A TEMPORAL DATABASE MODEL USING NESTED RELATIONS

Georgia Garani

A Thesis Submitted in Fulfilment of the Requirements for the Degree of

Doctor of Philosophy

in the

UNIVERSITY OF LONDON

October, 2003

School of Computer Science and Information Systems

BIRKBECK COLLEGE

“Kaˆ crÒnoj mšn ™sti tÕ di£sthma kaq' Ö pr£tteta… ti.”

’OlumpiÒdwroj, Commentarii in Ecclesiasten,
Vol. 98, p. 508, l. 8 (Migne, *Patrologia Graeca*)

ABSTRACT

Relaxing the First Normal Form (1NF) assumption of relational databases gives rise to Non-First Normal Form relations or nested relations for short. Nested relations overcome a number of problems that the apparently reasonable restriction of 1NF condition causes.

The need to support time in database systems, in order to model temporal events in the real world, has been addressed over the last two decades, reflecting the importance of that for almost every computer system application.

This thesis combines the features of previous nested and temporal models to develop a new integrated Temporal Nested Model (TNM).

TNM is a temporal, nested, attribute timestamping, heterogeneous database model, where time is represented as temporal elements. It is defined as an extension of a Nested Relational Model (NRM) which is also formally defined in the thesis.

All the operations of the NRM are formalised. The recursive rename operation for nested relations is defined for the first time. A recursive natural join operation is also formally defined, to cover all the different cases of the common attributes that participate in the join of two nested relations. This natural join operation is more general than all other natural join operations that have been defined so far.

The operations of NRM are next extended in order to support the temporal dimension of the TNM. Formal definitions for all the operations of the TNM are given and the operations are proved to be closed.

A formal proof is given which shows that the TNM is a superset of the Conventional Relational Model.

A number of examples of the management of temporal nested data using the TNM are also given. The queries illustrate the features of the temporal nested relational algebra, together with the expressive power of the new model and the ease of use of the algebra.

Finally, a comparison with other temporal models is given and the capabilities of the TNM are discussed.

A prototype implementation has also been undertaken in Miranda to illustrate the functionality of the models defined in this thesis.

CONTENTS

ACKNOWLEDGMENTS

A number of people have helped me significantly through out all these years of my research. The contribution of my supervisor, Dr Roger Johnson, to this work, is inestimable. He was always supportive, encouraging and patient, willing to give his guidance and advice to every problem that arose. Professor George Loizou, Head of the Computer Science Department of Birkbeck College, has helped me significantly with his unique way of parental interest and support. I am also grateful to Nikos Lorentzos, Associate Professor of the Agricultural University of Athens, for many discussions and constructive comments and suggestions on my work. Furthermore, many thanks are also due to the external examiner of this thesis, Professor Peter Gray, for his very helpful comments which were much appreciated.

I would also like to thank the System Group of the Computer Science Department and especially Phil Gregg and Andrew Watkins, for providing me with all the necessary computing facilities and Ms Betty Walters, the executive officer of the Computer Science Department, for providing an excellent working environment in which I was pleased to study.

Lastly and most importantly, my family deserves many thanks. I am indebted to my parents, Elias and Katerina, for their emotional and financial support. They were always so close to me, ready to share and solve my problems and worries. Angelos and Christina Efstathiou have been excellent parents in law. I would like to thank them for their moral support and patience. My sister, Myrto, has been wonderfully understanding and encouraging through out this long process.

There are not enough words to thank my husband, Dr Athanasios Efstathiou, for what he has offered me all these years. He was the source of inspiration and power for me. He has given me boundless love, support, and encouragement, which was so important for the completion of this research. Without him this work would not have been made possible. This thesis is dedicated to him.

The work documented in this thesis was partially funded by Birkbeck College Fees Awards Scheme for which I express my thanks.

CHAPTER 1

INTRODUCTION

1.1 Background

Time is a ubiquitous feature of real world phenomena. Every activity or change in the real world takes place in the context of time both at the microscopic and macroscopic level. Since prehistoric eras, people in every culture have been preoccupied with measuring and recording the passage of time. Today, in an age dominated by technology and information systems, the recording of time is even more vital and essential since only few real world applications do not have a temporal component.

Particularly in the field of database technology, the storage of data without a temporal dimension could signify, in the worst case, a vain attempt at recording information since information that changes over time cannot be recorded. Conventional databases without a temporal dimension record single states of real world phenomena. Every change of data, whether deletion, insertion or update, transitions the database from one state to another state. In consequence, past database states are not retained, resulting in a number of limitations:

- The evolution of real world phenomena over time cannot be recorded.
- Only queries that concern the current state of the database can be answered.
- Real world changes that will occur in the future or have occurred in the past cannot be recorded.
- Lack of functions to express common queries such as common points of two overlapping time intervals.

Over the years, organisations and researchers have been trying to solve this highly demanding problem of time by providing special, application dependent facilities. In the most basic approaches, time was treated as another field in a database table and queries concerning time were expressed with conventional SQL queries. These queries have an increased complexity, are error-prone, time consuming to formulate and lack expressive power.

The development of generalised facilities for the direct support of time in databases has been inevitable. Consequently, in the last twenty years there has been a growing interest in extending data models to incorporate the time dimension. Database management systems providing support for the storage and retrieval of time-dependent data are called temporal database management systems and the corresponding databases, temporal databases.

Time in temporal databases can be expressed in many different ways; it may be of interest to record the exact time when an event happened or the period in which an event took place or the duration of an event or even the periodicity of an incident – the frequency with which an incident occurs in time. Time is used to distinguish between past, present or future states. The recording of time allows the identification when facts are true in the modelled reality (valid time) or when facts are current in the database (transaction time). Time can stamp either tuples (tuple timestamping) or attributes (attribute timestamping) in relations. This diversity has produced many different temporal database models over the last twenty years.

In chapter 2 there is a detailed review of those temporal data models which relate most closely to the work in this thesis. In order to provide the reader with a general introduction to work in the wider area, there follows now a short overview of developments across the whole field of temporal databases. The first attempt to formally define a temporal database model was appeared in [Ben82]. Ben-Zvi calls his model Time Relational Model (TRM) and also proposes a query language for it. His model supports both valid and transaction time. TRM is a tuple timestamping model with five additional implicit time attributes, effective-time-start, effective-time-stop, registration-time-start, registration-time-stop and deletion-time. Static relation states as well as temporal states can be supported in his model. His temporal relational algebra is extremely limited. He defines a new operation, Time-View, which produces a relation state from a temporal state. Every operation defined in Ben-Zvi's algebra uses this new operation to construct a conventional relation to which standard relational operations can be applied. Consequently, Ben-Zvi does not provide a temporal formalism or a set of temporal query facilities.

Since this first attempt, a plethora of papers about temporal databases have appeared in the literature. Bibliographies about temporal databases have been published over the years showing the considerable activity of researchers in this field ([BADW82], [Mck86], [SS88], [Soo91], [Kli93], [TK96] and [WJW97]). Gadia, Tansel, Clifford, Lorentzos, McKenzie, Snodgrass, Jensen, to name just a few, have worked extensively in this field and made significant contributions. They have all contributed to the general understanding of the subject and have helped in the development of alternative temporal database modelling approaches and in more or less uniformity in the definitions of basic concepts and terms.

In 1993 the first book on temporal databases appeared ([TCG+93]). It is a collection of papers by the pioneers in the field covering different aspects of temporal databases such as formalisations of new temporal data models, temporal query languages and their completeness as well as the implementation of temporal database management systems. A glossary of temporal database concepts is included in this book as a result of e-mail discussions among the temporal database community. This glossary has since been revised and the latest version can now be found at http://www.cs.auc.dk/~csj/Glossary. A special issue of the IEEE Transactions on Data and Knowledge Engineering was devoted to temporal and real-time databases in August 1995. In 2000 a book was published on time granularities in databases, which is vital when designing and implementing databases supporting temporal data ([BJW00]). In 2003 another book was published, showing the endlessly interest on the temporal data and the relational model. It is a detailed investigation into the application of interval and relation theory to the problem of temporal database management ([DDL03]).The first international workshop on an Infrastructure for Temporal Databases was organised in 1993 in Arlington, Texas, followed by the 1995 International Workshop on Temporal Databases held in Zurich ([CT95]) and two Dagstuhl Seminars, one on Temporal Databases in 1997 ([EJS97]) and another on Integrating Spatial and Temporal Databases in 1998 (http://timelab.co.umist.ac.uk/events/dag98/). These workshops brought together researchers interested in the development of tools for the management of temporal data. In 2001 another symposium was held at

Redondo Beach, CA ([JSST01]) concerning both spatial and temporal databases.

In parallel with these developments, the temporal database research community has worked on the specification of a temporal query language. A significant work has been done by a committee comprising twenty one members residing in eight different countries that proposed a consensus temporal extension to SQL-92, called TSQL2 ([Sno95]), which has not become a standard.

A new data type "PERIOD" has been included in SQL/Temporal, which is not yet a standard. Currently, there are no plans to publish a temporal SQL standard.

Another book recently published by Snodgrass ([Sno00]) deals with the development of temporal database applications in SQL.

The activity described above clearly shows the importance of temporal databases and the interest of the database research community in the subject. It is remarkable, therefore, that there is still no widely accepted temporal query language and implementation available to users. Furthermore, the research field is still active, since answers have not yet been given to a variety of questions important in this area. These aspects include the definition of a data model that can support the essential semantics of time-varying relations, temporal data presentation, temporal data storage, efficient temporal query evaluation and temporal implementation strategies.

1.2 Motivation of the Thesis

A plethora of incompatible temporal data models have been proposed so far. Some of them are simple extensions to the Conventional Relational Model and others contain quite complicated new proposals. Their differences arise because of their varying support on the following:

- homogeneous or heterogeneous relations (having attribute values defined for the same or different time intervals in a tuple respectively),
- tuple or attribute timestamping representation,
- valid time (historical models), transaction time (rollback models) or valid and transaction time (bitemporal models),

- time representation as time points, time intervals or time elements (sets of time intervals),
- 1NF or N1NF relations.

This diversity exacerbates the already complex problem of modelling time in databases.

The motivation of this thesis is to define a model which would provide as few representation constraints as possible and at the same time, as much expressive power as possible. For this reason, features that would tend to limit the representational capabilities of the model are rejected. Therefore, it is claimed that the model defined in this thesis is based on features that are more general than previous models. In pursuit of this objective, heterogeneous relations have been chosen rather than homogeneous, attribute timestamping rather than tuple timestamping, time elements rather than time points and time intervals, and finally N1NF rather than 1NF. Each of the four characteristics chosen is a generalisation of the option declined.

To conclude, the objective of this thesis is to design a model that achieves a balanced combination of expressive power with ease of representation, use and understanding.

1.3 Contribution of the Thesis

The major contributions of this thesis are the following:

1. A **Temporal Nested valid time relational Model** (TNM) is formalised for the representation of temporal and nested data. For the manipulation of these data TNM algebra is formally defined. It is proved that all the **operations** of the algebra are **closed**. Additionally, **TNM** is proved to be a **consistent extension of the CRM** (Conventional Relational Model).

2. **TNM** is a well-defined model that **achieves a very high rating against evaluation criteria**. The criteria, derived from previous research in the field ([Mck88]), are mutually compatible and well established. The advantages of TNM against other previous proposed temporal models are thus demonstrated.

3. A **Nested Relational Model** (NRM) is formalised for the representation of nested data. A recursive algebra for NRM is proposed. All the operations are formally defined, including also the **rename operation** for nested relations. **NRM** is proved to be a **superset of the CRM**.

4. The **nested generalised natural join** operation is formally defined recursively for nested relations for the first time. The generalised natural join can be applied to all pairs of "joinable" nested relations independently of the number of the common attributes between the two relations and their types, i.e. atomic or relation-valued at either the top or lower levels (same or different) of the two relations. Six distinct cases are identified, distinguished by the above-mentioned properties of the common attributes participating in the natural join operation.

5. The **temporal nested generalised natural join** operation is formally defined as a consistent extension of the generalised natural join operation for nested relations.

The full expressive power of TNM is demonstrated by a series of examples.

To conclude, the result of this research is an integration of temporal and nested database models producing a formally defined generalised temporal nested database model, in which not only temporal data but also all other static data can be nested to any finite depth so that the full power of the nested and temporal features can be exploited within one model.

1.4 Outline of the Thesis

The remainder of this thesis is organised in eight chapters.

Chapter 2 presents a survey of relevant existing database models. The chapter is divided into two parts. In the first part, six N1NF models are described and their most important characteristics are presented. The natural join operation is described for each of these models and specific deficiencies are discussed. These shortcomings motivate the generalised nested natural join proposed in chapter 4.

Eight different representative research approaches in the field of temporal databases are presented in the second part of this chapter. The temporal models of these approaches are described and the same example relation is presented in each of these models to illustrate the differences in their representational capabilities. Deficiencies of these models are also identified.

In **Chapter 3** basic temporal definitions used in the thesis are given, the properties of the model defined in the subsequent chapters are described and the design decisions justified.

The Nested Relational Model, NRM, is formalised in **Chapter 4**. All the operations of the model are defined recursively. An extended definition of the generalised natural join operation for nested relations is provided.

In **Chapter 5** the Temporal Nested relational Model, TNM, is formalised. The algebra of the TNM is formally defined and all the operations are proved to be closed.

The full expressive power of TNM is demonstrated by a series of examples in **Chapter 6**.

In **Chapter 7** it is proved that NRM and TNM are consistent extensions of the Conventional Relational Model.

In **Chapter 8** the temporal models described in Chapter 2 and TNM are evaluated against 22 compatible criteria. It is shown that TNM satisfies the majority of these criteria.

Finally, **Chapter 9** summarises the achievements of this thesis and points out directions for possible future research.

Two appendices have also been included in the thesis. **Appendix A** contains a formal syntax of the TNM algebra. In **Appendix B** a brief description of the prototype implementation that has been undertaken in Miranda is included, as well as selected parts of the code and examples presented in the thesis, occasionally with their results.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

A number of researchers have made proposals to relax the First Normal Form (1NF) assumption using the Non-First Normal Form (N1NF or NF^2) relations to solve problems in new applications such as text processing, engineering design systems and office automation and thus overcome a number of limitations imposed by the apparently reasonable restriction that 1NF causes. In section 2.2 the advantages of the Non-First Normal Form relational database model are presented and different approaches to support it are described.

At the same time, in the last two decades many papers have appeared which address the problem of supporting time in database management systems. Numerous researchers have discussed the problem of modelling the time dimension of events that occur in the real world which is very important to almost every computer system application including banking, medical records and accounting. To solve this vital and highly demanding problem, they have proposed a variety of techniques that have addressed it from different viewpoints. Different approaches for the incorporation of time in database systems are presented in 2.3, where models introduced by a number of researchers are described.

2.2 Non-Temporal Nested Models

The Non-First Normal Form relational database model, or more simply the nested relational model, allows relations to have attributes which can have non-atomic values, i.e. the latter are themselves relations, subrelations of the relation to which they belong. The NF^2 model provides the basis for the object-oriented database model. Many models ([JS82], [FT83], [AB86], [SS86], [TF86], [RKS88]) have been defined since 1977, when Makinouchi ([Mak77]) proposed, for the first time, the relaxation of the 1NF assumption.

In spite of the large number of NF^2 models, only a few query languages have been proposed for the management of non-first normal form relations (e.g. [RKB87], [LD98] and [WTWL96]) by reason of its difficulty. These are extensions of existing query languages, SQL and Query by Example. To the knowledge of the author of this thesis, [LD98] is the most recent. In [LD98], a Query by Example language for nested tables, called QBEN, has been described which allows the formulation of complex queries.

The use of a NF^2 model eliminates many problems. The NF^2 model enables data about an object to be represented within one relation rather than distributing it over several relations. One major advantage is the fact that join operations which are substantially expensive in terms of execution time can be avoided.

The Non-First Normal Form database models that have been developed so far can be divided into two categories. Models of the first category are called non-recursive models (e.g. [JS82], [TF86], [OOM87]) and those of the second category are called recursive models (e.g. [SS86], [AB86], [RKS88], [Col90], [Lev92], [LR94a]). The two approaches are distinguished by the recursive or non-recursive nature of the operators that have been defined by the distinct researchers. The difference is that recursive operators can be applied repeatedly to the subrelations at the different levels of a relation, while the non-recursive operators cannot. In section 4.3 of this thesis the superiority of the recursive models compared to the non-recursive ones is explained and justified.

Previous research on some of the most important NF^2 recursive models is reviewed below in chronological order. Non-recursive models are not discussed since, as explained in section 4.3, they are not preferred. The natural join operator is examined in detail for each model. The ability to join two or more relations is one of the most powerful features of relational systems ([Dat00]). However, the natural join operation is the most complicated operation involving nested relations since it is a binary operation and can be performed between a relation or a subrelation of a relation and another relation or subrelation of it. The relations that participate in the natural join operation must have common attributes that can be either atomic or relation-valued (subrelations) and which can be either at the top level of the relations or at some lower level, either the same or not. Several researchers (e.g. [AB86], [Col90], [DL91], [Lev92], [LC96], [RKS88]) have defined natural join operations between two nested relations. However, to the author's knowledge, all the definitions given so far have a very limited functionality as is explained in subsequent sections.

2.2.1 Abiteboul and Bidoit's model

In [AB86], Abiteboul and Bidoit present a Non-First Normal Form database model called the Verso model. The data structure and operations of the model are formally defined. The main characteristic of the proposed algebraic query language is that it allows data restructuring. Five unary operations (extension, projection, selection, restriction and renaming) and five binary ones (union, intersection, difference, join and cartesian product) are introduced as well as a restructuring operation. They claim that these operations all together are as powerful as the conventional relational algebra.

In the Verso model, a *format* specifies the underlying structure of a Verso instance (a generalisation of a relational instance).

The extension operation is not formally defined. The projection operation presented in this model is not a generalisation of the projection for flat relations, since the only projection that can be performed for a flat relation is over the whole relation. However, arbitrary projections can be achieved but they usually require a restructuring of the original relation. Two versions of the selection operation are defined in [AB86]. Firstly, a simple version of the selection operation, the Verso-selection, is introduced and later an extension of the selection, called the "super-selection" which can be expressed by the Verso-selection, projection, and join operations. The restriction operation is itself restricted in that it can be applied only to the "root" of the format.

They also define a join operation. It can be performed only between instances over *compatible formats*. Two relations have *compatible formats* when they have the same atomic attributes at the top level of their schemes. To overcome several limitations apparent with this definition, they introduce the restructuring operation which allows for the structure of a relation to be modified. However, the restructuring operation is not a nonloss operation. Therefore, three format transformations are defined which characterise a *format dominance* (one format is *dominated* by another one iff each instance over the first format can be represented by an instance over the second format containing the same information). These format transformations are root and branch permutation, compaction and extension. However, these format transformations cannot transform entirely the structure of all relations.

As a consequence of the limitations of the join and resructuring operation, join operations of practical interest cannot be formulated. An example is given below which shows that the join operation between two relations cannot be performed even after restructuring the relations.

Example 2.1: Suppose that two relations are given: the TRAINING_1 relation (Fig. 2.1) and the DEPT_1 relation (Fig. 2.2). TRAINING_1 relation has two attributes, COMPANY and PROGRAMME. COMPANY is an atomic attribute that represents the company name and PROGRAMME is a subrelation which contains two attributes, TRN atomic attribute which denotes the trainer's name and CODE′ which is a subrelation containing only one atomic attribute CODE, which shows the codes of the courses a trainer has taught. Relation TRAINING_1 has the following scheme: TRAINING_1 = COMPANY PROGRAMME(TRN CODE′(CODE)) (for more explanations about this notation see section 4.2).

Relation DEPT_1 has three attributes, two atomic, D and DN, and one nested, UNIT. UNIT has three attributes, the atomic attributes UN and UD and the nested attribute TRAINER having two attributes, the atomic attribute TRN and the nested attribute C having the atomic attributes CN and Y. Semantically, D is the department number, DN is the department name, UN is the unit number, UD is the unit description, TRN is the trainer's name for each course, CN is the course number that each trainer has given in year Y.

Relation DEPT_1 has the following scheme: DEPT_1 = D DN UNIT(UN UD TRAINER(TRN C(CN Y))).

COMPANY	PROGRAMME	
	TRN	CODE′
		CODE
Apple	Jack	xx0
	Mark	xy1
		xy2
IBM	Tim	xy1
		xx2
Microsoft	Karen	xx1

Fig. 2.1: TRAINING_1

D	DN	UNIT				
		UN	UD	TRAINER		
				TRN	C	
					CN	Y
1	Research	511	Software Engineering	Mark	1	75
					2	76
					5	79
		552	Basic Research	Karen	1	82
					2	79
				Tim	5	79
		678	Planning	Mark	2	76
					4	82
2	Development	650	Design	Karen	1	75
		780	Maintenance	Tim	3	82
				Mark	2	76
		981	Planning	Jack	2	81
					3	82
					5	79

Fig. 2.2: DEPT_1

Suppose that we want to find the names of the companies (COMPANY) for which trainers (TRN) have taught courses to technical employees together with the course number of each course (CN). The two relations must be joined on TRN, their common atomic attribute. However, TRN is at different nesting levels in the two relations and is not at the top level of any of them. Therefore, restructuring operations must be applied to both relations to transform their structure, in order to “move” TRN at the top level. However, this is not possible in [AB86]. This is because root permutation, branch permutation, compaction and extension can either move the attributes at the same nesting level or at lower nesting levels. The conclusion is that the join operation has limited capabilities.

The cartesian product operation (defined after the join operation) requires the first operand to be an instance over a flat relation and this is again a significant weakness.

The algebra that they define is very elaborate. New definitions and notations are given, even for the simplest concepts, such as a (Verso) instance over a format. Furthermore, the key feature of their model, the restructuring operation, cannot reconstruct entirely the structures of the relations without loss of information, even when using a combination of all three transformations, root and branch permutations, compactions and extensions, as has been demonstrated above by Example 2.1. As a result, the potentiality of the operation is limited to a restricted spectre of cases.

2.2.2 Roth, Korth and Silberschatz's model

An extended relational calculus and an equivalent algebra for Non-First Normal Form relations was defined in [RKS88] in order to unify the various theories of Non-First Normal Form databases that had been proposed up to that time.

The Partitioned Normal Form (PNF) property is defined for nested relations. A relation R is in PNF if all the atomic attributes of R form a key for the relation and recursively, each relation-valued attribute of the relation is also in PNF. As a result, nested relations can be divided into those in PNF and those not in PNF. [RKS88] shows that relations in PNF have some good properties compared to other relations. However, in general, relations in PNF impose two important restrictions, that there is at least one atomic attribute at every nesting level of the relation and also that relation-valued attributes cannot be part of the key.

Two new operators, nest and unnest, are added to the basic set of operators. The basic algebra operators, union, intersection, difference, cartesian product, natural join and projection are extended to work within the class of PNF relations. However, in [LL91] it has been proved that the extended projection of a nested relation in PNF is not a precise generalisation of the standard projection operator with respect to unnesting, as it is claimed in

[RKS89]. Union, intersection and difference are defined recursively. Cartesian product remains unchanged. Natural join is defined recursively. According to the definition, two tuples contribute to the natural join if the extended intersection of their projections over common attributes is not empty. This extended natural join can be performed either between relations which have common attributes at the top level of their scheme or between relations whose only common attributes are attributes of a common higher-order attribute. However, the case where the common attributes, either atomic or relation-valued, are at lower nesting levels, the same or different, in the two relations, is not covered. The extended projection is a normal projection followed by a tuplewise extended union of the result. The union merges tuples that agree on the zero-order names left in the projected relation. Finally, the selection operator is not extended.

Roth, Korth and Silberschatz also show that PNF relations are closed under the extended operators. It is important to note that these operators can be applied to non-PNF relations as well. The result, however, is not always a PNF relation.

The author of this thesis believes that this approach, proposed to provide a simple and simultaneously complete model for nested relational databases, has a number of limitations. These limitations concern firstly, the fact that the algebraic operators proposed in [RKS88] are defined in such a way that work within the class of PNF relations. Therefore, they are closed only under PNF relations. In general, the operators defined in [RKS88] are not closed. Secondly, some of the defined operations cannot be applied to subrelations of nested relations as is for example projection, selection, join and cartesian product operations. Consequently, there are cases that are not covered by this approach and as a result, advantages of nested based relational approaches are missed.

2.2.3 Colby's model

A recursive algebra for nested relations is defined by Colby in [Col90]. The classical operators are extended with recursive definitions in order to be used with subrelations of relations as well as with relations. As a result, restructuring or other operators which would be used as a "navigator" are avoided (as is for example the subrelation constructor in [DL91] which provides navigational ability; for more explanations about the subrelation constructor see below in Deshpande and Larson’s model). However, the nest and unnest operations are occasionally needed for accessing subrelations. Therefore, nest and unnest operations are defined recursively. The difference from nest and unnest operations defined in previous models is that they can be applied to subrelations directly, without transforming any other attribute of the relation. This is achieved by the assistance of a nest and an unnest list which are defined exactly for this purpose.

Union, intersection and difference are defined in two different ways. In the first way, non-recursive definitions are used, exactly as in [TF86], where only entire tuples participate in the operations, but not subrelations. The second way involves recursive definitions similar to those defined in [AB84], [DL87] and [RKS88], where the operations can be applied recursively to the subrelations of the tuples that share common key attributes. However, the PNF assumption, defined in [RKS88], is not made. Selection and projection operations are defined using the notions of a select and a project list respectively. They can be performed recursively at all levels of a nested relation without restructuring.

The cartesian product operation can be performed recursively between a relation and either another relation or a subrelation of another relation.

Colby defines the join operation recursively, for two cases, firstly where the common subrelations are at the top level in both relations that participate in the join and secondly where the common attributes are atomic which, while not at the top level in the first relation, are at the top level in the second. The second case requires the use of a join path that behaves as the guide-line which penetrates the different nesting levels of the first relation. Colby also suggests that it does not make sense to perform a join operation between subrelations which are not at the top level of two different relations. However, it is shown below (Example 2.2) that this is not correct since there are cases where such a join operation is meaningful and can take place between two subrelations of two relations.

Example 2.2: Consider the example database, which contains two relations, the TRAINING_2 relation (Fig. 2.3) and the DEPT_2 relation (Fig. 2.4) Both relations are modified versions of relations TRAINING_1 (Fig. 2.1) and

DEPT_1 (Fig. 2.2) respectively. Relation TRAINING_2 has the following scheme: TRAINING_2 = COMPANY TRAINER(TRN C (CN Y)). Relation DEPT_2 has the following scheme: DEPT_2 = D DN UNIT(UN UD C(CN Y)).

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	1	75
		2	76
	Mark	1	82
		3	82
		2	79
IBM	Tim	3	82
		5	79
		4	82
Microsoft	Karen	2	77
		2	81

Fig. 2.3: TRAINING_2

D	DN	UNIT			
		UN	UD	C	
				CN	Y
1	Research	511	Software Engineering	1	75
				2	76
				5	79
		552	Basic Research	1	82
				2	79
		678	Planning	2	76
				4	82
2	Development	650	Design	1	75
				2	77
		780	Maintenance	3	82
		981	Planning	2	81
				3	82

Fig. 2.4: DEPT_2

Suppose that we want to find the names of the trainers and the departments in which they have taught at least one course. The two relations have to be joined on attribute C which is a subrelation in both relations. Therefore, there are cases (although complicated) where the join operation between common subrelations not at the top level of two different relations should be performed and is really meaningful.

Colby also demonstrates the equivalence of the recursive and the non-recursive algebras. Finally, some limitations of the model are briefly discussed.

These limitations refer to the weakness of the presented recursive algebra to support arbitrary algebraic expressions in lists (select lists, project lists etc.) of the operators, such as comparisons of values of compatible attributes situated at different nesting levels in a relation. As a result, there are still cases where the use of nest and unnest operators is necessary.

2.2.4 Deshpande and Larson's model

An algebra for nested relations is presented in [DL91] which is an improved version of the one proposed by Schek and Scholl ([SS86]). Non-recursive union, difference, select, cartesian product, project, subrelation constructor, rename and PNF-Transformer operators are defined first.

Operations on subrelations at any level take place with the aid of a new operator called the subrelation constructor. The subrelation constructor can transform the interior of a nested relation.

The PNF-Transformer operator transforms recursively a nested relation into a nested relation in Partitioned Normal Form (PNF). The select operator is extended to include set comparisons and relational algebra expressions over the relation-valued attributes. However, comparisons can only take place if the attributes that participate in the selection predicate are in the *scope* of the operand relation. As it is defined in [DL91], the scope of a pathname consists of all the other attributes which are "seen" as one traverses the path starting at the root of the scheme diagram of R and ending at the subrelation identified by the pathname.

With the project operator, if the project list contains two or more relation-valued attributes, the cartesian product of their instances is computed in the same tuple firstly and then the result relation is formed.

In addition, set operators, project, join, nest and unnest are defined recursively. These operators preserve PNF. Consequently, they inherite the limitations discussed in section 2.2.2.

Null values are also supported.

Deshpande and Larson ([DL91]) briefly present a join operation for nested relations. It is defined formally using a non-recursive cartesian product operation for nested relations.

Aggregate functions are included in their algebra, as opposed to all the other proposed models.

In order to keep the proposed algebra as simple as possible, they make some assumptions which restrict the expressive power of the model.

Overall, it is this author's opinion that this approach is incomplete since not all the different cases are taken into consideration under the pretext of simplicity, as has been explained above, as for example for the cases of the selection and join operations. In addition, the subrelation constructor overcomes some problems caused by the fact that the operators are not recursive, at the cost that occasionally this operator has to be invoked one or more times in the formulation of queries. Clearly, this invocation increases the execution time to answer queries.

2.2.5 Levene's model

Levene in [Lev92] presents the nested Universal Relation Model (nested UR model) which forms an extension of the classical UR model to nested relations. Levene claims that the nested UR model provides logical data independence, since users can view the nested database as if it was composed of a single nested relation. Null values are also taken into consideration in the formalised proposed model. The nested UR model is defined using the tools provided by the null extended nested relational model which was defined earlier in [Lev92].

For the null extended nested relational model, null extended domains and null extended relations are defined. A typical nested relation in the nested model proposed by Levene can be seen in Fig. 2.6, over the nested relation scheme (NRS) of Fig. 2.7. The nested relation TRAINING* corresponds to nested relation TRAINING_3 of Fig. 2.5.

Note: In NRS, the names of the relation-valued attributes, like the names of the nested relations, are followed by *.

COMPANY	TRAINER		
	TRN	CODE′	DEPT_DIVISION
		CODE	
Apple	Jack	xx0	Administration
	Mark	xy1 xy2	Development
IBM	Tim	xy1 xx2	Customer Services
Microsoft	Karen	xx1	Technical Support

Fig. 2.5: TRAINING_3

COMPANY	(TR? (CODE)* DEPT_DIVISION)*		
	TRN	(CODE)*	DEPT_DIVISION
		CODE	
Apple	Jack	xx0	Administration
	Mark	xy1 xy2	Development
IBM	Tim	xy1 xx2	Customer Services
Microsoft	Karen	xx1	Technical Support

Fig. 2.6: TRAINING* relation in Levene's model
(corresponds to relation TRAINING_3 of Fig. 2.5)

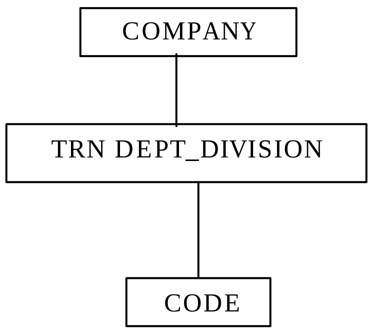

Fig. 2.7: The scheme tree of relation TRAINING*

A null extended algebra for the null extended nested relational model is defined. One of the main features that the null extended algebra provides, is the fact that the user does not need to know the structure of the nested relations in order to express a query in that algebra. All the basic operators of the algebra are defined extensively. Additionally, three operators, the null extended join, the null extended outer join and the null extended total projection are defined as extended versions of the corresponding operators of the UR model to nested relations.

The null extended join is more general than other previously defined joins for nested relations since it supports null values. However, according to the definition, not all relations can be joined together. In order to perform a join between two relations, the two relations must be *joinable* (*joinable* relations in Levene's model are a generalisation of *compatible formats* (see [AB86]), since they are not restricted to have the same attributes in their root nodes). If not, two restructuring operations must be applied, namely empty node insertion and root weighting, which transform the schemes of the relations to joinable NRSs. Clearly, restructuring operations are expensive operations in terms of optimisation and aggravate the performance of already computationally complex join operations in nested relations. Additionally, as shown in chapter 4 of this thesis where the NRM is proposed, the join operation defined there can be applied to all nested relations having one or more common attributes, without the need of applying a restructuring operation prior to performing the join (see also section 2.2.1). Therefore, the use of restructuring operations in the definition of the join operation is estimated to be a limitation in Levene's model.

Levene also examines null extended data dependencies and more specifically null functional dependencies, null extended functional dependencies and null extended join dependencies. Following this, null extended lossless decomposition and the extended chase procedure for nested relations are also defined. Finally, the special case when the nested database consists of a single nested relation is investigated.

Lastly, the motivation of [Lev92] is the solution of the problem of incomplete information; therefore, [Lev92] has to be evaluated from this point of view. Further, even the problem of defining the join operation of two nested relations is solved with the help of the insertion of empty nodes. The author of the present thesis estimates that this approach could be avoided.

2.2.6 Liu, Ramamohanarao and Chirathamjaree's model

In [LR94a] and [LR94b], Liu and Ramamohanarao present an algebra for nested relations. The definitions of selection, projection and intersection are extended to support nested relations. A natural join is defined for cases where the two relations have two common nested attributes at the top level. An extended cartesian product is defined which combines two relational operands with common higher-order attributes not only at the top level but also at the subschema levels. In addition, a path-dependent cartesian product is defined as an extension of the extended cartesian product, which is defined at the interior of the subschemes that are specified in the given paths. A new join operator, called P-join, is introduced as a combination of the extended natural join and a recursive join. Two versions of the P-join are distinguished: the P-join operator with a single join path and the P-join operator with multiple join paths. These definitions are defined as a combination of three other nested relation operations: cartesian product, selection and projection. In their approach two nodes are *selection-comparable* when one of the two nodes is a child of an ancestor of the other node. The P-join operation can be performed only when the common attributes appear on selection-comparable nodes in the scheme tree of the relation which results from the execution of the cartesian product operation between the two relations which participate in the join operation (P-join condition). Therefore, the functionality of the join operation is very limited. Specifically, according to their Decomposition P-join definition (which is a different name for the P-join operator with multiple join-paths) firstly, the two schemes must be decomposed into pairs of subschemes that satisfy the P-join condition. Then, the P-join operator is applied to each pair of subschemes that contain the common attributes and finally, the natural join is performed on the result relations. However, the subschemes of the same relation need to contain at least one common atomic attribute at their top level. Therefore, relations containing only nested attributes at the top level cannot be joined, even when they have one or more common attributes, as shown in the following example.

Example 2.3: Relations R and Q in Fig. 2.8 have been borrowed from [LR94a], where a join operation is illustrated. Now, it is noticed that if the two relations do not contain any atomic attribute at the top level of their schemes, i.e. the attributes A and I from relations R and Q respectively, then, the join operation cannot be performed since it is not possible to decompose each relation such that the intersection of all subschemes contains at least one common atomic attribute at the top level to fulfil the condition that must hold according to the formal definition of the Decomposition P-join operation.

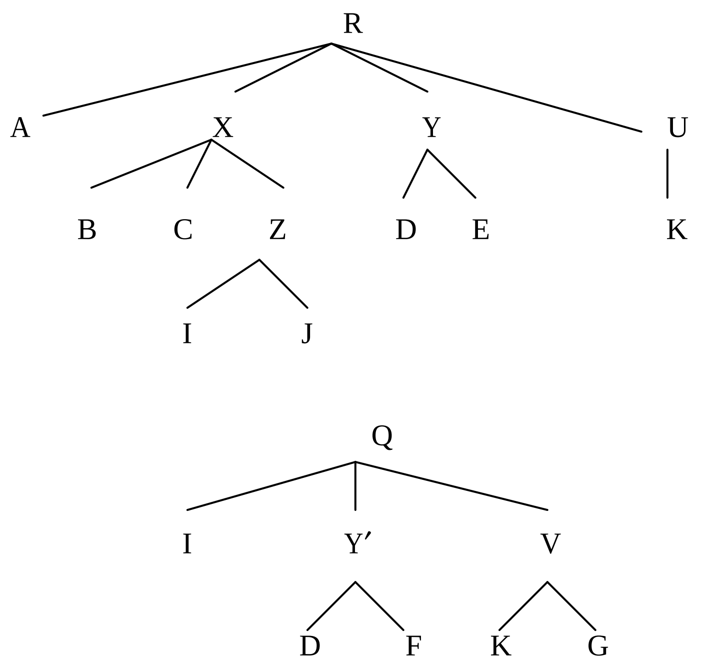

Fig. 2.8: Schemes of relations R and Q as can be found in [LR94a]

Liu and Chirathamjaree in [LC96] revise the P-join operator proposed in [LR94a] and present an algorithm that computes the P-join operator. They also evaluate its estimated cost.

In the opinion of the author of this thesis, the approach, whose main motivation is to give a more efficient and powerful definition of the join operation for nested relations, provides a restricted and complicated approach to the problem. The conditions that must be satisfied: i) selection-comparable nodes applied to the common attributes and ii) atomic attributes at the top levels in both relations that participate in the join, as well as the fact that the case when the common attributes are nested attributes either at the top or lower levels of the two relations, whether the same or different, is not discussed, provide a restricted way of performing the join operation.

2.3 Temporal Models

Current approaches to the management of temporal data can be categorised by reference to the following characteristics:

1) The time at which a piece of data is estimated to be true in the real world is called valid time. A relation, in which both data and its time of validity is recorded, is called a **historical relation**. It has been the subject of extensive research principally by [Cli82], [CC87], [Gad88], [LJ88b], [TG89] and [Sar90]. The time at which a piece of data is stored in a relation is called transaction time. If both this data and its transaction time are recorded in a relation then this relation is called a **rollback relation** ([SA86]).

If both valid and transaction times are recorded in a relation, the relation is called a **bitemporal relation** ([Sno87], [MS91], [Gad92]) (for more about valid time and transaction time see section 3.3.1).

2) Temporal database proposals are characterised by the alternative of associating *timestamps* with tuples or with individual attributes. Therefore, temporal models are divided into two categories: **tuple timestamping** and **attribute timestamping**. In tuple timestamping ([Ahn93], [Sar90]) each tuple is augmented by one or two attributes for the recording of timestamps. One additional attribute can be used to record either the time point at which the tuple becomes valid or the time at which the data is valid. Two additional attributes are used to record the start and stop time points of the corresponding time interval of validity of the corresponding data. This is discussed further below, in point 4 of this section.

The alternative is attribute timestamping ([Tan86], [Gad92]), when the time is associated with every attribute which is time-varying. Note that it is not necessary for every attribute to be time-varying in an attribute timestamping approach. Consequently, a history is formed for each time-varying attribute within each tuple. As a result, the degree of the relation is reduced by one or two compared with the tuple timestamping equivalent relation since timestamps are part of the attribute values (for more about tuple timestamping and attribute timestamping relations see section 3.3.2).

Temporal relations can also be divided into **1NF relations** ([Sar90]) and **N1NF relations** ([TG89]). Commonly tuple timestamping relations are 1NF, whereas attribute timestamping relations can form N1NF relations, or using different words, nested relations.

In Fig. 2.9 a relation is represented in tuple timestamping format while in Fig. 2.10 the same data is represented using attribute timestamping format.

COMPANY	TRN	CN	PERIOD
Apple	Jack	5.2	[2/11/1994, 25/4/1995) ∪ [7/8/96, 1/1/2010)
Apple	Mark	3.3	[2/1/1992, 8/11/1996)
Apple	Mark	3.5	[30/4/1995, 1/1/2010)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
IBM	Tim	5.0	[17/12/1995, 1/1/2010)
Microsoft	Karen	3.3	[25/6/1996, 1/1/2010)

Fig. 2.9: Relation T_TRAINING in tuple timestamping format

COMPANY	TRN	CN
Apple [2/1/1992, 1/1/2010)	Jack [2/11/1994, 25/4/1995) ∪ [7/8/1996, 1/1/2010) Mark [2/1/1992, 1/1/2010)	5.2 [2/11/1994, 25/4/1995) ∪ [7/8/1996, 1/1/2010) 3.3 [2/1/1992, 8/11/1996) 3.5 [30/4/1995, 1/1/2010)
IBM [17/12/1995, 1/1/2010)	Tim [17/12/1995, 1/1/2010)	5.2 [19/3/1997, 21/4/1997) 5.0 [17/12/1995, 1/1/2010)
Microsoft [25/6/1996, 1/1/2010)	Karen [25/6/1996, 1/1/2010)	3.3 [25/6/1996, 1/1/2010)

Fig. 2.10: Relation T_TRAINING in attribute timestamping format

3) Temporal database systems can be homogeneous or heterogeneous. A **temporally homogeneous** ([Gad88]) database is a database which is restricted to having temporal relations in which the lifespans of all attribute values – i.e. the time over which they are defined- in every tuple are identical. Models that employ tuple timestamping rather than attribute timestamping are necessarily temporally homogeneous since only temporally homogeneous relations are possible. It is obvious that homogeneous tuples are a subclass of heterogeneous tuples where the attributes in each tuple are defined over the same time period.

In a database with a **temporally heterogeneous** relational data model the lifespans of the attribute values in each tuple can be different ([JJ92]).

4) Temporal databases can also be characterised by the way time is expressed. Two basic approaches have been proposed. The first is to record time at two attributes, Start and Stop, which represent the **boundary points** of each attribute or tuple's interval of validity ([Ben82], [Sno87]).

In the second approach, time is recorded in just one attribute as an **interval** ([Lor88]).

Fig. 2.9 shows a relation where time is represented in terms of union of time intervals (PERIOD attribute) in contrast to Fig. 2.11 which uses start and stop points to represent the same time data.

COMPANY	TRN	CN	START	STOP
Apple	Jack	5.2	2/11/1994	25/4/1995
Apple	Jack	5.2	7/8/1996	1/1/2010
Apple	Mark	3.3	2/1/1992	8/11/1996
Apple	Mark	3.5	30/4/1995	1/1/2010
IBM	Tim	5.2	19/3/1997	21/4/1997
IBM	Tim	5.0	17/12/1995	1/1/2010
Microsoft	Karen	3.3	25/6/1996	1/1/2010

Fig. 2.11: Relation T_TRAINING where time is represented as two attributes, **START** and **STOP**

In the relation T_TRAINING in Fig. 2.11 stop points correspond to the end points of the intervals in PERIOD attribute of the relation in Fig. 2.9.

In the next sections, various temporal database models are presented and discussed. However, the discussion is restricted only to those models that support valid time because transation time is beyond the objective of the present thesis.

2.3.1 Tansel's model

Tansel has produced many papers over the last 15 years, presenting new temporal database models. His interest is focused particularly on the area of N1NF attribute timestamping models.

In his first model ([CT85], [Tan86], [Tan87] and [TAO89]), N1NF relations with a maximum of one nesting level are supported. A relation can contain four different types of attributes: atomic, set-valued, triplet-valued or set triplet-valued attributes. In the last two types of attributes, attribute values are stored along with either time points at which these values are obtained or time intervals over which these values are valid. Pack, unpack, triplet-formation and triplet-decomposition operations are defined to manipulate historical relations and together with the basic operations of the relational algebra form the elementary operations of the new proposed historical relational algebra. In addition, drop-time and slice operations are defined in terms of the elementary operations. The new operations transform one attribute type to another or apply a version of the selection operation to the time domain of a time-varying attribute respectively. Historical relations are first *normalised* and then algebraic and calculus operations are applied ([Tan86]).

Aggregation operations for historical relational databases are discussed in [TA86] and [Tan87]. The aggregate formation operation, which has been defined by Klug in [Klu82] for N1NF relations, is also used by Tansel. The aggregate formation operation partitions the relation so that all tuples in the same partition have the same value for a specific attribute and then, the aggregate function is applied to each of these partitions. A new operation is defined, named enumeration, which returns the relation instance of a historical relation at a specified time period. Aggregate functions can be directly applied to the result of the enumeration operation.

In [TG89] a temporal database model supporting nested relations is informally defined. The unnest, nest, projection, union, difference, intersection, cartesian product, selection, join, slice and transfer-time operations of the historical relational algebra are briefly described but formal definitions are not given for any of these operations. Structuring of nested historical relations is reviewed and the equivalence between attribute timestamping and tuple timestamping relations is discussed. Although in [TG89] the model supports the general case, where nested relations can have arbitrary levels of nesting, it is presented in a very informal way.

In [TT97] a non-homogeneous, N1NF model is presented where only one level of nesting is allowed. A nested relational tuple calculus, called NTC, is defined. NTC is compared to other temporal query languages in order to show the ascendancy of its expressive power over the other languages.

His most recently proposed temporal database model is presented in [Tan97]. This model is the most complete published so far and eliminates some of the shortcomings that previous versions presented by Tansel had suffered from and for this reason it is now discussed in detail.

Tansel in [Tan97] proposes a temporal, attribute timestamping relational data model where N1NF relations are allowed and only valid time is involved. Relations in the model can have arbitrarily many levels of nesting. Non-homogeneous relations are also allowed in the model. Time is attached to every time-varying attribute forming temporal atoms with the corresponding temporal data values. Therefore, temporal atoms are ordered pairs of the form <t, v>, where v is an attribute value and t is either a temporal set or a time interval and it denotes the time period t for which v is valid. Fig. 2.12 shows relation T_TRAINING in Tansel’s model representation.

COMPANY	TRAINER	
	TRN	CN-H
		CN
Apple	Jack	<{[2/11/1994, 25/4/1995) ∪ [7/8/1996, 1/1/2010)}, 5.2>
	Mark	<{[2/1/1992, 8/11/1996)}, 3.3> <{[30/4/1995, 1/1/2010)}, 3.5>
IBM	Tim	<{[19/3/1997, 21/4/1997)}, 5.2> <{[17/12/1995, 1/1/2010)}, 5.0>
Microsoft	Karen	<{[25/6/1996, 1/1/2010)}, 3.3>

Fig. 2.12: Relation T_TRAINING in Tansel’s model

In [Tan97] a temporal relational calculus which has been previously proposed in [TT97] is described and a temporal relational algebra is presented. Formal definitions for the selection, unnesting, nesting, temporal atom decomposition, temporal atom formation, slice, drop-time and diagonalisation operations are given.

A temporal atom decomposition operation creates a new relation from the original relation, with its degree increased by one, where one attribute containing temporal atoms is split into two attributes, one containing the temporal data and the other the corresponding attribute values.

Temporal atom formation is exactly the opposite operation, where the degree of the result relation is reduced by one compared to the original relation and a new attribute is created consisting of temporal atoms by combining two different attributes of the original relation, one consisting of atomic attribute values and the other of temporal sets.

The slice operation is a modified version of the slice operation that has been defined in [CT85]. It creates a new relation from the original relation, where the temporal set part of one attribute is combined with the temporal set part of another attribute by computing their union, intersection or difference.

The drop-time operation, as the name denotes, derives a new relation from the original one, where the temporal set part of an attribute containing temporal atoms is “dropped”. Thus, the specific attribute is converted from a temporal one to a static one.

The diagonalisation operation returns a new relation consisting of two copies of the original relation, one next to the other, so that one copy serves as a key to the other copy.

Definitions for the union, intersection, difference, projection and cartesian product operations are not given, since they are exactly the same as in the relational algebra. Tansel in [Tan97] claims that the definition of the join operation can be derived from other elementary operations of the temporal relational algebra, as in the case of the traditional relational algebra. For this reason the join definition is omitted from [Tan97]. However, in the opinion of the author of this thesis this is a major omission from the presented model, since the join operation for temporal attribute timestamping N1NF relations is much more complicated than the traditional join operation. It requires a detailed study for a formal and complete definition in order to cover all the different cases that can arise in connection with the type of the common attributes that participate in the join operation and the nesting levels at which the common attributes are found (see section 5.3.11).

The equivalence of the temporal relational calculus and the temporal relational algebra is also proved.

In addition, two operations transforming the structure of the nested temporal relations are defined, named branch unnest and branch nest operations. The two operations convert nested temporal relations to 1NF relations and vice versa. Tansel claims that the two operations are inverses of each other.

The collapse operation is another operation defined in [Tan97]. This operation has been introduced to solve the problem of producing weak relations. The collapse operation merges tuples that are the same, after applying drop-time operations to all the attributes of the relation that consist of temporal atoms. The temporal sets of these tuples are computed by taking the union of the temporal sets of the merged tuples. As a result, weak relations are transformed to standard relations.

Collapsed versions of the set operations are also defined. More specifically, notations for collapsed union, collapsed intersection and collapsed difference are given. These operations apply set operations on the temporal set components of tuples in the operand relations in case they have the same static tuple. These operations produce standard nested temporal relations only when they are applied to standard nested temporal relations. Otherwise, they produce weak relations. Besides, the operand relation must contain at least an atomic attribute at the top level to play the role of a key since these operations are intended to work on the whole data of an object.

Collapse, slice and drop-time operations are redundant operations, since they can be derived from other basic operations; however, they are included in [Tan97] since they are convenient and useful.

Aggregate functions are not included in [Tan97]. Moreover, Tansel does not provide a detailed presentation of predicates for the manipulation of temporal data.

Temporal query languages for the nested temporal relations, implementation issues and query investigation are not included in [Tan97].

Overall, in the author's opinion, the model is the most complete attribute timestamping model presented. Undoubtedly, the researcher has a wide experience of the subject, since his research started formally more than 15 years ago and he is one of the pioneers in defining attribute timestamping models. However, the model has some shortcomings which have been mentioned above. A fuller evaluation of the features of this model is given in chapter 8.

2.3.2 Gadia's model

Gadia has a parallel activity with Tansel. His first paper about temporal databases ([GV85]) appears at about the same time as Tansel's ([CT85]). Since then, he has written more than one dozen of papers concerning a new relational temporal database model and a query language.

Gadia emphasises the homogeneity property. In all his papers N1NF attribute timestamping relations are used.

In [GV85] a query language, called HTQUEL, is presented for a temporal model proposed by Gadia in [Gad88] (it appeared in the literature 3 years later). His homogeneous temporal relational database model is defined as a temporal parameterisation of static relations. Attribute values are represented as single valued functions of time. The temporal element is the basic data type to model time in his approach. He also introduces the notion of a temporal assignment to express the changing value of an attribute with time. According to his definition, a temporal assignment to an attribute is a function from a temporal element into the domain of that attribute. Gadia's temporal assignment corresponds to Tansel's set-triplet-valued assignment. Key attribute values in a tuple of a relation in his model do not change with time (see also [Gad92]). The author of this thesis believes that this is a significant limitation of this particular model since there are cases in the real world to be modelled where key attributes change with time. For an example of a case where the key attribute of a relation is time-varying see Fig. 2.13. Semantically, the relation in Fig. 2.13 shows the addresses of a number of people. The first two tuples of the relation represent the same person. However, the change of the name is due to the fact that this person got married and so her name changed. Therefore, the key attribute, NAME, is a time-varying attribute.

NAME	ADDRESS
Anna Black [d1, d6)	52, Ladbroke Grove [d1, d3) 11, Homer Street [d3, d6)
Anna Scott [d6, d10)	34, Regent Square [d6, d10)
Tom Thomas [d3, d10)	20, Holland Park [d3, d10)

Fig. 2.13: A relation where the key attribute (NAME) changes with time

An example of a relation in Gadia's model is shown in Fig. 2.14. It must be noted that future time is not supported in Gadia's model since he assumes in [GN98] that "the universe of time consists of an interval [0, NOW] of instants with a linear order ≤ on it. NOW denotes the current instant of time". Therefore, the future time point '31/12/2009' is replaced by 'NOW' in Fig. 2.14.

COMPANY	TRN	CN
[2/1/1992, NOW] Apple	[2/11/1994, 24/4/1995] ∪ [7/8/1996, NOW] Jack [2/1/1992, NOW] Mark	[2/11/1994, 24/4/1995] ∪ [7/8/1996, NOW] 5.2 [2/1/1992, 7/11/1996] 3.3 [30/4/1995, NOW] 3.5
[17/12/1995, NOW] IBM	[17/12/1995, NOW] Tim	[19/3/1997, 20/4/1997] 5.2 [17/12/95, NOW] 5.0
[25/6/1996, NOW] Microsoft	[25/6/1996, NOW] Karen	[25/6/1996, NOW] 3.3

Fig. 2.14: Relation T_TRAINING in Gadia's model

A relational algebra and a tuple calculus are presented for this model ([Gad88]) and their equivalence is proved. All the relational algebraic expressions are defined recursively.

The notion of weakly equal relations, which has been introduced in [Gad86b], is used throughout his papers. This definition is based on the fact that when two temporal relations have the same snapshots then, in a way, their information content is the same. Snapshots are very important in Gadia's model, since temporal databases are considered as time-varying static databases. Therefore, snapshots are considered as basic building blocks to his model and are used to extend properties of static relations to their temporal counterparts. However, it is arguable that the concept of snapshots may cause a wrong impression by suggesting that there is a certain implicit structure in every temporal relation.

A temporal selection operation is introduced in [Gad86b]. It is the natural restriction of a relation to a temporal element. Temporal selection is similar to the slice operation introduced by Tansel in [TG89].

As mentioned above, Gadia's model is a homogeneous model. His basic argument for the homogeneity requirement is that the snapshot relation of a homogeneous relation is a static relation without nulls. However, this is only the case when the corresponding snapshots relations are in 1NF. When N1NF relations are supported nulls can be omitted from snapshots naturally. Gadia admits that the homogeneity assumption causes many problems. He mentions in [Gad86a]: "The homogeneity requirement is a severe restriction in modelling real life situation". In addition, in a homogeneous model, the cross product operation is a limited version operation since it can be applied only to homogeneous tuples. For that reason, he introduces the multihomogeneity assumption ([Gad86a]), where a relation consists of a finite set of schemes and each tuple is homogeneous in each of these schemes. Gadia claims that multihomogeneous models can model a significant part of the real world. However, it is undoubtedly true that multihomogeneity, although it is more powerful than homogeneity, is more restricted than heterogeneity since the latter forms the general case and can model the real world without any limitation.

The temporal database model presented in [Gad88] is generalised in [GY88] to support N orthogonal temporal dimensions. A discussion is also given about how to give a precise classification of errors and updates. A kind of restructuring operation is also introduced, which changes the key of a relation to create a new relation weakly equal to the one from which it is derived.

In [GY91], a N1NF tuple calculus is introduced, called TCAL, based on Gadia's N1NF homogeneous temporal model. TCAL is compared to the 1NF tuple calculus TQuel, which has been proposed by Snodgrass in [Sno87]. Gadia argues that TCAL is more powerful than TQuel.

Gadia in [GNP92] describes informally the restructuring, union, difference, projection, selection and cross product operations. However, the most important and also most complicated operation, the join operation, has not been studied in detail, in any of his papers. Besides, in [GNP92], an incomplete model is introduced as a generalisation of the complete information temporal database model which has been presented in [GY88].

In his recent paper ([GN98]), Gadia uses the temporal model that was defined in his earlier papers and which has been described above, to discuss algebraic identities and query optimisation. The model is called a parametric model since databases in it can be viewed as a parametrisation of classical databases. In [GN98], homogeneous relations are divided into two categories: unihomogenous and multihomogeneous relations. Unihomogeneity is when the parametrisation of classical databases is with respect to a single time line and multihomogeneity with respect to more than one time line. Gadia's model is unihomogeneous from that point of view. The relational operators for the model are described but not all of them are formally defined. The projection operation needs special attention, since there are two cases: the internal projection and the user projection, depending on whether the resulting projected relation contains a key or not. This is a consequence of the fact that in the parametric model keys play a critical role since a user thinks in terms of relations that have keys. In addition, the natural join is briefly described for the first time. With regard to the cross product operation only a restricted version is discussed, the unihomogeneous cross product.

Future time is not taken into consideration in any of his papers.

As mentioned above, in Gadia's model relations are in N1NF. Relations having more than one nesting level are not discussed at all, in any of his papers.

Generally, the model proposed by Gadia provides a limited representation capability and lacks flexibility for the following reasons:

- only one nesting level is supported,
- it is homogeneous,
- null values are not supported,
- relational operators are not formalised,
- the join operation is not defined.

Therefore, in the author's opinion it is incomplete.

2.3.3 Clifford's model

Clifford was another pioneer in the area of temporal databases and the first to suggest incorporating the time dimension at the attribute level. Unfortunately, his sudden death signified the end of an important line of research in this field.

