(11):483-495.

Skarra, A.H. and Zdonik, S.B. 1987. 'Type evolution in an object-oriented database'. In *Research directions in object-oriented programming.* B. Shriver (ed.) MIT Press, Cambridge, MA. 393-416.

Snodgrass, R. 1987. 'The temporal query language TQUEL'. *ACM Trans. Database Syst.* **12**(2):247–298.

Snodgrass, R. 1992. 'Schema specification in TSQL2'. TSQL2 Commentary TSQL2 language design committee.

Snodgrass, R. and Ahn, I. 1985. 'A taxonomy of time in databases'. *SIGMOD Rec.* **14**236–246.

Snodgrass, R. and Ahn, I. 1986. 'Temporal databases'. *IEEE Computer.* **19**35–42.

Snodgrass, R.T., Ahn, I., Ariav, G., Batory, D.S., Clifford, J., Dyerson, C.E., Elmasri, R., Grandi, F., Jensen, C.S., Käfer, W., Kline, N., Kulkarni, K., Leung, T.Y.C., Lorentzos, N., Roddick, J.F., Segev, A., Soo, M.D. and Sripada, S.M. 1994. 'TSQL2 language specification'. *SIGMOD Rec.* **23**(1):65-86.

Sockut, G.H. and Iyer, B.R. 1993. 'Reorganising databases concurrently with usage'. Technical Report TR 03.488. IBM Santa Teresa Laboratory.

Stonebraker, M., Agrawal, R., Dayal, U., Neuhold, E.J. and Reuter, A. 1993. 'DBMS research at the crossroads: the Vienna update'. In *Proc. 19th International Conference on Very Large Databases*, Dublin, Ireland. R. Agrawal, S. Baker and D. Bell (eds.). Morgan Kaufmann, Palo Alto, CA. 688-692.

Tan, L. and Katayama, T. 1989. 'Meta operations for type management in object-oriented databases - a lazy mechanism for schema evolution'. In *Proc. First International Conference on Deductive and Object-Oriented Databases, DOOD '89.*, Kyoto, Japan. W. Kim, J.-M. Nicolas and S. Nishio (eds.). North-Holland. 241-258.

Tansel, A.U. 1986. 'Adding time dimension to relational model and extending relational algebra'. *Inf. Syst.* **11**(4):343-355.

Tansel, A.U., Clifford, J., Gadia, S.K., Jajodia, S., Segev, A. and Snodgrass, R.T. 1993. *Temporal databases: theory, design and implementation*. Benjamin Cummings, Redwood City, CA.

Tuzhilin, A. and Clifford, J. 1990. 'A temporal relational algebra as a basis for temporal relational completeness'. In *Proc. 16th International Conference on Very Large Databases*, Brisbane. D. McLeod, R. Sacks-Davis and H. Schek (eds.). Morgan Kaufmann. 13-23.

van Bommel, P. 1993. 'A randomised schema mutator for evolutionary database optimisation'. *Aust. Comput. J.* **25**(2):61-69.

Ventrone, V. and Heiler, S. 1991. 'Semantic heterogeneity as a result of domain evolution'. *SIGMOD Rec.* **20**(4):16-20.

Zaniolo, C. 1984. 'Database relations with null values'. *J. Comput. Syst. Sci.* **28**(1):142-166.

Zdonik, S.B. 1986. 'Version management in an object-oriented database'. In R. Conradi, T.M. Didriksen and D.H. Wanvik (eds.), Lecture Notes in Computer Science, Vol. 244. Springer-Verlag, Berlin. 405-422.

Zdonik, S.B. 1990. 'Object-oriented type evolution'. In *Advances in Database Programming Languages*. F. Bancilon and P. Buneman (eds.), ACM Press / Addison-Wesley, New York. 277-288.