Georgia State University
Digital Archive @ GSU

Computer Science Theses | Department of Computer Science

8-2-2007

A Domain-Specific Conceptual Query System

Xiuyun Shen

Follow this and additional works at: http://digitalarchive.gsu.edu/cs_theses

Recommended Citation
Shen, Xiuyun, "A Domain-Specific Conceptual Query System" (2007). *Computer Science Theses.* Paper 46.

This Thesis is brought to you for free and open access by the Department of Computer Science at Digital Archive @ GSU. It has been accepted for inclusion in Computer Science Theses by an authorized administrator of Digital Archive @ GSU. For more information, please contact digitalarchive@gsu.edu.

A DOMAIN-SPECIFIC CONCEPTUAL QUERY SYSTEM

by

XIUYUN SHEN

Under the Direction of Rajshekhar Sunderraman

ABSTRACT

This thesis presents the architecture and implementation of a query system resulted from a domain-specific conceptual data modeling and querying methodology. The query system is built for a high level conceptual query language that supports dynamically user-defined domain-specific functions and application-specific functions. It is DBMS-independent and can be translated to SQL and OQL through a normal form. Currently, it has been implemented in neuroscience domain and can be applied to any other domain.

INDEX WORDS: Bioinformatics, Database, Conceptual, Domain-Specific, Life Science Data, Normal Form, Query Language, User-Defined Function

A DOMAIN-SPECIFIC CONCEPTUAL QUERY SYSTEM

by

XIUYUN SHEN

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2007

Copyright by

XIUYUN SHEN

2007

A DOMAIN-SPECIFIC CONCEPTUAL QUERY SYSTEM

by

XIUYUN SHEN

Major Professor: Rajshekhar Sunderraman
Committee: Yangqing Zhang
Yingshu Li

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
August 2007

ACKNOWLEDGMENTS

I am deeply indebted to my advisor, Dr. Rajshekhar Sunderraman, for his constant support. Without his help, this work would not be possible. I also would like to thank the members of my committee - Dr. Yanqing Zhang, and Dr. Ying shu Li. Their valuable comments and suggestions enhanced the quality of this thesis work. I would also like to thank Hao Tian for giving me so much advice on my thesis and teaching me how to solve problems.

I also wish to thank Shaochieh Ou, and my colleagues at GSU, their advice is appreciated.

Lastly, I would like to thank my whole family for their support. I dedicate this thesis to my mother and my father.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ..... **iiv**
LIST OF FIGURES ..... **vvii**
LIST OF TABLES ..... **vivii**
LIST OF ABBREVIATIONS ..... **viivii**
CHAPTER 1 INTRODUCTION **1**
CHAPTER 2 DATA MODELS AND QUERY LANGUAGES **3**
2.1 Data Models 13
2.1.1 Entity-Relationship data models 5
2.1.2 Relational data models 6
2.1.3 Object-Oriented data models 8
2.1.4 Object-Role modeling 10
2.1.5 Data models in XML 10
2.1.6 Other data models 112
2.2 Query Languages 13
2.2.1 Traditional DBMS-supported query languages 15
2.2.2 Conceptual query languages 18
2.2.3 Domain-Specific query languages 19
Chapter 3 DOMAIN-SPECIFIC CONCEPTUAL QUERY SYSTEM **20**
3.1 System Architecture 20
3.2 From DSC-DM to Database Schema 22
3.2.1 User-Defined functions - UDF.XML 22
3.2.2 Other information - DSC.XML 24
3.2.3 Mapping from DSC-DM to database schema - Mapping.XML 24
3.2.4 Examples 266
3.3 From DSC-QL to SQL/OQL 29
3.3.1 DSC-QL syntax 29
3.3.2 Normal form of DSC-QL 31
3.3.3 Translation of DSC-QL 32
3.3.4 DSC-QL to SQL/OQL Example 34
3.4 DSC-ML 35
3.4.1 DSC-ML syntax 36
3. 4. 2 DSC-ML examples 36
3.5 Source Code 37
Chapter 4 CONCLUSION AND FUTURE WORK **38**
Bibliography **411**