His first paper appeared in 1982 ([Cli82]). In [CW83], a formal theory (the Historical Database Model-HDBM) of database semantics that includes time is developed as well as a calculus-based query language. The formulation of an intensional logic is used for this purpose. The tuples of relations are timestamped with the help of a new attribute named STATE. In addition, a special Boolean-valued attribute, EXISTS?, is introduced to indicate which entities exist or not at any given state. The two new attributes are not ordinary attributes, but are built-in parts of the model.

HDBM is explored further in [CT85] but this time from the operational point of view using a relational algebra. The temporal dimension is incorporated into the model at the attribute level. Relations are in N1NF since attributes that are time-varying have complex domains. Attributes can be either constants, time-varying or temporal. However, key attributes in a relation must be constant. In addition, each relation has a lifespan related to it. The lifespan represents the time period over which the objects are being modelled in the relation. An example relation in Clifford's model is given in Fig. 2.15.

LAWYER	STUDIO	SALARY	*lifespan*
Howell	1924 MGM	1924 30000	{[1924, 1939]}
	1930 Paramount	1925 35000	
	1939 MGM	1926 ?	
		1937 40000	
		1938 ?	
Rosen	1912 Universal	1945 70000	{[1912, 1952]}
	1923 Warner Br	1946 ?	
	1930 ?	1952 80000	
	1945 Rko		
Mcmanus	1923 Warner Br	1923 35000	{[1923, 1926]}
		1924 ?	
		1926 40000	

Fig. 2.15: Relation LAWYERS in Clifford's model ([CC87])

A discussion about various problems that arise when trying to define relational operations for historical databases is presented. More specifically, projection, selection, time-slice, join and when operations are discussed and examples are presented where the problems are demonstrated.

Formal definitions of these operations have been provided in [CC87]. Four different kinds of join are examined: T-join, equijoin, natural-join and time-join. All attributes are functions from time points to simple domains with the assignment of a lifespan to each attribute. Besides the lifespan of each attribute, each tuple is assigned a lifespan, too. Union, intersection and difference are defined over *merge-compatible* relations. Two relations are merge-compatible when they are union-compatible and at the same time, they have the same key. Depending on whether a historical relation is to be reduced to the value or the time dimension, two versions of the selection operation are defined, the select-if and the select-when operations respectively. Likewise, time-slice has two variations, static and dynamic. The static version returns a relation reduced to tuples consisting of those parts (of the tuples) defined over a specified lifespan. In contrast, in the dynamic version, lifespans for each tuple are not specified but are determined by the set of times that a specified attribute for that tuple maps to. The when operation returns the set of times during which a specific condition is satisfied in a given relation.

Clifford argues that tuple and attribute lifespans provide time-varying data and schemes in the model and a "suitable level of user control over the temporal dimension of the data". However, in the author's opinion, this model generates many problems. Assume that it is intended to represent the data of relation T_TRAINING (Fig. 2.9) in Clifford's model. After careful consideration it can be seen that this is not feasible, since in his model, all values of the attributes in a relation are viewed as functions from time points to simple domains. Therefore, since in this particular example each company may have employed more than one trainer at the same time and since each trainer may teach more than one course for overlapping periods of time (which is a logical assumption), the entire history of each trainer could not have been recorded in just one tuple but, on the contrary, can only be represented by a series of tuples. In the general case, since attribute values are functions from a lifespan on to a value domain, Clifford's model is effective only when there is a one-to-one relationship between the key attributes' values and each other attribute value for the same time in each tuple. This contradicts one of the main reasons for representing data in a relation using attribute timestamping approach, since data related to a single object is not represented in the same tuple but on the contrary, is split into different tuples. Consequently, relation T_TRAINING could be expressed in Clifford's model but in a way that is impractical and to no real benefit. To demonstrate this, part of the T_TRAINING relation represented in Clifford's model is shown in Fig. 2.16, where the problems that appear with this kind of representation are obvious. In addition, the key of such a relation is not easy to define, since key attributes in HRDM must be constant-valued. Even if it is assumed that the COMPANY attribute is constant-valued, the key for the relation could not be that attribute, because of the previous comment that the entire history of an object (in this specific case, the COMPANY object) would be split up in many tuples (see Fig. 2.16).

COMPANY	**TRN**	**CN**	***lifespan***
Apple	2/11/1994 Jack 25/4/1995 ? 7/8/1996 Jack	2/11/1994 5.2 25/4/1995 ? 7/8/1996 5.2	{[2/11/1994, 24/4/1995] ∪ [7/8/1996, 31/12/2009]}
Apple	2/1/1992 Mark	2/1/1992 3.3	{[2/1/1992, 7/11/1996]}
Apple	2/1/1992 Mark	30/4/1995 3.5	{[2/1/1992, 31/12/2009]}
⋮	⋮	⋮	⋮

Fig. 2.16: Relation T_TRAINING in Clifford's model

Therefore, HRDM has a limited capability and expressiveness. Also, one contradiction is the fact that, although key attributes must be constant-valued, a lifespan is also assigned to them. In fact, every attribute must have a lifespan associated with it, even if that attribute is constant or temporal, a statement which is also a contradiction.

Clifford's research in temporal databases continues in [CCT94] where two categories of historical database models are defined, temporally grouped and temporally ungrouped models, to distinguish between the two different modelling capabilities achieved by incorporating the temporal dimension at the tuple level or at the attribute level respectively. Temporally grouped models are those models which represent data in groups of related temporal values, while temporally ungrouped models cannot support this kind of grouping. Therefore, temporally ungrouped models are 1NF models; however, temporally grouped models are not fully N1NF models but only in the way they incorporate the temporal dimension. Clifford proposes the corresponding notions of historical relational completeness for each of these two categories.

Clifford shows that temporally ungrouped models can only have the same expressive power as the temporally grouped models if they are extended with a grouping mechanism; otherwise they are less expressive than temporally grouped models. This grouping procedure adds a special attribute to an ungrouped relation. The new attribute assigns a kind of identity to each tuple, which is why it is called a "group-id attribute". Three different languages are defined for the temporally ungrouped models: a temporal logic, a logic with explicit reference to time and a temporal algebra. He relaxes the previous assumption he has made for the HRDM that key attributes must be constant-valued, by assuming that key attributes need not be constant over time. However, the key notion in temporally ungrouped models creates many problems, for example, without knowledge of the key, tuples which describe the same object cannot be grouped together, since there is no way to associate them. In addition, if the key is not required to be constant over all times (and there is no reason to require this), there would be no way at all to group related tuples (i.e., tuples describing the same object) ([CCT94]). For the temporal algebra the following operators are defined: select, project, cartesian product, set difference, union, future linear recursive operator and past linear recursive operator. Future and past linear recursive operators are needed to be able to simulate until/since operators in temporal calculus. The definition of the temporal join operation is omitted since it is said that it can be expressed in terms of cartesian product, select and project operators.

Temporally grouped models include the temporal component in their structures directly. Temporally grouped completeness is defined with the support of the L_h calculus. L_h is reduced to the standard relational calculus when the temporal dimension is not included in the model. The HRDM is a temporally grouped data model. However, it is not temporally grouped complete, since there are queries which cannot be expressed in an equivalent expression of the HRDM algebra. In [CCT94], Clifford shows that temporally grouped and temporally ungrouped models do not provide the same modelling and querying capabilities and that temporally grouped models have supremacy over temporally ungrouped models. He concludes by saying that there is no complete algebra defined for a temporally grouped data model.

Clifford's work for temporally grouped and ungrouped models is developed further in [CCGT95]. An algebra for a temporally grouped inhomogeneous –i.e. where the homogeneity assumption is relaxed- and multisorted –i.e. it allows attribute values of all three sorts: user-defined time, time-invariant and time-variant attributes- model is defined. The standard relational operators, union, difference, cartesian product, projection and selection, are extended to support the temporal dimension. In addition, two active-domain operators, the tdom(R) and vdom(R), which compute the temporal and value domains of a relation R respectively, as well as the timeslice operator which restricts the lifespans of attributes that are functions from time points to simple domains, are defined. Coalescing and restructuring notions are also discussed ([CCGT95]). The coalescing operation merges in one tuple snapshot equivalent tuples. In [CCGT95] it is reported that this operation is meaningless in temporally grouped models. In this author's opinion this is incorrect, since there are cases, especially after the execution of an operation or a series of operations, where coalescing is necessary in order to "coalesce" tuples into a single tuple.

A regrouping operation for temporally grouped relations is also discussed. An important comment is made in [CCGT95], that the regrouping operation from one attribute to another is possible only when each one functionally determines the other. It is also said that regrouping is not a useful operation for temporally grouped relations and that is why it is not formally defined. However, in the opinion of the author of this thesis, there are cases where the regrouping operation is needed, as can be seen in chapter 6 of this thesis (e.g. Query 17).

Finally, in [CCT96] several examples are given in order to demonstrate the differences between temporally grouped and temporally ungrouped models when updating and querying data. This paper also repeats the claim for the supremacy of temporally grouped models over temporally ungrouped models.

2.3.4 McKenzie's model

McKenzie extends the relational algebra to support both temporal dimensions, valid time and transaction time ([Mck88]). In this review, only valid time will be discussed, since transaction time is not examined in this thesis and it has been proved by many researchers that the two aspects of time are orthogonal and can be studied separately.

McKenzie defines a historical algebra supporting valid time by extending the snapshot algebra. The design decisions made in order to define this algebra can be summarised to the following points:

- valid time is associated with attributes,
- valid time is represented as a set of chronons,
- attributes' value parts must be atomic-valued,
- attributes' valid time parts may be set-valued (relations are not necessarily in 1NF),
- timestamps of attributes in a given tuple may be different (non-homogeneous relations),
- traditional relational operators have been extended to support valid time directly,
- temporal nulls are allowed for some attributes of a given tuple; however, not all attributes of a given tuple can contain temporal nulls simultaneously,
- no two tuples of a given relation can be value equivalent (value-equivalence property).

In Fig. 2.17, an example of a relation in McKenzie's model is shown.

COMPANY	TRN	CN
Apple {2/1/1992, 3/1/1992, ..., 31/12/2009}	Jack {2/11/1994, 3/11/1994, ..., 24/4/1995, 7/8/1996, 8/8/1996, ..., 31/12/2009}	5.2 {2/11/1994, 3/11/1994, ..., 24/4/1995, 7/8/1996, 8/8/1996, ..., 31/12/2009}
Apple {2/1/1992, 3/1/1992, ..., 31/12/2009}	Mark {2/1/1992, 3/1/1992, ..., 31/12/2009}	3.3 {2/1/1992, 3/1/1992, ..., 7/11/1996}
Apple {2/1/1992, 3/1/1992, ..., 31/12/2009}	Mark {2/1/1992, 3/1/1992, ..., 31/12/2009}	3.5 {30/4/1995, 1/5/1995, ..., 31/12/2009}
IBM {17/12/1995, 18/12/1995, ..., 31/12/2009}	Tim {17/12/1995, 18/12/1995, ..., 31/12/2009}	5.2 {19/3/1997, 20/3/1997, ..., 20/4/1997}
IBM {17/12/1995, 18/12/1995, ..., 31/12/2009}	Tim {17/12/1995, 18/12/1995, ..., 31/12/2009}	5.0 {17/12/1995, 18/12/1995, ..., 31/12/2009}
Microsoft {25/6/1996, 26/6/1996, ..., 31/12/2009}	Karen {25/6/1996, 26/6/1996, ..., 31/12/2009}	3.3 {25/6/1996, 26/6/1996, ..., 31/12/2009}

Fig. 2.17: Relation T_TRAINING in McKenzie's model

From the above example it can be easily seen that McKenzie's model suffers from some disadvantages. One of the main weaknesses of his model is the redundancy caused by the value parts of attributes not being set-valued. Another problem is caused by the representation of time as a set of chronons. As shown in Fig. 2.17 where time is represented in days, the listing of all the days of an interval can be inappropriate when time intervals represent a long length of time.

Formal definitions for union, difference, cartesian product, selection and projection operations are provided as simple extensions of the corresponding operations of the snapshot algebra for snapshot relations with the addition of the appropriate treatment of the timestamps of the corresponding attributes. The projection operation has two versions: projection on a subset of attributes (the traditional projection operation) and projection on expressions, where tuples are projected on new attributes. The closure property of the projection operation is maintained with the restriction that tuples having empty valid component for all tuple components are removed. Therefore, the projection operation defined in [Mck88] is not an information nonloss operation.

A new operator is defined, called the historical derivation operator, as a combination of temporal selection and projection on the timestamps of the tuples' attributes. Specifically, for each tuple a new valid time component is calculated for each attribute as a function of specific intervals in the timestamps of the tuples' attributes.

Aggregation and unique aggregation are the two operators defined to compute aggregates, i.e. a summary of data contained in a given relation.

Historical intersection, T-join, historical natural join and quotient are formally defined in terms of the union, difference, cartesian product, selection, projection and derivation operators and examples are given.

The historical algebra is defined so as to satisfy as many of the list of 29 evaluation criteria which are defined in [Mck88]. All but 3 of these criteria are also presented and discussed in detail in [MS91] where it is argued that “they are well defined, have an objective basis for being evaluated and are arguably beneficial”. A more extended discussion of these criteria can be found in chapter 8 of this thesis.

The possibility of extending the algebra to support set-valued attributes is also discussed. McKenzie proposes to use the approach presented in [SS86], where an algebra to support N1NF relations is defined as an extension of the snapshot algebra. The operations can then be recursively defined. However, he has not continued his research in this direction, although he recognises the benefits from such an approach.

2.3.5 Snodgrass’s model

Snodgrass has made a major contribution in the area of temporal databases. Two main points distinguish his work in the field; firstly, he published a number of early papers where his main concern was to define a basic terminology for the field and a taxonomy of time in databases as well as surveys and reports of other temporal query languages and models supporting time-varying data that have been proposed over the years ([SA85], [SA86], [Sno86], [Sno90], [Sno92] and [SJS95]) and secondly, he has presented implementation approaches for time-oriented databases in his recent published book ([Sno00]), where a time-varying database application is developed in SQL.

In [Sno87], Snodgrass proposes TQUEL, a temporal extension of QUEL, which is now briefly discussed. The snapshot relational database model is used for the development of the semantics of TQUEL, since it is simple, well defined and has already been implemented.

The semantics for the TQUEL statements are also presented. TQUEL supports both transaction and valid times. Rollback queries supporting transaction time use the “as of” clause, while historical queries supporting valid time use the “valid” clause. The “valid” clause has two variants, the “valid at” for event relations and the “valid from … to …” for interval relations. The “when” clause of TQUEL is the temporal equivalent of the “where” clause of QUEL.

Both transaction and valid times can be represented as intervals (interval relations). The starting time of the interval is denoted by the “begin of”, while the stopping time is denoted by the “end of” operators. However, relations can be event relations as well. The operators “overlap”, “precede” and “extend” can be used in temporal predicates and expressions contained in the “valid” clause.

Relations can be either snapshot, rollback, historical or temporal. The “persistent” keyword is used for rollback or temporal relations, the “interval” or “event” keywords for historical or temporal relations; otherwise the relation is snapshot.

Finally, the paper compares TQUEL to ten other temporal query languages against 17 properties and is shown to satisfy most of them. These properties are discussed extensively in chapter 8 of this thesis.

TQUEL also supports new aggregates, which are formally defined in [SGM93], as well as all the aggregates supported in QUEL. These new temporal aggregates are: *first*, *last*, *rate*, *var*, *earliest*, *latest* and *rising*. The semantics of these aggregates can be found in [SGM93]. An approach (algorithm) for computing TQUEL aggregates is also given.

A typical relation supported by TQUEL is shown in Fig. 2.18.

COMPANY	TRN	CN	VALID TIME	
			(FROM)	(TO)
Apple	Jack	5.2	2/11/1994	25/4/1995
Apple	Jack	5.2	7/8/1996	1/1/2010
Apple	Mark	3.3	2/1/1992	8/11/1996
Apple	Mark	3.5	30/4/1995	1/1/2010
IBM	Tim	5.2	19/3/1997	21/4/1997
IBM	Tim	5.0	17/12/1995	1/1/2010
Microsoft	Karen	3.3	25/6/1996	1/1/2010

Fig. 2.18: Relation T_TRAINING in Snodgrass’s model (a historical relation)

Snodgrass reports both in [Sno87] and in [Sno92] that TQUEL is based on the temporal algebra proposed in [Mck88], therefore many of the same remarks can be applied as to McKenzie’s model. However, although in McKenzie’s model the valid time parts of attributes can be set-valued, this is not discussed by Snodgrass. Furthermore, since key attributes are not defined in his work, it is unclear if the above representation of the T_TRAINING relation (Fig. 2.18) is correct, since two of the tuples have equal values for all their atomic attributes. Other differences can also be found between McKenzies's temporal algebra and Snodgrass's temporal query language. An example is the association of valid time with attributes and the support of N1NF relations in McKenzie's model; therefore, TQUEL cannot be based on this temporal algebra without some modifications, which are not discussed in the paper.

In summary, Snodgrass appears to be more concerned about the implementation side of temporal databases and so, from the theoretical point of view, his work is not well defined and lacks formalisation.

2.3.6 Jensen and Snodgrass's model

Jensen and Snodgrass have collaborated for a number of years in the area of temporal databases and they have produced a significant number of papers about the understanding of the semantics of temporal data. Jensen's most important publications can also be found in [Jen00].

Their major contribution in this field is the development of a new temporal model, called the Bitemporal Conceptual Model (BCDM) ([JS92]). This new model supports both valid and transaction time. However, it is presented here since valid time relations can be seen as special cases of bitemporal relations supporting only valid time.

Relations in BCDM are considered from a conceptual standpoint rather than from the representational one used with all the other proposed temporal data models. Nevertheless, in [JS96a] it is mentioned that the term "conceptual" does not make the formalism of the new proposed model different from that of the other temporal data models. On the contrary, it is used to underline the design and query language capabilities of the new model. Relations in BCDM use tuple timestamping, since they consist of a set of tuples and each tuple includes an implicit attribute value, comprising an ordered pair of integers, denoting when the fact represented by this specific tuple is true in the modelled reality (valid time) and when it is current in the stored relation (transaction time). Therefore, time is represented in BCDM as temporal elements. Consequently, each single tuple represents the whole history of a fact. Moreover, relations in BCDM are in N1NF, since the timestamps associated with the tuples are sets of time chronons. Obviously, only homogeneous tuples are supported in the model.

Two operators are also defined, named the transaction-timeslice operator and the valid-timeslice operator, which take as arguments a bitemporal relation and either transaction or valid time and return a valid time relation or a transaction time relation, respectively, consisting of all tuples valid during the time value.

The following table (Fig. 2.19) represents the valid time T_TRAINING relation in their model.

COMPANY	TRN	CN	T
Apple	Jack	5.2	{2/11/1994, 3/11/1994, ..., 24/4/1995, 7/8/1996, 8/8/1996, ..., 31/12/2009}
Apple	Mark	3.3	{2/1/1992, 3/1/1992, ..., 7/11/1996}
Apple	Mark	3.5	{30/4/1995, 1/5/1995, ..., 31/12/2009}
IBM	Tim	5.2	{19/3/1997, 20/3/1997, ..., 20/4/1997}
IBM	Tim	5.0	{17/12/1995, 18/12/1995, ..., 31/12/2009}
Microsoft	Karen	3.3	{25/6/1996, 26/6/1996, ..., 31/12/2009}

Fig. 2.19: Relation T_TRAINING in Jensen and Snodgrass's model (valid time relation)

An algebra is also defined at the conceptual level for the BCDM ([JSS92]). Thus, the projection, selection, union, difference and natural join operations are formally defined. Two transformation functions are also defined, named coalescing and elimination of repetition transformations. The coalescing transformation takes value-equivalent tuples with overlapping or adjacent temporal elements and converts them to a single tuple. The elimination of repetition transformation reduces temporally redundant information (a bitemporal relation instance has temporally redundant information if it contains two distinct tuples that are value-equivalent and have timestamps that encode overlapping regions in bitemporal space), possibly at the expense of adding more tuples, since the transformation may partition the region covered by the argument rectangles on either transaction time or valid time ([JS96a]).

Three representations of bitemporal relations are examined which map to and from the conceptual bitemporal relations of BCDM. These are a tuple timestamped 1NF representation scheme, a backlog based representation scheme composed of 1NF timestamped change requests (either insertion requests or deletion requests but never update requests) and a N1NF attribute value timestamped representation scheme.

The BCDM forms the basis for the TSQL (Temporal Structured Query Language) proposal, an extension of SQL. Jensen and Snodgrass claim that the conceptual bitemporal data model that they have proposed is useful when time-varying semantics need to be expressed.

BCDM, although it retains the simplicity of the relational model, is inferior to other temporal proposed models from the representational point of view, because of the large number of timestamps that each tuple contains and the representation of timestamps as bitemporal elements ([JS99]).

An important research contribution is made by Jensen and Snodgrass in the area of dependency theory (temporal normal forms and temporal keys) for temporal databases in terms of BCDM schemas ([JSS94]). However, it is not discussed here since it is beyond the objective of the present thesis.

Furthermore, they examine and categorise temporal relations according to all possible different relative positions between valid and transaction timestamps; they call this “taxonomy of specialized properties of either event or interval temporal relations” ([JS94]).

Finally, in [JS96b] they introduce surrogates to represent real-world entities in the database, lifespans of attributes, derivation functions that compute new values from stored attribute values and observation and update patterns for time-varying attributes. All these notions provide different semantics for time-varying attributes and can be used in the design of database schemas.

To conclude, Jensen and Snodgrass have made an important contribution to the field of temporal databases. However, their proposed model, the BCDM, has shortcomings in its internal representation and in the display of temporal data to users, as discussed above.

2.3.7 Lorentzos’s model

Lorentzos is well known for his significant work on interval data which concerns not only temporal databases but also other areas of databases, such as spatial ([LTR99], [LRT99]) and spatio-temporal databases ([LSYK99], [RLT01], [RL01]), soil information systems ([LK89]), CAD, cartography and engineering. He has studied in depth the notion of generic intervals which form an important component in all the above mentioned applications. Additionally, he has used his results to extend the relational model to support temporal data, since time intervals are one of the many different generic intervals that can be used in databases. For this reason his work is discussed here in some detail.

In [Lor88], Lorentzos presents a detailed and formal extension of the relational model to support generic intervals. Different properties of the intervals are studied. One basic principle is the *Duality Principle* which says that every 1-dimensional point is isomorphic to an elementary interval. He also defines all possible relative positions between two 1-dimensional intervals (see also [LJ88a]) and then, extends his definitions to n-dimensional intervals.

Firstly, he defines the Interval Relational Model (IRM) which supports n-dimensional intervals and afterwards the Interval Extended Relational Model (XRM) which allows both intervals and points to be recorded. Union, difference, projection and cartesian product operations of the XRM are the same as that for the Conventional Relational Model (CRM). In the selection operation new predicates can be used for the comparison of two intervals. The new operations, compute, fold, extend and unfold, that are defined in the XRM are explained further below. The join operation is also formally defined. Another operation introduced in [LM94], called normalise, returns a relation where duplicate tuples are eliminated and adjacent or overlapping intervals are merged into one. It is a combination of unfold and fold operations. XRM is proved to be a proper superset of the CRM. The expressive power of the XRM is also demonstrated by a number of examples. Finally, he examines the new proposed model semantically, by defining the Interval and the Point Functional Dependencies, the key of an XRM relation and two new normal forms, the P and Q normal forms.

The management of 2-d interval relations is also discussed in [LM94] and new efficient algorithms are proposed for the normalisation, insertion and deletion of 2-d interval data. The normalise operation produces canonical interval relations, i.e. normalised representation of interval relations in which there is no duplication of data, as defined in [LPS94]. Additionally, two operations are defined, p-union and p-diff, counterparts to the set-union and set-difference operations of the CRM but which return canonical relations. These operations are then transformed to the optimised s-union and s-diff operations which also maintain the property of canonicity. Formal proofs of this property for the above mentioned operations, p-union, p-diff, normalise, s-union and s-diff, can be found in [LPS95].

In an interesting paper, [LM95], it is shown that some of the known temporal data models, which have been proposed over the years, can be used in other areas than temporal databases by replacing the set of Time values by other sets. This is a very important observation since it means that some particular attributes or parts of attributes can be interpreted equally well either as time or as another type of data. Furthermore, temporal data models are evaluated against some properties which are identified in the same paper for valid time 1NF and N1NF models. Finally, two new models are mentioned, I-1NF and I-NESTED models, supporting interval data management but are not discussed further or formalised.

An extension of SQL, called IXSQL, for the management of interval data is proposed in [LM97], based on the algebra described above for a 1NF model. Generic intervals are used as a new primitive data type. Relations can contain more than one interval attribute. New interval predicates, interval value functions and interval set functions are also introduced and formally defined. The concept of the key of an interval relation is also supported.

His proposed model can be used directly as a temporal model where time is treated as generic data type and not as a “stamp” for the related data values. Lorentzos’s temporal model, called TRA (a model for a Temporal Relational Algebra) ([LJ87], [JL87], [LJ88b]), is presented briefly below. The model is a minimal extension of the Conventional Relational Model. It is simple since 1NF is maintained. Data valid at some specific time either in the past, present or future can be supported in this model and the corresponding time can be represented equally well either as time points or as time intervals.

An example of a relation in Lorentzos’s model is shown in Fig. 2.20.

COMPANY	TRN	CN	PERIOD
Apple	Jack	5.2	[2/11/1994, 25/4/1995)
Apple	Jack	5.2	[7/8/1996, @)
Apple	Mark	3.3	[2/1/1992, 8/11/1996)
Apple	Mark	3.5	[30/4/1995, @)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
IBM	Tim	5.0	[17/12/1995, @)
Microsoft	Karen	3.3	[25/6/1996, @)

Fig. 2.20: Relation T_TRAINING in Lorentzos's model

Four new operations are defined: compute, fold, extend and unfold. Compute is defined so that functions can be used in the new model. Fold returns a relation where time is represented as time intervals from an input relation where time is represented as time points. Extend returns a relation from an initial relation which contains a new attribute consisting of all the time points that are extracted from the time intervals. Unfold is the inverse of the fold operation. Fold and unfold operations should not be confused with the nest and unnest operations of the N1NF models, since fold returns intervals from consecutive points while nest returns sets of attribute values. The cartesian product operation is also discussed in TRA. The result of the cartesian product operation of two time interval relations is a relation having four time attributes. This kind of relation is useful when periodic events need to be supported. An implementation is also briefly described.

Lorentzos's model is simple and more general than the other proposed models since temporal relations are one of the many different types of relations that it can support. Therefore, XRM cannot be considered as a pure temporal database model. However, it can be used for the representation of valid time relations as has been demonstrated.

In Lorentzos's model, attributes are timestamped since more than one time interval attribute can coexist in the same relation referring to different data. However, relations are maintained in 1NF. This causes a number of problems. Firstly, it is not representationally clear with which attributes each timestamp is associated. Secondly, data regarding the same object is not included in the same tuple. Indeed, they are contained in different tuples and, as a result, the history of an object does not consist of a single tuple but is split up into many. As a consequence, even though this specific representation is 1NF attribute timestamping, it does not take full advantage from either the 1NF or the attribute timestamping approach.

Further, fold and unfold operations must be used in temporal selection and temporal projection operations of the algebra and, as a result, these operations can be time and space consuming.

Nevertheless, his interesting results about generic intervals and points can easily be used as the groundwork for any study of temporal database models.

2.3.8 TSQL2

TSQL2 ([Sno95]) is a temporal extension to SQL-92 produced by a research community consisting of twenty-one members of eight different countries. It has been designed to query and manipulate time-varying data stored in a relational database.

The features of TSQL2 are summarised below:

- It is a language defined on a conceptual data model.
- It is based on a tuple timestamping data model.
- It supports three types of time, user-defined, valid and transaction.
- Time can be expressed either as a set of time instants (instant set) or as a union of non-adjacent and non-overlapping periods (elements). Note however, that temporal valid time can be expressed either as set of instants or as temporal elements, but transaction time can be expressed only as temporal elements.
- Six kinds of relations are supported: snapshot relations that support only user-defined time and neither valid nor transaction time, valid time state relations which support only valid time elements, valid time event relations which support only valid time instant sets, transaction time relations that support only transaction time elements, bitemporal state relations which support bitemporal elements and bitemporal event relations which support bitemporal instant sets.
- Valid time and transaction time are recorded in implicit attributes.
- Only one implicit attribute is allowed in a relation.
- TSQL2 is based on homogeneous tuples.
- Value-equivalent tuples, i.e. tuples having identical values for all their explicit attributes, are not allowed in a relation.

It is worth noting that the algebra underlying TSQL2 was defined after the language definition of TSQL2. An example of a relation in TSQL2 is shown in Fig. 2.21.

COMPANY	TRN	CN	V
Apple	Jack	5.2	[2/11/1994, 24/4/1995] ∪ [7/8/1996, 31/12/2009]
Apple	Mark	3.3	[2/1/1992, 7/11/1996]
Apple	Mark	3.5	[30/4/1995, 31/12/2009]
IBM	Tim	5.2	[19/3/1997, 20/4/1997]
IBM	Tim	5.0	[17/12/1995, 31/12/2009]
Microsoft	Karen	3.3	[25/6/1996, 31/12/2009]

Fig. 2.21: Relation T_TRAINING in TSQL2 (valid time state relation)

The scheme of the T_TRAINING relation is: T_TRAINING = {COMPANY, TRN, CN | V}, where attributes COMPANY, TRN, CN are explicit and V is an implicit timestamp attribute distinguished from the explicit attributes by the symbol |. Since V is an implicit attribute that contains valid time elements, T_TRAINING relation is a valid time state relation in TSQL2.

Although time in a tuple is not atomic in general, a 1NF tuple timestamping data model is assumed. This statement is justified in [Sno95] as follows: "The conceptual model and algebra are not meant for physical implementation due to the N1NF nature of the model. We therefore show how the semantics of the conceptual algebra can be supported with a 1NF representational model and accompanying algebra. The 1NF nature of this representation allows the use, or adaptation of, many well-established query optimisation and evaluation techniques". The algebra described briefly below is the conceptual algebra.

Besides, only valid time relations are discussed in this section, since transaction time and bitemporal relations are out of interest in the present thesis.

The snapshot relational algebra is extended to accommodate the TSQL2 characteristics. Six sets of relational algebra operators are defined to support the six different types of relations that TSQL2 supports.

Three new operators that do not have snapshot analogues are introduced in the set of valid time state operators and together with seven operators that are generalisations of the corresponding snapshot operators form the operators of the algebra.

Projection, selection, theta-join, left outer-join, union, difference, rename, $AT^{VS, BS}$ which transforms a valid time state relation into a bitemporal state relation and SN^{VS} which transforms a valid time state relation into a snapshot relation and slice form the set of valid time state operators. The set of valid time event operators consists of the same ten operators. The definitions of valid time state and valid time event operators are almost identical.

Coalescing is the process of collapsing all value-equivalent tuples into a single tuple.

However, the slice operator may produce non-coalesced tuples. Therefore, it violates the restriction that value-equivalent tuples are not allowed in the data model since it can produce value-equivalent tuples. Hence, it is not a closed operation.

The definition of the join operation between a snapshot relation and a valid time state relation is missing in the TSQL2 algebra. Besides, snapshot relations cannot be transformed to valid time relations, due to the lack of relevant operators.

TSQL2 supports only one valid time dimension since only one implicit valid time attribute is allowed in each valid time state or in each valid time event relation. This imposes restrictions to the modelling of temporal data. For example, if two or more valid times must be recorded in a valid time state or valid time event relation, then, only one of them can be implicit and the remainder have to be recorded in explicit attributes. Therefore, relations with multiple valid time attributes do not have a uniform way of representation. To extend the previous remark, when a binary operation has to be performed between two relations that both have multiple valid time attributes then, there are many different ways to formulate it, depending on which of these valid time attributes is implicit in both relations.

Another limitation of the model is that a temporal relation must always have at least one explicit attribute. In addition, implicit attributes do not participate in the key of the relation. Therefore, the key of a relation consists of explicit attributes only.

Although it was proposed to become a standard, this did not occur. The above stated limitations can justify this statement.

2.4 Summary

The two fundamental characteristics that compose the database model proposed in this thesis, N1NF and the temporal features have been discussed.

Several different approaches to the support of N1NF relations have been presented and extensively reviewed. Various problems that these approaches encounter have been explained and demonstrated by examples.

Various temporal database models have also been reviewed. The presentation of these models is not based on a specific piece of work but on the overall research which has been conducted by each researcher over the years of relevant activity in the subject. The main shortcomings of each approach have also been discussed.

This chapter provides a motivation for the chapters that follow, where a new temporal nested model is formalised.

CHAPTER 3

DESIGN CONSIDERATIONS

3.1 Introduction

In chapter 2 the temporal database models have been categorised according to certain properties that distinguish them and the most important proposals for temporal database models have been reviewed.

In this chapter the properties of the nested temporal model defined in this thesis are described and the decisions that have been made are justified.

Firstly, the various temporal features that characterise temporal models are outlined. More explicitly, they concern:

- the semantics of time representation (valid time/transaction time),
- whether timestamping is applied to a tuple or to individual attributes (tuple timestamping/attribute timestamping),
- whether attribute values are defined for the same or different time period in the same tuple (homogeneous tuple/heterogeneous tuple),
- whether time is represented as points or intervals (single chronons/intervals/temporal elements).

For each of these features the alternative approaches are discussed and their advantages and disadvantages are reviewed. The static properties of the model proposed in this thesis are then given and finally, the running example used in this thesis is introduced.

3.2 Basic Temporal Definitions

Time is naturally continuous. In databases it can be treated either as continuous, i.e. isomorphic to real numbers, or as discrete, i.e. isomorphic to natural numbers. Both views of time assume that time is linearly ordered.

Discrete time has commonly been adopted by the research community in temporal databases for two reasons: it is simple and easy to be implemented ([TCG+93]).

Therefore, time is interpreted in this thesis as discrete. $T = \{0, t_1, t_2, \ldots, t_n\}$ represents the set of ordered and equally spaced time points. 0 represents the least time instance and t_n represents the greatest time instance. Time units are application dependent.

Some basic temporal terms that are going to be used in the rest of this thesis are now defined.

3.2.1 Basic concepts of time

Definition 3.1 ***(Time domain)*** A *time domain* is a non-empty, finite, totally ordered set of consecutive elements of the same time type (e.g. years, hours, minutes, seconds). ([LJ88b])

Definition 3.2 ***(Time point)*** The elements of a time domain are called *time points*.

Definition 3.3 ***(Chronon)*** *Chronon* is the shortest non-decomposable unit of time (i.e. the time period between two consecutive time points) supported by a temporal database management system.

Definition 3.4 ***(Time interval, TI)*** *Time interval* is the finite set of consecutive time points between two given time points.

$TI = [t_i,\ t_j) = \{t_k \mid t_k \in T,\ t_i \le t_k < t_j\}$ where T is defined as the set of time points.

Time intervals are closed to the left and open to the right ([LJ88b]).

Definition 3.5 ***(Time interval's start point, start)*** *Start point* of a time interval is the minimum boundary point of a time interval (t_i - Definition 3.4).

Definition 3.6 ***(Time interval's stop point, stop)*** *Stop point* of a time interval is the maximum boundary point of a time interval (t_j - Definition 3.4).

Note that the stop point does not belong to the interval.

3.2.2 Temporal elements

Definition 3.7 ***(Temporal element, TE)*** *Temporal element* is a finite set of disjoint and non-adjacent time intervals.

Example 3.1: TE = {[1, 5), [8, 9), [15, 23)}.

Definition 3.8 ***(Temporal element's start point, START)*** The *START point* of a temporal element is the minimum start point of all the start points of the time intervals that belong to this temporal element.

TE = {[$t_{1\text{START}}$, $t_{1\text{STOP}}$), [$t_{2\text{START}}$, $t_{2\text{STOP}}$), ..., [$t_{k\text{START}}$, $t_{k\text{STOP}}$)}, where k=1.

START(TE) = MIN{$t_{1\text{START}}$, $t_{2\text{START}}$, ..., $t_{k\text{START}}$}

Definition 3.9 ***(Temporal element's stop point, STOP)*** The *STOP point* of a temporal element is the maximum stop point of all the stop points of the time intervals that belong to this temporal element.

TE = {[$t_{1\text{START}}$, $t_{1\text{STOP}}$), [$t_{2\text{START}}$, $t_{2\text{STOP}}$), ..., [$t_{k\text{START}}$, $t_{k\text{STOP}}$)}, where k=1.

STOP(TE) = MAX{$t_{1\text{STOP}}$, $t_{2\text{STOP}}$, ..., $t_{k\text{STOP}}$}

Temporal elements are closed under the set theoretic operations of union, difference and intersection, which are defined next.

Let TE_1 and TE_2 be two temporal elements. Then, the following definitions are given:

Definition 3.10 ***(Union of temporal elements,*** È$_{\textbf{TE}}$***)***

It is the temporal element defined as:

$TE_1 \cup_{TE} TE_2 = \{ t \mid t \in TE_1 \vee t \in TE_2\}$

Example 3.2: {[22/1/1994, 29/6/1994), [3/10/1995, 25/12/1995), [11/3/1996, 15/8/1998)} $\cup_{TE}$ {[1/1/1994, 3/4/1994), [30/12/1995, 20/7/1997)} = {[1/1/1994, 29/6/1994), [3/10/1995, 25/12/1995), [30/12/1995, 15/8/1998)}

Definition 3.11 ***(Difference of temporal elements,*** -$_{\textbf{TE}}$***)***

It is the temporal element defined as:

$TE_1 -_{TE} TE_2 = \{ t \mid t \in TE_1 \wedge t \notin TE_2\}$

Example 3.3: {[22/1/1994, 29/6/1994), [3/10/1995, 25/12/1995), [11/3/1996, 15/8/1998)} $-_{TE}$ {[1/1/1994, 3/4/1994), [30/12/1995, 20/7/1997)} = {[3/4/1994, 29/6/1994), [3/10/1995, 25/12/1995), [20/7/1997, 15/8/1998)}

Definition 3.12 ***(Intersection of temporal elements,*** Ç$_{\textbf{TE}}$***)***

It is the temporal element defined as:

$TE_1 \cap_{TE} TE_2 = = \{ t \mid t \in TE_1 \wedge t \in TE_2\}$

Example 3.4: {[22/1/1994, 29/6/1994), [3/10/1995, 25/12/1995), [11/3/1996, 15/8/1998)} $\cap_{TE}$ {[1/1/1994, 3/4/1994), [30/12/1995, 20/7/1997)} = {[22/1/1994, 3/4/1994), [11/3/1996, 20/7/1997)}

3.2.3 Attributes and time

Definition 3.13 ***(Temporal attribute)*** An attribute is a *temporal attribute* if it is defined on the domain of temporal elements.

Definition 3.14 ***(Time-invariant attribute)*** *Time-invariant attribute* is an attribute whose values are not associated with timestamps.

Time-invariant attributes can be updated e.g. in the case of an error, but a database does not keep a history of it.

Definition 3.15 ***(Time-varying attribute)*** *Time-varying attribute* is an attribute whose values are associated with timestamps.

Definition 3.16 ***(Timestamp)*** A *timestamp* is a time value associated with a timestamped object (i.e., an attribute value or tuple). ([TCG+93])

Definition 3.17 ***(Lifespan)*** The *lifespan* of a database object is the time over which the object is defined. ([JDB+98])

3.3 Categorisation of Temporal Database Models

Temporal data models can be categorised according to their distinguishing characteristics. These have been discussed briefly in section 2.3.

The decision to include specific features in a temporal data model depends on the overall benefits that each one provides.

The advantages and disadvantages of the different approaches to temporal database models are presented next. Also, section 3.4 gives an overall assessment of the features of the new model.

3.3.1 Valid time versus Transaction time

Two different kinds of time that can be stored in temporal databases have been widely accepted by the temporal database community, valid time and transaction time. The majority of the temporal models that have been presented in the literature consider only valid time.

Databases model reality. Consequently, when the time dimension of different facts is modelled in a database, the most important issue is to model the time when these facts are true in the modelled reality. This time, in temporal database terminology, is called valid time.

It should noted that including valid time in a database, while it provides a history of the values in a tuple, does not provide a log of changes to those values. Thus, if an incorrect value is entered for an attribute and is subsequently amended, no record of the earlier incorrect value will exist once the correction has been made. In other words, past versions of the database are lost.

As mentioned in [MS91], valid time is a multifaceted aspect of time since the existence of a single object or relationship may be defined by using different times. An example of this property is the time must is stored in barrels and the time wine is ready for consumption may both be used in specifying the wine’s production (in natural and not industrialised conditions).

From the literature it can be seen that the support of valid time has produced the most interesting problems and for this reason most proposed algebras address it.

Transaction time was introduced later than valid time. Transaction time represents the time at which the data remains stored in the database and therefore it stores versions of relations and not histories of modelled reality. Therefore, transaction time errors cannot be corrected since the transaction time data forms a temporal record of the values actually stored in the database whether or not they are a correct representation of the real world. Mistakes can be eliminated only by creating a new correct record with a later transaction time. Another difference between transaction and valid time is that, in the former, future time is not supported since it has no meaning, whereas in the latter predictions and knowledge of the future do make sense and are allowed.

It has been shown that the two time dimensions are orthogonal, so they can be studied separately ([Sno87]). However, the results of some research on valid time databases are relevant to transaction time databases and vice versa, at least on issues concerning some basic temporal concepts, for example operations on time intervals.

3.3.2 Tuple timestamping versus Attribute timestamping

All tuple timestamping models proposed so far maintain the 1NF property at least concerning the non-temporal attributes. N1NF tuple timestamping models exist, e.g. [Sar90] and [JS99]; however, only the temporal attributes are nested (time in a tuple is represented as a set of time points). All other attributes are atomic and so these models have been defined by adding a temporal attribute to 1NF relations. If a tuple timestamping model supports nested data, other than nested time data, then whenever any nested data changes, the whole tuple must be replicated with the updated attribute value and associated tuple timestamp. Therefore, real N1NF tuple timestamping models are not feasible, since time-varying nested attributes cannot be supported efficiently.

In contrast, attribute timestamping models support N1NF. 1NF attribute timestamping models can exist but do not take full advantage of the features that attribute timestamping provides, since the history of an object (attribute) cannot be maintained within a single tuple, therefore they have not been proposed thus far.

Saying that, it is important to mention Lorentzos' 1NF model. His model can be used to support either tuple or attribute timestamping. He does not endorse either in [Lor88]. However, his model has been considered as a 1NF attribute timestamping model by several researchers as in [Mck88]. A features' analysis of Lorentzos and all the other major models is given in chapter 8.

As has been mentioned in chapter 2, several models have been proposed which are based on the tuple timestamping approach. This approach, although popular, has a number of shortcomings which are summarised below. It should be noted that a number of shortcomings as well as a number of advantages are derived directly from the 1NF property of tuple timestamping relations.

1. It can support only homogeneous tuples. A temporal tuple is homogeneous if the lifespans of all attribute values within it are identical. Since with tuple timestamping there is only one timestamp per tuple it is necessarily homogeneous. Consequently, the expressive power of the model is limited, because attribute values with different lifespans within the same tuple cannot be supported.

In Fig. 3.1 the name of a person may change, for example a woman's name after marriage. However, the ID remains the same since it is unique for each person. NULL values in the ADDRESS attribute mean that for that specific time interval (TIME attribute) the address of that person is unknown. The existence of NULL values is unavoidable in this relation since attribute values within the same tuple must have same lifespans.

ID	NAME	ADDRESS	TIME
1	Anna Black	NULL	[d1, d3)
1	Anna Black	52, Ladbroke Grove	[d3, d5)
1	Anna Black	NULL	[d5, d6)
1	Anna Scott	NULL	[d6, d8)
1	Anna Scott	34, Regent Square	[d8, d10)
2	Tom Thomas	20, Holland Park	[d3, d10)

Fig. 3.1: Tuple timestamping (homogeneous) relation with time represented as time intervals

2. With tuple timestamping, one new attribute must be added to the relation to represent time when the time domain is expressed using intervals (Fig. 3.1) and two attributes are needed if it is expressed using start and stop points (Fig. 3.2). Consequently, since the number of attributes in the relation increases, relations require in general more storage space.

ID	NAME	ADDRESS	START	STOP
1	Anna Black	NULL	d1	d3
1	Anna Black	52, Ladbroke Grove	d3	d5
1	Anna Black	NULL	d5	d6
1	Anna Scott	NULL	d6	d8
1	Anna Scott	34, Regent Square	d8	d10
2	Tom Thomas	20, Holland Park	d3	d10

Fig. 3.2: Tuple timestamping (homogeneous) relation with time represented as start and stop points

3. Necessarily with tuple timestamping the key must be extended to include time even if the key attribute itself is time-invariant. For example, in Fig. 3.2 in order to retrieve a single name and address it is necessary to specify values not only for the ID but for the required time point or time interval as well.

4. By their definition, temporal 1NF models do not support multivalued attributes except perhaps the attributes in which time is recorded.

5. When more than one attribute is time-varying there can still only be one time domain in a tuple. This results in either the relation having to be vertically partitioned to give one relation per time-varying attribute or introducing a new tuple each time a data item is amended, with consequent duplication of data and the appearance of NULL values (Fig. 3.1).

In Fig. 3.1 it is obvious that for the first three attributes (ID, NAME, ADDRESS) a number of pieces of data are repeated.

6. If a relation contains two or more time-varying attributes, it is not obvious following an update which attribute value has changed in the newly created tuple (see Fig. 3.1). Therefore, each attribute value has to be compared to the corresponding attribute value of the previous tuple to identify any change that may happen.

7. The use of NULL values and data duplication can be eliminated but this results in fragmented relations with very few attributes in each table, as in Fig. 3.3. As a consequence, these relations have less expressive modelling power.

ID	NAME	TIME
1	Anna Black	[d1, d6)
1	Anna Scott	[d6, d10)
2	Tom Thomas	[d3, d10)

ID	ADDRESS	TIME
1	52, Ladbroke Grove	[d3, d5)
1	34, Regent Square	[d8, d10)
2	20, Holland Park	[d3, d10)

Fig. 3.3: The data of the table in Fig. 3.1 in two fragmented tables

This approach results in a wide gulf between the logical unit of data and its physical representation. This is a serious disadvantage for all proposed query languages based on this approach because much effort is needed to link together related pieces of data from different relations when queries are executed. Additionally, the join operation which is used to associate data from different relations is an expensive operation and should be avoided when possible.

However, the benefits that tuple timestamping offers are very important and also have to be mentioned.

1. Tuple timestamping relations can be represented uniquely in contrast to nested relations used for attribute timestamping models. In other words, the structure of each relation does not need to be changed, apart for the order

of the attributes in the relation (attributes in a relation are unordered). The benefits of the above proposition are that no restructuring operations are needed in order to transform the structure of a tuple timestamping relation and in addition it is impossible for two relations with different schemes to be equivalent.

2. In general, the users can easily understand the structure of a temporal database which consists only of tuple timestamping relations and express queries using this approach.

3. Algebraic operations can be defined straightforwardly as a simple extension of corresponding operations of the conventional relational algebra because of the 1NF property.

4. An approach where tuple timestamping relations are used can be implemented more easily.

5. Traditional functional dependencies can be applied to a tuple timestamping relation since it is at least in 1NF.

In summary, the tuple timestamping approach has all the advantages of traditional relational databases.

Attribute timestamping models append at each update a new attribute value together with its timestamp to the updated tuple by means of a nested relation for each time-varying attribute.

Attribute timestamping relations present a number of shortcomings. Some of these shortcomings are caused by the nested feature of the relations that an attribute timestamping database model involves. These are:

1. The structure of a nested relation is difficult to be understood. More explicitly, the larger the number of nesting levels a nested relation has, the more complex its structure and hence, it becomes harder to understand.

2. A consequence of the above proposition is that queries can become very complex, even in those cases where only one relation is involved.

3. An implementation of an attribute timestamping model is more complex than one for a corresponding tuple timestamping model.

4. Definitions of algebraic operations for an attribute timestamping temporal database model can become very complicated, since relations in general, can consist of a finite but unknown number of nesting levels.

5. Particularly for the join operation when it is required, it is extremely difficult both to define and implement.

6. An attribute timestamping nested relation can be represented in many different ways, using different structures. In other words, different schemes can lead to equivalent relations which may cause confusion. In addition, restructuring operations must be defined.

7. Traditional functional dependencies cannot be applied in attribute timestamping nested relations, since they are in N1NF and so new functional dependencies must be defined in ways analogous to [Lev92].

Nevertheless, the attribute timestamping approach offers significant advantages over tuple timestamping:

1. It can support N1NF or nested relations as shown in Fig. 3.4 where tuples have a set of composite values for NAME and ADDRESS attributes. (The relation in Fig. 3.4 represents the same data as the relation in Fig. 3.1).

ID	NAME	ADDRESS
1	Anna Black [d1, d6) Anna Scott [d6, d10)	52, Ladbroke Grove [d3, d5) 34, Regent Square [d8, d10)
2	Tom Thomas [d3, d10)	20, Holland Park [d3, d10)

Fig. 3.4: N1NF attribute timestamping heterogeneous relation

A number of the following advantages are derived directly from the nested character of attribute timestamping relations.

2. The attribute timestamping model can support temporally heterogeneous data. However, such a model can still be used to represent a temporally homogeneous model by the use of appropriate temporal constraints. Fig. 3.4 shows a heterogeneous attribute timestamping model in which the lifespan of NAME attribute for ID=1 is [d1, d10) but the lifespan of ADDRESS attribute for the same ID is [d3, d5) $\cup$ [d8, d10).

3. In attribute timestamping models, the attribute values are functions of time. Therefore, the fragmentation of an object description is avoided since the whole history of an object is modelled in one single tuple and as a result this gives a more natural view of data.

Naturally, the time domain of attribute values is a temporal element. Even if the time domain is physically fragmented as in Fig. 3.4 where the time domain of the ADDRESS attribute for ID=1 is [d3, d5) $\cup$ [d8, d10), the attribute data values can stay in the database as a single logical object. In addition, NULL values are avoided, in contrast to the situation when tuple timestamping is used. Thus, the integrity of the temporal history is maintained.

4. Within one relation many functionally independent attributes can be simultaneously time-varying as in Fig. 3.4 where NAME and ADDRESS attributes are both independently time-varying. In addition, time-varying attributes can be expressed using different time domains (i.e. time domains of different time types e.g. years, days, months, seconds) in the same relation.

5. Time-invariant attributes are not encumbered with a timestamp as in tuple timestamping models.

6. With attribute timestamping, duplication of time-invariant data is avoided, thus saving storage space and avoiding the update anomalies which are a consequence of data redundancy.

7. Usually, a change occurs not to the values of all attributes in the same tuple but only to the values of a small subset of the attributes. Consequently, in the attribute timestamping approach, when attribute values change, the new attribute values can be inserted into the same tuple thus, avoiding the creation of a new tuple.

8. Within nested relations temporal semantics are explicitly represented.

In summary, the same data represented in attribute timestamping format can be stored in a single table in contrast to the use of tuple timestamping where they are represented either in one table with duplicated data and null values or in two or more tables. As a result, attribute timestamping models provide a more natural view, closer to how a user might perceive reality and consequently likely to be easier to design or query.

3.3.3 Homogeneous models versus Heterogeneous models

As has been described in section 2.3, temporally homogeneous database models involve relations where all attribute values in the same tuple have been defined over the same lifespan, whereas in temporally heterogeneous database models relations have tuples with different lifespans of the attribute values within them. Therefore, a more complete representation of the real world is allowed when heterogeneous tuples are used.

Homogeneity is a natural consequence of tuple timestamping models. Therefore, in all proposed tuple timestamping models, homogeneity is an implicit feature. Although algebraic operations in homogeneous relations are easy to define they have some restrictions on their expressive power. Additionally, a homogeneous relation needs more storage space since the number of tuples that are required to describe the same information is larger than in an equivalent heterogeneous relation. Another important drawback of homogeneity is the fact that certain pieces of data cannot be modelled. For example, if a father dies before his child is born the lifespan of the father does not intersect with that of the child. Consequently, a tuple (f, c) cannot be recorded in relation Parent(Father, Child).

In contrast, attribute timestamping models are more flexible since they can be either homogeneous or heterogeneous.

Although a number of researchers have defined homogeneous temporal models, some of them have tried to relax this assumption ([Gad86a]) because of the various problems it causes.

Homogeneous models can be defined as a special case of heterogeneous models. Therefore, heterogeneous models form the general case for the time domains of attributes in the same tuple of a given relation.

3.3.4 Points versus Intervals

The way in which time is represented in temporal databases has been extensively studied (e.g. [Lor88], [Tom96]). There are three different approaches to represent it: a single time point, two time points which represent an interval and a set of time intervals which form a temporal element ([CC87], [Gad88], [MS91], [TCG+93]).

Although in some of the first proposals for temporal database models, time is expressed using single time points (i.e. events), as for example in [CC87], [Ari86] and [SS87], the majority of the temporal models use intervals and temporal elements to represent time (e.g. [Lor88], [Mck88], [BG93], [Tan97]). There are many reasons that lead to this approach. Firstly, a time point denotes either the start or the end of the lifespan of an object (relation, tuple or attribute). In order to store the whole history of the object, two different attributes need to be added in the relation, i.e. the start point and the stop point, so that the lifespan of that object can be shown. In contrast, time intervals contain the complete information about the lifespan of an object in a “compact style”. In the literature, a complete study of intervals has been given, where an algebra has been described for their manipulation and the operations defined have been proved to be closed ([Lor88]).

A temporal element is defined as the union of disjoint and non-adjacent time intervals. Therefore, in respect of its semantics, a temporal element and its time intervals can be used interchangeably. However, a relation where time is represented by time intervals requires more storage space than the equivalent one where temporal elements are used. This is because the number of tuples in the former relation, in general, is larger than in the latter case, where more than one tuple can be combined into a single tuple if they have the same atomic values for all their attributes, even if they are defined over disjoint and non-adjacent time intervals. In other words, temporal elements enable the entire history of an object to be presented in a single tuple.

However, it is also important to observe that the definition of algebras is more complex when time intervals and temporal elements are used since extra properties and operations must be defined for their support of time intervals ([Lor88]).

It can be proved straightforwardly that a time point is a special case of a time interval and a time interval is a special case of a temporal element (the former is called the Duality Principle ([Lor88]).

A detailed study of point and interval types can be found in [DDL03].

3.4 The Static Properties of the Model

The model proposed in this thesis is a temporal nested relational model, called the Temporal Nested Model (TNM). Relations can be nested to any finite depth. The basic motivation is to define a temporal model with as few constraints as possible for the user. For this reason, the model is defined in such a way that the user can express the data as naturally and as easily as possible, using as few relations as possible and as a consequence as few join operations as possible.

The proposed model is neither a tuple timestamping model nor an attribute timestamping model with the traditional meaning of the words. However, TNM combines the advantages of tuple timestamping and attribute timestamping models and minimises the limitations of these two approaches.

In the proposed model, temporal elements which timestamp attribute values form temporal attributes (see Definition 3.13). Each atomic attribute which changes over time has a temporal attribute connected with it which shows for each tuple the time period over which each value of the atomic attribute is valid. Temporal attributes in the same relation can be defined over different time domains. The atomic attribute and the corresponding temporal attribute, referred to together as a *temporal nested attribute*, form a temporal nested subrelation in the general case. However, temporal attributes may also appear at the top level of relations. Therefore, time-varying attributes are timestamped by taking advantage of the nested feature of the model.

Temporal nested subrelations can contain other temporal or non-temporal subrelations as well.

Nested attributes in TNM can contain time-varying attributes, atomic time-invariant attributes or even both. Compound keys are supported. Key attributes can be time-varying. A nested attribute can be a key attribute.

Example 3.5: An example of a nested key attribute is shown in Fig. 3.5. The key of that relation is the nested attribute NAME. Semantically, since a name of a person can change with time, as for example a woman's name after marriage, the whole tuple of this nested attribute, as a single object, uniquely identify the corresponding tuple of the relation.

NAME		ADDRESS		COURSE
N	N_PERIOD	A	A_PERIOD	
Anna Black	[d1, d6)	50, Homer St.	[d3, d6)	Ph.D.
Anna Scott	[d6, d10)	34, Porchester Sq.	[d8, d10)	
Tom Thomas	[d3, d7)	20, Holland Pk.	[d3, d7)	B.Sc.

Fig. 3.5: A nested relation with key the nested attribute NAME

One important feature of the model is that, when the key is time-varying, a single timestamp is applied to the whole key, whether a simple nested attribute or a compound key. The timestamp of the key provides a lifespan for the tuple in these cases. This is similar to Clifford and Croker's proposal for timestamping each whole tuple in addition to each time-varying attribute ([CC87]). However, the model proposed in this thesis makes use of temporally heterogeneous tuples rather than the homogeneous tuples they suggested.

