Mesodata: Engineering Domains for Attribute Evolution and Data Integration

by

Denise Bernadette Angela de Vries, *B.Comp.& Inf.Sc., B.Sc.(Hons)*
School of Informatics and Engineering,
Faculty of Science and Engineering

21 October 2005

A thesis presented to the
Flinders University of South Australia
in total fulfillment of the requirements for the degree of
Doctor of Philosophy

Adelaide, South Australia, 2006

Contents

List of Figures

List of Tables

Abstract

The introduction of databases for data storage and handling revolutionised the way we dealt with records and enabled simple and fast information processing, aggregation and summarisation. Database and information technology systems have evolved from simple file processing systems to powerful database systems. Data management technology has progressed from hierarchical and network systems to relational databases, data modelling tools and indexing and organisational techniques. The development of Relational Database Management Systems and automated systems put the layout and *form* into the unchanging metadata and gave us *record once* systems.

Unfortunately, the 'real world' upon which databases are modelled constantly changes. These changes may affect the schema for a variety of reasons including;

- Unanticipated requirements,
- A change in the universe of discourse,
- A change to the interpretation of facts about the universe of discourse,
- Changes in the form of updates to effect upgrades to the functionality or scope of a system,
- Changes in the form of updates to effect efficiency improvements,
- Changes caused by system operation,
- Error correction.

Different formalisms have been developed to deal with schema changes with the aim being to preserve information capacity and preserve semantic correctness. Schematic changes may be the result of evolving one system or may arise due to the need for merging two or more systems. Schematic conflicts occur which must be resolved and the schemata unified to produce a new version. To reach this goal there are graph based *schema integration* architectures, as well as, semi-automatic systems applying *schema matching* and *schema translation* techniques. These systems also utilise ontologies, thesauri, and so forth to integrate data from heterogeneous sources in order to process queries and views.

Data integration or conversion remains a partially resolved issue. Some metadata changes are managed by changes to application code and system down time for conversion procedures. However an attribute change may result in data loss, changed accuracy, and altered semantics. Whilst the use of ontologies, concept graphs and other knowledge interchange techniques are alleviating the problems of data integration, these structures are not yet an integral part of the database architecture.

This thesis argues a three-level architecture for relational databases with an interface positioned between data and metadata for complex domains. This intermediary level is the *mesodata* layer. This mesodata layer, separate from the metadata and data, provides complex structures, such as graphs, queues, and circular lists, in which to store domain values and their inter-relationships as well as supplying the 'intelligence' required to operate and manipulate them. The domain structures enable different orderings that form the bases of filters for enhanced querying and information retrieval. DBMS supplied mesodata types would allow for the re-usable inclusion of domain information such as in ontologies, taxonomies and concept graphs that to date have been only application specific.

Certification

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

As requested under Clause 14 of Appendix E of the *Flinders University Research Higher Degree Policies and Procedures Manual* I hereby agree to waive the conditions referred to in Clause 13(b) and (c), and thus

- Flinders University may lend this thesis to other institutions or individuals for the purpose of scholarly research;
- Flinders University may reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signed Dated

Denise Bernadette Angela de Vries

Acknowledgements

I would like to thank my supervisor Professor John Roddick for his advice, support and enthusiasm throughout my candidature.

There are many people in the School of Informatics and Engineering at Flinders University who have helped me in large ways and small, I believe I owe each person thanks. In particular, the members of the Knowledge Discovery and Intelligent Systems Group for constructive criticism, rigorous discussions and friendship, – (in room number order) Darin Chan, Dongqiang Yang, Trent Lewis, Martin Luerssen, Richard Leibbrandt, Darius Pfitzner, David Powers, Aaron Ceglar, Carl Mooney, Sally Rice, Anna Shillabeer, Edi Winarko, Ron Porter, Paul Calder, Amos Omondi, Tiffany Winn, and Lorraine Harker – and Murk Bottema and Jalina Widjaja for their assistance and comments.

I appreciate too the Flinders Postgraduate Students' Association for providing support, advice, resources and the research training courses and workshops that were so helpful at the beginning of my candidature. Thank you Leonie Randall and Audrey Nicholson.

I am very grateful for the all the assistance I received from Versatile Solutions Pty. Ltd, especially to Mr Arthur Verster for his time and effort.

However, none of this work could have been achieved without the unstinting support of Bart de Vries who must be the most generous, patient and caring person in the world.

Denise Bernadette Angela de Vries
October 2005
Adelaide.

Chapter 1

Introduction

The way we view, record and deal with information evolves. Traditionally, the definition of 'records' was intrinsically bound to the physical object on which the information was stored. Stone tablets, scrolls, lists, registers and index cards are a few physical formats that have been used through the ages. In paper-based manual systems, evolution of recording information did not present a great problem – we turned the page and ruled it up differently, renamed columns, used different terminology and proceeded to store our information. We could always review what had been stored historically by viewing the information exactly as it had been recorded. The static nature of this method means that notations that were recorded retained their semantics in context, that is the headings and layout of the form/paper imparted the structures and conventions as well as the values themselves. We, the human, translated and transformed the information when we retrieved it. It was simple. It was also so time consuming that much of what we now consider to be basic tasks, such as sorting, aggregating, summarising and reporting was infeasible.

Databases have been used to store large amounts of information since the 1970s. Database and information technology systems have evolved from simple file processing systems to powerful database systems. Data management technology has progressed from hierarchical and network systems to relational databases, data modelling tools and indexing and organisational techniques. The development of query languages, optimised query processing, transaction management and processing has resulted in the widespread use of relational databases. In the last twenty years, further developments in database technology have resulted in data models such as extended-relational, object oriented, object relational, deductive and temporal data models. Different application-oriented systems have arisen, such as spatial, temporal, multimedia and knowledge bases. As database theory developed, the schema, categorisation of data, and concept domains changed. However, the working life of a database can be a long one and there are many legacy systems and archived datasets that contain data that are still of interest to information managers, as well as newly created databases for which an extended lifetime is predicted.

The development of Relational Database Management Systems (RDBMS) and automated systems put the layout and *form* into the unchanging metadata and gave us *record once* systems. Database technology has provided the power to store and manipulate information in a variety of ways, however we still cannot reproduce the simplicity of dealing with information evolution as we previously did.

The common view of relational data modelling and relational database structures (and as a result database languages) is to consider the specification of attributes, normally defined over a restricted set of data types, as part of table definition. When the user's requirements indicate that attributes need only be defined over relatively simple (normally DBMS-supplied) data types this is generally adequate. However, in more complex applications, domain structure becomes an important issue and even for some simpler applications, there is often advantage to be gained from utilising more sophisticated domain structures, such as concept graphs (Roddick, Hornsby & de Vries 2003), hierarchies (Rice & Roddick 2000), intervals (Allen 1983), and so on. In practice, this rarely occurs and where it does, design decisions often mean that implementations are inconsistent and not transferable.

The definition of an attribute conceptually includes both the data type and its current set of valid values - its domain. However, in an RDBMS, only the data type is recorded. A domain may alter without its change being recorded and thus information is lost. Over the last thirty years, many international/universal domains have changed and some have come into being. A few examples are;

- Country names and their international country number,
- Telephone numbers and area codes,
- Postal codes/zip codes,
- Animal and plant taxonomies,
- Disease taxonomies,

- Astronomical and celestial taxonomies,
- Genome data.

The evolution of domains is inevitable. Database user requirements change due to a variety of reasons including new and changed laws, organisation mergers or splits, inventions, discoveries and developments in technology. The success of relational databases and their large market share means that large quantities of historical and current information is stored in them and as the database systems evolve there is loss of information. Domain history is not preserved unless it is part of the transactional definition. For example, if one were searching for a particular value in a paper-based system, time consuming though it was, subtle differences in data values were captured because the searcher understood the domain and therefore included or excluded records based on his/her own knowledge of the domain. For instance, a database query searching through historical medical records for an illness matching 'rubella' generally uses a string comparison only, thus the string 'German measles' would not be retrieved even though semantically it matched. The 'Year 2000' problem was an example of how important an attribute domain is. The meaning of two digits caused world-wide concern requiring legislation in many countries, large amounts of money and years of work to prevent a range of 'catastrophes', both real and imagined.

There have been many techniques developed to deal with database evolution but none can currently deal with all aspects of evolution and few of them deal specifically with the problem of attribute domain evolution. Middleware, using various approaches, has been used to alleviate evolution problems by translating, transforming or coercing data and metadata. However, semantics are lost when data is converted, coerced or replaced.

This research focuses on the utility of augmenting the information capacity of the attribute domain in a reusable and systematic manner. This also has the advantage of reducing the size of ontology definitions and allowing them to focus on the specific characteristics of the concept. This approach proposes to retain the overall structure and operations of the relational model and to enhance the capability of the attribute domain to handle more advanced semantics. This is achieved by more clearly delineating between domain definition (called *mesodata*) and schema definition (*metadata*). This approach can be considered as falling between the basic relational (minimalist) approach in which there is only a limited number of common domains, and the object-relational model in which commonality is facilitated through re-use. This middle-ground approach reflects the approach taken in research areas such as temporal, spatial and multi-dimensional modelling which argue for a flexible but unified way of handling common modelling problems. Indeed, using *mesodata* may facilitate better accommodation of such extensions.

In some regards, this work is related to that of schema integration/evolution. In these fields the provision of *mediators* or *wrappers* and the specification of *global schemas* are useful solutions where the integration solution is known and coded in advance. However, many specialised *ad-hoc* queries cannot be handled in this way as the data are generalised. There is also a wide variety of other situations where the sensible handling of commonly occurring domain structures would be useful, including temporal and spatial applications, schema versioning, data warehousing, search engines, validation and error correction and data mining (Mooney, de Vries & Roddick 2005).

Chapter 2

Literature Review

All is flux, nothing stays still.
Heraclitus (540 BC - 480 BC)

2.1 Database Organisation and Evolution

Databases have been implemented and used to store and manipulate large amounts of data in ever increasing amounts over the past three decades. During this time, database theory and design has evolved from simple file processing to powerful information management systems. Database technology has progressed from hierarchical and network systems to relational databases, data modelling tools and indexing and organisational techniques. The development of query languages, optimised query processing, transaction management and processing has resulted in the widespread use of relational databases. In the last twenty years, further developments in database technology have resulted in data models such as extended-relational, object oriented, object relational and deductive data models. Different application-oriented systems have arisen, such as spatial, temporal, multimedia and knowledge bases. Along with these developments, there have also been significant changes to computer hardware, especially in storage devices, and in computer languages and operating systems. The development of communications technologies facilitate the sharing of data, often between heterogeneous data sources. This progress, in all areas, has culminated in the current situation where it is economically viable for many businesses and organisations to have (and rely on) database systems.

This progress has also itself created a major problem: that of creating and maintaining an up-to-date information system. A database system can be perceived as being comprised of two elements the *extension* and the *intension.* The database extension is the content or 'population' of a database and contains actual values of table rows or instances. The database intension is a set of type definitions for the database and describes the format of each database table. This is also known as its Schema or Metadata (data about data).

This review concerns itself primarily with work related to relational database evolution and its various facets including schema evolution, schema translation, schema transformation, information capacity, data integration, change management and implementation. A consensus glossary (Jensen, Clifford, Elmasri, Gadia, Hayes, Jajodia, Dyreson, Grandi, Kafer, Kline, Lorentzos, Mitsopoulos, Montanari, Nonen, Peressi, Pernici, Roddick, Sarda, Scalas, Segev, Snodgrass, Soo, Tansel, Tiberio & Wiederhold 1998) provides the definitions that a database supports *schema evolution* if it allows modifications of the schema without loss of extant data and that no support for previous schemas is required, whereas it supports *schema versioning* if it allows the querying of all data, both retrospectively and prospectively, through user-definable version interfaces. The primary goal for database evolution and versioning is to preserve the integrity of the data. What will the impact of change be on views, queries and processes?

2.2 Causes of Change

Sjøberg's (1993) case study, a health management system, revealed that schema changes were significant both during the six months of development and the twelve months after the system was operational. In the study, the changes covered the gamut of evolutionary possibilities including each relation being changed, 139% increase in the number of relations, 274% increase in the number of fields, and 35% more additions than deletions. Sjøberg summarised several reasons for evolution, which include:

- People do not know in advance, or are not able to express, all the desired functionality of a large-scale application system. Only experience from using the system will enable the needs and requirements to be properly formulated.
- The application world is continually changing. A viable application system must be enhanced to accommodate these changes.

- Often the scale of the task requires incremental design, construction and commissioning. This results in requirements to change the installed subsystems.

Comyn-Wattiau et al. (2003), addressing unanticipated changes in database information systems, noted that '*the target applications are not fixed and the requirements are not always clear enough. Some divergence of the data semantics due to different viewpoints of the business policies may occur. Besides, it remains difficult to know if a logical schema meets completely business requirements. As a consequence, numerous unanticipated changes occur during the design process or even later when the system already exists.*'

A workshop on Evolution and Change in Data Management in 1999 (Roddick, Al-Jadir, Bertossi, Dumas, Estrella, Gregersen, Hornsby, Lufter, Mandreoli, Männistö, Mayol & Wedemeijer 1999) summarised six causes of change in data management.

- A change in the universe of discourse (UoD).
- A change to the interpretation of facts about the universe of discourse and the manner in which the task is realised in a system.
- Changes in the form of updates to effect upgrades to the functionality or scope of a system.
- Changes in the form of updates to effect efficiency improvements.
- Changes caused by system operation. For example, the discovery of new information which is then fed back into the system or the abnormal behaviour of a component.
- Error correction.

Furthermore, the cause of the change has an effect on the way in which the changes are managed. The differences between a planned or scheduled change and an unexpected, imposed change can cause very different procedures to be performed.

2.3 Change Management

Shankaranarayanan and Ram (2003) assert that managing *core schema evolution* includes identifying and incorporating changes to the schema while preserving the consistent state of the schema as well as propagating the changes to the data associated with the schema. Their list of issues that need to be addressed for managing core schema evolution are:

- Understanding all possible changes to the database schema.
- Understanding the implications of each change.
- Incorporating changes to an existing schema while ensuring that the consistent (and correct). state of the schema is maintained.
- Determining how a change (may/may not) affects other parts of that schema.
- Propagating changes to the data associated with the changed schema so that the data is consistent with the changed schema.
- Performing these changes dynamically without significantly impacting day-to-day operations of the database.

Evolutionary Operations

Schema evolution, as previously defined, can be viewed as a subset of schema versioning in which there is only one version maintained and available. Roddick et al. (1993) present a taxonomy of schema versioning issues with respect to the Entity-Relationship Model and the effects on the relational database model. The evolutionary operations are categorised as:

- Domain/Attribute Evolution
    - ⋆ Expanding an attribute domain
    - ⋆ Restricting an attribute domain
    - ⋆ Changing the domain of an attribute
    - ⋆ Adding an attribute to the database
    - ⋆ Renaming an attribute
- Relation Evolution
    - ⋆ Adding a relation
    - ⋆ Deactivating a relation
    - ⋆ Activating a relation

- Attribute-Relation Assignment Evolution
    - ⋆ Adding an attribute to a relation
    - ⋆ Deactivating an attribute
    - ⋆ Promoting an attribute
    - ⋆ Demoting an attribute
    - ⋆ Splitting a relation
    - ⋆ Partitioning a relation
    - ⋆ Joining two relations
    - ⋆ Coalescing two relations
- Schema Transaction Support
    - ⋆ Schema commit
    - ⋆ Schema rollback

2.4 Information Capacity

The information capacity of a schema is the set of all possible instances of that schema. The four relative information capacity measures between database structures as defined by Hull (1986, 1997) are, in progressively less restrictive order, calculus dominance, generic dominance, internal dominance and absolute dominance. These measures are used to evaluate the information capacity of two or more schemata by mathematically mapping between the schemata. An important point to note is that even when two schemas can be proved to have the same information capacity, it does not then follow that they are equivalent *semantically.*

Qian's (1996) formalisation of Abstract Data Types (ADT) for schema transformations presents a slightly different notion of information preservation which is strictly less restrictive than calculus dominance, strictly more restrictive than absolute dominance and incomparable to generic and internal dominance. These formal approaches are the foundation of later works into schema equivalence and schema integration.

Schema Equivalence

The information capacity of schema S_n is $I(S_n)$, the set of all possible instances of S_n and the *relative information capacity* of S_1 and S_2 is measured by an instance mapping associating the instances of S_1 and S_2, $f : I(S_1) \rightarrow I(S_2)$

Miller et al. (1994) describe *Equivalence* as the requirement that all data stored in one schema (S_1) can be accessed and updated through another schema (S_2).

- for queries the transformation function (f) must be total: $q(i_2) = q(f(i_1))$,
- to access all data f must be injective: i_1 must correspond to a unique i_2, a 1-1 cardinality,
- for updates f must be onto: $I(S_2)$,
- for equivalence ($S_1 \equiv S_2$) there exists a bijective (1-1 and onto) function: $f : I(S_1) \rightarrow I(S_2)$.

In practice, schema equivalence rarely exists.

Schema Dominance

Dominance $S_1 \preceq S_2$ allows all data stored under schema S_1 to be queried through S_2

- to access all data, there exists and injective function: $f : I(S_1) \rightarrow I(S_2)$, a 1-1 cardinality,
- every instance of S_1 can be transformed to an instance of S_2 without loss of information,
- S_2 may hold more information.

Davidson et al. (1998), recognising that information capacity preserving transformations do not necessarily preserve the semantics of databases, developed a declarative language called WOL (Well-founded Object Language) for expressing database transformations and constraints. They argue that approaches which allow a fixed set of well-defined transformations to be applied in series (for most methodologies the outcome is dependent on the order in which the schemas are integrated - they are not associative) are inherently limited in the class of transformations that can be expressed, and that while using a high-level language for transformations is necessary for general transformations, it is difficult to reason about, and prove, properties of transformations. This work tackles the difficulty of correctly transforming complex data structures (sets, records and variants) and recursive structures. Constraints on the source and target databases are crucial to notions of information preservation, but typically are not, or cannot, be expressed in the models of the underlying databases.

WOL allows a general class of transformations to be expressed and unifies the treatment of transformations and constraints. The class of constraints that can be expressed in WOL encompasses those found in most data models, such as keys, functional dependencies and inclusion dependencies.

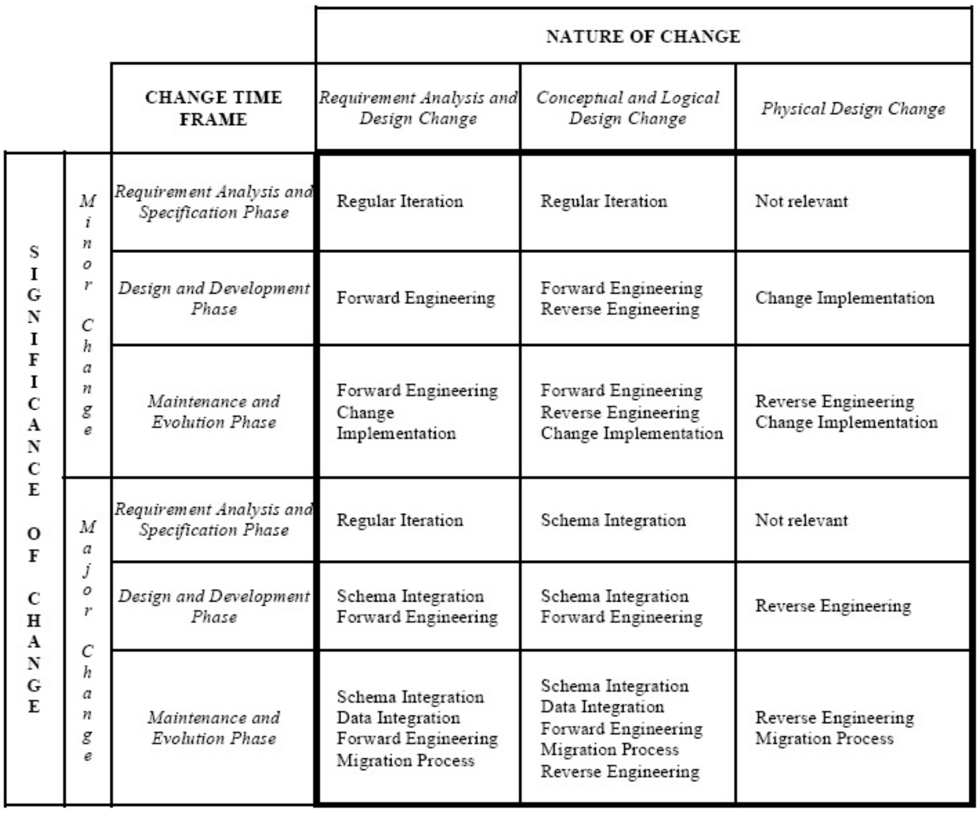
Albert (2000) presents a formalism for schema restructuring in which is included the importance of the concepts of *soundness* and *completeness*, through which a set of schema transformations can provide a syntactic characterisation of the semantic notions of dominance and equivalence of schemas.

Given an information model $< \mathcal{S}, \mathcal{L}, \mathcal{C} >$ where $\mathcal{S}$ is a schema, $\mathcal{L}$ is a query language and $\mathcal{C}$ is a family of dependencies, a τ transformation is *sound* if it always generates a schema that is equivalent to the original schema, that is, $(\mathcal{S}, \mathcal{C}) \equiv_{\mathcal{L}} \tau(\mathcal{S}, \mathcal{C})$ for every $(\mathcal{S}, \mathcal{C})$ in $< \mathcal{S}, \mathcal{L}, \mathcal{C} >$. The transformation is *weakly sound* if it always generates a schema that dominates the original schema, that is, $(\mathcal{S}, \mathcal{C}) \preceq_{\mathcal{L}} \tau(\mathcal{S}, \mathcal{C})$ for every $(\mathcal{S}, \mathcal{C})$ in $< \mathcal{S}, \mathcal{L}, \mathcal{C} >$. A set of transformations is sound if every transformation in the set is sound. A set of transformations is weakly sound if every transformation in the set is weakly sound or sound.

Let $\mathcal{T} = \{\tau_i | i = 0, 1, 2, \ldots, n\}$ be a set of schema transformations for some information model $< \mathcal{S}, \mathcal{L}, \mathcal{C} >$. The set $\mathcal{T}$ is *complete* if for any two enriched schemas $(\mathcal{S}_1, \mathcal{C}_1)$ and $(\mathcal{S}_2, \mathcal{C}_2)$ in $< \mathcal{S}, \mathcal{L}, \mathcal{C} >$ such that $(\mathcal{S}_1, \mathcal{C}_1) \equiv_{\mathcal{L}} (\mathcal{S}_2, \mathcal{C}_2)$, there exists a finite sequence of transformations, τ_{i_k} for $k = 1, 2, \ldots, n$ with each $\tau_{i_k} \in \mathcal{T}$, such that $(\mathcal{S}_2, \mathcal{C}_2) \backsimeq \tau_{i_n} \circ \tau_{i_{n-1}} \circ \ldots \circ \tau_{i_1}$ $(\mathcal{S}_1, \mathcal{C}_1)$. The set $\mathcal{T}$ of transformations is *weakly complete* if for any two enriched schemas $(\mathcal{S}_1, \mathcal{C}_1)$ and $(\mathcal{S}_2, \mathcal{C}_2)$ in $< \mathcal{S}, \mathcal{L}, \mathcal{C} >$ such that $(\mathcal{S}_1, \mathcal{C}_1) \preceq_{\mathcal{L}} (\mathcal{S}_2, \mathcal{C}_2)$, there exists a finite sequence of transformations, τ_{j_k} for $k = 1, 2, \ldots, m$ with each $\tau_{j_k} \in \mathcal{T}$, such that $(\mathcal{S}_2, \mathcal{C}_2)$ $\backsimeq \tau_{j_m} \circ \tau_{j_{m-1}} \circ \ldots \circ \tau_{j_1}$ $(\mathcal{S}_1, \mathcal{C}_1)$.

2.5 Techniques for Database Evolution

Comyn-Wattiau et al. (2003) present a framework for database systems evolution, summarised in Figure 2.1, that takes into account three dimensions of change and propose techniques to be used in cases of a requirement, a conceptual and logical, or a physical change. Their dimensions of change are the *nature of change*, the *change time frame*, and the *significance of change*. The subset of techniques specifically for major changes of the conceptual and logical design during the 'Maintenance and Evolution Phase' include schema integration, data integration, forward engineering, and change implementation.



		NATURE OF CHANGE		
	CHANGE TIME FRAME	*Requirement Analysis and Design Change*	*Conceptual and Logical Design Change*	*Physical Design Change*
SIGNIFICANCE OF CHANGE — *Minor Change*	*Requirement Analysis and Specification Phase*	Regular Iteration	Regular Iteration	Not relevant
	Design and Development Phase	Forward Engineering	Forward Engineering Reverse Engineering	Change Implementation
	Maintenance and Evolution Phase	Forward Engineering Change Implementation	Forward Engineering Reverse Engineering Change Implementation	Reverse Engineering Change Implementation
SIGNIFICANCE OF CHANGE — *Major Change*	*Requirement Analysis and Specification Phase*	Regular Iteration	Schema Integration	Not relevant
	Design and Development Phase	Schema Integration Forward Engineering	Schema Integration Forward Engineering	Reverse Engineering
	Maintenance and Evolution Phase	Schema Integration Data Integration Forward Engineering Migration Process	Schema Integration Data Integration Forward Engineering Migration Process Reverse Engineering	Reverse Engineering Migration Process

Figure 2.1. Techniques for Database Information Systems
(Comyn-Wattiau, Akoka & Lammari 2003)

2.6 Schema Integration

The goal of schema integration is to allow data from different sources to be used together to provide a unified view of the data. An integrated schema has characteristics that

- preserve the autonomy and function of local DBMS, and
- provide a uniform view of data,

Schema integration consists of the following sub-tasks (Elmasri & Navathe 2000):

1. Identifying correspondences and conflicts among the schemas. Correspondences of the same real-world concepts must be identified. Several types of conflicts among schemas may be discovered.
    - Naming Conflicts: synonym (identical entity with different names) and homonym (different entities with the same name)
    - Type conflicts: Same concept represented by different modelling constructs.
    - Domain (value set) conflicts: Different domains for the same attribute.
    - Conflicts among constraints: different primary keys, different cardinalities for the same relationship.
2. Translating schemas by employing translation algorithms between data models, be they relational, ER, hierarchical or object-oriented.
3. Integrating and transforming schemas, for example merging relations or classes with common attributes.

Schema integration architectures include the Common Data Model (CDM), the Schema Intension Graph (SIG), Hypergraph Data Model (HDM) and EVolutionary ER diagrams (EVER).

2.6.1 Common Data Model

Xu and Poulovassilis (1997), when addressing the integration of deductive databases, considered both the extensional and intensional parts of the component databases for integration. The Common Data Model (CDM) uses a binary relational Entity-Relationship model with subtyping to integrate the extensional parts. The authors proposed a semi-automatic method which requires only the declaration of the relationships between schema constructs to perform the integration. For the purposes of their model, they have generated the following definitions.

A database is a quintuple of sets <Schema, EDB, IDB, CDB, PDB> where:

- Schema (the schema) consists of the type declarations and the subtype relationships between entity types.
- EDB (the extensional database) consists of the data functions and the type extent functions.
- IDB (the intensional database) consists of the derived functions.
- CDB (the constraint database) consists of the integrity constraints.
- PDB (the procedural database) consists of all other intensional functions.

Each of these sets is integrated in turn. The Schemas and the EDBs are represented by directed graphs and from those graphs correspondences between nodes are declared. These mappings are then used to perform the integration into a CDM.

- IDB integration is performed by translating the definitions of the derived functions into the constructs of the integrated database, followed by a comparison of the semantics of pairs of derived functions to determine whether they can be integrated into one function.
- CDB integration is performed by translating their definitions to the constructs of Schema, EDB and IDB and then, as with the derived functions, perform a comparison if the semantics of the constraints. Constraints are transferred to the integrated DB if every component database contains the particular constraint or if some database(s) contain a constraint. That is, all constraints are translated even if a specific constraint is not defined in each and every component database.
- PDB integration is accomplished by translating the function definitions of the component databases into the database constructs of the thus far integrated database and incorporating the definitions into the PDB.

2.6.2 Schema Intension Graphs

Miller et al. (1993–1994) point out that whilst equivalent information capacity is a required condition, it is not sufficient to guarantee a natural correspondence between schemas and, in practice, database administrators rely on their own intuition when defining transformations between schemas. They define the Schema

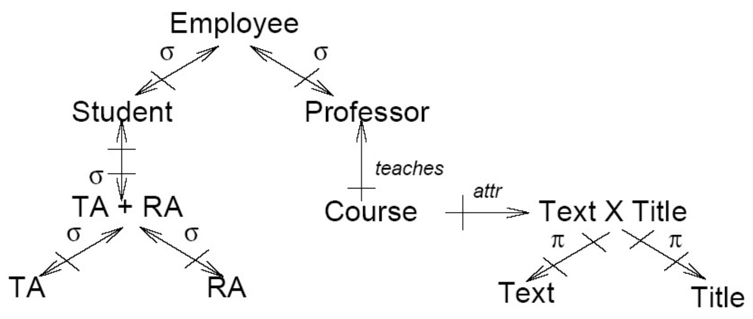


Figure 2.2. SIG - Schema Intension Graph
(Miller, Ioannidis & Ramakrishnan 1994*a*)

Translation problem as given two schemas one needs to know with respect to information capacity if each instance of the first schema can be represented as an instance in the second schema and whether the translation can be reversed?

Schemas, in practice, contain constraints that define which instances of a schema are meaningful in a certain context. Their research in this area shows that deciding information capacity equivalence and dominance of schemas is an undecidable problem. As a result, they developed tests to evaluate equivalence and dominance more restrictively. These tests utilise a set of schema transformations that declare that Schema S1 is dominated by schema S2 if and only if there is a sequence of transformations that converts S1 to S2. These transformations use Schema Intension Graphs (SIG) data models, as shown in Figure 2.2, with algorithms for deciding equivalence of schemas with constraints, to aid in understanding the relative information capacity of schemas containing constraints.

The SIG model must be data-centric rather than type-centric in order to reason about constraints on collections of entities rather than the internal structure of a single entity. This approach ignores *data type* changes and the conflicts and problems that type changes present.

2.6.3 Hypergraph Data Model

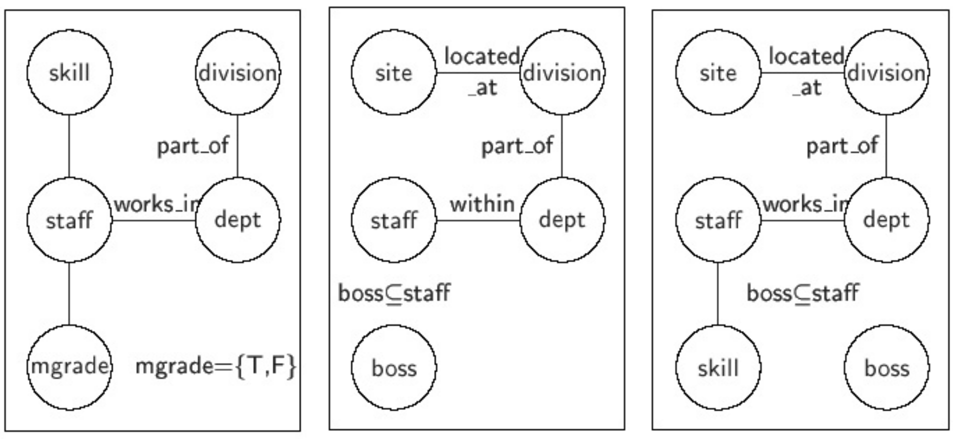
Schema transformation consists of the tasks of schema conforming, schema merging and schema restructuring. McBrien et al. (1997–1998) present a formal framework for ER schema transformation in which they have defined a set of primitive transformations based on schema equivalence. This is achieved by formalising a database as a set of sets, where an ER schema S is a quadruple <Ents, Incs, Atts, Assocs>

Ents $\subseteq$ Names is the set of entity type names.

Incs $\subseteq$ (Ents $\times$ Ents) each pair $< e_1, e_2 > \in$ Incs representing that e_1 is a subtype of e_2 and Incs is acyclic.

Atts $\subseteq$ Names is the set of attribute names.

Assocs $\subseteq$ (Names $\times$ Names $\times$ Cards $\times$ Cards) is the set of associations. For each relationship between two entity types e_1, $e_2 \in$ Ents, there is a tuple $(rel\_name, e_1, e_2, c_1, c_2) \in$ Assocs where c_1 indicates the lower and upper cardinalities of instances of e_2 for each instance of e_1, and c_2 indicates the lower and upper cardinalities of instances of e_1 for each instance of e_2. Note that rel_name may be Null if there is only one relationship between e_1 and e_2.



(a) S_1: staff skills (b) S_2: staff locations (c) S: global schema

Figure 2.3. HDM - Two Source Schemas and One Global Schema
(McBrien & Poulovassilis 2002)

Continuing this work and combining schema integration and schema evolution activities, the authors (McBrien & Poulovassilis 2002) propose using a Hypergraph Data Model (HDM), as illustrated in Figure 2.3, to build a global schema from heterogeneous source schemata and from this transformations may be used to translate queries between the global and source schemas.

Schema transformations defined on the HDM are reversible. Every *add* transformation step is reversed by a *delete* transformation with the same parameters, *renaming* transformations from $S_1 \rightarrow S_2$ are the reverse of $S_2 \rightarrow S_1$.

Contract transformations (i.e. deletions) map to void, queries and sub-queries over such constructs then translate to void. *Extend* transformations (i.e. additions) require domain knowledge, either from a human expert or a domain ontology and cannot be automated. Higher level modelling only works with names, tables, relations but not at the attribute data type level.

2.6.4 Evolutionary ER Diagrams

Liu et al. (1994) developed EVER, an EVolutionary ER diagram, for specifying the derivation relationships between schema versions, relationships among attributes, and the conditions for maintaining consistent views of programs. The EVER diagram serves as the visualization aid that graphically conveys changes to a database schema. The diagram is then transformed to an intermediate representation called version derivation graphs (VDGs) which are subsequently mapped into the structures of an underlying database.

In order to support the specification of changes to ER diagrams, the basic graphical constructs of ER diagrams are extended to present the relationships of schemas before and after a change. The following relationships can be expressed in an EVER diagram.

- The evolution relationship of the new schema,
- the relationships of attributes between the new schema and the old schema,
- the relationship of a new schema to the other schemas, and
- the invariant views of programs to the database.

The evolution relationship indicates the source of the new schema changes. The attribute relationships specify the effect of changes to an attribute on the others, and can be represented by functions. The change to an edge between an entity and a relationship type implies that the participation of the entity type in the relationship type needs to be established or dropped.

Consequently, the relationship type needs to be evolved by adding or deleting the key attribute of the affected entity type from the relationship type. The conditions for maintenance of invariant program views pre-empt changes that would cause conflicts so that the programs can access the evolved database consistently.

2.6.5 Schematic Conflicts

Schemas are said to be *semantically* equivalent when they model the same features in the universe of discourse (UoD). Conflicts can still occur if semantically different information is stored under the same name or semantically equivalent information is stored under different names. Table 2.1 summarises the possible conflicts that can occur between semantically equivalent schemas.

Table 2.1. Schematic Conflicts

	Value	**Attribute**	**Table**
Value	Domain conflicts between schema-1 and schema-2: e.g. expression conflicts, data unit conflicts, precision conflicts	The values in schema-1 are used as attributes in schema-2	A value in an attribute in database schema-1 is semantically equivalent to the scope of a table in database schema-2 or, the data in one table are the metadata in another table.
Attribute		Different definitions for semantically equivalent attributes: * 1-1 one attribute used to model the same information in each schema e.g. naming conflict, integrity constraint conflict, data representation conflict * 1-N and N-N different numbers of attributes used to model the same information in each schema. The N-N conflict is a generalisation of the 1-N conflict.	Data are stored as an attribute(s) in schema-1 and as table(s) in schema-2. (not semantically equivalent)
Table			* The total number of tables is different between the schemas but both are semantically equivalent. This results from value vs table and attribute vs table conflicts * The set of attributes is different between schemas, e.g. after an addition or a deletion of an attribute. Results in a '*missing*' attribute. Missing attributes can be implicit (derived) or explicit (not derivable).

2.7 Schema Matching

Schema matching is the mapping of semantically corresponding elements between two schemas. Rahm and Bernstein's (2001*a*) survey of approaches to automatic schema matching presents classifications of different ways to perform matching.

Instance vs schema: matching approaches can consider instance data (i.e. data contents) or only schema-level information.

Element vs structure matching: a match can be performed for individual schema elements, such as attributes, or for combinations of elements, such as complex schema structures.

Language vs constraint: a matcher can use a linguistic based approach (e.g. based on names and textual descriptions of schema elements) or a constraint-based approach (e.g. based on keys and relationships).

Matching cardinality: the overall match result may relate one or more elements of one schema to one or more elements of the other, yielding four cases: 1:1, 1:n, n:1, n:m. In addition, each mapping element may interrelate one or more elements of the two schemas. Furthermore, there may be different match cardinalities at the instance level.

Auxiliary information: most matchers rely not only on the input schemas S1 and S2 but also on auxiliary information, such as dictionaries, global schemas, previous matching decisions, and user input.

Although a great deal of the work done in schema matching is manual, semi-automatic schema matching systems have been developed, some of which are discussed in Section 2.12. Do et al. (2002) compared eight schema matching prototypes on the criteria of Input, Output, Quality Measures, and Effort. The results of this comparison are inconclusive due to the variety of evaluation measures used by the systems, however of the eight systems only two did not require manual pre-match effort and manual post-match effort was needed for finding missing matches, removing false positives, and verifying the correct results. The authors could not quantify the post-match effort, but it seems reasonable to assume that the more complex the schemas the more manual (pre and post) effort is required. Other work in this field can be found in (Berlin & Motro 2002, He & Chang 2004, Rosenthal, Seligman & Renner 2004, Bernstein, Melnik, Petropoulos & Quix 2004, Embley, Xu & Ding 2004).

2.8 Semantic Heterogeneity

Domain evolution, that is changes to the 'real-world' domain, may lead to semantic heterogeneity within a database. Ventrone and Heiler (1991) described different forms of domain evolution that introduce shifts in meaning such that simple or automatic mappings between 'old' and 'new' values are not tractable. Forms of evolutionary problems identified in this work were:

1. *Heterogeneous Instances*: Over time, different occurrences of the same value in a domain extension may have different meanings.

2. *Cardinality Changes*: Cardinality relationships between domains may also change over time.

3. *Granularity Changes*: Values may be added to a domain extension that represent a different granularity from the existing population.

4. *Encoding Changes*: Database values often have encoded meanings. These may be relics of predecessor manual systems or they may creep into systems over time, possibly to store information that is not otherwise provided for in the existing system.

5. *Time and Unit Differences*: Database values that users wish to compare may be incompatible due differences in time or units of measurement. Stored calculations in the same domain may, over time, be the products of different formulae.

6. *Identifier Changes*: In response to changing needs, indexing strategies may change over time, leading to parallel and even overlapping identifier schemes.