LIST OF FIGURES

Figure 2.1 ER diagram of the grade book example ……..6

Figure 2.2 Relational data model and database example …….7

Figure 2.3 Object-Oriented data model and database example ……9

Figure 2.4 Grade book database in XML format ……11

Figure 2.5 Query language history ……..14

Figure 3.1 System Architecture of Query System ……...20

Figure 3.2 a part of DSC-DM for neuroscience domain …….26

Figure 3.3 Data structure diagram of DSC-DM for an enrollment system …….37

LIST OF TABLES

TABLE 2.1 SQL STANDARDS ……15

LIST OF ABBREVIATIONS

Backus-Naur Form	BNF
Database Management System	DBMS
Data Definition Language	DDL
Data Manipulation Language	DML
Domain-Specific Conceptual Data Model	DSC-DM
Domain-Specific Conceptual Manipulation Language	DSC-ML
Domain-Specific Conceptual Query Language	DSC-QL
Domain-Specific Function	DSF
Enhanced Entity-Relationship	EER
Entity-Relationship	ER
Java Database Connectivity	JDBC
Object Definition Language	ODL
Object Database Management Group	ODMG
Object Management Group	OMG
Object-Oriented Database Schema	OODBS
Object-Oriented Data Model	OODM
Object Query Language	OQL
Object-Role Modeling	ORM
Relational Algebra	RA
Relational Database Schema	RDBS
Relational Data Model	RDM
Structured Query Language	SQL
User-Defined Function	UDF
Unified Modeling Language	UML
World Wide Web Consortium	W3C
Extensible Markup Language	XML

CHAPTER 1 INTRODUCTION

A new trend of current database applications is to provide a more and more powerful query capability to end-users so that they can formulate complicated queries to interact with the data in underlying databases directly. It is obvious that traditional query languages like SQL and OQL cannot meet this requirement, and an alternative is conceptual query language [1, 2]. Although form-based query interface is another readily usable solution, it typically suffers the limited expressive power and supports only predefined types of queries. Form-based query interfaces are application-dependent, can rapidly become obsolete as the underlying database schema evolves, and have to be re-programmed.

A domain-specific conceptual data modeling and querying methodology has been proposed in [3]. It provides a new user-oriented conceptual query approach with the support to dynamic user-defined functions (both domain-specific functions and application-specific functions). In this thesis, we present the implementation of this methodology and illustrate the conceptual query approach in neuroscience domain (built on an object-oriented DBMS, EyeDB [4]). DSC-DM (domain-specific conceptual data model) and DSC-QL (domain-specific conceptual query language) are two key components of the methodology. DSC-DM includes a data structure diagram, user-defined functions (UDF), and two special tables (meta-attribute table and annotation table). The data structure diagram is similar to the Enhanced Entity-Relationship (EER [5]) model with a more restricted inheritance relationship that must be total inheritance. UDF includes domain-specific functions (DSFs) and application-specific functions (ASFs), which are defined by domain-experts and model creators. The signature, semantics, and implementation of UDF are stored in an XML file (i.e. UDF.xml) that is loaded by DSC-QL Normalizer at run time for query normalization. Therefore, UDF can be

easily updated at any time without changing rest parts of the system. DSC-QL is a high level conceptual query language based on DSC-DM. It has simple syntax and uses only the concepts, relationships, and functions defined in DSC-DM. Besides, the supports to super-class and dot-path expression make it more practically usable and comfortable for end-users to navigate the concepts in the model. Consequently, end-users can formulate their queries only based on familiar concepts and functions in their research area. Through normalization and translation, DSC-QL can be translated into SQL, OQL and others. The normal form and semantics of DSC-QL have been formally defined in [6].

In general, all information in DSC-DM created by domain-experts and database specialists will be stored in several XML files and used to generate a matching database schema. The DSC-QL query formulated by end-users will be parsed, validated, and normalized. The normalized DSC-QL query, in turn, is translated into an equivalent query that can be recognized by the underlying DBMS. This query system has been implemented in neuroscience domain [7].