Time in TNM is represented by temporal elements.

The representation used in TNM is believed by the author of this thesis to have many advantages compared to the previously described models where the time domain of an attribute value is part of the same attribute as that value. Firstly, data describing an object is not fragmented into many relations, since they can be nested within the same relation. Moreover, extra operations such as Temporal Atom Decomposition, Temporal Atom Formation and Drop-Time ([Tan97]), which have been briefly described in section 2.3.1, can be avoided. In addition, when the relation is viewed from the external level it can be characterised as an attribute timestamping relation, while at an internal level (that of a temporal subrelation) it can be viewed as a tuple timestamping relation. The advantage of this approach is that atomic values and time values form different attributes and so can be referenced separately which is very important since they have different properties. Key attributes are not necessarily time-varying. However, TNM allows this potentiality for cases where it is appropriate semantically.

3.5 The Running Example of the Thesis

Two databases are used as examples in the thesis. The first one does not contain any temporal data while the second one does. However, the two examples contain similar information with the addition of temporal attributes in the second case. The nested database example is used to illustrate the NRM and the temporal nested database example is used for the TNM. The two databases are shown below and some explanation is given. In the examples in chapters 4, 5 and 6 there are cases where relations from the two databases are used in a slightly modified form for the sake of the examples. These will be indicated when they are used.

3.5.1 The nested database example

The running nested database example of the thesis consists of seven relations TRAINING, DEPT, LOCATION, CASH-POINT, EMPLOYMENT, PAYMENT and COURSE (Fig. 3.6-3.12). All these relations are nested relations. An explanation of the running example is now given in order to introduce the reader to the example which will be used in the rest of this thesis.

Relation TRAINING (Fig. 3.6) holds data about courses and trainers provided by IT companies. It consists of one atomic attribute, COMPANY, and one nested attribute, TRAINER. Attribute TRAINER in turn, consists of one atomic attribute, TRN, and one nested attribute, COURSE. Subrelation COURSE consists of one atomic attribute, CODE, and one nested attribute, C, which consists of two atomic attributes, CN and Y.

Semantically, the attributes of the TRAINING relation have the following meaning: COMPANY - company name, TRN - trainer name, CODE - course code, CN - course numberY - year in which the course was taken. A specific course can be identified uniquely by both course number (CN) and year (Y); a specific course consists of a number of different topics (see rel. COURSE-Fig. 3.12) which can be given by different trainers belonging to different companies.

Relation DEPT (Fig. 3.7) holds data about the different departments of a company as well as the trainers who have given courses to the staff of these departments. DEPT consists of three attributes, the atomic attributes D and DN and the nested attribute UNIT. Subrelation UNIT consists of three attributes, the atomic attributes UN and UD and the nested attribute COURSE_DETAILS. COURSE_DETAILS consists of two atomic attributes, TRN and COMPANY and one nested attribute, the C attribute. Subrelation C contains two atomic attributes, CN and Y.

The semantics of the attributes of relation DEPT are: D - department number, DN - department name, UN – unit number, UD – unit description, TRN – trainer name, COMPANY – company name, CN - course number and Y - year in which the course was taken. Relation DEPT (Fig. 3.7) is a modified version of relation DEPT in [SS86].

Relation LOCATION (Fig. 3.8) contains data about branches of different companies. It consists of one atomic attribute, COMPANY, and one nested attribute, ANNEX. Subrelation ANNEX consists of two atomic attributes, BUILDING and ADDRESS.

The attributes of relation LOCATION have the following semantics: COMPANY –company name, BUILDING – building name and ADDRESS – street name.

Relation CASH-POINT (Fig. 3.9) has data about cash-points that different banks own. CASH-POINT consists of two attributes, the atomic attribute BANK and the nested attribute BRANCH. Attribute BRANCH consists of two atomic attributes, SORT_CODE and ADDRESS.

The semantics of the attributes of relation CASH-POINT are: BANK – bank name, SORT_CODE – sort code of the branch and ADDRESS – street name.

Relation EMPLOYMENT (Fig. 3.10) contains data about the employees that work for different companies. EMPLOYMENT consists of two attributes, the atomic attribute NAME and the nested attribute JOB. Attribute JOB consists of two atomic attributes, COMPANY and JOB_DESCRIPTION.

Semantically, the meaning of the attributes of relation EMPLOYMENT is: NAME – employee name, COMPANY – company name, JOB_DESCRIPTION – job description.

Relation PAYMENT (Fig. 3.11) shows the salaries that different companies give for different jobs. PAYMENT consists of two attributes, the atomic attribute SALARY and the nested attribute JOB. Attribute JOB consists of two atomic attributes, COMPANY and JOB_DESCRIPTION.

The semantics of the attributes of relation PAYMENT are: SALARY – salary range, COMPANY – company name, JOB_DESCRIPTION – job description.

Relation COURSE (Fig. 3.12) contains data about the different courses that took place. It consists of four attributes, nested attributes C and SUBJECT and atomic attributes COURSE_DURATION and TITLE. Attribute C consists of two atomic attributes, CN and Y, and attribute SUBJECT consists of one atomic attribute, the TOPICS attribute.

Semantically, the meaning of the attributes of relation COURSE is: CN - course number, Y - year in which the course was taken, COURSE_DURATION – course duration (number of hours), TITLE – course title and TOPICS – course topics.

COMPANY	TRAINER			
	TRN	COURSE		
		CODE	C	
			CN	Y
Apple	Jack	xx0	1	75
			2	76
	Mark	xy1	1	82
			3	82
		xy2	2	79
IBM	Tim	xy1	3	82
		xx2	5	79
			4	82
Microsoft	Karen	xx1	2	77
			2	81

Fig. 3.6: TRAINING

D	DN	UNIT					
		UN	UD	COURSE_DETAILS			
				TRN	COMPANY	C	
						CN	Y
1	Research	511	Software Engineering	Mark	Apple	1	75
						2	76
						5	79
		552	Basic Research	Karen	Microsoft	1	82
						2	79
				Tim	IBM	5	79
		678	Planning	Mark	Apple	2	76
						4	82
2	Development	650	Design	Karen	Microsoft	1	75
		780	Maintenance	Tim	IBM	3	82
				Mark	Apple	2	76
		981	Planning	Jack	Apple	2	81
						3	82
						5	79

Fig. 3.7: DEPT

COMPANY	ANNEX	
	BUILDING	ADDRESS
TOSHIBA	North Building	Porchester Rd.
IBM	Maple House	Kendal Av.
	Main Building	Danebury Rd.
Microsoft	Pegasus House	Ashford St.
	Queen's Building	Park Rd.

Fig. 3.8: LOCATION

BANK	BRANCH	
	SORT_CODE	ADDRESS
Barclays	386600	Ashford St.
NatWest	560045	Park Rd.
	560038	Porchester Rd.
Lloyd's	478202	Ashford St.
	478210	Park Rd.

Fig. 3.9: CASH-POINT

NAME	JOB	
	COMPANY	JOB_DESCRIPTION
Anna	Microsoft	Secretary
	TOSHIBA	Secretary
Paul	IBM	Software Engineer
	Microsoft	Programmer
Mark	Apple	Director

Fig. 3.10: EMPLOYMENT

SALARY	JOB	
	COMPANY	JOB_DESCRIPTION
15,500-19,500	TOSHIBA	Secretary
	Apple	Secretary
18,000-23,000	Microsoft	Programmer
	Microsoft	Secretary
25,000-30,000	Apple	Director

Fig. 3.11: PAYMENT

C		COURSE_DURATION	TITLE	SUBJECT
CN	Y			TOPICS
1	75	80	Computer Skills	Access
2	77			Word
				Excel
2	82	120	Multimedia	Power Point
3	82			Internet
2	79	20	Programming	C++
				JAVA

Fig. 3.12: COURSE

3.5.2 The temporal nested database example

The running temporal nested database example consists of five relations, T_TRAINING, T_DEPT, T_LOCATION, T_CASH-POINT and T_COURSE (Fig. 3.13-3.17). All relations are temporal nested relations. These relations are modified versions of the corresponding nested relations described in section 3.5.1 (Fig. 3.6-3.9, 3.12). For this reason only the new attributes that have been introduced in the temporal nested version of the example are explained in this section.

Relation T_TRAINING (Fig. 3.13) contains data about trainers. More precisely, it shows in which companies they work, which courses they have taught and the period of time over which each course was taking place. Attribute TRAINER is a temporal nested attribute, which consists of one atomic attribute, TRN, and one temporal nested attribute, COURSE. Subrelation COURSE consists of one atomic attribute, CN, and one temporal attribute, CN_PER$_t$ attribute. Semantically, the meaning of the new or different attributes is: CN – course number (a course consists of a number of different topics (see relation T_COURSE below-Fig. 3.17) which can be given by different trainers belonging to different companies), CN_PER$_t$ – duration of each course.

Relation T_DEPT (Fig. 3.14) contains data about a company's staff. Attribute STAFF is a temporal nested attribute. It consists of three attributes, the atomic attributes UN and UD and the temporal nested attribute COURSE_DETAILS. COURSE_DETAILS consists of one atomic attribute, SNAME, one temporal attribute, STAFF_PER$_t$ and one temporal nested attribute, COURSE. Subrelation COURSE contains one atomic attribute, CN, and one temporal attribute, CN_PER$_t$. The semantics of the new attributes is: SNAME - staff name, STAFF_PER$_t$ – period of staff employment, CN_PER$_t$ - duration of each course.

Relation T_LOCATION (Fig. 3.15) is very similar to its corresponding nested version, the LOCATION relation (Fig. 3.8). The only difference is that the ANNEX attribute is a temporal nested attribute which consists of two atomic attributes, BUILDING and ADDRESS and one temporal attribute, ADDRESS_PER$_t$. The semantics of the new attribute is the following: ADDRESS_PER$_t$ – time during which a company's annex was at a specific address.

Similarly, the only difference between relation T_CASH-POINT (Fig. 3.16) and its corresponding nested version, the CASH-POINT relation (Fig. 3.9) is that attribute BRANCH of relation T_CASH-POINT is a temporal nested attribute which consists of three attributes, the atomic attributes SORT_CODE and ADDRESS and the temporal attribute ADDRESS_PER$_t$. The semantics of the new attribute is: ADDRESS_PER$_t$ – time period of a bank's branch at a specific address.

In relation T_COURSE (Fig. 3.17) the nested attribute C of nested relation COURSE (Fig. 3.12) has been replaced by the temporal nested attribute COURSE. Attribute COURSE consists of one atomic attribute, CN and one temporal attribute, CN_PER$_t$. The semantics of the new attribute is: CN_PER$_t$– duration of each course.

COMPANY	**TRAINER**		
	TRN	**COURSE**	
		CN	**CN_PER$_t$**
Apple	Jack	5.2	[2/11/1994, 25/4/1995) ∪ [7/8/1996, 1/1/2010)
	Mark	3.3	[2/1/1992, 8/11/1996)
		3.5	[30/4/1995, 1/1/2010)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
		5.0	[17/12/1995, 1/1/2010)
Microsoft	Karen	3.3	[25/6/1996, 1/1/2010)

Fig. 3.13: T_TRAINING

D	DN	STAFF					
		UN	UD	COURSE_DETAILS			
				SNAME	STAFF_PER$_t$	COURSE	
						CN	CN_PER$_t$
1	Research	511	Software Engineering	Paul	[13/5/1994, 5/9/1996)	5.2	[1/2/1995, 24/6/1995)
						5.0	[27/8/1995, 30/1/1996)
				Peter	[26/2/1996, 1/1/2010)	3.5	[1/1/1998, 28/10/1998)
		552	Basic Research	Anna	[30/4/1994, 27/8/1995) $\cup$ [4/6/1997, 19/11/1998)	3.1	[1/7/1995, 1/8/1995)
						3.3	[29/9/1997, 10/2/1998)
				Mary	[15/5/1995, 1/1/2010)	3.3	[17/1/1997, 28/4/1997)
		678	Planning	Katy	[24/1/1994, 10/7/1995)	3.2	[22/4/1995, 15/5/1995)
						5.4	[13/2/1994, 4/3/1995)
2	Development	650	Design	Steve	[2/1/1995, 27/6/1998)	5.0	[18/3/1996, 1/7/1996)
		780	Maintenance	Helen	[14/2/1996, 1/1/2010)	3.5	[17/8/1997, 1/1/2010)
				Pat	[21/6/1995, 31/1/1996)	2.2	[18/9/1995, 10/10/1995)

Fig. 3.14: T_DEPT

COMPANY	ANNEX		
	BUILDING	ADDRESS	ADDRESS_PER$_t$
Toshiba	North Building	Porchester Rd.	[3/8/1995, 1/1/2010)
IBM	Maple House	Kendal Av.	[17/1/1996, 22/5/1998)
	Main Building	Danebury Rd.	[10/6/1998, 1/1/2010)
Microsoft	Pegasus House	Ashford St.	[29/10/1994, 4/4/1997)
	Queen's Building	Park Rd.	[18/3/1995, 1/1/2010)

Fig. 3.15: T_LOCATION

BANK	BRANCH		
	SORT_CODE	ADDRESS	ADDRESS_PER$_t$
Barclays	386600	Ashford St.	[16/11/1995, 23/12/1998)
NatWest	560045	Park Rd.	[1/2/1993, 10/8/1998)
	560038	Porchester Rd.	[6/5/1994, 20/2/1995)
Lloyd's	478202	Ashford St.	[23/7/1995, 1/1/2010)
	478210	Park Rd.	[16/6/1995, 1/1/2010)

Fig. 3.16: T_CASH-POINT

COURSE		COURSE_DURATION	TITLE	SUBJECT
CN	CN_PER$_t$			TOPICS
5.0	[27/8/1995, 30/1/1996)	35	Presentation Skills	Power Point
				Word
				Outlook Express
3.3	[17/1/1997, 28/4/1997)	15	Multimedia	Power Point
				Internet
3.5	[17/8/1997, 10/1/2001)	180	Computer Skills	Access
5.4	[1/1/1995, 6/3/1995)			Excel
5.2	[13/2/1994, 4/3/1995)	80	Programming	C++
				JAVA

Fig. 3.17: T_COURSE

3.6 Summary

When the time dimension is added to database models it provides opportunities for having a number of different approaches to temporal database models. These approaches can be distinguished from each other by their answers to the following questions:

1. Semantically, does time represent valid time or transaction time?
2. Does the model use tuple timestamping or attribute timestamping?
3. Are the tuples temporally homogeneous or heterogeneous?
4. Is time represented by single chronons, intervals or temporal elements?

The advantages and disadvantages of each of these characteristics have been examined and the properties of the model proposed in the present thesis have been given.

In the last section of this chapter the running example, which is going to be used in the rest of this thesis to demonstrate the various features and algebraic operations of the model, is described.

CHAPTER 4

THE NESTED RELATIONAL MODEL (NRM)

4.1 Introduction

A new non-temporal nested relational model is defined in this chapter, called the Nested Relational Model, NRM. Relations in NRM can be nested to any finite depth.

The operations of the model are formally defined. Union, difference, intersection, projection, selection, unnest, nest, rename, cartesian product, natural join and T–join operations are recursively defined. For each definition, an example is presented in order to make it clearer. For the first time, the natural join operation is defined for any pair of nested relations which have one or more attributes in common, even when they are in different subrelations and at different nesting levels in each relation. The generalisation of natural join uses one or more of the six distinct cases of the nested natural join operation which are identified in this chapter, distinguished by certain properties of the attributes in the *join paths* between the relations that participate in the join operation. These properties depend on whether an attribute is either atomic or relation-valued and on whether it is at either the top level or lower level (same or different) of the two relations. The generalisation of natural join is shown to be applicable to all joinable nested relations. The recursive rename operation for nested relations is also formally defined for the first time. Formal definitions for aggregate functions for nested relations are also included in NRM.

It is important to emphasise that the NRM constitutes the base for the equivalent temporal nested model, TNM, which is defined in the next chapter, but in addition, forms by itself, a well-defined complete model for nested relations.

4.2 Basic Concepts and Terminology

In order to introduce the Nested Relational Model (NRM) in the next section it is necessary to present firstly the basic concepts and terminology that are going to be used. Some of the following definitions have been used before by the database community. However, a repetition of these definitions at the present point is necessary for completeness. Moreover, some terms and notation are introduced for the first time in the present thesis in order to provide the essential formalisation of the presented model.

Definition 4.1 ***(Relation-valued attributes or nested attributes)*** Relation-valued attributes or nested attributes are attributes which contain non-atomic values.

Relation-valued attributes or nested attributes can be considered as subrelations of the relations to which they belong.

Definition 4.2 ***(Non-first normal form relations or nested relations)*** Non-first normal form relations or nested relations are relations which contain relation-valued attributes or nested attributes.

In this thesis, relations with atomic attributes only will be called *flat relations*, whereas relations that contain relation-valued attributes or only atomic attributes will be referred to as *nested relations*. In other words, in this thesis, flat relations are considered as special cases of nested relations. Furthermore, attributes that contain non-atomic values will be referred to as *nested attributes* and attributes that contain only atomic values will be called *atomic attributes*.

Attr(R) is the set of attributes of relation r with scheme name R, i.e. Attr(R) = $\{R_1, R_2, ..., R_n\}$, where $n \geq 1$ and $R_1, R_2, ..., R_n$ are the attributes of R, either atomic or nested.

Definition 4.3 ***(Tree structure)*** Every nested relation r with relation scheme R can be represented as a tree with root node R. All the nested attributes of the relation are the non-leaf nodes of the tree and all the atomic attributes form the leaf nodes of the tree.

The tree structure is a very useful representation of a nested relation since the scheme of a nested relation can become complex and so the tree offers a clear graphical representation of the nested structure.

Example 4.1: The tree structure of the TRAINING relation (Fig. 3.6) is shown in Fig. 4.1.

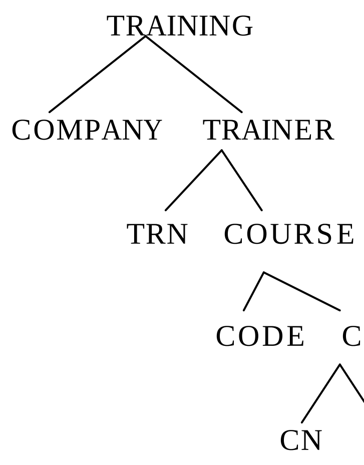

Fig. 4.1: Tree representation of relation TRAINING

Definition 4.4 ***(Nesting levels of a relation)*** The number of nesting levels of a relation is equal to the maximum number of nodes to be passed through starting from the root to reach any atomic attribute in the tree representation. The root of the relation is by definition at nesting level 0.

Example 4.2: The nesting levels of relation TRAINING (Fig. 4.1) are 4.

Consequently, the nesting level of an attribute in a relation can be computed by counting the number of nodes which must be passed through from the root node to get to that attribute. For example, atomic attribute TRN of relation TRAINING (Fig. 4.1) is at nesting level 2.

Definition 4.5 ***(Common attributes between two relations)*** Two (flat or nested) relations have an atomic attribute in common if they both contain an atomic attribute which has the same name and domain in both relations. Two nested relations have a nested attribute in common if they both contain a nested attribute which has the same name and the same scheme (the same attributes with the same names defined over the same domains).

The above definition can be applied recursively for nested attributes containing one or more nested attributes.

The path of an attribute is defined recursively below.

Definition 4.6 ***(Path)*** Let $L_{A_n \rightarrow A_j}$ be the path of nested or atomic attribute A_j belonging to nested attribute A_n, which is a child of the root of relation R. Then, $L_{A_n \rightarrow A_j}$ is defined as follows:

i) $L_{A_n \rightarrow A_j} = A_n$, where $A_j = A_n$

ii) $L_{A_n \rightarrow A_j} = A_n(L_{A_{n+1} \rightarrow A_j})$, where A_{n+1} is an attribute of A_n either equal to or containing A_j.

Then, the set of all attributes (atomic and nested) of R can be defined as $Attr(R) = \{R_{a1}, R_{a2}, \ldots, R_{ap}, R_{n1}, \ldots, R_{ni}, \ldots, R_{nq}\}$

$$= \{R_{a1}, R_{a2}, \ldots, R_{ap}, R_{n1}, \ldots, \bigcup_{k=0}^{m} L_{R_{ni} \rightarrow R_{ni_k}}, \ldots, R_{nq}\}$$

where:

- $R_{a1}, R_{a2}, \ldots, R_{ap}$ are atomic attributes at nesting level 1 of relation R ($p \geq 0$),
- $R_{n1}, \ldots, R_{ni}, \ldots, R_{nq}$ are nested attributes at nesting level 1 of relation R ($1 \leq i \leq q$),
- $R_{ni_k} = \begin{cases} R_{ni} & \text{for } k = 0 \\ R_{ni_k} & \text{for } k \neq 0 \text{ (i.e. an attribute that has nested attribute } R_{ni} \text{ as its ancestor)} \end{cases}$
- m is the number of descendants' attributes of nested attribute R_{ni}.

Example 4.3: The path is used for the definition of an attribute in a nested relation, in contrast to flat relations, since the whole path of an attribute is needed in order to identify that specific attribute. As an example, consider the nested relation DEPT (Fig. 3.7) with tree structure in Fig. 4.2.

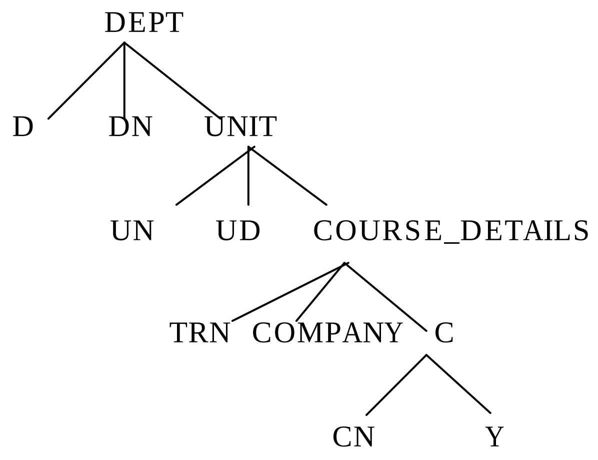

Fig. 4.2: Tree representation of relation DEPT

Then, the set of all attributes of relation DEPT is the following:

Attr(DEPT) = {D, DN, UNIT, UNIT(UN), UNIT(UD), UNIT(COURSE_DETAILS),
UNIT(COURSE_DETAILS(TRN)),
UNIT(COURSE_DETAILS(COMPANY)),
UNIT(COURSE_DETAILS(C)),
UNIT(COURSE_DETAILS(C(CN))),
UNIT(COURSE_DETAILS(C(Y)))}

and the path of the atomic attribute CN is:

$L_{UNIT \to CN}$ = UNIT($L_{COURSE\_DETAILS \to CN}$) = UNIT(COURSE_DETAILS($L_{C \to CN}$)) = UNIT(COURSE_DETAILS(C($L_{CN \to CN}$))) = UNIT(COURSE_DETAILS(C(CN))).

From the above example, it is apparent that the name of an attribute by itself is not enough in general to uniquely identify the attribute, since in nested relations an attribute is fully defined by reference to both its name and its position in the tree structure of the relation in which it belongs. In addition, there are cases in which two common attributes belong in the same relation but in different subrelations, as for example in the result relation of a join operation. Consequently, the only way for the two attributes to be distinguished from one another is by their paths. Therefore, the path of an attribute shows whether the attribute belongs to a nested attribute or not, as well as the nesting level of it. The path of an attribute identifies the attribute uniquely.

Definition 4.7 ***(Two nested relations having the same scheme)*** Two nested relations have the same scheme iff they contain only common attributes (atomic and/or nested) -see Definition 4.5.

An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set ([Ull95]). For the case of a nested database model, entity sets are nested relations and the definition of the key must be expanded in order to support nested attributes as well.

Informally, a nested relation can have either atomic or nested attributes or even a conbination of atomic and nested attributes as a key. Semantically, a nested attribute is a key of a nested relation, when each set of values of the nested attribute that belongs to the same tuple, uniquely identifies that tuple. That implies that each of these set of values of the nested attribute distinguishes, as an entirety, solely the tuple in which it belongs.

Formally, the definition of a key of a nested relation is given below:

Definition 4.8 ***(Key of a nested relation)*** The key of a nested relation r with relation scheme R, can be a set K consisting of atomic and/or nested attributes of R such that for any two tuples t_i and t_j in the relation the following constraint is valid at all times: $t_i[K] \neq t_j[K]$, where $i \neq j$ and with the additional property that removing any attribute from K leaves a set of attributes that is not a key of R.

Example 4.4: An example of a nested key attribute can be found in section 3.4 (Fig. 3.5).

It is considered, by the author of this thesis, that an approach where nested attributes are allowed to be part of key attributes is an important benefit for a nested model. Nested models, where nested attributes are not allowed to be part of key attributes, have a significant limitation, since relations, as the one presented in Fig. 3.5, cannot be supported. Therefore, there are cases that are not covered by such an approach.

Many authors have adopted the PNF assumption, defined by Roth, Korth and Silberschatz in [RKS88], in their approaches. A relation is in PNF when all the atomic attributes of the relation participate in the key of the relation and in addition, each nested attribute of the relation is also in PNF (see section 2.2.2). The PNF assumption presupposes that nested attributes cannot form part of a key in a nested relation, a significant restriction of a nested database model, as explained above.

Consequently, in the nested model defined in the present thesis, the relaxing of the restriction that other nested models impose, to allow nested attributes as part of the key, is a considerable extension and thus, an important benefit that the NRM offers.

In addition, time is allowed in key. There are two major objections for this approach:

1) Key should be short.

This is correct from the point of efficiency. From a theoretical point of view, however, the relational model does not impose any restriction of the form 'attributes of data type A are not allowed to participate in the key'. Therefore, since the work in the present thesis is the definition of a relational algebra and not the development of efficient methods, the allowance of union of intervals to be part in the key is absolutely correct.

2) Key should have a fixed length.

This also relates to efficiency. Again, however, the relational model does not impose any restriction related to the size of data that is recorded in an attribute. Hence, the same applies to an attribute that participates in a key. If implementation issues are considered, however, two solutions are presented below:

Assume that one table is R (T, A, B) and the key is T, of variable length.

1st Solution (Best Solution):

In a way not seen by the user, this table is internally maintained as R(Id, T, A, B). For the system, the key is Id and now, this key has a fixed size. This Id is not seen by the user. For the user, however, the key is T. Note: In SQL BLOB data types are supported now. They have varying length and enable recording images etc. They are not allowed to participate in the key. One disadvantage is that relations with attributes of a BLOB type may not be involved in UNION, EXCEPT and other operations.

Note that normally, commercial DBMSs use to consider an extra column in addition to those of users. For example, INGRES has such a column (ColID), where the systems records automatically a unique tuple identifier.

2nd Solution:

In a way not seen by the user, this table is internally maintained in two tables:

R(Id, A, B) with key Id and Time(Id, T) with key Id.

Now, key has a fixed size and the system performs a join, whenever necessary, in order to maintain the table or reply a query.

4.3 Operations in the NRM

The operations in the NRM are introduced in this section. The algebra of the non-temporal nested model forms the heart of the temporal nested model, TNM, which will be formally presented in the next chapter. It is important to emphasise at this point that the NRM is not just a model designed to constitute the base for the equivalent temporal one, but forms by itself, a well-defined complete model for nested relations.

The algebraic operations of the NRM are defined recursively.

Recursive algebraic definitions in nested models are undoubtedly preferable to the corresponding non-recursive ones. This is based on the following facts:

1) The non-recursive algebras allow operations only on entire tuples. In contrast, recursive algebras allow the direct manipulation of tuples either at the top level or at lower levels of the nested relations.

2) When an attribute at a lower nesting level of the nested relation needs to be accessed, because it participates in an operation expressed in a non-recursive algebra, one or more unnesting operations need to be applied resulting in the creation of many additional tuples. The non-recursive operation can then be performed and finally the relation is nested again. However, one of the main motivations for a model consisting of nested relations is the reduction in the number of tuples processed.

3) In the non-recursive algebras, queries can become long and complicated, while in the recursive algebras queries will be shown to be compact, simpler and more naturally expressed.

4) Restructuring operations are not required with recursive algebras unlike non-recursive ones.

5) Traditional query optimisation techniques can be used with recursive algebras. In contrast, nest and unnest operations which have to be used

frequently in non-recursive algebras, are not, in general, inverse operations. Therefore, traditional query optimisation techniques can be applied to queries which are expressed using recursive algebras since recursive operations can be performed at any nesting level without using nest or unnest operations ([Lev92]).

However, it has been shown that the recursive and non-recursive algebras are equivalent in expressive power ([Col90]).

All of the relational algebra operations defined for flat relations are now redefined using recursive definitions for nested relations. The “base case” of each recursive operator has the same definition as the non-recursive one; i.e., the recursive definition can be reduced to the non-recursive one when relations do not contain any nested attributes.

4.3.1 The *Recursive Nested Union* Operation (È$^{\text{È}}$)

Let r and q be two nested (in general) relations with relation schemes R and Q respectively. Assume that the two relations have the same relation scheme i.e. $R = Q = \{S(R), R_1, R_2, \ldots, R_n\}$ where S(R) is the set containing all the key nested attributes and all the atomic attributes of R and Q (the same for the two relations) and $\{R_1, R_2, \ldots, R_n\}$ are the non-key nested attributes of R and Q. Assume also that Attr(R) is the set of all attributes (atomic and nested) of the two relations, t_r is a tuple in relation r, t_q is a tuple in relation q and t is a tuple in the result relation ($r \cup^{\cup} q$).

Then, the union of the two relations r and q is defined as follows:

Definition 4.9 (*Recursive Nested Union*)

i) Non-recursive union for flat relations ($r \cup q$)

$$r \cup q = \{\, t \mid ((\exists\, t_r \in r)\ (t[Attr(R)] = t_r[Attr(R)]))$$
$$\vee\ ((\exists\, t_q \in q)\ (t[Attr(R)] = t_q[Attr(R)]))\}$$

ii) Recursive union for nested relations ($r \cup^{\cup} q$)

$$r \cup^{\cup} q = \{\, t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ ((t[S(R)] = t_r[S(R)] \cup t_q[S(R)])$$
$$\wedge\ ((t[R_1] = t_r[R_1] \cup^{\cup} t_q[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n] \cup^{\cup} t_q[R_n]))))\}$$

Example 4.5: Let relations TRAINING_2 (Fig. 4.3) and TRAINING_4 (Fig. 4.4) be two modified versions of relation TRAINING (Fig. 3.6) having the same scheme. Please note that relation TRAINING_2 is the same as that of Fig. 2.3

(section 2.2.3). However, the reason for this repetition is to simplify the reading of this specific example.

In both relations, TRAINING_2 and TRAINING_4, S(TRAINING_2) = S(TRAINING_4) = COMPANY.

The union of the two relations, according to the above definition, is shown in Fig. 4.5.

COMPANY	**TRAINER**		
	TRN	**C**	
		CN	**Y**
Apple	Jack	1	75
		2	76
	Mark	1	82
		3	82
		2	79
IBM	Tim	3	82
		5	79
		4	82
Microsoft	Karen	2	77
		2	81

Fig. 4.3: TRAINING_2

COMPANY	**TRAINER**		
	TRN	**C**	
		CN	**Y**
Apple	Jack	6	82
		2	76
	Mark	3	82
		2	79
IBM	Tim	5	84
Microsoft	Karen	2	77
		2	81

Fig. 4.4: TRAINING_4

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	1	75
		2	76
		6	82
	Mark	1	82
		3	82
		2	79
IBM	Tim	3	82
		5	79
		4	82
		5	84
Microsoft	Karen	2	77
		2	81

Fig. 4.5: TRAINING_2 $\cup^{\cup}$ TRAINING_4

4.3.2 The *Recursive Nested Difference* Operation ($-^{-}$)

Let r and q be two nested (in general) relations with relation schemes R and Q respectively. Assume that the two relations have the same relation scheme $\{S(R), R_1, R_2, \ldots, R_n\}$, where S(R) is the set of all the key nested attributes and all the atomic attributes of R and Q (the same for the two relations) and $\{R_1, R_2, \ldots, R_n\}$ are the non-key nested attributes of R and Q. Assume also that Attr(R) is the set of all attributes (atomic and nested) of the two relations, t_r is a tuple in relation r, t_q is a tuple in relation q and t is a tuple in the result relation ($r -^{-} q$).

Then, the difference of the two relations r and q is defined as follows:

Definition 4.10 (*Recursive Nested Difference*)

i) Non-recursive difference for flat relations (r - q)

$$r - q = \{ t \mid (\exists\, t_r \in r)\ (\forall\, t_q \in q)\ ((t[Attr(R)] = t_r[Attr(R)]) \wedge (t[Attr(R)] \neq t_q[Attr(R)]))\}$$

ii) Recursive difference for nested relations ($r -^{-} q$)

$$\begin{aligned} r -^{-} q = \{ t \mid\ & ((\exists\, t_r \in r)\ (\forall\, t_q \in q) \\ & ((t[S(R)] = t_r[S(R)] - t_q[S(R)]) \wedge (t[R_1] = t_r[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n]))) \\ & \vee ((\exists\, t_r \in r, \exists\, t_q \in q)\ ((t[S(R)] = t_r[S(R)] = t_q[S(R)]) \\ & \wedge (t[R_1] = t_r[R_1] -^{-} t_q[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n] -^{-} t_q[R_n]))))\} \end{aligned}$$

Example 4.6: The difference of the two relations TRAINING_2 (Fig. 4.3) and TRAINING_4 (Fig. 4.4) is shown in Fig. 4.6.

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	1	75
	Mark	1	82
IBM	Tim	3	82
		5	79
		4	82

Fig. 4.6: TRAINING_2 -$^{-}$ TRAINING_4

4.3.3 The *Recursive Nested Intersection* Operation (Ç$^{\text{Ç}}$)

The intersection of two nested (in general) relations r and q, having the same scheme R is a nested relation with scheme R that contains only the tuples which have exactly the same values in every attribute in both relations. Formally, let r and q be two nested relations with relation schemes R and Q respectively. Assume that the two relations have the same relation scheme R and let S(R) be all the key attributes of the relations (atomic and nested) and all the non-key atomic attributes of the relation scheme R. Let $\{R_1, \ldots, R_n\}$ be all the non-key nested attributes of R. Assume also that Attr(R) is the set of all attributes (atomic and nested) of the two relations, t_r is a tuple in relation r, t_q is a tuple in relation q and t is a tuple in the result relation ($r \cap^{\cap} q$).

Then, the intersection of the two relations r and q, is defined as follows:

Definition 4.11 (*Recursive Nested Intersection*)

i) Non-recursive intersection for flat relations ($r \cap q$)

$$r \cap q = \{ t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ (t[Attr(R)] = t_r[Attr(R)] = t_q[Attr(R)])\}$$

ii) Recursive intersection for nested relations ($r \cap^{\cap} q$)

$$r \cap^{\cap} q = \{ t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ ((t[S(R)] = t_r[S(R)] \cap t_q[S(R)]) \wedge ((t[R_1] = t_r[R_1] \cap^{\cap} t_q[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n] \cap^{\cap} t_q[R_n]))))\}$$

Example 4.7: The intersection of the two relations TRAINING_2 (Fig. 4.3) and TRAINING_4 (Fig. 4.4) is shown in Fig. 4.7.

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	2	76
	Mark	3	82
		2	79
Microsoft	Karen	2	77
		2	81

Fig. 4.7: TRAINING_2 $\cap^{\cap}$ TRAINING_4

4.3.4 The *Recursive Nested Projection* Operation (p^{p})

Let r be a nested (in general) relation with relation scheme R and let $\{R_{a1}, \ldots, R_{ak}\}$ be the subset of atomic attributes of R which are going to be projected and $\{R_{n1}, \ldots, R_{nm}\}$ the subset of nested attributes of R which are going to be projected either fully or attributes belonging to these nested attributes ($k, m \geq 0$).

In order to define the projection operation, the term *project list* needs to be defined firstly. In general, a project list is a list of pr oject paths. A project path of an attribute which is going to be projected is the path of that attribute (see Definition 4.6).

Definition 4.12 (*Project list*) L_π is a project list of R if

i) L_π is empty (the project list of an atomic attribute is empty).

ii) L_π is of the form $(R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})$, where $L_{n1}, \ldots, L_{nm}$ are project lists of nested attributes $R_{n1}, \ldots, R_{nm}$ respectively.

Then, the projection operation in a nested relation r, $\pi^{\pi}(rL_\pi)$, where t_r is a tuple in relation r and t is a tuple in the result relation, is defined as follows:

Definition 4.13 (*Recursive Nested Projection*)

i) $\pi(r) = r$

ii) $\pi^{\pi}(r(R_{a1}, \ldots, R_{ak}, R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})) = \{ t \mid (\exists\, t_r \in r)$

$$((t[R_{a1}] = t_r[R_{a1}]) \wedge \ldots \wedge (t[R_{ak}] = t_r[R_{ak}])$$

$$\wedge\, (t[R_{n1}] = \pi^{\pi}(t_r[R_{n1}]L_{n1})) \wedge \ldots \wedge (t[R_{nm}] = \pi^{\pi}(t_r[R_{nm}]L_{nm})))\}$$

Example 4.8: Given relation TRAINING_2 (Fig. 4.3) consider the following query: “Retrieve the course numbers for the courses that each company has run”. The result is shown in Fig. 4.8.

COMPANY	TRAINER
	C
	CN
Apple	1
	2
	1
	3
	2
IBM	3
	5
	4
Microsoft	2
	2

Fig. 4.8: π^{π}(TRAINING_2(COMPANY, TRAINER(C(CN))))

4.3.5 The *Recursive Nested Selection* Operation (s^{s})

Let r be a nested (in general) relation with relation scheme R and let R_a = {R_{a1}, ..., R_{ak}} and R_n = {R_{n1}, ..., R_{nm}} be the subsets of all atomic and nested attributes of R respectively that participate in the selection operation, where k and m are less than or equal to the number of atomic and nested attributes at the top level in the relation R, respectively. Let also, c be a set of conditions in R, which is of the form {c_a, c_n} where c_a={c_{a1}, ..., c_{ak}} is a set of conditions which must be true for the atomic attributes R_{a1}, ..., R_{ak} of R respectively and c_n ={c_{n1}, ..., c_{nm}} is a set of conditions that must hold for the nested attributes R_{n1}, ..., R_{nm} of R respectively. When both sets of conditions are applied simultaneously then, the result is obtained by computing the intersection of the two results. In addition, the condition can be no matter complicated, as for example equality of nested attributes. If, two multi-valued nested attributes are compared for equality, they are treated as sets so, since each nested attribute is, in fact, a relation, equal tuples are searched at the level of the nested relations.

In order to define the selection operation, the term *select list* needs to be defined firstly. In general, a select list is a list of select paths. A select path of an attribute that is going to participate in the selection, is the path of that attribute (see Definition 4.6). The select list is defined recursively.

Definition 4.14 (*Select list*) L_σ is a select list of R if

i) L_σ is empty (all the atomic attributes of relation r have empty select lists).

ii) L_σ is of the form $(R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})$ where $L_{n1}, \ldots, L_{nm}$ are select lists of nested attributes $R_{n1}, \ldots, R_{nm}$ respectively.

Then, a selection operation of the relation r, where t_r is a tuple in relation r and t is a tuple in the result relation, is defined as follows:

Definition 4.15 (*Recursive Nested Selection*)

i) $\sigma(r_{ca1, \ldots, cak}) = \{\, t \mid (\exists\, t_r \in r)$

$((t[Attr(R) - \{R_{a1}, \ldots, R_{ak}\}] = t_r[Attr(R) - \{R_{a1}, \ldots, R_{ak}\}])$

$\wedge ((t[R_{a1}] = t_r[R_{a1}]) \wedge c_{a1} = true)$

$\wedge \ldots \wedge ((t[R_{ak}] = t_r[R_{ak}]) \wedge c_{ak} = true))\}$

ii) $\sigma^\sigma(r_{cn1, \ldots, cnm}L_\sigma) = \{\, t \mid (\exists\, t_r \in r)$

$((t[Attr(R) - \{R_{n1}, \ldots, R_{nm}\}] = t_r[Attr(R) - \{R_{n1}, \ldots, R_{nm}\}])$

$\wedge (t[R_{n1}] = \sigma^\sigma(t_r[R_{n1}]_{cn1}L_{n1}) \neq \emptyset)$

$\wedge \ldots \wedge (t[R_{nm}] = \sigma^\sigma(t_r[R_{nm}]_{cnm}L_{nm}) \neq \emptyset))\}$

In the general case, the selection operation can be defined as the intersection of the two previously defined cases as follows:

$$\sigma^\sigma(r_c L_\sigma) = \sigma^\sigma(r_{ca1, \ldots, cak, cn1, \ldots, cnm}L_\sigma) = \sigma(r_{ca1, \ldots, cak}) \cap \sigma^\sigma(r_{cn1, \ldots, cnm}L_\sigma)$$

Example 4.9: Given relation TRAINING_2 (Fig. 4.3) consider the following query: "Find all the information of the TRAINING_2 relation of those courses that have been given by trainers Mark or Tim during the year 1982".

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Mark	1	82
		3	82
IBM	Tim	3	82
		4	82

Fig. 4.9: $\sigma^\sigma(\text{TRAINING\_2}_{((\text{TRAINER(TRN)} = \text{'Mark' OR 'Tim'}) \text{ AND } (\text{TRAINER(C(Y))} = 82))})$

The unnest operation (section 4.3.6) as well as the nest operation (section 4.3.7) are restructuring operations, since they change the scheme of the relation in which they are applied.

4.3.6 The *Recursive Unnest* Operation (μ^{μ})

Let r be a nested (in general) relation with relation scheme R.

Definition 4.16 (*Unnest list*) L_{μ} is an unnest list of R if it is of the form

i) R_i, where R_i is a nested attribute of R at the top level.

ii) (R_iL_i) where L_i is an unnest list of the nested attribute R_i.

Let Attr(R) be the set of all attributes of R and R_i a nested attribute of R, at the top level of R. Let also, t_r be a tuple in relation r and t a tuple in the result relation. Then, the unnest operation, $\mu^{\mu}(r_{L\mu})$, is defined as follows (see also [Col90]):

Definition 4.17 (*Recursive Unnest*)

i) $\mu(r_{Ri}) = \{ t \mid (\exists\ t_r \in r)\ ((t[Attr(R) - R_i] = t_r[Attr(R) - R_i]) \wedge (t[R_i]\ ?\ t_r[R_i]))\}$

ii) $\mu^{\mu}(r_{RiLi}) = \{ t \mid (\exists\ t_r \in r)\ ((t[Attr(R) - R_i] = t_r[Attr(R) - R_i])$

$\wedge\ (t[R_i] = \mu^{\mu}(t_r[R_i]_{Li})))\}$

Example 4.10: The result of unnesting relation TRAINING (Fig. 3.6) on the COURSE attribute, i.e. $\mu^{\mu}(\text{TRAINING}_{\text{TRAINER(COURSE)}})$, is shown in Fig. 4.10.

COMPANY	TRAINER			
	TRN	CODE	C	
			CN	Y
Apple	Jack	xx0	1	75
			2	76
	Mark	xy1	1	82
			3	82
	Mark	xy2	2	79
IBM	Tim	xy1	3	82
	Tim	xx2	5	79
			4	82
Microsoft	Karen	xx1	2	77
			2	81

Fig. 4.10: $\mu^{\mu}(\text{TRAINING}_{\text{TRAINER(COURSE)}})$

4.3.7 The *Recursive Nest* Operation (v^{v})

Let r be a nested (in general) relation with relation scheme R.

Definition 4.18 (*Nest list*) L_v is a nest list of R if it is of the form

i) $(R_1, \ldots, R_n)$ where $R_1, \ldots, R_n$ are attributes of R, either atomic or nested at the top level of R.

ii) (R_iL_i) where L_i is a nest list of the nested attribute R_i.

Let Attr(R) be the set of all attributes of R and $A_n = \{R_1, \ldots, R_n\}$ the set of attributes of R that are going to be nested to form a new nested attribute A.

Let also, t_r be a tuple in relation r, t a tuple in the result relation and s a tuple of the new nested attribute A. Then, the nest operation, $\nu^{\nu}(r_{L\nu \to A})$, is defined as follows (see also [Col90]):

Definition 4.19 (*Recursive Nest*)

i) $\nu(r_{An \to A}) = \{ t \mid (\exists\, t_r \in r)\ ((t[Attr(R) - A_n] = t_r[Attr(R) - A_n])$

$\wedge\ (t[A] = \{s[A_n] \mid (s\ ?\ r)\ (s[Attr(R) - A_n] = t_r[Attr(R) - A_n])\}))\}$

ii) $\nu^{\nu}(r_{(RiLi) \to A}) = \{ t \mid (\exists\, t_r \in r)\ ((t[Attr(R) - R_i] = t_r[Attr(R) - R_i])$

$\wedge\ (t[R_i] = \nu^{\nu}(t_r[R_i]_{Li \to A})))\}$

Example 4.11: In order to return to relation TRAINING (Fig. 3.6) from the relation $\mu^{\mu}(\text{TRAINING}_{\text{TRAINER(COURSE)}})$ of Fig. 4.10, a nest operation needs to be performed, i.e. $\nu^{\nu}(\mu^{\mu}(\text{TRAINING}_{\text{TRAINER(COURSE)}})_{\text{TRAINER(CODE, C)} \to \text{TRAINER(COURSE)}})$.

4.3.8 The *Recursive Nested Rename* Operation (ρ^{ρ})

The rename operation takes a specified relation and returns another that is identical to the given one except that at least one of its attributes has a different name ([Dat00]). The rename operation is useful before or after performing a number of operations, as for example for cases when there are duplicate names in the result relation after performing a join operation of two relations, or when the cartesian product operation is performed between two relations having attributes with the same name. When a rename operation takes place only the heading of the relation changes, the body (instance) remains the same.

Let r be a nested (in general) relation with relation scheme $R = \{R_1, R_2, \ldots, R_i, \ldots, R_n, A, B, \ldots, Z\}$, where $R_1, R_2, \ldots, R_i, \ldots, R_n$ are atomic attributes and A, B, ..., Z are nested attributes at the top level of relation R.

Then, the rename operation ρ^{ρ} of relation r is defined as follows:

Definition 4.20 (*Recursive Nested Rename*)

i) Rename of an atomic attribute R_i to $R_{i'}$ at the top level of relation R

$\rho[R_i \leftarrow R_{i'}](R) = \{R_1, R_2, \ldots, R_{i'}, \ldots, R_n, A, B, \ldots, Z\}$

ii) Rename of a nested attribute A to A′ at the top level of relation R

$\rho[A \leftarrow A'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, \bigcup_{k=0}^{m} L_{A'\rightarrow Ak}, B, \ldots, Z\}$

where m is the number of attributes that are descendants of A and for m = 0, A′ = A_0 (atomic attribute at the top level of R) and case (ii) reduces to case (i).

iii) Rename of an atomic or nested attribute A_i to A_i' at a lower level of relation R

$\rho^{\rho}[A_i \leftarrow A_i'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, A, A_1, \ldots, \bigcup_{k=0}^{m} L_{A\rightarrow Ai'k}, B, \ldots, Z\}$, where

A_1 is a child attribute of nested attribute A, A_i is an attribute at a lower level of relation R belonging to nested attribute A and m is the number of descendants that A_i has (m = 0, when atomic, in which case $A_{i'0} = A_{i'}$).

When more than one attribute has to be renamed the definition is recursive, as follows:

$\rho^{\rho}[R_{a1}\leftarrow R'_{a1}, \ldots, R_{ak}\leftarrow R'_{ak}, R_{n1}\leftarrow R'_{n1}, \ldots, R_{nm}\leftarrow R'_{nm}, R_{l1}\leftarrow R'_{l1}, \ldots, R_{lp}\leftarrow R'_{lp}](R) =$
$(\rho^{\rho}[R_{lp}\leftarrow R'_{lp}](\ldots(\rho^{\rho}[R_{l1}\leftarrow R'_{l1}](\rho[R_{nm}\leftarrow R'_{nm}](\ldots(\rho[R_{n1}\leftarrow R'_{n1}](\rho[R_{ak}\leftarrow R'_{ak}](\ldots(\rho[R_{a1}\leftarrow R'_{a1}]$
$(R))))))))))$

where $R_{a1}, \ldots, R_{ak}$ are atomic attributes at the top level of relation R, $R_{n1}, \ldots, R_{nm}$ are nested attributes at the top level of relation R and $R_{l1}, \ldots, R_{lp}$ are either atomic or nested attributes at lower levels (different, in general) of relation R and k, m, p ≥ 0. The names of the attributes having primes denote the new names that these attributes are going to be renamed.

Example 4.12: Consider the relation DEPT with tree structure in Fig. 4.2 and let attribute UD be renamed as UD′ and attribute C as C′. Then, the rename operation is defined as follows:

ρ^{ρ}[UD ← UD′, C ← C′](DEPT) =

ρ^{ρ}[C ← C′](ρ^{ρ}[UD ← UD′](DEPT)) =

ρ^{ρ}[C ← C′]({D, DN, UNIT, UNIT(UN), UNIT(UD′), UNIT(COURSE_DETAILS), UNIT(COURSE_DETAILS(TRN)), UNIT(COURSE_DETAILS(COMPANY)), UNIT(COURSE_DETAILS(C)), UNIT(COURSE_DETAILS(C(CN))), UNIT(COURSE_DETAILS(C(Y)))}) =

{D, DN, UNIT, UNIT(UN), UNIT(UD′), UNIT(COURSE_DETAILS), UNIT(COURSE_DETAILS(TRN)), UNIT(COURSE_DETAILS(COMPANY)), UNIT(COURSE_DETAILS(C′)), UNIT(COURSE_DETAILS(C′(CN))), UNIT(COURSE_DETAILS(C′(Y)))}

The tree structure of relation DEPT after the renaming of attributes UD and C is shown in Fig. 4.11.

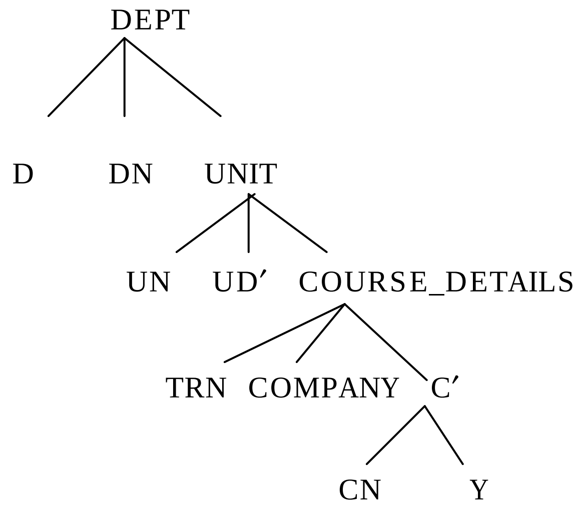

Fig. 4.11: The tree representation of relation DEPT after the renaming of attributes UD and C to UD′ and C′ respectively

4.3.9 The *Recursive Nested Cartesian Product* Operation (´ ´)

Let R be a relation scheme of relation r.

Definition 4.21 ***(Join path)*** L is a join path of R if either:

(i) L is empty or

(ii) L = R_iL_i where R_i is a nested attribute of R and L_i is a join path of R_i. ([Col90])

The join path can be represented as a branch of the tree structure of some nested relation R starting from a child of the root of the tree and going down to some node of the tree that represents either an atomic or nested attribute. In other words, the join path consists of all the nodes that are passed in order to reach a specific attribute.

Example 4.13: In relation DEPT (Fig. 3.7) an example of a join path is UNIT(COURSE_DETAILS(TRN)). The tree structure of the relation DEPT is shown in Fig. 4.2.

Let r and q be two nested (in general) relations with relation schemes R and Q respectively and let Attr(R) be all the attributes (atomic and nested) of R, Attr(Q) all the attributes (atomic and nested) of Q and L a join path of R. Let, also, R_i be a nested attribute of R, L_i a join path of R_i, t_r a tuple in relation r, t_q a tuple in relation q and t a tuple in the result relation. The cartesian product operation can be applied either at the top level of both relations or between a lower nesting level of a relation and the top level of another relation. The first case is exactly the same as the standard cartesian product for flat relations.

So, the cartesian product of two relations r and q is defined as follows (see also [Col90]):

Definition 4.22 (*Recursive Nested Cartesian Product*)

i) $\times(r, q) = \{ t \equiv (t[Attr(R)], t[Attr(Q)]) \mid$

$(\exists\, t_r \in r, \exists\, t_q \in q)\ ((t[Attr(R)] = t_r[Attr(R)]) \wedge (t[Attr(Q)] = t_q[Attr(Q)]))\}$

ii) $\times^{\times}(rL, q) = \times^{\times}(r(R_iL_i), q) \equiv \times^{\times}(q, r(R_iL_i)) =$

$\{ t \mid (\exists\, t_r \in r)\ ((t[Attr(R) - \{R_i\}] = t_r[Attr(R) - \{R_i\}]) \wedge (t[R_i] = \times^{\times}(t_r[R_i]L_i, q)))\}$

An example follows where the second case is demonstrated.

It follows from the formal definition of the recursive nested cartesian product operation that the result relation of the cartesian product of nested relations r and q consists of the attributes of relation r plus the attributes of relation q. In addition, the commutative property is again satisfied, as is the case in the CRM. Thus, it is always valid:

$\times^{\times}(r(R_iL_i), q) \equiv \times^{\times}(q, r(R_iL_i))$

Example 4.14: The cartesian product operation is performed between the COURSE attribute of relation TRAINING (Fig. 3.6) and the CASH-POINT relation (Fig. 3.9). Due to the large number of tuples in the result relation, only a part of it is displayed in Fig. 4.12.

Note that, in case the operands of the cartesian product are interchanged, the result remains exactly the same.

COMPANY	(TRN (CODE C BANK BRANCH))						
	TRN	(CODE C BANK BRANCH)					
		CODE	C		BANK	BRANCH	
			CN	Y		SORT_CODE	ADDRESS
Apple	Jack	xx0	1	75	Barclays	386600	Ashford St.
			2	76			
		xx0	1	75	NatWest	560045	Park Rd.
			2	76		560038	Porchester Rd.
		xx0	1	75	Lloyd's	478202	Ashford St.
			2	76		478210	Park Rd.
	Mark	xy1	1	82	Barclays	386600	Ashford St.
			3	82			
		xy1	1	82	NatWest	560045	Park Rd.
			3	82		560038	Porchester Rd.
		xy1	1	82	Lloyd's	478202	Ashford St.
			3	82		478210	Park Rd.
		xy2	2	79	Barclays	386600	Ashford St.
		xy2	2	79	NatWest	560045	Park Rd.
						560038	Porchester Rd.
		xy2	2	79	Lloyd's	478202	Ashford St.
						478210	Park Rd.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Fig. 4.12: $\times^{\times}$ (TRAINING(TRAINER(COURSE)), CASH-POINT)

Note: As can be seen from the above example, the cartesian product operation is not often a semantically meaningful operation. However, it helps in defining the join operation (see sections 4.3.10 and 4.3.11), since the join is a special case of a cartesian product operation and for this reason it is included here.

4.3.10 The *Recursive Nested Natural Join* operation ($\bowtie^{\bowtie}$)

The natural join operation is the most complicated operator, especially in nested models, since the two relations which participate in the natural join can have multiple common attributes (atomic or nested) which can be in different subrelations and at different nesting levels in each of the two joined relations.

When the two nested relations which participate in the natural join operation have two or more (atomic or nested) attributes in common in different subrelations and at different levels of nesting in each relation, the natural join operation can be informally stated as follows:

Step 1: Define a join path for each relation.

Step 2: Join the two relations using one of the six cases (formally defined later).

Step 3: In the resulting relation choose another pair of common attributes.

Step 4: Construct the paths for both attributes.

Step 5: Find the first different nodes on the paths progressing towards the common attributes.

Step 6: Assume that these two nodes are the root nodes of the two subrelations that are going to be joined.

Step 7: Perform the natural join of the two subrelations according to one of the six cases.

Step 8: Repeat the steps 3 to 7 until no more common attributes are in the result relation.

Note: When the natural join operation is performed the result relation may contain one or more new subrelations, which are created by joining two subrelations of the two original relations. The names of these new subrelations are formed from the attributes of which they are composed and are then enclosed in parentheses.

An example of the general case of the recursive nested natural join operation is given below.

Example 4.15: Consider the relations TRAINING_1 (Fig. 2.1) and DEPT (Fig. 3.7) and suppose that the following query is given: “In which years has the course with code number xy1 been taught?”.