7. *Field Recycling*: In many systems it is difficult or infeasible to alter certain characteristics of database. Perhaps record sizes cannot be altered because of application or system software dependencies. Changing the names of fields may involve reloading the database or recompiling hundreds of software modules. The response in many cases to this inflexibility is to recycle an existing field so that the new use may have different semantics from the old one.

The key solution strategy proposed by the authors is to make semantic information explicit so that it can be read and interpreted by the application code, thus replacing *semantic* heterogeneity with *syntactic* heterogeneity. The latter problem being more tractable. More explicit semantic information is included in the metadata of the database by specifying constraints, cardinality relationships, units of measurement, derivation algorithms and formulae, confidence measures and heuristics. This does not completely solve the problem, nor in particular, does it solve the problem of the evolving domain for an attribute.

2.9 Object-Relational Databases

Database platforms implementing SQL:1999 or SQL:2003 are not exclusively relational platforms, but are object-relational platforms. A database management system is an object-relational database system (ORDBMS), if it supports both the object-oriented data model and the relational database model, and consists of the following components;

- types, tables, and views,
- subtype and sub-table relationships,
- constraints and assertions,
- functions, stored procedures, and triggers,
- roles and privileges.

The two data models interoperate by allowing the values in the relational model to be object references. However, the relational model must be designed specifically to be an object-relational database. All hard-coded type, operator and function information must be extracted and replaced with a table-driven scheme with additional parser support added for SQL syntax for complex objects and inheritance.

Elmasri and Navathe (2004) note that ORDBs compared with RDBs have additional problems.

- Database design is more complicated because the designer must consider not only the underlying design considerations of application semantics but also its object-oriented nature.
- Query processing and optimisation is more difficult because of SQL extensions and user-defined functions and rules.

- Interaction of rules with transactions requires deferred execution of triggers which involves additional processing.

Evolution of ORDBs pertains to all changes that can be made any of the components of the database schema. Türker (2000) noted that *A database schema is formed by a set of schema element definitions and it evolves by adding, altering, or removing schema element definitions. It is important to note that some schema evolution operations may also have an effect on the actual database objects, for instance, on the rows of a table.* The evolutionary operations in an ORDB are shown in Table 2.2. Refer to Türker's (ibid) comprehensive survey of schema evolution in SQL:1999 for a detailed discussion of these operations, which have not altered in SQL:2003.

Table 2.2. Evolutionary Operations in ORDBs

	Create	**Alter**	**Drop**
Domain	✓	✓	✓
Type	✓	✓	✓
Table	✓	✓	✓
View	✓		✓
Assertion	✓		✓
Procedure	✓	✓	✓
Function	✓	✓	✓
Trigger	✓		✓
Role	✓		✓
Privilege	✓		✓

SQL:2003 introduced extensions to the CREATE TABLE LIKE and CREATE TABLE AS statements that are useful when evolving tables. The former has options to enable copying of more information, such as identity column options, the expressions used for generated columns, and the default values, while the latter creates a populated table independent of the underlying query with respect to future updates of the source table(s).(ISO/ANSI 2003, Eisenberg, Melton, Kulkarni, Michels & Zemke 2004)

Whilst user-defined types and functions provide more flexibility with the provision of inheritance, when managing evolution in ORDBs one must still apply the techniques developed for RDBs for the underlying relational tables, as well as some of the techniques for evolving Object-Oriented databases (OODBs) when evolving objects, as can be found in (Lemke 1994, Liu, Chang & Chrysanthis 1994, Baekgaard 1997, Liu, Zicari, Hursch & Lieberherr 1997, Peters & Özsu 1997, Claypool, Natarajan & Rundensteiner 1999, Parsons & Wand 2000, Franconi, Grandi

& Mandreoli 2001, Rashid 2002). Shankaranarayanan and Ram's (2003) report on core schema evolution management in databases reviews the variety of research for OODBs.

2.10 Data Conversion

Currently when a schema changes two events typically occur - the application is modified and recompiled to deal with the changes and the data are converted to the new format, either by *strict*, *lazy* or *no* conversion (Ferrandina, Meyer & Zicari 1994). Lazy conversion performs data conversion only when data are accessed and they are still recorded with superseded formats (or values), no conversion is done if the data are not accessed. The advantage of this approach is that only the data that are used are converted and the whole database does not need to be locked or taken off-line to perform the conversion. The disadvantages are that a record of schema changes must be recorded and accessible and that every time data are accessed they must be checked to see if they conform to the current schema. Until all data have been accessed there exist some that may be invalid or incomplete.

Strict conversion requires that as soon as there is a modification to the schema all data are converted to conform with the current definition. The advantage of this approach are that all data are immediately consistent with the new schema. The disadvantage is that all applications interacting with the database must be stopped and the database locked while the conversion takes place. Depending upon the nature of the modifications this can take a long time. In addition, information is lost and changes cannot be reversed.

2.10.1 Attribute Evolution

Discussing the problems of schema evolution in OODBMS, Lemke (1994) cites issues relating to the evolution of attributes with regard to behavioural consistency (including program compatibility) and instance compatibility. These issues are similar to attribute evolution in RDBMS and manifest themselves as program compatibility and data consistency.

Program Compatibility

- If an added attribute already exists, all application procedures/functions accessing the attribute may change their behaviour or become invalid if the new attribute definition is not a generalisation of the old one.

- If a deleted attribute is replaced by one with the same domain, nothing happens. If the new attribute has a different domain, application procedures/functions using this attribute might change their behaviour. If the deleted attribute is not replaced, code referencing the attribute becomes invalid.
- Attribute changes are divided into name and domain changes. For a name change, access to the attribute under the old name becomes undefined, therefore all procedures/functions using the old name are also invalid. The source code becomes invalid, thus disabling further changes and recompilations of the code. If the new name occurs elsewhere in the schema, all procedures referring to the old definition might become invalid with respect to the domain of the attribute, or might change their behaviour. Changing the domain of an attribute may make referencing methods invalid when recording values into the attribute which are no longer in the domain, or if values from the new domain do not obey the constraints of the attribute.

Data Consistency

- If an added attribute replaces an existing attribute definition, this is equivalent to a change of the attribute definition. Otherwise, the new attribute logically has to be added to relations and to the application. For existing relations, the new attribute should be populated with a default value.
- If an attribute is deleted without any replacement, all tuples in the relation logically lose this attribute.
- Changing the name of an attribute does not have an impact on existing values as the metadata is separate from the data.
- If the domain of the attribute is generalised, existing tuples are not affected because all existing values still belong to the domain. If the domain is specialised, only those tuples are affected whose attribute value is no longer within the domain. For a general change of the domain, the attribute value of all tuples logically has to be converted to a value of the new domain.

2.11 Data and View Integration

Research on data integration specifically for database evolution has not been found, however there is research regarding data integration for heterogeneous systems, data warehousing and information sharing.

Embury and Gray (1998) surveyed the use of active rules to support database application development and provided guidelines for the kind of behaviour to which they can be most successfully applied. They focused on three classes of database functionality to which the rules have been applied: integrity maintenance; support for database views and data integration; and the implementation of advanced transaction models.

An active database supports Event-Condition-Action (ECA) rules. The occurrence of various types of events (e.g. database transitions, time events, and external signals) triggers the evaluation of a condition, which when true causes the action to be carried out. With regard to ECA rules for views and data integration, the authors noted that the implementer of a view mechanism must consider two issues: how to provide efficient retrieval of data and how to deal with attempts to update the view classes or relations. Active rules can be used to trap accesses to view data and rules can be generated automatically from the original high-level view definition.

Reference is made to the earlier work done by Stonebraker et al. (1990) showing how active rules can be used to solve the view-update problem. This solution requires a specific update policy to be associated with each view. *A rule is created for each possible update to each virtual class, relation, or attribute whose action describes the update that must be made to the base data in order to create the effect of the required update to view.* This methodology requires that, at the time the view is defined, a policy for updates is also defined. This is not always possible as there exist many situations where the decision must be made within the context of the actual update based on the data values which are not known *a priori.*

Cited also is the later work by Ceri and Widom (1991) who proposed that four active rules are generated for each concrete table in a view definition. One rule triggered on inserts to a table, one on delete events, and two on update events. Thus, for an update on base table T, they generate incremental versions of the view definition expression by replacing references to T with the special transition tables inserted T, deleted T, old-updated T and new-updated T, giving a set of expressions that computes only those tuples that have been added to (or deleted from) the view relation. These generated rules are either those that cater for insertions or for deletions, rules for deletions are given a higher priority so that they are triggered before the insertion rules. Embury and Gray point out that while the ECA rules can be used to allow updates to views (both virtual and materialised) there can be conflicts of update policies generated from view definitions leading to violations of integrity constraints. In this situation, the Database Administrator must intervene in the process.

2.12 Mediation Techniques

Mediation techniques provide access to heterogeneous data sources by translating application queries into source specific commands or queries. These approaches are known as mediators, data wrappers, translators or middleware.

Generally, mediators are either Global-As-View (GAV) or Local-As-View (LAV) based. In GAV mediators, relations are written in terms of the source relations. Consequently addition or removal of a source requires modification of the mediator schema. In a LAV mediator every source relation is defined over the relations and schema of the mediator and it is therefore easier to add or remove relations but complicates query reformulation. Lenzerini (2002) presents a comprehensive tutorial on the theoretical issues of data integration. When sources are stable, GAV is the preferred approach. An evolved database implies a global schema (the new schema) with multiple sources (the old schema(s) and the new), hence this section focuses on GAV approaches. Parent and Spaccapietra (1998) summarised the steps to database integration as follows;

1. Pre-integration, where input schemas are transformed to make them more homogeneous (both syntactically and semantically),
2. Correspondence Identification, devoted to the identification and description of inter-schema relationships, and
3. Integration, the final step which solves inter-schema conflicts and unifies corresponding items into an integrated schema.

TSIMMIS

The Standford-IBM Manager of Multiple Information Sources (TSIMMIS) (Chawathe, Garcia-Molina, Hammer, Ireland, Papakonstantinou, Ullman & Widom 1994, Garcia-Molina, Papakonstantinou, Quass, Rajaraman, Sagiv, Ullman, Vassalos & Widom 1997, Li, Yerneni, Vassalos, Garcia-Molina, Papakonstantinou, Ullman & Valiveti 1998) offers a data model and a common query language to support the integration of information from multiple different sources. The components of TSIMMIS are:

- The Object-Exchange Model (OEM) to convey information among components.
- Mediators, to answer queries about different entities, are specified with a logic-based object-oriented language called Mediator Specification Language (MSL). This is a view definition language for the OEM and includes the necessary functions for data integration.
- Wrappers, specified with the Wrapper Specification Language (WSL), an extension to MSL, to allow for the description of source contents and querying capabilities. These convert user queries to source specific queries.
- A common query language to link components.
- Wrapper and mediator generators - wrappers may be generated using a template-based tool or user-written functions to connect the wrapper to the source and generate the queries on that source. At run-time, the Mediator Specification Interpreter (MSI) collects and integrates the necessary information from the sources, according to the specification. A mediator uses the raw sources, interfaced by a wrapper, or other mediators.

Garlic

The Clio and Garlic systems (Haas, Miller, Niswonger, Tork Roth, Schwarz & Wimmers 1999, Miller, Hernandez, Haas, Yan, Ho, Fagin & Popa 2001) provide an integrated view of a variety of heterogenous data sources. Clio's components are schema, correspondence, and mapping management. These components utilise metadata, query workloads, view definitions and mine the data to produce a global schema. The Garlic wrapper is an interface which describes the data and provides mechanisms for data retrieval. The data are described using a variant of Object Database Management Group (ODMG) Object Description Language (ODL) named Garlic Definition Language (GDL) which enables the wrapper to rename objects and attributes, change types and define relationships.

As more information was becoming available on the internet, research in data integration moved to include semi-structured data sources and integrate these dependent on their context and semantics. These later models utilise common vocabularies and ontologies.

MIX

Bornhövd and Buchmann (2000) present a framework for semantically meaningful data exchange. To allow meaningful exchange in heterogeneous databases there needs to be commonly agreed upon vocabularies or ontologies to describe the data and metadata. Implicit assumptions about the meaning of data are mapped, using data wrappers, to a common representation model 'Metadata-based Integration mode for data eXchange' or MIX. Domain specific ontologies are structured and organised in a MIX Based Integrated Architecture (MIBIA). They propose an approach to represent additional information at the extensional level using data wrappers as Java classes to represent these ontology concepts and their relationships.

IBIS

IBIS (Internet Based Information System) (Cali, Calvanese, De Giacomo, Lenzerini, Naggar & Vernacotola 2002), is a system for the semantic integration of heterogeneous data sources that allows the specification of integrity constraints in the global schema derived from the domain of interest. The system has four main components: Wrapping, Configuration, Core, and User Interface. The Core implements the data integration algorithms and performs evaluation of a query by extracting data from the sources and executing the query those data.

IBIS adapts the information extracted from sources that are typically autonomous and incomplete using a data extraction strategy based on a concept of *proximity* of values to the tuples constituting the answer to the query.

IBIS takes into account the integrity constraints over the global schema, which reflect the semantics of the application domain, and allows for the retrieval of data that were not usually retrieved in earlier data integration systems. This is done by encoding information about integrity constraints in an expanded query, so that the answers provided by evaluating the pre-processed queries are the best possible ones that can be obtained, given the available information.

SCROL

Ram and Park (2004) propose a common ontology called Semantic Conflict Resolution Ontology (SCROL) to capture context knowledge, that may be used to identify and resolve semantic conflicts among heterogeneous databases. SCROL is a domain independent ontology, that encodes commonly found semantic conflicts, which subsequently provides an automatic way of comparing and manipulating contextual knowledge of each information source.

This approach is similar to using a federated schema but has the advantage that the common ontology can be reused by a range of application domains. The limitation of this approach is that there needs to be a direct mapping for each schema or data component. The authors note that it is not possible to resolve conflicts such as 'known data value reliability' and 'spatial domain' that occur at the data value level.

MOMIS

MOMIS, developed by the DBGroup at the University of Modena and Reggio Emilia (Beneventano & Bergamaschi 2004), is a framework for information extraction and integration of heterogeneous information sources, which implements a semi-automatic methodology for data integration. The result of the integration process is a global schema, which provides a reconciled, integrated and a virtual view of the underlying sources, GVV (Global Virtual View). The GVV is composed of a set of (global) classes that represent the information contained in the sources, and it is the result of the integration process, i.e. a conceptualisation of the underlying domain (domain ontology) for the integrated sources. The GVV is then semi-automatically annotated according to a lexical ontology. This approach differs from others in that it constructs a domain ontology as the synthesis of the integration process, as opposed to using an a priori constructed ontology.

Multiplex, Fusionplex, Autoplex

Three generations of data integration systems have been developed at George Mason University (USA) (Motro, Berlin & Anokhin 2004), called Multiplex, Fusionplex and Autoplex. Multiplex, the basis for the latter two systems, provides the formal model of integration which distinguishes between schema consistency and instance consistency. The former assuming that schema differences are reconcilable, the latter assuming that instance differences are not reconcilable. In addition, it also provides approximate answers to queries for situations where there is either too much or too little information. Fusionplex utilises data quality features and user preferences to resolve instance differences and rank inconsistent answers to provide query responses. Autoplex uses machine learning techniques and user effort to discover member schemas and incorporate those into the global schema.

Yacob

Using domain knowledge in the form of concepts and their relationships for formulating and processing queries, the Yacob mediator (Sattler, Geist & Schallehn 2005) model is based on a resource description framework schema (RDFS) meta-model, combining the representation of concepts as terminological anchors for integration with information describing the mapping between global concepts and local schemas using a GLAV approach, combined with a query language CQuery (a derivative of XQuery) for formulating the queries.

MADS

In a similar approach, but specifically tackling the more complex issues of spatio-temporal data, are the MADS conceptual data model and MADS spatial and temporal domain ontologies (Sotnykova, Monties & Spaccapietra 2000). Spatial and temporal aspects include not only spatial and temporal entities but also space-related and time-related relationships between these entities which are managed within this model through the use of the ontologies.

2.13 Ontologies

An ontology is a formal, explicit description of concepts in an area of interest (domain of discourse). It is *a vocabulary of such terms (names of relations, functions, individuals), defined in a form that is both human and machine readable.* (Gruber 1993). The three categories, identified by Jasper and Uschold (1999), of benefit from using ontologies are: *Communication* - by providing consistency, lack of ambiguity, and integration of different perspectives; *Inter-operability* - by providing computer systems with an interchange format for translation between languages, representations and paradigms; and *Software Engineering* - by providing a shared understanding for specification, resulting in re-use and improved reliability.

The main components of an ontology are: (1) *concepts* or *classes* of the domain of discourse or tasks, which are often organised in taxonomies; (2) *relations* describing the types of interaction between concepts of the domain, e.g. is-a, subclass-of; (3) *functions* specifying a special type of relation such as definitions of calculated attributes; (4) *axioms* specifying rules that constrain the interpretation and use of these terms; (5) *instances* to represent specific elements. These components are then defined in an ontology as:

- Concept a.k.a. class, category, type, term, entity, set and thing.
- Slots a.k.a. roles, properties, relation, relationship, association, function and attribute.
- Facets a.k.a. role restrictions, criterion, constraint of, feature and predicate.

As can be inferred from the alternative labels and terms for the constituent parts of an ontology, there is no single standard for the structure and design of an ontology. Ontology representations are broadly categorised by Wache et al. (Wache, Vogele, Stuckenschmidt, Schuster, Neumann & Hubner 2002) in the following ways:

- Frame-Based systems, providing a structure for representing a concept or situation, such as OKBC (Open Knowledge Base Connectivity), Ontolingua, F-Logic (Frame Logic), XOL (XML-based ontology-exchange language), RDF (Resource Description Framework). Frames consist of slots for which values have to be specified. Properties and restrictions can be provided for these values.

- Description Logics, providing formal semantics and reasoning support, such as CLASSIC, GRAIL and LOOM. Knowledge is described in terms of concepts and role restrictions. Starting with atomic concepts and roles, an ontology is built by adding definitions of new concepts and their relationships in terms of existing concepts and roles. Description Logics are descendants of frame-based systems.

- Formal Concept Analysis, providing a mathematical model to integrate information from different sources into a common concept hierarchy.

- Object Languages are ontology languages designed for specific purposes within domains of interest. These languages are developed so that the ontologies can represent accurately the entities, concepts and detail required in the domain.

- Annotated Logics are use to resolve conflicts, by which the values of confidence or belief calculate the most promising fact to include in the common model.

The strength of ontologies lies in them being shared computer-based resources, and as Meersman and Jarrar point out (2002) *an ontology needs to be even more* generic, *across tasks and even task types, than a data model is for a number of given applications. Just adding a mere* 'is_a'- *taxonomy of terms is not sufficient, as the literature sometimes seems to suggest. An ontology needs to include (the meaning of) a much richer set of relationships, such as* instance_of, part_of, ..., *which depending on the domain all might deserve a 'generic semantics'.* Thus,

Vocabulary + Structure ⇒ Taxonomy
and
Taxonomy + Relationships, Rules and Constraints ⇒ Ontology

Although it may seem desirable to have a single ontology to simplify sharing and integration of concepts, this is not always possible. Kashyap and Sheth (1996) discuss issues of reconciling semantic and schematic perspectives with domain specific ontologies. An ontology may serve multiple users' perspectives and their applications, and striving to be all things to all users can produce an ontology that is difficult to build, maintain and use. Wache et al. (Wache, Vogele, Stuckenschmidt, Schuster, Neumann & Hubner 2001) identify three directions taken in ontology architecture, 1) the Single Ontology approach that uses one global ontology to provide a shared vocabulary; 2) Multiple Ontologies approach in which each information source is described by its own ontology and an inter-ontology mapping identifies semantically corresponding terms in the different source ontologies; 3) Hybrid Ontology approach in which the semantics of each source is described by its own ontology that are built from a global shared vocabulary.

There has been a significant amount of work done in recent years in the research and development of techniques for utilising ontologies for the *Semantic Web*. The aim being for computers to share, relate and combine information over the World Wide Web. See (Vianu 2001, Motik, Maedche & Volz 2002, Horrocks 2000, Horrocks 2002*a*, Horrocks 2002*b*, Doan 2002).

Schema evolution and ontology evolution share many of the same problems and solutions which are discussed by Roddick (1995) and Klein et al. (Klein & Fensel 2001, Klein 2001, Klein 2002, Klein, Fensel, Kiryakov & Ognyanov 2002, Noy & Klein 2002, Spyns, Meersman & Jarrar 2002) respectively. Extending the relational model to utilise these methods adds a powerful new dimension to database information systems.

2.14 Concept Graphs

The Conceptual Graph Standard edited by Sowa (2001) provides a guide for the implementation of conceptual graphs in systems. The conceptual graph is *an abstract representation for logic with nodes called concepts and conceptual relations linked together by arcs.* These provide an abstract model which can be used at different levels. At a conceptual level, it can be the basis for a specialised communication language between experts of different disciplines involved in a common project. At an implementation level, it can be the basis for a common representation tool used by several modules of a complex system, integrating knowledge and databases, inference engines, sophisticated human-computer interfaces, and learning modules. The standard defines a conceptual graph thus:

A *conceptual graph g* is a bipartite graph, which consists of two kinds of nodes called *concepts* and *conceptual relations.*

- Every *arc a* of g is a pair (r, c) consisting of a conceptual relation r and a concept c in g. The arc a is said to *belong* to r; it is said to *link* r to c; but it does not *belong* to c.
- A conceptual graph g may have concepts that are not linked to any conceptual relation; but every arc that belongs to any conceptual relation r in g must link r to exactly one concept c in g.

- Three kinds of conceptual graphs have distinguished names:
    1. The *blank* is an empty conceptual graph with no concepts, conceptual relations, or arcs.
    2. A *singleton* is a conceptual graph that consists of a single concept, but no conceptual relations or arcs.
    3. A *star* is a conceptual graph that consists of a single conceptual relation r, every arc that belongs to r, and every concept c that is linked by some arc (r, c) that belongs to r.

Conceptual graphs (CG) were developed for the representation of natural language semantics and are designed for communication with humans or between humans and machines. Sowa (2000) contends that CGs *... can help form a bridge between computer languages and the natural languages that everyone reads, writes, and speaks.* The conceptual graph is generally in a visual format and is represented in machine-readable text format using CGIF (Conceptual Graph Interchange Format) syntax. The CGIF syntax is specified in terms of the Extended BNF (EBNF) rules and metalevel conventions. Chein and Mugnier (1992, 1995) expound that CGs are not only graphical representations of knowledge but also, based firmly on labelled graph theory, enable the use of combinatorial algorithms for operations and manipulation. Corbett (2004) examines techniques for comparing and filtering CGs to allow interoperability of ontologies for knowledge interchange.

2.15 Knowledge Interchange

Common Logic (CL) is currently being proposed as a new language standard for knowledge interchange. It aims in providing a superset for other interchange formats, such as KIF (Knowledge Interchange Format), CGIF, CLML (Common Logic Markup Language), and DL (Description Logics) so that content exchanged between Common Logic conformant languages has the same semantics in each language. World Wide Web Consortium (W3C) is developing standards for two logic-based content languages for the 'semantic web': RDF (Resource Definition Facility), a language for expressing relationships and OWL (Web Ontology Language), a language for expressing constraints. People involved in the W3C project are also involved in CL development ensuring that the Common Logic semantics are inherited by RDF and OWL. An RDF graph, for example, is represented in an N-triple format which translated readily to a relation in a relational database.

2.16 Summary

Database evolution is inevitable. An RDBMS, to remain useful and informative, must change to adapt to new database technology, changes in the UoD, new interpretations of data, new or changed user requirements and so forth.

The goals of good database evolution are to preserve the integrity of the data and minimise the impact of modifications on views, queries and processes. These goals require many different methods and techniques to reach them and have motivated work in various areas of related research as can be seen in Figure 2.4. There are techniques for relation evolution and attribute-relation assignment evolution which include schema translation, transformation and matching. Mediation techniques provide ways to integrate heterogeneous data for queries and views.

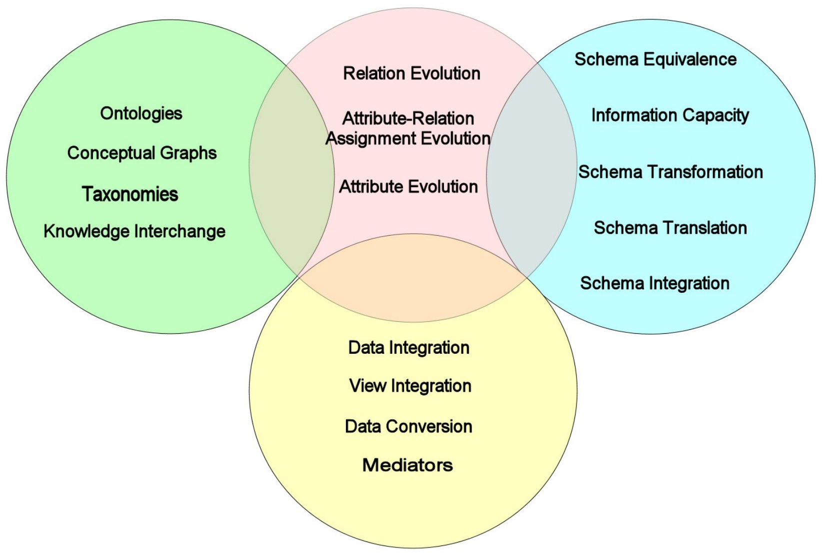


Figure 2.4. Database Evolution and Related Research Areas

There are however few techniques to manage schematic changes that affect the *data* that are in the source database(s). These changes are summarised in Table 2.3. Of particular interest to this research are attribute changes: expanding or restricting the attribute specification, as well as changing the data type. The research on ontologies and conceptual graphs has also highlighted that an attribute's valid values, its domain, can change over time without schema changes but requiring application modification.

Ontology structures reflect the complexity of and relationships within a concept domain. It is envisaged that the techniques used for knowledge interchange

Table 2.3. Schematic Changes that Affect Data

Evolutionary Operation	**Effect on Data**	**Effect on Application**	**Comment**
Expand an attribute specification	Existing values whilst being valid may no longer be accurate.	Application may change to manage new constraints	Expansion changes the precision of values stored. E.g. 42.00 implies a precision that the integer 42 does not.
Restrict an attribute specification	Values outside the range of valid values are lost.	Application may change to manage new constraints	Values outside the range must be converted or deleted. Existing values may be abbreviated, truncated or rounded. These values are no longer an exact copy of the original.
Change an attribute data type	Data loss can occur where there is no 1:1 mapping of old to new values.	Application may change to manage new data type.	The semantics have changed, manual verification may be required.
Delete an attribute	Existing values are lost	References to the attribute must be removed.	
Add an attribute		Application is changed to refer to new attribute	Data may not be complete or correct if the attribute is populated with a default value.
Delete a relation	Existing values are lost	References to the relation must be removed.	
Cardinality Change	If the cardinality changes from m:n to 1:n, there is data loss.	Application may change to manage the relationship change.	This may also be a change in semantics.

are also applicable to relational database development as well. Currently ontologies are viewed as resources that are positioned *outside* a user's information system and are used as an aid for the design of metadata, for translation of concepts or for transformation of related concepts. However, ontologies and concept graphs could be an integral part of an RDBMS. This thesis argues a three-level architecture for relational databases with an interface positioned between data and metadata for complex domains. This intermediary level is the *mesodata* layer.

Chapter 3

Mesodata in DBMS

This chapter presents the reasoning for a three level architecture for relational databases, introducing a layer between metadata and data - *mesodata* - to accommodate more complex domain definitions.[§]

3.1 Modelling

The first steps recommended for database modelling usually advise the modeller to analyse the Universe of Discourse (UoD) to determine significant features and their relationships so that the data model represents the 'real world'. This requirements analysis should elicit the data requirements in terms of primitive objects (entities or attributes), the classification and description of information about the objects, the identification and classification of the relationships between the objects, the rules governing the integrity of the data and the types of transactions that will be processed, as well as the interactions between the transactions and the data.

The data model usually has two components. A diagram which represents the data structures in a pictorial form, e.g. an entity-relationship diagram, and a data document. The data document describes in detail the data objects, relationships and rules required by the database, with a dictionary that provides the detail required to construct the physical database. In the physical model, the attributes are then formatted to a selected data type from the data formats provided by the DBMS platform.

[§]The contents of this chapter are an extended version of de Vries & Roddick (2004).

Relational database platforms offer a limited number of data formats for simple structures that can be created and manipulated with Structured Query Language (SQL) using the subset languages of Data Definition Language (DDL) and Data Manipulation Language (DML). Each database platform provides 'core' features of the SQL standard plus other SQL and proprietary features resulting in different 'flavours' of SQL. Even the data types, whilst sharing the same labels, do not necessarily share the same byte representation resulting in differences in valid ranges of values and storage requirements. For example, the data type FLOAT may be either a 4-byte floating point number, with a range of -3.402823E38 to -1.401298E-45 (negative) and 1.401298E-45 to 3.402823E38 (positive), or an 8-byte floating point number, with a range of -1.79769313486231E308 to -4.94065645841247E-324 (negative) and 4.94065645841247E-324 to 1.79769313486231E308 (positive). Refer to Table F.1 for a comparison of data types on different DBMS platforms.

The modeller must, therefore, be aware of both the attribute domain[1] values to be modelled as well as the range of values the data type can store. Generally this modelling method has been used to great effect for the past three decades. However, there are some obvious problems that arise as a consequence which include:

- The model is based upon a *snapshot* of the world as it is and is not always flexible enough to manage changes in the UoD.
- The semantics of the UoD are stored separately from the metadata in the data document.
- The modeller and the end-user do not know what future processing and reporting requirements may be. Some current features in the UoD that are not deemed significant enough to be incorporated into the model are later realised to be significant.
- The data types provided are simple data types that do not capture the *structure* of the attribute domain.

Stonebraker's matrix for classifying DBMS, Figure 3.1, classes RDBMS with SQL for simple data with queries and ORDBMS with SQL and user defined

[1]Confusingly, the term *domain* can refer to the Universe of Discourse (UoD), for which the database system was created, the range of valid values for an attribute within the database, or a data type. In this work a **domain** is defined as the range of valid values that a specific attribute may store.

functions for complex data with queries, however neither of these approaches captures the structure of the data domains nor the relationships between the domain values themselves.

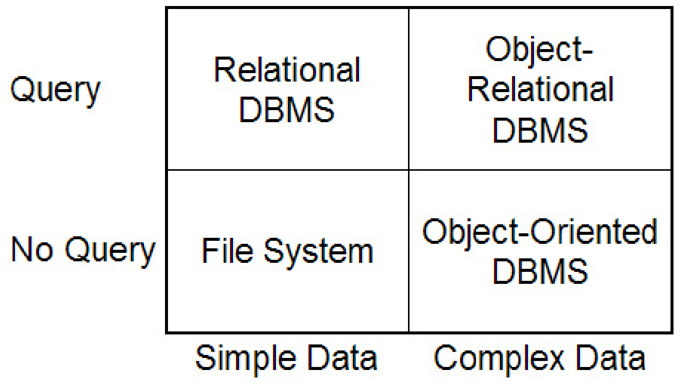


Figure 3.1. A Matrix for Classifying DBMS
(Stonebraker & Moore 1996)

Currently, when defining an attribute within a schema, elementary types, such as integer, string and so on, are available together with operators for comparison and manipulation. For example, it is not necessary to create, within the application, code for arithmetic operations on numeric data types. These are known and available on declaration of the type. Such types are supplied as they are both common and useful. However, there are many other instances of domain structures, such as graphs (Roddick et al. 2003), hierarchies (Rice & Roddick 2000) and intervals (Allen 1983), for which standardised and supplied definitions would be a practical enhancement, particularly to facilitate schema and domain evolution processes, and data integration (Rice, Roddick & de Vries 2006).

The usual practice when using, for instance, graphs is to include code in the application for depth-first or breadth-first traversal, topological ordering, finding the shortest path between selected nodes, calculating nearest and furthest nodes, and so on. This thesis proposes that those operations which relate to the data type *domain*, rather than the data *value* or the *application*, should reside closer to the structure of the database than to the population values of the database. Early in this project (Roddick et al. 2003), it was envisaged that complex types such as graphs and trees could reside alongside other relations with their metadata and data. However, it is difficult to position them into the metadata. It became apparent when developing manipulation techniques and operators that their 'true place' was in a level of their own, as domain values are neither data instances nor metadata. This thesis therefore contends that there should be a *mesodata* layer between the metadata and the data, as shown in Figure 3.2. The mesodata holds the 'intelligence' for the data type and provides the link between the base data type, the domain structure with its valid range of values, and the database application.

3.2 Mesodata

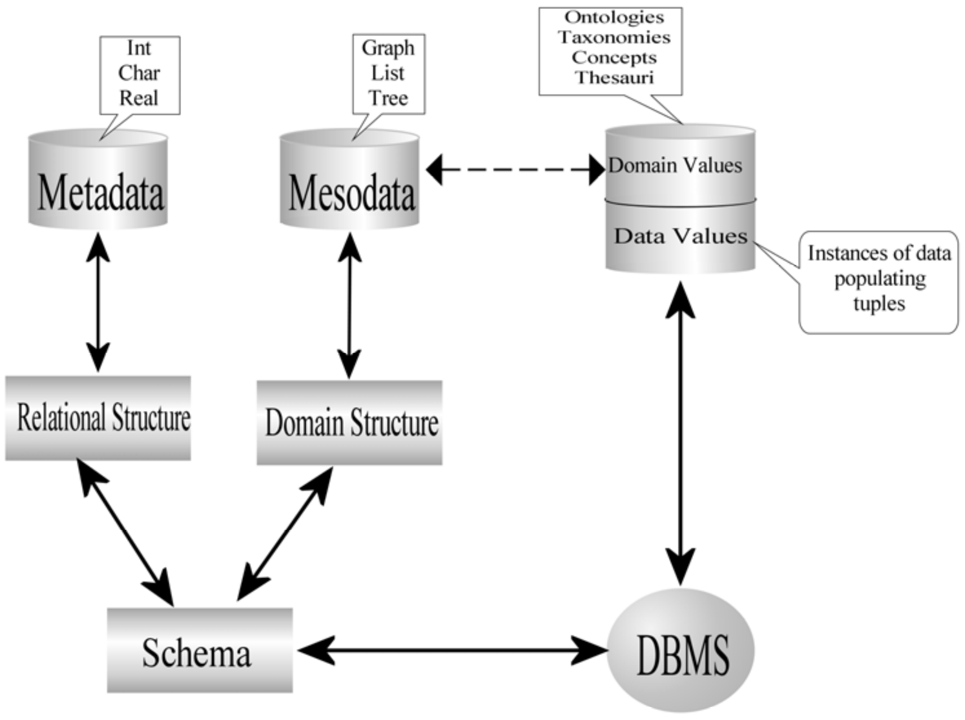


Figure 3.2. Mesodata Layer Between Metadata and Data

An important distinction must be made between the provision of, say, a graph as a mesodata type and the provision of a graph as a user-defined abstract data type (ADT). In the former, an attribute in a relation would take as its value an instance of an elementary type that exists *within a graph* whereas in the latter the value of the attribute exists *in the relation*. The presence or absence of the mesodata graph does not have any effect on the relation, standard SQL nor existing application code. Additionally, unlike the provision of libraries of user defined ADTs, the graph itself is not directly accessible or manipulable through the attribute.

A partial hierarchy of mesodata types for different domain structures is given in Figure 3.3. However, any candidate mesodata type would be a commonly used structure and have generally agreed and stable semantics.

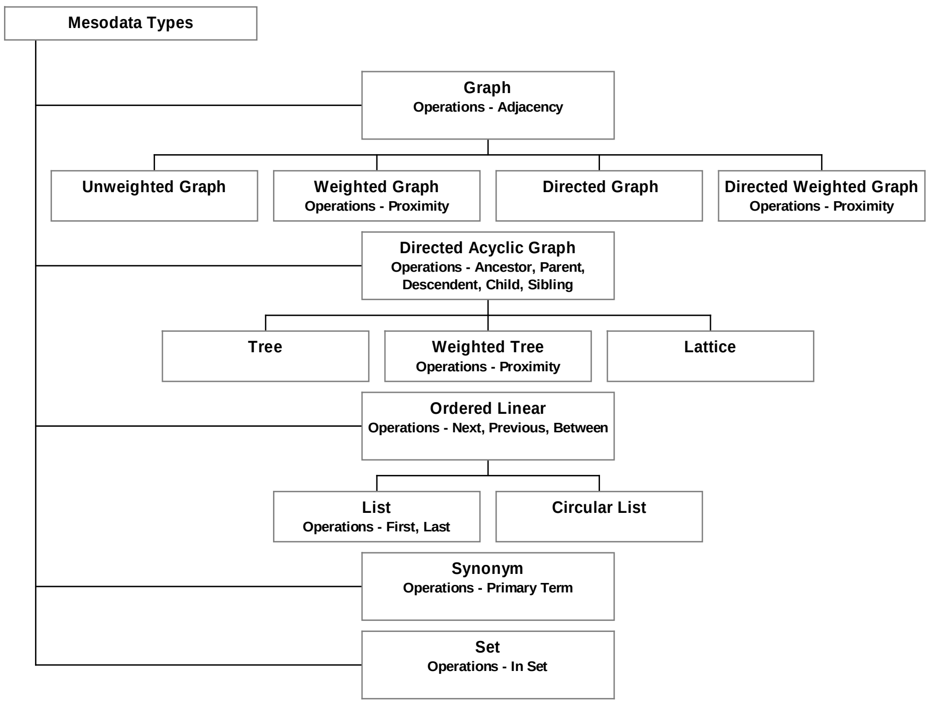


Figure 3.3: Hierarchy of Some Suggested Mesodata Types for Different Domain Structures

The semantics of information held in a database can be considered as a mapping function of the data value. That is,

$$S \leftrightarrow F(v) \tag{3.1}$$

where F is a mapping external to the database which maps the data value (eg. 1) to the understood concept (eg. Monday). The introduction of a mesodata layer allows regularly used mappings to be accommodated in the database, ie.

$$S \leftrightarrow F(M_1(M_2(\ldots M_k(v)\ldots))) \tag{3.2}$$

where M_i are mesodata layer mappings.

3.3 Mesodata Domains

Formally, a mesodata domain D is a set of identically typed instances $\{v_1, \ldots, v_j\}$ taken from either a simple domain or another mesodata domain such that $M_i(v_1, \ldots, v_j)$

is a mapping that provides a relationship based on the semantics of the structure between the instances. An instance d of D is either one of the enumerated values $\{v_1, \ldots, v_j\}$ or, if permitted, another value of the same type. Implicit in this is that a domain defined over a mesodata type, as for base types such as CHAR, INT etc., is:

- a DBMS-supplied structure consisting of a uniformly agreed data structure consisting of either other mesodata types or elementary data types.
- a set of common and uniformly agreed operators over the structure.

An attribute defined over such a domain would take an instance of an elementary value which would exist within the mesodata structure.