The two relations must be joined in order to answer this query. To do this, the 8-step algorithm, which was described earlier, must be executed.

The two relations TRAINING_1 and DEPT are joined on two pairs of common attributes. The two pairs are: (PROGRAMME(TRN), UNIT(COURSE_DETAILS(TRN))) and (COMPANY, UNIT(COURSE_DETAILS(COMPANY))), where the first attribute of each pair belongs to relation TRAINING_1 and the second belongs to relation DEPT.

Step 1: Initially, one of the pairs of join paths given above must be selected. For our example, the first join paths are PROGRAMME(TRN) and UNIT(COURSE_DETAILS(TRN)) for relations TRAINING_1 and DEPT respectively.

Step 2: The natural join is performed according to Case 3b (defined later) where the two common atomic attributes are not at the same nesting levels in the two joined relations. The tree representation of the result relation x_1 is shown in Fig. 4.13.

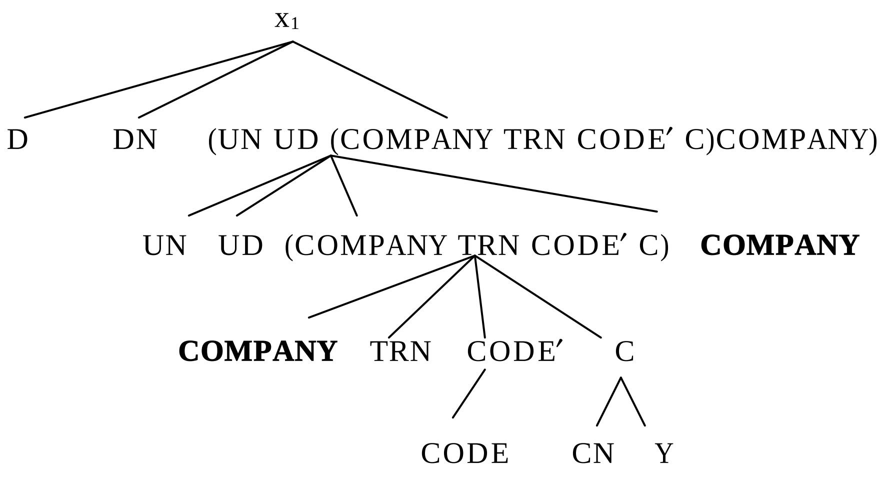

Fig. 4.13: The tree representation of relation x_1

Step 3: In the result relation only the attribute COMPANY appears twice.

Step 4: The paths for these two attributes, with the same name COMPANY, are: (UN UD (COMPANY TRN CODE′ C)COMPANY)(COMPANY) and (UN UD (COMPANY TRN CODE′ C)COMPANY)((COMPANY TRN CODE′ C) (COMPANY)).

Step 5: For these two attribute occurrences the first different nodes starting from the root are the (COMPANY TRN CODE′ C) attribute and the COMPANY attribute.

Step 6: So, the two subrelations that must be joined are the subrelation with root node (COMPANY TRN CODE′ C) and the subrelation which has only the atomic attribute COMPANY.

Step 7: The natural join operation is performed according to Case 1, which is described below.

The tree representation of the result relation is shown in Fig. 4.14 and the result relation of the natural join operation in Fig. 4.15.

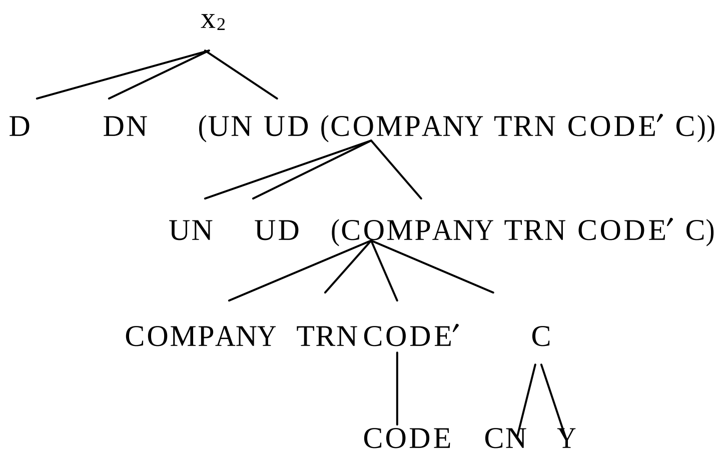

Fig. 4.14: The tree representation of the result relation

$x_2 = \bowtie^{\bowtie}$ (TRAINING_1, DEPT)

D	DN	(UN UD (COMPANY TRN CODE¢ C))						
		UN	UD	(COMPANY TRN CODE¢ C)				
				COMPANY	TRN	CODE′	C	
						CODE	CN	Y
1	Research	511	Software	Apple	Mark	xy1	1	75
			Engineering			xy2	2	76
							5	79
		678	Planning	Apple	Mark	xy1	2	76
						xy2	4	82
		552	Basic	IBM	Tim	xy1	5	79
			Research			xx2		
				Microsoft	Karen	xx1	1	82
							2	79
2	Development	981	Planning	Apple	Jack	xx0	2	81
							3	82
							5	79
		780	Maintenance	Apple	Mark	xy1	2	76
						xy2		
				IBM	Tim	xy1	3	82
						xx2		
		650	Design	Microsoft	Karen	xx1	1	75

Fig. 4.15: The result relation $x_2 = \rhd\lhd^{\rhd\lhd}$ (TRAINING_1, DEPT)

Definition 4.23 (***Generalised Natural Join***) Let r and q be two nested relations with relation schemes R and Q respectively and let $A = \{A_0, A_1, \ldots, A_j\}$ be the set of all common attributes that the two relations have, where $A_0, A_1, \ldots, A_j$ are atomic or nested attributes either at the top or lower levels in the two relations. Then, the generalised natural join is defined as follows:

$\rhd\lhd^{\rhd\lhd}(r, q) = \rhd\lhd^{\rhd\lhd}(s_jL_{s_jA_j}, s'_jL_{s'_jA_j})(\ldots(\rhd\lhd^{\rhd\lhd}(s_1L_{s_1A_1}, s'_1L_{s'_1A_1})(\rhd\lhd^{\rhd\lhd}(rL_{rA_0}, qL_{qA_0}))))$

where $\rhd\lhd^{\rhd\lhd}(rL_{rA_0}, qL_{qA_0}) = x_1$, $\rhd\lhd^{\rhd\lhd}(s_1L_{s_1A_1}, s'_1L_{s'_1A_1}) = x_2$, ..., $\rhd\lhd^{\rhd\lhd}(s_jL_{s_jA_j}, s'_jL_{s'_jA_j}) = x_{j+1}$ and $(s_1, s'_1), \ldots, (s_j, s'_j)$ pairs, are subrelations of $x_1, \ldots, x_j$ respectively with their root node being the first different nodes along the paths to the common attributes $A_1, \ldots, A_j$ respectively.

Fig. 4.16 shows all the different cases which are analysed below giving their formal semantics. Formal definitions are given even for the simplest cases in order to have a unified representation of the nested natural join operation.

The nested natural join operation presented in this section is defined recursively. The six cases which are examined in detail can be grouped into two more general categories. The first category involves joining two nested relations which have atomic attributes in common and the second involves joins between two nested relations which have nested attributes in common, i.e. subrelations. In each category three cases are examined depending on the join paths of the relations to be joined.

Category 1

1st relation / **2nd relation**	**atomic attribute top level**	**atomic attribute not top level**
atomic attribute top level	Case 1	Case 2
atomic attribute not top level	Case 2	Case 3a (same level) / Case 3b (not same level)

Category 2

1st relation / **2nd relation**	**nested attribute top level**	**nested attribute not top level**
nested attribute top level	Case 4	Case 5
nested attribute not top level	Case 5	Case 6a (same level) / Case 6b (not same level)

Fig. 4.16: The different cases of common attributes between two relations that participate in the nested natural join operation

The two tables in Fig. 4.16 can be represented in a different way. In the general case, the recursive nested natural join operation, written as $\triangleright\triangleleft^{\triangleright\triangleleft}$(rL, qM), where L,¯ and M,¯ are the lengths of the join paths L and M recursively, can be distinguished in the following different cases:

$$\rhd\lhd^{\rhd\lhd}(rL, qM) = \left\{ \begin{array}{ll} \text{i) } L,^{-} = \emptyset,\ M,^{-} = \emptyset & \text{Case 1 / Case 4} \\ \text{ii) } (L,^{-} \neq \emptyset,\ M,^{-} = \emptyset) \vee (L,^{-} = \emptyset,\ M,^{-} \neq \emptyset) & \text{Case 2 / Case 5} \\ \text{iii) } L,^{-},\ M,^{-} \neq \emptyset \wedge L,^{-} = M,^{-} & \text{Case 3a / Case 6a} \\ \text{iv) } L,^{-},\ M,^{-} \neq \emptyset \wedge L,^{-} \neq M,^{-} & \text{Case 3b / Case} \\ \text{6b} & \end{array} \right\}$$

Fig. 4.17

***Case 1**: Join a nested (or flat) relation to another nested (or flat) relation which have one or more atomic attributes at the top level in common*

This natural join is exactly the same as the standard natural join for flat relations. If either of the relations are nested (they contain subrelations), when the natural join operation is performed, the subrelations behave like common (atomic) attributes.

Definition 4.24: Let r and q be two relations (nested, in general) with relation schemes $\{R_1, R_2, \ldots, A_1, \ldots, A_j, \ldots, R_n\}$ and $\{Q_1, Q_2, \ldots, A_1, \ldots, A_j, \ldots, Q_m\}$ respectively where $j > 0$ and $n, m \geq j$ and different in general. The two relations r and q have in common the atomic attributes $A_1, \ldots, A_j$. Then, $\rhd\lhd(r, q)$ is defined as follows:

$$\begin{aligned} \rhd\lhd(r, q) = \{\, t \mid (\exists\, t_r \in r)(\exists\, t_q \in q) \\ &((t[A_1, \ldots, A_j] = t_r[A_1, \ldots, A_j] = t_q[A_1, \ldots, A_j]) \\ &\wedge (t[Attr(R) - \{A_1, \ldots, A_j\}] = t_r[Attr(R) - \{A_1, \ldots, A_j\}]) \\ &\wedge (t[Attr(Q) - \{A_1, \ldots, A_j\}] = t_q[Attr(Q) - \{A_1, \ldots, A_j\}]))\} \end{aligned}$$

Case 2: *Join two nested relations having one or more atomic attributes in common which in one relation are atomic attributes of a subrelation of the relation and in the other are at the top level*

This definition is the same as Colby's recursive definition of the natural join operation ([Col90]), where the join path of the one relation is not empty, but is defined from the subrelation of the relation which participates in the natural join operation (this subrelation contains at a lower level the common atomic attributes). In this case, the result relation consists of all the attributes of the one relation except the subrelation which participates in the natural join operation and the remaining attributes of the result are computed by joining the subrelation of that relation to the other relation. If the join path is empty, the natural join is performed according to Case 1. If it is not, the same procedure is followed as before until the join path becomes empty. Thus, the natural join operation is applied recursively.

Definition 4.25: Let r and q be two nested relations (in general) with relation schemes $\{R_1, R_2, \ldots, R_i, \ldots, R_n\}$ and $\{Q_1, Q_2, \ldots, A_1, \ldots, A_j \ldots, Q_m\}$ respectively where i, j > 0, $n \geq i$ and $m \geq j$. Assume without loss of generality that the two relations have in common one or more atomic attributes which in the one relation belong to the nested attribute R_i and in the other relation are the atomic attributes $A_1, \ldots, A_j$. Further, let rL represent relation r with join path L. Then, $\triangleright\triangleleft^{\triangleright\triangleleft}$ (rL, q) is defined as follows:

$$\triangleright\triangleleft^{\triangleright\triangleleft}(rL, q) = \triangleright\triangleleft^{\triangleright\triangleleft}(q, rL) = \triangleright\triangleleft^{\triangleright\triangleleft}(r(R_iL_i), q) = \{ t \mid (\exists\, t_r \in r)$$
$$((t[Attr(R) - \{R_i\}] = t_r[Attr(R) - \{R_i\}])$$
$$\wedge\ (t[R_i] = \triangleright\triangleleft^{\triangleright\triangleleft}(t_r[R_i]L_i, q) \neq \emptyset))\}$$

Case 3: *Join a nested relation to another nested relation which have one or more common atomic attributes which belong in different subrelations of the two relations (but in the same subrelation in each relation)*

There are two subcases depending on whether or not the nesting levels of the common atomic attributes in the different subrelations of the two relations are the same.

Case 3a: *The common atomic attributes are at the same nesting level in the two joined relations*

In this case, the result relation consists of all the attributes of the two relations which do not take part in the natural join operation and a new subrelation, which is formed by joining the subrelations which contain the common atomic attributes, by applying this method recursively until the common atomic attributes are reached. Only the tuples which have equal values in the common attributes are then selected.

Definition 4.26: Let r and q be two nested relations with relation schemes $R = \{R_1, R_2, ..., R_i, ..., R_n\}$ and $Q = \{Q_1, Q_2, ..., Q_j, ..., Q_m\}$ respectively where i, j > 0, $n \geq i$ and $m \geq j$. Suppose that at least the attributes R_i and Q_j of the two relations r and q respectively are nested attributes and that they contain the common atomic attributes.

Let L be a join path of R and let M be a join path of Q. L is of the form R_iL_i where L_i is a join path of R_i and M is of the form Q_jM_j where M_j is a join path of Q_j. Then, $\rhd\lhd^{\rhd\lhd}$ (rL, qM) is defined as follows:

$$\rhd\lhd^{\rhd\lhd}(rL, qM) = \rhd\lhd^{\rhd\lhd}(r(R_iL_i), q(Q_jM_j)) = \{ t \mid (\exists\, t_r \in r)(\exists\, t_q \in q)$$
$$((t[Attr(R) - \{R_i\}] = t_r[Attr(R) - \{R_i\}])$$
$$\wedge\ (t[Attr(Q) - \{Q_j\}] = t_q[Attr(Q) - \{Q_j\}])$$
$$\wedge\ (t[R_iQ_j] = \rhd\lhd^{\rhd\lhd}(t_r[R_i]L_i,\ t_q[Q_j]M_j) \neq \emptyset))\}$$

As shown above, the different levels of nesting are traversed until the subrelations which contain the common atomic attributes are reached. The common subrelations can then be joined using Definition 4.24.

Example 4.16: Consider the relations LOCATION (Fig. 3.8) and CASH-POINT (Fig. 3.9) and suppose that the following query is given: “Which banks have cash-points on the same road as Microsoft has a branch?”. In order to answer this query the natural join of the two relations must be computed. The common attribute is the atomic attribute ADDRESS which belongs in different subrelations of the two relations -in relation LOCATION in the subrelation ANNEX and in relation CASH-POINT in the subrelation BRANCH- but in each case at nesting level 2. The result relation is shown in Fig. 4.18.

COMPANY	(BUILDING	ADDRESS	SORT_CODE)	BANK
	BUILDING	ADDRESS	SORT_CODE	
TOSHIBA	North Building	Porchester Rd.	560038	NatWest
Microsoft	Pegasus House	Ashford St.	386600	Barclays
Microsoft	Queen’s Building	Park Rd.	560045	NatWest
Microsoft	Pegasus House	Ashford St.	478202	Lloyd’s
	Queen’s Building	Park Rd.	478210	

Fig. 4.18: $\rhd\lhd^{\rhd\lhd}$ (LOCATION, CASH-POINT)

Case 3b: *The common atomic attributes are not at the same nesting level in the two joined relations*

In this case, the tree representations of the two relations that take part in the natural join operation can help us to design the scheme tree of the result relation since even the simplest relations in this case are complex (the nesting levels of the relations are at least two and not necessarily the same).

The result relation is constructed iteratively. It consists of the common atomic attributes and all its siblings from both joined relations. The parent nodes of the atomic attributes in common in the two joined relations form the parent node of the common attributes and all its siblings. The siblings of the parent nodes remain siblings of the result parent node. The same procedure is followed until the root of both relations is reached, so that the hierarchy of the joined relations is maintained in the result relation.

Let r and q be two nested relations with relation schemes
$R = \{R_{i1}(R_{(i-1)1}(\ldots(R_{21}(A_{11}\ldots A_{1j}\ R_{1j+1}\ldots R_{1n})\ldots R_{2k})\ldots)R_{(i-1)l})\ldots R_{im}\}$ and
$Q = \{Q_{i'1}(Q_{(i'-1)1}(\ldots(Q_{21}(A_{11}\ldots A_{1j}Q_{1j+1}\ldots Q_{1n'})\ldots Q_{2k'})\ldots)Q_{(i'-1)l'})\ldots Q_{i'm'}\}$ respectively where $i, n, k, l, m, j, i', n', k', l'$ and m' are positive integers, not equal in general. The common atomic attributes are the attributes $A_{11}, \ldots, A_{1j}$. All the other attributes which do not form the join paths can be atomic or non-atomic but since they do not participate in the natural join operation, this is of no consequence.

Assume that: i) $i \neq i'$, which means that the two relations have different levels of nesting (in fact, the assumption is $i < i'$ without loss of generality) and ii) the common atomic attributes $A_{11} \ldots A_{1j}$ are the first attributes in the subrelations in which they belong and these subrelations are the first attributes of the subrelations in which they belong and so on, since the order of the attributes at the same nesting level and in the same subrelation is not significant.

In Fig. 4.19 and 4.20 the tree representations of the two relations are shown.

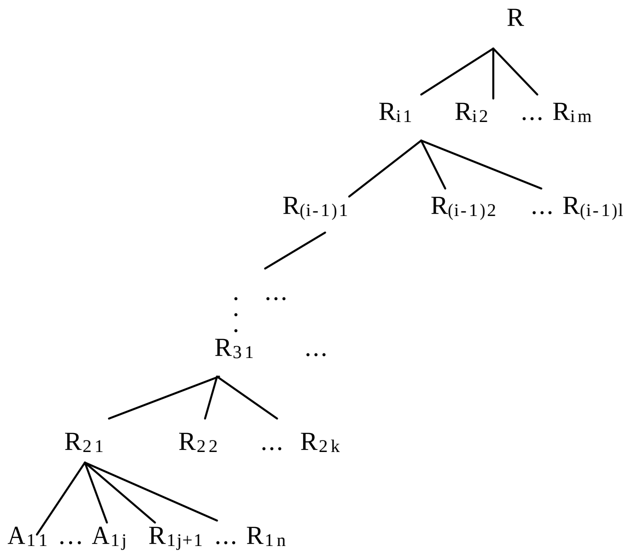
R
R_{i1} R_{i2} … R_{im}
$R_{(i-1)1}$ $R_{(i-1)2}$ … $R_{(i-1)l}$
…
R_{31} …
R_{21} R_{22} … R_{2k}
A_{11} … A_{1j} R_{1j+1} … R_{1n}

Fig. 4.19: The tree representation of relation r

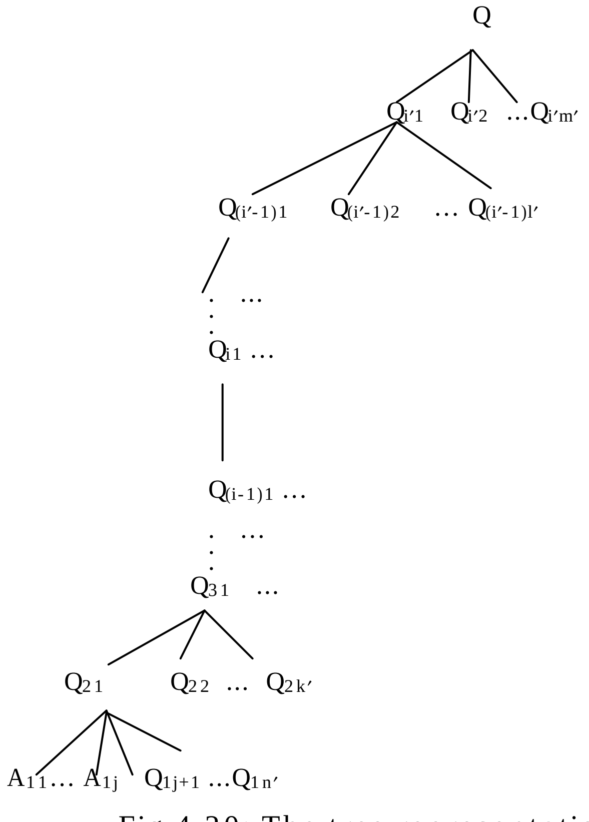
Q
$Q_{i'1}$ $Q_{i'2}$ … $Q_{i'm'}$
$Q_{(i'-1)1}$ $Q_{(i'-1)2}$ … $Q_{(i'-1)l'}$
…
Q_{i1} …
$Q_{(i-1)1}$ …
…
Q_{31} …
Q_{21} Q_{22} … $Q_{2k'}$
A_{11} … A_{1j} Q_{1j+1} … $Q_{1n'}$

Fig.4.20: The tree representation of relation q

For definition purposes the following is assumed: a name is given for all attributes belonging at the same nesting level and that are not part of the join paths in the two relations.

Thus, for relation R:

$R_i = R_{i2} \ldots R_{im}$

$R_{i-1} = R_{(i-1)2} \ldots R_{(i-1)l}$

…

$R_2 = R_{22} \ldots R_{2k}$

$R_1 = R_{1j+1} \ldots R_{1n}$

and similarly for relation Q:

$Q_{i'} = Q_{i'2} \ldots Q_{i'm'}$

$Q_{i'-1} = Q_{(i'-1)2} \ldots Q_{(i'-1)l'}$

…

$Q_{i+1} = Q_{(i+1)2} \ldots Q_{(i+1)p'}$

$Q_i = Q_{i2} \ldots Q_{iq'}$

…

$Q_2 = Q_{22} \ldots Q_{2k'}$

$Q_1 = Q_{1j+1} \ldots Q_{1n'}$

Then, $\triangleright\triangleleft^{\triangleright\triangleleft}$ (rL, qM) is defined as follows:

Definition 4.27:

$$\begin{aligned}\triangleright\triangleleft^{\triangleright\triangleleft}(rL, qM) = \{\, t \mid (\exists\, t_r \in r)\,(\exists\, t_q \in q)\; & ((t[Attr(Q_{i'}(\ldots(Q_{i+1})))] = t_q[Attr(Q_{i'}(\ldots(Q_{i+1})))]) \\ & \wedge (t[Attr(R_i)] = t_r[Attr(R_i)]) \\ & \wedge (t[Attr(Q_i)] = t_q[Attr(Q_i)]) \\ & \wedge (t[R_{i1}Q_{i1}] = \triangleright\triangleleft^{\triangleright\triangleleft}(t_r[R_{i1}]L_{i1},\ t_q[Q_{i1}]M_{i1})))\}\end{aligned}$$

The result relation s has the following scheme:

$$\begin{aligned}S = \{& Q_{i'1}\,(Q_{(i'-1)1}\,(\ldots\,(Q_{(i+1)1}\,(x\,(\ldots\,(z\,(A_{11}\ldots A_{1j}\, R_{1j+1}\ldots Q_{1j+1}\ldots)\, R_{22}\ldots Q_{22}\ldots)\ldots) \\ & R_{i2}\ldots Q_{i2}\ldots)Q_{(i+1)2}\ldots)\ldots)Q_{(i'-1)2}\ldots)Q_{i'2}\ldots\}\end{aligned}$$

The tree representation of the result relation s is given in Fig. 4.21.

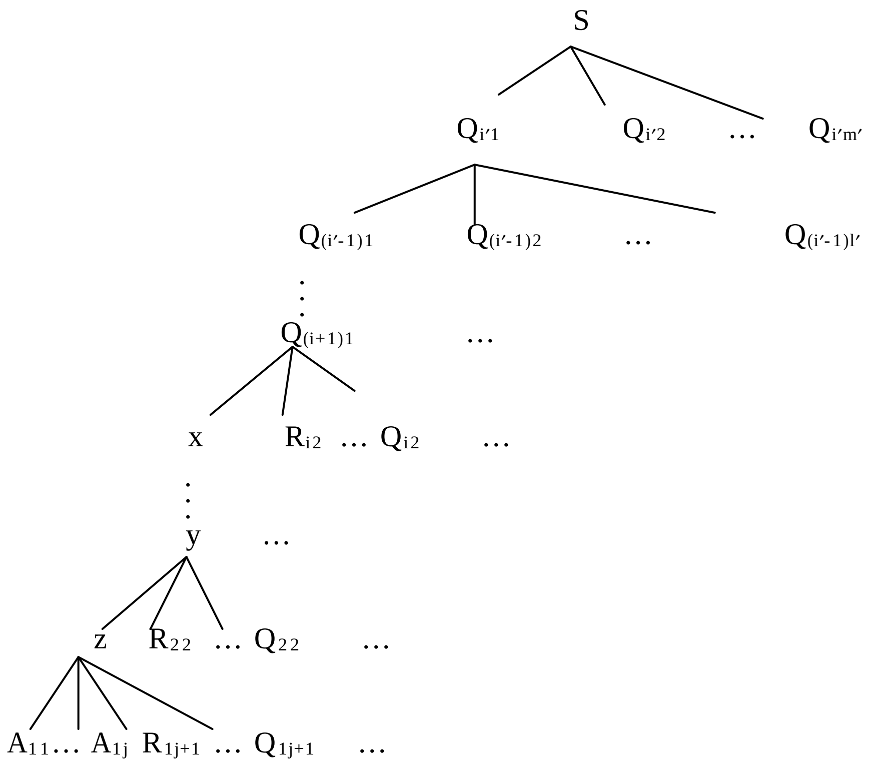

Fig. 4.21: The tree representation of the result relation $s = \bowtie^{\bowtie}(rL, qM)$

The x, y, and z nodes represent nested subrelations and so, conventionally are given names reflecting the attributes which they contain. Thus, for example $z = (A_{11} \ldots A_{1j}\ R_{1j+1} \ldots R_{1n}\ Q_{1j+1} \ldots Q_{1n'})$.

Example 4.17: Suppose that relations TRAINING_1 (Fig. 2.1) and DEPT_1 (Fig. 2.2) are given. Consider the query: “Retrieve the names of the companies and the years for which trainers have taught courses to technical employees together with the code number of each course”. A natural join operation is required in order to answer this query. The natural join operation is performed according to Definition 4.27. The scheme tree of the result relation of the natural join operation of the two relations DEPT_1 and TRAINING_1 is shown in Fig. 4.22 and the result table in Fig. 4.23.

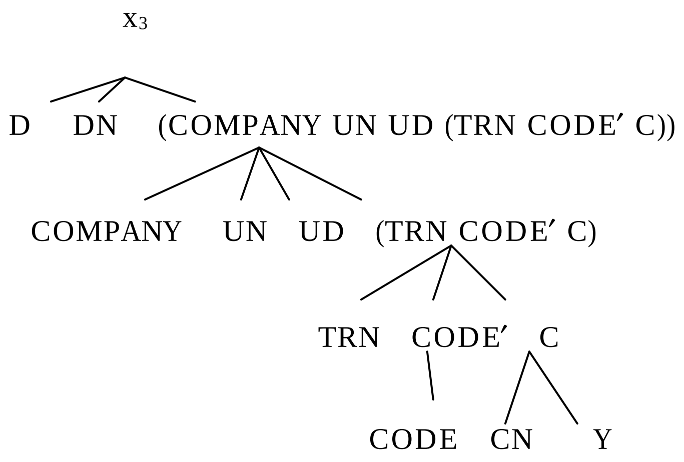

Fig. 4.22: The tree representation of the result relation

$x_3 = \rhd\lhd^{\rhd\lhd}$ (TRAINING_1(PROGRAMME(TRN)), DEPT_1(UNIT(TRAINER(TRN))))

D	DN	(COMPANY UN UD (TRN CODE′ C))						
		COMPANY	UN	UD	(TRN CODE′ C)			
					TRN	CODE′	C	
						CODE	CN	Y
1	Research	Apple	511	Software Engineering	Mark	xy1	1	75
						xy2	2	76
							5	79
		Apple	678	Planning	Mark	xy1	2	76
						xy2	4	82
		IBM	552	Basic Research	Tim	xy1	5	79
						xx2		
		Microsoft	552	Basic Research	Karen	xx1	1	82
							2	79
2	Development	Apple	981	Planning	Jack	xx0	2	81
							3	82
							5	79
		Apple	780	Maintenance	Mark	xy1	2	76
						xy2		
		IBM	780	Maintenance	Tim	xy1	3	82
						xx2		
		Microsoft	650	Design	Karen	xx1	1	75

Fig. 4.23: The result relation

$x_3 = \rhd\lhd^{\rhd\lhd}$ (TRAINING_1(PROGRAMME(TRN)), DEPT_1(UNIT(TRAINER(TRN))))

Case 4: *Join two nested relations which have one or more subrelations at the top level in common*

The natural join definition presented here is a slightly modified version of Colby's natural join definition ([Col90]). The natural join of two relations, which have one or more nested attributes at the top level in common, is a new relation which consists of tuples which share the same values of the common nested attributes and all the other attributes (atomic or nested) have as values the corresponding values of the non-shared attributes of the two relations which participate in the natural join operation. Subsequently, the scheme of the result relation is composed of the nested attributes in common of the two joined relations and all the remaining attributes of the two joined relations which are not in common.

Let r and q be two nested relations with relation schemes $\{R_1, R_2, \ldots, A_1, \ldots, A_j, \ldots, R_n\}$ and $\{Q_1, Q_2, \ldots, A_1, \ldots, A_j, \ldots, Q_m\}$ respectively where $j > 0$ and $n, m \geq j$ and different in general. The two relations r and q have in common the nested attributes $A_1, \ldots, A_j$ at the top level. Then, $\rhd\lhd$ (r, q) is defined as follows:

Definition 4.28:

$$\begin{aligned}\rhd\lhd (r, q) = \{ t \mid (\exists\, t_r \in r)(\exists\, t_q \in q)\ & ((t[Attr(R) - \{A_1, \ldots, A_j\}] = t_r[Attr(R) - \{A_1, \ldots, A_j\}]) \\ & \wedge (t[Attr(Q) - \{A_1, \ldots, A_j\}] = t_q[Attr(Q) - \{A_1, \ldots, A_j\}]) \\ & \wedge (t[A_1, \ldots, A_j] = \rhd\lhd(t_r[A_1, \ldots, A_j], t_q[A_1, \ldots, A_j]) \\ & \qquad = (A_{1r} \cap^{\cap} A_{1q}) \wedge \ldots \wedge (A_{jr} \cap^{\cap} A_{jq})))\}\end{aligned}$$

where $A_{1r}, \ldots, A_{jr}$ are the common nested attributes of relation r and $A_{1q}, \ldots, A_{jq}$ are the common nested attributes of relation q.

The recursive nested intersection of two nested relations has been defined in section 4.3.3 (see Definition 4.11).

Example 4.18: Consider the relations EMPLOYMENT (Fig. 3.10) and PAYMENT (Fig. 3.11) and the query "What is the scale of payment for Anna?" which can be answered by applying the natural join operation to the two relations. The joined attribute is the subrelation JOB which is at the top level in both relations. The result relation is given in Fig. 4.24.

NAME	JOB		SALARY
	COMPANY	JOB_DESCRIPTION	
Anna	TOSHIBA	Secretary	15,500-19,500
Anna	Microsoft	Secretary	18,000-23,000
Paul	Microsoft	Programmer	18,000-23,000
Mark	Apple	Director	25,000-30,000

Fig. 4.24: ⊳⊲ (EMPLOYMENT, PAYMENT)

Case 5: *Join two nested relations which have one or more subrelations in common which in the one relation are subrelations of a subrelation of a relation and in the other relation are at the top level*

This natural join is similar to Case 2 where the join path of one relation is not empty but instead of atomic attributes in common the two relations have subrelations in common. Therefore, the natural join operation is performed as in Case 2, the only difference being that the common attributes A_1 ... A_j in relation R are nested attributes that belong to subrelation R_i while in the other relation Q are the nested attributes A_1 ... A_j at the top level (see Definition 4.25). The result relation consists of all the attributes of the first relation except the subrelation which participates in the natural join operation and the remaining attribute of the result is computed by joining the subrelation of the first relation to the second relation. When the final iteration down the join path is reached, the natural join is performed according to Definition 4.28 where the first relation consists only of the common subrelations.

Case 6: *Join two nested relations which have one or more common subrelations which belong at different subrelations of the two relations (but in the same subrelation in each relation)*

In this case, both relations which participate in the natural join operation have non-empty join paths. As in Case 3, two subcases can be distinguished, depending on whether or not the common subrelations are at the same nesting level in the two joined relations.

Case 6a: *The common subrelations are at the same nesting level in the two joined relations*

In this case, the resulting relation consists of all the attributes of the two relations which do not take part in the natural join operation and a new subrelation is formed by joining the subrelations which contain the subrelations in common, and by applying this method recursively, until the subrelations in common are reached from which only the subtuples which have equal values are selected (intersection operation is applied – see Definition 4.11).

Let r and q be two nested relations with relation schemes R = {R_1, R_2, ..., R_i, ..., R_n} and Q = {Q_1, Q_2, ..., Q_j, ..., Q_m) respectively where $i, j > 0$, $n \geq i$ and $m \geq j$. Suppose that the attributes R_i and Q_j of the two relations r and q respectively are nested and that they contain the common subrelations.

Let L be a join path of R and M be a join path of Q. L is of the form R_iL_i where L_i is a join path of R_i and M is of the form Q_jM_j where M_j is a join path of Q_j. Then, $\rhd\lhd^{\rhd\lhd}$ (rL, qM) is defined as follows:

Definition 4.29:

$$\begin{aligned}\rhd\lhd^{\rhd\lhd}(rL, qM) = \rhd\lhd^{\rhd\lhd}(r(R_iL_i), q(Q_jM_j)) = \{\, t \mid & (\exists\, t_r \in r)\ (\exists\, t_q \in q) \\ & ((t[Attr(R) - \{R_i\}] = t_r[Attr(R) - \{R_i\}]) \\ & \wedge (t[Attr(Q) - \{Q_j\}] = t_q[Attr(Q) - \{Q_j\}]) \\ & \wedge (t[R_iQ_j] = \rhd\lhd^{\rhd\lhd}(t_r[R_i]L_i,\ t_q[Q_j]M_j) \neq \varnothing))\}\end{aligned}$$

As shown above, the different levels of nesting are traversed until the common subrelations are reached. The common subrelations can then be joined using Definition 4.28, with the two relations participating in the natural join operation consisting only of the common subrelations and so this natural join is equal to the intersection of the common subrelations (see also section 4.3.3).

Example 4.19: Consider the example database, which contains two relations, the DEPT_2 relation (Fig. 4.25) and the TRAINING_2 relation (Fig. 4.3). Both relations are modified versions of relations DEPT (Fig. 3.7) and TRAINING (Fig. 3.6) respectively. Relation DEPT_2 has the following scheme: DEPT_2 = D DN UNIT(UN UD C(CN Y)). The tree representation of the relation DEPT_2 is shown in Fig. 4.26. Relation TRAINING_2 has the following scheme:

TRAINING_2 = COMPANY TRAINER(TRN C (CN Y)). The tree representation of the relation TRAINING_2 is given in Fig. 4.27.

Consider the following query: “Find the departments for which Tim has taught courses to their employees”. To answer this query the natural join operation is needed to be performed according to the above definition. The scheme tree of the result relation is shown in Fig. 4.28 and the result table in Fig. 4.29.

D	DN	UNIT			
		UN	UD	C	
				CN	Y
1	Research	511	Software Engineering	1	75
				2	76
				5	79
		552	Basic Research	1	82
				2	79
		678	Planning	2	76
				4	82
2	Development	650	Design	1	75
				2	77
		780	Maintenance	3	82
		981	Planning	2	81
				3	82

Fig. 4.25: DEPT_2

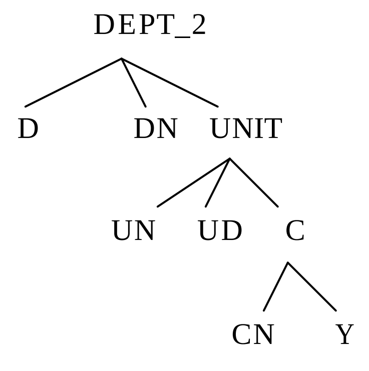

Fig. 4.26: The tree representation of relation DEPT_2

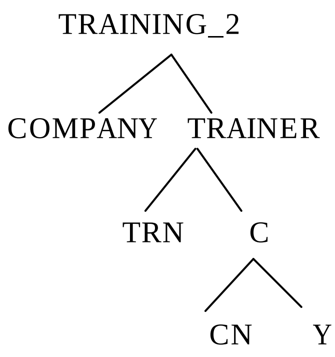

Fig. 4.27: The tree representation of relation TRAINING_2

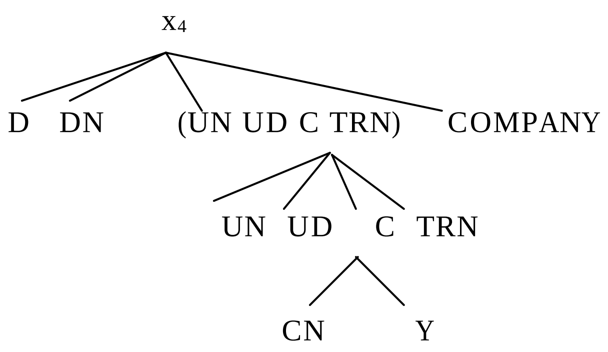

Fig. 4.28: The tree representation of the result relation

$x_4 = \bowtie^{\bowtie}$ (DEPT_2(UNIT(C)), TRAINING_2(TRAINER(C)))

D	DN	(UN UD C TRN)					COMPANY
		UN	UD	C		TRN	
				CN	Y		
1	Research	511	Software Engineering	1	75	Jack	Apple
				2	76		
		552	Basic Research	1	82	Mark	
				2	79		
		678	Planning	2	76	Jack	
1	Research	511	Software Engineering	5	79	Tim	IBM
		678	Planning	4	82	Tim	
2	Development	650	Design	1	75	Jack	Apple
		780	Maintenance	3	82	Mark	
		981	Planning	3	82	Mark	
2	Development	780	Maintenance	3	82	Tim	IBM
		981	Planning	3	82	Tim	
2	Development	650	Design	2	77	Karen	Microsoft
		981	Planning	2	81	Karen	

Fig. 4.29: The result relation

$x_4 = \triangleright\triangleleft^{\triangleright\triangleleft}$ (DEPT_2(UNIT(C)), TRAINING_2(TRAINER(C)))

Case 6b: *The common subrelations are at different nesting levels in the two joined relations*

This case is similar to Case 3b. Once again, the tree representations of the two relations that take part in the natural join operation can help to the designing of the scheme tree of the result relation. The resulting relation is constructed iteratively and consists of the common subrelations together with all its siblings starting from the bottom. One level up, the parent nodes of the subrelations in common in the two joined relations form the parent node of the common subrelations and all its siblings. The siblings of the parent nodes are still siblings of the result parent node. The same procedure is followed until the root of both relations is reached, so that the hierarchy of the joined relations is maintained in the result relation as well.

Let r and q be two nested relations with relation schemes

$R = \{R_{i1}(R_{(i-1)1}(\ldots(R_{21}(A_{11}\ldots A_{1j}\ R_{1j+1}\ldots R_{1n})\ldots R_{2k})\ldots)R_{(i-1)l})\ldots R_{im}\}$ and

$Q = \{Q_{i'1}(Q_{(i''-1)1}(\ldots(Q_{21}(A_{11}\ldots\ A_{1j}\ Q_{1j+1}\ldots Q_{1n'})\ldots Q_{2k'})\ldots)Q_{(i'-1)l'})\ldots Q_{i'm'}\}$ respectively where i, n, k, l, m, j, i′, n′, k′, l′ and m′ are positive integers, not equal in general. The common subrelations are the subrelations A_{11} ... A_{1j}. All the other attributes which do not form the join paths can be atomic or nested but since they do not participate in the natural join operation, this makes no difference. The same assumption is made as in Case 3b, i.e. i) $i \neq i'$, which means that the two relations have, in general, different levels of nesting (in fact, the assumption is $i < i'$ without loss of generality) and ii) the common subrelations A_{11} ... A_{1j} are the first attributes in order of the subrelations in which they belong and these subrelations are the first attributes of the subrelations in which they belong and so on, since the order of the attributes at the same nesting level and in the same subrelation is insignificant.

The formal definition is not given here, since it is the same as Definition 4.27.

Example 4.20: In the following example the DEPT_2 relation (Fig. 4.25) is joined to the TRAINING relation (Fig. 3.6) in order to answer the following query: "Find the departments for which trainers Mark and Karen have taught courses to their employees and the code of these courses". The scheme tree of the result relation is shown in Fig. 4.30 and the result relation in Fig. 4.31.

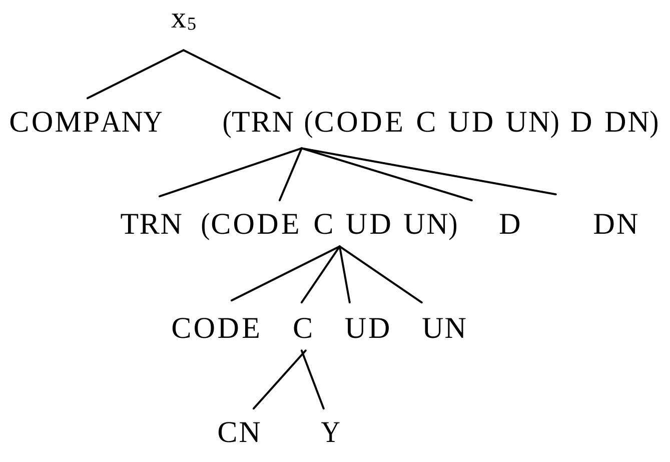

Fig. 4.30: The tree representation of the result relation

$x_5 = \triangleright\triangleleft^{\triangleright\triangleleft}$ (DEPT_2(UNIT(C)),TRAINING(TRAINER(COURSE(C))))

COMPANY	(TRN (CODE C UD UN) D DN)							
	TRN	(CODE C UD UN)					D	DN
		CODE	C		UD	UN		
			CN	Y				
Apple	Jack	xx0	1	75	Software Engineering	511	1	Research
			2	76				
		xx0	2	76	Planning	678		
	Mark	xy1	1	82	Basic Research	552	1	Research
		xy2	2	79	Basic Research	552		
	Jack	xx0	1	75	Design	650	2	Development
	Mark	xy1	3	82	Maintenance	780	2	Development
		xy1	3	82	Planning	981		
IBM	Tim	xx2	5	79	Software Engineering	511	1	Research
		xx2	4	82	Planning	678		
	Tim	xy1	3	82	Maintenance	780	2	Development
		xy1	3	82	Planning	981		
Microsoft	Karen	xx1	2	77	Design	650	2	Development
		xx1	2	81	Planning	981		

Fig. 4.31: The result relation

$x_5 = \triangleright\triangleleft^{\triangleright\triangleleft}$ (DEPT_2(UNIT(C)),TRAINING(TRAINER(COURSE(C))))

4.3.11 The *Recursive Nested Q-Join* Operation ($\triangleright\triangleleft_Q^{\triangleright\triangleleft}$)

The Θ-join operation is a special case of the join operation where the two relations are joined on the basis of some comparison operator other than equality.

It can be expressed by applying a selection operation to the result of the cartesian product operation of two relations. The cartesian product is applied at the top levels of the two nested relations and then, a recursive nested selection operation follows which compares two attributes in the resulting relation. The two attributes need not be at the same nesting level in the resulting relation. The recursive nested selection operation is defined in section 4.3.5.

Let r and q be two nested (in general) relations with relation schemes R and Q respectively. Let also, X and Y be two atomic attributes belonging to relations R and Q respectively and Θ the condition that they must satisfy. Assume, without loss of generality, that Y belongs to a deeper nesting level than X and $L_{\sigma Y' \to Y}$ is the select path of Y starting at node Y′ which is at the same nesting level as X (when X and Y are at the same nesting level the select path is empty). So, the recursive nested Θ–join operation of the two relations r and q is defined as follows:

Definition 4.30 (*Recursive Nested* Q*-Join*)

$r \triangleright\triangleleft_{\Theta}^{\triangleright\triangleleft} q = \sigma^{\sigma}((r \times q)_{X \Theta Y} L_{\sigma Y' \to Y})$

4.3.12 The *Recursive Nested Division* Operation (¸ ˒)

The division operation has not been addressed in any of the previous proposed algebra for nested models as far as the author of this thesis is aware. It is believed that this is due to the following reasons:

1. the division operation is not a primitive operation,
2. it is not often used,
3. it is not implemented in any commercial product,
4. it is, by nature, hard to define.

The division operation can be expressed as a number of projections, differences and a cartesian product operation between two relations.

4.3.13 Functions

Beyond the above relational algebra operations, many authors define additional operations that enable the use of scalar functions ([Lor88]) and aggregate functions ([Klu82], [Tan86]).

For simplicity reasons, in the present thesis, such functions are incorporated directly within the remainder relational algebra operations. Scalar functions are used in the sequel wherever necessary, but their definitions have been omitted as obvious.

On the other hand, aggregate functions for nested relations have not been discussed in any other model presented in chapter 2, but in [DL91]. Saying that, it is important to mention that, outside the relational world, the object database world and the functional data model may both be capable of dealing with sets of objects [DS85]. However, the whole perspective of the present thesis is completely relational and so, a functional or object data model is out of its scope.

Aggregate functions are redefined below.

Let f be a nested aggregate function (f ? {N-MAX, N-MIN, N-SUM, N-AVG, N-COUNT}, where N-MAX, N-MIN, N-SUM, N-AVG and N-COUNT are the nested versions for the corresponding aggregate functions MAX, MIN, SUM, AVG and COUNT for flat relations), f′ an aggregate function for flat relations (f′ ? {MAX, MIN, SUM, AVG, COUNT}), r a nested relation, X an atomic or nested attribute at a lower nesting level of r, Par the parent attribute of atomic attribute Y of r (Y is at the same or higher nesting level than X and it is the attribute over which attribute X is summarised) and X/Y denotes that attribute X is summarised over attribute Y. Then, f[X/Y](r) is defined as follows:

Definition 4.31 (*Nested Aggregate Function*)

$f[X/Y](r) = f'(\{t_i[X] \mid t_i \ ?\ t,\ t\ ?\ Par(Y) \wedge t_i[X] \neq null\})$

Note: Attribute X can be a nested attribute only when the nested aggregate function f is N-COUNT. For all other cases, X attribute must be an atomic attribute.

For an example see Query 7 in section 6.2 of chapter 6.

4.4 Summary

In this chapter, a database model and algebra have been defined for nested relations of arbitrary nesting levels.

All the operators have been recursively defined. As a result, there is no need to flatten the nested relations when a series of operations are executed and so the data redundancy and duplication caused by unnesting relations is avoided. Furthermore, the representation of the data is claimed to be in a "natural form", since even complex objects can be modelled in one relation and thus, it is easier for users to understand when working with the data.

A detailed presentation and definitions of the rename and natural join operations for nested relations have been included in this chapter. In particular, a systematic review of the various forms of natural join between nested relations and subrelations is given. Six distinct cases of natural join have been analysed according to the positions and types of the common attributes that participate in the natural join operation.

NRM is the tool that is going to be used to build the temporal nested model in the next chapter. However, by itself, it provides a complete model for nested relations.

CHAPTER 5

THE TEMPORAL NESTED MODEL (TNM)

5.1 Introduction

The Temporal Nested Model (TNM) is defined in this chapter, as an extension of the NRM presented in chapter 4. Relations can be nested to any finite depth as in the NRM. In the general case, time is represented as temporal elements that form temporal attributes, which together with the corresponding time-varying attributes form temporal nested attributes. Therefore, the temporal dimension of the model is nested and is not integral with the corresponding time-dependent value as in other previous proposed temporal nested models (e.g. [Tan97]). As a result, the full power of the nested model is gained and simultaneously temporal elements can be readily referenced with or without their associated time-varying attribute values.

All the operations of the algebra for the TNM are defined recursively. In particular, a detailed definition of the natural join operation for temporal nested relations is presented where different cases are examined. These cases are distinguished by the types (atomic, temporal, nested or temporal nested attributes) and the nesting levels of the common attributes that participate in the natural join operation.

A formal syntax of the TNM algebra is also given in Appendix A.

Finally, the operations of the TNM are proved to be closed.

5.2 Representation of TNM Relations

A relation in the TNM is a temporal nested relation which can be represented either in a tabular representation (see Fig. 3.13-3.17) or in a tree representation. Specifically, for the tree representation, a relation R in the

TNM can be described as a tree with root node R and with all the nested and temporal nested attributes, R_n and R_{tn} respectively, as non-leaf nodes of the tree and all the atomic and temporal attributes, R_a and R_t respectively, as leaves of the tree. In order for the attributes R_a and R_t to be distinguished, all the temporal attributes have a special indication on their names, the subscript $_t$. Thus, it is easy to distinguish atomic and temporal attributes between themselves and treat them differently (see the algebraic operations of the TNM in section 5.3).

An example follows where a tree representation of a temporal nested relation is given.

Example 5.1: The tree structure of relation T_LOCATION (Fig. 3.15) is shown in Fig. 5.1.

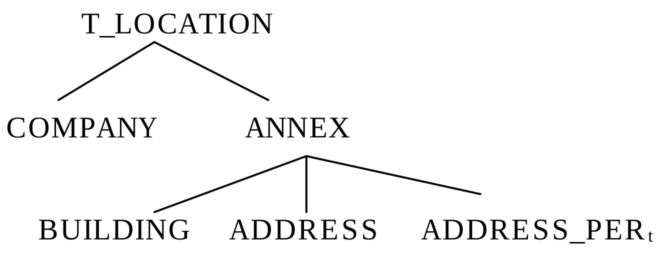

Fig. 5.1: The tree representation of relation T_LOCATION

5.3 Operations in the TNM

In this section, the operations of the algebra of the NRM, which were defined in section 4.3, are extended to support temporal data.

In the general case, temporal data are represented as temporal attributes connected to the corresponding time-varying attributes. Each time-varying attribute together with the corresponding temporal attribute form a temporal nested attribute.

The operations of the algebra that are defined next are named according to their common names with the prefix “T” to denote their temporal version.

A formal syntax of all the TNM operations is given in Appendix A. Formally, this defines well-formed formulae (WFF), i.e. formulae that are grammatically correct using BNF grammar ([Gra84]).

For sections 5.3.1, 5.3.2 and 5.3.3, let r and q be two temporal nested relations with the same relation scheme R. Let Attr(R_a) be all the atomic attributes, Attr(R_n) all the nested attributes that do not contain temporal attributes, Attr(R_t) all the temporal attributes, Attr(R_{tn}) all the nested attributes that contain temporal attributes of R and Attr(S) the set of all the atomic attributes and all the key temporal, nested and temporal nested attributes of R and Q. Let t_r be a tuple in relation r, t_q a tuple in relation q and t a tuple in the result relation.

5.3.1 The *Recursive Temporal Nested Union* Operation ($\text{È}_{\mathbf{t}}^{\text{È}}$)

The union of two temporal nested relations r and q, r $\cup_t^{\cup}$ q, is a new temporal nested relation with identical headings to relations r and q, consisting of all tuples appearing in either or both of the two relations and, in addition, for all those tuples with the same values for all the atomic attributes, the temporal elements for all the temporal attributes are computed by taking the unions of the temporal elements of the corresponding temporal attributes of the two relations r and q (see Definition 3.10).

Then, the recursive temporal nested union of the two relations r and q, r $\cup_t^{\cup}$ q, can be formally defined as follows:

Definition 5.1 (*TUnion*)

i) Non-recursive union for temporal flat relations (r $\cup_t$ q)

$$\begin{aligned} r \cup_t q = \{\, t \mid & ((\exists\, t_r \in r)(\exists\, t_q \in q)((t[\text{Attr}(R_a)] = t_r[\text{Attr}(R_a)] = t_q[\text{Attr}(R_a)]) \\ & \qquad \wedge (t[\text{Attr}(R_t)] = t_r[\text{Attr}(R_t)] \cup_{TE} t_q[\text{Attr}(R_t)]))) \\ & \vee ((\exists\, t_r \in r)((t[\text{Attr}(R_a)] = t_r[\text{Attr}(R_a)]) \wedge (t[\text{Attr}(R_t)] = t_r[\text{Attr}(R_t)]))) \\ & \vee ((\exists\, t_q \in q)((t[\text{Attr}(R_a)] = t_q[\text{Attr}(R_a)]) \wedge (t[\text{Attr}(R_t)] = t_q[\text{Attr}(R_t)])))\} \end{aligned}$$

ii) Recursive union for temporal nested relations (r $\cup_t^{\cup}$ q)

$$\begin{aligned} r \cup_t^{\cup} q = \{\, t \mid & (\exists\, t_r \in r)(\exists\, t_q \in q)((t[\text{Attr}(S)] = t_r[\text{Attr}(S)] \cup t_q[\text{Attr}(S)]) \\ & \wedge (t[\text{Attr}(R_t)] = t_r[\text{Attr}(R_t)] \cup_{TE} t_q[\text{Attr}(R_t)]) \\ & \wedge (t[\text{Attr}(R_n)] = t_r[\text{Attr}(R_n)] \cup^{\cup} t_q[\text{Attr}(R_n)]) \\ & \wedge (t[\text{Attr}(R_{tn})] = t_r[\text{Attr}(R_{tn})] \cup_t^{\cup} t_q[\text{Attr}(R_{tn})]))\} \end{aligned}$$

Example 5.2: The two tables T_TRAINING (Fig. 5.2) and T_TRAINING_1 (Fig. 5.3) are given. In both relations, Attr(S) = COMPANY. The semantics of these two tables are given in subsection 3.5.2 of chapter 3. Please note that relation T_TRAINING is the same as that of Fig. 3.13. However, the reason for this repetition is to simplify the reading of this specific example. Fig. 5.4 shows the result table of the TUnion of these two relations.

COMPANY	**TRAINER**		
	TRN	**COURSE**	
		CN	**CN_PER$_t$**
Apple	Jack	5.2	[2/11/1994, 25/4/1995) $\cup$ [7/8/1996, 1/1/2010)
	Mark	3.3	[2/1/1992, 8/11/1996)
		3.5	[30/4/1995, 1/1/2010)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
		5.0	[17/12/1995, 1/1/2010)
Microsoft	Karen	3.3	[25/6/1996, 1/1/2010)

Fig. 5.2: T_TRAINING

COMPANY	**TRAINER**		
	TRN	**COURSE**	
		CN	**CN_PER$_t$**
Apple	Mark	3.3	[10/10/1993, 1/1/2010)
		3.7	[8/10/1992, 15/5/1994)
IBM	Mark	4.1	[1/9/1995, 1/1/2010)
		5.5	[13/8/1996, 28/7/1998)
Microsoft	Karen	3.3	[5/7/1997, 18/3/1998)

Fig. 5.3: T_TRAINING_1

COMPANY	**TRAINER**		
	TRN	**COURSE**	
		CN	**CN_PER$_t$**
Apple	Jack	5.2	[2/11/1994, 25/4/1995) $\cup$ [7/8/1996, 1/1/2010)
	Mark	3.3	[2/1/1992, 1/1/2010)
		3.5	[30/4/1995, 1/1/2010)
		3.7	[8/10/1992, 15/5/1994)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
		5.0	[17/12/1995, 1/1/2010)
	Mark	4.1	[1/9/1995, 1/1/2010)
		5.5	[13/8/1996, 28/7/1998)
Microsoft	Karen	3.3	[25/6/1996, 1/1/2010)

Fig. 5.4: T_TRAINING $\cup_t^{\cup}$ T_TRAINING_1

5.3.2 The *Recursive Temporal Nested Difference* Operation ($-_t^-$)

The difference of two temporal nested relations r and q, r $-_t^-$ q, is a new temporal nested relation with identical headings to relations r and q, consisting of all tuples appearing in relation r but not in relation q and in addition, for all those tuples with the same values for all the atomic attributes, the temporal elements for all the temporal attributes are computed by taking the differences of the temporal elements of the corresponding temporal attributes of the two relations r and q (see Definition 3.11). In the resulting relation, tuples having empty temporal elements must be discarded. Then, the recursive temporal nested difference of the two relations r and q, r $-_t^-$ q, can be formally defined as follows:

Definition 5.2 (*TDifference*)

i) Non-recursive difference for temporal flat relations (r $-_t$ q)

$$\begin{aligned}
r -_t q = \{\, t \mid\ & ((\exists\, t_r \in r)(\forall\, t_q \in q)\,((t[Attr(R_a)] = t_r[Attr(R_a)]) \\
& \qquad \wedge (t[Attr(R_a)] \neq t_q[Attr(R_a)]) \\
& \qquad \wedge (t[Attr(R_t)] = t_r[Attr(R_t)]))) \\
& \vee ((\exists\, t_r \in r,\ \exists\, t_q \in q)\,((t[Attr(R_a)] = t_r[Attr(R_a)] = t_q[Attr(R_a)]) \\
& \qquad \wedge (t[Attr(R_t)] = (t_r[Attr(R_t)] -_{TE} t_q[Attr(R_t)]) \neq \emptyset)))\}
\end{aligned}$$

ii) Recursive difference for temporal nested relations (r $-_t^-$ q)

$$\begin{aligned}
r -_t^- q = \{\, t \mid\ & ((\exists\, t_r \in r)(\forall\, t_q \in q)\,((t[Attr(S)] = t_r[Attr(S)] - t_q[Attr(S)]) \\
& \qquad \wedge (t[Attr(R) - Attr(S)] = t_r[Attr(R) - Attr(S)]))) \\
& \vee ((\exists\, t_r \in r)(\exists\, t_q \in q)\,((t[Attr(S)] = t_r[Attr(S)] = t_q[Attr(S)]) \\
& \qquad \wedge (t[Attr(R_t)] = (t_r[Attr(R_t)] -_{TE} t_q[Attr(R_t)]) \neq \emptyset) \\
& \qquad \wedge (t[Attr(R_n)] = t_r[Attr(R_n)]) \\
& \qquad \wedge (t[Attr(R_{tn})] = t_r[Attr(R_{tn})]))) \\
& \vee ((\exists\, t_r \in r,\ \exists\, t_q \in q)\,((t[Attr(S)] = t_r[Attr(S)] = t_q[Attr(S)]) \\
& \qquad \wedge (t[Attr(R_t)] = t_r[Attr(R_t)] = t_q[Attr(R_t)]) \\
& \qquad \wedge (t[Attr(R_n)] = t_r[Attr(R_n)] -^- t_q[Attr(R_n)]) \\
& \qquad \wedge (t[Attr(R_{tn})] = t_r[Attr(R_{tn})] -_t^- t_q[Attr(R_{tn})])))\}
\end{aligned}$$

Example 5.3: The TDifference of relations T_TRAINING (Fig. 5.2) and T_TRAINING_1 (Fig. 5.3) is shown in Fig. 5.5.