3.3.1 Definition of the Mesodata Domain

The mesodata layer extracts the domain to a separate level such that the Mesodata Domain (Mdom) is the domain of the mesodata type of the base type, for example a weighted graph of strings or a list of graphs of strings. Mdom is defined as:

```
Mdom :: dom(attribute) | dom(mesodata)
dom(mesodata) :: dom(mesodatatype(dom(mesodata)) |
                 dom(mesodatatype(dom(attribute))
dom(mesodatatype) :: dom(wgraph)|dom(wdgraph)|
                     dom(list)|dom(clist)|...
                     any mesodata structure
dom(attribute) :: dom(basetype)
basetype ::  all valid database base types.
```

A traditional relational database can be viewed as consisting of relations that are a subset of the Cartesian product of their attributes' domains (Elmasri & Navathe 2004).

$$R \subseteq \{dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)\} \tag{3.3}$$

where R is the relation
A is an attribute
dom is the domain of the attribute A.

Following Maier's (1983) definition, a relation scheme R is a finite set of attribute names $\{A_1, A_2, \ldots, A_n\}$. Corresponding to each attribute name A_i is a set D_i, $1 \leq i \leq n$, called the domain of A_i, also denoted as $dom(A_i)$. Let $D = D_1 \cup D_2 \cup \ldots \cup D_n$. A relation r on relation scheme R is a finite set of mappings $\{t_1, t_2, \ldots, t_n\}$ from R to D with the restriction that for each mapping $t \in r$, $t(A_i)$ must be in D_i, $1 \leq i \leq n$.

The domain of an attribute A_i defined over a mesodata domain $M_j\{v_1, v_2, \ldots, v_n\}$ retains the mapping from R to D and adds a mapping via a function f from A_i to M_j such that

$$\begin{aligned}
&f(A_i) \rightarrow M_j \\
&\forall a \in dom(A_i)\ : a \in F_i(M_j) \\
&dom(A_i) \subseteq F_i(M_j) \\
&t(A_i)\ \in\ F_i(M_j) \\
&\text{Let } Mdom(A_i) = F_i(M_{A_i})
\end{aligned}$$

The redefinition of the relation R thus becomes

$$R \subseteq \{Mdom(A_1) \times Mdom(A_2) \times \ldots \times Mdom(A_n)\} \tag{3.4}$$

that is, the cartesian product of the mesodata defined domains.

3.3.2 Extended Querying

The mesodata domain extends the SQL WHERE definition to include operators specific to the domain structure. These mesodata operators return a set of values $\{v_1, v_2, \ldots, v_n\}$.

Let $\{a, b, c, d \mid R(a, b, c, d)\}$
With currently available comparison operators represented by Θ
`Select * FROM R WHERE` $a\ \Theta\ value$ is
$\{a, b, c, d \mid (\exists a, b, c, d)$
$(R(a, b, c, d) \wedge$
$a\ \Theta\ value)\}$
Let the domain of a be defined over mesodata domain $M(v_1, \ldots, v_n)$ with operators $\bigotimes$.
`Select * FROM R WHERE` $a \bigotimes value$ becomes
$\{a, b, c, d \mid (\exists a, b, c, d)$

$(R(a, b, c, d) \wedge$
a IN $\{value \bigotimes M\})\}$

As this is an extension of SQL, also valid is

`Select * FROM R WHERE` $a \bigotimes value_1$ `AND` $b \ \Theta \ value_2$

$\{a, b, c, d \ | \ (\exists a, \exists b, c, d)$
$(R(a, b, c, d) \wedge$
a IN $\{value_1 \bigotimes M\} \wedge$
$b \ \Theta \ value_2)\}$

3.4 Structured Domains

Consequent to the structure and operations over that structure, there is *orderliness* in mesodata domains. Often, within the data, there is an assumed order, either numeric, alphabetic, spatial or temporal. Ng (1998, 2001) also describes user declared semantic ordering in domains. Mesodata types such as graphs, store relationships and paths between the nodes providing the metrics for adjacency and proximity, whereas mesodata types such as lists and trees, are by their structure *ordered*. Thus these mesodata domains are, at minimum, partially ordered sets.

A partial order on a set S is a relation $\sqsubseteq$ with the following properties: (Abramsky & Jung 1994)

1. $\sqsubseteq$ is reflexive : $x \sqsubseteq x$ for all $x \in S$.
2. $\sqsubseteq$ is transitive : $(x \sqsubseteq y$ and $y \sqsubseteq z)$ implies $x \sqsubseteq z$ for all $x, y, z \in S$.
3. $\sqsubseteq$ is antisymmetric : $(x \sqsubseteq y$ and $y \sqsubseteq x)$ implies $x = y$ for all $x, y \in S$

Thus, for $(M, \sqsubseteq)$ and element $m \ni M$,
$\downarrow m$ is the set of all elements of M that are below m, and
$\uparrow m$ is the set of elements of M that are above m.

$$a \in \downarrow m \iff a \sqsubseteq m$$

$$a \in \uparrow m \iff a \sqsupseteq m$$

With these properties, a mesodata type's operators are employed to return subsets of the domain, that is the operators create *filters* for a domain.

3.4.1 Filters

A filter on a set S is a collection F of subsets of S where (Moshier 2000)

1. F is directed with respect to $\supseteq$
   i.e. if $X, Y \in F$, then there is some $Z \in F$ so that $X \supseteq Z$ and $Y \supseteq Z$

2. F is downward closed with respect to $\supseteq$
   i.e. if $X \in F$ and $Y \supseteq X$ (and $Y \subseteq S$) then $Y \in F$

therefore $X, Y \in F$ implies $X \cap Y \in F$

If $D = (D, \sqsubseteq)$ then $\sqsubseteq$ determines the filter $F \sqsubseteq$, there is an element d in D where the set of all elements above d exists in X.

$$X \in F \sqsubseteq \iff \exists d \in D,\ \uparrow d \subseteq X$$

If $\uparrow x \subseteq X$ and $\uparrow y \subseteq Y$ then there exists z so that $\uparrow z \ \subseteq \uparrow x \cap \uparrow y \subseteq X \cap Y$.

A filter when applied to an ordered mesodata type returns a set of domain values, for example, with a mesodata domain structure of TREE, the operator `DESCENDENT` returns the set of values in the subtree of a specified node.

3.4.2 Topological Spaces

Data structures such as graphs do not appear, initially, to possess the properties needed to implement filters as defined above. These structures are, however, built with operators for *adjacency* and *proximity.* Using these operators, a metric space topology can be generated, thus providing the order required. The following definitions are from Munkres (1975).

Definition: A topology on a set X is a collection τ of subsets of X having the following properties:

1. $\emptyset$ and X are in τ.

2. The union of the elements of any sub-collection of τ is in τ.

3. The intersection of the elements of any finite sub-collection of τ is in τ.

A set X for which a topology τ has been specified is called a topological space

Definition: If X is a set, a **basis** for a topology on X is a collection $\mathcal{B}$ of subsets of X (called **basis elements**) such that

1. For each $x \in X$, there is at least one basis element B containing x.
2. If x belongs to the intersection of two basis elements B_1 and B_2, then there is a basis element B_3 containing x such that $B_3 \subset B_1 \cap B_2$.

A metric space topology is generated by sets of the form

$$V_\varepsilon(p) = (x \in X \mid d(x,p) < \varepsilon) \; for\ some\ \varepsilon > 0$$

where d is a metric having the following properties:

1. $d(x,y) \geq 0$ for all $x, y \in X : d(x,y) = 0$ iff $x = y$
2. $d(x,y) = d(y,x)$ for all $x, y \in X$
3. $d(x,y) + d(y,z) \geq d(x,z)$, for all $x, y, z \in X$ (triangular inequality)

Let $\mathcal{N}(p) = \{V_\varepsilon(p) : \varepsilon > 0\}$ then

1. $p \in V_\varepsilon(p)\ \forall \varepsilon$
2. the elements of $\mathcal{N}(p)$ are directed with respect to $\supseteq$ and can therefore form the basis for a filter. See Figure 3.4

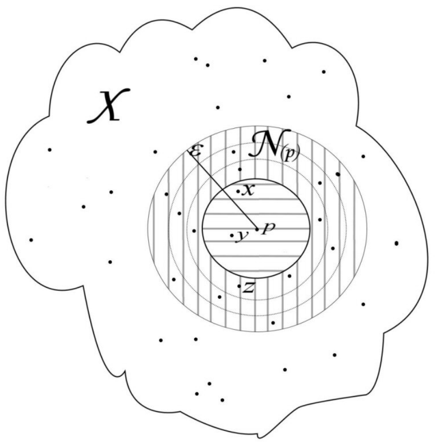


Figure 3.4. A Filter in Metric Space

For example, with mesodata domain structure of WGRAPH, the operator CLOSETO returns a set of values within a defined threshhold value (ε) of a specified node.

Table 3.1. Partial List of Mesodata Types with Extended SQL Operators

Domain Structure	Mesodata Type	Operations (Extended SQL Op.)	Source Relation(s)
Unweighted Graph	GRAPH	Adjacency (NEXTTO)	Binary relation (FROM, TO)
Weighted Graph	WGRAPH	Adjacency, Proximity (CLOSETO) (EQUALTO)	Ternary relation (FROM, TO, WEIGHT)
Directed Graph	DGRAPH	Adjacency	Binary relation (FROM, TO)
Directed Weighted Graph	DWGRAPH	Adjacency, Proximity	Ternary relation (FROM, TO, WEIGHT)
Tree	TREE	InSubtree(DESCENDENT), Parent(PARENT), Ancestor(ANCESTOR), Child(CHILD), Sibling(SIBLING)	Binary Relation(PARENT, CHILD)
Weighted Tree	WTREE	In subtree, Parent, Ancestor, Sibling, Proximity	Ternary Relation(PARENT, CHILD, WEIGHT)
List	LIST	Next (NEXT), Previous (PREV), First(FIRST), Last(LAST), Between(INBETWEEN)	Binary Relation(SEQUENCE, ITEM)
Circular List	CLIST	Next, Previous, Between	Binary Relation(SEQUENCE, ITEM)
Set	SET	In Set(INSET)	Unary Relation(ITEM)
Synonym	SYNONYM	Primary Term (SYNONYM)	n-ary (n>1) Relation(PRIMARY, SECONDARY, ..., nARY)

3.5 Mesodata Operators

A partial list of mesodata types in Table 3.1 presents new operators that can be performed over different domain structures. These operations extend the currently available SQL operators and enhance querying power by providing not only the structure of the domain but also the 'intelligence' with which to manipulate the specific type of structure. Note also that mesodata types are populated by values that are stored in n-ary relations and therefore remain mathematically equivalent to a multi-set - the building block of a relational database.

3.6 Comparison of Mesodata with User-Defined Types

A user-defined type (UDT) is a named data type that is created in a database by the user. A UDT can be a distinct type which shares a common representation with a built-in data type or a structured type which has a sequence of named attributes each of which have a type. A structured type can be a subtype of another structured type thus defining a type hierarchy. The values of a specific UDT are considered to be compatible only with values of the same UDT or UDTs in the same type hierarchy.

A UDT with its methods and domain is consistent throughout a database, it affects ALL columns that are specified as that type. A mesodata domain can be referenced by any number (including zero) of columns without it altering any other column or data type. A column may have its mesodata domain changed without altering the column's data. The mesodata domain's values may be updated without affecting a column's data as mesodata are stored in a separate layer in the system.

A mesodata structure can be based on any base data type within the DBMS. The mesodata layer represents the structure of the domain whereas the UDT represents structure of the data value. Table 3.2 presents comparisons between the properties and uses of UDTs and mesodata types.

Table 3.2: Comparison of User Defined Types (UDT) and Mesodata Types

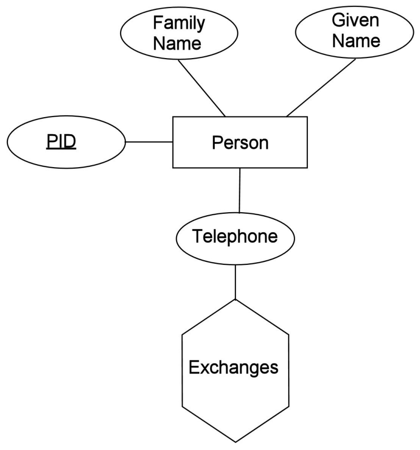
	UDT	Mesodata Type	Comments
Unique Name	✓	✓	Must have a unique name within the schema
Reusable	✓	✓	Can be re-used within the database and in other databases
In-built operators	✗	✓	UDTs require user defined methods to be written for each type. Mesodata types have built in operators with SQL commands.
Alter affects column spec	✓	✗	UDT changes are inherited by all columns specified by the type. Mesodata type changes are independent of column specification.
Drop affects column spec	✓	✗	UDT deletions affect all columns specified by the type. Mesodata type deletions are independent of column specifications
Stores data instance	✓	✗	Data values are stored in the table according to the rules and constraints of the UDT. Mesodata types store domain data in a separate layer from the data layer.
Uses base types	✓	✓	UDTs are constructed from base types. Mesodata types are complex structures in which components are base types, eg a graph in which the nodes are character strings.
Uses UDTs	✓	✗	UDTs may be based on other UDTs. Mesodata types are only constructed from base types or from other Mesodata types.
Uses mesodata types	✗	✓	UDTs are defined on types that are stored in metadata,Mesodata are in a separate layer. Mesodata can be constructed from other Mesodata types eg. A tree of graphs of character strings.
			continued next page

	UDT	Mesodata Type	Comments
Allows complex data storage	✓	✗	UDTs enable storage of complex data within the database. Mesodata does not.
Capture domain structure	✗	✓	UDTs capture complex **data** structures. Mesodata types capture complex **domain** structures.
Must exist before column spec	✓	✗	A UDT must exist before a column is specified. Mesodata types are independent of column specifications and can be created after a table is created and populated.
Backwards compatible	✗	✓	As Mesodata are separate from the relational schema, the addition of a Mesodata layer is compatible with currently existing schemata. UDTs reside in the relational schema and thus require modifications to the existing schemata before they can be utilised.

3.7 Conceptual Model Incorporating Mesodata

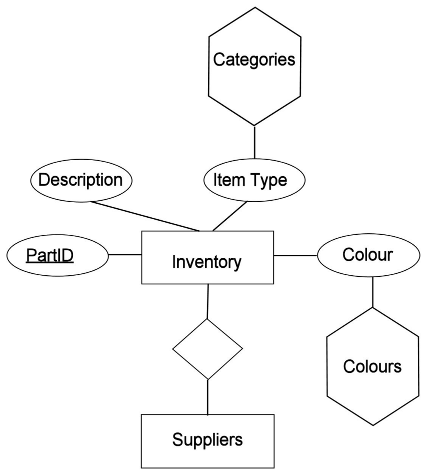
When modelling a database, entity-relationship (E-R) diagrams are often used to capture the semantic information about the 'real world'. Chen's (1976) widely used symbols for ER modelling were intentionally designed to omit information that was not considered relevant *A complete description of an entity or relationship may not be recorded in the database of an enterprise. It is impossible (and, perhaps, unnecessary) to record every potentially available piece of information about entities and relationships. From now on, we shall consider only the entities and relationships (and the information concerning them) which are to enter into the design of a database.*

This thesis argues that attribute *domain* information should be included in the modelling process. In order to represent, at the conceptual level, the inclusion of mesodata in the ER diagram, the hexagon is suggested as the symbol for the domain and labelled with the domain name, as in the examples shown in Figures 3.5 to 3.9.



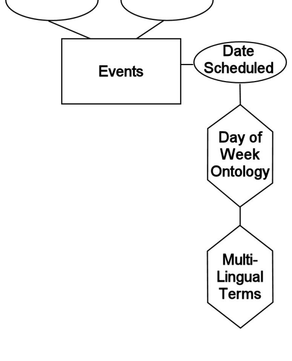
The hexagon represents an attribute referencing a mesodata domain. In this example the attribute **Telephone** is defined over the mesodata domain **Exchanges**

Figure 3.5. ERD An Attribute Referencing a Mesodata Domain



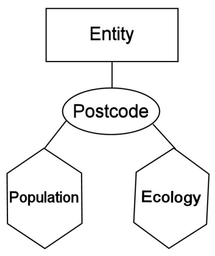
The hexagons represent the domains for the attributes **Item Type** and **Colour**, the mesodata domains are **Categories** and **Colours**. Any number of attributes, can reference mesodata domains.

Figure 3.6. ERD Multiple Attributes Referencing Mesodata Domains



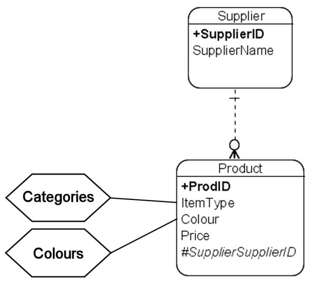
In the 'Days of the Week' example (Figure 5.5 page 75), two domains are used together. The joining of the domains, **Day of Week Ontology** and **Multilingual Terms**, is represented by 'stacking' the hexagons and connecting the attribute to the mesodata domains with a single connector line.

Figure 3.7: ERD An Attribute Referencing Joined Mesodata Domains



A single attribute can reference multiple mesodata domains in order to utilise different contexts for querying. In this example, the domains of **Population** and **Ecology** are referenced by the attribute **Postcode**. Each domain symbol is joined to the attribute with a separate connector line.

Figure 3.8: ERD An Attribute Referencing Multiple Mesodata Domains



The hexagon may also be rotated, as well as used in ER diagrams using different notations, as in this UML ER diagram.

Figure 3.9. ERD UML Notation

A complete set of modelling tools for mesodata components is not within the scope of this research. The entity-relationship diagrams are a suggestion for their inclusion in the conceptual modelling of a database.

3.8 Summary

Traditionally RDBs are modelled to capture selected features in a snapshot of the UoD. The model stores the form and layout in the metadata separately from the full description in the data document. There are only a few simple base types from which to choose and though UDTs, in object relational platforms, model complex data there is no platform supplied provision for complex *domains.*

A mesodata layer, separate from the metadata and data, provides complex structures in which to store domain values and their inter-relationships as well as supplying the 'intelligence' required to operate and manipulate them. Mesodata types are populated by values that are stored in n-ary relations and therefore remain mathematically equivalent to a multi-set, the foundation of a relational database. The domain structures enable different orderings that form the bases of filters for enhanced querying and information retrieval. DBMS supplied mesodata types would allow for the re-usable inclusion of domain information such as in ontologies, taxonomies and concept graphs that to date have been only application specific.

Chapter 4

Reference Data Language

Chapter 3 presented the conceptual model and underlying rationale for the mesodata layer in RDBMS. This chapter presents the development of the Mesodata Definition Language and Query Language extensions to SQL necessary to implement the integration of complex domains in the relational database architecture.

4.1 Aims

In order to incorporate mesodata types into an RDBMS, the main goals and features of the new architecture were identified as:

- A separate layer to store domain values.
- Platform supplied complex structures.
- Intelligence built into these structures by way of comparison and manipulation operators.
- The domain layer to be backwards compatible with existing systems.
- The domains to be reusable within the database.

During the conceptual modelling phase of this research, various data structures were identified as being both useful and common for data storage and processing. These structures as listed in Table 3.1 in the previous chapter have been well researched, with known algorithms for their creation and traversal. To date, however, each use of such a complex structure has been specifically written for a database and, in general, not reused. The problem faced was not the design of the domain structures but, rather, how to have these positioned in a database so that they are as straight-forward to use as other platform supplied base types.

The realisation that a mesodata layer was required, formed the foundation for definition commands that are necessary to create, alter and drop domains, in addition to those that are needed to connect attributes with the domains and enable querying over them.

4.2 Mesodata Definition Language

4.2.1 Create Domain Syntax

The `CREATE DOMAIN` command, present in SQL2 and later versions, defines a domain in a schema and is identified by a <domain name>. *The purpose of a domain is to constrain the set of valid values that can be stored in a column of a base table by various operations. A domain definition specifies a data type. It may also specify a <domain constraint> that further restricts the valid values of the domain and a <default clause> that specifies the value to be used in the absence of an explicitly specified value or column default* (ISO/ANSI 2003). However, Melton (2002) comments that '.. *they have proved to be less useful than originally hoped, and future editions of the SQL standard may actually delete the facility entirely*'.

As the characteristics of a mesodata domain are in keeping with those defined in the SQL standard, and the existing command is rarely used, existing domain commands have been redefined (and extended) to specify also the domain structure - the <mesodatatype>.

```
CREATE DOMAIN <domain_name>
       AS <mesodatatype>
       OF <basetype>
       OVER <relation_name>
or
CREATE DOMAIN <domain_name>
       AS <mesodatatype>
       DOM <existing_domain_name>
mesodatatype:
GRAPH or WGRAPH or DGRAPH or DWGRAPH  or TREE
```

```
or WTREE or LIST or CLIST or SET or  SYNONYM
basetype:
any valid basetype in database platform
relation_name:
existing table name populated with the domain values
```

`CREATE DOMAIN` creates a structure in the mesodata layer populated with the values stored in the specified `relation name.`The domain can be referenced by any relation's attribute that has the same `basetype` definition. The domain name, its basetype and source relation are stored in a system file in the mesodata layer so that relational table attributes can be defined using the domain name. If the specified source relation's structure is incompatible with the mesodata type, the domain is not created and an error message is displayed.

Example:

```
CREATE DOMAIN COLOURS AS WGRAPH OF CHAR(30) OVER tblCOLOURS;
```

4.2.2 Drop Domain Syntax

```
DROP DOMAIN <domain_name>
```

`DROP DOMAIN` deletes the domain structure and its contents from the mesodata layer. An existing reference between a table's attribute and the mesodata domain prevents the deletion of the domain and an error message is displayed.

4.2.3 Alter Domain Syntax

```
ALTER DOMAIN <domain_name> <alter_specification>
alter_specification:
RENAME <new domain_name>
CHANGE MESODATATYPE <old mesodatatype> <new mesodatatype>
CHANGE BASETYPE <old basetype> <new basetype>
CHANGE OVER <old relation_name> <new relation_name>
```

Example:

```
ALTER DOMAIN COLOURS RENAME COLOR;
ALTER DOMAIN COLOURS CHANGE MESODATATYPE WGRAPH WTREE;
ALTER DOMAIN COLOURS CHANGE BASETYPE CHAR(30) CHAR(40);
ALTER DOMAIN COLOURS CHANGE OVER tblCOLOURS tblCOLCHART;
```

`ALTER DOMAIN` allows changes to an existing mesodata domain.

`RENAME` changes all entries referring to the previous domain name to the newly specified name without deleting domain values or references to the domain.

`CHANGE MESODATATYPE` and `CHANGE BASETYPE` erase the current values from the mesodata domain then re-populate it from the source relation. An existing reference between a table's attribute and the mesodata domain prevents these changes to the domain and an error message is displayed.

`CHANGE OVER` deletes the domain's current values and re-populates the domain with values from the specified new source relation and updates the system file in the mesodata layer with the relation name. If the specified new source relation's structure is incompatible with the mesodata type, the domain's values are not changed and an error message is displayed.

4.2.4 Refresh Domain Syntax

```
REFRESH DOMAIN <domain_name>
```

`REFRESH DOMAIN` re-populates the mesodata structure with values from the relation specified when it was created (or altered). If the specified source relation's structure is incompatible with the mesodata type, the system file and the domain's values are not changed and an error message is displayed.

4.2.5 Describe Domain Syntax

```
DESC[RIBE] DOMAIN <domain_name>
```

`DESCRIBE DOMAIN` displays the structure, its basetype and source relation of the specified domain.

4.2.6 Show Domains Syntax

```
SHOW DOMAINS
```

`SHOW DOMAINS` displays all current domains in the mesodata system file with details of the domain names with the tables and attributes referencing them.

4.3 Mesodata Extended SQL

In order to utilise mesodata types with existing relational databases, command phrases were developed that extend current SQL commands. These extensions allow attribute specifications to refer to the mesodata domains for the create, alter and drop operations of schema definition. Thus, all valid SQL statements can be used with the following extensions.

4.3.1 Create Table Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
                   table_name [(create_definition,...)]

create_definition:
column_name type [NOT NULL | NULL] [DEFAULT default_value]
type:
any platform provided type (length)
or existing Domain name
```

Example:

```
CREATE TABLE tblPRODUCTS (
ProdID int(5) NOT NULL,
Description char(20),
Colour COLOURS,
PRIMARY KEY (ProdID)
);
```

`CREATE TABLE` with an attribute specified by an existing domain name, creates the attribute in the table with the same basetype specification as the domain. A system file, in the mesodata layer, records the domain name, table name and attribute. Executing a `DESCRIBE TABLE` will display the attribute's (inherited) basetype specification.

4.3.2 Alter Table Syntax

```
ALTER [IGNORE] TABLE table_name alter_spec [, alter_spec ...]
alter_specification:
ADD COLUMN (create_definition, create_definition,...)
```

```
    CHANGE COLUMN old_column_name (create_definition)
    MODIFY COLUMN column_name (create_definition)
    DROP COLUMN  column_name
create_definition:
same as for CREATE TABLE
```

`ALTER TABLE` statements referring to a column that is, or has been, defined with an existing domain name will update both the metadata for the table and the system table in the mesodata layer.

Examples:

```
ALTER TABLE tblPRODUCTS
            ADD COLUMN(Category CATEGORIES);
ALTER TABLE tblPRODUCTS
            CHANGE COLUMN Colour (ProdColour CHAR(30));
ALTER TABLE tblPRODUCTS
            MODIFY COLUMN ProdColour (Colour COLOURS);
ALTER TABLE tblPRODUCTS DROP COLUMN Colour;
```

4.3.3 Drop Table Syntax

```
DROP TABLE table_name
```

A `DROP TABLE` command updates the system files in the mesodata layer if any of the table's attributes were specified by a mesodata domain. A source relation for a mesodata domain may be dropped, after its creation, as the values are now stored in the mesodata layer.

4.3.4 Describe Mesodata Type Syntax

```
DESC[RIBE] MESODATATYPE mesodatatype_name
```

`DESCRIBE MESODATATYPE` displays the metadata required for the source relation of the mesodata domain values.

4.3.5 Show Mesodata Types Syntax

```
SHOW MESODATATYPES
```

`SHOW MESODATATYPES` accesses the system files in the mesodata layer to display the attribute names, and the relations to which they belong, that reference mesodata domains.

4.4 Extensions to Manipulation Language

The addition of mesodata domains enables querying over data instances present in the relations as well as over domain values in the mesodata layer. The `SELECT` statement retains the standard SQL syntax with new operators for the `WHERE search-condition` .

4.4.1 Select Syntax

```
SELECT [query-specification] select-list
   FROM table-reference-list
   [WHERE search-condition]
   [GROUP BY column-name [, column-name]... ]
   [HAVING search-condition]
   [[UNION | UNION ALL |INTERSECT | MINUS]
   select-statement]...
   [ORDER BY {unsigned integer | column-name}
   [ASC|DESC]]
```

The new comparison operators are built into the domain structures and, consequently, are dependent upon the features of that structure. These operations return a set of domain values that are then processed as `WHERE` *value* `IN` $\{v_1, v_2, \ldots, v_n\}$.

`CLOSETO:` `column-name CLOSETO` *value*
Domain Structure: Weighted graph, Weighted tree, Directed weighted graph.

`FAR:` `column-name FAR` *value*
Domain Structure: Weighted graph, Weighted tree, Directed weighted graph.

`THRESHOLD:`	`column-name CLOSETO` *value* `THRESHOLD` *distance* `column-name FAR` *value* `THRESHOLD` *distance* Domain Structure: Weighted graph, Weighted tree, Directed weighted graph.
`NEXTTO:`	`column-name NEXTTO` *value* Domain Structure: Graph, Weighted graph, Weighted tree, Directed weighted graph.
`PARENT:`	`column-name PARENT` *value* Domain Structure: Tree, Weighted Tree, Lattice.
`ANCESTOR:`	`column-name ANCESTOR` *value* Domain Structure: Tree, Weighted Tree, Lattice.
`CHILD:`	`column-name CHILD` *value* Domain Structure: Tree, Weighted Tree, Lattice.
`SIBLING:`	`column-name SIBLING` *value* Domain Structure: Tree, Weighted Tree, Lattice.
`DESCENDANT:`	`column-name DESCENDANT` *value* Domain Structure: Tree, Weighted Tree, Lattice.
`NEXTITEM:`	`column-name NEXTITEM` *value* Domain Structure: List, Circular list.
`PREVITEM:`	`column-name PREVITEM` *value* Domain Structure: List, Circular list.
`INBETWEEN:`	`column-name INBETWEEN` $value_1$ `AND` $value_2$ Domain Structure: List, Circular list.
`FIRST:`	`column-name FIRST` Domain Structure: List.
`LAST:`	`column-name LAST` Domain Structure: List.
`IN-SET:`	`column-name IN-SET` Domain Structure: Set.

4.5 Summary

The mesodata layer is positioned between the data and metadata and is accessible only through the data definition and manipulation commands presented in this chapter. These commands have been developed to be self-evident in meaning and follow standard SQL syntax. All extensions suggested are designed to co-exist with core SQL functionality. These extensions to SQL enable the integration of mesodata domains into a relational database and remove the need for specially written application code to manage complex domain structures.

Chapter 5

Application of Mesodata

Who controls the past controls the future.
Who controls the present controls the past.
George Orwell,
Nineteen Eighty-Four

This chapter, initially published in de Vries & Roddick (2004), describes the nature and use of mesodata with particular regard to its effect on domain evolution, data integration and querying. Examples of the applications of mesodata are presented to illustrate the advantages the mesodata layer brings to the relational database.

5.1 Domain Evolution

Domain Evolution refers to changes to 'real world' features that have been modelled in databases. The evolution of domains is inescapable. Database user requirements change due to a variety of reasons including new and changed laws, organisation mergers or splits, inventions, discoveries and developments in technology. The success of relational databases and their large market share means that masses of historical and current information is stored in them and as the database systems evolve there is loss of information. Domain history is not preserved unless it is part of the transactional definition.

A domain's values may alter without its changing being recorded within the DBMS and thus information is lost. Over the last thirty years, many international and universal domains have changed and some have come into being. A few examples are;

- Country names and their international country number
  (there are three different codes - alpha-2, alpha-3 and numeric-3 code - in ISO 3166),
- Telephone numbers and area codes,
- Postal codes/zip codes,
- Animal taxonomies,
- Disease taxonomies,
- Astronomical and celestial taxonomies,
- Genome data.

5.2 Change Management

General business principles for 'Change Management' can be applied to deal with system evolution. Interestingly, most concur with Shankaranarayanan and Ram (2003) core issues in schema evolution (q.v. page 8). These principles being:

Pro-activity The earlier that required changes can be identified and implemented, the lower the overall cost of the change will be, both in time and money.

Analyses Impact and Planning.

- *Impact Analysis* Impact and risk analysis allows the DBA to examine the involved risk to determine the best course of action. The goal being to understand all possible changes to the database schema, as well as to understand the implications of each change. There are often several ways to implement a single change. However, the impact of each type of change may be different. It should be determined how a change may or may not affect other parts of the schema, the data and applications. Some present higher risks: the risk of failure, the risk associated with a more difficult change, the risk of additional change(s) being needed, the risk of extended down-time, and so forth.
- *Planning Analysis* A well-planned change is cost and time effective. It is, of course, preferable to do it correctly the first time than to roll-back and start again.

Predictability/Reliability An organisation must know that the cost of effort is worth it. A high level of predictability is required for continued success and functionality. Changes must be incorporated into the existing schema while maintaining the consistent and correct state of the schema, as well as ensuring that data are consistent with the changed schema.

Availability Most changes require down-time to implement the change. Reducing the amount of down-time required to implement change will increase application availability and the organisation's productivity.

The most difficult of these is pro-activity. Analysis during the developmental stage of a system attempts to capture future needs though one cannot predict changes with any confidence. There will always be unanticipated internal and external organisational factors to accommodate and the evolution of a domain may occur so gradually that it is not noticed until incompatibility problems arise. Several of these factors are discussed in the empirical study of a commercial database system in Chapter 6.

Mesodata allows domains to be engineered so that attributes can be defined to possess additional intelligence and structure and thus reflect more accurately ontological considerations, including changes in the domain itself. The mesodata layer facilitates attribute domain evolution, integration of heterogeneous data and enables enhanced querying over existing data by the inclusion of the domain structure.

5.3 Attribute Domain Evolution

Attribute domain evolution is the evolution of the valid range of values that a database attribute (field) may store and the semantics they infer. For example, an integer field of 4 bytes can store values in the range of -2,147,483,648 to 2,147,483,647 whereas a float field of 4 bytes has a range of negative values from -3.402823E+38 to -1.401298E-45 and positive values from 1.401298E-45 to 3.402823E+38. The domain has changed even though the storage requirement has not altered. The domain of an attribute within a specific database, however, does not always coincide with the whole range of valid values a data type can store. The rules and constraints of the attribute's domain are either encoded within the attribute specification or verified and validated within the application code, to ensure that the recorded data are within the range of values that captures the semantics required within the Universe of Discourse (UoD). For example, whilst the `DATE` data type has a valid range over several millennia, within an application the valid range may be much narrower such as for birth dates in a *Genealogical* table, where the valid range may be several hundred to a thousand years, or within an *Employee* table where the attribute for birth dates would have a far more limited range of values (Figure 5.1).

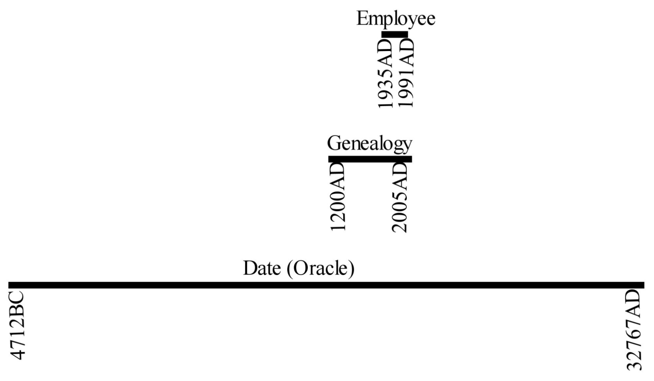


Figure 5.1. Ranges of Year Domains

Domain evolution may necessitate a schema change to an attribute, expanding or contracting the data type or changing from one data type to another. Currently when a schema changes, two events typically occur - the application is modified and recompiled to deal with the changes and the data are converted to the new format, either by *strict*, *lazy* or *no* conversion (Ferrandina et al. 1994).

Lazy conversion performs data conversion only when data are accessed and they are still recorded with superseded formats (or values), no conversion is done if the data are not accessed. Until all data have been accessed there exist some that are invalid, incomplete or uncertain.

Strict conversion requires that as soon as there is a modification to the schema all data are converted to conform with the current definition. During conversion all applications interacting with the database must be stopped and the database locked. Moreover, information is lost and changes cannot be reversed.

Adding a mesodata layer to the database structure, alleviates evolutionary factors. Mesodata types can store semantics as well as operators and operations in data structures other than base types, reducing the impact of not identifying changes early in the development cycle. A mesodata domain allows some changes to be made without the need for data to be converted or integrated and the application itself may not need to change, thus reducing the risks of failure and down-time while maintaining the reliability of the database system.

5.4 Categories of Domain Evolution

Domain evolution can be broadly categorised into three types:

Attribute Representation Change: expansion or contraction of field, for example, `CHAR(15)` to `CHAR(20)` or vice versa, change of base type: integer to float, numeric to character, character to enumerated list.

Domain Constraint Change: the possible range of values that may be recorded has changed without the metadata changing or the currently stored data changing, for example, the minima and/or maxima change. The new constraints may, or may not, be applied retrospectively.

Perception (meaning) Change: the semantics of the data change, for example, Reference 116Q15 no longer is interpreted as 'Burbridge Road' and is now 'Sir Donald Bradman Drive', however, both interpretations are required for historical purposes.

Ventrone and Heiler (1991) described seven forms of domain evolution that lead to semantic heterogeneity within databases which fall into the above three categories as follows:

Attribute Representation Change	Identifier Changes
Domain Constraint Change	Cardinality Changes
Perception (meaning) Change	Encoding Change Field recycling Granularity Change Heterogeneous instances Time and Unit Difference

To date all domain changes require manual effort for their successful incorporation into a DBMS, however using a mesodata layer in the database can reduce these problems, as illustrated in the following examples. The mesodata types selected for the examples are neither prescriptive nor proscriptive: just as the Database Administrator (DBA) judges which attribute data type to use, so too must the decision of which mesodata type to employ lie with the DBA.

5.4.1 Attribute Representation Change

Example: A character code is replaced by a number code. The specification CHAR(20) is altered to an INTEGER.

Current Typical Solution: Multiple steps are required to change an attribute's specification. These are:

- add a new attribute of type INTEGER to relation,
- write conversion procedures to convert the old data values to new values and populate the new attribute,
- delete the old attribute,
- rename the new attribute to the old name,
- update application to handle different type.

Mesodata Solution: Use the mesodata type, LIST, that maps the existing CHAR(20) values to the new INTEGER values. The attribute in the relation remains unchanged as does the application, as the operators to access the changed attribute type are built into the mesodata type.
For example:
`old AppCode = ‘widgetA’`
`new AppCode = 2131`
using the mesodata domain layer, we have,
`AppCode = Mdom(‘widgetA’) = 2131`

Both code values **2131** and **widgetA** are accessible and valid. Information capacity holds as both *equivalence* and *dominance* requirements are met.

It is recognised that not all attribute type changes can be handled using mesodata, for example from BLOB to INT, however there are many instances where the evolutionary process can be alleviated.

5.4.2 Domain Constraints Change

Example: ‘Country of birth’ is an attribute contained in a number of databases, the allowable values of which have changed significantly during the twentieth century. When a country name changes, it may be a one-to-one change, such as *Rhodesia* to *Zimbabwe*, a many-to-one change, for example {*West Germany* , *East Germany*} to *Germany* or one-to-many change, as in *Yugoslavia* to {*Bosnia Herzegovina, Croatia, Macedonia, Serbia, FYR Montenegro, Slovenia.*}

Current Typical Solution: Convert all old values and replace them with new values. Not only is this an ongoing task, it also results in loss of information.

Mesodata Solution: Utilise the mesodata types WGRAPH or TREE to map old values to the new values. The domain of 'countries' includes *All* country names, current and superseded, which are then accessible to the DBMS with the extended SQL operators and original values are not lost.

5.4.3 Domain Perception (meaning) Change

Example: A perception change may entail an absolute change where there is new interpretation of values in a domain or it may be the addition of synonyms. The days of the week stored numerically from 1 to 7 inclusive may interpret the value '1' as 'Monday', equally valid are the interpretations 'lunes', 'lundi', 'maandag', 'Montag', 'segunda-feira' and so forth.

Current Typical Solution: The application may be parameter driven to select a single preferred interpretation (such as language setting) or the users must learn the dominant term.

Mesodata Solution: A mesodata layer of SYNONYMS allows regularly used mappings to be accommodated in a database. Therefore we have
`1 = 'Monday' = 'lundi'` etc.

5.4.4 Minimise Change

Mesodata helps to reduce potential systems changes to one of two simpler solutions

1. A change to the schema definition that requires no change to either the application or data.

2. A change to the mesodata reference relation with or without a change to the schema but again, without the need to change either the application or the data.

Schema integration and transformation is not required as the mesodata type has the operators and 'intelligence' to replace these tasks. Information capacity is not only maintained, as both requirements of equivalence and dominance are met, but also in many cases expanded as the cartesian product of the mesodata domains is greater than the original domain of the relation.

5.5 Data Integration

A key problem is raised by semantic heterogeneity, as occurs within a database through domain evolution, as well as when data duplicated across multiple databases are represented differently in the underlying database schemas. Solutions need to be developed to manage this diversity. Being able to identify and specify the relationships between two or more items of replicated data and constructing a mapping to store those relationships, provides a tool to create an integrated view of overlapping datasets from multiple databases. As well as this, the mesodata mappings provide for semantic similarity to be measured so that conceptually near records can be included in views and queries. This project has investigated the utility of different complex data structures (Roddick, Hornsby & de Vries 2003, Rice, Roddick & de Vries 2006) for domains from synonym lists to mappings that include conceptual distance and methods by which similarity or dissimilarity can be measured.

As an example, consider the problems resulting from the logical integration of data from two databases defined over different schemata for the purposes of executing database queries, as shown in Figure 5.2.

RelA			RelB		
	PartId	CHAR(5)		PartId	CHAR(5)
	Description	CHAR(20)		ItemDesc	CHAR(20)
	Colour	CHAR(8)		ItemCol	CHAR(6)
	Category	CHAR(10)		ItemType	CHAR(12)
	SupplierCategory	NUM(5)			

Figure 5.2. Heterogenous but Similar Schemata

While Description and ItemDesc may be semantically equivalent (and thus simply merged), Category may exist within a product hierarchy in which ItemType is a non-leaf node, Colour and ItemCol may take their values from nodes of a colour graph (such as that in Figure 5.3) and SupplierCategory may have no equivalent

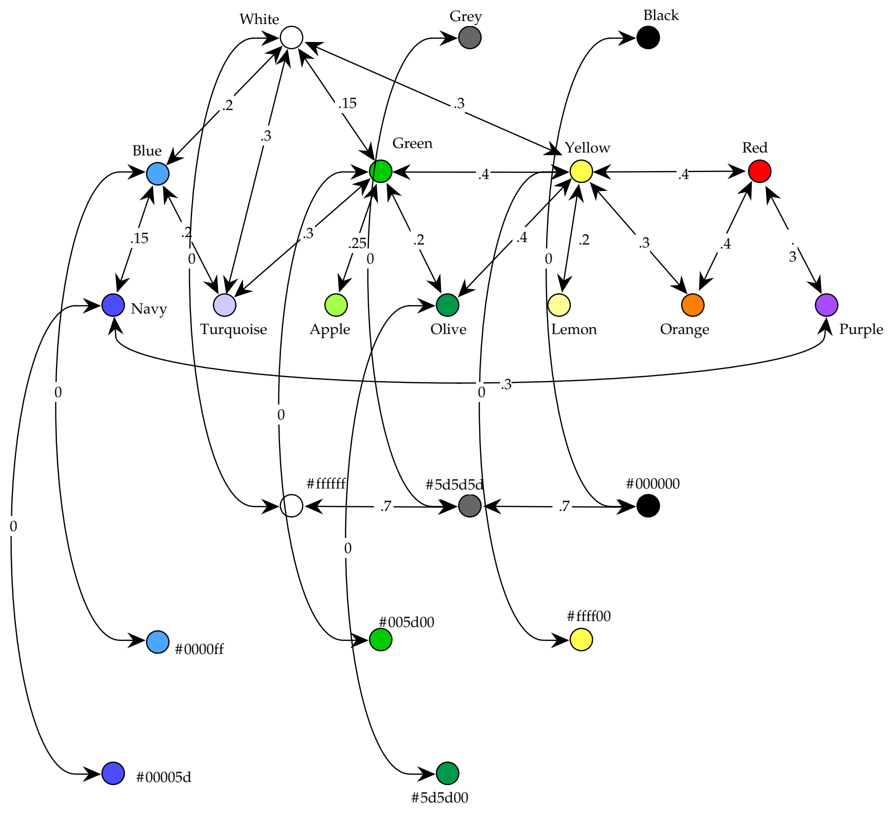


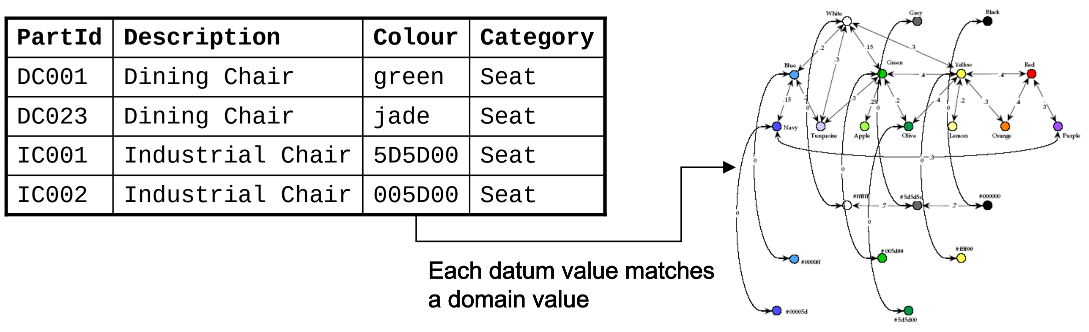
Figure 5.3. Example Colour Chart as Weighted Graph

attribute. Consider now the following query executed over the combined data and using a hybrid schema RelAB:

```
SELECT     PartId, Colour
  FROM     RelAB
  WHERE    Category = ‘Seat’
  AND      Colour = ‘Green’
```

The mesodata response to this is to utilise the colour graph to *translate* ItemCol to the closest defined Colour and to use the product hierarchy to convert instances of ItemType to Category. In addition, this could be achieved fairly automatically if the DBMS had access to appropriate domain definitions. Unfortunately, in many cases, the overhead of creating these definitions is too large and other options, such as coercing data to the most general common format, is adopted. Indeed, in practice, the constraints imposed by DBMS often have a large impact on design decisions.

In addition to the simplifications inherent for design and implementation issues, by adopting the accommodation of intelligent domains they can also be utilised to provide richer querying capability. For example, if an attribute is defined over a hierarchy or a graph, there is access to advanced semantic concepts such as *descendant*, *closeness* and so on (Kedad & Métais 1999, Roddick et al. 2003). In the example below, the operators DESCENDANT and CLOSETO are defined to operate over specific mesodata types.



PartId	Description	Colour	Category
DC001	Dining Chair	green	Seat
DC023	Dining Chair	jade	Seat
IC001	Industrial Chair	5D5D00	Seat
IC002	Industrial Chair	005D00	Seat

Figure 5.4: Attribute 'Colour' Referencing Mesodata Type Weighted Graph

```
SELECT     PartId, Colour
  FROM     RelAB
  WHERE    Category DESCENDANT 'Seat'
  AND      Colour CLOSETO 'Green'
```

Thus, in our example for the attribute 'Colour' (Figure 5.4), the base type would remain unchanged (eg. CHAR(8)) as would the data values. However, the values of both Relation A and Relation B exist within a weighted graph of colours and operations such as *Find all parts with a colour close to green* would not then require specific code within the application but would be handled by the DBMS through reference to the mesodata type.

5.6 Enhanced Queries

Stonebraker & Moore's (1996) classification of database management systems categorises relational database systems using SQL as 'simple data with queries' and object-relational database systems using SQL and user-defined functions as 'complex data with queries'. This thesis proposes an additional category of 'complex domains with queries' into which both relational and object-relational database systems can fit.

Capturing the inherent organisation of a domain within a database system is only done when the importance of the relationships between attribute values are specified in the system requirements. For example, accounting systems not only record the accounts' transactions but also record the hierarchy of the *chart of accounts* so that various reports may be generated within the hierarchy. There are, however, many domain structures that are overlooked when specifying a relational schema as their intrinsic value is not recognised or the complexity of the domain's structure has been managed within the application code.

5.6.1 Example of a Circular Domain

Time is one such domain where many of its aspects are usually managed within the application and not the metadata. Days of week in Stanford KSL's *simple time ontology* are hard coded with their English names. Specifically dated holidays may be defined by the user, such as 1 January, but weekends and weekdays are not defined. The simple time ontology has been designed primarily to look at time in a *linear* way and it awkwardly deals with some recurring time events but not with all. There are two ways we deal with time events: one is the linear time as in the simple time ontology, the other is *circular* time where we only deal with recurring events. Weeks, months, and seasons are not linear concepts but circular, as are clock time and astrological phases. A clock face is circular with good reason.

The domain *days of the week* is commonly used in applications for scheduling events. One is not so interested in the *date* upon which an event occurs but rather which *day* it takes place, such as every Wednesday or Friday. A week is conceptualised as seven consecutive days, as well as, a weekend and five weekdays. Days of the week are cyclic; Monday to Wednesday is different from Wednesday to Monday.

A circular list mesodata type supplies operations for traversal, comparison, addition and subtraction. This mesodata type populated from an ontology for 'Days of Week' provides the context and semantics of the circular list and thus we have operations such as PREVITEM and NEXTITEM, 'Wednesday' + n, INBETWEEN 'Monday' AND 'Friday' and also the ability to query for 'weekday' or 'weekend'. As illustrated in Figure 5.5, the mesodata domain stores the values from the ontology, which in this example utilises a circular list of synonyms that has terms both in English and French, enabling querying in either language without the need for further code to translate.

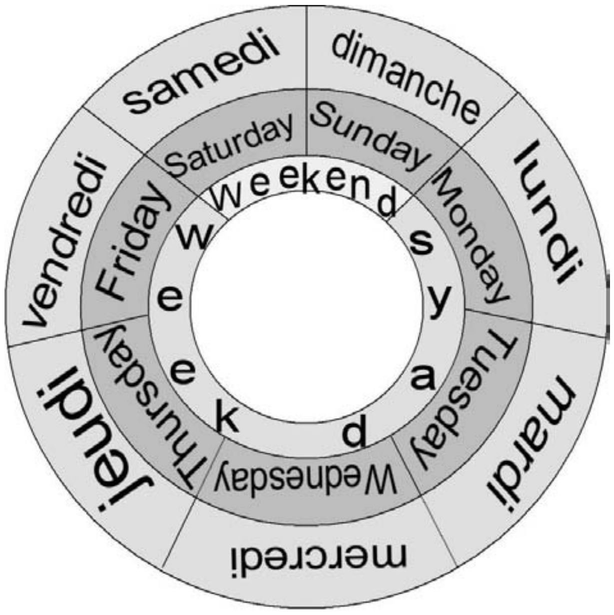


Figure 5.5. Days of the Week with English and French Terms

5.7 An Object-Relational Example

The following example illustrates the use of mesodata in both an object-relational and relational system to highlight the difference between a user-defined data type and a mesodata type.

An object is required to store information about a person including a telephone number. A person's telephone number (Figure 5.6) may be seen as (A) an attribute of the entity PERSON if it is assumed that there will be only one number per person, or (B) as a related entity with a one-to-many cardinality. Either way the value of the number is regarded as a simple data type. In a relational database the definition for PERSON may be

```
Option A
create table PERSON(
    PID          INTEGER     not null,
    FAMILY_NAME  CHAR(30)    null    ,
    GIVEN_NAME   CHAR(30)    null    ,
    TELEPHONE    CHAR(30)    null    ,
    constraint PK_PERSON primary key (PID))
```

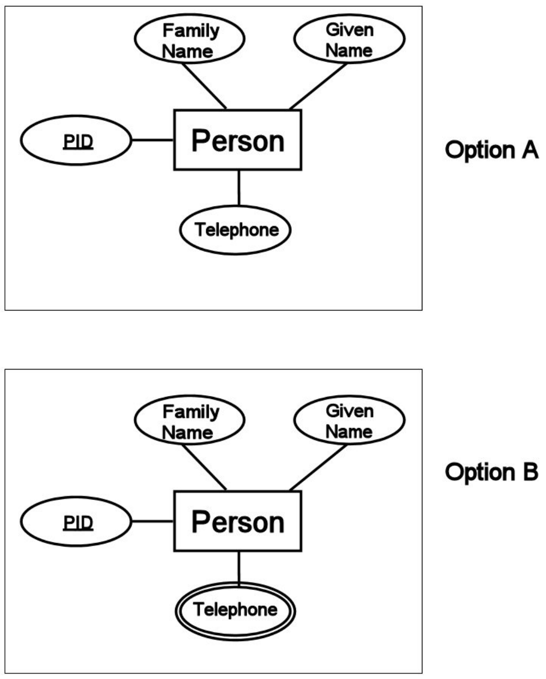


Figure 5.6. Configurations for Telephone Numbers

```
Option B
create table PERSON(
    PID          INTEGER     not null,
    FAMILY_NAME  CHAR(30)    null    ,
    GIVEN_NAME   CHAR(30)    null    ,
    constraint PK_PERSON primary key (PID))

create table TELEPHONE(
    PID          NUMBER(5)   not null,
    TELEPHONE    CHAR(30)    not null,
    constraint PK_TELEPHONE primary key (PID, TELEPHONE)
    constraint FK_TELEPHON__PERSON foreign key  (PID)
       references PERSON (PID))
```

whereas in an object-relational database the relation definition covering both option A and B may be

```
CREATE TYPE PHONE_ARRAY IS VARRAY (10) OF CHAR(30)

create table PERSON(
    PID         INTEGER    not null,
    FAMILY_NAME  CHAR(30)  null    ,
    GIVEN_NAME   CHAR(30)  null    ,
    TELEPHONE    phone_array       ,
    constraint PK_PERSON primary key (PID))
```

in addition to which a user-defined function must be written to retrieve the individual telephone numbers.

5.7.1 Hierarchical Domain

Even though the more complex data type of variable array has been created in the latter example, the values themselves are still being stored in the same format, i.e. CHAR(30). Neither format retains information about the *domain* of telephone numbers.

In Australia these numbers are not random. Land line telephone numbers belong to a state, a region and an exchange. There is a hierarchical structure to their allocation and from a number one can retrieve the exchange name, state, latitude, longitude and adjoining exchanges. The additional information contained in this domain is not often captured as only the values of telephone numbers are stored.

In order undertake an analysis of business activity according to geographic location, in either of the above constructs, would currently require schema modification and data integration to accommodate the additional tables, attributes and foreign keys for the domain data values as well as specifically written application code to traverse the domain structure in order to produce analysis reports. These modifications produce a new version of the database.

Creating a mesodata TREE containing the hierarchy of the telephone domain and referring the attribute of the telephone number to it, in either the object-relational or relational platform, enables the analysis reports to be generated without modifications to the application code or the schema of the database.

5.8 Summary

Though an attribute change in itself may not be a complex process it is not a trivial task. Database evolution and maintenance consists of many such simple steps as shown in (Sjøberg 1993) most of which also necessitate changes to application code and system down time. The mesodata layer, an additional domain definition layer containing domain structure and intelligence, provides the means to manage some aspects of attribute domain evolution. Its use when a domain changes, when the semantics of a domain alter or when the attribute's specification is modified can reduce or remove the necessity of schema conversion, schema integration, data conversion and application change as well as maintain or expand the schema's information capacity.

Semantic heterogeneity problems are alleviated by populating mesodata structures with data from ontologies, taxonomies, concept graphs and so forth due to the 'intelligence' of the structures that manage the inter-relationships of the domain values. Incorporating *domain* information also enhances querying power by adding advanced semantic concepts to comparison operators used to retrieve information from the database.

Although the mesodata layer has been developed with relational databases in mind, their incorporation into object relational databases is not only feasible but would provide the same benefits as argued for RDBs.

Chapter 6

Empirical Study of a Database System

This chapter reports on the study of the evolution of a commercial database system. It includes an overview of the system investigated, a description of its evolution and discussion of how the mesodata approach would benefit an evolving system.

6.1 Motivation for the Study

Modelling data in traditional relational databases requires specifying attributes over a restricted set of data types. These modelling techniques, in general, capture the 'nature' of the data value but do not capture information about the attribute domain or how a specific value may be related to other values within that domain. A number can be compared with other numbers, however the semantics of the number is discernible only when the domain is viewed as a whole. Year 2005 has a very different meaning from postcode 2005 and both domains require knowledge of the unit of measure to interpret the value and their relationships to other values. For example, postcode 2005 is not necessary adjacent to postcode 2006. Exacerbating this modelling problem is the fact that domains and database systems' requirements change, necessitating metadata changes, application code modifications and data conversions.

Structural alterations to databases have been well researched, as described in the literature review, with various techniques explored for schema evolution, schema transformation and schema integration, but none can currently deal with all aspects of evolution and few of them deal specifically with the problem of attribute domain evolution.

To ascertain the magnitude of the modelling problem, an empirical investigation was undertaken of the evolution of a commercial database system over nine years to measure and delineate changes to the database that are (a) structural and (b) attribute domain related. The goal of this study was to discover from these data what impact the mesodata approach would have on the effort needed to incorporate changing demands on the system's schema and as a consequence its data.

Sjøberg's (1993) work provides change statistics for a database system over eighteen months, covering six months of development and twelve months of field trials. This study complements his work as it follows the changes in a database system over many years. The data in this study describe the evolution of the *released versions* rather than the details of the 'develop, test and revise' cycle as researched by Sjøberg.

6.2 System Overview and Evolution

AGEIS is a relational database system (Figure 6.1)[1] which is designed to cater for the needs of aged care organizations in Australia by providing various functions including the following;

- Financial control, including debtors, creditors, general ledger, inventory and asset management,
- Management of aged care facilities and services,
- Government funding claim and reconciliation,
- Statistics and reporting.

This system and its development are representative of small-to-medium sized database systems and worthy of empirical investigation. The software is installed at eight sites. One client has 1800 independent living units, with about 2800 residents, and about 5,000 on the waiting list. They have 6 nursing homes with approximately 700 residents, and 300 on the waiting list. They also have about

[1]AGEIS is a commercial product of Versatile Solutions Pty. Ltd.

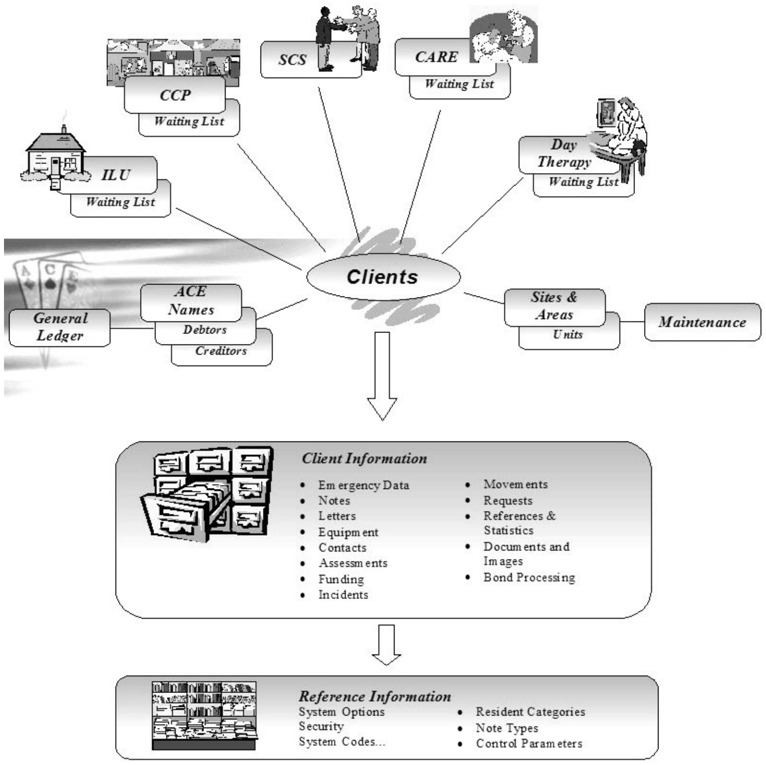


Figure 6.1. Schematic of the Database System
©Versatile Solutions Pty Ltd

230 community care clients, with about 200 on the waiting list. Their food services division supplies around 600,000 meals per annum to its aged-care facilities as well as to external organisations. In addition, they have about 900 staff.

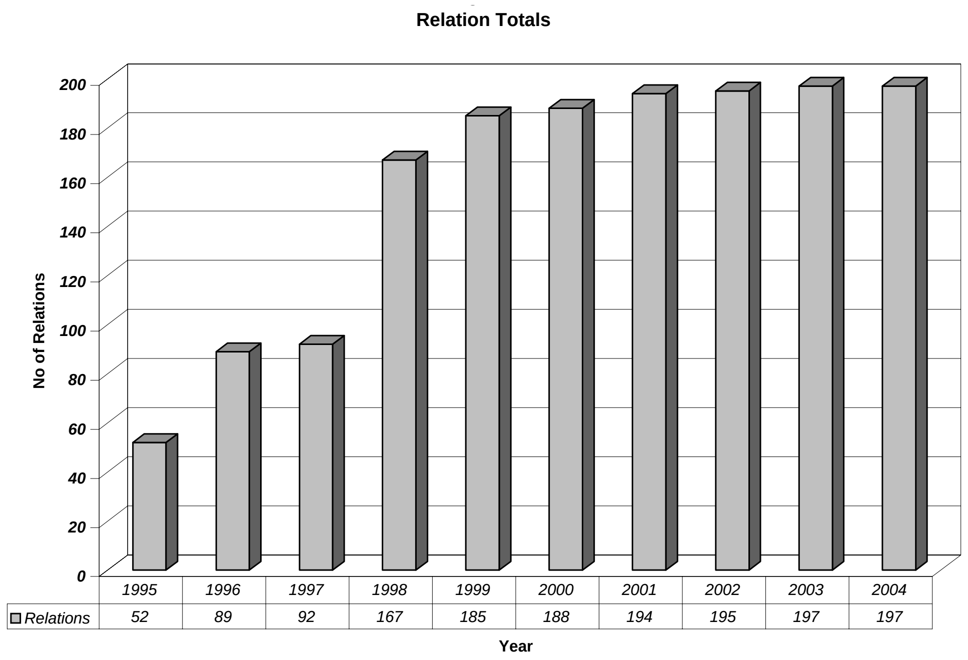
Since 1995, data definitions have been dated with their last modification and from 1999, an audit trail has been maintained of all metadata modifications. The date stamp is the date on which the definition was moved from 'development' to 'release'. In December 2004, a snapshot was taken of the current metadata configuration. This snapshot in conjunction with the audit trail has been analysed to provide a 'biography' of a database system.

6.2.1 System Metrics

The current version of the database system, at the time of the snapshot, consists of 197 relations - 16 of which are 'work files' for generating summaries and reports - with 3087 attributes representing 1446 distinct domains, 287 indices and 139 logical views. During the period analysed, the system has grown from an original 52 relations to 197 relations, as shown in Figure 6.2, expanding from 401

Milestones of the System	
pre-1995	Legacy system: menu driven with a text-based user interface
1995	The 'ACE' Financial System: event driven with a graphical user interface
1996 - 1997	Asset Management added to 'ACE'
1998	'AGEIS' integrating the original 'ACE' financial system with new modules for other aspects of aged care administration
1999	Staff Roster management added
2000	Role-based System Security introduced
2001 - 2004	Configuration modifications

attributes[2] initially to 3087, as illustrated in Figure 6.3. The database did not, of course, undergo a monotonic growth of attributes and relations. During the nine years, as new relations were created (Figure 6.4), attributes were created, deleted and modified (Figure 6.5). The motivations for these modifications to the database system fall into three classifications: (1) Clients' requests for new features, (2) New or changed laws, and (3) New or changed computing environment.

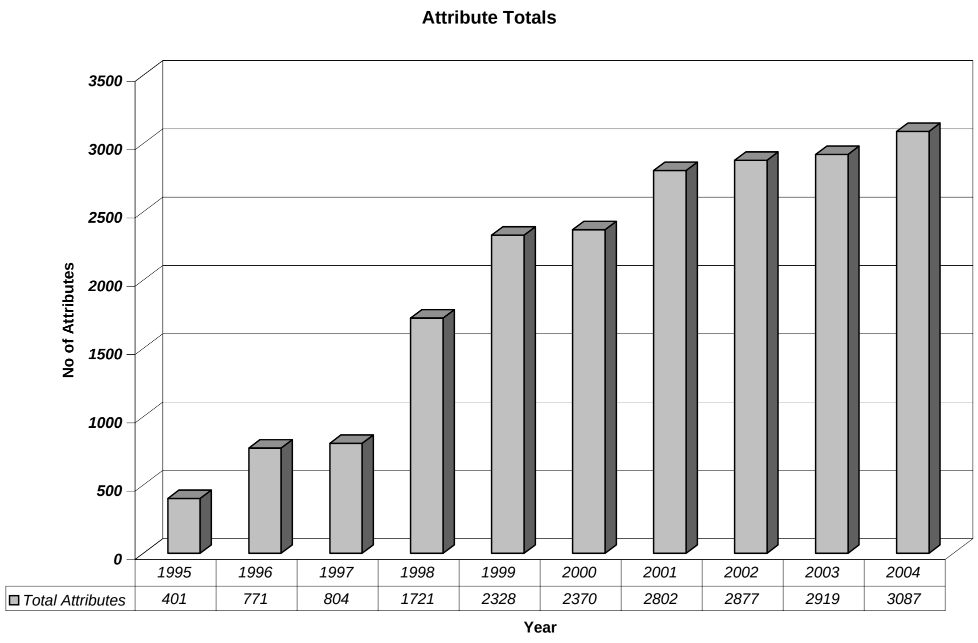


	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
▨ Relations	52	89	92	167	185	188	194	195	197	197

Figure 6.2. Growth of Relations

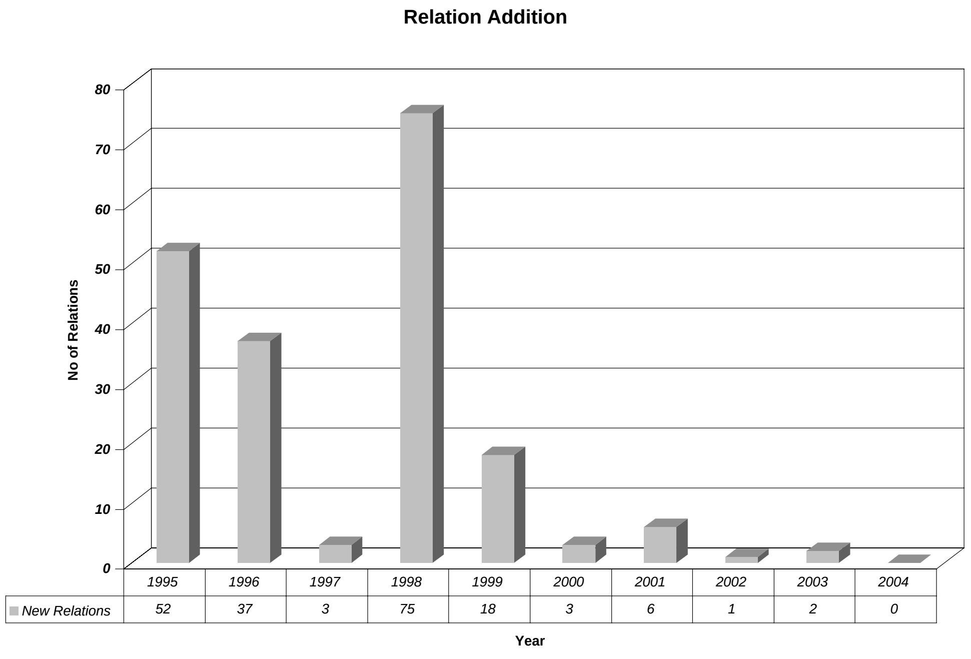
pre-1995 Legacy system with a 'green screen' text-based user interface for resi-

[2]NB: The number of attributes in a relation was not able to be calculated without audit trail data. Therefore, prior to 1999, the *precise* number of attributes and consequently domains is unknown. Numbers shown for 1995 to 1998 are a best approximation.



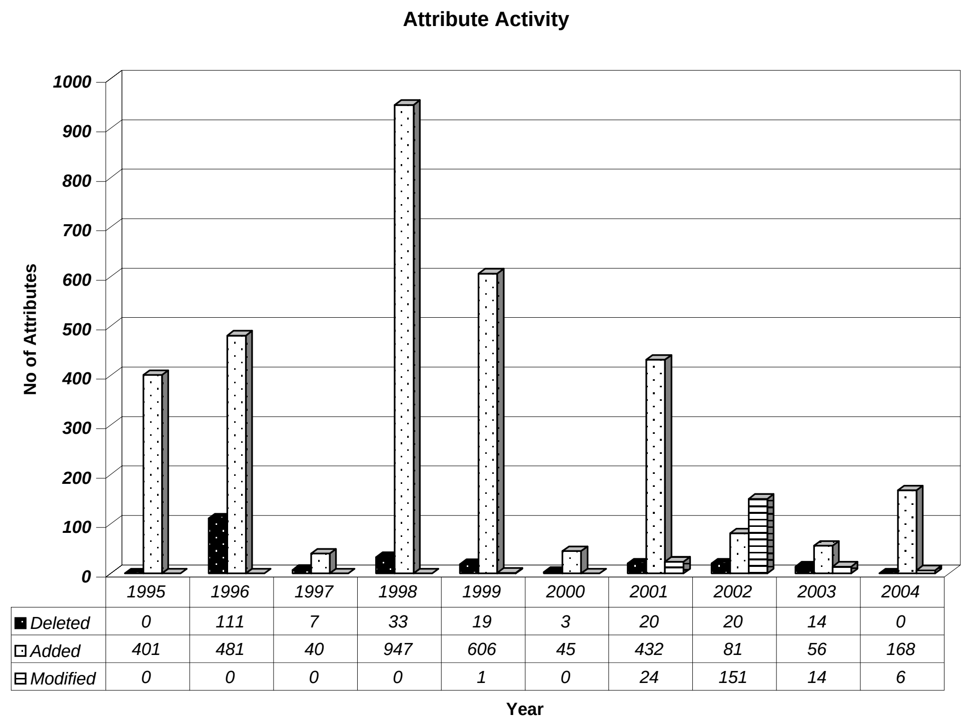
	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
▢ Total Attributes	401	771	804	1721	2328	2370	2802	2877	2919	3087

Figure 6.3. Growth of Attributes



	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
■ New Relations	52	37	3	75	18	3	6	1	2	0

Figure 6.4. New Relations



	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
■ Deleted	0	111	7	33	19	3	20	20	14	0
□ Added	401	481	40	947	606	45	432	81	56	168
⊟ Modified	0	0	0	0	1	0	24	151	14	6

Figure 6.5. Attribute Movement

dent care.

1995 The ‘ACE’ Financial System was created with an event driven graphical user interface incorporating the legacy system. The development was motivated by clients requesting a GUI interface and integration with an ‘MS Windows’-compatible platform. The application code platform changed from RPG to Visual Age. This version contained 52 relations.

1996 - 1997 New and current clients requested that Asset Management be added to ‘ACE’. The database expanded to 89 relations and also required modifications to the existing relations pertinent to Banking, Sales and Purchases. The addition of this financial module required an additional 481 attributes and 111 attributes were no longer necessary and were deleted.

1998 ‘AGEIS’ integrating the original ‘ACE’ financial system with new modules for other aspects of aged care administration. The database at this stage had 167 relations. The development of AGEIS was actuated by clients’ requests for more features in the software as well as government policies on reporting requirements for the Aged Care Industry.

1999 A Staff Roster management module was added at a client’s request to replace a separate legacy DOS system in order to interface between the financial management and an external payroll package. Taxation calculations on sales and purchases altered as a consequence of new taxation laws to be introduced in 2000. The database had 185 relations.

2000 Role-based System Security introduced to comply with the new Privacy Act, as well as to cater for the changes in a client's hardware configuration. Prior to this only authorised personnel had access to a workstation and, therefore, the software. From this point, the integrated system was required to run on all workstations accessible to a range of personnel with varying levels of authority. The database contained 188 relations.

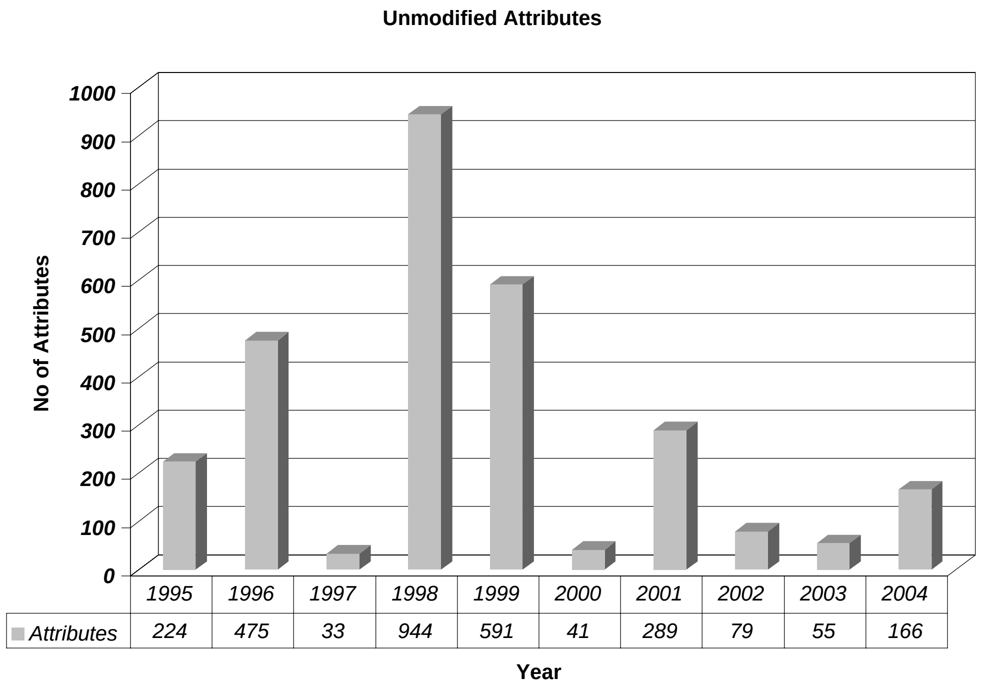
2001 - 2004 Clients' upgrading their hardware and operating systems required some minor modifications. Developer optimisations were also carried out, resulting in a total of 197 relations.

6.2.2 Stable Characteristics

2897 attributes (1381 domains), i.e. about 94% of the attribute data definitions, in this system have not been altered since their creation. Whilst this figure may seem initially to defy the widely held belief that clients do not always know what they want, one should remember that these are attributes that have been through the whole development cycle prior to their release. It is, however, reassuring that this cycle did produce such a stable system and perhaps reflects the abilities of the software engineers and developers, as well as the good communication between the developers and their clients. In particular, note that the major changes to the system in 1998 and 1999 were very stable (Figure 6.6) with 99% of attributes added in those two years remaining unchanged.

6.2.3 Deleted Values

Almost all of the deleted attributes in this database system were as a result of application change rather than data values no longer being required. The total number of deletions over the nine years is 227 (Figure 6.7), of which 165 deletions were from extract, summary or work relations and 7 deletions were direct moves from one relation to another. Of the remaining 55, only one domain was no longer required, with all the other domains being stored in different configurations. That is, some domains were denormalised, some merged and others split with the



	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
■ Attributes	224	475	33	944	591	41	289	79	55	166

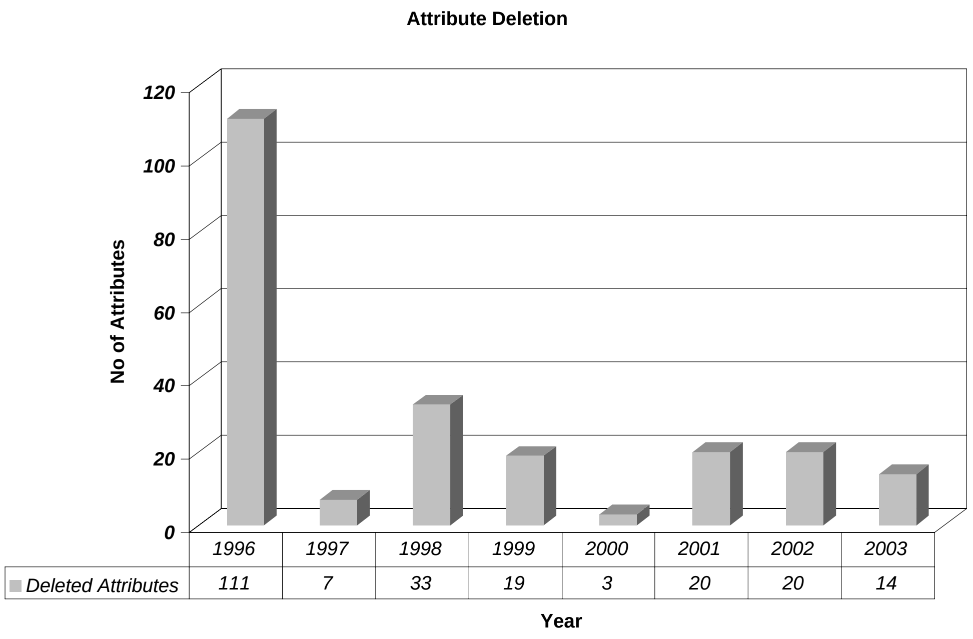
Figure 6.6. Unmodified Attributes

results stored as new attributes. During the analysed period four relations were deleted.

6.2.4 Modified Domains

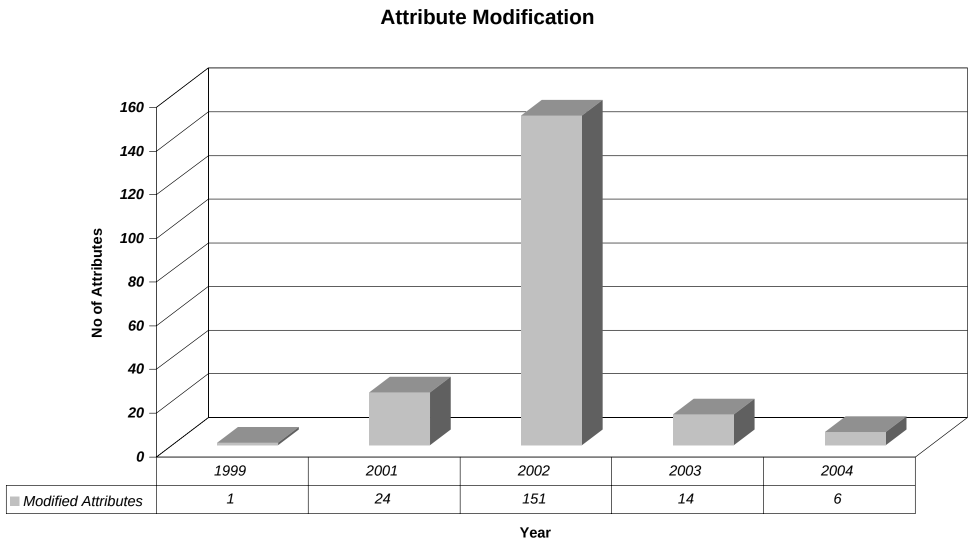
Data relevant to modifying domains and attributes were collected from 1999 onwards, hence modifications to attributes for the first four years are reported as zero in Figure 6.5. The six percent of attributes that have been modified pertain to 69 different domains affecting 196 attributes. The peak period of attribute modification took place in 2002, (Figure 6.8), to deal mainly with hardware and operating system changes rather than as result of added or altered functions to the system. These changes include 158 data definition changes, of which 5 were type changes, with the remaining 38 attributes having their data dictionary (description) changed with the actual domain modification (constraints and semantics) being handled within the application code. Four domains (7 attributes) that had their data definitions altered, reverted to their original specification during 2002.

Modified and new domains affected 161 of the 197 relations (Figure 6.9), that is only 36 relations have remained unaltered since their creation. While we have previously noted that 94% of attributes are stable, 82% of relations are not. On



	1996	1997	1998	1999	2000	2001	2002	2003
■ Deleted Attributes	111	7	33	19	3	20	20	14

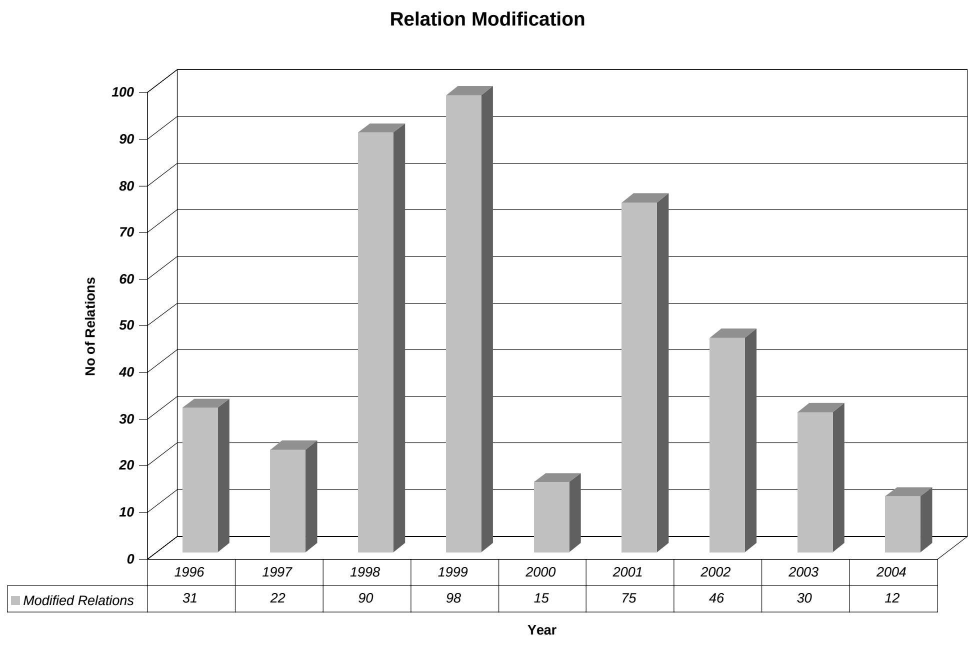
Figure 6.7. Deleted Attributes



	1999	2001	2002	2003	2004
■ Modified Attributes	1	24	151	14	6

Figure 6.8. Modified Attributes

average a relation has been modified 3.6 times with a maximum of 8 times.



	1996	1997	1998	1999	2000	2001	2002	2003	2004
▨ Modified Relations	31	22	90	98	15	75	46	30	12

Figure 6.9. Modified Relations

6.3 Data Conversion and Maintenance

In the nine years this database has undergone every type of evolutionary operation pertinent to schema evolution which were categorised in (Roddick, Craske & Richards 1993) as follows:

- Domain/Attribute Evolution (*Attribute Domain Change*)
  - ⋆ Expanding an attribute domain[3]
  - ⋆ Restricting an attribute domain
  - ⋆ Changing the domain of an attribute
  - ⋆ Adding an attribute to the database
  - ⋆ Renaming an attribute
- Relation Evolution (*Structural Change*)
  - ⋆ Adding a relation
  - ⋆ Deactivating a relation

[3]Roddick's use of the term *domain* refers to the data type

  - ⋆ Activating a relation

- Attribute-Relation Assignment Evolution (*Structural Change*)
  - ⋆ Adding an attribute to a relation
  - ⋆ Deactivating an attribute
  - ⋆ Promoting an attribute
  - ⋆ Demoting an attribute
  - ⋆ Splitting a relation
  - ⋆ Partitioning a relation
  - ⋆ Joining two relations
  - ⋆ Coalescing two relations

Versatile Solutions, the owner of the system, undertakes *strict conversion* of data to ensure that data are consistent with a new schema. Population of new and modified attributes requires using default values or converted values from other domains, which may have been specialised, generalised or calculated. New schema design required conversions to create links where they had not existed before while preserving historical information. Versatile Solutions' CEO, Arthur Verster stated that '*It is crucial not to lose client trust in the validity of historical data, as in this industry the data are needed to report to government and comply with the law*'. Thus attesting to the importance of information capacity preservation (Hull 1986, Qian 1996, Miller, Ioannidis & Ramakrishnan 1993, Miller et al. 1994*a*, Miller, Ioannidis & Ramakrishnan 1994*b*) and data conversion/integration in database evolution.

The most effort-intensive data conversions took place when converting from legacy systems. The factors contributing to the complexity of data conversions were;

- Lack of data entry standards resulting in multiple terms for one meaning (synonyms),
- Valid domain values changing over time resulting in some entries no longer being valid and others acquiring different semantics,
- The introduction of laws that meant that values had to be calculated according to a new set of criteria,

- Client organisational changes affecting veracity/validity of existing data values,
- The new model representing entities, attributes and relationships differently from the legacy model resulting in conflicts.

The conversion of much of the data first required industry domain experts to manually inspect, and correct, the data for redundancies, errors, ambiguities and missing information. After which the data was processed by specifically written conversion procedures to populate the relations. For just one domain, at one site, this entailed 80 working-hours of preparation before the data were programmatically converted. Note that this is a small-medium sized enterprise (SME) and recognise that the cost, both in time and money, for data conversion and integration is significant.

Database evolution, as presented in this thesis, includes not only metadata evolution but also attribute domain evolution, which can be broadly categorised into three types;

Attribute Representation Change: expansion or contraction of field,

Domain Constraint Change: the possible range of values that may be recorded has changed without the metadata changing or the currently stored data changing,

Perception (meaning) Change: the semantics of the data change.

Using mesodata structures allows historic data to be kept instead of deleting and/or editing, minimising information loss and reducing the need to convert data with every release or version. The different mesodata types reflect the properties of the domain, e.g. a tree, matrix or circular list, and not just the values within the domain. The inbuilt intelligence of the mesodata type also reduces the need to change application code as a change in the domain affects only the domain and not the whole system. Constraints and rules residing within an *intelligent domain* do not require metadata changes or code changes.

For example, there is an attribute within the database for recording a resident's 'country of birth'. The the allowable values of which have changed significantly during the twentieth century. When a country name changes it may be a one-to-one change, such as *Rhodesia* to *Zimbabwe*, or one-to-many change, for example *Yugoslavia* to {*Bosnia Herzegovina, Croatia, Macedonia, Serbia, FYR*

Montenegro, Slovenia}. The dilemma being to record the name of a country as it was at the time the person was born or as it is now.

In addition to the evolution of the domain itself, there were no entry standards in the legacy system to govern the form that a country's name should take. This lack of standards resulted in a variety of terms being used for a single country, such as *N.Z., NZ and New Zealand*, and *U.K., UK, United Kingdom, G.B., GB, Great Britain, England, Scotland, Wales, Northern Ireland.* To resolve these issues all old values were converted and replaced with new values.

The *mesodata* solution for the attribute for 'country of birth' is to utilise the mesodata types WGRAPH or TREE to map old values to the new values. The domain of 'countries' includes *All* country names, current and superseded, which are then accessible to the DBMS with the extended SQL operators and original values are not lost.

This empirical study has shown that over five years 6% of attributes have evolved with 81% being attribute representation changes and 19% domain constraint or perception changes. All of these changes could have been handled using mesodata types. Additionally, domains defined within the mesodata layer are re-useable, which would have reduced the 196 attribute changes to 69 domain changes. Few of these domain modifications would have required schema, data or code alterations after initially implementing the three-layered database. Evaluating what effect the mesodata approach would have on the system's development, it is estimated that it would reduce cost by at least 10%[4].

6.4 Summary

The database system in this empirical study is representative of a system in an SME that has evolved from a relatively simple financial system to an integrated system handling multiple areas within its universe of discourse (UoD). The data available for this analysis consist of date-stamped changes to relations over a period of nine years, as well as, date-stamped modifications to attributes over five years. Investigation of these data reveal the system's growth as summarised in Table 6.1. The primary motivations for modifications to the system fall into three classifications: (1) Clients' requests, (2) New or changed laws, and (3) New

[4]A more precise estimation cannot be made as there are no data concerning application code modifications as a result of domain change.

Table 6.1. Analysis Results

		Feature	**Category of Change**
379%	Increase	Number of Relations	Structural
2968%	Increase	Number of Attributes	Structural & Attribute Domain
82%	Modified	Number of Relations	Structural
6%	Modified	Number of Attributes	Attribute Domain
7%	Deleted	Number of Attributes	Structural

or changed computing environment. It is also inferred that attribute specifications are quite stable after the development cycle as 94% of attributes did not change. In contrast, relation structures are not, with 82% of them modfied at least 3.6 times with a maximum of 8 times. The 6% of attributes that evolved over five years required effort-intensive data coercion and conversion, in addition to application code modifications.

Investigation of the applicability of the mesodata approach to system design revealed that the initial conversion from legacy data would have been simplified and that the inbuilt intelligence of mesodata types would have accommodated the subsequent changes to attribute domains thus reducing effort and costs.

Chapter 7

Prototype Model

This chapter provides an overview of the prototype that has been developed to validate the concepts presented in the previous chapters. The prototype has primarily been developed as a proof of concept and consequently the goal was not to implement a complete solution containing all mesodata types previously described, but rather to provide an example of the efficacy of mesodata domains.

7.1 Prototype Evaluation

Introducing a mesodata layer into a relational database system has been argued in this thesis to provide the following benefits:

- The facilitation of attribute evolution by using mesodata types to store domain values so that attribute representation changes, constraint changes and perception changes are implemented with minimum information loss and application code changes.
- The facilitation of data integration by using mesodata types to store similar terms and their inter-relationships within a domain so that data need not be coerced and/or converted.
- The provision of enhanced querying by incorporating all valid domain values in the mesodata layer including values that may not be present in the current data layer.

7.1.1 Evaluation Criteria

The evaluation of the prototype model has two major sections, firstly that the model functions correctly with the new data definition and data manipulation commands documented in Chapter 4, and secondly that the claimed benefits are verified in the results. The first phase of evaluation, the prototype *functionality*, detailed in Appendix D, verifies that all proposed syntax is executed correctly. The second phase more importantly shows that mesodata domains do provide the benefits claimed.

The evaluation criteria for the second phase of testing are;

- Enhanced Querying,
- Attribute Representation Change,
- Attribute Constraints Change,
- Attribute Perception Change,
- Domain Values Change,
- Data Integration.

A sample session of the execution of the prototype model is documented in Appendix B.

7.1.2 Prototype Platform

The prototype model was built using the MySQL (version 4.0.12) database platform and msql-connector-java-3.08 and written in Java version 1.4.1-b21. The model simulates an SQL server with a three layer database platform. The main components of the model, as shown in Figure 7.1, are the client application, the mesodata wrapper, the MySQL server, the mesodata layer, the metadata and data.

The client application user interface is a command line tool for entry of both standard and extended SQL statements and for the display of query results. The mesodata wrapper consists of an SQL parser to process the extended SQL statements and to interact with the mesodata layer. The MySQL platform and its interaction with the database has not been altered in any way, to ensure backwards compatibility of the mesodata layer with existing relational databases.

To reduce anomalies in processing, the database attributes are specified with base types that map precisely between MySQL and Java (q.v. Appendix G).

7.1.3 Prototype Components

In order to reproduce a three layer system, two tables are created as 'system' files (denoted as *mesodata types* in the deployment diagram), that store structural information regarding mesodata domains. These files are

- the DOMAIN table to store the created domains, their mesodata types, base types and source relation and,
- the MESODATA REFERENCE table to store the attribute names, and relation names, that have been defined over a particular domain.

The implemented mesodata types in the prototype are *weighted graph*, *directed weighted graph* and *list*. The *graphs* are constructed with inbuilt operators for EQUALTO, CLOSETO, and ADJACENCY using the unifying semantic distance model (Roddick et al. 2003) in which a graph-based approach is used to quantify the distance between two data values. This approach facilitates a notion of distance, both as a simple traversal distance and as weighted arcs. In this prototype, the graphs have weighted arcs with values from 0 to 1 representing the degree of similarity with equality (synonym) being 0 and the default maximum value for 'close similarity' set to 0.5.

The populated domain structures, *domain data*, are also stored as tables within the mesodata layer. These tables are deemed as not directly accessible from the client application but only via the extended SQL operators that are built into the mesodata type. In order to visually separate the mesodata 'system' files from the data relations, all mesodata files names are in upper case and other relations in lower case. Mesodata type specifications, that is, the required structure of a source relation and the valid operators over them, are stored as constants within the mesodata definition language (meso DDL) and manipulation language (meso DML). It is envisaged that mesodata types can be described by XML, allowing for more flexibility in the design of the structures. The structures themselves are easily described, encoding the inbuilt logic is more difficult and is on-going research.

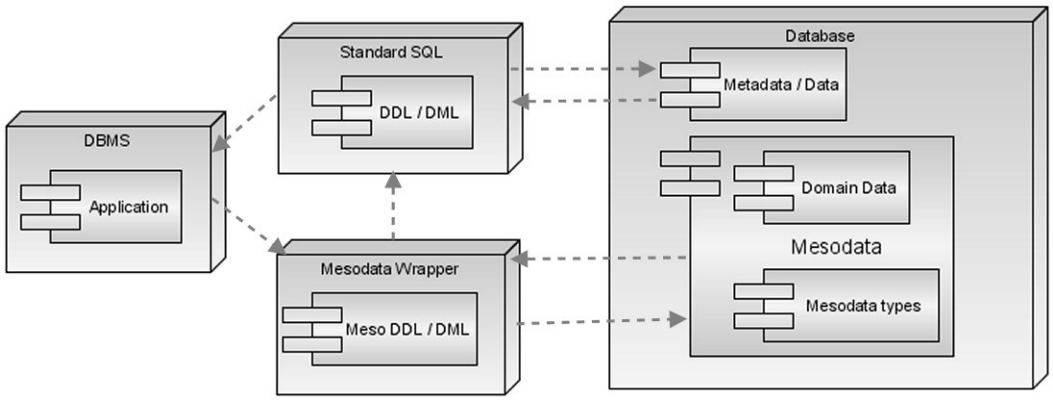


Figure 7.1. Deployment Diagram

7.1.4 Query Parser

The 'engine' of the prototype model is the mesodata wrapper. This component parses the SQL statements to capture the new mesodata operators and process them within the mesodata layer. An entered statement is split into Mesodata Query Language (Mesodata QL) and Standard SQL phrases and the Mesodata QL phrase is executed first over the domain data and then over the database data. For example, a query for *Find all products in a shade of green* is