COMPANY	TRAINER		
	TRN	COURSE	
		CN	CN_PER$_t$
Apple	Jack	5.2	[2/11/1994, 25/4/1995) ∪ [7/8/1996, 1/1/2010)
	Mark	3.3	[2/1/1992, 10/10/1993)
		3.5	[30/4/1995, 1/1/2010)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
		5.0	[17/12/1995, 1/1/2010)
Microsoft	Karen	3.3	[25/6/1996, 5/7/1997) ∪ [18/3/1998, 1/1/2010)

Fig. 5.5: T_TRAINING $-_t^-$ T_TRAINING_1

5.3.3 The *Recursive Temporal Nested Intersection* Operation (Ç$_t^Ç$)

The intersection of two temporal nested relations r and q, $r \cap_t^{\cap} q$, is a new temporal nested relation with identical headings to relations r and q, consisting of all tuples appearing in both of the relations r and q and in addition, for all those tuples with the same values for all the atomic attributes, the temporal elements for all the temporal attributes are computed by taking the intersections of the temporal elements of the corresponding temporal attributes of the two relations r and q (see Definition 3.12). In the resulting relation, tuples having empty temporal elements must be discarded. Then, the recursive temporal nested intersection of the two relations r and q, $r \cap_t^{\cap} q$, is defined as follows:

Definition 5.3 (*TIntersection*)

i) Non-recursive intersection for temporal flat relations ($r \cap_t q$)

$$r \cap_t q = \{\, t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ ((t[Attr(R_a)] = t_r[Attr(R_a)] = t_q[Attr(R_a)]) \wedge (t[Attr(R_t)] = (t_r[Attr(R_t)] \cap_{TE} t_q[Attr(R_t)]) \neq \varnothing))\}$$

ii) Recursive intersection for temporal nested relations ($r \cap_t^{\cap} q$)

$$r \cap_t^{\cap} q = \{\, t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ ((t[Attr(S)] = t_r[Attr(S)] \cap t_q[Attr(S)])$$
$$\wedge\ (t[Attr(R_t)] = (t_r[Attr(R_t)] \cap_{TE} t_q[Attr(R_t)]) \neq \emptyset)$$
$$\wedge\ (t[Attr(R_n)] = t_r[Attr(R_n)] \cap^{\cap} t_q[Attr(R_n)])$$
$$\wedge\ (t[Attr(R_{tn})] = t_r[Attr(R_{tn})] \cap_t^{\cap} t_q[Attr(R_{tn})]))\}$$

Example 5.4: The TIntersection of relations T_TRAINING (Fig. 5.2) and T_TRAINING_1 (Fig. 5.3) is shown in Fig. 5.6.

COMPANY	TRAINER		
	TRN	COURSE	
		CN	CN_PER$_t$
Apple	Mark	3.3	[10/10/1993, 8/11/1996)
Microsoft	Karen	3.3	[5/7/1997, 18/3/1998)

Fig. 5.6: T_TRAINING $\cap_t^{\cap}$ T_TRAINING_1

5.3.4 The *Recursive Temporal Nested Projection* Operation (p_t^{P})

The projection operator gives as a result a "vertical" subset of a given relation. TProjection is similar to the recursive nested projection operation. It can be expressed at all levels without restructuring. In the resulting relation, tuples having the same values for all the atomic attributes are coalesced, by taking the unions of the temporal elements of their corresponding temporal attributes (see Definition 3.10).

Let r be a temporal nested relation with relation scheme R. Let $Attr(R_a) = \{R_{a1}, \ldots, R_{ak}\}$ be the subset of all the atomic attributes of R which are going to be projected, $Attr(R_n) = \{R_{n1}, \ldots, R_{nm}\}$ the subset of all the nested attributes of R which are going to be projected either fully or attributes belonging to these nested attributes with $L_{n1}, \ldots, L_{nm}$ their project paths respectively, either empty or not, and $Attr(R_t)$ the subset of all the temporal attributes of R which are going to be projected. Let $Attr(R_{tn}) = \{R_{tn1}, \ldots, R_{tnm'}\}$ be the subset of all the temporal nested attributes of R which are going to be projected either fully or attributes belonging to these temporal nested attributes ($L_{tn1}, \ldots, L_{tnm'}$ are the project paths of attributes $R_{tn1}, \ldots, R_{tnm'}$ respectively, either empty or not). Atomic and temporal attributes behave the same way for the projection operation.

Then, the recursive temporal nested projection in the relation r, $\pi_t^{\pi}(rL_{\pi})$, where L_{π} is a project list of R (see Definition 4.12), t_r a tuple in relation r and t a tuple in the resulting relation, is defined as follows:

Definition 5.4 (*TProjection*)

i) Projection of the whole temporal nested relation ($\pi_t(r)$)

$$\pi_t(r) = r$$

ii) Non-recursive projection of a temporal nested attribute R_{tn} at the top level of R consisting of subsets $Attr(R_a)$, $Attr(R_n)$ and $Attr(R_t)$ of atomic, nested and temporal attributes respectively at its top nesting level ($\pi_t(r(R_{tn}))$)

$$\begin{aligned}\pi_t(r(R_{tn})) = \{\, t \mid (\exists\, t_r \in r)\ ((t[Attr(R_a)] = t_r[Attr(R_a)])\\ \wedge\ (t[Attr(R_n)] = t_r[Attr(R_n)])\\ \wedge\ (t[Attr(R_t)] = t_r[Attr(R_t)]))\}\end{aligned}$$

iii) Recursive projection for temporal nested relations ($\pi_t^{\pi}(rL_{\pi})$)

$$\begin{aligned}\pi_t^{\pi}(rL_{\pi}) = \pi_t^{\pi}(r(R_{a1}, \ldots, R_{ak}, Attr(R_t), R_{n1}L_{n1}, \ldots, R_{nm}L_{nm}, R_{tn1}L_{tn1}, \ldots, R_{tnm'}L_{tnm'}))=\\ \{\, t \mid (\exists\, t_r \in r)\ ((t[R_{a1}] = t_r[R_{a1}]) \wedge \ldots \wedge (t[R_{ak}] = t_r[R_{ak}])\\ \wedge\ (t[Attr(R_t)] = t_r[Attr(R_t)])\\ \wedge\ (t[R_{n1}] = \pi^{\pi}(t_r[R_{n1}]L_{n1})) \wedge \ldots \wedge (t[R_{nm}] = \pi^{\pi}(t_r[R_{nm}]L_{nm}))\\ \wedge\ (t[R_{tn1}] = \pi_t^{\pi}(t_r[R_{tn1}]L_{tn1})) \wedge \ldots \wedge (t[R_{tnm'}] = \pi_t^{\pi}(t_r[R_{tnm'}]L_{tnm'}))))\}\end{aligned}$$

Example 5.5: The result relation of the TProjection operation of attributes COMPANY and COURSE in relation T_TRAINING (Fig. 5.2) is shown in Fig. 5.7.

COMPANY	COURSE	
	CN	CN_PER_t
Apple	5.2	[2/11/1994, 25/4/1995) ∪ [7/8/1996, 1/1/2010)
	3.3	[2/1/1992, 8/11/1996)
	3.5	[30/4/1995, 1/1/2010)
IBM	5.2	[19/3/1997, 21/4/1997)
	5.0	[17/12/1995, 1/1/2010)
Microsoft	3.3	[25/6/1996, 1/1/2010)

Fig. 5.7: π_t^{π}(T_TRAINING(COMPANY, TRAINER(COURSE)))

5.3.5 The *Recursive Nested TimeSlice* Operation (s^s)

TProjection cannot be used to “project” a relation along a given temporal element, i.e. how a relation looks like at a given temporal element. A new operation needs to be defined, namely TimeSlice, which takes the intersection of the given temporal element and each temporal element of the relation (see Definition 3.12). In the resulting relation, tuples having empty temporal elements are not considered. The TimeSlice operation is similar to the Slice operation proposed by Tansel ([Tan86]).

Let r be a temporal nested relation with relation scheme R. Let, also, Attr($R_{a,n}$) be all the atomic and nested attributes, Attr(R_{tn}) = {R_{tn1}, ..., R_{tnm}} all the temporal nested attributes at the top level of R, Attr(R_t) all the temporal attributes, TE a temporal element, t_r a tuple in relation r and t a tuple in the result relation.

Definition 5.5 (*TimeSlice*)

i) Non-recursive timeslice for a temporal nested attribute R_{tn} (with Attr($R_{a,n}$) and Attr(R_t) at the top level) of a temporal nested relation r along a given temporal element TE ($s_{TE}(t[R_{tn}])$)

$$s_{TE}(t[R_{tn}]) = \{\ t \mid (\exists\ t_r \in r)\ ((t[Attr(R_{a,n})] = t_r[Attr(R_{a,n})]) \\ \wedge\ ((t[Attr(R_t)] = t_r[Attr(R_t)] \cap_{TE} TE)\ ?\ \varnothing))\}$$

ii) Recursive timeslice for temporal nested relations along a given temporal element TE ($s^s_{TE}(r)$)

$$s^s_{TE}(r) = \{\ t \mid (\exists\ t_r \in r)\ ((t[Attr(R_{a,n})] = t_r[Attr(R_{a,n})]) \\ \wedge\ ((t[Attr(R_t)] = t_r[Attr(R_t)] \cap_{TE} TE)\ ?\ \varnothing) \\ \wedge\ (t[R_{tn1}] = s^s_{TE}(t_r[R_{tn1}])) \wedge \ldots \wedge (t[R_{tnm}] = s^s_{TE}(t_r[R_{tnm}]))))\}$$

Example 5.6: In Fig. 5.8 the TimeSlice of relation T_TRAINING (Fig. 5.2) for the temporal element [3/12/1996, 28/9/1997) is shown.

COMPANY	TRAINER		
	TRN	COURSE	
		CN	CN_PER$_t$
Apple	Jack	5.2	[3/12/1996, 28/9/1997)
	Mark	3.5	[3/12/1996, 28/9/1997)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
		5.0	[3/12/1996, 28/9/1997)
Microsoft	Karen	3.3	[3/12/1996, 28/9/1997)

Fig. 5.8: $s^s_{[3/12/1996,\ 28/9/1997)}$(T_TRAINING)

5.3.6 The *Recursive Temporal Nested Selection* Operation (s_t^s)

The selection operation extracts specified tuples from a given relation r that satisfy a specified condition. In addition to the standard selection operation, TSelection can use special temporal comparison operators when the specified condition involves relative positions among temporal elements or among time points and temporal elements (i.e. BEFORE, AFTER, MEETS, OVERLAPS, COVERS defined in [Lor88]).

Let r be a temporal nested relation with relation scheme R, Attr(R) be all the attributes of R and let Attr(R_a) be the subset of all the atomic attributes, Attr(R_t) the subset of all the temporal attributes, Attr(R_n) the subset of all the nested attributes and Attr(R_{tn}) the subset of all temporal nested attributes of R that participate in the selection operation. Let also c be a set of conditions in R, which is of the form {c_a, c_t, c_n, c_{tn}}, where c_a is a set of conditions which must be true for the subset Attr(R_a), c_t a set of conditions which must be true for the subset Attr(R_t), c_n a set of conditions which must be true for the subset Attr(R_n) and c_{tn} a set of conditions which must be true for the subset Attr(R_{tn}). As has been mentioned above, c_t can be a set of predicates for temporal elements. Furthermore, it is important to notice that c_{tn} concerns, eventually, the lowest nesting level of atomic and temporal attributes, since temporal nested attributes consist of atomic and temporal attributes. Then, the recursive temporal nested selection of relation r, where L_σ is a select list of r (see Definition 4.14), t_r a tuple in relation r and t a tuple in the resulting relation is defined as follows:

Definition 5.6 (*TSelection*)

i) Non-recursive selection concerning the set of temporal attributes Attr(R_t) that occur at the top level of a temporal nested relation r ($\sigma_t(r_{ct})$)

$$\sigma_t(r_{ct}) = \{\ t \mid (\exists\ t_r \in r)\ ((t[Attr(R) - Attr(R_t)] = t_r[Attr(R) - Attr(R_t)])$$
$$\wedge\ (t[Attr(R_t)] = t_r[Attr(R_t)]) \wedge (c_t = true))\}$$

ii) Recursive selection concerning the set of temporal nested attributes Attr(R_{tn}) ($\sigma_t^\sigma(r_{c_{tn1}, \ldots, c_{tnm'}} L'_\sigma)$)

$$\sigma_t^\sigma(r_{c_{tn1}, \ldots, c_{tnm'}} L'_\sigma) = \{\, t \mid (\exists\, t_r \in r)$$
$$((t[Attr(R) - Attr(R_{tn})] = t_r[Attr(R) - Attr(R_{tn})])$$
$$\wedge\ (t[R_{tn1}] = \sigma_t^\sigma(t_r[R_{tn1}]_{c_{tn1}} L_{tn1}) \neq \emptyset)$$
$$\wedge \ldots \wedge\ (t[R_{tnm'}] = \sigma_t^\sigma(t_r[R_{tnm'}]_{c_{tnm'}} L_{tnm'}) \neq \emptyset))\}$$

iii) Recursive selection for temporal nested relations ($\sigma_t^\sigma(r_c L_\sigma)$)

$$\sigma_t^\sigma(r_c L_\sigma) = \sigma(r_{ca1, \ldots, cak}) \cap \sigma_t(r_{ct}) \cap \sigma^\sigma(r_{cn1, \ldots, cnm} L_\sigma) \cap \sigma_t^\sigma(r_{c_{tn1}, \ldots, c_{tnm'}} L'_\sigma)$$

Example 5.7: Assume that the following query is given: "Find all information for trainers Mark and Tim and for courses that took place for period overlapping the time interval [1/1/1997, 1/1/1998) from the T_TRAINING table (Fig. 5.2). The result table is given in Fig. 5.9.

COMPANY	TRAINER		
	TRN	COURSE	
		CN	CN_PER$_t$
Apple	Mark	3.5	[30/4/1995, 1/1/2010)
IBM	Tim	5.2	[19/3/1997, 21/4/1997)
		5.0	[17/12/1995, 1/1/2010)

Fig. 5.9: σ_t^σ(T_TRAINING $_{\text{(TRAINER(TRN)='Mark' OR 'Tim') AND (TRAINER(COURSE(CN\_PER}_t\text{)) OVERLAPS [1/1/1997, 1/1/1998))}}$)

5.3.7 The *Recursive Temporal Unnest* Operation (μ_t^μ)

The definition of the recursive unnest operation for temporal nested relations is the same as the definition of the recursive unnest operation (μ^μ) for nested relations (see Definition 4.17 in section 4.3.6).

5.3.8 The *Recursive Temporal Nest* Operation ($?_t^?$)

The definition of the recursive nest operation for temporal nested relations is the same as the definition of the recursive nest operation ($?^?$) for nested relations (see Definition 4.19 in section 4.3.7).

5.3.9 The *Recursive Temporal Nested Rename* Operation (r_t^r)

The definition of the recursive rename operation for temporal nested relations is the same as the recursive rename operation for nested relations (see Definition 4.20 in section 4.3.8).

5.3.10 The *Recursive Temporal Nested Cartesian Product* Operation ($´_t^{´}$)

Let r and q be two temporal nested relations with relation schemes R and Q respectively. The TCartesianProduct of r and q is a new temporal nested relation consisting of all possible combinations of tuples of the two relations. Attributes of relation Q can be placed either next to the last attribute of relation R, which means that the cartesian product operation operates at the top level of both relations and so it is exactly the same as the standard cartesian product operation for flat relations or next to an attribute which is not at the top nesting level of relation R which means that the cartesian product operates between a lower nesting level of relation R and the top nesting level of relation Q.

Let Attr(R) and Attr(Q) be all attributes (atomic, temporal, nested and temporal nested) of R and Q respectively. Let also, L be a join path of R (see Definition 4.21), R_{tn} a temporal nested attribute of R, L_{tn} a join path of attribute R_{tn}, t_r a tuple in relation r, t_q a tuple in relation q and t a tuple in the resulting relation.

Then, the cartesian product of the two relations r and q, $\times_t^{\times}$ (rL, q), is defined as follows:

Definition 5.7 (*TCartesianProduct*)

i) Non-recursive cartesian product between two temporal nested relations r and q when L is empty ($\times_t$ (r, q))

$$\times_t (r, q) = \{ t \equiv (t[Attr(R)], t[Attr(Q)]) \mid (\exists\, t_r \in r, \exists\, t_q \in q) ((t[Attr(R)] = t_r[Attr(R)]) \wedge (t[Attr(Q)] = t_q[Attr(Q)]))\}$$

ii) Recursive cartesian product between two temporal nested relations r and q when L is not empty ($\times_t^{\times}$ (rL, q))

$$\times_t^{\times} (rL, q) = \times_t^{\times} (r(R_{tn}L_{tn}), q) \equiv \times_t^{\times} (q, r(R_{tn}L_{tn}))$$
$$= \{ t \mid (\exists\, t_r \in r) ((t[Attr(R) - \{R_{tn}\}] = t_r[Attr(R) - \{R_{tn}\}])$$
$$\wedge (t[R_{tn}] = \times_t^{\times} (t_r[R_{tn}]L_{tn}, q)))\}$$

It follows from the formal definition of the recursive temporal nested cartesian product operation that the result relation of the cartesian product of temporal nested relations r and q consists of the attributes of relation r plus the attributes of relation q.

Note: The definition of the TCartesianProduct operation for temporal nested relations (Definition 5.7) is the same as the recursive cartesian product definition for nested relations (see Definition 4.22), since temporal attributes of the two relations that participate in the TCartesianProduct operation are not compared by definition but behave like standard atomic attributes. Furthermore, the commutative property is always valid in the recursive temporal nested cartesian product operation, as is the case in the recursive nested cartesian product operation.

5.3.11 The *Recursive Temporal Nested Natural Join* Operation ($\triangleright\triangleleft_t^{\triangleright\triangleleft}$)

In general, the join operation is a special case of the cartesian product operation between two relations where the tuples of the two relations contributing to any given combination, satisfy some specified condition (selection operation). When the specified condition includes time-varying attributes, the intersections of the temporal elements of the corresponding temporal attributes of the two relations are computed (see Definition 3.12). If the result of the intersection is the empty set, the result tuple is discarded. In what is described below the specified condition is equality, i.e. a natural join operation can be performed only when the two temporal nested relations which participate in the join operation have one or more attributes in common. Since the attributes in common can be atomic, temporal, nested or temporal nested either at the top or at a lower nesting level in the two relations, different cases of natural join have to be examined.

The cases can be grouped according to the join paths of the two relations that are going to be joined and are, in general, the same as for the non-temporal nested relations presented in Fig. 4.17.

For all the six different cases below, t_r is a tuple in relation r, t_q a tuple in relation q and t a tuple in the resulting relation.

Case 1: *Join two temporal nested relations which have one or more atomic and temporal attributes at the top level in common*

Definition 5.8: Let r and q be two temporal nested relations with relation schemes R and Q respectively and Attr(R) and Attr(Q) the sets of all attributes of R and Q respectively. The two relations r and q have in common the subsets of atomic and temporal attributes $Attr(R_a)$ and $Attr(R_t)$ respectively. Then, $\rhd\lhd_t$ (r, q) is defined as follows:

$$\begin{aligned}\rhd\lhd_t (r, q) = \{\, t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ & ((t[Attr(R) - \{Attr(R_a), Attr(R_t)\}] = t_r[Attr(R) - \{Attr(R_a), Attr(R_t)\}]) \\ & \wedge (t[Attr(Q) - \{Attr(R_a), Attr(R_t)\}] = t_q[Attr(Q) - \{Attr(R_a), Attr(R_t)\}]) \\ & \wedge (t[Attr(R_a)] = t_r[Attr(R_a)] = t_q[Attr(R_a)]) \\ & \wedge (t[Attr(R_t)] = (t_r[Attr(R_t)] \cap_{TE} t_q[Attr(R_t)]) \neq \emptyset))\}\end{aligned}$$

Case 2: *Join two temporal nested relations having one or more atomic and temporal attributes in common which in one relation are attributes of a subrelation of the relation and in the other are at the top level*

Definition 5.9: Let r and q be two temporal nested relations with relation schemes $R = \{R_1, R_2, \ldots, R_{tn}, \ldots, R_k\}$ and $Q = \{Q_1, Q_2, \ldots, Attr(Q_a), Attr(Q_t), \ldots, Q_{k'}\}$ respectively, where $k, k' > 0$ and $k \neq k'$ (in general). Let also, Attr(R) and Attr(Q) be the sets of all attributes of relations r and q respectively. The two relations have in common one or more atomic and temporal attributes which in one relation belong to temporal nested attribute R_{tn} and in the other relation are the subsets $Attr(Q_a)$ and $Attr(Q_t)$ of atomic and temporal attributes respectively. Let L be a join path of R. L is of the form $R_{tn}L_{tn}$ where L_{tn} is a join path of R_{tn}. Then, $\rhd\lhd_t^{\rhd\lhd}$ (rL, q) is defined as follows:

$$\begin{aligned}\rhd\lhd_t^{\rhd\lhd} (rL, q) = \rhd\lhd_t^{\rhd\lhd} (q, rL) = \rhd\lhd_t^{\rhd\lhd} (r(R_{tn}L_{tn}), q) = \{\, t \mid (\exists\, t_r \in r)\ & ((t[Attr(R) - \{R_{tn}\}] = t_r[Attr(R) - \{R_{tn}\}]) \\ & \wedge (t[R_{tn}] = \rhd\lhd_t^{\rhd\lhd} (t_r[R_{tn}]L_{tn}, q) \neq \emptyset))\}\end{aligned}$$

Note: This definition is the same as Definition 4.25 (Case 2) for non-temporal nested relations in the NRM.

Case 3: *Join two temporal nested relations having one or more atomic and temporal attributes in common which belong in different subrelations of the two relations (but in the same subrelation in each relation)*

The same two cases can be distinguished here, as in the NRM, depending on whether or not the nesting levels of the common atomic and temporal attributes in the different subrelations of the two relations are the same.

Case 3a: *The common atomic and temporal attributes are at the same nesting level in the two joined relations*

Definition 5.10: Let r and q be two temporal nested relations with relation schemes $R = \{R_1, R_2, \ldots, R_{tn}, \ldots, R_k\}$ and $Q = \{Q_1, Q_2, \ldots, Q_{tn}, \ldots, Q_{k'}\}$ respectively where $k, k' > 0$ and $k \neq k'$ (in general). Let also, Attr(R) and Attr(Q) be the sets of all attributes of relations r and q respectively. Suppose that at least the attributes R_{tn} and Q_{tn} of the two relations r and q respectively are temporal nested attributes and that they contain the common atomic and temporal attributes.

Let L be a join path of R and M a join path of Q. L is of the form $R_{tn}L_{tn}$ where L_{tn} is a join path of R_{tn} and M is of the form $Q_{tn}M_{tn}$ where M_{tn} is a join path of Q_{tn}. Then, $\rhd\lhd_t^{\rhd\lhd}$ (rL, qM) is defined as follows:

$$\begin{aligned}\rhd\lhd_t^{\rhd\lhd}(rL, qM) = \rhd\lhd_t^{\rhd\lhd}(qM, rL) = \rhd\lhd_t^{\rhd\lhd}(r(R_{tn}L_{tn}), q(Q_{tn}M_{tn})) = \\ \{t \mid (\exists\, t_r \in r)\,(\exists\, t_q \in q) \\ ((t[Attr(R) - \{R_{tn}\}] = t_r[Attr(R) - \{R_{tn}\}]) \\ \wedge (t[Attr(Q) - \{Q_{tn}\}] = t_q[Attr(Q) - \{Q_{tn}\}]) \\ \wedge (t[R_{tn}Q_{tn}] = \rhd\lhd_t^{\rhd\lhd}(t_r[R_{tn}]L_{tn},\ t_q[Q_{tn}]M_{tn}) \neq \emptyset))\}\end{aligned}$$

Example 5.8: Suppose that relations T_LOCATION (Fig. 3.15), T_CASH-POINT (Fig. 3.16) and the following query are given “Which banks have branches at the same road as the given companies during the same time period?”.

The result relation is shown in Fig. 5.10.

COMPANY	(BUILDING ADDRESS ADDRESS_PER$_t$ SORT_CODE)				BANK
	BUILDING	ADDRESS	ADDRESS_PER$_t$	SORT_CODE	
Microsoft	Pegasus House	Ashford St.	[16/11/1995, 4/4/1997)	386600	Barclays
Microsoft	Queen's Building	Park Rd.	[18/3/1995, 10/8/1998)	560045	NatWest
Microsoft	Queen's Building	Park Rd.	[16/6/1995, 1/1/2010)	478210	Lloyd's
	Pegasus House	Ashford St.	[23/7/1995, 4/4/1997)	478202	

Fig. 5.10: $\triangleright\triangleleft_t^{\triangleright\triangleleft}$ (T_LOCATION(ANNEX(ADDRESS, ADDRESS_PER$_t$)), T_CASH-POINT(BRANCH(ADDRESS, ADDRESS_PER$_t$)))

Case 3b: *The common atomic and temporal attributes are not at the same nesting level in the two joined relations*

The definition is the same as Definition 4.27 (Case 3b) for non-temporal nested relations in the NRM.

Case 4: *Join two temporal nested relations which have one or more temporal nested attributes at the top level in common*

Definition 5.11: Let r and q be two temporal nested relations, with relation schemes R and Q respectively. Let also, Attr(R) and Attr(Q) be the sets of all attributes of relations r and q respectively. The two relations r and q have in common the temporal nested attributes Attr(R_{tn}) = {R_{tn1}, R_{tn2}, …, R_{tnk}} and Attr(Q_{tn}) = {Q_{tn1}, Q_{tn2}, …, Q_{tnk}} respectively (k > 0) at the top level. Then, $\triangleright\triangleleft_t$ (r, q) is defined as follows:

$$\begin{aligned}\triangleright\triangleleft_t (r, q) = \{\, t \mid\ & (\exists\, t_r \in r)\ (\exists\, t_q \in q) \\ & ((t[\text{Attr}(R) - \text{Attr}(R_{tn})] = t_r[\text{Attr}(R) - \text{Attr}(R_{tn})]) \\ & \wedge (t[\text{Attr}(Q) - \text{Attr}(Q_{tn})] = t_q[\text{Attr}(Q) - \text{Attr}(Q_{tn})]) \\ & \wedge (t[\text{Attr}(R_{tn})] = \triangleright\triangleleft_t (t_r[\text{Attr}(R_{tn})],\ t_q[\text{Attr}(Q_{tn})]) \\ & \quad = ((R_{tn1} \cap_t^{\cap} Q_{tn1}) \wedge (R_{tn2} \cap_t^{\cap} Q_{tn2}) \wedge \ldots \wedge (R_{tnk} \cap_t^{\cap} Q_{tnk}))\ ?\ \emptyset))\}\end{aligned}$$

Case 5: *Join two temporal nested relations having one or more subrelations in common which in one relation are subrelations of a subrelation of a relation and in the other are at the top level*

The definition is the same as Definition 5.9 (Case 2).

Case 6: *Join two temporal nested relations having one or more common subrelations which belong at different subrelations of the two relations (but in the same subrelation in each relation)*

Case 6 in the TNM is similar to Case 6 in the NRM (subsection 4.3.10).

In the following example the natural join of two temporal nested relations, having one common temporal nested attribute belonging to different subrelations and at different nesting levels in the two relations, is computed.

Example 5.9: Let assume that the T_TRAINING relation (Fig. 3.13) and the T_DEPT relation (Fig. 3.14) are given. In order to answer the query "Find which trainers have given courses to which staff members?" the natural join of the two relations must be computed. The two relations have in common the temporal nested attribute COURSE which is located at nesting level 2 at the T_TRAINING relation and at nesting level 3 at the T_DEPT relation. The result relation is shown in Fig. 5.11.

D	**DN**	**(UN UD (SNAME STAFF_PER$_t$ COURSE TRN) COMPANY)**							
		UN	**UD**	**(SNAME STAFF_PER$_t$ COURSE TRN)**					**COMPANY**
				SNAME	**STAFF_PER$_t$**	**COURSE**		**TRN**	
						CN	**CN_PER$_t$**		
1	Research	511	Software Engineering	Paul	[13/5/1994,5/9/1996)	5.2	[1/2/1995, 25/4/1995)	Jack	Apple
				Peter	[26/2/1996,1/1/2010)	3.5	[1/1/1998, 28/10/1998)	Mark	
		511	Software Engineering	Paul	[13/5/1994,5/9/1996)	5.0	[17/12/1995,30/1/1996)	Tim	IBM
		552	Basic Research	Anna	[30/4/1994,27/8/1995) ∪ [4/6/1997,19/11/1998)	3.3	[29/9/1997, 10/2/1998)	Karen	Microsoft
				Mary	[15/5/1995, 1/1/2010)	3.3	[17/1/1997, 28/4/1997)	Karen	
2	Development	780	Maintenance	Helen	[14/2/1996, 1/1/2010)	3.5	[17/8/1997, 1/1/2010)	Mark	Apple
		650	Design	Steve	[2/1/1995, 27/6/1998)	5.0	[18/3/1996, 1/7/1996)	Tim	IBM

Fig. 5.11: $\rhd\lhd_t^{\rhd\lhd}$ (T_TRAINING(TRAINER(COURSE)), T_DEPT(STAFF(COURSE_DETAILS(COURSE))))

It can be concluded that the only difference between the natural join operation for temporal nested relations in the TNM, defined in this section, and the natural join operation for non-temporal nested relations in the NRM, defined in section 4.3.10, is when it reaches a nesting level where a temporal attribute occurs. In this case, either the Definition 5.8 of Case 1 or the Definition 5.11 of Case 4 of this section is applied. Therefore, the TJoin operation of the TNM has been defined as a consistent extension of the recursive nested natural join operation defined for the NRM.

5.3.12 The *Recursive Temporal Nested Q-Join* Operation ($\rhd\lhd_{t_T}^{\rhd\lhd}$)

Let r and q be two temporal nested (in general) relations with relation schemes R and Q respectively. Let X and Y be two atomic or temporal attributes belonging to relations R and Q respectively and Θ the condition that they must satisfy. Assume, without loss of generality, that Y belongs to a deeper nesting level than X and $L_{\sigma Y' \to Y}$ is the select path of Y starting at node Y′ which is at the same nesting level as X (when X and Y are at the same nesting level the select path is empty). So, the Θ-TJoin of two relations r and q is defined as follows:

Definition 5.12 (*Q-TJoin)*

$r \rhd\lhd_{t_T}^{\rhd\lhd} q = \sigma_t^{\sigma}((r \times_t q)_{X \Theta Y} L_{\sigma Y' \to Y})$

Note: The definition of the Θ-TJoin operation for temporal nested relations is the same as the definition of the recursive Θ-Join operation for nested relations (Definition 4.30); the only difference is that the comparison (selection operation) can be performed also between temporal attributes.

5.3.13 The *Recursive Temporal Nested Division* Operation (¸t˒)

The recursive temporal nested division operation is not defined for the same reasons discussed in section 4.3.12.

5.3.14 Temporal Functions

The START and STOP functions have been defined in section 3.2.2. Different researchers have defined various temporal functions and a detailed list can be found in [LM97]. Hence, repetition of such functions is omitted here. Informal description of the functionality of such functions is given in the sequel wherever they are used.

5.4 Closure Property of Operations

The temporal nested operations defined in section 5.3 are proved, in this section, to be closed in U, where U is the underlying domain of the temporal nested relations. For all the following propositions, let r be a temporal nested relation with relation scheme R(Attr(R_a), Attr(R_t), Attr(R_{tn})), where Attr(R_a) = {R_{a1}, R_{a2}, ..., R_{ak}} is the set of all atomic attributes ($k \geq 0$), Attr(R_t) = {R_{t1}, R_{t2}, ..., R_{tq}} is the set of all temporal attributes ($q \geq 0$) and Attr(R_{tn}) = {R_{tn1}, R_{tn2}, ..., R_{tnm}} is the set of all temporal nested attributes in R ($m \geq 0$). Nested attributes can be considered as a special case of temporal nested attributes and so they are not included in the relation scheme. Let also, D_{ai} be the underlying domain of the atomic attribute R_{ai} (where $0 \leq i \leq k$), D_{tp} the underlying domain of the temporal attribute R_{tp} (where $0 \leq p \leq q$) and P(DOM(R_{tnj})) the underlying domain of the temporal nested attribute R_{tnj} (where $0 \leq j \leq m$).

The underlying domain of relation r is:

$D_{a1} \times D_{a2} \times \ldots \times D_{ak} \times D_{t1} \times D_{t2} \times \ldots \times D_{tq} \times P(DOM(R_{tn1})) \times P(DOM(R_{tn2})) \times \ldots \times P(DOM(R_{tnm}))$.

Note: A temporal attribute behaves in the same way as an atomic attribute in the following proofs. However, the domains of temporal attributes are distinct from those of atomic attributes.

Proposition 5.1 The TUnion operation is closed in U.

Proof: Let q be a temporal nested relation with the same relation scheme and the same underlying domain as relation r.

Then, the underlying domain of relation s, where $s = r \cup_t^{\cup} q$, is also $D_{a1} \times D_{a2} \times \ldots \times D_{ak} \times D_{t1} \times D_{t2} \times \ldots \times D_{tq} \times P(DOM(R_{tn1})) \times P(DOM(R_{tn2})) \times \ldots \times P(DOM(R_{tnm}))$, according to Definition 5.1. So, the output of the TUnion operation is a temporal nested relation with the same scheme and the same underlying domain as the input relations r and q. Thus, the TUnion operation is closed in U.

Proposition 5.2 The TDifference operation is closed in U.

Proof: The proof is omitted since is the same as that of the TUnion operation.

Proposition 5.3 The TIntersection operation is closed in U.

Proof: The proof is omitted since is the same as that of the TUnion operation.

Proposition 5.4 The TProjection operation is closed in U.

Proof: The output of the TProjection operation, $\pi_t^{\pi}(rL_{\pi})$, of the temporal nested relation r, is a temporal nested relation whose underlying domain is a proper subset of the underlying domain of relation r, according to Definition 5.4. Thus, the TProjection operation is closed in U.

Proposition 5.5 The TimeSlice operation is closed in U.

Proof: The output of the TimeSlice operation, $s^{s}_{TE}(r)$, of the temporal nested relation r, is a temporal nested relation whose underlying domain is the underlying domain of relation r, according to Definition 5.5. Thus, the TimeSlice operation is closed in U.

Proposition 5.6 The TSelection operation is closed in U.

Proof: The output of the TSelection operation, $\sigma_t^{\sigma}(r_c L_{\sigma})$, of the temporal nested relation r, is a temporal nested relation whose underlying domain is the underlying domain of relation r, according to Definition 5.6. Thus, the TSelection operation is closed in U.

Proposition 5.7 The TCartesianProduct operation is closed in U.

Proof: Let q be a temporal nested relation with relation scheme $Q(Attr(Q_a), Attr(Q_t), Attr(Q_{tn}))$, where $Attr(Q_a) = \{Q_{a1}, Q_{a2}, \ldots, Q_{ak'}\}$ the set of all atomic attributes ($k' \geq 0$), $Attr(Q_t) = \{Q_{t1}, Q_{t2}, \ldots, Q_{tq'}\}$ the set of all temporal attibutes in Q ($q' \geq 0$) and $Attr(Q_{tn}) = \{Q_{tn1}, Q_{tn2}, \ldots, Q_{tnm'}\}$ the set of all temporal nested attibutes in Q ($m' \geq 0$). Let also, $D'_{ai'}$ be the underlying domain of the atomic attribute $Q_{ai'}$ (where $0 \leq i' \leq k'$), $D'_{tp'}$ the underlying domain of the temporal attribute $Q_{tp'}$ (where $0 \leq p' \leq q'$) and $P(DOM(Q_{tnj'}))$ the underlying domain of the temporal nested attribute $Q_{tnj'}$ (where $0 \leq j' \leq m'$).

The underlying domain of relation q is $D'_{a1} \times D'_{a2} \times \ldots \times D'_{ak'} \times D'_{t1} \times D'_{t2} \times \ldots \times D'_{tq'} \times P(DOM(Q_{tn1})) \times P(DOM(Q_{tn2})) \times \ldots \times P(DOM(Q_{tnm'}))$.

Then, the output of the TCartesianProduct operation of the two temporal nested relations r and q, $\times_t^{\times}$ (rL, q), is a temporal nested relation with underlying domain $D_{a1} \times D_{a2} \times \ldots \times D_{ak} \times D_{t1} \times D_{t2} \times \ldots \times D_{tq} \times P(DOM(R_{tn1})) \times P(DOM(R_{tn2})) \times \ldots \times P(DOM(R_{tnm})) \times D'_{a1} \times D'_{a2} \times \ldots \times D'_{ak'} \times D'_{t1} \times D'_{t2} \times \ldots \times D'_{tq'} \times P(DOM(Q_{tn1})) \times P(DOM(Q_{tn2})) \times \ldots \times P(DOM(Q_{tnm'}))$, according to Definition 5.7. So, the output of the TCartesianProduct operation is a temporal nested relation whose underlying domain is the cartesian product of the underlying domains of the two input relations, r and q. Thus, the TCartesianProduct operation is closed in U.

Proposition 5.8 The TJoin operation is closed in U.

Proof: The proof is omitted since is the same as that of the TCartesianProduct operation.

Proposition 5.9 The Θ-TJoin operation is closed in U.

Proof: The proof is omitted since is the same as that of the TCartesianProduct operation.

5.5 Summary

In this chapter, a temporal database model (TNM) and algebra have been defined using nested relations. The advantage of this approach is that it combines for the first time a simple temporal extension with nested relational theory, thus exploiting the suitability of the Nested Relational Model for representing temporal complex objects.

All the operators that are used are recursively defined. The result is that there is no need to flatten the temporal nested relations when queries are executed. Data duplication does not occur. Furthermore, the representation of the data is claimed to be in a “natural form” and thus, it is easier for users to understand when querying the data.

The most interesting and at the same time difficult operation to define is the natural join operation. It has been given special consideration in this chapter, within the scope of the TNM. As in the previous chapter, detailed attention has been paid to solving the problems that are presented when computing the natural join operation of two relations, due to the dissimilarity of the common attributes that the two relations might share.

Lastly, the closure property for all the operations of the TNM has been proved.

CHAPTER 6

MODEL IN USE

6.1 Introduction

This chapter gives a number of examples of the management of temporal nested data using the Temporal Nested Model (TNM) described in chapters 3 and 5. The queries illustrate the features of the temporal nested relational algebra that has been defined also in chapter 5.

The examples are presented incrementally and thus, have been divided in two different categories; the first one includes queries that involve non-temporal nested data, showing the expressive power of the NRM, defined in chapter 4 and the second one deals with temporal nested relations for the management of temporal nested data, demonstrating the full expressive power of the TNM.

Note that in the examples that follow, for simplicity reasons, the result relation is the one obtained after the application of the necessary number of unnest operations. Moreover, when a new attribute is computed from an aggregate or scalar function, a new name is given by the user to that new attribute, without the need of using the rename operation.

6.2 Management of Nested Data

The non-temporal nested model presented in chapter 4, is a well-defined and formalised nested model where data restructuring operations are avoided. In this chapter, examples are provided to show the ease of use of the NRM algebra. Relations have no restrictions on the number of nesting levels they can contain. The nested model presented, provides a better way of representing and querying complex data as demonstrated by the queries that follow since they are short and do not require nest, unnest or any other restructuring operations for the manipulation of nested data.

A number of examples are presented that contain only operations on nested data, demonstrating how this model works and functions. Queries refer to the nested database example described in section 3.5 (Fig. 3.6–3.12). For each query, the resulting relation is also given. In the resulting relation the names of the new subrelations are derived from the names of the attributes they contain using a bracketed notation (see also section 4.3.10). For some queries, comparisons are made with other proposed models, in order to demonstrate the claimed superiority of the NRM and the weakness of other proposed algebras to express these queries.

Query 1: What are the descriptions of the units that belong to department 1 and who are the trainers who have given courses to staff members of these units (ref. to Fig. 3.7)? Display also the value for the department.

$$\pi^{\pi}((\sigma^{\sigma}(DEPT_{D\,=\,1}))\ D,\ UD,\ TRN)$$

D	(UD (TRN))	
	UD	(TRN)
		TRN
1	Software Engineering	Mark
	Basic Research	Karen
		Tim
	Planning	Mark

Fig. 6.1: The resulting relation of Query 1

A projection operation on a selected part of the DEPT relation is needed to answer the above query. Three attributes of the relation are projected which can be found at different nesting levels; attribute D at nesting level 1 (top level), attribute UD at nesting level 2 and attribute TRN at nesting level 3. However, the projection operation takes place as normal, without changing the structure of the relation using unnest and nest operations and thus, the nesting arrangement of the relation is maintained in the resulting relation as well. Therefore, in the resulting relation, D, UD and TRN are still at nesting levels 1, 2 and 3 respectively, as in the input relation DEPT.

Query 2: Find the department names and the companies that have provided these departments with trainers (ref. to Fig. 2.2, 2.3).

$$\pi^{\pi}((\text{DEPT\_1} \rhd\!\lhd^{\rhd\lhd} \text{TRAINING\_2})\ \text{DN, COMPANY})$$

DN	(COMPANY)
	COMPANY
Research	IBM
Development	IBM

Fig. 6.2: The resulting relation of Query 2

Although the two relations that have to be joined, DEPT_1 and TRAINING_2, contain the common nested attribute TRAINER at different nesting levels, and also a projection operation is applied to two attributes at different nesting levels in the resulting relation after performing the natural join operation, the structure of these complex objects is preserved while accessing them and the query can be answered easily with two basic operations, natural join and projection, ignoring the complexity of the operands. The output relation is nested completely without requiring any nesting operations.

This query cannot be performed in Abiteboul and Bidoit's model ([AB86]) since the natural join operation cannot be performed between the TRAINING_2 and DEPT_1 relations, as they do not contain any common atomic attributes at the top level (see section 2.2.1).

Query 3: Find the tuples with course numbers equal to the number of the department for the whole tuple (ref. to Fig. 3.7).

$$\sigma^{\sigma}(\mathrm{DEPT}_{D\,=\,CN})$$

D	DN	UNIT					
		UN	UD	COURSE_DETAILS			
				TRN	COMPANY	C	
						CN	Y
1	Research	511	Software Engineering	Mark	Apple	1	75
		552	Basic Research	Karen	Microsoft	1	82
2	Development	780	Maintenance	Mark	Apple	2	76
		981	Planning	Jack	Apple	2	81

Fig. 6.3: The resulting relation of Query 3

The above query shows the advantage of the selection operation proposed in section 4.3.5 that allows arbitrary expressions to be specified in the select condition, as for example equality of values of attributes that are not at the same nesting level in the relation, without unnesting and nesting the relation. The query is expressed algebraically in exactly the same way as if the two compared attributes were at the top level of the original relation.

Query 4: Find the names of the banks and the companies that are situated at the same road (ref. to Fig. 3.8, 3.9).

$$\nu^{\nu}((\mu^{\mu}(\pi^{\pi}((\mathrm{LOCATION} \rhd\!\lhd^{\rhd\lhd} \mathrm{CASH\text{-}POINT})_{\mathrm{COMPANY,BANK,ADDRESS}}))_{(\mathrm{ADDRESS})})_{(\mathrm{COMPANY,BANK})\rightarrow(\mathrm{COMPANY\ BANK})})$$

ADDRESS	(COMPANY BANK)	
	COMPANY	BANK
Porchester Rd.	TOSHIBA	NatWest
Ashford St.	Microsoft	Barclays
		Lloyd's
Park Rd.	Microsoft	Lloyd's
		NatWest

Fig. 6.4: The resulting relation of Query 4

In this example, and in similar cases, nest and unnest operations are necessary since they can restructure the relations and as a result, present the same data in a different format that is required by the given query.

However, extra nest and unnest operations are avoided in the above query since the natural join and projection operations are defined recursively in the NRM algebra.

In Abiteboul and Bidoit's model this query cannot be performed since the two relations that participate in the natural join operation do not have any common attributes at the top level.

Query 5: Find the names of the trainers that have given the "Computer Skills" training course (ref. to Fig. 3.6, 3.12).

$$\pi^{\pi}((\sigma^{\sigma}(\text{TRAINING} \bowtie^{\bowtie} \text{COURSE})_{\text{TITLE= "Computer Skills"}})\ \text{TRN})$$

TRN
Jack
Karen

Fig. 6.5: The resulting relation of Query 5

One can easily see the advantage of joining subrelations which are at different nesting levels (in this example, the subrelation C at nesting level 3 in relation TRAINING and at nesting level 1 in relation COURSE), without the need to unnest and nest the data and without any other restructuring operations assumed by other proposed models (e.g. [AB86], [Col90]).

In contrast, in Levene's model the natural join can be applied only if relation COURSE is extended with two empty nodes at levels 1 and 2 so that the common attribute C to appear at the same nesting level 3 in both relations. Then, the two relations are *joinable*, according to Levene's definition and therefore, can be joined (see also section 2.2.5).

The above example shows that the nested algebra proposed in this thesis provides a simple way of answering queries, since even just the algebraic solution of the query can be translated naturally to the above well-phrased query; moreover, the query does not distinguish between nested and flat relations, as the query would be expressed in the same way if the two relations, TRAINING and COURSE, were flat relations. This is explained by the recursive nature of the NRM operations.

Query 6: Find the names of the banks which are located on the same road as the companies for which Tim or Karen have worked for, together with the names of these companies (ref. to Fig. 3.6, 3.8, 3.9).

$$\pi^{\pi}((((\sigma^{\sigma}(\text{TRAINING}_{(\text{TRN}=\text{“Tim” OR TRN = “Karen”})})) \rhd\lhd^{\rhd\lhd} \text{LOCATION}) \rhd\lhd^{\rhd\lhd} \text{CASH-POINT}) \\ \text{COMPANY, BANK})$$

COMPANY	BANK
Microsoft	Barclay's
Microsoft	NatWest
Microsoft	Lloyd's

Fig. 6.6: The resulting relation of Query 6

This query requires two natural join operations. However, since the natural join defined in this thesis can be performed between any possible relations sharing common attributes, it does not involve any preliminary checks to determine if the two operand relations are qualified for the natural join. In other models, for example in Colby ([Col90]) or in Abiteboul and Bidoit's models ([AB86]), it is not certain if the natural join operation can be performed between a nested relation and the output of the natural join of two nested relations, since, as explained in chapter 2 of this thesis, for each of these models the natural join operation is subject to some restrictions. This, however, is not discussed either in [Col90] or in [AB86].

On the other hand, in NRM any possible combination of relations, sharing at least one common attribute, can be joined.

This query also demonstrates how complex queries can be answered easily in the query language proposed in this thesis.

Query 7: What is the title of the course that has the maximum number of different topics (ref. to Fig. 3.12)? Display also the the number of different topics that this course has.

π^{π}(COURSE(TITLE, N-COUNT[TOPICS / TITLE]$\leftarrow$MTOPICS)) $\bowtie^{\bowtie}$

π^{π}(π^{π}(COURSE(TITLE, N-COUNT[TOPICS / TITLE]$\leftarrow$MTOPICS1))

(MAX(MTOPICS1)) $\leftarrow$MTOPICS)

TITLE	MTOPICS
Computer Skills	3

Fig. 6.7: The resulting relation of Query 7

Aggregate functions for nested attributes have been redefined in chapter 4 (see section 4.3.13).

The above query is expressed in the TNM using the following steps:

STEP 1: In the original relation COURSE, the number of different topics per title is computed, it is named MTOPICS1 and projected on TITLE and MTOPICS1 attributes.

STEP 2: From the result of step 1, MAX(MTOPICS1) is computed, named MTOPICS and projected.

STEP 3: In the original relation COURSE, the number of different topics per title is computed, it is named MTOPICS and projected on TITLE and MTOPICS.

STEP 4: The results of steps 2 and step 3 are joined together.

Note that, for simplicity reasons, when a new attribute is computed from an aggregate or scalar function, a new name is given by the user to that new attribute, without the need of using the rename operation.

It is noteworthy that if the relation COURSE was a flat relation then, the SUMMARIZE operation would be used to produce the same result in combination with the traditional aggregate functions COUNT and MAX.

It must be said that this query or any other query containing aggregate functions on nested attributes cannot be expressed in any other relational model discussed in chapter 2 apart from [DL91] with the use of an additional operator, the subrelation constructor.

The query is represented in Deshpande and Larson's model as follows:

π[TITLE, MAX[SUBJECT′]] (?(C, COURSE_DURATION, TITLE, SUBJECT, SUBJECT′); SUBJECT′ := COUNT[TOPICS](SUBJECT)? (COURSE)) where ? is the subrelation constructor.

Query 8: Find all trainers who have given more courses than Karen has (ref. to Fig. 3.6).

$\pi^{\pi}((\sigma^{\sigma}$

$(\pi^{\pi}((\nu^{\nu}(\mu^{\mu}(\ \pi_t^{\pi}(\text{T\_TRAINING}(\text{TRN, COURSE}))_{\text{COURSE}})_{(\text{CODE, C})\rightarrow\text{COURSE}}))$

$(\text{TRN, N-COUNT[CN/TRN]} \leftarrow \text{MCN}))$

$\times^{\times}$

$\pi^{\pi}((\sigma^{\sigma}(\text{TRAINING}_{\text{TRN= "Karen"}}))(\text{N-COUNT[CN/TRN]}\leftarrow\text{MCN1})))_{\text{MCN > MCN1}})\ \text{TRN})$

TRN
Mark
Tim

Fig. 6.8: The resulting relation of Query 8

Two copies of the TRAINING relation are needed for this query in order to perform the cartesian product operation between them. However, to make the query simpler, a projection operation is applied to the first copy of the relation and an aggregate function is also used to count the number of nested tuples which corresponds to the number of different courses that each trainer (TRN) has given. Moreover, an unnest and then a nest operation are also used to a projected part of the original relation to convert the relation to the right one, before the computation of the aggregate function. With the second copy of the relation, a projection is performed on a selection of the relation. The same aggregate function is also used here, applied to the same attribute as before. The cartesian product is performed afterwards between a binary relation and a unary one containing only one tuple.

Once again, the above query can demonstrate the expressive power of the proposed nested model and the facility in stating complex queries. This query, as the previous one, cannot be expressed in any other nested relational model presented in chapter 2 apart from [DL91], yet with the problem discussed above.

6.3 Management of Temporal Nested Data

Temporal data in TNM are represented as temporal elements. As defined in section 3.2.2 a temporal element is a finite set of disjoint and non-adjacent time intervals (see Definition 3.7). Consequently, the management of temporal data consists of handling time intervals.

The behaviour of time intervals has been investigated by a number of researchers, as mentioned in section 3.3.4. To the best of the knowledge of the author of this thesis the most extended, analytical and complete approach to intervals has been given by Lorentzos in [Lor88]. He described the representation of generic intervals in a database model and defined an algebra to manipulate them. Consequently, the management of temporal data in the TNM adopts the approach of [Lor88] in general. The operators that are used to compare the relative positions of two temporal elements are based on the set of operators for intervals defined in [Lor88]. Although these set operators cover all possible relative positions of two intervals, they are multitudinous and difficult to memorise. For this reason, only a limited number of them are used in TNM, the most important, useful and basic and all the rest can be derived from them. Therefore, the following set of operators for time intervals are considered to be known and can be used in the TNM (Fig. 6.9). Their definitions can be found in [DDL03.

BEFORE

AFTER

=

MEETS

OVERLAPS

COVERS

Fig. 6.9: Operators for two time intervals

In the TNM, as described in chapters 3 and 5, in the general case, temporal elements can be found in temporal attributes which form part of temporal nested attributes. So, a temporal nested attribute consists of a number of atomic and/or non-temporal nested attributes and one or more temporal attributes which contain the time periods over which the corresponding data instances of the atomic and/or non-temporal nested attributes are valid. In this respect and at that level, a temporal nested attribute can be considered as a subrelation that represents a tuple timestamping relation. Consequently, when the different nesting levels of a temporal nested relation are traversed from top to bottom and the temporal nested attributes are reached, the operations are performed as if the temporal nested attributes were tuple timestamping relations, despite the fact that the original relations are attribute timestamping temporal nested relations. This is one of the most significant benefits the TNM model offers, since it combines the benefits of tuple and attribute timestamping database models presented in section 3.3.2.

The management of temporal nested data is one of the most complicated issues in temporal databases. Combining temporal data with nested data, gives a result that is even more complex. Therefore, an algebra which can provide a simple method for the manipulation of temporal nested data is an important advance.

In what follows, queries managing temporal nested data are presented. In this section some queries are also expressed in Tansel's model ([Tan97]). Tansel's model, as has been discussed in chapter 2, is the only temporal model which provides full support of nested relations and in addition, as it will be shown in chapter 8, Tansel's model and TNM are the only models that satisfy all criteria concerning the representation features. For this reason, a comparison is made only between TNM and Tansel's model for some queries in this section, when it is considered worthwhile.

The following examples illustrate the expressive power of the proposed TNM model and show that it is both easy to use and effective in formulating queries.

Query 9: Find the names and course histories of staff members who were working in the Research Department at 1/1/1998 (ref. to Fig. 3.14).

$$\pi_t^{\pi}((\sigma_t^{\sigma}(\text{T\_DEPT})_{(\text{DN=“Research” AND STAFF\_PER}_t \text{ OVERLAPS } [1/1/1998,\ 2/1/1998))})\ \text{SNAME, COURSE})$$

SNAME	COURSE	
	CN	CN_PER$_t$
Peter	3.5	[1/1/1998, 28/10/1998)
Anna	3.1	[1/7/1995, 1/8/1995)
	3.3	[29/9/1997, 10/2/1998)
Mary	3.3	[17/1/1997, 28/4/1997)

Fig. 6.10: The resulting relation of Query 9

This query is a classical temporal query. It involves a selection of historical tuples from T_DEPT relation together with a temporal projection. The structure of the nested data is preserved while accessing them as well as while performing the operations on them. In this example, the temporal elements of attribute STAFF_PER$_t$ are compared with a time point. Time points can be considered as elementary time intervals ([Lor88]). Therefore, set operators between time intervals can be extended to include operators between time intervals and time points as well.

In Tansel's model the T_DEPT relation is represented in the following way:

D	DN	STAFF			
		UN	UD	COURSE_DETAILS	
				SNAME	COURSE
					CN
1	Research	511	Software Engineering	<{[13/5/1994, 5/9/1996)}, Paul>	<{[1/2/1995, 24/6/1995)}, 5.2>
					<{[27/8/1995, 30/1/1996)}, 5.0>
				<{[26/2/1996, 1/1/2010)}, Peter>	<{[1/1/1998, 28/10/1998)}, 3.5>
.	.	.	.	.	.
.	.	.	.	.	.