```
select * from product where Colour closeto 'green'.
```

The phrase `Colour closeto 'green'` is executed over the domain data to retrieve all valid values for comparison. The phrase `select * from product` is a standard SQL phrase that is combined with the results from the domain and is subsequently handled by the SQL server over the database data.

When an entered statement is a more complex combination of standard and extended operators, the Mesodata QL phrase is separated and processed first and the result is then recombined with the original standard SQL statement's phrases for further processing. A query such as *Find all customers, and their invoice numbers, who bought a type of chair in a shade of yellow* is

```
select CustomerName, sales.InvoiceNo, ItemType, Colour
from customers, product, sales, salesitem
where ItemType CLOSETO 'chair' and
Colour CLOSETO 'yellow' and
customers.CustomerID = sales.CustomerID and
sales.InvoiceNo = salesitem.InvoiceNo and
product.PartID = salesitem.PartID
```

In this example, the two mesodata QL phrases `ItemType CLOSETO 'chair'` and `Colour CLOSETO 'yellow'` are processed over the domain data and their results combined with the remaining standard SQL phrases. Figure 7.2 illustrates the major activities that take place within the mesodata wrapper component.

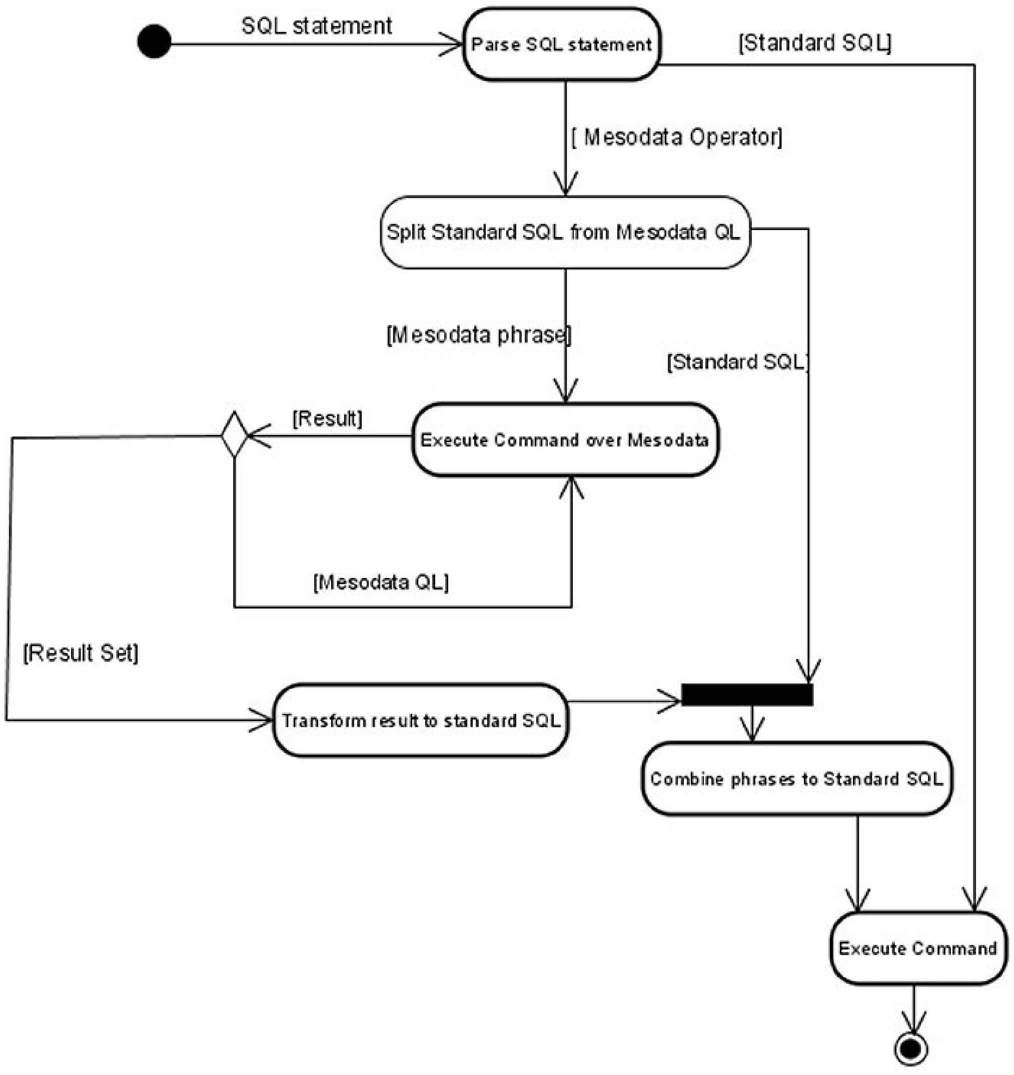


Figure 7.2. Activity Diagram for Mesodata Wrapper

7.2 Example Database

The example database was initially created containing five relations as shown in the ER diagram in Figure 7.3 with Mesodata domains for Categories and Colours. The mesodata domains were created as

- a weighted graph of twenty-five Colours, and

- a directed weighted graph of seventeen Categories.

The source relations for these domains were populated with SQL source files listed in Appendix C.1 and C.2 respectively.

The relations were populated with sample data of

- 27 records in relation Product (Appendix C.4),
- 9 records in relation Suppliers (Appendix C.6),
- 27 records in relation Customers (Appendix C.5),
- 25 records in relation Sales (Appendix C.7), and
- 25 records in relation Salesitem (Appendix C.8).

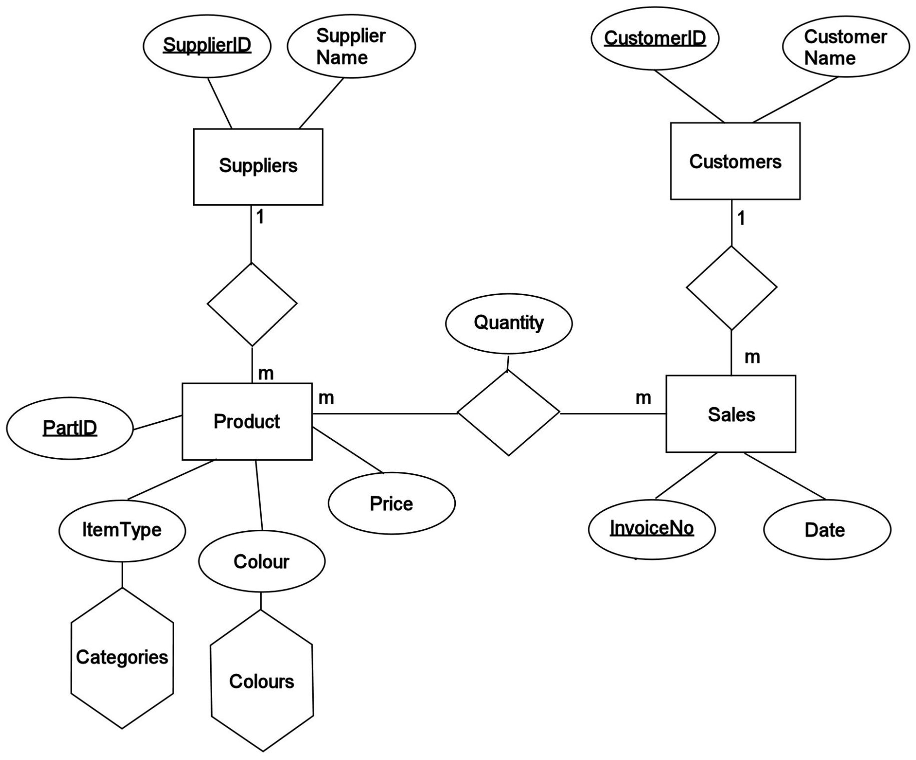


Figure 7.3. Entity-Relationship Model of Test Database

7.2.1 Evaluation of Model

Scenarios were designed to represent the six evaluation criteria listed previously. A few examples of each of these criteria are presented here, with the complete evaluation schedule detailed in Appendix E.

7.2.2 Enhanced Querying

In the example database there are two mesodata domains, COLOURS and CATEGORIES which contain more domain values than are recorded in the relation *product*. These domains also hold a similarity measure between each value in the domain. The attribute *product.Colour* is defined over the domain COLOURS and the attribute *product.ItemType* is defined over the domain CATEGORIES.

Queries to retrieve records that are equal to (=) a specified value return only an exact match to the value whereas a query for records that are similar (closeto) return a larger record set. The '=' comparison operator being standard SQL, and 'closeto' a mesodata QL extension.

7.2.2.1 Query: 'Which products are green?'

```
select * from product where Colour = "green"
```

PartID	ItemType	Colour	SupplierID	Price
EG123	ErgonomicChair	green	2001	60.00
DC001	DiningChair	green	2007	90.00

7.2.2.2 Query: 'Which products are a shade of green?'

```
select * from product where Colour closeto "green"
```

PartID	ItemType	Colour	SupplierID	Price
IC001	ClericalChair	olive	2002	150.00
IC002	ClericalChair	lime	2002	150.00
EG123	ErgonomicChair	green	2001	60.00
EG456	ErgonomicChair	emerald	2001	60.00
DC001	DiningChair	green	2007	90.00
DC023	DiningChair	chartreuse	2007	96.00
DC510	DiningChair	jade	2006	126.00
SC345	BarStool	turquoise	2005	46.00
SC125	BarStool	seagreen	2005	55.00

PartID	ItemType	Colour	SupplierID	Price
RC831	Recliner	jade	2006	250.00
RC444	Recliner	lightgreen	2006	250.00
RC234	Recliner	darkgreen	2006	250.00
KC020	KitchenStool	lime	2008	40.00
KC021	KitchenStool	apple	2008	40.00
LC040	Sofa	aqua	2004	650.00
LC551	Lounge3seater	verdigris	2007	899.00
DT345	DiningTable	turquoise	2005	500.00
CT831	CoffeeTable	jade	2005	50.00

7.2.2.3 Query: 'List the customers who have purchased a recliner'

```
select CustomerName, product.PartID, Colour, ItemType,
sales.InvoiceNo from customers, product, sales, salesitem
where ItemType = "Recliner" and
customers.CustomerID = sales.CustomerID and
sales.InvoiceNo = salesitem.InvoiceNo and
product.PartID = salesitem.PartID
```

CustomerName	PartID	Colour	ItemType	InvoiceNo
Barry de Veen	RC444	lightgreen	Recliner	106
John Haggar	RC234	darkgreen	Recliner	107
Peter Adams	RC831	jade	Recliner	121

7.2.2.4 Query: 'List the customers who have bought a type of chair in an apple sort of colour'

```
select CustomerName, product.PartID, Colour, ItemType,
sales.InvoiceNo from customers, product, sales, salesitem
where Colour CLOSETO "apple" and ItemType CLOSETO "chair" and
customers.CustomerID = sales.CustomerID and
sales.InvoiceNo = salesitem.InvoiceNo and
product.PartID = salesitem.PartID
```

CustomerName	PartID	Colour	ItemType	InvoiceNo
Keith Myers	KC020	lime	KitchenStool	101
Keith Myers	IC002	lime	ClericalChair	103
Barbara Lincoln	EG456	emerald	ErgonomicChair	104
Garry Cronin	SC125	seagreen	BarStool	105
Barry de Veen	RC444	lightgreen	Recliner	106
Barry de Veen	KC021	apple	KitchenStool	110
Keith Myers	SC125	seagreen	BarStool	112
Catherine Hartstein	EG456	emerald	ErgonomicChair	119
Peter Adams	RC831	jade	Recliner	121
Garry Cronin	DC023	chartreuse	DiningChair	124
Shirley Hetherington	IC002	lime	ClericalChair	125

7.2.3 Domain Perception Change

Example: The attribute *product.Colour* is coded as a string of the hexadecimal RGB values of a shade of colour. The hexadecimal strings are added to the domain COLOURS but there is no conversion of existing records in the relation *product*. Existing word terms for colours must be perceived to be the same as the hexadecimal values.

7.2.3.1 Query: 'Which products are a shade of green?' and 'Which products are like colour #008000?' are perceived to be the same and therefore return the same record set.

```
select * from product where Colour closeto "green"
```

PartID	ItemType	Colour	SupplierID	Price
IC001	ClericalChair	olive	2002	150.00
IC002	ClericalChair	lime	2002	150.00
EG123	ErgonomicChair	green	2001	60.00
EG456	ErgonomicChair	emerald	2001	60.00
DC001	DiningChair	green	2007	90.00
DC023	DiningChair	chartreuse	2007	96.00

PartID	ItemType	Colour	SupplierID	Price
DC510	DiningChair	jade	2006	126.00
SC345	BarStool	turquoise	2005	46.00
SC125	BarStool	seagreen	2005	55.00
RC831	Recliner	jade	2006	250.00
RC444	Recliner	lightgreen	2006	250.00
RC234	Recliner	darkgreen	2006	250.00
KC020	KitchenStool	lime	2008	40.00
KC021	KitchenStool	apple	2008	40.00
LC040	Sofa	aqua	2004	650.00
LC551	Lounge3seater	verdigris	2007	899.00
DT345	DiningTable	turquoise	2005	500.00
CT831	CoffeeTable	jade	2005	50.00

```
select * from product where Colour closeto "#008000"
```

PartID	ItemType	Colour	SupplierID	Price
IC001	ClericalChair	olive	2002	150.00
IC002	ClericalChair	lime	2002	150.00
EG123	ErgonomicChair	green	2001	60.00
EG456	ErgonomicChair	emerald	2001	60.00
DC001	DiningChair	green	2007	90.00
DC023	DiningChair	chartreuse	2007	96.00
DC510	DiningChair	jade	2006	126.00
SC345	BarStool	turquoise	2005	46.00
SC125	BarStool	seagreen	2005	55.00
RC831	Recliner	jade	2006	250.00
RC444	Recliner	lightgreen	2006	250.00
RC234	Recliner	darkgreen	2006	250.00
KC020	KitchenStool	lime	2008	40.00
KC021	KitchenStool	apple	2008	40.00
LC040	Sofa	aqua	2004	650.00
LC551	Lounge3seater	verdigris	2007	899.00

PartID	ItemType	Colour	SupplierID	Price
DT345	DiningTable	turquoise	2005	500.00
CT831	CoffeeTable	jade	2005	50.00

7.2.4 Domain Constraints Change

The domain constraints, that is the possible range of valid values, have changed for the domains COLOURS, as noted in the previous example, as well as for the domain CATEGORIES, for which *product.ItemTypes* are now classified at different levels of the domain hierarchy. New records reflecting these constraints changes are added to the relation *product*. The utilisation of a mesodata domain requires no change to queries to retrieve records containing the new terms.

7.2.4.1 Query: 'List our product range of tables.'

```
select * from product where ItemType closeto "table"
```

PartID	ItemType	Colour	SupplierID	Price
DT345	DiningTable	turquoise	2005	500.00
DT125	DiningTable	white	2005	355.00
CT831	CoffeeTable	jade	2005	50.00
CT444	CoffeeTable	darkblue	2005	59.00
WT450	table	#A52A2A	2009	167.95
WT451	table	#7B3F00	2009	167.95
WT452	table	#000000	2009	167.95
WT453	table	#63A671	2009	167.95

7.2.4.2 Query: 'List our product range of tables that are a shade of brown.'

```
select * from product where ItemType closeto "table" and Colour
closeto "brown"
```

PartID	ItemType	Colour	SupplierID	Price
WT450	table	#A52A2A	2009	167.95
WT451	table	#7B3F00	2009	167.95

7.2.5 Data Integration

The database now has two formats for recording the Colour of a product, the name of the colour and the hexadecimal RGB string, as well as different terms for the categories of product. By using mesodata domains to store the relationships between these different terms, there is no call for the conversion from one protocol to another. Records related to the different standards for Colour and ItemType are added to the relations *sales* and *salesitem*.

7.2.5.1 Query: 'List the invoices for sales of any of our tables'

```
select sales.InvoiceNo, product.PartID, Colour, ItemType from
sales, salesitem, product where salesitem.InvoiceNo =
sales.InvoiceNo and salesitem.PartID = product.PartID and ItemType
closeto "table"
```

InvoiceNo	PartID	Colour	ItemType
109	DT345	turquoise	DiningTable
116	DT345	turquoise	DiningTable
118	DT125	white	DiningTable
123	CT831	jade	CoffeeTable
131	WT452	#000000	table
133	CT831	jade	CoffeeTable

7.2.6 Attribute Representation Change

The product PartIDs now have number codes instead of character codes. In order to retain the original historical codes and avoid data conversion and application code changes, a new mesodata domain is created to provide the synonymous new numeric code for each character code. The attribute *product.PartID* is modified to refer to the new mesodata domain. There is no data loss as the attribute retains its original specification. Alternatively, the attribute can be changed to refer to the integer field of the mesodata type. This then entails replacing character codes with numeric codes in the product relation. In either case, queries may now refer to either the old or the new codes.

7.2.6.1 Query: Which product has the new code 1444?

```
select * from product where PartID equalto 1444
```

PartID	ItemType	Colour	SupplierID	Price
CT444	CoffeeTable	darkblue	2005	59.00

7.2.6.2 Query: Which products have either the codes 5001 or IC004?

```
select SupplierID, PartID, ItemType from product where PartID
equalto 5001 or PartID = ‘ic004’
```

SupplierID	PartID	ItemType
2002	IC001	ClericalChair
2002	IC004	IndustrialChair

7.3 Summary of Evaluation

The prototype model of a three-level database containing a mesodata layer met the functional criteria for the suggested extensions to Data Definition Language and Data Manipulation Language. The prototype implements three of the possible mesodata type structures, by which it demonstrates that mesodata domains enable enhanced querying, as well as facilitating data integration and attribute evolution.

This implementation of a prototype model provides empirical evidence that the utilisation of mesodata types endows relational databases with features that to date have required conversion of data, loss of information, schema evolution, schema translation, and rewriting application code. Additionally, the example queries show that both standard SQL and the suggested extensions to SQL are compatible and that ad hoc queries are possible over domains with heterogeneous data values. The use of an existing relational database platform demonstrates backwards compatibility of the mesodata architecture.

Chapter 8

Conclusions and Further Research

The introduction of databases for data storage and handling revolutionised the way we dealt with records and enabled simple and fast information processing, aggregation and summarisation. Database and information technology systems have evolved from simple file processing systems to powerful database systems. Data management technology has progressed from hierarchical and network systems to relational databases, data modelling tools and indexing and organisational techniques. The development of Relational Database Management Systems and automated systems put the layout and *form* into the unchanging metadata and gave us *record once* systems.

8.1 Database Evolution

Unfortunately, the 'real world' upon which databases are modelled constantly changes. These changes may affect the schema, as described by Sjøberg (1993), Roddick et al. (1999), Comyn-Wattiau et al. (2003), as well as this project's study presented in Chapter 6, for a variety of reasons including:

- Unanticipated requirements - all the desired functionality of a system may not be known in advance.
- A change in the universe of discourse - new or changed regulations, features or facts may need to be removed or accommodated in the system.

- A change to the interpretation of facts about the universe of discourse and the manner in which the task is realised in a system.
- Changes in the form of updates to effect upgrades to the functionality or scope of a system.
- Changes in the form of updates to effect efficiency improvements.
- Changes caused by system operation. For example, the discovery of new information which is then fed back into the system or the abnormal behaviour of a component.
- Error correction.

8.2 Techniques for Database Evolution

Different formalisms have been developed to deal with schema changes with the aim being to preserve information capacity and preserve semantic correctness. Schematic changes may be the result of evolving one system or may arise due to the need for merging two or more systems. For whichever reason, conflicts occur, such as naming, type, domain and cardinality, which must be resolved and the schemata unified to produce a new version. To reach this goal there are graph based *schema integration* architectures, for example the Common Data Model (CDM), the Schema Intension Graph (SIG), Hypergraph Data Model (HDM) and EVolutionary ER diagrams (EVER), as well as, semi-automatic systems applying *schema matching* and *schema translation* techniques, such as TSIMMIS, MIX, MADS and others. These systems also utilise ontologies, thesauri, and so forth to integrate data from heterogeneous sources in order to process queries and views.

8.3 Data Integration

Data integration or conversion remains a partially resolved issue. Metadata changes that move attributes from one relation to another or delete attributes from a relation are managed by changes to application code and system down time for conversion procedures. However an attribute change, while in itself may not be a complex process, is not a trivial task. This type of change may result in data loss, changed accuracy, and altered semantics.

Whilst the use of ontologies, concept graphs and other knowledge interchange techniques are alleviating the problems of data integration, these structures are not yet an integral part of the database architecture. A mesodata layer, separate from the metadata and data, provides complex structures in which to store domain values and their inter-relationships as well as supplying the 'intelligence' required to operate and manipulate them. The domain structures enable different orderings that form the bases of filters for enhanced querying and information retrieval. DBMS supplied mesodata types would allow for the re-usable inclusion of domain information such as in ontologies, taxonomies, thesauri and concept graphs that to date have been only application specific.

8.4 Mesodata Layer

This thesis suggests that common domain structures, such as graphs, queues, circular lists, and so on, if available as mesodata, complete with appropriate operations and DBMS support, would both simplify and enhance database modelling. The inclusion of a mesodata layer in the definition of attribute domains introduces

- more flexibility into the modelling process,
- promotes re-use of domains,
- increases the level of abstraction,
- simplifies the implementation of schema change,
- facilitates attribute evolution, and
- enables enhanced querying.

8.5 Future Research

8.5.1 DB Platform Support for Mesodata

As part of the development of the prototype model, extensions to SQL have been suggested to provide support for a mesodata layer, the implemented mesodata types, as well as for other suggested complex structures. Further research is necessary to enable fully database platform support for a mesodata layer.

8.5.2 XML

A path not followed was the investigation of XML to build mesodata types. The main obstacle being that the language did not provide a way to express the algorithms required to construct the mesodata types with ‘intelligence’. The mesodata types, themselves, are easily described, there is, however, no provision for the inclusion of procedures for the traversal and manipulation of the structures that are necessary. Development of this feature would allow for user-defined structures to be used as well as platform provided ones.

8.5.3 Ontologies of Data Structures

Somewhat related to the previous point, there are of course many domain structures that have not been researched and described in this thesis. Further research and exploration of other useful domain structures could provide ontologies for data structures that include algorithms/logic for their use.

8.5.4 Mesodata types based on UDTs

All mesodata types researched in this project are based on simple data types. The user-defined types in object-relational databases incorporated into domains structures would provide even more flexibility into data modelling.

8.5.5 Modelling Tools

In order to incorporate a third layer into a database, modelling tools need to be developed to both design new systems, as well as to re-engineer evolving systems.

8.5.6 Other Database Technologies

Incorporation of a mesodata layer into data warehousing, data mining, and knowledge base technologies where reducing data loss and expanding information capacity also augurs well, should provide for very interesting future research.

Appendix A

Publications Resulting From This Thesis

The following conference papers have been published as a result of work associated with this thesis.

Roddick, J. F., K. Hornsby, and D. de Vries (2003). 'A Unifying Semantic Distance Model for Determining the Similarity of Attribute Values'. 26th Australasian Computer Science Conference (ACSC2003), Adelaide, Australia, ACS.

Abstract: The relative difference between two data values is of interest in a number of application domains including temporal and spatial applications, schema versioning, data warehousing (particularly data preparation), internet searching, validation and error correction, and data mining. Moreover, consistency across systems in determining such distances and the robustness of such calculations is essential in some domains and useful in many. Despite this, there is no generally adopted approach to determining such distances and no accommodation of distance within SQL or any commercially available DBMS. For non-numeric data values calculating the difference between values often requires application specific support but even for numeric values the practical distance between two values may not simply be their numeric difference or Euclidean distance. In this paper, a model of semantic distance is developed in which a graph-based approach is used to quantify the distance between two data values. The approach facilitates a notion of distance, both as a simple traversal distance and as weighted arcs. Transition costs, as an additional expense of passing through a node, are also accommodated. Furthermore, multiple distance measures can be incorporated and a method of 'localisation' is discussed which allows relevant information to take precedence over less relevant information. Some results from our investigations, including our SQL based implementation, are presented.

de Vries, D., S. Rice, and J. F. Roddick (2004). 'In Support of Mesodata in Database Management Systems'. 15th International Conference on Database and Expert Systems Applications DEXA 2004, Zaragoza, Spain, Springer-Verlag.

Abstract: In traditional relational database modelling there is a strict separation between the definition of the relational schema and the data itself. This simple two level architecture works well when the domains over which attributes are required to be defined are relatively simple. However, in cases where attributes need to be defined over more complex domain structures, such as graphs, hierarchies, circular lists and so on, the aggregation of domain and relational definition becomes confused and a separation of the specification of domain definition from relational structure is appropriate. This aggregation of domain definition with relational structure also occurs in XMLS and ontology definitions. In this paper we argue for a three level architecture when considering the design and development of domains for relational and semi-structured data models. The additional level facilitating more complete domain definition - mesodata - allows domains to be engineered so that attributes can be defined to possess additional intelligence and structure and thus reflect more accurately ontological considerations. We argue that the embedding of this capability within the modelling process augments, but lies outside of, current schema definition methods and thus is most appropriately considered separately.

de Vries, D. and J. F. Roddick (2004). 'Facilitating Database Attribute Domain Evolution Using Mesodata'. Third International Workshop on Evolution and Change in Data Management (ECDM2004), Shanghai, China, Springer-Verlag.

Abstract: Database evolution can be considered a combination of schema evolution, in which the structure evolves with the addition and deletion of attributes and relations, together with domain evolution in which an attribute's specification, semantics and/or range of allowable values changes. We present a model in which mesodata - an additional domain definition layer containing domain structure and intelligence - is used to alleviate and in some cases obviate the need for data conversion or coercion. We present the nature and use of mesodata as it affects domain evolution, such as when a domain changes, when the semantics of a domain alter and when the attribute's specification is modified.

Mooney, C. H., D. De Vries, and J. F. Roddick (2005). 'A Multi-level Framework for the Analysis of Sequential Data'. Data Mining: Theory, Methodology, Techniques, and Applications. S. J. Simoff and G. J. Williams, Springer-Verlag.

Abstract: Traditionally text mining has had a strong link with information retrieval and classification, for search engine purposes, and has aimed to classify documents according to known knowledge. Association rule mining and sequence mining on the other hand have had a different goal; one of eliciting relationships within or about the data being mined. Recently there has been some research conducted using sequence mining techniques on digital document collections by treating the text as sequential data.

In this paper we propose a multi-level framework that is applicable to text analysis and that improves the knowledge discovery process by finding additional or hitherto unknown relationships within the data being mined. We believe that this can lead to the detection or fine tuning of the context of documents under consideration and may lead to a more informed classification of those documents. Moreover, since we use a semantic map at varying stages in the framework, we are able to impose a greater degree of focus and therefore a greater transitivity of semantic relatedness that facilitates the improvement in the knowledge discovery process.

Rice, S., J. F. Roddick, and D. de Vries (2006). 'Defining and Implementing Domains with Multiple Types using Mesodata Modelling Techniques'. 3rd Asia-Pacific Conference on Conceptual Modelling (APCCM 2006), Hobart, Australia, January 16-19 2006, ACS. [to be published]

Abstract: The integration of data from different sources often leads to the adoption of schemata that entails a loss of information in respect of one or more of the data sets being combined. The coercion of data to conform to the type of the unified attribute is one of the major reasons for this information loss. We argue that for maximal information retention it would be useful to be able to define attributes over domains capable of accommodating *multiple types*, that is, domains that potentially allow an attribute to take its values from more than one base type.

Mesodata is a concept that provides an intermediate conceptual layer between the definition of a relational structure and that of attribute definition to aid the specification of complex domain structures within the database. Mesodata modelling techniques involve the use of data types and operations for common data structures defined in the mesodata layer to facilitate accurate modelling of complex data domains, so that any commonality between similar domains used for different purposes can be exploited.

This paper shows how the mesodata concept can be extended to facilitate the creation of domains defined over multiple base types, and also allow the same set of base values to be used for domains with different semantics. Using an example domain containing values representing three different type of incomplete knowledge about the data item (coarse granularity, vague terms, or intervals) we show how operations and data structures for types already existing within the mesodata can simplify the task of developing a new *intelligent domain.*

Appendix B

Sample Session