Fig. 6.11: T_DEPT relation in Tansel's model

The Temporal Relational Algebra (TRA) expression for the above query in Tansel's model is shown below:

$\pi_{SNAME,\, COURSE}\ (\ T_{SNAME}\ (\sigma_{DN= \text{"Research"}\ \wedge\ 1/1/1998\ \in\ SNAME.T}\ (\mu_{COURSE\_DETAILS}\ (\mu_{STAFF}(T\_DEPT)))))$

As can be seen, this query requires additional operations in Tansel's model. In particular, two extra unnest operations and one drop-time operation are needed to express this query, due to the non-recursive nature of the algebra and the temporal atoms' representation of time-varying attributes in Tansel's model.

Query 10: During which time periods did staff members follow courses (ref. to Fig. 3.14)?

$$\pi_t^{\pi}((\text{T\_DEPT})\ \text{SNAME},\ \text{CN\_PER}_t)$$

SNAME	(CN_PER_t)
	CN_PER_t
Paul	[1/2/1995, 24/6/1995) ∪ [27/8/1995, 30/1/1996)
Peter	[1/1/1998, 28/10/1998)
Anna	[1/7/1995, 1/8/1995) ∪ [29/9/1997, 10/2/1998)
Mary	[17/1/1997, 28/4/1997)
Katy	[13/2/1994, 4/3/1995) ∪ [22/4/1995, 15/5/1995)
Steve	[18/3/1996, 1/7/1996)
Helen	[17/8/1997, 1/1/2010)
Pat	[18/9/1995, 10/10/1995)

Fig. 6.12: The resulting relation of Query 10

The time periods during which staff members have followed courses are summarised. The temporal data, which pertains to different course numbers, is merged into one temporal element. This is achieved by the facility of the TNM projection operation to coalesce tuples having the same values for all the atomic attributes, by taking the unions of the temporal elements of their corresponding temporal attributes (see section 5.3.4).

In Tansel's model, since the projection operation is defined in exactly the same way as in the relational algebra, unnest operations need to take place initially to unnest the STAFF, COURSE_DETAILS and COURSE attributes before the projection operation is performed. A temporal atom decomposition operation needs also to be performed to split the CN attribute of T_DEPT relation into its temporal sets and value parts and place them as the last two columns of the result. Their names will be CN.T and CN.v respectively (see Fig. 6.11).

The query is expressed in Tansel's algebra as follows:

$$\pi_{\text{SNAME, CN.T}}\ (\ \partial_{\text{CN}}\ (\mu_{\text{COURSE}}\ (\mu_{\text{COURSE\_DETAILS}}\ (\mu_{\text{STAFF}}(\text{T\_DEPT})))))$$

Therefore, in Tansel's model, the result relation contains two attributes, one of them consisting of temporal sets (CN.T).

This query shows once more the greater simplicity of the TNM algebra compared to Tansel's one.

Query 11: List the starting time point of every course that each trainer has given (ref. to Fig. 3.13).

$$\pi_t^{\pi}((\text{T\_TRAINING})\ \text{TRN},\ \text{START}(\text{CN\_PER}_{\mathbf{t}})\leftarrow\text{BEGIN})$$

TRN	(BEGIN)
	BEGIN
Jack	2/11/1994
Mark	2/1/1992
	30/4/1995
Tim	19/3/1997
	17/12/1995
Karen	25/6/1996

Fig. 6.13: The resulting relation of Query 11

The above query is a pure temporal query. The temporal operator that is used is the START operator, which extracts the start point of a temporal element (see Definition 3.8). Note that, for simplicity reasons, a new name is given by the user to the new attribute computed from the START operator, without the need of using the rename operation.

Once again, the structure of the nested data is preserved in the resulting relation. Although the query is dealing with attributes belonging to nested attributes, TRN and $\text{CN\_PER}_{\mathbf{t}}$, it is expressed in the same way as if the input T_TRAINING relation, was a flat relation.

Query 12: How many courses took place in 1998 (ref. to Fig. 3.13)?

$$\text{COUNT}(\pi_t^{\pi}((\sigma_t^{\sigma}(\text{T\_TRAINING}_{(\text{CN\_PER}_t\ \text{OVERLAPS}\ [1/1/1998,\ 1/1/1999)))})\text{CN}))\leftarrow\text{TOTAL\_CN}$$

TOTAL_CN
4

Fig. 6.14: The resulting relation of Query 12

The aggregate function COUNT is used to count the number of courses that took place during a specific time period. The aggregate function can be applied in the normal way to attribute CN, despite being an atomic attribute belonging to a nested attribute, since it is the only one projected in the resulting relation. In addition, the temporal operator OVERLAPS is used in order to compute whether a temporal element has common points with a given time interval.

In Tansel's model two unnest operations are also needed before the selection operation to unnest completely the T_TRAINING relation, since the selection operation can be performed only for attributes at the top nesting level of the relation.

Query 13: Find the name of the bank that had a branch at the same address for the longest period of time (ref. to Fig. 3.16).

$$\pi_t^{\pi}((\sigma_t^{\sigma}(\text{T\_CASH-POINT}_{(\text{MAX(DURATION(ADDRESS\_PER}_t)))}))\text{BANK})$$

BANK
Lloyd's

Fig. 6.15: The resulting relation of Query 13

DURATION is a scalar function that returns the duration of a temporal element.

This query uses a projection of an atomic attribute of a selected tuple. The tuple which is selected must satisfy a given condition. The condition tests every tuple of the relation and extracts the valid tuple. The above query is expressed in the same way as if the relation was a flat relation in the CRM.

However, there are cases where a give n aggregate function must be applied to each nested set of tuples of a given nested attribute separately. This case is examined below, in queries 20 and 21.

Query 14: Find the titles of the courses that each trainer has given (ref. to Fig. 3.13 and 3.17).

$$\pi_t^{\pi}\ ((\text{T\_TRAINING} \rhd\lhd_t^{\rhd\lhd} \text{T\_COURSE})\ \text{TRN, TITLE})$$

TRN	TITLE
Jack	Programming
Mark	Computer Skills
Tim	Presentation Skills
Karen	Multimedia

Fig. 6.16: The resulting relation of Query 14

The natural join in the above query is performed according to Case 5 (see subsection 5.3.11). A projection operation is then performed to extract two attributes of the resulting relation.

In Tansel's model additional unnest operations are needed before the natural join operation.

COURSE	COURSE_DURATION	TITLE	SUBJECT
CN			TOPICS
.	.	.	.
.	.	.	.

T_COURSE

COMPANY	TRAINER	
	TRN	COURSE
		CN
.	.	.
.	.	.

T_TRAINING

Fig. 6.17: T_COURSE and T_TRAINING relations in Tansel's model

The query is expressed in Tansel's algebra as follows:

$\pi_{TRN,\ TITLE}((\mu_{COURSE}(\mu_{TRAINER}(T\_TRAINING))) \bowtie (\mu_{COURSE}(T\_COURSE)))$

Query 15: What was Karen's company when she gave a course to Mary, which course was it and when was it (ref. to Fig. 3.13 and 3.14)?

$\pi_t^{\pi}((\sigma_t^{\sigma}(T\_DEPT \bowtie_t^{\bowtie} T\_TRAINING)_{(TRN=\text{“Karen” AND SNAME=“Mary”})})COMPANY,\ COURSE)$

Note: The resulting relation of the natural join operation of T_DEPT and T_TRAINING relations can be seen in Fig. 5.11.

COMPANY	(COURSE)	
	COURSE	
	CN	CN_PER_t
Microsoft	3.3	[17/1/1997, 28/4/1997)

Fig. 6.18: The resulting relation of Query 15

The above query is a temporal query since it involves “when” expressions, although there is no need to include any external temporal comparisons. This is one of the advantages the natural join operation of the TNM offers. The temporal comparison is contained internally in the natural join operation when the two relations to be joined have common temporal nested attributes, as in this case (COURSE attribute). The temporal comparison corresponds to the computation of the overlapping periods of the common temporal attributes for equal values of the common atomic attributes.

Therefore, the query remains compact and simple.

Query 16: Find the courses that have been given by trainers who work for the IBM company and had completed before IBM moved from Maple House (ref. to Fig. 3.13 and 3.15). Display also the trainers’ names.

$$\pi_t^{\pi}((\sigma_t^{\sigma}(\text{T\_TRAINING} \rhd\lhd_t^{\rhd\lhd} \text{T\_LOCATION})_{(\text{COMPANY="IBM" AND CN\_PER}_t \text{ BEFORE STOP(ADDRESS\_PER}_t\text{) AND BUILDING="Maple House"})})_{\text{TRN, CN}})$$

TRN	(CN)
	CN
Tim	$\boxed{5.2}$

Fig. 6.19: The resulting relation of Query 16

Both BEFORE and STOP temporal operators are used in this query. The natural join operation is performed between the relations T_TRAINING and T_LOCATION, which share the COMPANY atomic attribute at the top level. The natural join is performed according to Case 1 (see subsection 5.3.11). The two temporal attributes, CN_PER_t and ADDRESS_PER_t, belonging to T_TRAINING and T_LOCATION relations respectively, do not contribute to the natural join since they represent different information. A number of selection operations take place afterwards and then a projection to produce the final result.

T_LOCATION relation in Tansel’s model is represented as shown in Fig. 6.20.

COMPANY	ANNEX	
	BUILDING	ADDRESS
Toshiba	<{[3/8/1995, 1/1/2010)}, North Building>	<{[3/8/1995, 1/1/2010)}, Porchester Rd.>
IBM	<{[17/1/1996, 22/5/1998)}, Maple House>	<{[17/1/1996, 22/5/1998)}, Kendal Av.>
	<{[10/6/1998, 1/1/2010)}, Main Building>	<{[10/6/1998, 1/1/2010)}, Danebury Rd.>
Microsoft	<{[29/10/1994, 4/4/1997)}, Pegasus House>	<{[29/10/1994, 4/4/1997)}, Ashford St.>
	<{[18/3/1995, 1/1/2010)}, Queen's Building>	<{[18/3/1995, 1/1/2010)}, Park Rd.>

Fig. 6.20: T_LOCATION relation in Tansel's model

In Tansel's model the above query is expressed as follows:

$$\pi_{\text{TRN, CN.v}} (\sigma_{\text{COMPANY="IBM"} \wedge \text{STOP(ADDRESS.T)>STOP(CN.T)} \wedge \text{BUILDING.v="Maple House"}} (\mu_{\text{COURSE}}(\mu_{\text{ANNEX, TRAINER}}$$
$$(\text{T\_TRAINING} \rhd\lhd \text{T\_LOCATION}))))$$

The result is shown in the following table (Fig. 6.21):

TRN	CN
Tim	5.2

Fig. 6.21: The resulting relation of Query 16 in Tansel's model

Two comments must be made when the TNM query is compared with the equivalent one in Tansel's model. Firstly, more operations are needed in Tansel's model, in particular three unnest operations and secondly, in Tansel's model the result is represented in fully unnested format and thus, the structures of the input relations are not maintained in the result.

Query 17: Find the course number of the course that lasted the shortest period of time and the names of the staff members that followed it (ref. to Fig. 3.13 and 3.14).

$$\nu^{\nu}((\mu^{\mu}((\pi_t^{\pi}((\sigma_t^{\sigma}(\text{T\_TRAINING}_{(\text{MIN(DURATION(CN\_PER}_t)))}))\text{CN})) \rhd\lhd_t^{\rhd\lhd}$$
$$(\pi_t^{\pi}(\text{T\_DEPT(SNAME,CN)})))_{(\text{CN})})_{\text{SNAME}\rightarrow(\text{SNAME})})$$

CN	(SNAME)
	SNAME
5.2	Paul

Fig. 6.22: The resulting relation of Query 17

The aggregate function MIN, in conjunction with the scalar function DURATION, is used in this query to compute the shortest temporal element of the CN_PER_t attribute of the T_TRAINING relation. In addition, the query makes use of the nest and unnest operations in order to restructure the result relation. These two restructuring operations cannot be avoided with this query, since otherwise, the result relation would have a different, undesirable structure. However, additional nest and unnest operations are not needed for the selection, projection and natural join operations, although the data are nested.

In contrast, in Tansel's model additional unnest operations are needed before the initial projection operations, since the projection operation cannot be performed on attributes at lower nesting levels. In Tansel's model the projection operation is defined in exactly the same way as in the conventional relational algebra.

Query 18: Find the numbers of courses that Paul has followed after he finished course number 5.2 and also the start time points of these courses (ref. to Fig. 3.14).

$$\pi_t^{\pi}((\sigma_t^{\sigma}(T\_DEPT1 \times_t T\_DEPT2)_{(SNAME1=\text{“Paul”}\ AND\ SNAME2=\text{“Paul”}\ AND\ CN2=5.2\ AND\ ((CN\_PER_t1\ AFTER\ CN\_PER_t2)\ OR\ (CN\_PER_t2\ MEETS\ CN\_PER_t1)))})\ CN1,\ START(CN\_PER_t1) \leftarrow BEGIN)$$

CN1	BEGIN
5.0	27/8/1995

Fig. 6.23: The resulting relation of Query 18

This query requires the cartesian product operation of relation T_DEPT with itself, in order to compare tuples of this relation. To be precise, initially, rename operations for all the attributes of the T_DEPT relation are needed in order to perform the cartesian product operation between two relations with no common attributes. However, the rename operations are not included in the above query for simplicity reasons. The selection operation consists of five select conditions. The temporal set operators, AFTER and MEETS, are used to denote two selection conditions that can both happen. In the final projection operation, the START operator is used to extract the start time points of the selected temporal elements. Note that, for simplicity reasons, a new name is given by the user to the new attribute computed from the START operator, without the need of using the rename operation.Even though the query is complicated, the algebraic representation is simpler than the non-recursive equivalent one, where more operators are required. In Tansel's model for example, unnest operations are required before the final projection operations.

Query 19: Who are the current trainers of all courses that Paul has ever followed (ref. to Fig. 3.13, 3.14)? Display also the course numbers.

If TODAY( ) is a function that returns the current date, the query can be formulated as:

$$\pi_t^{\pi}((\sigma_t^{\sigma}(\text{T\_TRAINING} \triangleright\triangleleft_t^{\triangleright\triangleleft} (\pi_t^{\pi}(\sigma_t^{\sigma}(\text{T\_DEPT}_{(\text{SNAME=“Paul”})}))\text{CN}))_{\text{STOP(CN\_PER}_t\text{)=TODAY( ) OR STOP(CN\_PER}_t\text{) AFTER TODAY( )}})\ \text{TRN, CN})$$

TRN	**(CN)**
	CN
Jack	$\boxed{5.2}$
Tim	$\boxed{5.0}$

Fig. 6.24: The resulting relation of Query 19

To answer the above query a natural join operation needs to be performed between two relations. However, the natural join is performed between a relation and a projection of a selection of another relation. This is due to the fact that otherwise, the natural join operation between the two original relations would be executed between the common temporal nested attribute COURSE, which would cause a wrong result in terms of this specific query. Therefore, the natural join operation that takes place here is a natural join operation between the temporal nested relation T_TRAINING and another flat relation with only one attribute, CN and consequently, sharing a common atomic attribute, CN.

In the result relation, the structures of the two original relations are preserved. Additionally, the STOP function is used in order to select the stop time point of the CN_PER_t temporal attribute if it is equal to or after the current date, since the current trainers are being retrieved. It is noticeable that the query remains simple and easy to perform despite the complicated wording of its specification.

Query 20: For each trainer, find the longest period of time for which they have given a course (ref. to Fig. 3.13).

$\pi_t^{\pi}((\nu^{\nu}(\mu^{\mu}(\ \pi_t^{\pi}(\text{T\_TRAINING}(\text{TRN, COURSE}))_{\text{COURSE}})_{(\text{CN, CN\_PER}t)\rightarrow\text{COURSE}}))$

$(\text{TRN, N-MAX}[\text{DURATION}(\text{CN\_PER}_t)/\text{TRN}]\leftarrow\text{K}))$

TRN	K
Jack	5067
Mark	5362
Tim	5125
Karen	4935

Fig. 6.25: The resulting relation of Query 20

Initially, an unnest operation is needed to a projected part of T_TRAINING relation and then, a nest operation, so that TRN attribute to be completely nested. Moreover, another projection is required to project attributes TRN and K (a new attribute computed by an aggregate function that calculates the longest period of time each trainer has given a course).

It is important to note here that aggregate functions have not been discussed in Tansel's model.

Query 21: For each trainer, find the title of the course he/she has given and which has lasted the longest period of time (ref. to Fig. 3.13, 3.17).

$\pi_t^{\pi}($

$(((\pi_t^{\pi}((\nu^{\nu}(\mu^{\mu}(\ \pi_t^{\pi}(\text{T\_TRAINING}(\text{TRN, COURSE}))_{\text{COURSE}})_{(\text{CN, CN\_PER}t)\rightarrow\text{COURSE}}))$

$(\text{TRN, N-MAX}[\text{DURATION}(\text{CN\_PER}_t)/\text{TRN}]\leftarrow\text{K}))) \triangleright\triangleleft_t^{\triangleright\triangleleft}$

$(\pi_t^{\pi}((\nu^{\nu}(\mu^{\mu}(\ \pi_t^{\pi}(\text{T\_TRAINING}(\text{TRN, COURSE}))_{\text{COURSE}})_{(\text{CN, CN\_PER}t)\rightarrow\text{COURSE}}))$

$(\text{TRN, CN, DURATION}(\text{CN\_PER}_t)\leftarrow\text{K})))) \triangleright\triangleleft_t^{\triangleright\triangleleft}$

$\pi_t^{\pi}(\text{T\_COURSE}(\text{CN, TITLE}))) \text{ TRN, TITLE})$

TRN	TITLE
Jack	Programming
Mark	Computer Skills
Tim	Presentation Skills
Karen	Multimedia

Fig. 6.26: The resulting relation of Query 21

This query includes two natural join operations. The first join is performed between the result of query 20 and a projected part of a subquery of query 20, which consists of three attributes, TRN, CN and K (a new attribute computed by the scalar function DURATION that calculates the duration of each course for each trainer).
Since, one of the projected attributes, TITLE, is derived from the T_COURSE relation, a natural join operation needs to be performed additionally, to join the previous result to a projected part of the T_COURSE relation.The above query is quite long and complicated, since an extra join operation is needed, for the computation of the N-MAX aggregate function and the projection of the CN attribute. However, this join operation cannot be avoided even in Codd's relational flat model.

Query 22: Which courses did Helen and Peter both follow simultaneously and when (ref. to Fig. 3.14)?
1st version:

$\pi_t^{\pi}((\sigma_t^{\sigma}((\pi_t^{\pi}(\rho^{\rho}[\text{SNAME}\leftarrow\text{SNAME1},\text{CN}\leftarrow\text{CN1},\text{CN\_PER}_t\leftarrow\text{CN\_PER}_t1](\text{T\_DEPT}))$

$(\text{SNAME1},\text{CN1},\text{CN\_PER}_t1)) \quad \times_t$

$(\pi_t^{\pi}(\rho^{\rho}[\text{SNAME}\leftarrow\text{SNAME2},\text{CN}\leftarrow\text{CN2},\text{CN\_PER}_t\leftarrow\text{CN\_PER}_t2](\text{T\_DEPT}))$

$(\text{SNAME2},\text{CN2},\text{CN\_PER}_t2)))$

$_{(\text{SNAME1=“Peter” AND SNAME2=“Helen” AND CN1=CN2 AND CN\_PER}_t\text{1 OVERLAPS CN\_PER}_t2))}$

$\text{CN1}, (\text{CN\_PER}_t1 \cap_{TE} \text{CN\_PER}_t2)\leftarrow\text{CPERIOD})$

CN1	**CPERIOD**
3.5	[1/1/1998, 28/10/1998)

Fig. 6.27: The resulting relation of the 1st version of Query 22

The above query requires a comparison between tuples of the same relation. This can be carried out by computing the cartesian product of a projected number of attributes of a temporal nested relation with itself. Rename operations must take place initially to change the names of the projected attributes so that the cartesian product operation can be performed afterwards between two relations with no common attributes. The temporal selection operation includes the set operator OVERLAPS between two temporal elements. Finally, two attributes need to be projected. One of them has to be computed by taking the intersection of two temporal elements. Note that, for simplicity reasons, a new name (CPERIOD) is given by the user to that new attribute, without the need of using the rename operation.

The same query can be answered also in a better way, avoiding a number of rename operations and using a natural join operation instead of the cartesian product operation used in version 1. The natural join is performed between two relations that have the common subrelation COURSE at the top level in common (see Case 4 in subsection 5.3.11).

2nd version:

$$\pi_t^{\pi}(((\sigma_t^{\sigma}(\pi_t^{\pi}(\rho^{\rho}[SNAME\leftarrow SNAME1](T\_DEPT))(SNAME1,COURSE))_{SNAME1=\text{“Peter”}})\ \triangleright\triangleleft_t^{\triangleright\triangleleft}$$
$$(\sigma_t^{\sigma}(\pi_t^{\pi}(\rho^{\rho}[SNAME\leftarrow SNAME2](T\_DEPT))(SNAME2,COURSE))_{SNAME2=\text{“Helen”}}))CN,CN\_PER_t)$$

CN	CN_PER$_t$
3.5	[1/1/1998, 28/10/1998)

Fig. 6.28: The resulting relation of the 2nd version of Query 22

In Tansel's model ([Tan97]) this question can be answered by the following query:

$$\pi_{CN1}(\S\cap,_{CN1,CN2}((\pi_{SNAME1,CN1}(\sigma_{SNAME1.v=\text{“Peter”}}(\mu_{COURSE1}(\mu_{COURSE\_DETAILS1}(\mu_{STAFF1}(T\_DEPT1))))))$$
$$\underset{CN1.V=CN2.V}{\triangleright\triangleleft}\quad(\pi_{SNAME2,CN2}(\sigma_{SNAME2.v=\text{”Helen”}}(\mu_{COURSE2}(\mu_{COURSE\_DETAILS2}(\mu_{STAFF2}(T\_DEPT2))))))))$$

Rename operations are omitted from the above query for simplicity reasons.

CN1
<{[1/1/1998, 28/10/1998)}, 3.5>

Fig. 6.29: The resulting relation of Query 22 in Tansel's model

It is clear that extra operations cannot be avoided in Tansel's model. In particular, six unnest operations and a slice operation are needed to answer the above query.

6.4 Summary

A variety of different kinds of queries has been presented in the algebraic format of the NRM and TNM. The examples illustrate a wide variety of applications of the management of nested and temporal data. For each example, the result table is given as well as a brief explanation to justify why this particular query has been chosen.

A number of these queries has also been expressed in other models. The superiority of the NRM and TNM algebras compared to them is thus shown and discussed. In particular, TNM has been compared with Tansel's model. Tansel's model has been chosen among all other temporal models since it is the only one that provides full nested support as TNM does.

All examples have explicitly shown that queries in the NRM and TNM are simple but at the same time powerful, short but at the same time complete, naturally presented but at the same time effective. Therefore, the full expressive power of both models presented in this thesis, NRM and TNM, has been demonstrated.

CHAPTER 7

MAPPING THE CONVENTIONAL RELATIONAL MODEL (CRM) TO THE TNM

7.1 Introduction

Although the majority of the new models that have been defined until now are claimed to be supersets of the Conventional Relational Model, or in other words, consistent extensions of the CRM, a formal proof is rarely provided ([Lor88]).

In this chapter, a complete and formal proof is provided, in order to show that the model proposed in this thesis, the TNM, is a consistent extension of the CRM. For this purpose, the properties of the NRM, also introduced in this thesis, are used.

Firstly, a brief introduction to comparisons of different database models is presented and the method that is going to be adopted in the following sections in order to do the mapping of the two models is discussed. Next, all the properties of the CRM are reviewed. Following that, the properties of the NRM, defined in chapter 4 of this thesis, are presented. Then, the NRM is proved to be a superset of the CRM. Subsequently, all the properties of the TNM are given comprehensively after which, the TNM is proved to be a superset of the NRM. Finally, it is proved that the TNM is a superset of the CRM.

7.2 Comparisons of Database Models

In the literature, a number of different approaches for comparing two database models have been presented. Most frequently, a newly defined model is compared to the Conventional Relational Model (namely, the snapshot model ([MS91]), in order to prove that the new model is an extension of the CRM. However, in reality, very few proposed models have provided a formal proof of this proposition. All the others are simply limited to claiming the above proposition without any verification.

One method of comparing two database models is to determine mappings between these data models ([TL82]). There are four basic features of the database models, which have to be compared as part of the overall mapping of the two models and they are the following:

- the structures,
- the constraints,
- the operations and
- the databases of the model.

In [TL82], eight different types of mappings are presented depending on whether the mapping is constructive – i.e. a database (instance) according to one schema is mapped to another database (instance) according to another schema – or not, if the two schemas use two different data models or the same data model and finally if the operations are included or not in the mapping. The type of mapping between the CRM and the TNM that has to be proved, belongs to the category of database cooperation mapping, since it is constructive, the two schemas use two different data models and the operations are included in the mapping.

In what follows, the mapping between the CRM and the TNM consists in comparing the basic characteristics of the two models, which are:

- the data types (the underlying domains of attributes),
- the databases,
- the structures,
- the operators,
- the operations and
- the functions of each model.

This list is partially modified from the list proposed in [TL82]. In particular, the constraints that have been mentioned in [TL82] are omitted since all the possible constraints which are applied to the CRM can also be applied to the TNM. Three other properties of a data model have been added in this list, the data types, the operators and the functions. They are included here since they have also been considered in XRM (the Interval-Extended Relational Model, [Lor88]) which is a subset of TNM. For this reason, data types, operators and functions must be considered for the mapping between the TNM and the CRM, in addition to the other three basic characteristics of database models.

More specifically, regarding the operations, for temporal data models a specific approach has been used to prove that a newly proposed algebra is a consistent extension of the snapshot algebra, when two different database models are compared. The approach which has been mentioned in [MS91], uses the *snapshot reducibility property* with the assistance of the function *Transforms* which converts a snapshot relation instance to its temporal equivalent. The *snapshot reducibility property* says that the same relation is obtained either by applying a snapshot operator to a snapshot relation and then applying the function *Transforms* to the result or by applying firstly the function *Transforms* to the snapshot relation and then applying the temporal operator to the result ([Lle94]). A number of researchers have used this approach to show that the algebras they have defined are supersets of the snapshot algebra ([Deb94], [Lle94]).

In the following sections, the data types, databases, structures, operators, operations and functions of each model are presented and then, these properties are compared between themselves respectively in order to prove that the newly defined model, the TNM, is a superset of the CRM.

7.3 The Conventional Relational Model (CRM)

The components of the CRM are described below.

7.3.1 Data types-Domains

In the relational model, data types and domains are two similar concepts. A domain is a user or system-defined data type. In fact, in order to define a domain, the data type from which the data values forming the domain are drawn, must be specified ([EN00]).

Formally, a domain is a set of values.

Definition 7.1 (*Atomic attribute domain*) The domain of an atomic attribute R_a, DOM(R_a), is DOM(R_a) $\subseteq$ D, where D is the underlying database domain.

Consequently, $D = DOM(R_{a1}) \times DOM(R_{a2}) \times \ldots \times DOM(R_{ak})$ where $R_{a1}, R_{a2}, \ldots, R_{ak}$ are the atomic attributes of all relations in the CRM and $k \geq 1$.

7.3.2 Databases

Data, in the databases of the CRM, are perceived as relations, in at least 1NF format. An example relation in the CRM is shown in Fig. 7.1.

COMPANY	TRN	CN	Y
Apple	Jack	1	75
Apple	Mark	3	82
IBM	Tim	3	82
IBM	Tim	5	79
Microsoft	Karen	2	77

Fig. 7.1: A relation in the CRM

7.3.3 Structures

According to the CRM, the scheme of a conventional relation R is $R(R_{a1}, R_{a2}, \ldots, R_{ak})$ where $R_{a1}, R_{a2}, \ldots, R_{ak}$ are the (atomic) attributes of R and $k \geq 1$.

7.3.4 Relational Operators

All the well-known comparison operators, {=, ?, <, =, >, =}, are supported in the CRM. New operators can also be defined with the aid of these standard operators.

7.3.5 Operations

The relational operations are the well-known and well-defined union, difference, intersection, projection, selection, rename, cartesian product, and natural join operations of the CRM. The division operation can be omitted since it is not a primitive operation and so it can be expressed in terms of other primitive operations (difference, cartesian product and projection). The definitions of all the others above-mentioned operations are not included in this section since they are standard definitions.

7.3.6 Functions

All the well-known functions, i.e. {+, -, *}, are supported in the CRM. New functions can also be defined with the aid of these standard functions.

7.4 The Nested Relational Model (NRM)

The components of the NRM are described below.

7.4.1 Data types-Domains

Domains are data types of arbitrary internal complexity ([Dat00]). Therefore, such domains can consist of relation-type values. Attributes defined on that domains are relation-valued attributes, that is, they contain values that are relations.

The domain of a nested attribute is defined recursively below.

Assume that R_{n1}, R_{n2}, ..., R_{nk} are, in general, all the atomic and nested attributes that belong to nested attribute R_n and P is the powerset of a set S.

Definition 7.2 (*Nested attribute domain*) The domain of a nested attribute R_n, $DOM(R_n)$, is defined recursively as

i) $DOM(R_n) \subseteq D$, where D is the underlying database domain, for the special case where R_n is an atomic attribute. .

ii) $DOM(R_{n1}) \times DOM(R_{n2}) \times \ldots \times DOM(R_{nk})$, for $k \geq 1$, where R_{n1}, R_{n2}, ..., R_{nk} are atomic attributes of R_n.

iii) $P(DOM(R_{n1})) \times P(DOM(R_{n2})) \times \ldots \times P(DOM(R_{nk}))$, for $k \geq 1$, where R_{n1}, R_{n2}, ..., R_{nk} are nested attributes of R_n, in general.

Note: An atomic attribute can be considered as a special case of a nested attribute - case (i).

7.4.2 Databases

In the NRM, databases are sets of nested relations. Nested relations do not satisfy the 1NF assumption. A relation in the NRM is shown in Fig. 7.2. This relation is the equivalent relation in the NRM of the example relation in the CRM of Fig. 7.1.

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	1	75
	Mark	3	82
IBM	Tim	3	82
		5	79
Microsoft	Karen	2	77

Fig. 7.2: A relation in the NRM (TRAINING_5)

7.4.3 Structures

Definition 7.3 (*Nested Relation Scheme*) The scheme of a relation R in the NRM is defined recursively as

$RS = R(R_1S_1, R_2S_2, \ldots, R_nS_n)$, where $n \geq 1$, $R_1, R_2, \ldots, R_n$ are the attribute names of R, either atomic or relation-valued and

$$S_i = \begin{cases} \varnothing \text{ (empty set)} & \text{if } R_i \text{ is an atomic attribute} \\ (R_{i1}S_{i1}, R_{i2}S_{i2}, \ldots, R_{ik}S_{ik}) & \text{if } R_i \text{ is a nested attribute and } k \geq 1 \end{cases}$$

where $1 \leq i \leq n$.

Example 7.1: The scheme of relation TRAINING_5 (Fig. 7.2) is TRAINING_5(COMPANY TRAINER(TRN C(CN Y))).

7.4.4 Relational Operators

The set of conventional relational comparison operators of the CRM, {=, ?, <, =, >, =}, is also supported in the NRM.

7.4.5 Operations

The union, difference, intersection, projection, selection, rename, cartesian product, natural join and Θ-join recursive operations of the NRM have been defined formally in chapter 4. Two additional operations, nest and unnest, have also been defined in the NRM.

7.4.6 Functions

The set of functions in the CRM is also supported in the NRM.

7.5 Mapping the CRM to the NRM

The components of the CRM and the NRM, that have been described in sections 7.3 and 7.4 respectively, are going to be mapped in this section, in order to prove that the NRM is a proper superset of the CRM.

7.5.1 Data types - Domains

Proposition 7.1 The set of domains in the CRM is a proper subset of the set of domains in the NRM.

Proof: The nested attribute domain is defined recursively (Definition 7.2). Therefore, from that definition, for the special cases i) where k=0, i.e. the attribute is atomic or ii) where $k \geq 1$, i.e. the attribute is nested consisting of atomic attributes only (which can be considered as a flat relation), the nested attribute domain definition of the NRM is reduced to the atomic attribute domain definition of the CRM (Definition 7.1).

Consequently, since the set of domains in the NRM can be reduced, for specific special cases, to the set of domains in the CRM, the former is a proper superset of the set of domains in the CRM.

7.5.2 Databases

Proposition 7.2: The set of databases in the CRM is a proper subset of the set of databases in the NRM.

Proof: Databases in the NRM have been introduced in order to relax the 1NF assumption that is satisfied in the CRM. Thus, the 1NF assumption of flat relations is a special case of the general N1NF assumption which characterises relations in the NRM. By definition, a flat relation is also a relation of the nested model. Therefore, the set of databases in the NRM is a proper superset of the set of databases in the CRM.

7.5.3 Structures

Proposition 7.3: The set of structures in the CRM is a proper subset of the set of structures in the NRM.

Proof: The definition of the scheme in the NRM is given recursively (Definition 7.3). For the special case, where S_i, for all i, is equal to the empty set, the definition is reduced to that of the CRM, since all attributes of the relation are atomic.

7.5.4 Relational Operators

Proposition 7.4: The set of relational comparison operators in the CRM is isomorphic to the set of relational operators in the NRM.

Proof: The proof is omitted for obvious reasons.

7.5.5 Operations

In the following, it is shown by a number of propositions that each operation in the NRM is an extended operation of the relevant operation in the CRM. Before this is done, some preliminary discussion is necessary, regarding the effect of relational operations to the key of relations.

Let *Unary* be a unary operation and let R1 = *Unary*(R0). Then, the first obvious remark is that this operation does not have any effect on the key of R0, i.e. the key of R0 remains the same. The second one is that the key of R0 is not inherited to R1. These observations apply to any data model, and to the CRM as well. As an example of the second remark, consider a flat relation R0 and assume that K is its primary key. Then the CRM select operation $R1=s_F(R0)$, also yields a flat relation, R1. Since R1 is a subset of R0, it follows that it does not contain two distinct tuples with identical values for K. However, it is not implied by this fact that K is also the key of R1, it is only the user who may specify what the key of R1 is.

As another example, let the scheme of R0 be R0(K, A, B), where K is its key. If $R1=p_{A,B}(R0)$, it is known that R1 does not contain duplicate tuples and, definitely, it is again the user who may specify its key.

Hence, the conclusion is that a unary CRM operation does not affect the key (if defined) of the input relation and it does not propagate it to the result relation. This same conclusion can also be drawn for binary operations of the CRM. Subsequently, the same conclusion can be drawn for any operation in any data model, therefore for all the operations of either the NRM or the TNM.

Proposition 7.5: The union operation in the NRM is an extended version of the union operation in the CRM.

Proof: The union operation in the NRM is defined recursively (Definition 4.9). From the recursive definition, it is deduced that for the special case where the relations are in 1NF format, the definition is reduced to the non-recursive union definition for flat relations (case i), since the relations do not contain any nested attributes. This definition then, is the definition of the union operation in the CRM.

Proposition 7.6: The difference operation in the NRM is an extended version of the difference operation in the CRM.

Proof: The proof is similar to that of Proposition 7.5.

Proposition 7.7: The intersection operation in the NRM is an extended version of the intersection operation in the CRM.

Proof: The proof is similar to that of Proposition 7.5.

Proposition 7.8: The projection operation in the NRM is an extended version of the projection operation in the CRM.

Proof: From Definition 4.13 (case ii):

$$\pi^{\pi}(r(R_{a1}, \ldots, R_{ak}, R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})) = \{ t \mid (\exists\, t_r \in r)$$
$$((t[R_{a1}] = t_r[R_{a1}]) \wedge \ldots \wedge (t[R_{ak}] = t_r[R_{ak}])$$
$$\wedge\, (t[R_{n1}] = \pi^{\pi}(t_r[R_{n1}]L_{n1})) \wedge \ldots \wedge (t[R_{nm}] = \pi^{\pi}(t_r[R_{nm}]L_{nm})))\}.$$

For the special case where relation r is flat, since all attributes of relation r are atomic, $R_{ni}L_{ni} = \varnothing$, for all i ($1 \leq i \leq m$), and the definition of the projection operation is reduced to:

$$\pi^{\pi}(r(R_{a1}, \ldots, R_{ak}, R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})) = \pi^{\pi}(r(R_{a1}, \ldots, R_{ak})) =$$
$$\{ t \mid (\exists\, t_r \in r)\ ((t[R_{a1}] = t_r[R_{a1}]) \wedge \ldots \wedge (t[R_{ak}] = t_r[R_{ak}]))\},$$

which is the definition of the projection operation in the CRM. So, the projection operation in the NRM is an extended version of the projection operation in the CRM.

Proposition 7.9: The selection operation in the NRM is an extended version of the selection operation in the CRM.

Proof: The proof is similar to that of Proposition 7.8. For the special case where relation r is flat, since all attributes of relation r are atomic, L is empty and Definition 4.15 is reduced to:

iii) $\sigma^{\sigma}(r_c L_{\sigma}) = \sigma(r_c) = \sigma(r_{ca1, \ldots, cak}) = \{ t \mid (\exists\, t_r \in r)$
$((t[Attr(R) - \{R_{a1}, \ldots, R_{ak}\}] = t_r[Attr(R) - \{R_{a1}, \ldots, R_{ak}\}])$
$\wedge ((t[R_{a1}] = t_r[R_{a1}]) \wedge c_{a1} = true)$
$\wedge \ldots \wedge ((t[R_{ak}] = t_r[R_{ak}]) \wedge c_{ak} = true))\}$,

which is the traditional selection operation for flat relations in the CRM.

Proposition 7.10: The rename operation in the NRM is an extended version of the rename operation in the CRM.

Proof: From Definition 4.20-case (ii), the rename of a nested attribute at the top level of a relation is:

$$\rho[A \leftarrow A'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, \bigcup_{k=0}^{m} L_{A' \rightarrow Ak}, B, \ldots, Z\}.$$

This definition is reduced to:

$\rho[A \leftarrow A'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, A', B, \ldots, Z\}$, for the special case where the attribute to be renamed, A, is an atomic attribute at the top level of relation R, since $\bigcup_{k=0}^{m} L_{A' \rightarrow Ak} = A'$ (m=0, i.e. there are not any descendants of A). This is equivalent to the rename operation in the CRM.

Proposition 7.11: The cartesian product operation in the NRM is an extended version of the cartesian product operation in the CRM.

Proof: Case (i) (or case (ii), for L=Ø) of Definition 4.22 is the traditional cartesian product operation for flat relations in the CRM.

Proposition 7.12: The natural join operation in the NRM is an extended version of the natural join operation in the CRM.

Proof: The natural join (Definition 4.27) which operates for cases where the common atomic or nested attributes belong to different subrelations and at different nesting levels in the two relations), $\bowtie^{\bowtie}$ (rL, qM), is defined as follows:

$$\begin{aligned}\bowtie^{\bowtie}(rL, qM) = \{\, t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ & ((t[Attr(Q_{i'}(\ldots(Q_{i+1})))] = t_q[Attr(Q_{i'}(\ldots(Q_{i+1})))])\\ &\wedge (t[Attr(R_i)] = t_r[Attr(R_i)])\\ &\wedge (t[Attr(Q_i)] = t_q[Attr(Q_i)])\\ &\wedge (t[R_{i1}Q_{i1}] = \bowtie^{\bowtie}(t_r[R_{i1}]L_{i1},\ t_q[Q_{i1}]M_{i1})))\}\end{aligned}$$

This natural join can be reduced to the conventional natural join for flat relations if the special case is assumed, where the common attributes R_{i1} and Q_{i1} are atomic attributes at the top level of the two relations and thus, $t_q[Attr(Q_{i'}(\ldots(Q_{i+1})))]$ is empty and $t[R_{i1}Q_{i1}] = \bowtie^{\bowtie}(t_r[R_{i1}]L_{i1}, t_q[Q_{i1}]M_{i1}) = \bowtie(t_r[R_{i1}], t_q[Q_{i1}])$, since L_{i1} and M_{i1} are empty.

Formally, the above definition, for L and M empty, is reduced to:

$$\begin{aligned}\bowtie^{\bowtie}(rL, qM) = \bowtie(r, q) = \{\, t \mid (\exists\, t_r \in r)\ (\exists\, t_q \in q)\ & ((t[Attr(R_i)] = t_r[Attr(R_i)])\\ &\wedge (t[Attr(Q_i)] = t_q[Attr(Q_i)])\\ &\wedge (t[R_{i1}] = t_r[R_{i1}] = t_q[Q_{i1}]))\}\end{aligned}$$

which is the traditional definition of the natural join operation in the CRM.

7.5.6 Functions

Proposition 7.13: The set of functions in the CRM is isomorphic to the set of functions in the NRM.

Proof: The proof is omitted for obvious reasons.

Proposition 7.14: The NRM is a superset of the CRM.

Proof: This is a result of Propositions 7.1-7.13 since, as it has been explained in section 7.2, in order to prove that a database model is a superset of another database model, it is necessary and sufficient to prove that every property of the latter (data types, databases, structures, operators, operations and functions) is also a property of the former.

7.6 The Temporal Nested Model (TNM)

The components of the TNM are described below.

7.6.1 Data types-Domains

The set of underlying domains in the TNM is similar to the set of underlying domains in the NRM (see section 7.4.1), augmented with the set of domains of temporal elements.

The domain of a temporal nested attribute is defined recursively below.

Let R_{tn} be a temporal nested attribute of R and $\{R_{tn1}, R_{tn2}, \ldots, R_{tnk}\}$ all the attributes of R_{tn}, in general ($k \geq 0$). Let also, P(TE) be the powerset of TE, the temporal elements. Then,

Definition 7.4 (*Temporal nested attribute domain*) If R_{tn} is a temporal nested attribute and i is its nesting level, then $DOM(R_{tn})$, the domain of R_{tn}, is:

(i) for $i = 0$, $DOM(R_{tn}) \subseteq D$, where D is the underlying domain ($k=0$)

(ii) for $i = 1$, $DOM(R_{tn1}) \times DOM(R_{tn2}) \times \ldots \times DOM(R_{tn(k-1)}) \times DOM(R_{tnk}) = DOM(R_{tn1}) \times DOM(R_{tn2}) \times \ldots \times DOM(R_{tn(k-1)}) \times P(TE)$ since it is assumed that $R_{tn1}, R_{tn2}, \ldots, R_{tn(k-1)}$ are atomic attributes of R_{tn} and R_{tnk} is the temporal attribute of R_{tn} ($k \geq 1$).

(iii) for $i > 1$, $P(DOM(R_{tn1})) \times P(DOM(R_{tn2})) \times \ldots \times P(DOM(R_{tnk}))$, where $k \geq 1$.

7.6.2 Databases

In the TNM, databases are sets of temporal nested relations. A relation in the TNM is shown in Fig. 7.3. This relation is the temporal analogue relation of the example relation in the NRM of Fig. 7.2.

COMPANY	TRAINER			
	TRN	C		
		CN	Y	CN_PER_t
Apple	Jack	1	75	[3/2/1975, 6/5/1975) ∪ [10/9/1975, 20/12/1975)
	Mark	3	82	[23/3/1982, 17/7/1982)
IBM	Tim	3	82	[1/4/1982, 15/10/1982)
		5	79	[1/9/1979, 4/11/1979)
Microsoft	Karen	2	77	[8/6/1977, 27/8/1977)

Fig. 7.3: A relation in the TNM

7.6.3 Structures

The scheme of a relation R in the TNM is defined recursively as in the NRM (Definition 7.3). The definition remains the same, even when the recursive procedure reaches nesting levels where temporal attributes are. The temporal attributes are regarded as typical atomic attributes for the definition of the scheme of a relation in the TNM.

7.6.4 Relational Operators

The set of all the well-known relational operators of the CRM, {=, ?, <, =, >, =}, is also supported in the TNM. Furthermore, additional operators are also supported in the TNM, i.e. BEFORE, AFTER, MEETS, OVERLAPS, COVERS, mentioned in section 6.3. More specifically, the comparison operators that involve attributes of a domain other than temporal element also remain the same in the model defined. Besides, for comparisons between temporal attributes, the operators that are used are: =, ≠, BEFORE, AFTER, MEETS, COVERS, OVERLAPS.

7.6.5 Operations

The TUnion, TDifference, TIntersection, TProjection, TSelection, TCartesianProduct and TNaturalJoin recursive operations of the TNM have been defined formally in chapter 5. Furthermore, a new operation, the TimeSlice operation, has been also defined in that chapter.

7.6.6 Functions

The set of all the functions in the CRM is also supported in the TNM when atomic attributes are involved. However, these functions cannot be applied to temporal elements. TNM has to be extended to include functions between temporal elements.

7.7 Mapping the NRM to the TNM

The components of the NRM and the TNM, that have been described in sections 7.4 and 7.6 respectively, are going to be mapped in this section, in order to prove that the TNM is a proper superset of the NRM.

7.7.1 Data types - Domains

Proposition 7.15 The set of domains in the NRM is a proper subset of the set of domains in the TNM.

Proof: Definition 7.4, for the special case where the temporal nested attribute, R_{tn}, does not include any temporal attributes is reduced to Definition 7.2. Consequently, from the definition, since the set of domains in the TNM can be reduced for some certain cases, to the set of domains in the NRM, the former is a proper superset of the corresponding set of domains in the NRM.

7.7.2 Databases

Proposition 7.16: The set of databases in the NRM is a proper subset of the set of databases in the TNM.

Proof: Databases in the TNM are sets of temporal nested relations (see section 7.6.2). For the special cases, where all the data that they contain are invariable over time, the relations do not include any temporal attributes and they are converted to relations isomorphic to the nested relations of the NRM.

7.7.3 Structures

Proposition 7.17: For each structure in the NRM there is at least one structure in the TNM.

Proof: See section 7.6.3.

7.7.4 Relational Operators

Proposition 7.18: The set of relational operators in the TNM is a proper superset of the set of relational operators in the NRM.

7.7.5 Operations

Proposition 7.19: The operations in the TNM are extended versions of the corresponding operations in the NRM.

Proof: By its definition (chapter 5), the TNM is an extension of the NRM. Each operation of the TNM can be reduced to the corresponding operation of the NRM for cases where the relations that participate in the operations do not contain any temporal attributes (i.e. all data are invariable over time).

7.7.6 Functions

Proposition 7.20: The set of functions in the TNM is a proper superset of the set of functions in the NRM.

Proof: All the CRM functions are used in the NRM and in the TNM when atomic attributes are involved. In other words, the functions that involve attributes of a domain other than temporal element do remain the same in the models defined.

Proposition 7.21: The TNM is a superset of the NRM.

Proof: This is implied from Propositions 7.14-7.20.

7.8 Mapping the CRM to the TNM

It is now time to prove the following proposition.

Proposition 7.20: The TNM is a superset of the CRM.

Proof: The following proposition is true:

$$\text{TNM} \supset \text{NRM (Proposition 7.21)} \qquad \text{(a)}$$

In addition, the TNM has been defined as an extension of the XRM ([Lor88]), since for all nesting levels in relations in the TNM, where the temporal attributes occur, the behaviour at these levels is the same as in relations in the XRM and the operations are defined precisely as those in [Lor88] for the XRM. Consequently, the following proposition is also valid:

$$\text{TNM} \supset \text{XRM} \qquad \text{(b)}$$

From (a) and (b) is concluded the following:

$$\text{TNM} \supset \text{NRM} \cup \text{XRM} \qquad \text{(c)}$$

Furthermore, it is true that:

$\text{NRM} \supset \text{CRM}$ (Proposition 7.14), (d)

$\text{XRM} \supset \text{CRM}$ (from [Lor88]) (e)

From (c), (d) and (e) is deduced that:

$\text{TNM} \supset \text{CRM}$

7.9 Summary

In this chapter it has been shown that the TRN is a superset of the CRM. The general method that has been adopted for this proof is the mapping of two data models presented in [TL82]. The basic features of the two models that have been compared are the data types, the databases, the structures, the operators, the operations and the functions of the models. The comparisons have formally proved that the CRM is a subset of the TNM or using different words, that the TNM is a consistent extension of the CRM.

CHAPTER 8

COMPARISON WITH OTHER MODELS

8.1 Introduction

In chapter 2, a number of different database models have been presented and discussed, nested as well as temporal. In this chapter, these models are grouped into different categories, according to some of their basic properties. Thus, four tables are given in section 8.2 where these models are classified according to their characteristics.

A number of criteria which can be used to evaluate the relative merits of some of the most important temporal models which have been proposed throughout the years, are presented in 8.3. These criteria have been derived from previous research in this field (see [Mck88]). However, in what follows, some of them have been revised, since approaches to the evaluation of temporal models have advanced. The criteria have also been restricted to address valid time algebras only. In addition, they have been grouped into four general categories according to their semantics.

Finally, the evaluation of valid time algebras against these criteria is given in section 8.4 where TNM is also included and compared to other previous proposed algebras. The advantages of TNM against the other algebras can thus be demonstrated. A discussion follows that explains and comments on the results.

8.2 Classification of Models

A detailed analysis of different proposed data models has been presented and discussed in chapter 2 of this thesis. These models are examples of either nested models or temporal models. These areas form two important research topics in database systems. The present thesis has tried to join these two different fields by proposing a new temporal and nested data model, the TNM.

In what follows, some of the most interesting proposed models are classified according to various criteria. These criteria have been discussed in chapter 3 of this thesis where the design decisions were explained and justified.

- The table in Fig. 8.1 groups the models according to two basic characteristics, the static or temporal features of the models and the normal or nested form of the models.

	1NF	**N1NF**
Non-Temporal (Static)	Codd	Scheck and Scholl Özsoyoglu, Özsoyoglu and Matos Abiteboul and Bidoit Roth, Korth and Silberschatz Colby Deshpande and Larson Liu, Ramamohanarao and Chirathamjaree Levene Garani
Temporal	Snodgrass Lorentzos	Tansel Gadia Clifford McKenzie Jensen and Snodgrass TSQL2 Garani

Fig. 8.1: Classification of relational database models

It can be seen that the majority of researchers have chosen the nested form in preference to the first normal form since, although it is more complicated to define and use, it provides a more effective way of defining a database model.

- Another table is given in Fig. 8.2 where static nested models are classified according to whether they support one level or multiple levels of nesting.

One level of nesting	**Many levels of nesting**
Scheck and Scholl	Abiteboul and Bidoit
Özsoyoglu, Özsoyoglu and Matos	Roth, Korth and Silberschatz
	Colby
	Deshpande and Larson
	Liu, Ramamohanarao and Chirathamjaree
	Levene
	Garani

Fig. 8.2: Classification of nested models

- The third table, Fig. 8.3, groups temporal nested models according to whether they are fully N1NF models (second column) or they are N1NF models only in the way they incorporate the temporal dimension (first column). They both correspond to what Clifford calls temporally grouped models (TG) in [CCT94], namely "models that provide built-in support for the grouping of related temporal values".

Partly N1NF (Temporal dimension)	**Fully N1NF**
Gadia	Tansel
Clifford	Garani
McKenzie	
Jensen and Snodgrass	
TSQL2	

Fig. 8.3: Classification of temporal nested models

• Temporal models are divided into tuple timestamping and attribute timestamping models in Fig. 8.4.

Tuple timestamping	**Attribute timestamping**
Snodgrass	Tansel
Jensen and Snodgrass	Gadia
TSQL2	Clifford
	McKenzie
	Lorentzos
	Garani

Fig. 8.4: Classification of temporal models

8.3 Evaluation Criteria

Several researchers have introduced criteria which must be satisfied by each of the new proposed temporal database models ([CT85], [Sno87], [Mck88]). In this section, these criteria are going to be discussed, analysed and classified according to their semantics. A number of these criteria will be shown to be inappropriate for the evaluation of the relative merit of the temporal algebras. In addition, the criteria are restricted to those that concern algebras which support valid time and so some of the previously proposed criteria are not applicable to the present discussion.

The most detailed presentation of the desirable criteria for a temporal model can be found in [Mck88]. Therefore, these criteria are listed in Fig. 8.5, for a later discussion.

1. All attributes in a tuple are defined for the same interval(s)
2. Consistent extension of the snapshot algebra
3. Data periodicity is supported
4. Each collection of valid attribute values is a valid tuple
5. Each set of valid tuples is a valid relation state
6. Formal semantics is specified
7. Has the expressive power of a temporal calculus
8. Historical data loss is not an operator side-effect
9. Implementation exists
10. Includes aggregates
11. Incremental semantics defined
12. Intersection, Θ-join, natural join and quotient are defined
13. Is, in fact, an algebra
14. Model doesn't require null attribute values
15. Multi-dimensional timestamps are supported
16. Optimisation strategies are available
17. Reduces to the snapshot algebra
18. Restricts relation states to first-normal form
19. Supports a three-dimensional visualisation of historical states and operations
20. Supports basic algebraic equivalence
21. Supports relations of all four classes (snapshot, rollback, historical or temporal relations)
22. Supports scheme evolution (transaction time model)
23. Supports static attributes
24. Supports rollback operations (rollback relations must be able to roll back to past states for query evaluation)
25. Treats valid time and transaction time orthogonally
26. Tuples, not attributes, are timestamped
27. Unique representation of each historical state
28. Unisorted (not multisorted)
29. Update semantics is specified

Fig. 8.5: Criteria for evaluating temporal algebras in [Mck88]

A number of criteria in [Mck88] (Fig. 8.5) are mutually incompatible. Specifically, the following criteria are mutually incompatible: criterion 19 and criteria 26, 1 and 20; criteria 5 and 27; criteria 19 with 27 and 18 (an algebra can be defined that satisfies any two of these criteria but not all three simultaneously).

Since only valid time algebras are considered in the present thesis some of the criteria listed in Fig. 8.5 are not applicable and so can be omitted. These are criteria 21, 22, 24 and 25. Furthermore, criteria 11, 19 and 29 are also excluded from the present discussion since they are outside the scope of the present work although they could be the subject of future research.

Additionally, criteria 1, 18, 26 and 27 are considered inappropriate. This results from the various advantages that heterogeneous N1NF attribute timestamping temporal models offer, as has been explained in section 3.3 of this thesis. The fact that the majority of the proposed temporal models are heterogeneous N1NF attribute timestamping models, in spite of the complicated definitions of their algebraic operators, makes out a case for the above decision. Therefore, criterion 27 is not included and criteria 1, 18 and 26 have been reversed. Hence, they are converted to the following criteria:

Criterion 1: Heterogeneous tuples are supported

Criterion 18: Relations are in N1NF

Criterion 26: Attributes are timestamped

The full list of the revised criteria can now be found in Fig. 8.6.

1. Heterogeneous tuples are supported
2. Consistent extension of the snapshot algebra
3. Data periodicity is supported
4. Each collection of valid attribute values is a valid tuple
5. Each set of valid tuples is a valid relation state
6. Formal semantics is specified
7. Has the expressive power of a temporal calculus
8. Historical data loss is not an operator side-effect
9. Implementation exists
10. Includes aggregates
11. Intersection, Θ-join and natural join are defined
12. Is, in fact, an algebra
13. Model doesn't require null attribute values
14. Multi-dimensional timestamps are supported
15. Optimisation strategies are available
16. Reduces to the snapshot algebra
17. Relations are in N1NF
18. Supports basic algebraic equivalence
19. Supports static attributes
20. Attributes are timestamped
21. Unisorted (not multisorted)
22. Recursive definition of operations

Fig. 8.6: Compatible criteria for evaluating valid time algebras

The criteria are now mutually compatible in contrast to the set of criteria in [Mck88], where certain subsets are incompatible as has been mentioned above.

Please note that criterion 17 implies fully N1NF relations (see Fig. 8.3) and also that quotient operation is not included in criterion 11 since it can be derived from other operations and it has not been defined in any of the temporal database models proposed to date, as far as the author of this thesis is aware. One additional criterion has been included in the list, concerning the recursive definition of operations. The advantages of the recursive algebraic definitions compared to the corresponding non-recursive ones have been discussed in section 4.3 of this thesis.

The 22 criteria of table in Fig. 8.6 can now be classified into 4 major categories according to their semantics. These categories concern the simplicity, the formality and expressiveness of the algebras, the representation choices and the support of some additional characteristics and are shown in Fig. 8.7.