```
Enter database name
> tritier

Enter Login name

Enter password

Enter SQL Statement or quit to finish

> show catalogue                                                    (1)

Tables in tritier
DOMTABLE
MESOTABLE

Enter SQL Statement or quit to finish

>  CREATE TABLE colourgraph (NodeI char(15) NOT NULL , NodeJ char(15)
NOT NULL, Distance float NOT NULL, primary key (NodeI, nodeJ), index
(NodeI), index (NodeJ))                                             (2)

Enter SQL Statement or quit to finish

>  source adjColours.sql                                            (3)

Enter SQL Statement or quit to finish

>  select * from colourgraph                                        (4)

NodeI        NodeJ        Distance
white        lightblue    0.4
lightblue    darkblue     0.2
lightblue    aqua         0.15
white        lightgreen   0.4
lightgreen   aqua         0.2
lightgreen   green        0.15
lightgreen   lime         0.15
white        yellow       0.4
yellow       lime         0.4
yellow       lightyellow  0.2
yellow       orange       0.3
lightblue    lightyellow  0.35
lightblue    lightgreen   0.35
olive        green        0.15
emerald      green        0.15
chartreuse   lime         0.1
jade         green        0.1
turquoise    aqua         0
seagreen     aqua         0.1
darkgreen    green        0.2
apple        lime         0
verdigris    lightgreen   0.1
olive        brown        0.25
brown        orange       0.3
orange       burntorange  0.15
chestnut     brown        0.15
aqua         teal         0.18
darkblue     navy         0.1
navy         black        0.4

Enter SQL Statement or quit to finish

>  CREATE TABLE categorygraph (NodeI char(20) NOT NULL , NodeJ char(20)
NOT NULL, Distance float NOT NULL, primary key (NodeI, nodeJ), index
(NodeI), index (NodeJ))                                                (5)

Enter SQL Statement or quit to finish

>  show catalogue                                                      (6)

Tables in tritier
DOMTABLE
MESOTABLE
categorygraph
colourgraph

Enter SQL Statement or quit to finish

>  source categories.sql                                               (7)

Enter SQL Statement or quit to finish

>  select * from categorygraph                                         (8)

NodeI        NodeJ          Distance
chair        table          1
chair        domchair       0.25
chair        officechair    0.25
officechair  clericalchair  0
officechair  ergonomicchair 0
```

```
domchair     diningchair    0.1
officechair  barstool       0.1
domchair     recliner       0.1
domchair     kitchenstool   0.1
domchair     sofa           0.1
domchair     lounge3seater  0.1
table        diningtable    0
table        coffeetable    0
table        sidetable      0
table        changetable    0
table        boardroomtable 0

Enter SQL Statement or quit to finish

>  CREATE domain COLOURS as wgraph of char(15) over colourgraph      (9)
COLOURS created

Enter SQL Statement or quit to finish

>  show catalogue                                                   (10)

Tables in tritier
COLOURS
DOMTABLE
MESOTABLE
categorygraph
colourgraph

Enter SQL Statement or quit to finish

>  show domains                                                     (11)

DOMID DOM_NAME  MESODATA_TYPE  BASE_TYPE  RELATION_NAME
1     COLOURS   wgraph         char(15)   colourgraph

Enter SQL Statement or quit to finish

>  create domain CATEGORIES as dwgraph of char(20) over categorygraph
                                                                    (12)

CATEGORIES created

Enter SQL Statement or quit to finish

>  show domains                                                     (13)

DOMID DOM_NAME   MESODATA_TYPE   BASE_TYPE   RELATION_NAME
1     COLOURS    wgraph          char(15)    colourgraph
2     CATEGORIES dwgraph         char(20)    categorygraph

Enter SQL Statement or quit to finish
```

```
>  show catalogue                                                    (14)

Tables in tritier
CATEGORIES
COLOURS
DOMTABLE
MESOTABLE
categorygraph
colourgraph

Enter SQL Statement or quit to finish

>  Alter domain COLOURS name SHADES                                  (15)

Enter SQL Statement or quit to finish

>  show domains                                                      (16)

DOMID DOM_NAME   MESODATA_TYPE   BASE_TYPE   RELATION_NAME
1     SHADES     wgraph          char(15)    colourgraph
2     CATEGORIES dwgraph         char(20)    categorygraph

Enter SQL Statement or quit to finish

>  show catalogue                                                    (17)

Tables in tritier
CATEGORIES
DOMTABLE
MESOTABLE
SHADES
categorygraph
colourgraph


Enter SQL Statement or quit to finish

>  desc SHADES                                                       (18)

Field    Type     Collation                Key
nodeI    char(15) latin1_swedish_ci        PRI
nodeJ    char(15) latin1_swedish_ci        PRI
distance float    binary

Enter SQL Statement or quit to finish

>  Alter domain SHADES MESODATATYPE dwgraph                          (19)


Enter SQL Statement or quit to finish
```

```
>  show domains                                                     (20)

DOMID  DOM_NAME    MESODATA_TYPE  BASE_TYPE  RELATION_NAME
1      SHADES      dwgraph        char(15)   colourgraph
2      CATEGORIES  dwgraph        char(20)   categorygraph

Enter SQL Statement or quit to finish

>  refresh domain SHADES                                            (21)

Enter SQL Statement or quit to finish

>  alter domain SHADES BASETYPE char(25)                            (22)

Enter SQL Statement or quit to finish

>  show domains                                                     (23)

DOMID  DOM_NAME    MESODATA_TYPE  BASE_TYPE  RELATION_NAME
1      SHADES      dwgraph        char(25)   colourgraph
2      CATEGORIES  dwgraph        char(20)   categorygraph

Enter SQL Statement or quit to finish

>  desc SHADES                                                      (24)

Field    Type     Collation          Key
nodeI    char(25) latin1_swedish_ci  PRI
nodeJ    char(25) latin1_swedish_ci  PRI
distance float     binary

Enter SQL Statement or quit to finish

>  CREATE TABLE shadesgraph (NodeI char(30) NOT NULL , NodeJ char(30)
NOT NULL, Distance float NOT NULL, primary key (NodeI, nodeJ), index
(NodeI), index (NodeJ))                                             (25)

Enter SQL Statement or quit to finish

>  show catalogue                                                   (26)

Tables in tritier
CATEGORIES
DOMTABLE
MESOTABLE
SHADES
categorygraph
colourgraph
shadesgraph

Enter SQL Statement or quit to finish
```

```
>  source shadescolours.sql                                          (27)

Enter SQL Statement or quit to finish

>  alter domain SHADES OVER shadesgraph                              (28)

Enter SQL Statement or quit to finish

>  show domains                                                      (29)

DOMID  DOM_NAME    MESODATA_TYPE  BASE_TYPE  RELATION_NAME
1      SHADES      dwgraph        char(25)   shadesgraph
2      CATEGORIES  dwgraph        char(20)   categorygraph

Enter SQL Statement or quit to finish

>  drop domain SHADES                                                (30)

Enter SQL Statement or quit to finish

>  show catalogue                                                    (31)

Tables in tritier
CATEGORIES
DOMTABLE
MESOTABLE
categorygraph
colourgraph
shadesgraph

Enter SQL Statement or quit to finish

>  CREATE domain COLOURS as wgraph of char(15) over colourgraph     (32)
COLOURS created

Enter SQL Statement or quit to finish

>  show domains                                                      (33)

DOMID DOM_NAME    MESODATA_TYPE  BASE_TYPE  RELATION_NAME
3     COLOURS     wgraph         char(15)   colourgraph
2     CATEGORIES  dwgraph        char(20)   categorygraph

Enter SQL Statement or quit to finish

>  show catalogue                                                    (34)

Tables in tritier
CATEGORIES
COLOURS
DOMTABLE
MESOTABLE
categorygraph
colourgraph
shadesgraph

Enter SQL Statement or quit to finish
```

```
>  CREATE TABLE product (PartID char(5) NOT NULL, ItemType CATEGORIES,
Colour COLOURS, SupplierID int, Price numeric(9), PRIMARY KEY (PartID))
                                                                   (35)

Enter SQL Statement or quit to finish

>  show catalogue                                                  (36)

Tables in tritier
CATEGORIES
COLOURS
DOMTABLE
MESOTABLE
categorygraph
colourgraph
product
shadesgraph

Enter SQL Statement or quit to finish

>  show domains                                                    (37)

DOMID  DOM_NAME    MESODATA_TYPE  BASE_TYPE  RELATION_NAME
3      COLOURS     wgraph         char(15)   colourgraph
2      CATEGORIES  dwgraph        char(20)   categorygraph

Enter SQL Statement or quit to finish

>  show mesodatatypes                                              (38)

MTID  TABLE_NAME  FIELD_NAME  DOMID
1     product     ItemType    2
2     product     Colour      3

Enter SQL Statement or quit to finish

>  desc product                                                    (39)

Field       Type          Collation          Key
PartID      char(5)       latin1_swedish_ci  PRI
ItemType    char(20)      latin1_swedish_ci
Colour      char(15)      latin1_swedish_ci
SupplierID  int(11)       binary
Price       decimal(9,0)  binary

Enter SQL Statement or quit to finish

>  CREATE TABLE suppliers (SupplierID int NOT NULL , SupplierName
char(30), PRIMARY KEY (SupplierID))                                (40)
```

```
Enter SQL Statement or quit to finish

>  CREATE TABLE customers (CustomerID int NOT NULL, CustomerName
char(30), PRIMARY KEY (CustomerID))                                  (41)

Enter SQL Statement or quit to finish

>  CREATE TABLE sales (InvoiceNo int NOT NULL, InvDate date, CustomerID
int, PRIMARY KEY (InvoiceNo))                                        (42)

Enter SQL Statement or quit to finish

>  CREATE TABLE salesitem (SalesItem int NOT NULL, InvoiceNo int,
PartID char(5), Quantity numeric(5), PRIMARY KEY(SalesItem))         (43)

Enter SQL Statement or quit to finish

>  alter table product modify Price numeric(9,2)                     (44)

Enter SQL Statement or quit to finish

>  desc product                                                      (45)

Field       Type         Collation          Key
PartID      char(5)      latin1_swedish_ci  PRI
ItemType    char(20)     latin1_swedish_ci
Colour      char(15)     latin1_swedish_ci
SupplierID  int(11)       binary
Price       decimal(9,2)  binary

Enter SQL Statement or quit to finish

>  alter table product add column Instock numeric(4)                 (46)

Enter SQL Statement or quit to finish

>  desc product                                                      (47)

Field       Type          Collation          Key
PartID      char(5)       latin1_swedish_ci  PRI
ItemType    char(20)      latin1_swedish_ci
Colour      char(15)      latin1_swedish_ci
SupplierID  int(11)       binary
Price       decimal(9,2)  binary
Instock     decimal(4,0)  binary

Enter SQL Statement or quit to finish

>  alter table product drop column Instock                           (48)

Enter SQL Statement or quit to finish

>  desc product                                                      (49)

Field       Type         Collation          Key
PartID      char(5)      latin1_swedish_ci  PRI
```

```
ItemType    char(20)     latin1_swedish_ci
Colour      char(15)     latin1_swedish_ci
SupplierID  int(11)      binary
Price       decimal(9,2) binary

Enter SQL Statement or quit to finish

>  alter table product add column category CATEGORIES                (50)

Enter SQL Statement or quit to finish

>  desc product                                                      (51)

Field       Type          Collation          Key
PartID      char(5)       latin1_swedish_ci  PRI
ItemType    char(20)      latin1_swedish_ci
Colour      char(15)      latin1_swedish_ci
SupplierID  int(11)       binary
Price       decimal(9,2)  binary
category    char(5)       latin1_swedish_ci

Enter SQL Statement or quit to finish

>  show mesodatatypes                                                (52)

MTID  TABLE_NAME  FIELD_NAME  DOMID
1     product     ItemType    2
2     product     Colour      3
3     product     category    2

Enter SQL Statement or quit to finish

>  alter table product modify category char(6)                       (53)

Enter SQL Statement or quit to finish

>  desc product                                                      (54)

Field       Type          Collation          Key
PartID      char(5)       latin1_swedish_ci  PRI
ItemType    char(20)      latin1_swedish_ci
Colour      char(15)      latin1_swedish_ci
SupplierID  int(11)       binary
Price       decimal(9,2)  binary
category    char(6)       latin1_swedish_ci

Enter SQL Statement or quit to finish

>  show mesodatatypes                                                (55)
```

```
MTID  TABLE_NAME  FIELD_NAME  DOMID
1     product     ItemType    2
2     product     Colour      3

Enter SQL Statement or quit to finish

>  alter table product drop column category                        (56)

Enter SQL Statement or quit to finish

>  desc product                                                    (57)

Field       Type          Collation          Key
PartID      char(5)       latin1_swedish_ci  PRI
ItemType    char(20)      latin1_swedish_ci
Colour      char(15)      latin1_swedish_ci
SupplierID  int(11)       binary
Price       decimal(9,2)  binary

Enter SQL Statement or quit to finish

>  show mesodatatypes                                              (58)

MTID  TABLE_NAME  FIELD_NAME  DOMID
1     product     ItemType    2
2     product     Colour      3

Enter SQL Statement or quit to finish

> source furnitureB.sql                                            (59)

Enter SQL Statement or quit to finish

>  select * from product                                           (60)

PartID  ItemType                Colour          SupplierID  Price
IC001   ClericalChair           olive                 2002  150.00
IC002   ClericalChair           lime                  2002  150.00
EG123   ErgonomicChair          green                 2001   60.00
EG456   ErgonomicChair          emerald               2001   60.00
DC001   DiningChair             green                 2007   90.00
DC023   DiningChair             chartreuse            2007   96.00
DC510   DiningChair             jade                  2006  126.00
SC345   BarStool                turquoise             2005   46.00
SC125   BarStool                seagreen              2005   55.00
RC831   Recliner                jade                  2006  250.00
RC444   Recliner                lightgreen            2006  250.00
RC234   Recliner                darkgreen             2006  250.00
KC020   KitchenStool            lime                  2008   40.00
KC021   KitchenStool            apple                 2008   40.00
```

```
LC040    Sofa                    aqua                    2004   650.00
LC551    Lounge3seater           verdigris               2007   899.00
IC003    IndustrialChair         white                   2002   150.00
IC004    IndustrialChair         lightyellow             2002   150.00
EG120    ErgonomicChair          orange                  2008    60.00
EG453    ErgonomicChair          lightblue               2008    60.00
DC004    DiningChair             white                   2006    90.00
DC026    DiningChair             darkblue                2006    96.00
DC512    DiningChair             white                   2006   126.00
DT345    DiningTable             turquoise               2005   500.00
DT125    DiningTable             white                   2005   355.00
CT831    CoffeeTable             jade                    2005    50.00
CT444    CoffeeTable             darkblue                2005    59.00

Enter SQL Statement or quit to finish

>  select * from product where Colour = "green"                       (61)

PartID   ItemType             Colour          SupplierID  Price
EG123    ErgonomicChair       green                 2001   60.00
DC001    DiningChair          green                 2007   90.00

Enter SQL Statement or quit to finish

>  select * from product where Colour closeto "green"                 (62)

PartID   ItemType             Colour           SupplierID   Price
IC001    ClericalChair        olive                  2002   150.00
IC002    ClericalChair        lime                   2002   150.00
EG123    ErgonomicChair       green                  2001    60.00
EG456    ErgonomicChair       emerald                2001    60.00
DC001    DiningChair          green                  2007    90.00
DC023    DiningChair          chartreuse             2007    96.00
DC510    DiningChair          jade                   2006   126.00
SC345    BarStool             turquoise              2005    46.00
SC125    BarStool             seagreen               2005    55.00
RC831    Recliner             jade                   2006   250.00
RC444    Recliner             lightgreen             2006   250.00
RC234    Recliner             darkgreen              2006   250.00
KC020    KitchenStool         lime                   2008    40.00
KC021    KitchenStool         apple                  2008    40.00
LC040    Sofa                 aqua                   2004   650.00
LC551    Lounge3seater        verdigris              2007   899.00
DT345    DiningTable          turquoise              2005   500.00
CT831    CoffeeTable          jade                   2005    50.00

Enter SQL Statement or quit to finish

>  select * from product where Colour = "green" and itemtype =
"recliner"                                                            (63)
```

```
PartID   ItemType   Colour  SupplierID  Price

Enter SQL Statement or quit to finish

>  select * from product where Colour closeto "green" and itemtype =
"recliner"                                                       (64)

PartID   ItemType            Colour         SupplierID     Price
RC831    Recliner            jade                 2006    250.00
RC444    Recliner            lightgreen           2006    250.00
RC234    Recliner            darkgreen            2006    250.00

Enter SQL Statement or quit to finish

>  select * from product where Colour closeto "green" and itemtype
closeto "chair"                                                  (65)

PartID   ItemType            Colour         SupplierID  Price
IC001    ClericalChair       olive              2002   150.00
IC002    ClericalChair       lime               2002   150.00
EG123    ErgonomicChair      green              2001    60.00
EG456    ErgonomicChair      emerald            2001    60.00
DC001    DiningChair         green              2007    90.00
DC023    DiningChair         chartreuse         2007    96.00
DC510    DiningChair         jade               2006   126.00
SC345    BarStool            turquoise          2005    46.00
SC125    BarStool            seagreen           2005    55.00
RC831    Recliner            jade               2006   250.00
RC444    Recliner            lightgreen         2006   250.00
RC234    Recliner            darkgreen          2006   250.00
KC020    KitchenStool        lime               2008    40.00
KC021    KitchenStool        apple              2008    40.00
LC040    Sofa                aqua               2004   650.00
LC551    Lounge3seater       verdigris          2007   899.00

Enter SQL Statement or quit to finish

>  source customers.sql                                          (66)

Enter SQL Statement or quit to finish

>  source suppliers.sql                                          (67)

Enter SQL Statement or quit to finish

>  source sales.sql                                              (68)

Enter SQL Statement or quit to finish

>  source salesitem.sql                                          (69)

Enter SQL Statement or quit to finish
```

```
>  select CustomerName, product.PartID, Colour, sales.InvoiceNo from
customers, product, sales, salesitem where Colour = "apple" and
customers.CustomerID = sales.CustomerID and sales.InvoiceNo =
salesitem.InvoiceNo and product.PartID = salesitem.PartID          (70)

CustomerName                 PartID   Colour         InvoiceNo
Barry de Veen                KC021    apple              110

Enter SQL Statement or quit to finish

>  select CustomerName, product.PartID, Colour, sales.InvoiceNo from
customers, product, sales, salesitem where Colour CLOSETO "apple" and
customers.CustomerID = sales.CustomerID and sales.InvoiceNo =
salesitem.InvoiceNo and product.PartID = salesitem.PartID          (71)

CustomerName                 PartID   Colour         InvoiceNo
Keith Myers                  KC020    lime               101
Keith Myers                  IC002    lime               103
Barbara Lincoln              EG456    emerald            104
Garry Cronin                 SC125    seagreen           105
Barry de Veen                RC444    lightgreen         106
Barbara Lincoln              DT345    turquoise          109
Barry de Veen                KC021    apple              110
Keith Myers                  SC125    seagreen           112
Johanna Baker                DT345    turquoise          116
Catherine Hartstein          EG456    emerald            119
Peter Adams                  RC831    jade               121
Stella Shepherd              CT831    jade               123
Garry Cronin                 DC023    chartreuse         124
Shirley Hetherington         IC002    lime               125

Enter SQL Statement or quit to finish

>  select CustomerName, product.PartID, ItemType, Colour,
sales.InvoiceNo from customers, product, sales, salesitem where
customers.CustomerID = sales.CustomerID and sales.InvoiceNo =
salesitem.InvoiceNo and product.PartID = salesitem.PartID          (72)

CustomerName           PartID ItemType          Colour       InvoiceNo
Keith Myers            KC020  KitchenStool      lime              101
Denise Devine          IC004  IndustrialChair   lightyellow       102
Keith Myers            IC002  ClericalChair     lime              103
Barbara Lincoln        EG456  ErgonomicChair    emerald           104
Garry Cronin           SC125  BarStool          seagreen          105
Barry de Veen          RC444  Recliner          lightgreen        106
John Haggar            RC234  Recliner          darkgreen         107
Lee Provins            EG453  ErgonomicChair    lightblue         108
Barbara Lincoln        DT345  DiningTable       turquoise         109
Barry de Veen          KC021  KitchenStool      apple             110
```

```
Stephen May            EG453  ErgonomicChair    lightblue         111
Keith Myers            SC125  BarStool          seagreen          112
Barry de Veen          IC004  IndustrialChair   lightyellow       113
Marion Cartwright      DC004  DiningChair       white             114
Denise Devine          EG453  ErgonomicChair    lightblue         115
Johanna Baker          DT345  DiningTable       turquoise         116
Ian Pill               EG120  ErgonomicChair    orange            117
John Haggar            DT125  DiningTable       white             118
Catherine Hartstein    EG456  ErgonomicChair    emerald           119
Stuart Barich          IC003  IndustrialChair   white             120
Peter Adams            RC831  Recliner          jade              121
Janis Jones            DC026  DiningChair       darkblue          122
Stella Shepherd        CT831  CoffeeTable       jade              123
Garry Cronin           DC023  DiningChair       chartreuse        124
Shirley Hetherington   IC002  ClericalChair     lime              125

Enter SQL Statement or quit to finish

>  select CustomerName, product.PartID, Colour, ItemType,
sales.InvoiceNo from customers, product, sales, salesitem where
ItemType = "Recliner" and customers.CustomerID = sales.CustomerID and
sales.InvoiceNo = salesitem.InvoiceNo and product.PartID =
salesitem.PartID                                                       (73)

CustomerName           PartID Colour         ItemType       InvoiceNo
Barry de Veen          RC444  lightgreen     Recliner           106
John Haggar            RC234  darkgreen      Recliner           107
Peter Adams            RC831  jade           Recliner           121

Enter SQL Statement or quit to finish

>  select CustomerName, product.PartID, Colour, ItemType,
sales.InvoiceNo from customers, product, sales, salesitem where Colour
CLOSETO "apple" and ItemType CLOSETO "chair" and customers.CustomerID =
sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and
product.PartID = salesitem.PartID                                      (74)

CustomerName           PartID Colour         ItemType       InvoiceNo
Keith Myers            KC020  lime           KitchenStool       101
Keith Myers            IC002  lime           ClericalChair      103
Barbara Lincoln        EG456  emerald        ErgonomicChair     104
Garry Cronin           SC125  seagreen       BarStool           105
Barry de Veen          RC444  lightgreen     Recliner           106
Barry de Veen          KC021  apple          KitchenStool       110
Keith Myers            SC125  seagreen       BarStool           112
Catherine Hartstein    EG456  emerald        ErgonomicChair     119
Peter Adams            RC831  jade           Recliner           121
Garry Cronin           DC023  chartreuse     DiningChair        124
Shirley Hetherington   IC002  lime           ClericalChair      125
```

```
Enter SQL Statement or quit to finish

>  source hexColours.sql                                                  (75)

Enter SQL Statement or quit to finish

>  refresh domain COLOURS                                                 (76)

Enter SQL Statement or quit to finish

>  select * from product where Colour = "#008000"                         (77)

PartID ItemType        Colour   SupplierID  Price

Enter SQL Statement or quit to finish

>  select * from product where Colour closeto "#008000"                   (78)

PartID ItemType        Colour   SupplierID  Price
IC001  ClericalChair   olive          2002  150.00
IC002  ClericalChair   lime           2002  150.00
EG123  ErgonomicChair  green          2001   60.00
EG456  ErgonomicChair  emerald        2001   60.00
DC001  DiningChair     green          2007   90.00
DC023  DiningChair     chartreuse     2007   96.00
DC510  DiningChair     jade           2006  126.00
SC345  BarStool        turquoise      2005   46.00
SC125  BarStool        seagreen       2005   55.00
RC831  Recliner        jade           2006  250.00
RC444  Recliner        lightgreen     2006  250.00
RC234  Recliner        darkgreen      2006  250.00
KC020  KitchenStool    lime           2008   40.00
KC021  KitchenStool    apple          2008   40.00
LC040  Sofa            aqua           2004  650.00
LC551  Lounge3seater   verdigris      2007  899.00
DT345  DiningTable     turquoise      2005  500.00
CT831  CoffeeTable     jade           2005   50.00

Enter SQL Statement or quit to finish

>  select * from product where Colour closeto "green"                     (79)

PartID ItemType        Colour   SupplierID  Price
IC001  ClericalChair   olive          2002  150.00
IC002  ClericalChair   lime           2002  150.00
EG123  ErgonomicChair  green          2001   60.00
EG456  ErgonomicChair  emerald        2001   60.00
DC001  DiningChair     green          2007   90.00
DC023  DiningChair     chartreuse     2007   96.00
DC510  DiningChair     jade           2006  126.00
```

```
SC345  BarStool        turquoise     2005   46.00
SC125  BarStool        seagreen      2005   55.00
RC831  Recliner        jade          2006  250.00
RC444  Recliner        lightgreen    2006  250.00
RC234  Recliner        darkgreen     2006  250.00
KC020  KitchenStool    lime          2008   40.00
KC021  KitchenStool    apple         2008   40.00
LC040  Sofa            aqua          2004  650.00
LC551  Lounge3seater   verdigris     2007  899.00
DT345  DiningTable     turquoise     2005  500.00
CT831  CoffeeTable     jade          2005   50.00

Enter SQL Statement or quit to finish

>  source furnitureC.sql                                                  (80)
Enter SQL Statement or quit to finish
>  select * from product                                                  (81)

PartID ItemType          Colour      SupplierID  Price
IC001  ClericalChair     olive           2002   150.00
IC002  ClericalChair     lime            2002   150.00
EG123  ErgonomicChair    green           2001    60.00
EG456  ErgonomicChair    emerald         2001    60.00
DC001  DiningChair       green           2007    90.00
DC023  DiningChair       chartreuse      2007    96.00
DC510  DiningChair       jade            2006   126.00
SC345  BarStool          turquoise       2005    46.00
SC125  BarStool          seagreen        2005    55.00
RC831  Recliner          jade            2006   250.00
RC444  Recliner          lightgreen      2006   250.00
RC234  Recliner          darkgreen       2006   250.00
KC020  KitchenStool      lime            2008    40.00
KC021  KitchenStool      apple           2008    40.00
LC040  Sofa              aqua            2004   650.00
LC551  Lounge3seater     verdigris       2007   899.00
IC003  IndustrialChair   white           2002   150.00
IC004  IndustrialChair   lightyellow     2002   150.00
EG120  ErgonomicChair    orange          2008    60.00
EG453  ErgonomicChair    lightblue       2008    60.00
DC004  DiningChair       white           2006    90.00
DC026  DiningChair       darkblue        2006    96.00
DC512  DiningChair       white           2006   126.00
DT345  DiningTable       turquoise       2005   500.00
DT125  DiningTable       white           2005   355.00
CT831  CoffeeTable       jade            2005    50.00
CT444  CoffeeTable       darkblue        2005    59.00
WT450  table             #A52A2A         2009   167.95
CC234  domchair          #009966         2009    85.50
WC117  officechair       #00C957         2009   250.00
```

```
WT451  table           #7B3F00         2009   167.95
WT452  table           #000000         2009   167.95
WT453  table           #63A671         2009   167.95
CC235  domchair        #000000         2009    85.50
CC236  domchair        #90EE90         2009    85.50
CC237  domchair        #008080         2009    85.50
WC118  officechair     #006400         2009   175.00
WC119  officechair     #7CFC00         2009   200.00
WC120  officechair     #7B3F00         2009   250.00

Enter SQL Statement or quit to finish

>  select * from product where Colour closeto "#90EE90"          (82)

PartID ItemType          Colour      SupplierID  Price
IC001  ClericalChair     olive           2002   150.00
IC002  ClericalChair     lime            2002   150.00
EG123  ErgonomicChair    green           2001    60.00
EG456  ErgonomicChair    emerald         2001    60.00
DC001  DiningChair       green           2007    90.00
DC023  DiningChair       chartreuse      2007    96.00
DC510  DiningChair       jade            2006   126.00
SC345  BarStool          turquoise       2005    46.00
SC125  BarStool          seagreen        2005    55.00
RC831  Recliner          jade            2006   250.00
RC444  Recliner          lightgreen      2006   250.00
RC234  Recliner          darkgreen       2006   250.00
KC020  KitchenStool      lime            2008    40.00
KC021  KitchenStool      apple           2008    40.00
LC040  Sofa              aqua            2004   650.00
LC551  Lounge3seater     verdigris       2007   899.00
IC003  IndustrialChair   white           2002   150.00
EG453  ErgonomicChair    lightblue       2008    60.00
DC004  DiningChair       white           2006    90.00
DC512  DiningChair       white           2006   126.00
DT345  DiningTable       turquoise       2005   500.00
DT125  DiningTable       white           2005   355.00
CT831  CoffeeTable       jade            2005    50.00
CC234  domchair          #009966         2009    85.50
WC117  officechair       #00C957         2009   250.00
WT453  table             #63A671         2009   167.95
CC236  domchair          #90EE90         2009    85.50
CC237  domchair          #008080         2009    85.50
WC118  officechair       #006400         2009   175.00
WC119  officechair       #7CFC00         2009   200.00

Enter SQL Statement or quit to finish

>  Select * from product where ItemType = "table" and Colour closeto
"brown"                                                           (83)
```

```
PartID ItemType Colour  SupplierID  Price
WT450  table    #A52A2A     2009   167.95
WT451  table    #7B3F00     2009   167.95

Enter SQL Statement or quit to finish

>  Select * from product where ItemType closeto "table" and Colour
closeto "brown"                                                   (84)

PartID ItemType Colour  SupplierID  Price
WT450  table    #A52A2A     2009   167.95
WT451  table    #7B3F00     2009   167.95

Enter SQL Statement or quit to finish

>  source salesB.sql                                              (85)
Enter SQL Statement or quit to finish
>  source salesitemB.sql                                          (86)
Enter SQL Statement or quit to finish
>  select CustomerName, product.PartID, Colour, sales.InvoiceNo from
customers, product, sales, salesitem where Colour = "brown" and
customers.CustomerID = sales.CustomerID and sales.InvoiceNo =
salesitem.InvoiceNo and product.PartID = salesitem.PartID         (87)

CustomerName   PartID Colour InvoiceNo

Enter SQL Statement or quit to finish

>  select CustomerName, product.PartID, Colour, sales.InvoiceNo from
customers, product, sales, salesitem where Colour CLOSETO "brown" and
customers.CustomerID = sales.CustomerID and sales.InvoiceNo =
salesitem.InvoiceNo and product.PartID = salesitem.PartID         (88)

CustomerName   PartID Colour InvoiceNo
Ian Pill       EG120  orange     117
Angela Brown   IC001  olive      127
Barry de Veen  DC001  green      134
Ian Morgan     IC001  olive      136

Enter SQL Statement or quit to finish

>  select CustomerName, product.PartID, ItemType, Colour,
sales.InvoiceNo from customers, product, sales, salesitem where
customers.CustomerID = sales.CustomerID and sales.InvoiceNo =
salesitem.InvoiceNo and product.PartID = salesitem.PartID         (89)
```

```
CustomerName           PartID ItemType         Colour       InvoiceNo
Keith Myers            KC020  KitchenStool     lime             101
Denise Devine          IC004  IndustrialChair  lightyellow      102
Keith Myers            IC002  ClericalChair    lime             103
Barbara Lincoln        EG456  ErgonomicChair   emerald          104
Garry Cronin           SC125  BarStool         seagreen         105
Barry de Veen          RC444  Recliner         lightgreen       106
John Haggar            RC234  Recliner         darkgreen        107
Lee Provins            EG453  ErgonomicChair   lightblue        108
Barbara Lincoln        DT345  DiningTable      turquoise        109
Barry de Veen          KC021  KitchenStool     apple            110
Stephen May            EG453  ErgonomicChair   lightblue        111
Keith Myers            SC125  BarStool         seagreen         112
Barry de Veen          IC004  IndustrialChair  lightyellow      113
Marion Cartwright      DC004  DiningChair      white            114
Denise Devine          EG453  ErgonomicChair   lightblue        115
Johanna Baker          DT345  DiningTable      turquoise        116
Ian Pill               EG120  ErgonomicChair   orange           117
John Haggar            DT125  DiningTable      white            118
Catherine Hartstein    EG456  ErgonomicChair   emerald          119
Stuart Barich          IC003  IndustrialChair  white            120
Peter Adams            RC831  Recliner         jade             121
Janis Jones            DC026  DiningChair      darkblue         122
Stella Shepherd        CT831  CoffeeTable      jade             123
Garry Cronin           DC023  DiningChair      chartreuse       124
Shirley Hetherington   IC002  ClericalChair    lime             125
Denise Devine          CC236  domchair         #90EE90          126
Angela Brown           IC001  ClericalChair    olive            127
Shirley Hetherington   DC026  DiningChair      darkblue         128
Stuart Barich          LC040  Sofa             aqua             129
Stella Shepherd        RC831  Recliner         jade             130
Stephen May            WT452  table            #000000          131
Brian Heydon           DC004  DiningChair      white            132
Marion Cartwright      CT831  CoffeeTable      jade             133
Barry de Veen          DC001  DiningChair      green            134
Victoria Heineman      RC831  Recliner         jade             135
Ian Morgan             IC001  ClericalChair    olive            136
Ian Morgan             IC003  IndustrialChair  white            137
Stella Shepherd        CC235  domchair         #000000          138
Denise Devine          DC512  DiningChair      white            139

Enter SQL Statement or quit to finish

>  select product.PartID, Colour, ItemType, sales.InvoiceNo from
product, sales, salesitem where Colour CLOSETO "apple" and
sales.InvoiceNo = salesitem.InvoiceNo and product.PartID =
salesitem.PartID                                                   (90)

PartID Colour     ItemType        InvoiceNo
```

```
KC020  lime       KitchenStool       101
IC002  lime       ClericalChair      103
EG456  emerald    ErgonomicChair     104
SC125  seagreen   BarStool           105
RC444  lightgreen Recliner           106
DT345  turquoise  DiningTable        109
KC021  apple      KitchenStool       110
SC125  seagreen   BarStool           112
DT345  turquoise  DiningTable        116
EG456  emerald    ErgonomicChair     119
RC831  jade       Recliner           121
CT831  jade       CoffeeTable        123
DC023  chartreuse DiningChair        124
IC002  lime       ClericalChair      125
CC236  #90EE90    domchair           126
IC001  olive      ClericalChair      127
LC040  aqua       Sofa               129
RC831  jade       Recliner           130
CT831  jade       CoffeeTable        133
DC001  green      DiningChair        134
RC831  jade       Recliner           135
IC001  olive      ClericalChair      136

Enter SQL Statement or quit to finish

>  select product.PartID, Colour, ItemType, sales.InvoiceNo from
product, sales, salesitem where Colour CLOSETO "black" and
sales.InvoiceNo = salesitem.InvoiceNo and product.PartID =
salesitem.PartID                                                (91)

PartID Colour     ItemType        InvoiceNo
WT452  #000000    table              131
CC235  #000000    domchair           138

Enter SQL Statement or quit to finish

>  select CustomerName, product.PartID, Colour, ItemType,
sales.InvoiceNo from customers, product, sales, salesitem where
ItemType = "Recliner" and customers.CustomerID = sales.CustomerID and
sales.InvoiceNo = salesitem.InvoiceNo and product.PartID =
salesitem.PartID                                                (92)

CustomerName        PartID Colour     ItemType InvoiceNo
Barry de Veen       RC444  lightgreen Recliner    106
John Haggar         RC234  darkgreen  Recliner    107
Peter Adams         RC831  jade       Recliner    121
Stella Shepherd     RC831  jade       Recliner    130
Victoria Heineman   RC831  jade       Recliner    135

Enter SQL Statement or quit to finish
```

```
> select CustomerName, product.PartID, Colour, ItemType,
sales.InvoiceNo from customers, product, sales, salesitem where Colour
CLOSETO "apple" and ItemType CLOSETO "chair" and customers.CustomerID =
sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and
product.PartID = salesitem.PartID                                     (93)

CustomerName           PartID Colour     ItemType        InvoiceNo
Keith Myers            KC020  lime       KitchenStool       101
Keith Myers            IC002  lime       ClericalChair      103
Barbara Lincoln        EG456  emerald    ErgonomicChair     104
Garry Cronin           SC125  seagreen   BarStool           105
Barry de Veen          RC444  lightgreen Recliner           106
Barry de Veen          KC021  apple      KitchenStool       110
Keith Myers            SC125  seagreen   BarStool           112
Catherine Hartstein    EG456  emerald    ErgonomicChair     119
Peter Adams            RC831  jade       Recliner           121
Garry Cronin           DC023  chartreuse DiningChair        124
Shirley Hetherington   IC002  lime       ClericalChair      125
Denise Devine          CC236  #90EE90    domchair           126
Angela Brown           IC001  olive      ClericalChair      127
Stuart Barich          LC040  aqua       Sofa               129
Stella Shepherd        RC831  jade       Recliner           130
Barry de Veen          DC001  green      DiningChair        134
Victoria Heineman      RC831  jade       Recliner           135
Ian Morgan             IC001  olive      ClericalChair      136

Enter SQL Statement or quit to finish

> select product.PartID, Colour, ItemType, sales.InvoiceNo from
customers, product, sales, salesitem where ItemType CLOSETO "chair" and
customers.CustomerID = sales.CustomerID and sales.InvoiceNo =
salesitem.InvoiceNo and product.PartID = salesitem.PartID             (94)

PartID Colour     ItemType        InvoiceNo
KC020  lime       KitchenStool       101
IC002  lime       ClericalChair      103
EG456  emerald    ErgonomicChair     104
SC125  seagreen   BarStool           105
RC444  lightgreen Recliner           106
RC234  darkgreen  Recliner           107
EG453  lightblue  ErgonomicChair     108
KC021  apple      KitchenStool       110
EG453  lightblue  ErgonomicChair     111
SC125  seagreen   BarStool           112
DC004  white      DiningChair        114
EG453  lightblue  ErgonomicChair     115
EG120  orange     ErgonomicChair     117
EG456  emerald    ErgonomicChair     119
RC831  jade       Recliner           121
```

```
DC026  darkblue   DiningChair         122
DC023  chartreuse DiningChair         124
IC002  lime       ClericalChair       125
CC236  #90EE90    domchair            126
IC001  olive      ClericalChair       127
DC026  darkblue   DiningChair         128
LC040  aqua       Sofa                129
RC831  jade       Recliner            130
DC004  white      DiningChair         132
DC001  green      DiningChair         134
RC831  jade       Recliner            135
IC001  olive      ClericalChair       136
CC235  #000000    domchair            138
DC512  white      DiningChair         139

Enter SQL Statement or quit to finish

> select * from product where ItemType closeto "table"              (95)

PartID ItemType    Colour    SupplierID  Price
DT345  DiningTable turquoise     2005   500.00
DT125  DiningTable white         2005   355.00
CT831  CoffeeTable jade          2005    50.00
CT444  CoffeeTable darkblue      2005    59.00
WT450  table       #A52A2A       2009   167.95
WT451  table       #7B3F00       2009   167.95
WT452  table       #000000       2009   167.95
WT453  table       #63A671       2009   167.95

Enter SQL Statement or quit to finish

>  CREATE TABLE codelist (NodeI char(5) NOT NULL , NodeJ int NOT NULL,
primary key (NodeI, nodeJ), index (NodeI), index (NodeJ))           (96)

Enter SQL Statement or quit to finish

>  source codelist.sql                                              (97)

Enter SQL Statement or quit to finish

>  create domain NEWCODES as LIST of char(5) over codelist          (98)

Enter SQL Statement or quit to finish

>  show domains                                                     (99)

DOMID DOM_NAME   MESODATA_TYPE  BASE_TYPE RELATION_NAME
3     COLOURS    wgraph         char(15)  colourgraph
2     CATEGORIES dwgraph        char(20)  categorygraph
4     NEWCODES   list           char(5)   codelist

Enter SQL Statement or quit to finish
```

```
>  alter table product modify column PartID NEWCODES NOT NULL      (100)

Enter SQL Statement or quit to finish

>  show mesodatatypes                                              (101)

MTID  TABLE_NAME FIELD_NAME DOMID
1     product    ItemType   2
2     product    Colour     3
4     product    PartID     4

Enter SQL Statement or quit to finish

>  select * from product                                           (102)

PartID ItemType          Colour      SupplierID  Price
IC001  ClericalChair     olive            2002  150.00
IC002  ClericalChair     lime             2002  150.00
EG123  ErgonomicChair    green            2001   60.00
EG456  ErgonomicChair    emerald          2001   60.00
DC001  DiningChair       green            2007   90.00
DC023  DiningChair       chartreuse       2007   96.00
DC510  DiningChair       jade             2006  126.00
SC345  BarStool          turquoise        2005   46.00
SC125  BarStool          seagreen         2005   55.00
RC831  Recliner          jade             2006  250.00
RC444  Recliner          lightgreen       2006  250.00
RC234  Recliner          darkgreen        2006  250.00
KC020  KitchenStool      lime             2008   40.00
KC021  KitchenStool      apple            2008   40.00
LC040  Sofa              aqua             2004  650.00
LC551  Lounge3seater     verdigris        2007  899.00
IC003  IndustrialChair   white            2002  150.00
IC004  IndustrialChair   lightyellow      2002  150.00
EG120  ErgonomicChair    orange           2008   60.00
EG453  ErgonomicChair    lightblue        2008   60.00
DC004  DiningChair       white            2006   90.00
DC026  DiningChair       darkblue         2006   96.00
DC512  DiningChair       white            2006  126.00
DT345  DiningTable       turquoise        2005  500.00
DT125  DiningTable       white            2005  355.00
CT831  CoffeeTable       jade             2005   50.00
CT444  CoffeeTable       darkblue         2005   59.00
WT450  table             #A52A2A          2009  167.95
CC234  domchair          #009966          2009   85.50
WC117  officechair       #00C957          2009  167.95
WT452  table             #000000          2009  167.95
WT453  table             #63A671          2009  167.95
CC235  domchair          #000000          2009   85.50
CC236  domchair          #90EE90          2009   85.50
```

```
CC237  domchair          #008080          2009   85.50
WC118  officechair       #006400          2009  175.00
WC119  officechair       #7CFC00          2009  200.00
WC120  officechair       #7B3F00          2009  250.00

Enter SQL Statement or quit to finish

>  select * from codelist                                             (103)

NodeI  NodeJ
CT444   1444
CT831   1831
DC001   2001
DC004   2004
DC023   2023
DC026   2026
DC510   2510
DC512   2512
DT125   3125
DT345   3345
EG120   4120
EG123   4123
EG453   4453
EG456   4456
IC001   5001
IC002   5002
IC003   5003
IC004   5004
KC020   6020
KC021   6021
LC040   7040
LC551   7551
RC234   8234
RC444   8444
RC831   8831
SC125   9125
SC345   9345

Enter SQL Statement or quit to finish

>  select * from product where PartID equalto 1444                    (104)

PartID ItemType     Colour   SupplierID Price
CT444  CoffeeTable  darkblue    2005    59.00

Enter SQL Statement or quit to finish

>  select SupplierID, PartID, ItemType from product where PartID
equalto 5001 or PartID = 'ic004'                                      (105)
```

```
SupplierID PartID ItemType
  2002      IC001  ClericalChair
  2002      IC004  IndustrialChair

Enter SQL Statement or quit to finish
quit
```

Appendix C

Sample Session SQL Files

C.1 adjColours.sql

```
INSERT INTO colourgraph VALUES (‘white’, ‘lightblue’,0.40);
INSERT INTO colourgraph VALUES (‘lightblue’, ‘darkblue’,0.2);
INSERT INTO colourgraph VALUES (‘lightblue’, ‘aqua’,0.15);
INSERT INTO colourgraph VALUES (‘white’, ‘lightgreen’,0.40);
INSERT INTO colourgraph VALUES (‘lightgreen’, ‘aqua’,0.20);
INSERT INTO colourgraph VALUES (‘lightgreen’, ‘green’,0.15);
INSERT INTO colourgraph VALUES (‘lightgreen’, ‘lime’,0.15);
INSERT INTO colourgraph VALUES (‘white’, ‘yellow’,0.40);
INSERT INTO colourgraph VALUES (‘yellow’, ‘lime’,0.40);
INSERT INTO colourgraph VALUES (‘yellow’, ‘lightyellow’,0.20);
INSERT INTO colourgraph VALUES (‘yellow’, ‘orange’,0.30);
INSERT INTO colourgraph VALUES (‘lightblue’, ‘lightyellow’,0.35);
INSERT INTO colourgraph VALUES (‘lightblue’, ‘lightgreen’,0.35);
INSERT INTO colourgraph VALUES (‘olive’, ‘green’,0.15);
INSERT INTO colourgraph VALUES (‘emerald’, ‘green’,0.15);
INSERT INTO colourgraph VALUES (‘chartreuse’, ‘lime’,0.10);
INSERT INTO colourgraph VALUES (‘jade’, ‘green’,0.10);
INSERT INTO colourgraph VALUES (‘turquoise’, ‘aqua’,0.00);
INSERT INTO colourgraph VALUES (‘seagreen’, ‘aqua’,0.10);
INSERT INTO colourgraph VALUES (‘darkgreen’, ‘green’,0.20);
INSERT INTO colourgraph VALUES (‘apple’, ‘lime’,0.00);
INSERT INTO colourgraph VALUES (‘verdigris’, ‘lightgreen’,0.10);
INSERT INTO colourgraph VALUES (‘olive’, ‘brown’,0.25);
INSERT INTO colourgraph VALUES (‘brown’, ‘orange’,0.30);
INSERT INTO colourgraph VALUES (‘orange’, ‘burntorange’,0.15);
INSERT INTO colourgraph VALUES (‘chestnut’, ‘brown’,0.15);
INSERT INTO colourgraph VALUES (‘aqua’, ‘teal’,0.18);
INSERT INTO colourgraph VALUES (‘darkblue’, ‘navy’,0.10);
INSERT INTO colourgraph VALUES (‘navy’, ‘black’,0.4);
```

C.2 categories.sql

```
INSERT INTO categorygraph VALUES (‘chair’, ‘table’, 1);
INSERT INTO categorygraph VALUES (‘chair’, ‘domchair’, 0.25);
INSERT INTO categorygraph VALUES (‘chair’, ‘officechair’, 0.25);
INSERT INTO categorygraph VALUES (‘officechair’, ‘clericalchair’,
0.00);
INSERT INTO categorygraph VALUES (‘officechair’, ‘ergonomicchair’,
0.00);
INSERT INTO categorygraph VALUES (‘domchair’, ‘diningchair’, 0.1);
INSERT INTO categorygraph VALUES (‘officechair’, ‘barstool’, 0.1);
INSERT INTO categorygraph VALUES (‘domchair’, ‘recliner’, 0.1);
INSERT INTO categorygraph VALUES (‘domchair’, ‘kitchenstool’, 0.1);
INSERT INTO categorygraph VALUES (‘domchair’, ‘sofa’, 0.1);
INSERT INTO categorygraph VALUES (‘domchair’, ‘lounge3seater’, 0.1);
INSERT INTO categorygraph VALUES (‘table’, ‘diningtable’, 0.00);
INSERT INTO categorygraph VALUES (‘table’, ‘coffeetable’, 0.00);
INSERT INTO categorygraph VALUES (‘table’, ‘sidetable’, 0.00);
INSERT INTO categorygraph VALUES (‘table’, ‘changetable’, 0.00);
INSERT INTO categorygraph VALUES (‘table’, ‘boardroomtable’, 0.00);
```

C.3 shadescolours.sql

```
INSERT INTO shadesgraph VALUES (‘white’, ‘lightblue’,0.40);
INSERT INTO shadesgraph VALUES (‘lightblue’, ‘darkblue’,0.2);
INSERT INTO shadesgraph VALUES (‘lightblue’, ‘aqua’,0.15);
INSERT INTO shadesgraph VALUES (‘white’, ‘lightgreen’,0.40);
INSERT INTO shadesgraph VALUES (‘lightgreen’, ‘aqua’,0.20);
INSERT INTO shadesgraph VALUES (‘lightgreen’, ‘green’,0.15);
INSERT INTO shadesgraph VALUES (‘lightgreen’, ‘lime’,0.15);
INSERT INTO shadesgraph VALUES (‘white’, ‘yellow’,0.40);
INSERT INTO shadesgraph VALUES (‘yellow’, ‘lime’,0.40);
INSERT INTO shadesgraph VALUES (‘yellow’, ‘lightyellow’,0.20);
INSERT INTO shadesgraph VALUES (‘yellow’, ‘orange’,0.30);
INSERT INTO shadesgraph VALUES (‘lightblue’, ‘lightyellow’,0.35);
INSERT INTO shadesgraph VALUES (‘lightblue’, ‘lightgreen’,0.35);
INSERT INTO shadesgraph VALUES (‘olive’, ‘green’,0.15);
INSERT INTO shadesgraph VALUES (‘emerald’, ‘green’,0.15);
INSERT INTO shadesgraph VALUES (‘chartreuse’, ‘lime’,0.10);
INSERT INTO shadesgraph VALUES (‘jade’, ‘green’,0.10);
INSERT INTO shadesgraph VALUES (‘turquoise’, ‘aqua’,0.00);
INSERT INTO shadesgraph VALUES (‘seagreen’, ‘aqua’,0.10);
INSERT INTO shadesgraph VALUES (‘darkgreen’, ‘green’,0.20);
INSERT INTO shadesgraph VALUES (‘apple’, ‘lime’,0.00);
INSERT INTO shadesgraph VALUES (‘verdigris’, ‘lightgreen’,0.10);
INSERT INTO shadesgraph VALUES (‘olive’, ‘brown’,0.25);
INSERT INTO shadesgraph VALUES (‘brown’, ‘orange’,0.30);
INSERT INTO shadesgraph VALUES (‘orange’, ‘burntorange’,0.15);
INSERT INTO shadesgraph VALUES (‘chestnut’, ‘brown’,0.15);
INSERT INTO shadesgraph VALUES (‘aqua’, ‘teal’,0.18);
INSERT INTO shadesgraph VALUES (‘darkblue’, ‘navy’,0.10);
INSERT INTO shadesgraph VALUES (‘navy’, ‘black’,0.4);
```

C.4 furnitureB.sql

```
INSERT INTO product VALUES (‘IC001’, ‘ClericalChair’, ‘olive’ ,2002,
149.99);
INSERT INTO product VALUES (‘IC002’, ‘ClericalChair’, ‘lime’ ,2002,
149.99);
INSERT INTO product VALUES (‘EG123’, ‘ErgonomicChair’, ‘green’ ,2001,
59.80);
INSERT INTO product VALUES (‘EG456’, ‘ErgonomicChair’, ‘emerald’ ,2001,
59.80);
INSERT INTO product VALUES (‘DC001’, ‘DiningChair’, ‘green’ ,2007,
89.90);
INSERT INTO product VALUES (‘DC023’, ‘DiningChair’, ‘chartreuse’,2007,
95.50);
INSERT INTO product VALUES (‘DC510’, ‘DiningChair’, ‘jade’ ,2006,
125.50);
INSERT INTO product VALUES (‘SC345’, ‘BarStool’, ‘turquoise’ ,2005,
45.90);
INSERT INTO product VALUES (‘SC125’, ‘BarStool’, ‘seagreen’ ,2005,
55.00);
INSERT INTO product VALUES (‘RC831’, ‘Recliner’, ‘jade’ ,2006, 250.00);
INSERT INTO product VALUES (‘RC444’, ‘Recliner’, ‘lightgreen’ ,2006,
250.00);
INSERT INTO product VALUES (‘RC234’, ‘Recliner’, ‘darkgreen’ ,2006,
250.00);
INSERT INTO product VALUES (‘KC020’, ‘KitchenStool’, ‘lime’ ,2008,
40.50);
INSERT INTO product VALUES (‘KC021’, ‘KitchenStool’, ‘apple’ ,2008,
40.50);
INSERT INTO product VALUES (‘LC040’, ‘Sofa’, ‘aqua’ ,2004, 650.00);
INSERT INTO product VALUES (‘LC551’, ‘Lounge3seater’, ‘verdigris’
,2007, 899.00);
INSERT INTO product VALUES (‘IC003’, ‘IndustrialChair’, ‘white’ ,2002,
149.99);
INSERT INTO product VALUES (‘IC004’, ‘IndustrialChair’, ‘lightyellow’
,2002, 149.99);
INSERT INTO product VALUES (‘EG120’, ‘ErgonomicChair’, ‘orange’ ,2008,
59.80);
INSERT INTO product VALUES (‘EG453’, ‘ErgonomicChair’, ‘lightblue’
,2008, 59.80);
INSERT INTO product VALUES (‘DC004’, ‘DiningChair’, ‘white’ ,2006,
89.90);
INSERT INTO product VALUES (‘DC026’, ‘DiningChair’, ‘darkblue’ ,2006,
95.50);
INSERT INTO product VALUES (‘DC512’, ‘DiningChair’, ‘white’ ,2006,
125.50);
INSERT INTO product VALUES (‘DT345’, ‘DiningTable’, ‘turquoise’ ,2005,
500.00);
INSERT INTO product VALUES (‘DT125’, ‘DiningTable’, ‘white’ ,2005,
355.00);
INSERT INTO product VALUES (‘CT831’, ‘CoffeeTable’, ‘jade’ ,2005,
50.00);
INSERT INTO product VALUES (‘CT444’, ‘CoffeeTable’, ‘darkblue’ ,2005,
59.00);
```

C.5 customers.sql

```
INSERT INTO customers VALUES (1001, ‘Peter Adams’);
INSERT INTO customers VALUES (1002, ‘Johanna Baker’);
INSERT INTO customers VALUES (1003, ‘Stuart Barich’);
INSERT INTO customers VALUES (1004, ‘Angela Brown’);
INSERT INTO customers VALUES (1005, ‘Marion Cartwright’);
INSERT INTO customers VALUES (1006, ‘Garry Cronin’);
INSERT INTO customers VALUES (1007, ‘Barry de Veen’);
INSERT INTO customers VALUES (1008, ‘Denise Devine’);
INSERT INTO customers VALUES (1009, ‘Rosalie Dunn’);
INSERT INTO customers VALUES (1010, ‘John Haggar’);
INSERT INTO customers VALUES (1011, ‘Clive Hallett’);
INSERT INTO customers VALUES (1012, ‘Catherine Hartstein’);
INSERT INTO customers VALUES (1013, ‘Victoria Heineman’);
INSERT INTO customers VALUES (1014, ‘Shirley Hetherington’);
INSERT INTO customers VALUES (1015, ‘Brian Heydon’);
INSERT INTO customers VALUES (1016, ‘Janis Jones’);
INSERT INTO customers VALUES (1017, ‘Barbara Lincoln’);
INSERT INTO customers VALUES (1018, ‘Stephen May’);
INSERT INTO customers VALUES (1019, ‘Geoff McRae’);
INSERT INTO customers VALUES (1020, ‘Ian Morgan’);
INSERT INTO customers VALUES (1021, ‘Keith Myers’);
INSERT INTO customers VALUES (1022, ‘Ian Pill’);
INSERT INTO customers VALUES (1023, ‘Lee Provins’);
INSERT INTO customers VALUES (1024, ‘Stephen Schroeter’);
INSERT INTO customers VALUES (1025, ‘Stella Shepherd’);
INSERT INTO customers VALUES (1026, ‘Stephen Tongue’);
INSERT INTO customers VALUES (1027, ‘Susanne Wright’);
```

C.6 suppliers.sql

```
INSERT INTO suppliers VALUES (2001, ‘Aardvark Furniture
Manufacturers’);
INSERT INTO suppliers VALUES (2002, ‘Blacksmith Commercial’);
INSERT INTO suppliers VALUES (2003, ‘Creative Office Furniture’);
INSERT INTO suppliers VALUES (2004, ‘Salon Furniture’);
INSERT INTO suppliers VALUES (2005, ‘Tables R Flat’);
INSERT INTO suppliers VALUES (2006, ‘Chairs Galore’);
INSERT INTO suppliers VALUES (2007, ‘Comfy Home Supplies’);
INSERT INTO suppliers VALUES (2008, ‘Sit Yourself’);
INSERT INTO suppliers VALUES (2009, ‘World of Furniture’);
```

C.7 sales.sql

```
INSERT INTO sales VALUES (101, ‘2005-1-31’, 1021);
INSERT INTO sales VALUES (102, ‘2005-2-7’, 1008);
INSERT INTO sales VALUES (103, ‘2005-3-10’, 1021);
INSERT INTO sales VALUES (104, ‘2005-3-22’, 1017);
INSERT INTO sales VALUES (105, ‘2005-4-1’, 1006);
INSERT INTO sales VALUES (106, ‘2005-4-5’, 1007);
INSERT INTO sales VALUES (107, ‘2005-4-11’, 1010);
INSERT INTO sales VALUES (108, ‘2005-7-1’, 1023);
INSERT INTO sales VALUES (109, ‘2005-7-3’, 1017);
INSERT INTO sales VALUES (110, ‘2005-7-6’, 1007);
INSERT INTO sales VALUES (111, ‘2005-7-8’, 1018);
INSERT INTO sales VALUES (112, ‘2005-7-14’, 1021);
INSERT INTO sales VALUES (113, ‘2005-7-19’, 1007);
INSERT INTO sales VALUES (114, ‘2005-7-24’, 1005);
INSERT INTO sales VALUES (115, ‘2005-8-1’, 1008);
INSERT INTO sales VALUES (116, ‘2005-8-14’, 1002);
INSERT INTO sales VALUES (117, ‘2005-9-3’, 1022);
INSERT INTO sales VALUES (118, ‘2005-9-30’, 1010);
INSERT INTO sales VALUES (119, ‘2005-10-3’, 1012);
INSERT INTO sales VALUES (120, ‘2005-10-4’, 1003);
INSERT INTO sales VALUES (121, ‘2005-10-12’, 1001);
INSERT INTO sales VALUES (122, ‘2005-10-15’, 1016);
INSERT INTO sales VALUES (123, ‘2005-11-7’, 1025);
INSERT INTO sales VALUES (124, ‘2005-11-26’, 1006);
INSERT INTO sales VALUES (125, ‘2005-12-4’, 1014);
```

C.8 salesitem.sql

```
INSERT INTO salesitem VALUES (1, 101, ‘KC020’, 3);
INSERT INTO salesitem VALUES (2, 102, ‘IC004’, 1);
INSERT INTO salesitem VALUES (3, 103, ‘IC002’, 2);
INSERT INTO salesitem VALUES (4, 104, ‘EG456’, 5);
INSERT INTO salesitem VALUES (5, 105, ‘SC125’, 1);
INSERT INTO salesitem VALUES (6, 106, ‘RC444’, 4);
INSERT INTO salesitem VALUES (7, 107, ‘RC234’, 3);
INSERT INTO salesitem VALUES (8, 108, ‘EG453’, 3);
INSERT INTO salesitem VALUES (9, 109, ‘DT345’, 2);
INSERT INTO salesitem VALUES (10, 110, ‘KC021’, 2);
INSERT INTO salesitem VALUES (11, 111, ‘EG453’, 5);
INSERT INTO salesitem VALUES (12, 112, ‘SC125’, 4);
INSERT INTO salesitem VALUES (13, 113, ‘IC004’, 5);
INSERT INTO salesitem VALUES (14, 114, ‘DC004’, 3);
INSERT INTO salesitem VALUES (15, 115, ‘EG453’, 2);
INSERT INTO salesitem VALUES (16, 116, ‘DT345’, 4);
INSERT INTO salesitem VALUES (17, 117, ‘EG120’, 5);
INSERT INTO salesitem VALUES (18, 118, ‘DT125’, 4);
INSERT INTO salesitem VALUES (19, 119, ‘EG456’, 5);
INSERT INTO salesitem VALUES (20, 120, ‘IC003’, 5);
INSERT INTO salesitem VALUES (21, 121, ‘RC831’, 4);
INSERT INTO salesitem VALUES (22, 122, ‘DC026’, 5);
INSERT INTO salesitem VALUES (23, 123, ‘CT831’, 2);
INSERT INTO salesitem VALUES (24, 124, ‘DC023’, 5);
INSERT INTO salesitem VALUES (25, 125, ‘IC002’, 5);
```

C.9 hexColours.sql

```
INSERT INTO colourgraph VALUES (‘apple’, ‘#7CFC00’, 0.00);
INSERT INTO colourgraph VALUES (‘aqua’, ‘#00FFFF’, 0.00);
INSERT INTO colourgraph VALUES (‘black’, ‘#000000’, 0.00);
INSERT INTO colourgraph VALUES (‘brown’, ‘#A52A2A’, 0.00);
INSERT INTO colourgraph VALUES (‘burntorange’, ‘#FF8C00’, 0.00);
INSERT INTO colourgraph VALUES (‘chartreuse’, ‘#7FFF00’, 0.00);
INSERT INTO colourgraph VALUES (‘chestnut’, ‘#7B3F00’, 0.00);
INSERT INTO colourgraph VALUES (‘darkblue’, ‘#00008B’, 0.00);
INSERT INTO colourgraph VALUES (‘darkgreen’, ‘#006400’, 0.00);
INSERT INTO colourgraph VALUES (‘emerald’, ‘#00C957’, 0.00);
INSERT INTO colourgraph VALUES (‘green’, ‘#008000’, 0.00);
INSERT INTO colourgraph VALUES (‘jade’, ‘#009966’, 0.00);
INSERT INTO colourgraph VALUES (‘lightblue’, ‘#ADD8E6’, 0.00);
INSERT INTO colourgraph VALUES (‘lightgreen’, ‘#90EE90’, 0.00);
INSERT INTO colourgraph VALUES (‘lightyellow’, ‘#FFFFE0’, 0.00);
INSERT INTO colourgraph VALUES (‘lime’, ‘#00FF00’, 0.00);
INSERT INTO colourgraph VALUES (‘navy’, ‘#000080’, 0.00);
INSERT INTO colourgraph VALUES (‘olive’, ‘#808000’, 0.00);
INSERT INTO colourgraph VALUES (‘orange’, ‘#FFA500’, 0.00);
INSERT INTO colourgraph VALUES (‘seagreen’, ‘#2E8B57’, 0.00);
INSERT INTO colourgraph VALUES (‘teal’, ‘#008080’, 0.00);
INSERT INTO colourgraph VALUES (‘turquoise’, ‘#40E0D0’, 0.00);
INSERT INTO colourgraph VALUES (‘verdigris’, ‘#63A671’, 0.00);
INSERT INTO colourgraph VALUES (‘white’, ‘#FFFFFF’, 0.00);
INSERT INTO colourgraph VALUES (‘yellow’, ‘#FFFF00’, 0.00);
```

C.10 furnitureC.sql

```
INSERT INTO product VALUES (‘WT450’, ‘table’, ‘#A52A2A’,2009, 167.95);
INSERT INTO product VALUES (‘CC234’, ‘domchair’, ‘#009966’,2009,
85.50);
INSERT INTO product VALUES (‘WC117’, ‘officechair’, ‘#00C957’, 2009,
250.00);
INSERT INTO product VALUES (‘WT451’, ‘table’, ‘#7B3F00’, 2009, 167.95);
INSERT INTO product VALUES (‘WT452’, ‘table’, ‘#000000’, 2009, 167.95);
INSERT INTO product VALUES (‘WT453’, ‘table’, ‘#63A671’, 2009, 167.95);
INSERT INTO product VALUES (‘CC235’, ‘domchair’, ‘#000000’, 2009,
85.50);
INSERT INTO product VALUES (‘CC236’, ‘domchair’, ‘#90EE90’, 2009,
85.50);
INSERT INTO product VALUES (‘CC237’, ‘domchair’, ‘#008080’, 2009,
85.50);
INSERT INTO product VALUES (‘WC118’, ‘officechair’, ‘#006400’, 2009,
175.00);
INSERT INTO product VALUES (‘WC119’, ‘officechair’, ‘#7CFC00’, 2009,
200.00);
INSERT INTO product VALUES (‘WC120’, ‘officechair’, ‘#7B3F00’, 2009,
250.00);
```

C.11 salesB.sql

```
INSERT INTO sales VALUES (126, ‘2006-1-17’, 1008);
INSERT INTO sales VALUES (127, ‘2006-1-30’, 1004);
INSERT INTO sales VALUES (128, ‘2006-2-1’, 1014);
INSERT INTO sales VALUES (129, ‘2006-5-15’, 1003);
INSERT INTO sales VALUES (130, ‘2006-5-8’, 1025);
INSERT INTO sales VALUES (131, ‘2006-6-24’, 1018);
INSERT INTO sales VALUES (132, ‘2006-7-29’, 1015);
INSERT INTO sales VALUES (133, ‘2006-7-3’, 1005);
INSERT INTO sales VALUES (134, ‘2006-8-16’, 1007);
INSERT INTO sales VALUES (135, ‘2006-8-25’, 1013);
INSERT INTO sales VALUES (136, ‘2006-8-8’, 1020);
INSERT INTO sales VALUES (137, ‘2006-9-12’, 1020);
INSERT INTO sales VALUES (138, ‘2006-9-27’, 1025);
INSERT INTO sales VALUES (139, ‘2006-11-11’, 1008);
```

C.12 salesitemB.sql

```
INSERT INTO salesitem VALUES (26, 126, ‘CC236’‘, 5);
INSERT INTO salesitem VALUES (27, 127, ‘IC001’, 1);
INSERT INTO salesitem VALUES (28, 128, ‘DC026’, 3);
INSERT INTO salesitem VALUES (29, 129, ‘LC040’, 4);
INSERT INTO salesitem VALUES (30, 130, ‘RC831’, 1);
INSERT INTO salesitem VALUES (31, 131, ‘WT452’, 2);
INSERT INTO salesitem VALUES (32, 132, ‘DC004’, 1);
INSERT INTO salesitem VALUES (33, 133, ‘CT831’, 2);
INSERT INTO salesitem VALUES (34, 134, ‘DC001’, 5);
INSERT INTO salesitem VALUES (35, 135, ‘RC831’, 4);
INSERT INTO salesitem VALUES (36, 136, ‘IC001’, 3);
INSERT INTO salesitem VALUES (37, 137, ‘IC003’, 3);
INSERT INTO salesitem VALUES (38, 138, ‘CC235’, 2);
INSERT INTO salesitem VALUES (39, 139, ‘DC512’, 4);
```

C.13 codelist.sql