- **Simplicity of the snapshot model**
  Criteria: 2, 16, 19
- **Formally defined algebra**
  Criteria: 6, 8, 10, 11, 12, 18, 22
- **Representation properties**
  Criteria: 1, 4, 5, 13, 17, 20, 21
- **Support of remainder characteristics**
  Criteria: 3, 7, 9, 14, 15

Fig. 8.7: Classification of criteria

8.4 Evaluation of Valid Time Algebras

As discussed in chapter 2 of this thesis, various researchers have proposed temporal models. These models differ from each other in a number of characteristics concerning the representation as well as the definitions of the algebra supported and some other additional features for each of these models.

In what follows, a comparison of different temporal database models is made based on the criteria listed in section 8.3 (Fig. 8.6). Nine temporal database models are compared and evaluated against these criteria. These models are the models presented in chapter 2 of this thesis since they are considered to be the most important ones. Most of the researchers that proposed these models have produced a number of papers throughout the years of their research. Consequently, they proposed a temporal model which they subsequently improved. The most recent versions of these models are taken into consideration.

In the following table (Fig. 8.8) the references in which a description of each of these models can be found, together with the model name, are given for each of the researchers.

Identifier	**Citation**	**Data Model**
Tansel	[Tan93], [Tan97]	TRA and TRC
Gadia	[GN98]	Parametric model
Clifford	[CCGT95]	M_{TGhi}
McKenzie	[Mck88]	-
Snodgrass	[Sno87], [Sno93], [SGM93]	TQUEL
Jensen and Snodgrass	[Jen00]	BCDM
Lorentzos	[LM97]	IXSQL
Snodgrass et al.	[Sno95]	TSQL2
Garani	present thesis	TNM

Fig. 8.8: Temporal data models

			1NF		Partly N1NF					Fully N1NF	
	No.	Criteria	Snodgrass	Lorentzos	Gadia	Clifford	McKenzie	Jensen and Snodgrass	TSQL2	Tansel	Garani
Simplicity of the snapshot model	1.	Consistent extension of the snapshot algebra	Y	Y	Y	Y	Y	Y	Y	?	Y
	2.	Reduces to the snapshot algebra	?	P	?	Y	Y	Y	Y	Y	Y
	3.	Supports static attributes	N	Y	N	Y	Y	N	Y	Y	Y
Formally defined algebra	4.	Formal semantics is specified	N	Y	P	P	Y	Y	P	P	Y
	5.	Includes aggregates	Y	Y	N	N	Y	N	P	N	Y
	6.	Historical data loss is not an operator side-effect	?	Y	?	P	Y	N	N	?	Y
	7.	Intersection and join are defined	N	N	P	N	Y	Y	Y	N	Y
	8.	Is, in fact, an algebra	N	Y	N	N	Y	Y	Y	Y	Y
	9.	Supports basic algebraic equivalences	N	Y	P	P	P	P	P	P	Y
	10.	Recursive definition of operations	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N	Y
Representation properties	11.	Heterogeneous tuples are supported	N	Y	N	N	Y	N	N	Y	Y
	12.	Each collection of valid attribute values is a valid tuple	N	Y	N	N	N	N	N	Y	Y
	13.	Each set of valid tuples is a valid relation state	Y	Y	N	N	N	N	N	Y	Y
	14.	Model doesn't require null attribute values	Y	Y	N	N	Y	Y	Y	Y	Y
	15.	Relations are in fully N1NF	N	N	N	N	N	N	N	Y	Y
	16.	Attributes are timestamped	N	Y	Y	Y	Y	N	N	Y	Y
	17.	Unisorted	Y	Y	N	N	N	N	Y	Y	Y
Support of remainder characteristics	18.	Data periodicity is supported	N	Y	N	N	N	N	N	N	N
	19.	Implementation exists	Y	P	N	N	P	N	P	N	P
	20.	There is an equivalent temporal calculus	Y	N	N	Y	Y	N	Y	Y	N
	21.	Algebraic transformation optimisation strategies are available	P	Y	Y	N	P	P	N	N	P
	22.	Multi-dimensional timestamps are supported	N	Y	N	N	N	N	N	N	Y

Y: Yes P: Partially satisfied

N: No ?: Not specified

N/A: Not applicable

Fig. 8.9: Evaluation of valid time algebras against specific criteria

In Fig. 8.9 the valid time algebras proposed by Tansel, Gadia, Clifford, McKenzie, Snodgrass, Jensen and Snodgrass, Lorentzos, Snodgrass et al. (TSQL2) and Garani have been evaluated against the 22 criteria presented in section 8.3. The following results can be derived:

Snodgrass's model is a tuple calculus and query language rather than an algebra. Consequently, formal semantics are not defined for the algebra and so it is not clear if it can reduce to the snapshot algebra. Static attributes are not supported since the model uses tuple timestamping. Also, it is not clear whether or not historical data loss is an operator side effect and whether it supports basic algebraic equivalence. The model is homogeneous since it uses tuple timestamping and relations are in 1NF. Any collection of valid attribute values is not a valid tuple since the implicit attributes that specify the end points of a tuple's timestamp must be time ordered. Data periodicity and multi-dimensional tuples are not supported. Optimisation techniques are investigated in the context of TQUEL in [AS86].

Lorentzos's model partially satisfies criterion 2 that TRA reduces to the snapshot algebra given that the reduction to the snapshot algebra could be achieved by the insertion of null values since attribute timestamps are heterogeneous. In Lorentzos' model time-varying and time-invariant attributes are allowed. Also, no temporal calculus is proposed. Historical versions of intersection and join are not defined but they can be deduced. Relations in Lorentzos's model must be in 1NF. Multi-dimensional timestamps seem to be supported in his algebra, although he does not discuss this particular effect in [LM97]. Finally, optimisation strategies are available.

Gadia's parametric model is a homogeneous model where all attributes are timestamped. As such, all attribute values are defined over a specific time period and so they cannot behave as static attributes but as time-varying ones. Another consequence of homogeneity is that, in general, each set of valid attribute values is not a valid tuple, because the valid time components of the attributes do not, in general, fulfil the homogeneity property. Any set of tuples does not form a valid relation since relations in the parametric model are required to have keys which play a vital role in the model. It is not clear from the definition of the model in [GN98] if the model can be reduced to the snapshot algebra nor if it has the expressive power of a temporal calculus. Formal semantics is only partially specified for the algebra. The majority of the algebraic operations have been defined informally. Intersection and T-join are not defined and natural join is only informally defined. Basic algebraic tautologies are partially supported as shown in [GN98].

It is not clear if historical data loss is an operator side effect. The homogeneity assumption imposes the requirement for null attribute values. Aggregates are not included in the model.

Gadia's model is multisorted since it includes three types of expressions, relational expressions, domain expressions and Boolean expressions.

No implementation has been reported.

Optimisation issues are taken into account in his algebra and an algorithm is defined which transforms a query to an equivalent more efficient one. Data periodicity is not considered and the same is also true for multi-dimensional timestamps.

In Clifford's model formal semantics is specified for some of the operators although some operators are not defined at all, as for example the intersection and join operations. Consequently, it is unclear if historical data loss is an operator side effect or not for these operations. For all the other operations defined in [CCGT95] this criterion is satisfied.

Basic algebraic equivalences seem to hold in his model since the basic temporal operators defined in [CCGT95] correspond to the standard relational operators of the snapshot relational algebra. However, it is not fully clear. Each collection of valid attribute values is not a valid tuple since the homogeneity property needs to be satisfied; moreover, each set of valid tuples is not a valid relation state because key attributes cannot have the same time components for two equal key values.

The algebra is multisorted since some operations return relations but one operation, lifespan, returns a scalar value, i.e. not a relation. Data periodicity, multi-dimensional timestamps and aggregates are not supported. The same stands also for an implementation and an optimisation scheme.

Overall, Clifford's model ([CCGT95]) is clearly superior to the one he previously proposed in [CC87] and which was evaluated in [Mck88] and [Lor88].

McKenzie includes an evaluation of his model ([Mck88]) against the specific criteria in [Mck88]. A brief description is also included here for completeness reasons and since the list with the criteria has been slightly modified.

The model allows non-homogeneous attribute timestamps; therefore, it can reduce to the snapshot algebra only through the introduction of distinguished nulls when taking snapshots. The distributive property of the cartesian product operation over difference is not supported in McKenzie's model. All the other basic algebraic equivalences are applied. Value-equivalent tuples are not allowed in a relation; therefore, by reason of this restriction, any arbitrary collection of valid attribute tuples is not a valid relation state. Any set of valid attribute values does not form a valid tuple since the algebra does not allow empty timestamps for all attributes in the same tuple.

The algebra is multisorted since it defines operators on both snapshot states and historical states.

Data periodicity and multi-dimensional timestamps are not supported. An implementation (a prototype of the algebra without aggregates) has been undertaken. Optimisation strategies based on the algebraic equivalences are available. However, other optimisation techniques have not been investigated but only briefly discussed.

Jensen and Snodgrass's BCDM model is defined as a consistent extension of the snapshot algebra and since it is a tuple timestamping model, it satisfies the criterion that it reduces to the snapshot algebra. In a tuple timestamping model such as this one, static attributes cannot be supported since every attribute value in a tuple has a temporal element associated with it. An equivalent temporal calculus is not defined in their model. Historical data loss is an operator side effect of their cartesian product operation since the model is tuple timestamping and the cartesian product is defined using intersection semantics. Basic algebraic equivalences are supported only for the operations that have been defined as extensions of the corresponding snapshot operations. Each tuple consists of a number of explicit attribute values and an implicit timestamp value; thus, every collection of valid attribute values does not guarantee that it can create a valid tuple. It is a homogeneous model since tuples are timestamped. However, it is N1NF since the timestamps associated with the tuples can be sets of time chronons. Value-equivalent tuples are not allowed in the model and so any arbitrary set of valid tuples cannot be a valid relation state.

The model is multisorted with the following object types: valid time relation states, transaction time relation states and bitemporal relation states. Data periodicity, aggregates and multi-dimensional timestamps are not supported. Optimisation strategies can be performed since the model is defined as an extension of the snapshot model but no more details are given. Finally, an implementation and an equivalent calculus do not exist.

In TSQL2 formal semantics, aggregates and basic algebraic equivalences are partially included. Historical data loss is an operator side-effect since the cartesian product operation is defined using intersection semantics and so the valid time components of each relation are restricted in the result relation. Only homogeneous tuples are supported in the model. Each collection of valid attribute values is not a valid tuple since each tuple consists of a number of explicit attribute values and an implicit timestamp value. Each set of valid tuples is not a valid relation state since value-equivalent tuples are not allowed in the model. Relations are partly in N1NF since only time can be expressed as a set of time instants. In TSQL2 tuples are timestamped. Data periodicity and multi-dimensional timestamps are not supported. TSQL2 has been implemented partly through the development of Tiger, an advanced temporal database system ([BJ96]). Finally, algebraic transformation optimisation strategies are not available.

Tansel does not prove that his algebra (TRA) is a consistent extension of the snapshot algebra. Formal semantics is not specified for some of his algebraic operations (i.e. union, intersection, difference, projection, cartesian product). He claims that they are defined in exactly the same way as they are in the relational algebra. However, temporal atoms need a special treatment when these operations are performed in his model. The join operation is not formally defined. Basic algebraic equivalences are not discussed in his model.

Nevertheless, since in his definition of the algebra he claims that some operations are the same as traditional relational operations and the commutative, associative and distributive equivalences hold, it could be said that criterion 9 is partially supported. All the operations of his algebra are defined non-recursively.

There is no support of temporal aggregates and implementation and optimisation issues are not included. Data periodicity and multi-dimensional timestamps are not supported.

TNM supports all the evaluation criteria but two. These are data periodicity and a temporal calculus. A detailed justification follows where the satisfying criteria are explained by cross-references to earlier sections.

TNM is proved to be a consistent extension of the snapshot algebra in section 5.3 where all the operations are formally defined as extended versions of the corresponding operations in the NRM and as a consequence of the corresponding operations in the CRM (see also Proposition 7.19 in section 7.7.5 and Propositions 7.5-7.12 in section 7.5.5).

The reducibility to the snapshot algebra can also be derived from section 5.3 and chapter 7.

By definition, static attributes are supported in the TNM model. The same is also true for crtiteria 11, 15 and 16.

Historical data loss is not an operator side-effect in TNM, since all valid time information input to an operator is preserved in the operator’s output as can be easily proved by the formal definitions of the operations, unless the operation being performed dictates removal (e.g. time-slice, intersection). In addition, the cartesian product operation has not been defined using intersection semantics (as is the case in other models, e.g. [Gad88]]) and therefore, historical data is preserved in that operation as well.

Formal semantics is specified in chapter 5. Aggregates are included in section 4.3.13. Intersection and join operations are defined in sections 5.3.3 and 5.3.11-12 respectively.

It is, in fact, an algebra, since the types of the objects supported, as well as the allowable operations have been defined. In addition, all operations are closed, as it is proved in section 5.4.

All the operations of the TNM have been defined recursively in chapter 5.

Basic algebraic equivalences are also supported in TNM. For example, the following algebraic equivalences can easily be proved to hold.

$R \cup_t^{\cup} Q \equiv Q \cup_t^{\cup} R$

$R \times_t^{\times} Q \equiv Q \times_t^{\times} R$

$\sigma_t^{\sigma}((\sigma_t^{\sigma}(r_{c_1}))_{c_2}) \equiv \sigma_t^{\sigma}((\sigma_t^{\sigma}(r_{c_2}))_{c_1})$

$R \cup_t^{\cup} (Q \cup_t^{\cup} S) \equiv (R \cup_t^{\cup} Q) \cup_t^{\cup} S$

$\sigma_t^{\sigma}((R \cup_t^{\cup} Q)_c) \equiv \sigma_t^{\sigma}(R_c) \cup_t^{\cup} \sigma_t^{\sigma}(Q_c)$

In the TNM, each collection of valid attribute values is a valid tuple, since the value of an attribute is independent of the value of other attributes in a tuple, except for key. This is also possible of the fact that tuples of valid relations in the model can be heterogeneous. Any other attribute dependence constraints are not imposed in TNM.

Also, each set of valid tuples is a valid relation state, since there are not any constraints in the way a TNM relation has been defined, except the fact that tuples with identical values for all their atomic attributes are coalesced; therefore, tuples with identical values for their atomic attributes can neither overlap nor be adjacent in time.

TNM is a heterogeneous model and as such, null attribute values are not required.

It is unisorted, since it defines only one object type, the temporal nested relation. All operations take as input and provide as output a single type of object, the temporal nested relation.

In addition, optimisation strategies are only available when based on the algebraic equivalences, since basic algebraic equivalences already provides algebraic transformation optimisations. A detailed study of optimising algebraic expressions can be found in [Gra84].

Multi-dimensional timestamps are supported. Although there are not such examples in the thesis, there are not any constraints in the allowed number of temporal attributes associated with an attribute in relations of the TNM model. On the contrary, all the operations have been defined in such a way to support more than one temporal attributes connected to the same attribute (section 5.3).

Overall, the following can be noticed:

- TNM performs better than the majority of other temporal database models.
- TNM does not support data periodicity; however, this is outperformed by the fact that it is a nested model and therefore more powerful.
- TNM is comparable with Tansel's model, but one major advantage is that its operations are recursive and this simplifies the formulation of queries.

8.5 Summary

The plethora of different models proposed in the area of temporal databases demands the evaluation of these models and their comparison. Temporal database models must satisfy a minimum set of properties. These properties concern the preservation of the simplicity of the snapshot model, the algebra supported by each model, the representation capabilities of the model and some additional characteristics referring to the proposed models.

In this chapter nine different temporal database models have been evaluated against 22 criteria. These criteria are well defined and compatible. It has been shown that the model proposed in this thesis, TNM, satisfies the large majority of the criteria and exceeds all the other temporal models.

CHAPTER 9

CONCLUSION AND FUTURE RESEARCH

9.1 Concluding Remarks

In spite of the considerable activity in the area of temporal databases in the last two decades and the plethora of new proposed temporal database models, no temporal database model has achieved the same level of acceptance as Codd's relational model in the world of conventional databases.

Different temporal database models have been proposed. They differ significantly in the proposed structure of their relations relating to incorporation of the temporal component as well as in the algebras they define. They utilise either 1NF tuple timestamping, 1NF attribute timestamping, N1NF tuple timestamping or N1NF attribute timestamping relations supporting either time points, time intervals or temporal elements. They all present a number of advantages as well as a number of deficiencies.

However, the majority of N1NF attribute timestamping models do not include nesting of data other than temporal data. Therefore, their relations can be nested only in the way they incorporate the temporal dimension. This limits their expressive power and representational capabilities. Because of the complexity, little research has been done in the area of "real" N1NF attribute timestamping temporal database models where all sort of data can be nested ([Tan97]). Therefore, there is still a lot of work to be done in this specific area of temporal database research concerning the structure of nested relations and the corresponding algebra.

The research reported in this thesis has attempted to fill this gap by defining a new temporal nested valid time relational model, the TNM. TNM is an attribute timestamping heterogeneous model which supports temporal elements not as part of the temporal atoms as in [Tan97] but as distinct temporal attributes. Additional operations to extract the temporal part of a temporal atom are thus avoided since the temporal nested version of the traditional projection operation can be used instead. Moreover, in TNM, unnest and nest operations do not need to be performed before or after the execution of any other operation that concerns an attribute that is not at the top level of the relation, as is the case in [Tan97]. Operations can be performed at any level of the TNM relations directly.

TNM has been defined as a superset of NRM, a new Nested Relational Model also been defined in this thesis. NRM, in its turn, is a superset of the Conventional Relational Model. This is an important property of the two new models.

The algebras of both NRM and TRM have been defined recursively. The major contributions relating to the NRM are the formal definitions of the rename and natural join operations. Particularly for the natural join operation, a generalised natural join operation has been defined that can join any pair of joinable nested relations. The generalised natural join operation for nested relations uses one or more of the six cases of natural join which have also been defined in the thesis. These cases can be distinguished by the type of the common attributes, i.e. atomic or nested attributes and their positions (nesting levels) in the relation scheme.

This generalised natural join operation has been extended to support the temporal dimension. Consequently, the temporal nested generalised natural join operation has been defined for the TNM. All the other operations of the TNM have also been formally defined and proved to be closed, which is an important property of the proposed model.

The expressive power of TNM has been demonstrated by a number of examples.

Finally, TNM has been compared with eight other temporal models using a set of 22 compatible criteria. The advantages of TNM over other models have been illustrated in chapter 8.

In conclusion, this research proposes a temporal database model that combines the nested features with the temporal dimension to generalise temporal relational databases.

9.2 Future Research

It is not claimed in the thesis that all the issues related to the problem of defining a temporal database model have been resolved. Further research is still needed in several areas.

The areas for future work are briefly discussed below:

- **Support of transaction time**

Transaction time concerns the time an event is stored in the database. Transaction time has not been studied as much as valid time in the literature. Therefore, an interesting direction for research is the extension of TNM to support, in addition, transaction time.

- **Definition of a query language**

As briefly described in Chapter 1 of the thesis, several attempts have been made to define a temporal query language, for example TQuel ([Sno93]), SQL^T ([Tan93b]) and TSQL2 ([Sno95]). Further research is needed to define a query language as an extension of SQL to support the temporal nested features of TNM and to evaluate it in comparison to other existed temporal query languages.

- **Optimisation strategies**

Optimisation techniques for the efficient evaluation of queries in the TNM can be developed. Particularly for the generalised temporal nested natural join operation further research is needed so that it can be optimised.

- **Functional dependencies**

Temporal functional dependencies have been studied by several researchers ([TG89], [NA89], ([Lor91]). A review of the different types of dependencies proposed for temporal databases can be found in [JSS94]. An interesting direction of research is the study of functional dependencies for the TNM. Since TNM combines temporal and nested features, previous research of both temporal and nested functional dependencies must be considered for this study.

- **Management of spatial data**

Spatial databases (the handling of data related to space in the databases) have been an active area of research over the last two decades in parallel with temporal databases. Spatial databases have been studied either independently or when integrated with temporal databases (spatiotemporal databases). An interesting topic in this field is the development of a spatiotemporal model and query language.

Therefore, the incorporation of spatial data to TNM is an additional challenge. It seems likely that TNM can be extended to provide a spatiotemporal nested model by including the spatial dimension in an analogous way to the temporal dimension.

- **Temporal extensions in XML**

XML (Extensible Markup Language) is emerging as the new standard for the exchange of data with structures on the Web. Therefore, its main characteristic is that it is naturally nested. Structures in XML can be nested to any finite depth.

Since the contents of XML documents may change with time and past versions of them may also be of interest, the definition of a data model for temporal XML documents is important. Therefore, given that TNM supports nested data, a promising direction of research is to use the results arising from this research, to extend the XML standard to include temporal functionalities.

REFERENCES

[AB84] Abiteboul S. and Bidoit N. Non First Normal Form Relations to Represent Hierarchical Organized Data. *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS),* Waterloo, Ontario, Canada, 191-200 (1984).

[AB86] Abiteboul S. and Bidoit N. Non First Normal Form Relations: An Algebra Allowing Data Restructuring. *Journal of Computer and System Sciences*, **33**(3), 361-393 (1986).

[Ahn93] Ahn I. SQL+T: A Temporal Query Language. *Proceedings of the Infrastructure for Temporal Databases*, Arlington (1993).

[Ari86] Ariav G. A Temporal Oriented Data Model. *ACM Transaction on Database Systems*, **11**(4), 499-527 (1986).

[AS86] Ahn I. and Snodgrass R. Performance Evaluation of a Temporal Database Management System. *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, Washington, DC, 96-107 (1986).

[BADW82] Bolour A., Anderson T., Dekeyser L. and Wong H. The Role of Time in Information Processing: A Survey. *ACM SIGMOD Record*, **12**(3), 27-50 (1982).

[Ben82] Ben-Zvi J. *The Time Relational Model*. Ph.D. Dissertation, University of California, Los Angeles (1982).

[BG93] Bhargava G. and Gadia S. Relational Database Systems with Zero Information Loss. *IEEE Transactions on Knowledge and Data Engineering*, **5**(1), 76-87 (1993).

[BJ96] Böhlen M. and Jensen C. Seamless Integration of Time into SQL. *Technical Report R-96-2049*, Department of Computer Science, Aalborg University, Denmark (1996).

[BJW00] Bettini C., Jajodia S. and Wang S. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Berlin: Springer-Verlag (2000).

[CC87] Clifford J. and Croker A. The Historical Relational Data Model

(HRDM) and Algebra Based on Lifespans. *IEEE 3rd International Conference on Data Engineering,* Los Angeles, California, 528-537 (1987).

[CCGT95] Clifford J., Croker A., Grandi F. and Tuzhilin A. On Temporal Grouping. In [CT95], 194-213 (1995).

[CCT94] Clifford J., Croker A. and Tuzhilin A. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, **19**(1), 64-116 (1994).

[CCT96] Clifford J., Croker A. and Tuzhilin A. On Data Representation and Use in a Temporal Relational DBMS. *Information Systems Report*, **7**(3), 308-327 (1996).

[Cli82] Clifford J. A Model for Historical Databases. *Proceedings of Logical Bases for Data Bases*, Toulouse, France (1982).

[CMP95] Clack C., Myers C. and Poon E. *Programming with Miranda.* Prentice Hall (1995).

[Col90] Colby L.S. A Recursive Algebra for Nested Relations. *Information Systems*, **15**(5), 567-582 (1990).

[CT85] Clifford J. and Tansel A.U. On an Algebra For Historical Relational Databases: Two Views. *Proceedings of the 3rd International Workshop on Statistical and Scientific Databases*, Austin, Texas, 247-265 (1985).

[CT95] Clifford J. and Tuzhilin A. (Eds.) *Recent Advances in Temporal Databases. Proceedings of the International Workshop on Temporal Databases.* Zürich, Switzerland: Springer-Verlag, (1995).

[CW83] Clifford J. and Warren D.S. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems,* **8**(2), 214-254 (1983).

[Dat00] Date C.J. *An Introduction to Database Systems (7th edition),* Addison-Wesley Publishing Company (2000).

[DDL03] Date C.J., Darwen H. and Lorentzos N.A. *Temporal Data and the Relational Model.* Morgan Kaufmann Publishers, 2003.

[Deb94] Debabrata D. *A Design Methodology for Temporal Databases.* Ph.D. Dissertation, University of Rochester (1994).

[DL87] Deshpande V. and Larson P.A. An Algebra for Nested Relations. *Technical Report CS-87-65*, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (1987).

[DL91] Deshpande V. and Larson P.A. An Algebra for Nested Relations with Support for Nulls and Aggregates. *Technical Report CS-91-16*, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (1991).

[DS85] Dayal U. and Smith J.M. PROBE: A Knowledge-Oriented Database Management System. *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, Islamorada, 227-257 (1985).

[EJS98] Etzion O., Jajodia S. and Sripada S.M. (Eds.) *Temporal Databases: Research and Practice*. (the book grows out of a Dagstuhl Seminar, June 23-27, 1997). Lecture Notes in Computer Science 1399, Berlin: Springer-Verlag (1998).

[EN00] Elmasri R. and Navathe S.B. *Fundamentals of Database Systems (3rd edition)*. Addison-Wesley (2000).

[FT83] Fisher P.C. and Thomas S.J. Operators for Non-First-Normal Form Relations. *Proceedings of the IEEE Computer Society's 7th International Conference on Computer Software and Applications (COMPSAC)*, Chicago, Illinois, 464-475 (1983).

[Gad86a] Gadia S.K. Toward a Multihomogeneous Model for a Temporal Database. *Proceedings of the International Conference on Data Engineering*, Los Angeles, California, 390-397 (1986).

[Gad86b] Gadia S.K. Weak Temporal Relations. *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Cambridge, Massachusetts, 70-77 (1986).

[Gad88] Gadia S.K. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, **13**(4), 418-448 (1988).

[Gad92] Gadia S.K. A Seamless Generic Extension of SQL for Querying Temporal Data. *Technical Report TR-92-02*, Computer Science Department, Iowa State University (1992).

[Gar00a] Garani G. The Temporal Nested Model (TNM) and its Algebra.

University of London Publications, ISBN: 0718716388 (2000).

[Gar00b] Garani G. Evaluation of Non-First Normal Form Database Models. *University of London Publications*, ISBN: 071871640X (2000).

[Gar00c] Garani G. Temporal Database Models: A Critical Approach. *University of London Publications*, ISBN: 0718716396 (2000).

[GJ00a] Garani G. and Johnson R. Nest and Unnest in Nested Relations Revisited. *University of London Publications*, ISBN: 071871637X (2000).

[GJ00b] Garani G. and Johnson R. Joining Nested Relations and Subrelations. *Information Systems*, **25**(4), 287-307 (2000).

[GN98] Gadia S.K. and Nair S.S. Algebraic Identities and Query Optimisation in a Parametric Model for Relational Temporal Databases. *IEEE Transactions on Knowledge and Data Engineering*, **10**(5), 793-807 (1998).

[GNP92] Gadia S.K., Nair S.S. and Poon Y.C. Incomplete Information in Relational Temporal Databases. *Proceedings of the 18th International Conference on Very Large Data Bases*, Vancouver, Canada, 395-406 (1992).

[Gra84] Gray P. *Logic, Algebra and Databases*. Ellis Horwood Limited, 1984.

[GV85] Gadia S.K. and Vaishnav J.H. A Query Language for a Homogeneous Temporal Database. *Proceedings of the 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Portland, Oregon, 51-56 (1985).

[GY88] Gadia S.K. and Yeung C.S. A Generalised Model for a Relational Temporal Database. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, 251-259 (1988).

[GY91] Gadia S.K. and Yeung C.S. Inadequacy of Interval Timestamps in Temporal Databases. *Information Sciences*, **54**, 1-22 (1991).

[JCG+92] Jensen C.S., Clifford J., Gadia S.K., Segev A. and Snodgrass R.T. A Glossary of Temporal Database Concepts. *SIGMOD Record*, **21**(3), 35-43 (1992).

[JDB+98] Jensen C.S., Dyreson C.E., Böhlen M, Clifford J., Elmasri R., Gadia S.K., Grandi F., Hayes P., Jajodia S., Käfer W., Kline N., Lorentzos N.A., Mitsopoulos Y., Montanari A., Nonen D., Peressi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio P. and Wiederhold G. The Consensus Glossary of Temporal Database Concepts-February 1998 Version. In [EJS98], 367-405 (1998).

[Jen00] Jensen C.S. *Temporal Database Management*. Dr. Techn. Thesis, Aalborg University, Denmark (2000).

[JG95] Johnson R. and Garani G. A Temporal Database Model Using Nested Relations. *Technical Report No. 9518*, Birkbeck College, University of London (1995).

[JG96] Johnson R. and Garani G. A Temporal Database Model Using Nested Relations. Revised Edition, *Technical Report No. 9608*, Birkbeck College, University of London (1996).

[JG97] Johnson R. and Garani G. Joining Nested Sub-Relations. *Technical Report No. 9701*, Birkbeck College, University of London (1997).

[JJ92] Jang Y.P. and Johnson R. A Heterogeneous Temporal Nested Relational Data Model. *InTeRel Report No. 5*, Department of Computer Science, Birkbeck College, University of London (1992).

[JL87] Johnson R. and Lorentzos N.A. Temporal Data Management. *Information Update Database Technology*, Pergamon Infotech **1**, 5-11 (1987).

[JS82] Jaeschke G. and Schek H.J. Remarks on the Algebra of Non First Normal Form Relations. *Proceedings of the ACM Symposium on Principles of Database Systems*, Los Angeles, California, 124-138 (1982).

[JS92] Jensen C. and Snodgrass R. The TEMPIS Project. Proposal for a Data Model for the Temporal Structured Query Language. *TEMPIS Technical Report No. 37*, Department of Computer Science, University of Arizona, Tuscon (1992).

[JS94] Jensen C. and Snodgrass R. Temporal Specialization and

Generalization. *IEEE Transactions on Knowledge and Data Engineering*, **6**(6), 954-974 (1994).

[JS96a] Jensen C. and Snodgrass R. Semantics of Time-Varying Information. *Information Systems*, **21**(4), 311-352 (1996).

[JS96b] Jensen C. and Snodgrass R. Semantics of Time-Varying Attributes and their Use for Temporal Database Design. *Proceedings of the 5th International Conference on EDBT,* Avignion, France, 366-377 (1996).

[JS99] Jensen C. and Snodgrass R. Temporal Data Management. *IEEE Transactions on Knowledge and Data Engineering*, **11**(1), 36-44 (1999).

[JSS92] Jensen C., Soo M. and Snodgrass R. Unification of Temporal Data Models. *Technical Report TR 92-15*, Department of Computer Science, University of Arizona, Tuscon (1992).

[JSS94] Jensen C., Snodgrass R. and Soo M. Extending Existing Dependency Theory to Temporal Databases. *Technical Report R-94-2050*, Department of Mathematics and Computer Science, Aalborg University, Denmark (1994).

[JSST01] Jensen C.S., Schneider M., Seeger B. and Tsotras V.J. (Eds.) *Advances in Spatial and Temporal Databases: Proceedings of the 7th International Symposium (SSTD)*, Redondo Beach, CA, USA. Lecture Notes in Computer Science 2121, Berlin: Springer-Verlag (2001).

[Kli93] Kline N. An Update of the Temporal Database Bibliography. *SIGMOD Record*, **22**(4), 66-80 (1993).

[Klu82] Klug A. Equivalence of Relational Algebra and Relational Calculus Query Languages having Aggregate Functions. *Journal of the ACM,* **29**(3), 699-717 (1982).

[LC96] Liu H.-C. and Chirathamjaree C. An Efficient Join for Nested Relational Databases. *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA '96),* Zurich, Switzerland, 289-301 (1996).

[LD98] Lorentzos N.A. and Dondis A. Query by Example for Nested Tables. *Proceedings of the 9th International Conference on*

Database and Expert Systems Applications (DEXA '98), Vienna, Austria, 716-725 (1998).

[Lev92] Levene M. *The Nested Universal Relation Database Model.* Lecture Notes in Computer Science 595, Berlin: Springer-Verlag (1992).

[LJ87] Lorentzos N.A. and Johnson R. TRA: A Model for a Temporal Relational Algebra. *Proceedings of the Conference on Temporal Aspects in Information Systems*, Sophia-Antipolis, France, 95-108 (1987).

[LJ88a] Lorentzos N.A. and Johnson R. An Extension of the Relational Model to Support Generic Intervals. *Proceedings of the International Conference on Extending Database Technology* (EDBT' 88), Venice, Italy, 528-542 (1988).

[LJ88b] Lorentzos N.A. and Johnson R. Extending Relational Algebra to Manipulate Temporal Data. *Information Systems*, **13**(3), 289-296 (1988).

[LK89] Lorentzos N.A. and Kollias V. The Handling of Depth and Time Intervals in Soil Information Systems. *Computers in Geosciences*, **15**, 395-401 (1989).

[LL91] Levene M. and Loizou G. Correction to Null Values in Nested Relational Databases by M.A. Roth, H.F. Korth and A. Silberschatz. *Acta Informatica*, **28**, 603-605 (1991).

[Lle94] Llewellyn M.J. *Temporal Extensions to the Relational Data Model*. Ph.D. Dissertation. University of Central Florida (1994).

[LM94] Lorentzos N.A. and Manolopoulos Y. Efficient Management of 2-d Interval Relations. *Proceedings of the 5th International Conference on Data and Expert Systems Applications (DEXA '94)*, Athens, Greece, 72-82 (1994).

[LM95] Lorentzos N.A. and Manolopoulos Y. Functional Requirements for Historical and Interval Extensions to the Relational Model. *Data and Knowledge Engineering*, **17**(1), 59-86 (1995).

[LM97] Lorentzos N.A. and Mitsopoulos Y. SQL Extension for Interval Data. *IEEE Transactions on Knowledge and Data Engineering*, **9**(3), 480-499 (1997).

[Lor88] Lorentzos N.A. *A Formal Extension of the Relational Model for the Representation and Manipulation of Generic Intervals*. Ph.D. Thesis, Department of Computer Science, Birkbeck College, University of London, August 1988.

[Lor91] Lorentzos N.A. Management of Intervals and Temporal Data in the Relational Model. *Technical Report No. 49*, Informatics Laboratory, Agricultural University of Athens, Greece (1991).

[LPS94] Lorentzos N.A., Poulovassilis A. and Small C. Implementation of Update Operations for Interval Relations. *The Computer Journal*, **37**(3), 164-176 (1994).

[LPS95] Lorentzos N.A., Poulovassilis A. and Small C. Manipulation Operations for an Interval-Extended Relational Model. *Data and Knowledge Engineering*, **17**(1), 1-29 (1995).

[LR94a] Liu Hong-Cheu and Ramamohanarao K. Multiple Paths Join for Nested Relational Databases. *Proceedings of the 5th Australian Database Conference*, 30-44 (1994).

[LR94b] Liu Hong-Cheu and Ramamohanarao K. Algebraic Equivalences among Nested Relational Expressions. *Proceedings of the 3rd International Conference on Information and Knowledge Management* (CIKM '94), Gaithersburg, MD, USA, 234-243 (1994).

[LRT99] Lorentzos N.A., Rios Viqueira J. and Tryfona N. Quantum-Based Spatial Extension to the Relational Model. *Proceedings of the 7th Panhellenic Conference on the Informatics*, Ioannina, Greece, III 34-44 (1999).

[LSYK99] Lorentzos N.A., Sideridis A., Yialouris C. and Kollias V. An Integrated Spatiotemporal System. *Computers and Electronics in Agriculture*, **22**, 233-242 (1999).

[LTR99] Lorentzos N.A., Tryfona N. and Rios Viqueira J. Relational Algebra for Spatial Data Management. *Proceedings of the International Workshop on Integrated Spatial Databases: Digital Images and GIS*, Portland, Maine, USA, 192-208 (1999).

[Mak77] Makinouchi A. A consideration on Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model.

Proceedings of the 3rd International Conference on Very Large Data Bases, Tokyo, Japan, 447-453 (1977).

[Mck86] McKenzie E. Bibliography: Temporal Databases. *SIGMOD Record*, **15**(4), 40-52 (1986).

[Mck88] McKenzie J.E. *An Algebraic Language for Query and Update Temporal Databases*. Ph.D. Thesis, The University of North Carolina at Chapel Hill, 1988.

[MS91] McKenzie J.E. and Snodgrass R.T.: Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, **23**(4), 501-543 (1991).

[NA89] Navathe S. and Ahmed R. A Temporal Relational Model and a Query Language. *Information Sciences*, **49**, 147-175 (1989).

[OOM87] Özsoyoglu G., Özsoyoglu Z.M. and Matos V. Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. *ACM Transactions on Database Systems*, **12**(4), 566-592 (1987).

[RKB87] Roth M.A., Korth H.F. and Batory D.S. SQL/NF: A Query Language for ¬1NF Relational Databases. *Information Systems*, **12**(1), 99-114 (1987).

[RKS88] Roth M.A., Korth H.F. and Silberschatz A. Extended Algebra and Calculus for Nested Relational Databases. *ACM Transactions on Database Systems*, **13**(4), 389-417 (1988).

[RKS89] Roth M.A., Korth H.F. and Silberschatz A. Null Values in Nested Relational Databases. *Acta Informatica*, **26**, 615-642 (1989).

[RL01] Rios Viqueira J. and Lorentzos N.A. Spatio-temporal SQL Extension. *Proceedings of the 8h Panhellenic Conference on Informatics*, Nicosia, Cyprus, Vol. 1, 264-273 (2001).

[RLT01] Rios Viqueira J., Lorentzos N.A. and Tryfona N. Formalism for Spatio-temporal Data Management. *Proceedings of the 5th Hellenic European Conference on Computer Mathematics and its Applications* (2001).

[SA85] Snodgrass R. and Ahn I. A Taxonomy of Time in Databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, New York, 236-246 (1985).

[SA86] Snodgrass R. and Ahn I. Temporal Databases. *IEEE Computer*, **19**(9), 35-42 (1986).

[Sar90] Sarda N.L. Algebra and Query Language for a Historical Data Model. *The Computer Journal*, **33**(1), 11-18 (1990).

[SGM93] Snodgrass R., Gomez S. and McKenzie E. Aggregates in the Temporary Query Language TQuel. *IEEE Transactions on Knowledge and Data Engineering*, **5**(5), 826-842 (1993).

[SJS95] Segev A., Jensen C. and Snodgrass R. Report on the 1995 International Workshop on Temporal Databases. *SIGMOD Record*, **24**(4), 46-52 (1995).

[Sno00] Snodgrass R. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers (2000).

[Sno86] Snodgrass R. Research Concerning Time in Databases. Project Summaries. *SIGMOD Record*, **15**(4), 19-39 (1986).

[Sno87] Snodgrass R. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, **12**(2), 247-298 (1987).

[Sno90] Snodgrass R. Temporal Databases. Status and Research Directions. *SIGMOD Record*, **19**(4), 83-89 (1990).

[Sno92] Snodgrass R. Temporal Databases. *Proceedings of the International Conference on GIS-From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning*, Pisa, Italy, 22-64 (1992).

[Sno93] Snodgrass R. An Overview of TQuel. In [TCG+93], 141-182 (1993).

[Sno95] Snodgrass R. (ed.) *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers (1995).

[Soo91] Soo M.D. Bibliography on Temporal Databases. *SIGMOD Record*, **20**(1), 14-23 (1991).

[SS86] Schek H.-J. and Scholl M.H. The Relational Model with Relation-Valued Attributes. *Information Systems*, **11**(2), 137-147 (1986).

[SS87] Segev A. and Shoshani A. Logical Modelling of Temporal Data. *Proceedings of ACM SIGMOD Conference on Management of Data*, San Francisco, 454-466 (1987).

[SS88] Stam R.B. and Snodgrass R. A Bibliography on Temporal Databases. *IEEE Data Engineering Bulletin*, **11**(4), 53-61 (1988).

[TA86] Tansel A.U. and Arkun M.E. Aggregation Operations in Historical Relational Databases. *Proceedings of the 3rd International Workshop on Statistical and Scientific Databases*, Luxemburg, 116-121 (1986).

[Tan86] Tansel A.U. Adding Time Dimension to Relational Model and Extending Relational Algebra. *Information Systems*, **11**(4), 343-355 (1986).

[Tan87] Tansel A.U. A Statistical Interface for Historical Relational Databases. *Proceedings of the 3d International Conference on Data Engineering*, Los Angeles, California, 538-546 (1987).

[Tan93a] Tansel A.U. A Generalised Relational Framework for Modelling Temporal Data. In [TCG+93], 183-201 (1993).

[Tan93b] Tansel A.U. SQL^T: A Temporal Extension to SQL. *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX (1993).

[Tan97] Tansel A.U. Temporal Relational Data Model. *IEEE Transactions on Knowledge and Data Engineering*, **9**(3), 464-479 (1997).

[TAO89] Tansel A.U., Arkun M.E. and Özsoyoglu G. Time-by-Example Query Language for Historical Databases. *IEEE Transactions and Data Engineering*, **15**(4), 464-478 (1989).

[TC90] Tuzhilin A. and Clifford J. A Temporal Relational Algebra as a Basis for Temporal Relational Completeness. *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia 13-23 (1990).

[TCG+93] Tansel A.U., Clifford J., Gadia S., Jajodia S., Segev A. and Snodgrass R. *Temporal Database. Theory, Design and Implementation.* The Benjamin/Cummings Series on Database Systems and Applications (1993).

[TF86] Thomas S.J. and Fischer P.C. Nested Relational Structures. *Advances in Computing Research. A Research Annual. The Theory of Databases*, JAI Press Inc., **3**, 269-307 (1986).

[TG89] Tansel A.U. and Garnett L. Nested Historical Relations.

Proceedings of the ACM SIGMOD International Conference on Management of Data, Portland, Oregon, 284-293 (1989).

[TK96] Tsotras V.J. and Kumar A. Temporal Database Bibliography Update. *SIGMOD Record*, **25**(1), 41-51 (1996).

[TL82] Tsichritzis D.C. and Lochovsky F.H. *Data Models*. Prentice-Hall (1982).

[Tom96] Toman D. Point vs. Interval-based Query Languages for Temporal Databases. *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Montreal, Canada, 58-67 (1996).

[TT97] Tansel A.U. and Tin E. The Expressive Power of Temporal Relational Query Languages. *IEEE Transactions on Knowledge and Data Engineering*, **9**(1), 120-134 (1997).

[Ull95] Ullman J.D. *Principles of Database and Knowledge-Base Systems*. Computer Science Press (1995).

[WJW97] Wu Y., Jajodia S. and Wang X.S. Temporal Database Bibliography Update. In [EJS98], 338-366 (1997).

[WTWL96] Wegner L., Thelemann S., Wilke S. and Lievaart R. QBE-like Queries and Multimedia Extensions in a Nested Relational DBMS. *Proceedings of the International Conference on Visual Information Systems*, Melbourne, Australia, 437-446 (1996).

APPENDICES

APPENDIX A

FORMAL SYNTAX OF THE TNM ALGEBRA

expression

:: = one-relation-expression | two-relation-expression

one-relation-expression

:: = temporal-nested-renaming | temporal-nested-selection |
temporal-nested-projection | time-slice | temporal-unnest | temporal-nest

two-relation-expression

:: = temporal-nested-projection binary-operation expression

temporal-nested-renaming

:: = ρ_t^{ρ} [attribute-commalist1] (term)

attribute-commalist1

:: = fattribute ← fattribute | fattribute ← fattribute, attribute-commalist1

fattribute

:: = attribute1 | function4(attribute1)

attribute1

:: = attribute | nested-aggregate-attribute

attribute

:: = basic-attribute | nested-attribute

basic-attribute

:: = atomic-attribute | ta | function1(ta) nested-aggregate-attribute

:: = function3[attribute/basic-attribute]

function3

:: = N-MAX | N-MIN | N-SUM | N-COUNT | N-AVG

function4

:: = MAX | MIN | AVG | COUNT | SUM

term

:: = relation | (expression)

temporal-nested-projection

:: = π_t^{π} (term (attribute-commalist2)) | term

attribute-commalist2

:: = fattribute | fattribute, attribute-commalist2

binary-operation

:: = $\cup_t^{\cup}$ | $\cap_t^{\cap}$ | $-_t^{-}$ | $\times_t^{\times}$ | $\bowtie_t^{\bowtie}$ | $\bowtie_{t_T}^{\bowtie}$

temporal-nested-selection

:: = σ_t^{σ}(term $_{\text{comparison}}$)

comparison

:: = attribute-term | attribute-term logical-operator comparison

logical-operator

:: = AND | OR | AND NOT | OR NOT

attribute-term

:: = attribute-term1 | temporal-attribute-term attribute-term1

:: = FAA θ FAA

FAA

:: = constant | atomic-attribute | attribute-term1 | nested-aggregate-attribute

θ

:: = < | > | = | <= | >= | ≠

temporal-attribute-term

:: = FTA temporal-operator FTA | FTA

FTA

:: = constant | ta | function1(ta) | temporal-attribute-term

ta

:: = temporal-attribute1 | function2(temporal-attribute1)

temporal-attribute1

:: = temporal-attribute | temporal-attribute temporal-operator1 temporal-attrib

temporal-operator1

: = $\cup_{TE}$ | $\cap_{TE}$ | $-_{TE}$

function1

:: = MAX | MIN

function2

:: = DURATION | START | STOP

temporal-operator

:: = BEFORE | AFTER | MEETS | COVERS | OVERLAPS | =

time-slice

:: = s^{s}_{TE} (term)

TE

:: = temporal-element

temporal-unnest

:: = μ_t^{μ}(term$_{nested\text{-}attribute}$)

temporal-nest

:: = ν_t^{ν}(term$_{attribute\text{-}commalist2 \rightarrow nested\text{-}attribute}$)

APPENDIX B

PROTOTYPE IMPLEMENTATION

B.1 Introduction

To illustrate the functionality of the models defined in this thesis, a prototype implementation has been undertaken in Miranda

Miranda is a functional programming language which runs under UNIX. It is used especially in the areas of proof systems and specification, as a vehicle for rapid software prototyping, and for teaching functional programming [CMP95]. It makes use of *lazy* semantics. This permits the use of potentially infinite data structures. Miranda also supports an elegant style of problem decomposition. A program, actually a *script*, is a collection of equations defining various functions and data structures.

Issues related to the implementation, the files of Miranda, and the declaration of tables and functions are briefly discussed in this appendix. Selected parts of the code are also listed. Finally, examples presented in the thesis, occasionally with their results, are included.

B.2 Implementation

The most important operations have been implemented. In particular, the four non-temporal operators, rename, projection, selection and cartesian product have been fully implemented. The same is also true for their temporal versions. Limitations on the implementation of join are outlined as follows.

In the non-temporal nested join only one nested column is allowed at each nesting level. Within the framework of a prototype, this is considered to be a reasonable assumption, to ease implementation.

Another limitation is that all the six cases of the join operator allow joining on only one pair of columns. This is also considered to be a reasonable assumption within a prototype implementation. Note in particular that an attempt, to implement a join on two columns, would require an almost entire revision of the whole development. This is due to the fact that the join operator is a top-level function that calls other functions. Hence, changes to the structures of join would have to be reflected to the functions it calls. Given also that these functions call others, a cascade of revisions would then have to follow.

A side effect of joining on only one common column is that the temporal nested join operator has not been implemented either. The reason is that this last operator requires joining on relations that have at least two common attributes, one of which must be temporal. Note however that, if the hypothetical case of joining on two common attributes had been implemented, the temporal nested join would have required only a trivially simple additional piece of code, to compute the intersection of the corresponding temporal elements in the two temporal columns. Note on the other hand that relevant implementation, such as the union of temporal elements, has already been incorporated in the implementation of other operations, such as selection and projection.

Finally, some functions, START, STOP, COUNT etc, have not been implemented either. Given however, that these functions do not play a critical role relevant to the primary objectives of the thesis, it is considered that their omition can fully be neglected.

Overall, it is considered that, in spite of the above-stated implementation limitations, the prototype serves satisfactorily as a proof of concept, in that a fuller implementation would not contribute substantially more with respect to the objectives of this thesis.

B.2.1 Description of files

The implementation consists of fourteen (14) Miranda files, relationalFile0.m - relationalFile12.m and main.m. The main.m file includes all the other files and compiles them automatically. Some more files contain sample code, in particular all the example relations of the thesis. All the files contain comments that explain the functionality to a reasonable degree. A brief description of them is given below:

File name	**Description**
main.m	It contains all the top level calls.
relationalFile0.m	It contains type definitions used throughout this implementation.
relationalFile1.m	It includes general methods, used throughout the implementation, and time manipulation functions.
relationalFile2.m	It contains basic functions for selection, projection and cartesian product operators.
relationalFile3.m	It contains basic functions for the selection operator.
relationalFile4.m	It contains basic functions for the join operator.
relationalFile5.m	It contains basic functions for the rename operator.
relationalFile6.m	It contains case 1 of the join operator.
relationalFile7.m	It contains case 2 of the join operator.
relationalFile8.m	It contains cases 3a and 3b of the join operator.
relationalFile9.m	It contains case 4 of the join operator.
relationalFile10.m	It contains case 5 of the join operator.
relationalFile11.m	It contains cases 6a and 6b of the join operator.
relationalFile12.m	It identifies the most suitable method for operator join.

B.2.2 Declaration of tables

There are several ways to introduce tables in Miranda. The current implementation has chosen the *user-defined type* representation because it is more generic, powerful and extensible, eg. NumberColumn "Id" 1, ... A relational table is represented as a list of such entries:

```
Relation "Emp" [[NumberColumn "Id" 1, StringColumn "Name" "James"],
                [NumberColumn "Id" 2, StringColumn "Name" "Jack"], ...]
```

Column types may be numerical, char list or Boolean. Since user-defined types are used to represent columns, the column definition can be extended to any desired level of complexity, although this can occasionally be cumbersome.

Time columns are defined as follows:

```
time == (num, num, num)
timeInterval == (time, time)
temporalElement == [timeInterval]
```

Therefore, the main type definitions of a database table, columnType, are:

```
columnType ::=      NC string num |
                    SC string string |
                    BC string bool |
                    RC relationalTable|
                    TC temporalElement
```

As an example, the nested relation LOCATION (Fig. 3.8) is declared as follows.

```
location :: relationalTable
location = Relation "LOCATION"
    [[SC "COMPANY" "Toshiba", RC (Relation "ANNEX"
        [[SC "BUILDING" "North Building", SC "ADDRESS" "Porchester Rd."]])],
    [SC "COMPANY" "IBM", RC (Relation "ANNEX"
        [[SC "BUILDING" "Maple House", SC "ADDRESS" "Kendal Av."],
        [SC "BUILDING" "Main Building", SC "ADDRESS" "Danebury Rd."]])],
    [SC "COMPANY" "Microsoft", RC (Relation "ANNEX"
        [[SC "BUILDING" "Pegasus House", SC "ADDRESS" "Ashford St."],
        [SC "BUILDING" "Queen's Building", SC "ADDRESS" "Park Rd."]])]]
```

The temporal version of LOCATION relation, T_LOCATION (ref. Fig. 3.15), is declared as follows.

```
tlocation :: relationalTable
tlocation = Relation "T_LOCATION"
      [[SC "COMPANY" "Toshiba", RC (Relation "ANNEX"
          [[SC "BUILDING" "North Building", SC "ADDRESS" "Porchester Rd.",
                      TC "ADDRESS_PER" [((3, 8, 1995), (1, 1, 2010))]]])],
      [SC "COMPANY" "IBM", RC (Relation "ANNEX"
          [[SC "BUILDING" "Maple House", SC "ADDRESS" "Kendal Av.",
                      TC "ADDRESS_PER" [((17, 1, 1996), (22, 5, 1998))]],
          [SC "BUILDING" "Main Building", SC "ADDRESS" "Danebury Rd.",
                  TC "ADDRESS_PER" [((10, 6, 1998), (1, 1, 2010))]]])],
      [SC "COMPANY" "Microsoft", RC (Relation "ANNEX"
           [[SC "BUILDING" "Pegasus House", SC "ADDRESS" "Ashford St.",
                  TC "ADDRESS_PER" [((29, 10, 1994), (4, 4, 1997))]],
           [SC "BUILDING" "Queen's Building", SC "ADDRESS" "Park Rd.",
                  TC "ADDRESS_PER" [((18, 3, 1995), (1, 1, 2010))]]])]]
```

B.2.3 Functions

The main functions that have been developed are summarized below. Note that many auxiliary functions have also been developed, to support the main functions.

Function Name	**Description**
TableProjection	It selects a given subset of columns in a table.
tableProduct	It calculates the cartesian product of two tables.
rename	It renames one or more columns in a table.
selectFrom	It selects entries from a table that satisfy a condition.
selectNotIn	It selects entries from a table that do not satisfy a condition.
joinTables	It joins two tables on a pair of columns.

The temporal functions below are used to manipulate time.

Temporal function	**Description**
areDisjoint	It tests whether two time intervals are disjoint.
equals	It tests whether two time intervals are equal.
before	It tests whether a time interval is before another.
after	It tests whether a time interval is after another.
meets	It tests whether the start (end) point of the first time interval is the same with the end (first) point of the second.
inBetween	It tests whether a time point is between the start

	and end point of a time interval.
overlaps	It tests whether two time intervals overlap.
covers	It tests whether a time interval covers another time.
tUnion	It calculates the union of two time intervals.
tIntersect	It calculates the intersection of two time intervals. If the time intervals are disjoint it returns the empty list.
tDifference	It calculates the difference of two time intervals.
tECovers	It tests whether the first temporal element, which consists of one time interval, covers all the time intervals of the second temporal element.
tEAfter	It tests whether the first temporal element, which consists of one time interval, is after every time interval of the second temporal element.
tEBefore	It tests whether the first temporal element, which consists of one time interval, is before every time interval of the second temporal element.
tEMeets	It tests whether the first temporal element, which consists of one time interval, meets at least one time interval of the second temporal element.
tEOverlaps	It tests whether the first temporal element, which consists of one time interval, overlaps with at least one time interval of the second temporal element.