```
INSERT INTO codelist VALUES (‘CT444’, 1444);
INSERT INTO codelist VALUES (‘CT831’, 1831);
INSERT INTO codelist VALUES (‘DC001’, 2001);
INSERT INTO codelist VALUES (‘DC004’, 2004);
INSERT INTO codelist VALUES (‘DC023’, 2023);
INSERT INTO codelist VALUES (‘DC026’, 2026);
INSERT INTO codelist VALUES (‘DC510’, 2510);
INSERT INTO codelist VALUES (‘DC512’, 2512);
INSERT INTO codelist VALUES (‘DT125’, 3125);
INSERT INTO codelist VALUES (‘DT345’, 3345);
INSERT INTO codelist VALUES (‘EG120’, 4120);
INSERT INTO codelist VALUES (‘EG123’, 4123);
INSERT INTO codelist VALUES (‘EG453’, 4453);
INSERT INTO codelist VALUES (‘EG456’, 4456);
INSERT INTO codelist VALUES (‘IC001’, 5001);
INSERT INTO codelist VALUES (‘IC002’, 5002);
INSERT INTO codelist VALUES (‘IC003’, 5003);
INSERT INTO codelist VALUES (‘IC004’, 5004);
INSERT INTO codelist VALUES (‘KC020’, 6020);
INSERT INTO codelist VALUES (‘KC021’, 6021);
INSERT INTO codelist VALUES (‘LC040’, 7040);
INSERT INTO codelist VALUES (‘LC551’, 7551);
INSERT INTO codelist VALUES (‘RC234’, 8234);
INSERT INTO codelist VALUES (‘RC444’, 8444);
INSERT INTO codelist VALUES (‘RC831’, 8831);
INSERT INTO codelist VALUES (‘SC125’, 9125);
INSERT INTO codelist VALUES (‘SC345’, 9345);
```

Appendix D

Prototype Functionality

Table D.1. Prototype Functionality

Function	**Action**	**Example in Appendix B**
USE DATABASE	System tables DOMTABLE and MESOTABLE opened (created if not existing)	1
CREATE DOMAIN	System Table DOMTABLE updated. System Table created with the domain name and populated with values from source relation.	9-13
Trap error for Domain already exists Invalid mesodata type Incompatible source relation Source not found Incorrect syntax	Error Message	
ALTER DOMAIN NAME MESODATATYPE	 System table of domain values renamed. System table DOMTABLE updated System table of domain values deleted. System table of domain values created and populated from source.	 15, 17 16 19 20

Function	**Action**	**Example in Appendix B**
	Entry in DOMTABLE updated.	20
BASETYPE	System table of domain values altered.	22, 24
	System table DOMTABLE updated	23
RELATION	System table of domain values deleted.	28
	System table of domain values created and populated from new source.	
	Entry in DOMTABLE updated.	29
Trap error for Domain does not exist New name exists Invalid mesodata type Invalid base type Source relation does not exist Incorrect syntax	Error message	
DROP DOMAIN	System table of domain values deleted.	30,31
	Entry in DOMTABLE deleted.	33
	Source relation saved.	31
Trap error for Domain does not exist Reference found for domain Incorrect syntax	Error Message	
CREATE TABLE		
with mesodata domain	base type inherited by table specification.	35,39
	Entry in MESOTABLE	38
without mesodata domain	standard table created	2
Trap error for Domain does not exist. Incorrect syntax	Error Message	
ALTER TABLE		
without mesodata domain	standard alter table	44- 49
with mesodata domain		
ADD COLUMN	base type inherited by table specification.	50-52
	Entry in MESOTABLE created	
CHANGE COLUMN	Table specification altered	53- 55
MODIFY COLUMN	Entry in MESOTABLE deleted	
from reference to mesodata type to base type		

Function	**Action**	**Example in Appendix B**
CHANGE COLUMN MODIFY COLUMN to reference to mesodata type DROP COLUMN	base type inherited by table specification. Entry in MESOTABLE. Table specification altered. Entry in MESOTABLE deleted.	98- 100 56 - 58
Trap error for Domain does not exist. Incorrect syntax	Error Message	
REFRESH DOMAIN	System table of domain values rebuilt from source relation.	76
SHOW DOMAINS SHOW MESODATATYPES	Display Domains Display Mesodata types referenced by attributes	13 38
SELECT statements Standard SQL only with mesodata operators only with multiple mesodata domains and operators with mesodata operators plus standard SQL operators multiple mesodata domains and operators plus standard SQL operators	 execute as normal Result set retrieved correctly Result set retrieved correctly Result set retrieved correctly Result set retrieved correctly	 60,62,63 62,78, 79,82 65,84 64,71 74,93

Appendix E

Prototype Domain Querying

Table E.1. Prototype Querying

Function	**Example**	**SQL**	**Example in Appendix B**
Enhanced querying	Colours are defined over a mesodata domain weighted graph. ItemTypes are defined over a Mesodata domain directed weighted graph.	CREATE TABLE product (PartID char(5) NOT NULL, ItemType CATEGORIES, Colour COLOURS, SupplierID int, Price numeric(9), PRIMARY KEY (PartID))	35
	Queries for products with a Colour a shade of 'green' retrieves tuples matching similar colour values within the domain.	select * from product where Colour = 'green' select * from product where Colour closeto 'green'	61 62
	Queries for products that are a type of 'chair' retrieves tuples matching similar category values within the domain.	select * from product where Colour closeto 'green' and ItemType = 'recliner' select * from product where Colour closeto 'green' and ItemType closeto 'chair'	63 65

Function	Example	SQL	Example in Appendix B
	Queries over multiple relations projecting different views	select CustomerName, product.PartID, Colour, sales.InvoiceNo from customers, product, sales, salesitem where Colour = ‘apple’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	70
		select CustomerName, product.PartID, Colour, sales.InvoiceNo from customers, product, sales, salesitem where Colour CLOSETO ‘apple’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	71
		select CustomerName, product.PartID, ItemType, Colour, sales.InvoiceNo from customers, product, sales, salesitem where customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	72
		select CustomerName, product.PartID, Colour, ItemType, sales.InvoiceNo from customers, product, sales, salesitem where ItemType = ‘Recliner’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	73

Function	**Example**	**SQL**	**Example in Appendix B**
		select CustomerName, product.PartID, Colour, ItemType, sales.InvoiceNo from customers, product, sales, salesitem where Colour CLOSETO ‘apple’ and ItemType CLOSETO ‘chair’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	74
Domain Perception Change	The product colour is coded by a hexadecimal string for the shade instead of word terms. New values are added to the domain data.	hexColours.sql (see Appendix C.9) refresh domain COLOURS	75 76
	Queries referring to new domain values retrieve matching similar values in data.	select * from product where Colour = ‘#008000’ select * from product where Colour closeto ‘#008000’ select * from product where Colour closeto ‘green’	77 78 79
Domain Constraints Change	ItemTypes classified at different levels of categories. Records are added to the product relation with new ItemTypes and hexadecimal Colour values.	furnitureC.sql (see Appendix C.10)	80

Function	**Example**	**SQL**	**Example in Appendix B**
	Queries referring to both new and old categories and colours retrieve matching records.	select * from product select * from product where Colour closeto '#90EE90' select * from product where ItemType = 'table' and Colour closeto 'brown' select * from product where ItemType closeto 'table' and Colour closeto 'brown'	81 82 83 84
Data Integration	Data conforming to different entry standards for Colour and ItemType are present in the relation product. Records related to these added to other relations.	salesB.sql (see Appendix C.11) salesitemB.sql (see Appendix C.12)	85 86
	Queries over multiple relations projecting different views	select CustomerName, product.PartID, Colour, sales.InvoiceNo from customers, product, sales, salesitem where Colour = 'brown' and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	87

Function	Example	SQL	Example in Appendix B
		select CustomerName, product.PartID, Colour, sales.InvoiceNo from customers, product, sales, salesitem where Colour CLOSETO ‘brown’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	88
		select CustomerName, product.PartID, ItemType, Colour, sales.InvoiceNo from customers, product, sales, salesitem where customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	89
		select product.PartID, Colour, ItemType, sales.InvoiceNo from product, sales, salesitem where Colour CLOSETO ‘apple’ and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	90
		select product.PartID, Colour, ItemType, sales.InvoiceNo from product, sales, salesitem where Colour CLOSETO ‘black’ and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	91

Function	Example	SQL	Example in Appendix B
		select CustomerName, product.PartID, Colour, ItemType, sales.InvoiceNo from customers, product, sales, salesitem where ItemType = ‘Recliner’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	92
		select CustomerName, product.PartID, Colour, ItemType, sales.InvoiceNo from customers, product, sales, salesitem where Colour CLOSETO ‘apple’ and ItemType CLOSETO ‘chair’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	93
		select CustomerName, product.PartID, Colour, ItemType, sales.InvoiceNo from customers, product, sales, salesitem where ItemType CLOSETO ‘chair’ and customers.CustomerID = sales.CustomerID and sales.InvoiceNo = salesitem.InvoiceNo and product.PartID = salesitem.PartID	94

Function	**Example**	**SQL**	**Example in Appendix B**
		select * from product where ItemType closeto ”table”	95
Attribute Representation Change	PartIDs are changed from char(5) to integer codes.		
	A new domain is created to manage the new codes	codelist.sql (see Appendix C.13 create domain NEWCODES as LIST of char(5) over codelist	96, 97 98
	Relation product altered to reference a Mesodata list containing numeric code equivalents to string codes	alter table product modify column PartID NEWCODES NOT NULL show mesodatatypes	100 101
	Query for products using numeric code	select * from product select * from product where PartID equalto 1444	102 104
	Query for products using both the old and new codes.	select SupplierID, PartID, ItemType from product where PartID equalto 5001 or PartID = ‘ic004’	105

Appendix F

Data type Comparisons

Table F.1. Comparison of Data Types

Type name VB	Type name MySQL	Type Name PostgreSQL	SQL Server	Oracle	Description	Range
NUMERIC						
Byte	TINYINT		tinyint		1-byte binary	0 to 255
Integer	SMALLINT	int2	smallint	smallint	2-byte integer	- 32,768 to 32,767
	MEDIUMINT				3-byte integer	-8,388,608 to 8,388,607. The unsigned range is 0 to 16,777,215
Long	INT INTEGER	int4 integer	int	integer	4-byte integer	-2,147,483,648 to 2,147,483,647
Object					4 bytes	Any Object reference (same as long)
	BIGINT	int8	bigint		8-byte integer	The signed range is $-9,223,372,036,854,775,808$ to $9,223,372,036,854,775,807$. The unsigned range is 0 to 18,446,744,073,709,551,615.
Single	FLOAT	float4	real		4-byte floating-point number	-3.402823E38 to -1.401298E-45 (-ve) 1.401298E-45 to 3.402823E38 (+ve)
Double	DOUBLE REAL	float8	float		8-byte floating-point number	-1.79769313486231E308 to -4.94065645841247E-324 (-ve) 4.94065645841247E-324 to 1.79769313486231E308 (+ve)
				float		number up to 38 digits of precision
MONETARY						
		money	smallmoney		4 byte	-21,474,836.48 to +214,748,36.47
Currency			money		8-byte number with fixed decimal point	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
	DECIMAL				8-byte	Range is dependent upon display size and precision
CHARACTER	NON-UNICODE					
	CHAR (0 - 255)	char (1-n)	char	char (1 - 2000)	1 to max bytes fixed	*MySQL* allows CHAR(0) that only can take 2

Type name VB	**Type name MySQL**	**Type Name PostgreSQL**	**SQL Server**	**Oracle**	**Description**	**Range**
			character (1- 8000)		length	values: ‘NULL’ or “””’.
	VARCHAR (0-255)	varchar(1-n)	varchar(1 - 8000)	varchar2(1-4000) raw(max 2000)	0 to max variable length	RAWs are used to store data that won’t be converted
	TINYBLOB TINYTEXT				max 255 chars	
	BLOB TEXT				max 65,535 chars	
	MEDIUMBLOB MEDIUM-TEXT				max 16,777,215 chars	
			text	long; long raw	max 2 GB	Oracle’s LONGs store character data that are converted when moved from one database to the other. RAWs are used to store data that won’t be converted. LONGs are only supported for backward compatibility
	LONGBLOB LONGTEXT			clob	max 4 GB	
		text			unlimited variable length	
	UNICODE					
	nchar(n) max 255		nchar(n) max 4000	nchar(n) max 2,000	fixed length	
	national var-char(n) max 255		nvarchar(n) max 4000	nvarchar(n) max 4000	variable length	
			ntext		max 1 GB	
				nclob	max 4 GB	
string max 2 billion					String of characters	String variables are stored as sequences of unsigned 16-bit (2-byte) numbers ranging in value from 0 through 65,535. Each number represents a single Unicode character. A string can contain up to approximately 2 billion (2^{31}) Unicode characters.

Type name VB	Type name MySQL	Type Name PostgreSQL	SQL Server	Oracle	Description	Range
Variant					Date/time, floating-point number, integer, string, or object. 16 bytes, plus 1 byte for each character if a string value.	Date values: 1 January 100 to 31 December 9999. Numeric values: same range as Double. String values: same range as String. Can also contain Error or Null
	ENUM				Enumeration of Value Set	maximum of 65,535 distinct values
	SET				Set of values	maximum of 64 members
BOOLEAN						
Boolean		bool			2 bytes	True or False
	BIT / BOOL		bit (0 or 1)		1 byte	MySQL TINYINT(1) $Value == 0\ True / \neq 0\ False$
DATE/TIME						
Date					8-byte date/time value	1 January 100 00:00:00 to 31 December 9999 23:59:59
	DATE				3 bytes	Date without Time 1000-01-01 - 9999-12-31
	DATETIME				8 bytes	1000-01-01 00:00:00 - 9999-12-31 23:59:59
	TIMESTAMP				4 bytes	1970-01-01 00:00:00 to sometime 2037
	TIME				3 bytes	Time -838:59:59 - 838:59:59
	YEAR				1 byte	Year (integer -32,768 to 32,767)
			datetime		8 bytes	1753-01-01 to 999-12-31 accuracy 3.33 millisec
			smalldatetime		4 bytes	1900-01-01 to 2079-12-31 accuracy 1 min
				date	7 bytes Fixed-length date + time value.	4712BC-01-01 00:00:00 to 9999AD-12-31 23:59:59
		date			4 bytes	4713 BC-01-01 to 32767 AD-12-31
		time			8 bytes	w/o time zone 00:00:00.00 - 23:59:59.99
		time			12 bytes	with timezone 00:00:00.00 - 23:59:59.99
		timestamp			8 bytes	4713 BC-01-01 00:00:00.00 to 1465001AD-12-31 23:59:59.99 (acc 1 microsecond)

Type name VB	Type name MySQL	Type Name PostgreSQL	SQL Server	Oracle	Description	Range
BINARY						
			binary max 8000		fixed length	
			varbinary max 8000		variable length	
			image max 2 GB		variable length	
				bfile max 4GB		pointer to binary file on disk enables access to binary file LOBs that are stored in file systems outside the Oracle database
				blob max 4GB	variable length	
		bytea			4 bytes + binary string	

Appendix G

Mapping MySQL to Java types

Table G.1. Mapping SQL and Java data types (MySQL 2003)

MySQL Data Types	**Java types**
CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET	java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Blob, java.sql.Clob
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	java.lang.String, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.math.BigDecimal
DATE, TIME, DATETIME, TIMESTAMP	java.lang.String, java.sql.Date, java.sql.Timestamp

MySQL Type Name	**Returned as Java Class for ResultSet.getObject()**
BIT(1)	java.lang.Boolean
BIT(> 1)	byte[ ]
TINYINT	IF 'tinyInt1isBit' is set to 'true' (the default) AND storage size is '1' java.lang.Boolean ELSE java.lang.Integer
BOOL , BOOLEAN	See TINYINT
SMALLINT[(M)] [UNSIGNED]	java.lang.Integer (regardless if UNSIGNED or not)

MySQL Type Name	**Returned as Java Class for ResultSet.getObject()**
MEDIUMINT[(M)] [UNSIGNED]	java.lang.Integer (regardless if UNSIGNED or not)
INT,INTEGER[(M)] [UNSIGNED]	java.lang.Integer, if UNSIGNED java.lang.Long
BIGINT[(M)] [UNSIGNED]	java.lang.Long, if UNSIGNED java.math.BigInteger
FLOAT[(M,D)]	java.lang.Float
DOUBLE[(M,B)]	java.lang.Double
DECIMAL[(M[,D])]	java.math.BigDecimal
DATE	java.sql.Date
DATETIME	java.sql.Timestamp
TIMESTAMP[(M)]	java.sql.Timestamp
TIME	java.sql.Time
YEAR[(2—4)]	java.sql.Date (1 January at midnight)
CHAR(M)	java.lang.String (unless the character set for the column is BINARY, then byte[ ] is returned.
VARCHAR(M) [BINARY]	java.lang.String (unless the character set for the column is BINARY, then byte[ ] is returned.
BINARY(M)	byte[ ]
VARBINARY(M)	byte[ ]
TINYBLOB	byte[ ]
TINYTEXT	java.lang.String
BLOB	byte[ ]
TEXT	java.lang.String
MEDIUMBLOB	byte[ ]
MEDIUMTEXT	java.lang.String
LONGBLOB	byte[ ]
LONGTEXT	java.lang.String
ENUM('value1','value2',...)	java.lang.String
SET('value1','value2',...)	java.lang.String

Bibliography

Abramsky, S. & Jung, A. (1994), Domain theory, *in* S. Abramsky, D. M. Gabbay & T. S. E. Maibaum, eds, 'Handbook of Logic in Computer Science', Vol. 3, Clarendon Press, pp. 1–168.

Ahmed-Nacer, M. & Estublier, J. (2000), 'Schema evolution in software engineering databases: A new approach in ADELE', *Computer and Artificial Intelligence Journal* **19**, 183 – 203.

Albert, J. (2000), Theoretical foundations of schema restructuring in heterogeneous multidatabase systems, *in* '9th international conference on Information and knowledge management', ACM Press, McLean, Virginia, United States, pp. 461–470.

Allen, J. F. (1983), 'Maintaining knowledge about temporal intervals', *Communications of the ACM* **26**(11), 832 – 843.

Baekgaard, L. (1997), Transaction-based specification of database evolution., *in* D. W. Embley & R. C. Goldstein, eds, 'ER'97 16th International Conference on Conceptual Modeling', Vol. 1331 of *Lecture Notes in Computer Science*, Springer, Los Angeles, California, USA, pp. 127–140.

Bechhofer, S., Broekstra, J., Decker, S., Erdmann, M., Fensel, D., Goble, C., van Harmelen, F., Horrocks, I., Klein, M., McGuinness, D., Motta, E., Patel-Schneider, P., Staab, S. & Studer, R. (2000), An informal description of standard OIL and instance OIL, Technical report, DARPA.

Beneventano, D. & Bergamaschi, S. (2004), The MOMIS methodology for integrating heterogeneous data sources, *in* 'IFIP World Computer Congress', Toulouse France.

Berlin, J. & Motro, A. (2002), Database schema matching using machine learning with feature selection., *in* A. Banks Pidduck, J. Mylopoulos, C. C. Woo &

M. T. Özsu, eds, ‘Conference on Advanced Systems Engineering (CAiSE)’, Vol. 2348, Springer, Toronto, Canada, pp. 452–466.

Bernstein, P. A., Melnik, S., Petropoulos, M. & Quix, C. (2004), ‘Industrial-strength schema matching.’, *ACM SIGMOD Record* **33**(4), 38–43.

Bertino, E. & Martino, L. (1993), *Object-Oriented Database Systems Concepts and Architectures*, Addison-Wesley Publishing Company, Wokingham.

Bertossi, L. & Schwind, C. (2004), ‘Database repairs and analytic tableaux’, *Annals of Mathematics and Artificial Intelligence* **40**(1-2), 5–35.

Blaschka, M., Sapia, C. & Hofling, G. (1999), On schema evolution in multidimensional databases, *in* M. K. Mohania & A. M. Tjoa, eds, ‘1st International Conference on Data Warehousing and Knowledge Discovery, DaWaK ’99’, Vol. 1676 of *Lecture Notes in Computer Science*, Springer, Florence, Italy, pp. 153–164.

Bornhövd, C. & Buchmann, A. P. (2000), Semantically meaningful data exchange in loosely coupled environments, *in* ‘6th International Conference on Information Systems Analysis and Synthesis (ISAS2000)’, Orlando, Fl., USA.

Braga, R. M. M., Werner, C. M. L. & Mattoso, M. (2000), Using ontologies for domain information retrieval, *in* ‘11th International Workshop on Database and Expert Systems Applications (DEXA’00)’, DEXA Workshops, IEEE Computer Society, Greenwich, UK, pp. 836–840.

Buneman, P., Jung, A. & Ohori, A. (1991), ‘Using powerdomains to generalize relational databases’, *Theoretical Computer Science* **91**, 23–55.

Cali, A., Calvanese, D., De Giacomo, G., Lenzerini, M., Naggar, P. & Vernacotola, F. (2002), IBIS: Data integration at work, *in* ‘10th Italian Conference on Database Systems’, pp. 291–298.

Ceri, S., Gennaro, C., Paraboschi, S. & Serazzi, G. (2003), ‘Effective scheduling of detached rules in active databases’, *IEEE Transactions On Knowledge And Data Engineering* **15**(1), 2–13.

Ceri, S. & Widom, J. (1991), Deriving production rules for incremental view maintenance, *in* ‘17th International Conference on Very Large Data Bases’, Morgan Kaufmann Publishers Inc., pp. 577–589.

Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D. & Widom, J. (1994), The TSIMMIS project: Integration of heterogeneous information sources, *in* '16th Meeting of the Information Processing Society of Japan', Tokyo, Japan, pp. 7–18.

Chein, M. & Mugnier, M.-L. (1992), 'Conceptual graphs: Fundamental notions', *Revue d'Intelligence Artificielle* **6**(4), 365–406.

Chein, M. & Mugnier, M.-L. (1995), Conceptual graphs are also graphs, Research Report 95003, Universite Montpellier.

Chen, P. P.-S. (1976), 'The entity-relationship model - toward a unified view of data', *ACM Trans. Database Systems* **1**(1), 9–36.

Clamen, S. M. (1992), Type evolution and instance adaptation, Technical report, Carnegie Mellon University.

Claypool, K. T., Natarajan, C. & Rundensteiner, E. A. (1999), Optimizing the performance of schema evolution sequences, Technical Report WPI-CS-TR-99-06, Worcester Polytechnic Institute, Massachusetts.

Comyn-Wattiau, I., Akoka, J. & Lammari, N. (2003), A framework for database evolution management, *in* '2nd International Workshop on Unanticipated Software Evolution', Warsaw Poland.

Corbett, D. (2004), Interoperability of ontologies using conceptual graph theory, *in* 'Lecture Notes in Computer Science', Vol. 3127, Springer, pp. 375–387.

Cui, C., Jones, D. & O'Brien, P. (2002), 'Semantic B2B integration: Issues in ontology-based approaches', *ACM SIGMOD Record* **31**(1), 43–48.

Cui, Z. & O'Brien, P. (2000), Domain ontology management environment, *in* '33rd Hawaii International Conference on System Sciences (HICSS'00)', IEEE Computer Society, Hawaii, p. 8015.

Davidson, S., Buneman, P. & Kosky, A. (1998), 'Semantics of database transformations', *Lecture Notes in Computer Science* **1358**, 55–91.

Davies, J., Duke, A. & Stonkus, A. (2001), Ontoshare: Using ontologies for knowledge sharing, Technical report, BTexact Technologies. check entry.

De Giacomo, G., Lembo, D., Lenzerini, M. & Rosati, R. (2004), Tackling inconsistencies in data integration through source preferences, *in* '2004 International workshop on information quality in information systems (IQIS '04)', ACM Press, Paris, France, pp. 27–34.

de Vries, D., Rice, S. & Roddick, J. F. (2004), In support of mesodata in database management systems, *in* '15th International Conference on Database and Expert Systems Applications (DEXA 2004)', Lecture Notes in Computer Science, Springer Verlag, Zaragoza, Spain.

de Vries, D. & Roddick, J. (2004), Facilitating database attribute domain evolution using mesodata, *in* '3rd International Workshop on Evolution and Change in Data Management (ECDM2004)', Vol. 3289, Springer-Verlag, Shanghai, China, pp. 429–440.

Dey, D., Storey, V. C. & Barron, T. M. (1999), 'Improving database design through the analysis of relationships', *ACM Transactions on Database Systems* **24**(4), 453–474.

Do, H.-H., Melnik, S. & Rahm, E. (2002), 'Comparison of schema matching evaluations', *Lecture Notes in Computer Science* **2593**, 221–237.

Doan, A. (2002), Learning to Map between Structured Representations of Data, Phd, University of Washington.

Eisenberg, A., Melton, J., Kulkarni, K. G., Michels, J.-E. & Zemke, F. (2004), 'SQL: 2003 has been published.', *ACM SIGMOD Record* **33**(1), 119–126.

Elmasri, R. & Navathe, S. B. (2000), *Fundamentals of Database Systems*, 3rd edn, Addison-Wesley, Reading, Mass; Menlo Park, Calif.

Elmasri, R. & Navathe, S. B. (2004), *Fundamentals of Database Systems*, 4th edn, Addison-Wesley, Reading, Mass; Menlo Park, Calif.

Embley, D. W., Xu, L. & Ding, Y. (2004), 'Automatic direct and indirect schema mapping: Experiences and lessons learned.', *ACM SIGMOD Record* **33**(4), 14–19.

Embury, S. M. & Gray, P. M. D. (1999), Database internal applications, *in* N. W. Paton, ed., 'Active Rules in Database Systems', Springer, New York, pp. 339 – 366.

Ferrandina, F., Meyer, T. & Zicari, R. (1994), Implementing lazy database updates for an object database system, *in* '20th International Conference on Very Large Databases', Santiago, Chile, pp. 261–272.

Fonseca, F. T., Egenhofer, M. J., Agouris, P. & Camara, C. (2002), 'Using ontologies for integrated geographic information systems', *Transactions in GIS* **6**(3).

Franconi, E., Grandi, F. & Mandreoli, F. (2000), 'A semantic approach for schema evolution and versioning in object-oriented databases', *Lecture Notes in Computer Science* **1861**, 1048–1062.

Franconi, E., Grandi, F. & Mandreoli, F. (2001), 'Schema evolution and versioning: A logical and computational characterisation', *Lecture Notes in Computer Science* **2065**, 85–99.

Fuh, Y.-C., Dessloch, S., Chen, W., Mattos, N., Tran, B. T., Lindsay, B. G., DeMichel, L., Rielau, S. & Mannhaupt, D. (1999), Implementation of SQL3 structured types with inheritance and value substitutability, *in* M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik & M. L. Brodie, eds, '25th International Conference on Very Large Data Bases', Morgan Kaufmann, Edinburgh, Scotland, UK, pp. 565–574.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V. & Widom, J. (1997), 'The TSIMMIS approach to mediation: Data models and languages', *Journal of Intelligent Information Systems* **8**(2), 117–132.

Grandi, F. (2002), A relational multi-schema data model and query language for full support of schema versioning, *in* '10th Italian Conference on Database Systems (SEBD 2002)', pp. 323–336.

Grandi, F. (2004), SVMgr: A tool for the management of schema versioning, *in* P. Atzeni, W. Chu, H. Lu, S. Zhou & T. W. Ling, eds, '23rd International Conference on Conceptual Modeling (ER2004)', Vol. 3288, Springer-Verlag, Shanghai, China, pp. 860–861.

Gruber, T. R. (1993), 'A translation approach to portable ontology specifications', *Knowledge Acquisition* **5**(2), 199–220.

Haas, L., Miller, R., Niswonger, B., Tork Roth, M., Schwarz, P. & Wimmers, E. (1999), 'Transforming heterogeneous data with database middleware: Beyond integration', *IEEE Data Engineering Bulletin* **22**(1), 31–36.

Hakimpour, F. & Geppert, A. (2001), Resolving semantic heterogeneity in schema integration, *in* 'International conference on Formal Ontology in Information Systems (FOIS01)', ACM Press, Ogunquit, Maine, USA., pp. 297–308.

He, B. & Chang, K. C.-C. (2004), 'A holistic paradigm for large scale schema matching.', *ACM SIGMOD Record* **33**(4), 20–25.

Horrocks, I. (2000), A denotational semantics for standard OIL and instance OIL, Technical report, Department of Computer Science University of Manchester, UK.

Horrocks, I. (2002*a*), 'DAML+OIL: a description logic for the semantic web', *IEEE Data Engineering Bulletin* **25**(1), 4–9.

Horrocks, I. (2002*b*), DAML+OIL: A reason-able web ontology language, *in* C. S. Jensen, K. G. Jeffery, J. Pokorny, S. Altenis, E. Bertino, K. Böhm & M. Jarke, eds, '8th International Conference on Extending Database Technology', Vol. 2287 of *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, Prague, Czech Republic, pp. 2–13.

Hull, R. (1986), 'Relative information capacity of simple relational database schemata', *Society for Industrial and Applied Mathematics* **15**(3), 856 – 886.

Hull, R. (1997), Managing semantic heterogeneity in databases: A theoretical perspective, *in* 'ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)', ACM Press, Tucson, AZ, pp. 51–61.

ISO/ANSI (2003), Information technology - database languages - SQL - part 2: Foundation (SQL/foundation), *in* J. Melton, ed., 'ISO/IEC 9075-2:2003 (E)', ISO/IEC JTC 1/SC 32, Geneva.

Jasper, R. & Uschold, M. (1999), A framework for understanding and classifying ontology applications, *in* 'IJCAI99 Workshop on Ontologies and Problem-Solving Methods(KRR5)', Stockholm, Sweden.

Jensen, C. S., Clifford, J., Elmasri, R., Gadia, S. K., Hayes, P., Jajodia, S., Dyreson, C., Grandi, F., Kafer, W., Kline, N., Lorentzos, N., 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. & Wiederhold, G. (1998), A consensus glossary of temporal database concepts - February 1998 version, *in* O. Etzion, S. Jajodia &

S. Sripada, eds, 'Temporal Databases - Research and Practice', Vol. 1399 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 367–405.

Kashap, V. & Sheth, A. P. (1996), 'Semantic and schematic similarities between database objects: a context-based approach', *The VLDB Journal* **5**(4), 276 – 304.

Kedad, Z. & Métais, E. (1999), Dealing with semantic heterogeneity during data integration., *in* J. Akoka, B. Mokrane, I. Comyn-Wattiau & E. Métais, eds, '18th International Conference on Conceptual Modelling', Vol. 1728 of *Lecture Notes in Computer Science*, Springer, Paris France, pp. 325–339.

Klein, M. (2001), Combining and relating ontologies: an analysis of problems and solutions., *in* A. Gomez-Perez, M. Gruninger, H. Stuckenschmidt & M. Uschold, eds, 'Workshop on Ontologies and Information Sharing, IJCAI'01', Seattle, USA.

Klein, M. (2002), Supporting evolving ontologies on the internet, *in* A. B. Chaudhri, R. Unland, C. Djeraba & W. Lindner, eds, 'XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops XMLDM, MDDE, and YRWS', Vol. 2490 of *Lecture Notes in Computer Science*, Springer, Prague, Czech Republic, pp. 597–606.

Klein, M. & Fensel, D. (2001), Ontology versioning for the semantic web, *in* 'International Semantic Web Working Symposium (SWWS)', Stanford University, California, USA.

Klein, M., Fensel, D., Kiryakov, A. & Ognyanov, D. (2002), Ontology versioning and change detection on the web, *in* A. Gómez-Pérez & V. R. Benjamins, eds, 'Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, Thirteenth International Conference', Lecture Notes in Computer Science, Springer, Siguenza, Spain, pp. 197–212.

Lakshmanan, L. V. S., Sadri, F. & Subramanian, S. N. (1999), On efficiently implementing SchemaSQL on an SQL database system, *in* M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik & M. L. Brodie, eds, '25th International Conference on Very Large Data Bases', Morgan Kaufmann, Edinburgh, Scotland, UK, pp. 471–482.

Lemke, T. (1994), Schema evolution in OODBMS: A selective overview of problems and solutions., Technical Report IDEA.WP.22.O.002, University of Bonn.

Lenzerini, M. (2002), Data integration: a theoretical perspective, *in* '21st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems', ACM Press, Madison, Wisconsin, pp. 233–246.

Li, C., Yerneni, R., Vassalos, V., Garcia-Molina, H., Papakonstantinou, Y., Ullman, J. & Valiveti, M. (1998), Capability based mediation in TSIMMIS, *in* '1998 ACM SIGMOD international conference on Management of data', ACM Press, Seattle, Washington, United States, pp. 564–566.

Liu, C.-T., Chang, S.-K. & Chrysanthis, P. K. (1994), Database schema evolution using EVER diagrams, *in* 'Workshop on advanced visual interfaces', Advanced Visual Interfaces, ACM Press New York, NY, USA, Bari, Italy, pp. 123 – 132.

Liu, L., Zicari, R., Hursch, W. L. & Lieberherr, K. J. (1997), 'The role of polymorphic reuse mechanisms in schema evolution in an object-oriented database', *Knowledge and Data Engineering* **9**(1), 50–67.

Maier, D. (1983), *The Theory of Relational Databases*, Computer Science Press.

McBrien, P. & Poulovassilis, A. (1997), A formal framework for ER schema transformation, *in* 'International Conference on Conceptual Modeling / the Entity Relationship Approach', pp. 408–421.

McBrien, P. & Poulovassilis, A. (1998*a*), 'A formalisation of semantic schema integration', *Information Systems* **23**(5), 307–334.

McBrien, P. & Poulovassilis, A. (1998*b*), 'A general formal framework for schema transformation', *Data and Knowledge Engineering* **28**(1), 47–71,.

McBrien, P. & Poulovassilis, A. (2002), Schema evolution in heterogeneous database architectures, a schema transformation approach, *in* '14th International Conference on Advanced Information Systems Engineering (CAiSE'02)', Springer-Verlag, pp. 484–499.

Meersman, R. & Jarrar, M. (2002), Formal ontology engineering in the DOGMA approach, *in* R. Meersman & Z. Tari, eds, 'CoopIS/DOA/ODBASE 2002', Vol. 2519, Springer-Verlag, pp. 1238–1254.

Melton, J. & Simon, A. R. (2002), *SQL:1999 Understanding Relational Language Components*, Academic Press, San Francisco.

Miller, R., Hernandez, M., Haas, L., Yan, L., Ho, C. T. H., Fagin, R. & Popa, L. (2001), 'The Clio Project: Managing heterogeneity', *ACM SIGMOD Record* **30**(1), 78–83.

Miller, R. J., Ioannidis, Y. E. & Ramakrishnan, R. (1993), The use of information capacity in schema integration and translation, *in* R. Agrawal, S. Baker & D. Bell, eds, '19th International Conference on Very Large Data Bases, VLDB'93', Morgan Kaufmann, Palo Alto, CA, Dublin, Ireland, pp. 120–133.

Miller, R. J., Ioannidis, Y. E. & Ramakrishnan, R. (1994*a*), 'Schema equivalence in heterogeneous systems: Bridging theory and practice', *Information Systems* **19**(1), 3–31.

Miller, R. J., Ioannidis, Y. E. & Ramakrishnan, R. (1994*b*), Schema intension graphs: A formal model for the study of schema equivalence, Technical report, University of Wisconsin-Madison.

Mooney, C. H., de Vries, D. & Roddick, J. F. (2005), A multi-level framework for the analysis of sequential data, *in* S. J. Simoff & G. J. Williams, eds, 'Data Mining: Theory, Methodology, Techniques, and Applications', Lecture Notes in Artificial Intelligence, Springer.

Moshier, A. (2000), Mathematical foundations of domain theory, Technical report, University of Birmingham. unprinted.

Motik, B., Maedche, A. & Volz, R. (2002), A conceptual modeling approach for semantics-driven enterprise applications, *in* R. Meersman & Z. Tari, eds, 'On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002', Vol. 2519 of *Lecture Notes in Computer Science*, Springer, Irvine, California, USA, pp. 1082–1099.

Motro, A., Berlin, J. & Anokhin, P. (2004), 'Multiplex, fusionplex, and autoplex - three generations of information integration', *ACM SIGMOD Record* **33**(4), 51–57.

Munkres, J. R. (1975), *Topology: A First Course*, Prentice Hall, Englewood Cliffs, New Jersey.

MySQL (2003), 'SQL shareware software: documentation and source code'. `http://www.mysql.com`

Ng, W. (1998), Inferring functional dependencies in linearly ordered databases, *in* G. Quirchmayr, E. Schweighofer & T. J. M. Bench-Capon, eds, '9th International Conference on Database and Expert Systems Applications (DEXA98)', Vol. 1460, Springer-Verlag, Heidelberg, p. 186195.

Ng, W. (2001), 'An extension of the relational data model to incorporate ordered domains', *ACM Transactions on Database Systems* **26**(3), 344–383.

Nirenburg, S. & Raskin, V. (2004), The static knowledge sources: Ontology, fact database and lexicons, *in* 'Ontological Semantics', Language, Speech, and Communication, MIT Press, pp. 191–246.

Noy, N. F. (2004), 'Semantic integration: a survey of ontology-based approaches', *ACM SIGMOD Record* **33**(4), 65–70.

Noy, N. F. & Klein, M. (2002), Ontology evolution: Not the same as schema evolution, Technical Report SMI-2002-0926, Standford Medical Informatics.

Parent, C. & Spaccapietra, S. (1998), 'Issues and approaches of database integration', *Communications of the ACM* **41**(5), 166–178.

Parent, C., Spaccapietra, S. & Zimányi, E. (2000), MurMur: Database management of multiple representations, *in* 'AAAI-2000 Workshop on Spatial and Temporal Granularity', Austin, Texas, pp. 83–86.

Parsons, J. & Wand, Y. (2000), 'Emancipating instances from the tyranny of classes in information modeling.', *ACM Transactions on Database Systems* **25**(2), 228– 260.

Partridge, C. (2002), The role of ontology in integrating semantically heterogeneous databases, Technical Report 05/02, National Research Council, LADSEB-CNR.

Peters, R. J. & Özsu, M. T. (1997), 'An axiomatic model of dynamic schema evolution in objectbase systems', *ACM Transactions on Database Systems* **22**(1), 75–114.

Qian, X. (1996), Correct schema transformations, *in* P. M. G. Apers, M. Bouzeghoub & G. Gardarin, eds, 'Advances in Database Technology - 5th International Conference on Extending Database Technology (EDBT'96)', Vol. 1057 of *Lecture Notes in Computer Science*, Springer, Avignon, France, pp. 114–128.

Rahm, E. & Bernstein, P. A. (2001*a*), On matching schemas automatically, Technical MSR-TR-2001-17, University of Leipzig.

Rahm, E. & Bernstein, P. A. (2001*b*), 'A survey of approaches to automatic schema matching', *VLDB Journal* **10**(4), 334–350.

Ram, S. & Park, J. (2004), 'Semantic conflict resolution ontology (SCROL): An ontology for detecting and resolving data and schema-level semantic conflicts', *IEEE Transactions on Knowledge and Data Engineering* **16**(2), 189–202.

Ram, S. & Ramesh, V. (1999), Schema integration: past, present, and future, *in* A. Elmagarmid, M. Rusinkiewicz & A. Sheth, eds, 'Management of Heterogeneous and Autonomous Database Systems', Morgan Kaufmann Publishers Inc., pp. 119–155.

Rashid, A. (2002), Aspect-oriented schema evolution in object databases: A comparative case study, Technical report, Computing Department, Lancaster University, Lancaster LA1 4YR, UK. check entry.

Rice, S. & Roddick, J. F. (2000), Lattice-structured domains, imperfect data and inductive queries, *in* M. T. Ibrahim, J. Kng & N. Revell, eds, '11th International Conference on Database and Expert Systems Applications (DEXA2000)', Vol. 1873 of *Lecture Notes in Computer Science*, Springer, Greenwich, London, UK, pp. 664–674.

Rice, S., Roddick, J. F. & de Vries, D. (2006), Defining and implementing domains with multiple types using mesodata modelling techniques, *in* '3rd Asia-Pacific Conference on Conceptual Modelling (APCCM 2006)', Hobart, Australia.

Roddick, J. F. (1995), 'A survey of schema versioning issues for database systems', *Information and Software Technology* **37**(7), 383–393.

Roddick, J. F., Al-Jadir, L., Bertossi, L., Dumas, M., Estrella, F., Gregersen, H., Hornsby, K., Lufter, J., Mandreoli, F., Männistö, T., Mayol, E. & Wedemeijer, L. (1999), 'Evolution and change in data management - issues and directions', *ACM SIGMOD Record* **29**(1), 21–25.

Roddick, J. F., Craske, N. G. & Richards, T. J. (1993), A taxonomy for schema versioning based on the relational and entity relational models, *in* '12th International Conference on Entity-Relationship Approach', pp. 143–154.

Roddick, J. F., Hornsby, K. & de Vries, D. (2003), A unifying semantic distance model for determining the similarity of attribute values, *in* M. Oudshoorn, ed., '26th Australasian Computer Science Conference (ACSC2003)', Vol. 16, ACS, Adelaide, Australia, pp. 111–118.

Rosenthal, A., Seligman, L. J. & Renner, S. (2004), 'From semantic integration to semantics management: case studies and a way forward.', *ACM SIGMOD Record* **33**(4), 44–50.

Sattler, K.-U., Geist, I. & Schallehn, E. (2005), 'Concept-based querying in mediator systems', *The VLDB Journal* **14**(1), 97–111.

Shankaranarayanan, G. & Ram, S. (2003), Research issues in database schema evolution - the road not taken, Technical Report Technical Report 2003-15, University of Arizona.

Sintek, M., Tschaitschian, B., Abecker, A. & Bernardi, A. (2000), Using ontologies for advanced information access, *in* J. Domingue, ed., '3rd International Conference and Exhibition on The Practical Application of Knowledge Management (PAKeM 2000)', Manchester, UK.

Sjøberg, D. (1993), 'Quantifying schema evolution', *Information and Technology Software* **35**(1), 35 – 44.

Sotnykova, A., Monties, S. & Spaccapietra, S. (2000), Semantic integration in MADS conceptual model, *in* '17th International CODATA Conference', Vol. Integration of Heterogeneous Databases and Data Warehousing, Baveno, Italy.

Sowa, J. F. (2000), *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co., Pacific Grove, CA, USA.

Sowa, J. F. (2001), 'Conceptual graph standard'.
`http://users.bestweb.net/ sowa/cg/cgstand.htm`

Spaccapietra, S., Parent, C., Vangenot, C. & Cullot, N. (2004), On using conceptual modeling for ontologies., *in* C. Bussler, S.-k. Hong, W. Jun, R. Kaschek, Kinshuk, S. Krishnaswamy, S. W. Loke, D. Oberle, D. Richards, A. Sharma, Y. Sure & B. Thalheim, eds, 'Web Information Systems - WISE 2004 International Workshops', Vol. 3307 of *Lecture Notes in Computer Science*, Springer, Brisbane, Australia, pp. 22–33.

Spaccapietra, S., Yu, S. & Al-Jadir, L. (2005), Somebody, sometime, somewhere, something, *in* 'International Workshop on Ubiquitous Data Management (UDM2005)In Memoriam Prof. Yahiko Kambayashi, In conjunction with IEEE ICDE 2005', Vol. 2005, Tokyo, Japan.

Spyns, P., Meersman, R. & Jarrar, M. (2002), 'Data modelling versus ontology engineering', *ACM SIGMOD Record* **31**(4), 12–17.

Staudt Lerner, B. (2000), 'A model for compound type changes encountered in schema evolution', *ACM Transactions on Database Systems* **25**(1), 83 – 127.

Stonebraker, M., Jhingran, A., Goh, J. & Potamianos, S. (1990), On rules, procedure, caching and views in data base systems, *in* '1990 ACM SIGMOD international conference on Management of data', ACM Press, Atlantic City, New Jersey, United States, pp. 281–290.

Stonebraker, M. & Moore, D. (1996), *Object-Relational DBMSs: The Next Great Wave*, Morgan Kaufman, San Francisco.

Türker, C. (2000), Schema evolution in SQL-99 and commercial (object-) relational DBMS, *in* '9th International Workshop on Foundations of Models and Languages for Data and Objects', Database Schema Evolution and Meta-Modeling, Dagstuhl Castle Germany.

Uschold, M. & Gruninger, M. (2004), 'Ontologies and semantics for seamless connectivity', *ACM SIGMOD Record* **33**(4), 58–64.

Ventrone, V. & Heiler, S. (1991), 'Semantic heterogeneity as a result of domain evolution', *ACM SIGMOD Record* **20**(4), 16–20.

Vianu, V. (2001), A web odyssey: from Codd to XML, *in* '20th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS '01)', ACM Press, Santa Barbara, California, USA, pp. 1–15.

Wache, H., Vogele, T., Stuckenschmidt, H., Schuster, G., Neumann, H. & Hubner, S. (2001), Ontology-based integration of information a survey of existing approaches, *in* 'IJCAI-01 Workshop: Ontologies and Information Sharing', Seattle, WA, pp. 108–117.

Wache, H., Vogele, T., Stuckenschmidt, H., Schuster, G., Neumann, H. & Hubner, S. (2002), 'Ontology-based integration of information a survey of existing approaches'.

Wand, Y., Storey, V. C. & Weber, R. (1999), 'An ontological analysis of the relationship construct in conceptual modeling.', *ACM Transactions on Database Systems* **24**(4), 494–518.

Waszkiewicz, P. (2003), How do domains model topologies?, *in* '19th Conference on the Mathematical Foundations of Programming Semantics', Institute of Computer Science, Jagiellonian University, Krakow, Poland Boole Centre for Research in Informatics, University College, Cork, Ireland, Montreal Canada.

Wei, H.-C. & Elmasri, R. (2000), 'Schema versioning and database conversion techniques for bi-temporal databases', *Annals of Mathematics and Artificial Intelligence* **30**(1-4), 23–52.

Xu, L. & Poulovassilis, A. (1997), A method for integrating deductive databases, *in* 'British National Conference on Databases', pp. 215–231.

Yan, L. L., Miller, R. J., Haas, L. M. & Fagin, R. (2001), 'Data-driven understanding and refinement of schema mappings', *ACM SIGMOD Record* **30**(2), 485 – 496.

Yugopuspito, P. & Araki, K. (1999), Evolution of relational database to object-relational database in abstract level, *in* 'International Workshop on Principles of Software Evolution', Fukuoka City, Japan, pp. 103–107.

Zhou, L., Rundensteiner, E. A. & Shin, K. G. (1997), 'Schema evolution of an object-oriented real-time database system for manufacturing automation', *Knowledge and Data Engineering* **9**(6), 956–977.