Some application examples of the listed functions are the following.

areDisjoint ((21, 3, 2003), (23, 3, 2003)) ((11, 4, 2001), (5, 6, 2002))
inBetween((21,3,2003),(23,3,2003))(5,6,2002)
tDifference((21,3,2003),(23,3,2003))((11,4,2001),(5,6,2002))
tEAfter((1,1,2003),(3,3,2003))[((30,4,1994),(27,8,1995)),((4,6,1997),(19,11,1998))]
tECovers((1,1,2003),(3,3,2003))[((30,4,1994),(27,8,1995)),((4,6,1997),(19,11,1998))]

B.3 Miranda Code

Part of the code, that has been developed, is listed in this section. In particular, the section contains the whole of the code in files relationalFile0.m and main.m and selected parts of the code in the remainder files.

```
|| File name: relationalFile0.m

|| This file contains type definitions used throughout this prototype implementation.
|| +++++++++++++++++++++++++++++++++++++++++++++++++++++++++

|| string: Type definition for a list of characters.

string == [char]

|| columnType: Main type definition for a database table.
|| A simple relational table here includes numerical, string and
|| boolean types. Further types (user defined) may be included.
|| NC stands for Numerical type Column.
|| SC stands for String type Column.
|| BC stands for Boolean type Column.
|| RC stands for Recursive type Column.
|| TC stands for Temporal type Column.

columnType ::= NC string num |
               SC string string |
               BC string bool |
               TC string temporalElement |
               RC relationalTable

|| ____________________________________________________________
|| tableEntry: In our model, each table entry is a list of column
|| values.

tableEntry == [columnType]

|| ____________________________________________________________
|| relationalTable: The main definition of a relational table.
|| A relational table here is created using a constructor (Relation),
|| a string identifier (tag) to hold the name of the table and a list
|| of entries for each row of the table.

relationalTable ::= Relation string [tableEntry]

|| ____________________________________________________________
|| boolFunction: Definition of all functions applied to column which
|| return True or False. These functions are used to select column
|| based on function values.

boolFunction ::=NF (num -> bool) |
               SF (string -> bool) |
               BF (bool -> bool) |
               TF (temporalElement -> bool)

|| ____________________________________________________________
|| strBoolPair: Pair definition to hold a boolean function and the
|| name tag of the column to which the function must apply.

strBoolPair == (string, boolFunction)

|| ______________________________________________________________
|| columnEntryTuple: Auxiliary tuple type of a table column
|| and a table entry.
|| Used to join two tables.

columnEntryTuple == (columnType, tableEntry)

|| ______________________________________________________________
|| doubleEntryTuple: Auxiliary tuple type of two table entries.
|| Used to join two tables.

doubleEntryTuple == (tableEntry, tableEntry)

|| ______________________________________________________________
|| stringEntryListTuple: Auxiliary tuple type of a string and a list
|| of table entries.
|| Used to join two tables.

stringEntryListTuple == (string, [tableEntry])

|| ______________________________________________________________
|| strEntryEntryTuple: Auxiliary tuple type of a string and two
|| table entries.
|| Used to join two tables.

strEntryEntryTuple == (string, tableEntry, tableEntry)

|| ______________________________________________________________
|| stringEntryTuple: Auxiliary tuple type of a string and a table Entry.
|| Used to join two tables.

stringEntryTuple == (string, tableEntry)

|| ______________________________________________________________
|| doubleEntryListTuple: Auxiliary tuple type of two lists of table
|| entries.
|| Used to join two tables.

doubleEntryListTuple == ([tableEntry], [tableEntry])

|| ______________________________________________________________
|| stringTableTuple: Auxiliary tuple of a string and a relational
|| table.
|| Used to join two tables.

stringTableTuple == (string, relationalTable)

|| ______________________________________________________________
|| joinTriple: Auxiliary tuple of a num, a bool and a string.
|| Used to join two tables.

joinTriple == (num, bool, string)
```

```
|| ______________________________________________________________
|| strStrTuple: Pair definition to hold two strings.

strStrTuple == (string, string)

|| ______________________________________________________________
|| Time-related definitions:

|| time: Defines a time point consisting of day, month and year.

time == (num, num, num)

|| ______________________________________________________________
|| timeInterval: Defines a time interval consisting of a start point
|| (included) and a stop point (excluded).

timeInterval == (time, time)

|| ______________________________________________________________
|| temporalElement: Defines a list of time intervals.

temporalElement == [timeInterval]

|| ______________________________________________________________
|| tETuple: a 2-tuple of temporal elements.

tETuple == (temporalElement, temporalElement)

|| ______________________________________________________________
|| Helper types for time manipulations:

|| timeList: Defines a list of time points.

timeList == [time]

|| ______________________________________________________________
|| doubleNum: Defines a tuple of 2 numerical values.

doubleNum == (num, num)
```

```
|| File name: relationalFile1.m

|| Includes general methods used throughout this prototype implementation.

%include "relationalFile0.m"

|| +++++++++++++++++++++++++++++++++++++++++++++++++++++++

|| addEntry: Dynamically adds a new entry to a table.

addEntry :: tableEntry -> relationalTable -> relationalTable

|| ______________________________________________________________
|| deleteEntry: Dynamiccally removes an existing entry from a table.

deleteEntry :: tableEntry -> relationalTable -> relationalTable
```

|| ____
|| remove: Multi-type function to remove an element from a list.

remove :: * -> [*] -> [*]

|| ____
|| colName: gives the column name of a column type.

colName :: columnType -> string

|| ____
|| isRecColumn: Tests a column to see whether it is a nested table.

isRecColumn:: columnType -> bool

|| ____
|| isTempColumn: Tests a column to see whether it is a temporal column.

isTempColumn :: columnType -> bool

|| ____
|| count: Couns the number of elements in a list.

count :: [*] -> num

|| ____
|| areEqual: Checks two lists for equality.
|| Two lists are equal if they have the same members
|| regardless of their orders.

areEqual :: [*] -> [*] -> bool

|| ____
|| memberOfEntryList: Checks whether an entry is a member of an entry list.

memberOfEntryList :: tableEntry -> [tableEntry] -> bool

|| ____
|| areEqualEntries: Cheks whether two entries are equals.

areEqualEntries :: tableEntry -> tableEntry -> bool

|| ____
|| isSubsetOf: Checks whether an entry is the subset of another entry.

isSubsetOf :: tableEntry -> tableEntry -> bool

|| ____
|| getColName: Top level call to remove parenthesis from column names.

getColName :: string -> string

|| ____
|| getColName2: Removes ')' from a column name.

getColName2 :: string -> string

|| ____

|| getColName3: Removes all '(' from a column name.

getColName3 :: string -> string -> string

|| ____
|| retainColNames: Removes all parenthesis and extra tags from column names
|| in a table.

retainColNames :: relationalTable -> relationalTable

|| ____
|| retainColNames2: Removes all parenthesis and extra tags from column names
|| in a table entry list.

retainColNames2 :: [tableEntry] -> [tableEntry]

|| ____
|| retainColNames3: Removes all parenthesis and extra tags from column names
|| in a table entry.

retainColNames3 :: tableEntry -> tableEntry

|| ____
|| retainColNames4: Removes all parenthesis and extra tags from a column name.

retainColNames4 :: columnType -> columnType

|| ____
|| Time manipulation functions:

|| ____
|| day: Gives the day part of a time point.

day :: time -> num

|| ____
|| month: Gives the month part of a time point.

month :: time -> num

|| ____
|| year: Gives the year part of a time point.

year :: time -> num

|| ____
|| start: Gives the start point of a time interval.

start :: timeInterval -> time

|| ____
|| stop: Gives the end point of a time interval.

stop :: timeInterval -> time

|| ____
|| tEStarts: Gives a list of start points of a given list
|| of time intervals.

tEStarts :: temporalElement -> timeList

|| ____
|| tEStops: Gives a list of end points of a given list
|| of time intervals.

tEStops :: temporalElement -> timeList

|| ____
|| compareTime: Compares 2 time points for a boolean comparison.

compareTime :: (num -> num -> bool) -> time -> time -> bool

|| ____
|| compareTime2: Compares 2 num tuples for a boolean comparison.

compareTime2 :: (num -> num -> bool) -> doubleNum -> doubleNum -> bool

|| ____
|| lessOf:: Given two time points chooses the earlier one.

lessOf :: time -> time -> time

|| ____
|| moreOf:: Given two time points chooses the later one.

moreOf :: time -> time -> time

|| ____
|| tEStartPoint: Gives the minimum start point of a list of start
|| points.

tEStartPoint :: timeList -> time

|| ____
|| tEStopPoint: Gives the maximum end point of a list of end
|| points.

tEStopPoint :: timeList -> time

|| ____
|| tEStart: Gives start point of a temporal element.

tEStart :: temporalElement -> time

|| ____
|| tEStop: Gives stop point of a temporal element.

tEStop :: temporalElement -> time

|| ____
|| areDisjoint: Checks 2 time intervals to see whether they are disjoint.

areDisjoint :: timeInterval -> timeInterval -> bool

|| ____
|| equals: Checks 2 time intervals t1 and t2 to see whether they are equal.

```
equals :: timeInterval -> timeInterval -> bool

|| ______________________________________________________________
|| before: Checks 2 time intervals t1 and t2 to see whether t1 is before t2.

before :: timeInterval -> timeInterval -> bool

|| ______________________________________________________________
|| after: Checks 2 time intervals t1 and t2 to see whether t1 is after t2.

after :: timeInterval -> timeInterval -> bool

|| ______________________________________________________________
|| meets: Checks 2 time intervals t1 and t2 to see whether t1's start point
|| (end point) is the same as t2's end point (start point).

meets :: timeInterval -> timeInterval -> bool

|| ______________________________________________________________
|| Maybe needed: \/ (compareTime (=) d a)

|| ______________________________________________________________
|| inBetween: Checks a time interval ti and a timepoint t to see whether t lies
|| between start point and end point of ti.

inBetween :: timeInterval -> time -> bool

|| ______________________________________________________________
|| overlaps: Checks 2 time intervals t1 and t2 to see whether t1 and t2 overlap.

overlaps :: timeInterval -> timeInterval -> bool

|| ______________________________________________________________
|| covers: Checks 2 time intervals t1 and t2 to see whether t1 covers t2.

covers :: timeInterval -> timeInterval -> bool

|| ______________________________________________________________
|| tUnion: Calculates the union of two time intervals.

tUnion :: timeInterval -> timeInterval -> temporalElement

|| ______________________________________________________________
|| union: Calculates the union of two temporal elements.

union :: temporalElement -> temporalElement -> temporalElement

|| ______________________________________________________________
|| union2: Checks a time interval against a list of time intervals to find
|| all possible unions.

union2 :: timeInterval -> temporalElement -> temporalElement -> tETuple

|| ______________________________________________________________
|| tIntersect: Calculates the intersection of two time intervals.
|| It will return an empty list if time intervals are disjoint.
```

tIntersect :: timeInterval -> timeInterval -> temporalElement

|| ____
|| intersect: Calculates the intersection of two temporal elements.

intersect :: temporalElement -> temporalElement -> temporalElement

|| ____
|| intersect2: Checks a time interval against a list of time intervals to find
|| all possible intersections.

intersect2 :: timeInterval -> temporalElement -> temporalElement

|| ____
|| tDifference: Calculates the difference of two time intervals.

tDifference :: timeInterval -> timeInterval -> temporalElement

|| ____
|| tDifference2: Calculates the difference of two overlapping time intervals.

tDifference2 :: timeInterval -> timeInterval -> temporalElement

|| ____
|| difference: Calculates the difference of two temporal elements.

difference :: temporalElement -> temporalElement -> temporalElement

|| ____
|| difference2: Checks a time interval against a list of time intervals to find
|| all possible differences.

difference2 :: timeInterval -> temporalElement -> temporalElement

|| ____
|| Union functions based on temporal elements:

|| ____
|| getUnion: Gets the union of two table entries based on their
|| temporal elements.
|| Calls getUnion2 function to do the actual work.

getUnion :: tableEntry -> tableEntry -> [tableEntry]

|| ____
|| extractTemp: Separates temporal and non-temporal columns of a table entry.

extractTemp :: tableEntry -> tableEntry -> tableEntry -> doubleEntryTuple

|| ____
|| getUnion2: Gets the union of two table entries by getting the union
|| of their temporal elements.

getUnion2 :: doubleEntryTuple -> doubleEntryTuple -> [tableEntry]
|| ____
|| getTempUnion: Gets the union of two lists of temporal elements.
|| Calls getTempUnion2 function to do the actual work.

getTempUnion :: tableEntry -> tableEntry -> tableEntry

|| ____
|| getTempUnion2: Given a columnType x, finds an element in a list
|| of temporal elements to get their union. If nothing is found, an
|| error is printed.

getTempUnion2 :: columnType -> tableEntry -> tableEntry

|| ____
|| getAllUnion: Gets the union of table entries at the top level.
|| Calls getAllUnion2 function.

getAllUnion :: relationalTable -> relationalTable

|| ____
||getAllUnion2: Gets the union of lists of table entries.

getAllUnion2 :: [tableEntry] -> [tableEntry] -> [tableEntry]

|| ____
|| getAllUnion3: Gets the union of a table entry and a list of table entries.

getAllUnion3 :: tabl eEntry -> [tableEntry] -> [tableEntry]

|| ____
|| merge1: Gets the union of table entries for a relational table.

merge1 :: relationalTable -> relationalTable

|| ____
|| merge2: Gets the union of table entries recursively.

merge2 :: [tableEntry] -> [tableEntry]

|| ____
|| merge3: Gets the union of two table entries.

merge3 :: tableEntry -> tableEntry

|| ____
|| merge4: Gets the union of a recursive column.

merge4 :: columnType -> columnType

|| ____
|| tEBefore: Checks a time interval (ti) and a temporal element (te) to see whether t1 is before te1.

tEBefore :: timeInterval -> temporalElement -> bool

|| ____
|| tEAfter: Checks a time interval (ti) and a temporal element (te) to see whether t1 is after te1.

tEAfter :: timeInterval -> temporalElement -> bool

|| ____
|| tEMeets: Checks a time interval (ti) and a temporal element (te) to see whether ti meets te.

tEMeets :: timeInterval -> temporalElement -> bool

|| ____
|| tEOverlaps: Checks a time interval (ti) and a temporal element (te) to see whether ti and te overlap.

tEOverlaps :: timeInterval -> temporalElement -> bool

|| ____
|| tECovers: Checks a time interval (ti) and a temporal element (te) to see whether ti covers te.

tECovers :: timeInterval -> temporalElement -> bool

|| File name: relationalFile2.m

%include "relationalFile0.m"
%include "relationalFile1.m"

|| ++

|| isColumnTag: Simple method to identify a column using its tag.

isColumnTag :: string -> columnType -> bool

|| ____
|| resolvePath: Creates full path name for columns in a table.

resolvePath:: relationalTable -> relationalTable

|| ____
|| resolvePath2: Creates full path names for a list of entries
|| given a string and a depth.

resolvePath2 :: string -> num -> [tableEntry] -> [tableEntry]

|| ____
|| resolvePath3: Creates full path names for a list of columns given
|| a string and a depth.

resolvePath3 :: string -> num -> tableEntry -> tableEntry

|| ____
|| resolvePath4: Creates full path names for a column given
|| a string and a depth.

resolvePath4 :: string -> num -> columnType -> columnType

|| ____
|| rpar: Generates a given number of closing parenthesis.

rpar :: num -> string

|| ____
|| selectCol: Used to select a column recursively based on its tag.

selectCol :: string -> columnType -> tableEntry

|| ____
|| selectEntryByStr: Selects all columns from a list of columns using
|| the given column tag.

selectEntryByStr :: string -> tableEntry -> tableEntry

|| ____
|| selectEntryByStrLst: Selects a list of columns whose names are
|| provided by a list of string tags.

selectEntryByStrLst :: [string] -> tableEntry -> tableEntry

|| ____
|| selectEntryLstByStrLst: Selects entries from a given entry list
|| whose names are provided by a list of tags.

selectEntryLstByStrLst :: [string] -> [tableEntry] -> [tableEntry]

|| ____
|| tableProjection2: Selects a subset of table entries based on given
|| column names after all recursive column names have been resolved.

tableProjection2 :: [string] -> relationalTable -> relationalTable

|| ____
|| flattenRelTable: Flattens a relational table by calling helper
|| function flattenEntryList.

flattenRelTable :: relationalTable -> relationalTable

|| ____
|| flattenEntryList: Flattens a list of table entries by calling helper
|| function flattenColumnList.

flattenEntryList :: [tableEntry] -> [tableEntry]

|| ____
|| flattenColumnList: Flattens a list of columns by calling helper
|| function flattenColumn.

flattenColumnList :: [columnType] -> [columnType]

|| ____
|| flattenColumn: Flattens recursive columns.

flattenColumn :: columnType -> [columnType]

|| ____
|| tableProduct2: Product of two relational tables at the top level.
|| All possible combinations of table entries are included.
|| No recursive application involved.

tableProduct2 :: relationalTable -> relationalTable -> relationalTable

|| ____
|| tableProduct3: Product of a table with an inner table of another table.
|| All possible combinations of table entries are included.

tableProduct3 :: relationalTable -> (string, relationalTable) -> relationalTable

|| ____
|| tableProduct4: Applies the product of a table to a list of table entries
|| for recursive application.

tableProduct4 :: relationalTable -> string -> [tableEntry] -> [tableEntry]

|| ____
|| tableProduct5: Applies the product of a table to a list of table columns
|| for recursive application.

tableProduct5 :: relationalTable -> string -> tableEntry -> tableEntry

|| ____
|| tableProduct6: Applies the product to a recursive column which holds the
|| required inner table.

tableProduct6 :: relationalTable -> columnType -> columnType

|| ____
|| tableProduct7: Applies the product to a recursive column if the inner table is
|| the one required.

tableProduct7 :: relationalTable -> string -> columnType -> columnType

|| ____
|| getRecTableNames: Gets the name of a table and calls getRecTableNames2
|| to get the names of all recursive tables.

getRecTableNames :: relationalTable -> [string]

|| ____
|| getRecTableNames2: Gets table names recursively for a list of table entries.

getRecTableNames2 :: [tableEntry] -> [string]

|| ____
|| getRecTableNames3: Gets table names recursively for a list of columns.

getRecTableNames3 :: tableEntry -> [string]

|| ____
|| getRecTableNames4: Gets the name of a table column if it is an inner table.

getRecTableNames4 :: columnType -> [string]

|| File name: relationalFile3.m

%include "relationalFile0.m"
%include "relationalFile1.m"
%include "relationalFile2.m"

|| +++

|| selectColByPair: Applies a boolean function to a column identified

|| by column tag: return True if the application off function on
|| column is true, false otherwise. Also returns false if the function
|| is not of the right type.

selectColByPair :: strBoolPair -> columnType -> (bool, columnType)

|| ____
|| selectEntryByPair: Returns true if the entry list
|| contains a column whose tag is provided and the application
|| of the given function on the column is successful. Returns
|| false otherwise.

selectEntryByPair :: strBoolPair -> tableEntry -> tableEntry -> (bool, tableEntry)

|| ____
|| selectEntryByPairLst: Returns true if the application of all
|| boolean functions prodived on all columns provided are successful.
|| Returns false otherwise.

selectEntryByPairLst :: [strBoolPair] -> tableEntry -> (bool, tableEntry)

|| ____
|| selectEntryByPairLst2: Returns table entry if the application of all
|| boolean functions prodived on all columns provided are successful.
|| Returns an empty list otherwise.

selectEntryByPairLst2 :: [strBoolPair] -> tableEntry -> tableEntry

|| ____
|| selectEntryLstByPairLst: Returns all table entries if the application of all
|| boolean functions prodived on all columns provided are successful.
|| Returns an empty list otherwise.
|| Empty lists within empty lists are flattened.

selectEntryLstByPairLst :: [strBoolPair] -> [tableEntry] -> [tableEntry]

|| ____
|| extractColNames: Extracts column names from a list of string-boolean function pairs.

extractColNames :: [strBoolPair] -> [string]

|| ____
|| getInnerTable: Gets the inner table from a recursive column.

getInnerTable :: columnType -> relationalTable

|| ____
|| getTableColumns: Top level function to get a list of table columns recursively.
|| It calls getAllColumnNames.

getTableColumns :: relationalTable -> [string]

|| ____
|| getAllColumnNames: Collects all column names of all entries in a table.

getAllColumnNames :: [tableEntry] -> [string]

|| ____

|| getColumnNames: Collects all non recursive column names of a table entry.

getColumnNames :: tableEntry -> [string]

|| ____
|| addListMembers: Adds contents of a list to another list if not already there.

addListMembers :: [*] -> [*] -> [*]

|| ____
|| isSubSet: Checks whether a list is a subset of another list.

isSubSet :: [*] -> [*] -> bool

|| ____
|| isSelectListValid: Checks whether column names in a select list all
|| refer to valid column names in a table.

isSelectListValid :: [strBoolPair] -> [tableEntry] -> bool

|| ____
|| selectFrom2: Selects all entries in a table which satisfy a given
|| list of conditions after having all recursive column names resolved.

selectFrom2 :: [strBoolPair] -> relationalTable -> relationalTable

|| ____
|| selectNotIn2: Selects all entries in a table which do not satisfy a given
|| list of conditions after having all recursive column names resolved.

selectNotIn2 :: [strBoolPair] -> relationalTable -> relationalTable

|| File name: relationalFile4.m

%include "relationalFile0.m"
%include "relationalFile1.m"
%include "relationalFile2.m"

|| ++

|| replaceColumnTag: Replaces a column tag by a new string identifier for
|| a given column.

replaceColumnTag :: string -> string -> columnType -> columnType

|| ____
|| replaceEntryTag: Replaces a column tag by a new string identifier for
|| a table entry.

replaceEntryTag :: string -> string -> tableEntry -> tableEntry

|| ____
|| replaceEntryListTag: Replaces a column tag by a new string identifier for
|| all table entry list members.

replaceEntryListTag :: string -> string -> [tableEntry] -> [tableEntry]

```
|| _____________________________________________________________
|| sepColFromEntries: Given a list of table entries, it breaks each entry
|| into two entries, one containing a given column and the other containing
|| all other columns.

sepColFromEntries :: string -> [tableEntry] -> [strEntryEntryTuple]

|| _____________________________________________________________
|| sepColFromEntry: Using a column tag, separates a column from the rest of a
|| table entry and into a new list.

sepColFromEntry :: string -> tableEntry -> tableEntry -> strEntryEntryTuple

|| _____________________________________________________________
|| joinColTabLists: Attempts to join two list of table entries recursively.

joinColTabLists :: [strEntryEntryTuple] -> [strEntryEntryTuple] -> [tableEntry]

|| _____________________________________________________________
|| joinColTabLists2: Attempts to join a table entry recursively with a list
|| of table entries.

joinColTabLists2 :: strEntryEntryTuple -> [strEntryEntryTuple] -> [tableEntry]

|| _____________________________________________________________
|| joinColTabLists3: If the joining columns are of the same type and value, it
|| will join the entries.

joinColTabLists3 :: strEntryEntryTuple -> strEntryEntryTuple -> [tableEntry]

|| _____________________________________________________________
|| areColumnsEq: Cheks whether two columns are of equal type and value.

areColumnsEq :: columnType -> columnType -> bool

|| _____________________________________________________________
|| separateNonRec: Separates non recursive and recursive columns into
|| two different lists.

separateNonRec :: tableEntry -> tableEntry -> tableEntry -> (tableEntry, tableEntry)

|| _____________________________________________________________
|| removeRecTag: removes RC tag from the begining of a recursive column.

removeRecTag :: columnType -> relationalTable

|| _____________________________________________________________
|| makeRecTag: Attaches an RC tag to the begining of a recursive column.

makeRecTag :: relationalTable -> columnType

|| _____________________________________________________________
|| findLevel1: Finds at what level in a table a given column resides.

findLevel1 :: string -> relationalTable -> num -> joinTriple

|| _____________________________________________________________
```

|| findLevel2: Finds at what level in a list of table entries a given column resides.

findLevel2 :: string -> tableEntry -> num -> joinTriple

|| ____
|| findLevel3: Finds at what level in a table entry a given column resides.
|| Used for recursive columns only.

findLevel3 :: string -> tableEntry -> num -> joinTriple

|| ____
|| tableName: Gives name of a table.

tableName :: relationalTable -> string

|| ____
|| recColumnName: Gives name of a recursive column.

recColumnName :: columnType -> string

|| ____
|| getLevel: Gets the level at which a column exists.
|| Used for join operation.

getLevel :: joinTriple -> num

|| ____
|| isAtomic: Indicates whether a column is atomic or not.
|| Used for join operation.

isAtomic :: joinTriple -> bool

|| ____
|| getTabName: Gives column name if column is recursive.
|| returns an empty string for atomic columns.
|| Used for join operation.

getTabName :: joinTriple -> string

|| ____
|| cleanTable: Removes all entries in a table which have empty recursive
|| columns.

cleanTable :: relationalTable -> relationalTable

|| ____
|| cleanEntryList: Removes all entries in a list of table entries which have
|| empty recursive columns.

cleanEntryList :: [tableEntry] -> [tableEntry]

|| ____
|| cleanEntry: Removes an entry if it has an empty recursive column.
|| It first separates recursive and non recursive columns and then calls
|| cleanEntry2().

cleanEntry :: tableEntry -> [tableEntry]

|| ____
|| cleanEntry2: Removes an entry if it has an empty recursive column.
|| It calls cleanEntry3() to do the same recursively.

cleanEntry2 :: (tableEntry, tableEntry) -> [tableEntry]

|| ____
|| cleanEntry3: Checks a recursive column to see if it needs cleaning.

cleanEntry3 :: tableEntry -> tableEntry

|| ____
|| isAnEmptyRec: Checks a recursive column for emptiness.

isAnEmptyRec :: columnType -> bool

|| File name: relationalFile5.m
|| This file contains base functions for rename operator

%include "relationalFile0.m"

|| ++

|| renameColumn: Renames a column heading with a new one.

renameColumn :: strStrTuple -> columnType -> columnType

|| ____
|| renameEntry: Renames a column heading for a list of columns.

renameEntry :: strStrTuple -> tableEntry -> tableEntry

|| ____
|| renameEntryList: Renames a column heading for a list of table
|| entries.

renameEntryList :: strStrTuple -> [tableEntry] -> [tableEntry]

|| ____
|| renameTableList: Renames column heading of a list of columns for
|| a list of table entries.

renameTableList :: [strStrTuple] -> [tableEntry] -> [tableEntry]

|| ____
|| rename2: Renames column heading of a list of columns for a
|| relational table.

rename2 :: [strStrTuple] -> relationalTable -> relationalTable

```
|| File name: relationalFile6.m
|| Join case 1.

%include "relationalFile0.m"
%include "relationalFile4.m"

|| ++++++++++++++++++++++++++++++++++++++++++++++++++++

|| joinTables1: Main function to join two normal relational tables based on two
|| columns of similar types.
```

joinTables1 :: stringTableTuple -> stringTableTuple -> relationalTable

```
|| File name: relationalFile7.m
|| Join case 2.

%include "relationalFile0.m"
%include "relationalFile4.m"
%include "relationalFile6.m"

|| ++++++++++++++++++++++++++++++++++++++++++++++++++++

|| joinTables2: Main function to join a top level and a nested relational table
|| based on two columns of similar types.
```

joinTables2 :: stringTableTuple -> stringTableTuple -> num -> relationalTable

```
|| ________________________________________________________
|| joinTables20:
```

joinTables20 :: stringTableTuple -> stringTableTuple -> num -> relationalTable

```
|| ________________________________________________________
|| joinTables21:
```

joinTables21 :: stringTableTuple -> stringEntryListTuple -> num -> [tableEntry]

```
|| ________________________________________________________
|| joinTables22:
```

joinTables22 :: stringTableTuple -> stringEntryTuple -> num -> [tableEntry]

```
|| ________________________________________________________
|| joinTables23:
```

joinTables23 :: stringTableTuple -> string -> (tableEntry, tableEntry) -> num -> [tableEntry]

```
|| ________________________________________________________
|| joinTables24:
```

joinTables24 :: stringTableTuple -> string -> tableEntry -> num -> tableEntry

```
|| ________________________________________________________
|| joinTables25:
```

joinTables25 :: tableEntry -> tableEntry -> [tableEntry]

```
|| File name: relationalFile8.m
|| Join case 3a and 3b.

%include "relationalFile0.m"
%include "relationalFile4.m"
%include "relationalFile6.m"

|| +++++++++++++++++++++++++++++++++++++++++++++++++++++

|| Case 3a:

|| joinTables3a: Top level method to join tables according to case 3a.
|| Joins 2 nested columns at the same level.

joinTables3a :: stringTableTuple -> stringTableTuple -> num -> relationalTable

|| ___________________________________________________________
|| joinTables3a2: Joins two tables according to case 3a.
|| It acts on tables directly and carries the nesting level.

joinTables3a2 :: stringTableTuple -> stringTableTuple -> num -> relationalTable

|| ___________________________________________________________
|| joinTables3a3: Joins two tables according to case 3a.
|| It acts on entry lists and carries nesting level.

joinTables3a3 :: stringEntryListTuple -> stringEntryListTuple -> num -> [tableEntry]

|| ___________________________________________________________
|| joinTables3a4: Joins a table entry with a list of entries recursively.

joinTables3a4 :: stringEntryTuple -> stringEntryListTuple -> num -> [tableEntry]

|| ___________________________________________________________
|| joinTables3a5: Separates recursive and non recursive columns in table entries for joining.

joinTables3a5 :: stringEntryTuple -> stringEntryTuple -> num -> [tableEntry]

|| ___________________________________________________________
|| joinTables3a6: Joins two recursive columns.

joinTables3a6 :: stringEntryTuple -> stringEntryTuple -> num -> tableEntry

|| ___________________________________________________________
|| joinTables3a7: Joins two table entries if the recurive column is not empty.

joinTables3a7 :: tableEntry -> tableEntry -> tableEntry -> [tableEntry]

|| ___________________________________________________________
|| Case 3b:

|| joinTables3b: Top level method to join tables according to case 3b.
|| Joins 2 nestted columns at different levels.

joinTables3b :: stringTableTuple -> stringTableTuple -> num -> num -> relationalTable

|| ___________________________________________________________
```

|| joinTables3b2: Method to join tables according to case 3b.
|| It acts on tables directly and carries the nesting level.

joinTables3b2 :: stringTableTuple -> stringTableTuple -> num -> num -> relationalTable

|| ____
|| joinTables3b3: Method to join tables according to case 3b.
|| It acts on entry lists and carries nesting level.

joinTables3b3 :: stringEntryListTuple -> stringTableTuple -> num -> num -> [tableEntry]

|| ____
|| joinTables3b4: Joins a table entry with a list of entries recursively.

joinTables3b4 :: stringEntryTuple -> stringTableTuple -> num -> num -> [tableEntry]

|| ____
|| joinTables3b5: Separates recursive and non recursive columns in table entries for joining.

joinTables3b5 :: stringEntryTuple -> stringTableTuple -> num -> num -> tableEntry

|| File name: relationalFile9.m
|| Join case 4.

%include "relationalFile0.m"
%include "relationalFile1.m"
%include "relationalFile4.m"
%include "relationalFile6.m"
%include "relationalFile8.m"

|| ++

|| joinTables4: Top level function to join two tables according
|| to case 4.

joinTables4 :: stringTableTuple -> stringTableTuple -> relationalTable

|| ____
|| joinTables41: Joins two table entry lists based on case 4.
|| It recursively applies each element of the first list to the
|| entire second list.

joinTables41 :: stringEntryListTuple -> stringEntryListTuple -> [tableEntry]

|| ____
|| joinTables42: Joins an entry to an entry list based on case 4.
|| It recursively applied the entry to each element (entry) of the second list.

joinTables42 :: stringEntryTuple -> stringEntryListTuple -> [tableEntry]

|| ____
|| joinTables43: Joins two entries based on case 4.

joinTables43 :: stringEntryTuple -> stringEntryTuple -> [tableEntry]

|| ____

|| joinTables44: Joins two recursive columns.

joinTables44 :: stringEntryTuple -> stringEntryTuple -> tableEntry

|| ____
|| joinTables45: Joins two tables (within recursive columns) based on case 4.

joinTables45 :: stringTableTuple -> stringTableTuple -> relationalTable

|| ____
|| joinTables46: Joins two entry lists (within a recursive column)
|| based on case 4.

joinTables46 :: [tableEntry] -> [tableEntry] -> [tableEntry]

|| ____
|| joinTables47: Auxiliary function to join two entry lists according
|| to case 4.

joinTables47 :: [tableEntry] -> [tableEntry] -> [tableEntry] -> [tableEntry]

|| ____
|| joinTables48: Checks whether an entry belongs to an entry list.
|| If not, it ignores the entry, otherwise, it will record the entry for
|| future use.

joinTables48 :: tableEntry -> [tableEntry] -> [tableEntry] -> [tableEntry]

|| File name: relationalFile10.m
|| Join case 5.

%include "relationalFile0.m"
%include "relationalFile4.m"
%include "relationalFile6.m"
%include "relationalFile9.m"

|| ++

|| joinTables5: Main function to join a top level and a nested relational table
|| based on two columns of similar types.

joinTables5 :: stringTableTuple -> stringTableTuple -> num -> relationalTable

|| ____
|| joinTables50:

joinTables50 :: stringTableTuple -> stringTableTuple -> num -> relationalTable

|| ____
|| joinTables51:

joinTables51 :: stringTableTuple -> stringEntryListTuple -> num -> [tableEntry]

|| ____
|| joinTables52:

joinTables52 :: stringTableTuple -> stringEntryTuple -> num -> [tableEntry]

|| ____
|| joinTables53:

joinTables53 :: stringTableTuple -> string -> (tableEntry, tableEntry) -> num -> [tableEntry]

|| ____
|| joinTables54:

joinTables54 :: stringTableTuple -> string -> tableEntry -> num -> tableEntry

|| ____
|| joinTables55:

joinTables55 :: tableEntry -> tableEntry -> [tableEntry]

|| File name: relationalFile11.m
|| Join case 6a and 6b.

%include "relationalFile0.m"
%include "relationalFile4.m"
%include "relationalFile6.m"
%include "relationalFile8.m"
%include "relationalFile9.m"

|| +++

|| Case 6a:

|| joinTables6a: Top level method to join tables according to case 6a.
|| Joins 2 nested columns at the same level.

joinTables6a :: stringTableTuple -> stringTableTuple -> num -> relationalTable

|| ____
|| joinTables6a2: Joins two tables according to case 6a.
|| It acts on tables directly and carries the nesting level.

joinTables6a2 :: stringTableTuple -> stringTableTuple -> num -> relationalTable

|| ____
|| joinTables6a3: Joins two tables according to case 6a.
|| It acts on entry lists and carries nesting level.

joinTables6a3 :: stringEntryListTuple -> stringEntryListTuple -> num -> [tableEntry]

|| ____
|| joinTables6a4: Joins a table entry with a list of entries recursively.

joinTables6a4 :: stringEntryTuple -> stringEntryListTuple -> num -> [tableEntry]

|| joinTables6a5: Separates recursive and non recursive columns in table entries for joining.

joinTables6a5 :: stringEntryTuple -> stringEntryTuple -> num -> [tableEntry]

|| ____
|| joinTables6a6: Joins two recursive columns.

joinTables6a6 :: stringEntryTuple -> stringEntryTuple -> num -> tableEntry

|| ____
|| Case 6b:

|| joinTables6b: Top level method to join tables according to case 6b.
|| Joins 2 nested columns at different levels.

joinTables6b :: stringTableTuple -> stringTableTuple -> num -> num -> relationalTable

|| ____
|| joinTables6b2: Method to join tables according to case 6b.
|| It acts on tables directly and carries the nesting level.

joinTables6b2 :: stringTableTuple -> stringTableTuple -> num -> num -> relationalTable

|| ____
|| joinTables6b3: Method to join tables according to case 6b.
|| It acts on entry lists and carries nesting level.

joinTables6b3 :: stringEntryListTuple -> stringTableTuple -> num -> num -> [tableEntry]

|| ____
|| joinTables6b4: Joins a table entry with a list of entries recursively.

joinTables6b4 :: stringEntryTuple -> stringTableTuple -> num -> num -> [tableEntry]

|| ____
|| joinTables6b5: Separates recursive and non recursive columns in table entries for joining.

joinTables6b5 :: stringEntryTuple -> stringTableTuple -> num -> num -> tableEntry

|| File name: relationalFile12.m
|| Responsible for finding the most suitable method for join.

%include "relationalFile0.m"
%include "relationalFile4.m"
%include "relationalFile6.m"
%include "relationalFile7.m"
%include "relationalFile8.m"
%include "relationalFile9.m"
%include "relationalFile10.m"
%include "relationalFile11.m"

|| ++

|| joinTables0:

joinTables0 :: stringTableTuple -> stringTableTuple -> relationalTable

```
|| File name: main.m

|| Contains all top level calls.


%include "relationalFile0.m"
%include "relationalFile1.m"
%include "relationalFile2.m"
%include "relationalFile3.m"
%include "relationalFile4.m"
%include "relationalFile5.m"
%include "relationalFile6.m"
%include "relationalFile7.m"
%include "relationalFile8.m"
%include "relationalFile9.m"
%include "relationalFile10.m"
%include "relationalFile11.m"
%include "relationalFile12.m"
%include "samples1.m"
%include "ttraining.m"
%include "tcourse.m"
%include "tcashpoint.m"
%include "tlocation.m"
%include "tdept.m"

|| +++++++++++++++++++++++++++++++++++++++++++++++++++++

|| tableProjection: Selects a subset of table entries based on given
|| column names by first resolving recursive column names and then calling
|| helper functions tableProjection2 and flattenRelTable.

tableProjection :: [string] -> relationalTable -> relationalTable
tableProjection x y = getAllUnion (flattenRelTable (tableProjection2 x (resolvePath y)))

|| ____________________________________________________________
|| tableProduct: Product of two relational tables.
|| All possible combinations of table entries are included.
|| Recursive cases included.

tableProduct :: (string, relationalTable) -> (string, relationalTable) -> relationalTable
tableProduct (a, b) (c, d)    = getAllUnion (tableProduct2 (resolvePath b) (resolvePath d)),        if (u & v)
                              = error "Error: Table product on two inner tables",                  if ((~u) & (~v))
                             = getAllUnion (tableProduct3 (resolvePath b) (c, resolvePath d)),     if (u & (~v))
                            = getAllUnion (tableProduct3 (resolvePath d) (a, resolvePath b)),      otherwise
                                      where   u = (a = "")
                                              v = (c = "")

|| ____________________________________________________________
|| selectFrom: Selects all entries in a table which satisfy a given
|| list of conditions by:
|| 1) resolving all recursive column names,
|| 2) calling tableProjection2 on the result,
|| 3) calling flattenRelTable on the result.

selectFrom :: [string] -> [strBoolPair] -> relationalTable -> relationalTable
selectFrom x y z = getAllUnion (flattenRelTable (tableProjection2 x (selectFrom2 y (resolvePath
z))))
```

```
|| ______________________________________________________________
|| selectNotIn: Selects all entries in a table which do not satisfy a given
|| list of conditions by:
|| 1) resolving all recursive column names,
|| 2) calling tableProjection2 on the result,
|| 3) calling flattenRelTable on the result.
```

selectNotIn :: [string] -> [strBoolPair] -> relationalTable -> relationalTable
selectNotIn x y z = getAllUnion (flattenRelTable (tableProjection2 x (selectNotIn2 y (resolvePath z))))

```
|| ______________________________________________________________
|| joinTables: Main function to join two relational tables of any complexity
|| based on two columns of similar types.
```

joinTables :: stringTableTuple -> stringTableTuple -> relationalTable
joinTables (x, r1) (y, r2) = getAllUnion (retainColNames (joinTables0 (x, resolvePath r1) (y, resolvePath r2)))

```
|| ______________________________________________________________
|| rename: Renames column heading of a list of columns for a
|| relational table. It calls rename2 function after resolving
|| all column names.
```

rename :: [strStrTuple] -> relationalTable -> relationalTable
rename x y = getAllUnion (rename2 x (resolvePath y))

B.4 Illustration Examples

The coding in Miranda, of examples presented in the thesis, is given below. Occasionally, the result is also given, assuming that it does not occupy much space.

Example 4.8

```
tableProjection["COMPANY","TRAINING_2(TRAINER(C(CN)))"]t2
```

Result:

```
Relation "TRAINING_2"
   [[SC "COMPANY" "Apple", RC (Relation "TRAINER"
            [[RC (Relation "C"
                   [[NC "TRAINING_2(TRAINER(C(CN)))" 1],
                   [NC "TRAINING_2(TRAINER(C(CN)))" 2]])],
             [RC (Relation "C"
                   [[NC "TRAINING_2(TRAINER(C(CN)))" 1],
                   [NC "TRAINING_2(TRAINER(C(CN)))" 3],
                   [NC "TRAINING_2(TRAINER(C(CN)))" 2]])]])],
    [SC "COMPANY" "IBM", RC (Relation "TRAINER"
                [[RC (Relation "C"
                        [[NC "TRAINING_2(TRAINER(C(CN)))" 3],
                        [NC "TRAINING_2(TRAINER(C(CN)))" 5],
                        [NC "TRAINING_2(TRAINER(C(CN)))" 4]])]])],
      [SC "COMPANY" "Microsoft", RC (Relation "TRAINER"
                [[RC (Relation "C"
                        [[NC "TRAINING_2(TRAINER(C(CN)))" 2]])]])]]
```

Example 4.9:

A revised version of the example is given, due to the fact that relevant implementation is missing (only for 'Mark', since OR has not been implemented).

```
selectFrom["COMPANY","TRAINING_2(TRAINER(TRN))","TRAINING_2(TRAINER(C(CN)))",
"TRAINING_2(TRAINER(C(Y)))"][("TRAINING_2(TRAINER(TRN))", SF ((=) "Mark")),
("TRAINING_2(TRAINER(C(Y)))", NF ((=) 82))]t2
```

Result:

```
Relation "TRAINING_2"
  [[SC "COMPANY" "Apple", RC (Relation "TRAINER"
    [[SC "TRAINING_2(TRAINER(TRN))" "Mark", RC (Relation "C"
      [[NC "TRAINING_2(TRAINER(C(CN)))" 1, NC "TRAINING_2(TRAINER(C(Y)))" 82],
      [NC "TRAINING_2(TRAINER(C(CN)))" 3, NC "TRAINING_2(TRAINER(C(Y)))" 82]])]])]]
```

Example 4.9:

A revised version of the example is given, due to the fact that relevant implementation is missing (only for 'Tim', since OR has not been implemented).

```
selectFrom["COMPANY","TRAINING_2(TRAINER(TRN))","TRAINING_2(TRAINER(C(CN)))",
"TRAINING_2(TRAINER(C(Y)))"][("TRAINING_2(TRAINER(TRN))", SF ((=) "Tim")),
("TRAINING_2(TRAINER(C(Y)))", NF ((=) 82))]t2
```

Result:

```
Relation "TRAINING_2"
  [[SC "COMPANY" "IBM", RC (Relation "TRAINER"
     [[SC "TRAINING_2(TRAINER(TRN))" "Tim", RC (Relation "C"
      [[NC "TRAINING_2(TRAINER(C(CN)))" 3, NC"TRAINING_2(TRAINER(C(Y)))" 82],
      [NC "TRAINING_2(TRAINER(C(CN)))" 4, NC "TRAINING_2(TRAINER(C(Y)))" 82]])]])]]
```

Example 4.12:

```
rename[("DEPT(UNIT(UD))", "DEPT(UNIT(UD')"),
("DEPT(UNIT(COURSE_DETAILS(C(CN))))",
"DEPT(UNIT(COURSE_DETAILS(C'(CN))))"),("DEPT(UNIT(COURSE_DETAILS(C(Y))))",
"DEPT(UNIT(COURSE_DETAILS(C'(Y))))") ]d
```

Example 4.14:

```
tableProduct("COURSE",t)("", cashpoint)
```

Other Example:

```
tableProduct("",cashpoint)("",employment)
```

Example 4.16:

```
joinTables("LOCATION(ANNEX(ADDRESS))", location)
            ("CASH-POINT(BRANCH(ADDRESS))", cashpoint)
```

<u>Result</u>:

```
Relation "LOCATION/CASH-POINT"
    [[SC "COMPANY" "Toshiba", SC "BANK" "Natwest", RC (Relation "ANNEX/BRANCH"
      [[SC "ADDRESS" "Porchester Rd.", SC "BUILDING" "North Building", SC "SORT_CODE"
                                                                        "560038"]])],
    [SC "COMPANY" "Microsoft", SC "BANK" "Barclays", RC (Relation "ANNEX/BRANCH"
      [[SC "ADDRESS" "Ashford St.", SC "BUILDING" "Pegasus House", SC "SORT_CODE"
                                                                        "386600"]])],
    [SC "COMPANY" "Microsoft", SC "BANK" "Natwest", RC (Relation "ANNEX/BRANCH"
      [[SC "ADDRESS" "Park Rd.", SC "BUILDING" "Queen's Building", SC "SORT_CODE"
                                                                        "560045"]])],
    [SC "COMPANY" "Microsoft", SC "BANK" "Lloyd's", RC (Relation "ANNEX/BRANCH"
      [[SC "ADDRESS" "Ashford St.", SC "BUILDING" "Pegasus House", SC "SORT_CODE"
                                                                        "478202"],
      [SC "ADDRESS" "Park Rd.", SC "BUILDING" "Queen's Building", SC "SORT_CODE"
                                                                        "478210"]])]]
```

Example 4.17:

```
joinTables("TRAINING_1(PROGRAMME(TRN))", t1)("DEPT_1(UNIT(TRAINER(TRN)))",d1)
```

Example 4.18:

```
joinTables("JOB", employment)("JOB", payment)
```

Result:

```
Relation "EMPLOYMENT/PAYMENT"
        [[SC "NAME" "Anna", SC "SALARY" "15,500-19,500", RC(Relation "JOB"
                [[SC "COMPANY" "Toshiba", SC "JOB_DESCRIPTION" "Secretary"]])],
        [SC "NAME" "Anna", SC "SALARY" "18,000-23,000", RC (Relation "JOB"
                [[SC "COMPANY" "Microsoft", SC "JOB_DESCRIPTION" "Secretary"]])],
        [SC "NAME" "Paul", SC "SALARY" "18,000-23,000", RC (Relation "JOB"
                [[SC "COMPANY" "Microsoft", SC "JOB_DESCRIPTION" "Programmer"]])],
        [SC "NAME" "Mark", SC "SALARY" "25,000-30,000", RC (Relation "JOB"
                [[SC "COMPANY" "Apple", SC "JOB_DESCRIPTION" "Director"]])]]
```

Example 4.19:

```
joinTables("C",d2)("C",t2)
```

Result:

```
Relation "DEPT_2/TRAINING_2"
    [[SC "DN" "Research", NC "D" 1, SC "COMPANY" "Apple", RC (Relation "UNIT/TRAINER"
        [[SC "UD" "Software Engineering", NC "UN" 511, SC "TRN" "Jack", RC (Relation "C"
                        [[NC "CN" 1, NC "Y" 75], [NC "CN" 2, NC "Y" 76]])],
        [SC "UD" "Basic Research", NC "UN" 552, SC "TRN" "Mark", RC (Relation "C"
                        [[NC "CN" 1, NC "Y" 82], [NC "CN" 2, NC "Y" 79]])],
        [SC "UD" "Planning", NC "UN" 678, SC "TRN" "Jack", RC (Relation "C"
                        [[NC "CN" 2, NC "Y" 76]])]])],
    [SC "DN" "Research", NC "D" 1, SC "COMPANY" "IBM", RC (Relation "UNIT/TRAINER"
        [[SC "UD" "Software Engineering", NC "UN" 511, SC "TRN" "Tim", RC (Relation "C"
                        [[NC "CN" 5, NC "Y" 79]])],
        [SC "UD" "Planning", NC "UN" 678, SC "TRN" "Tim", RC (Relation "C"
                        [[NC "CN" 4, NC "Y" 82]])]])],
    [SC "DN" "Development", NC "D" 2, SC "COMPANY" "Apple", RC (Relation "UNIT/TRAINER"
        [[SC "UD" "Design", NC "UN" 650, SC "TRN" "Jack", RC (Relation "C"
                        [[NC "CN" 1, NC "Y" 75]])],
        [SC "UD" "Maintenance", NC "UN" 780, SC "TRN" "Mark", RC (Relation "C"
                        [[NC "CN" 3, NC "Y" 82]])],
        [SC "UD" "Planning", NC "UN" 981, SC "TRN" "Mark", RC (Relation "C"
                        [[NC "CN" 3, NC "Y" 82]])]])],
    [SC "DN" "Development", NC "D" 2, SC "COMPANY" "IBM", RC (Relation "UNIT/TRAINER"
        [[SC "UD" "Maintenance", NC "UN" 780, SC "TRN" "Tim", RC (Relation "C"
                        [[NC "CN" 3, NC "Y" 82]])],
        [SC "UD" "Planning", NC "UN" 981, SC "TRN" "Tim", RC (Relation "C"
                        [[NC "CN" 3, NC "Y" 82]])]])],
[SC "DN" "Development", NC "D" 2, SC "COMPANY" "Microsoft", RC (Relation "UNIT/TRAINER"
        [[SC "UD" "Design", NC "UN" 650, SC "TRN" "Karen", RC (Relation "C"
                        [[NC "CN" 2, NC "Y" 77]])],
        [SC "UD" "Planning", NC "UN" 981, SC "TRN" "Karen", RC (Relation "C"
                        [[NC "CN" 2, NC "Y" 81]])]])]]
```

Example 4.20:

```
joinTables("C",d2)("C",t)
```

Example 5.5:

```
tableProjection["COMPANY", "T_TRAINING(TRAINER(COURSE(CN)))",
"T_TRAINING(TRAINER(COURSE(CN_PER)))"]tt
```

Result:

```
Relation "T_TRAINING"
  [[SC "COMPANY" "Apple", RC (Relation "TRAINER"
      [[RC (Relation "COURSE"
          [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 5.2,
                          TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                      [((2,11,1994),(25,4,1995)),((7,8,1996),(1,1,2010))]]])],
                                  [RC (Relation "COURSE"
                              [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 3.3,
                                   TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                              [((2,1,1992),(8,11,1996))]],
                              [NC "T_TRAINING(TRAINER(COURSE(CN)))" 3.5,
                                   TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                             [((30,4,1995),(1,1,2010))]]])]])],
                           [SC "COMPANY" "IBM", RC (Relation "TRAINER"
                                  [[RC (Relation "COURSE"
                              [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 5.2,
                                   TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                             [((19,3,1997),(21,4,1997))]],
                              [NC "T_TRAINING(TRAINER(COURSE(CN)))" 5.0,
                                   TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                            [((17,12,1995),(1,1,2010))]]])]])],
                         [SC "COMPANY" "Microsoft", RC (Relation "TRAINER"
                                  [[RC (Relation "COURSE"
                              [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 3.3,
                                        TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                                     [((25,6,1996),(1,1,2010))]]])]])]]
```

Example 5.7:

A revised version of the example is given, due to the fact that relevant implementation is missing (only for 'Tim', since OR has not been implemented).

```
selectFrom["COMPANY","T_TRAINING(TRAINER(TRN))","T_TRAINING(TRAIN
ER(COURSE(CN)))","T_TRAINING(TRAINER(COURSE(CN_PER)))"][("T_TRAINI
NG(TRAINER(TRN))",SF((=)"Tim")),("T_TRAINING(TRAINER(COURSE(CN_PER
)))",TF((tECovers ((1,1,1997),(1,1,1998)))))]tt
```

Result:

```
Relation "T_TRAINING"
   [[SC "COMPANY" "IBM", RC (Relation "TRAINER"
     [[SC "T_TRAINING(TRAINER(TRN))" "Tim", RC (Relation "COURSE"
          [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 5.2,
            TC "T_TRAINING(TRAINER(COURSE(CN_PER)))" [((19,3,1997),(21,4,1997))]]])]])]]
```

Other Examples (Temporal Cartesian Product):

```
tableProduct("COURSE",tt)("", tcashpoint)
tableProduct("",tcashpoint)("",tcourse)
```

Query 1:

```
selectFrom ["D", "DEPT(UNIT(UD))", "DEPT(UNIT(COURSE_DETAILS(TRN)))"][("
D", NF ((=) 1))]d
```

Result:

```
Relation "DEPT"
   [[NC "D" 1, RC (Relation "UNIT"
         [[SC "DEPT(UNIT(UD))" "Software Engineering", RC (Relation
       "COURSE_DETAILS"
                    [[SC "DEPT(UNIT(COURSE_DETAILS(TRN)))" "Mark"]])],
         [SC "DEPT(UNIT(UD))" "Basic Research", RC (Relation "COURSE_DETAILS"
                    [[SC "DEPT(UNIT(COURSE_DETAILS(TRN)))" "Karen"],
                    [SC "DEPT(UNIT(COURSE_DETAILS(TRN)))" "Tim"]])],
         [SC "DEPT(UNIT(UD))" "Planning", RC (Relation "COURSE_DETAILS"
                    [[SC "DEPT(UNIT(COURSE_DETAILS(TRN)))" "Mark"]])]])]]
```

Query 5:

```
selectFrom["COURSE/TRAINING(COURSE/TRAINER(TRN))"][("COURSE/TRAINING(CO
URSE/TRAINER(COURSE/COURSE(TITLE)))", SF ((=) "Computer Skills"))]
(joinTables("C", t)("C", course))
```

Result:

```
Relation "COURSE/TRAINING"
          [[RC (Relation "COURSE/TRAINER"
                    [[SC "COURSE/TRAINING(COURSE/TRAINER(TRN))" "Jack"]])],
          [RC (Relation "COURSE/TRAINER"
                    [[SC "COURSE/TRAINING(COURSE/TRAINER(TRN))" "Karen"]])]]
```

Query 6:

A revised version of the query is given, due to the fact that relevant implementation is missing (only for 'Karen', since OR has not been implemented).

```
tableProjection["COMPANY","BANK"](joinTables("TRAINING/LOCATION(ANNEX(AD
 DRESS))",joinTables("COMPANY",selectFrom["COMPANY"][("TRAINING(TRAINER(TRN))"
,SF((=)"Karen"))]t)("COMPANY",location))("CASH-POINT(BRANCH(ADDRESS))", cashpoint))
```

Result:

```
Relation "TRAINING/LOCATION/CASH-POINT"
                    [[SC "COMPANY" "Microsoft", SC "BANK" "Barclays"],
                    [SC "COMPANY" "Microsoft", SC "BANK" "Natwest"],
                    [SC "COMPANY" "Microsoft", SC "BANK" "Lloyd's"]]
```

Query 10:

```
tableProjection["T_DEPT(STAFF(COURSE_DETAILS(SNAME)))”,"T_DEPT(STAFF(COURS
E_DETAILS(COURSE(CN_PER))))"]td
```

Result:

```
Relation "T_DEPT"
          [[RC (Relation "STAFF"
               [[RC (Relation "COURSE_DETAILS"
                    [[SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Paul",
                              RC (Relation "COURSE"
                                      [[TC "T_DEPT(STAFF(COURSE_DETAILS(COURSE(CN_PER))))"
                                              [((27,8,1995),(30,1,1996)),((1,2,1995),(24,6,1995))]]])],
                    [SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Peter",
                              RC (Relation"COURSE"
                                      [[TC "T_DEPT(STAFF(COURSE_DETAILS(COURSE(CN_PER))))"
```

```
                                        [((1,1,1998),(28,10,1998))]]])]])],
          [RC (Relation "COURSE_DETAILS"
             [[SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Anna",
                   RC (Relation "COURSE"
                        [[TC"T_DEPT(STAFF(COURSE_DETAILS(COURS E(CN_PER))))"
                              [((29,9,1997),(10,2,1998)),((1,7,1995),(1,8,1995))]]])],
             [SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Mary",
                   RC (Relation "COURSE"
                        [[TC "T_DEPT(STAFF(COURSE_DETAILS(COURSE(CN_PER))))"
                              [((17,1,1997),(28,4,1997))]]])]])],
          [RC (Relation "COURSE_DETAILS"
             [[SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Katy",
                   RC (Relation "COURSE"
                        [[TC "T_DEPT(STAFF(COURSE_DETAILS(COURSE(CN_PER))))"
                              [((13,2,1994),(4,3,1995)),((22,4,1995),(15,5,1995))]]])]])]])],
       [RC (Relation "STAFF"
          [[RC (Relation "COURSE_DETAILS"
             [[SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Steve",
                   RC (Relation "COURSE"
                        [[TC "T_DEPT(STAFF(COURSE_DETAILS(COURSE(CN_PER))))"
                              [((18,3,1996),(1,7,1996))]]])]])],
          [RC (Relation "COURSE_DETAILS"
             [[SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Helen",
                   RC (Relation "COURSE"
                        [[TC"T_DEPT(STAFF(COURSE_DETAILS(COURSE(CN_PER))))"
                              [((17,8,1997),(1,1,2010))]]])],
             [SC "T_DEPT(STAFF(COURSE_DETAILS(SNAME)))" "Pat",
                   RC (Relation "COURSE"
                        [[TC "T_DEPT(STAFF(COURSE_DETAILS(COURSE(CN_PER))))"
                              [((18,9,1995),(10,10,1995))]]])]])]])]]
```

Query 11:

A revised version of the query is given, due to the fact that relevant implementation is missing (START has not been implemented).

```
tableProjection["T_TRAINING(TRAINER(TRN))","T_TRAINING(TRAINER(COURSE(CN_PE
R)))"]tt
```

Result:

```
Relation "T_TRAINING"
    [[RC (Relation "TRAINER"
                [[SC "T_TRAINING(TRAINER(TRN))" "Jack", RC (Relation "COURSE"
                              [[TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                 [((2,11,1994),(25,4,1995)),((7,8,1996),(1,1,2010))]]]])],
                [SC "T_TRAINING(TRAINER(TRN))" "Mark", RC (Relation "COURSE"
                              [[TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                 [((2,1,1992),(1,1,2010))]]])]])],
    [RC (Relation "TRAINER"
                [[SC "T_TRAINING(TRAINER(TRN))" "Tim", RC (Relation "COURSE"
                              [[TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                 [((19,3,1997),(21,4,1997))]]])]])],
    [RC (Relation "TRAINER"
               [[SC "T_TRAINING(TRAINER(TRN))" "Karen", RC (Relation "COURSE"
                              [[TC "T_TRAINING(TRAINER(COURSE(CN_PER)))"
                                 [((25,6,1996),(1,1,2010)) ]]])]])]]
```

Query 12:

A revised version of the query is given, since relevant implementation is missing (COUNT has not been implemented).

```
selectFrom["T_TRAINING(TRAINER(COURSE(CN)))"]
             [("T_TRAINING(TRAINER(COURSE(CN)))",
TF(tEOverlaps((1,1,1998),(1,1,1999))))]tt
```

Result:

```
Relation "T_TRAINING"
        [[RC (Relation "TRAINER"
                [[RC (Relation "COURSE"
                        [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 5.2]])],
                 [RC (Relation "COURSE"
                        [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 3.3],
                        [NC "T_TRAINING(TRAINER(COURSE(CN)))" 3.5]])]])],
        [RC (Relation "TRAINER"
          [[RC (Relation "COURSE"
                    [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 5.2],
                    [NC "T_TRAINING(TRAINER(COURSE(CN)))" 5.0]])]])],
[RC (Relation "TRAINER"
          [[RC (Relation "COURSE"
                    [[NC "T_TRAINING(TRAINER(COURSE(CN)))" 3.3]])]])]]
```