CHAPTER 2 DATA MODELS AND QUERY LANGUAGES

Data describe phenomena that reflect a perception of the world. That is, data correspond to discrete, recorded facts about phenomena from which we gain information about the world [13]. Data management is a very broad concept; it includes all the disciplines which are related to managing data as a valuable resource. The official definition provided by Data Management Association (DAMA) [14] is that "Data Resource Management is the development and execution of architectures, policies, practices and procedures that properly manage the full data lifecycle needs of an enterprise." PC Magazine [15] gives another definition that describes the data management at different levels: (1) the part of the operating system that manages the physical storage and retrieval of data on a disk or other device; (2) software that allows users to create, store, retrieve and manipulate files interactively; (3) the function that manages data as an organizational resource; (4) the management of all data/information in an organization, which includes data administration, the standards for defining data and the way in which people perceive and use them. From the definitions above, we can see that data management includes a variety of topics, each of which involves many different technologies. Following are several common data management topics: data model, data storage, data query, data collection, database administration, data backup and security, data quality assurance, data warehousing and mining, data analysis, data integration and distribution, data validation, data cleaning and curation, data monitoring, and so on. Data model and data query are two most basic and common topics.

2.1 Data Models

A data model is a data representation in an organization or system. The representation of data is the first step of data management. As it is mentioned above, data describe a

perception of the world, from which human beings. Information is an increment of knowledge that can be inferred from data [16]. Since both data and information play the same significant roles in data management, we do not distinguish them from each other in our research. And we use the term “data” and “information” interchangeably in the following contents.

From the application perspective, the purpose of data modeling is to develop an accurate model or graphical representation, of a client's information and its needs. That is, Data modeling functions as a “blueprint” for building the physical database. The representation of information previously uses natural language, and the main sources of data and knowledge include papers, books, literatures and reports etc. However, natural language is a very ambiguous tool for representing information. It is not precise and machine-processable even though it works very well for the communication between human beings. As an alternative, data model is devised for computer-oriented representation of information. Another goal of data model is to provide a sufficiently abstract interpretation of an object in the world so that minor difference in understanding will not change the nature of the object. A formal mathematical definition of data model was given in [13].

There are different ways to categorize data models. One way is to divide them into two groups: strictly typed and loosely typed data models. In strictly typed data models, all data must belong to certain category, while loosely typed data models do not have this limitation. The current commonly used categorization is to group data models based on their functionality into conceptual data models and physical models. Conceptual data models which sometimes called domain models are built to document high-level abstract ideas and are used to identify and document domain concepts with project stakeholders. Logical data models focuses on exploring domain concepts and their relationships and relationship cardinalities.

Thus, logical data models depict the logical entity types, the data attributes describing those entities, and the relationships between the entities. Data definition languages (DDL) can be generated at this level. Conceptual data models are often created as alternatives to logical data models. Physical data models, also the best known data models, determine the actual design of a database. They are used to design the internal schema of a database and depict data tables, the data columns of those tables and the relationships between data tables.

The follow sections briefly introduce a few common used data models.

2.1.1 Entity-Relationship data models

Entity-Relationship (ER) data model is one of the most common conceptual data models. It was originally proposed by Chen in 1976 [19]. A basic component of this model is the Entity-Relationship diagram which can be used to represent data objects. This data model is very commonly used for database design by the database designer; The Entity-Relationship Model is a conceptual data model, the basic constructs of the ER model includes entities, relationships, and attributes. Entities are concepts, real or abstract, about which information is being collected. Relationships are associations between the entities. Attributes are properties which describe the entities or relationships between entities. There is no standard for representing data objects in ER diagrams. Different methodologies can use their own notation. The common ones are Bachman, crow's foot, and IDEFIX. Most constraints in ER data model are explicit, and can be expressed in the ER diagram using different notations. ER data model has many limitations.Since Chen introduced the first ER model, ER data models have been extended or revised in many different ways in order to overcome their limitations as described by Teorey et al. (1986) [20], Gogolla and Hohenstein (1991) [21], Elmasri et al. [22], and Badia (2004) [23]. Following is an example:

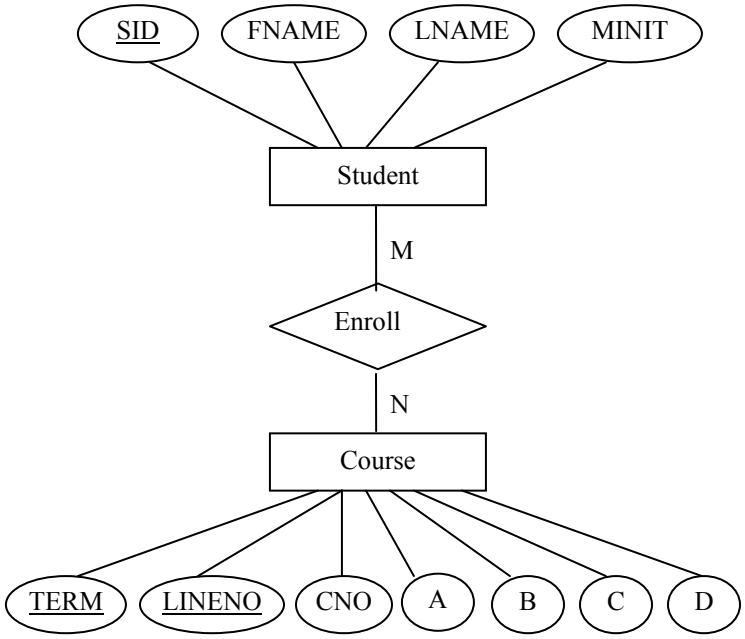

Figure 2.1 ER diagram of the grade book example

2.1.2 Relational data models

The **relational data model** for database management is a database model based on predicate logic and set theory. It was first formulated and proposed in 1969 by Edgar Codd [17] with aims that included avoiding, without loss of completeness, the need to write computer programs to express database queries and enforce database integrity constraints.

The Relation is the basic element in a relational data model. A relation consists of a heading and a body. A heading is a set of attributes. A body (of an *n*-ary relation) is a set of *n*-tuples. Strictly speaking, a relational database is a collection of relations (frequently called tables). Examples of relational databases are Oracle, Microsoft SQL Server, MySQL, and others. The major advantages of relational data models are that they have simple structures and constraints; the strong mathematical foundation is also a very important advantage of

relational data model. The some major disadvantages of relational data models are that it is very hard to model complex data and only support some primitive data types.

In this chapter, a company system from [5] is used as the example to illustrate the different data models. Its relational data model and one sample database are as following.

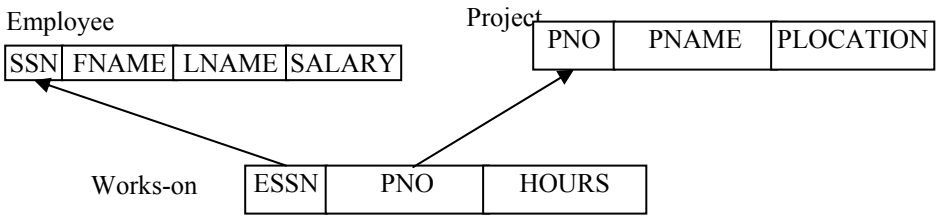

Figure 2.2 (a)

Employe

SSN	FNAME	LNAME	SALARY
123456789	John	Smith	30000
333445656	Franklin	Wong	40000
999887777	Alleta	Zelaya	25000

Works-on

ESSN	PNO	HOURS
123456789	1	32.5
123456789	2	7.5
333445656	2	10.0
333445656	3	10.0
99988777	30	30.0

Project

PNO	PNAME	PLOCATION
1	ProductX	Bellaire
2	ProductY	Sugarland
3	ProductZ	Houston
30	Newbenefit	Stanford

Figure 2.2 (b)

Figure 2.2 Relational data model and database example [5]
(a) The relational data model; (b) a database instance of a company system

The specification EMPLOYEE (SSN, FNAME, LNAME, SALARY) is an example of a relation scheme. A relation scheme generally represents an entity type in relational data models. Various restrictions on data can be specified on a relational database in the form of constraints. Structured Query Language (SQL) is standard language for relational database. And the operations based on relational data models are relational calculus and algebra.

2.1.3 Object-Oriented data models

Object-Oriented data models (OODMs) are originated from the object-oriented concept. An object-oriented data model consists of the following basic object-oriented concepts: (1) object and object identifier: any entity in a real world is modeled as a object, each object associates a unique object ID (OID) which is like primary keys of a relation in relational data model. (2) Attributes and methods: any object has a state and a behavior which are accessed from outside. (3) Class: class is a means of grouping all the objects which share the same set of attributes and methods. Each object is a concrete instance *of an* object class defining the attributes and operations of all its instances. (4) Class hierarchy and inheritance.

Compared to RDBMS, Object oriented data base systems have a lot of advantages: composite objects and relationships, offering more complex data types and class hierarchy. The disadvantage of OODBMS is that modification of database shema even the modification limited to one class often means system recompiling.

OQL is standard query language for object-oriented database. Some common Object Oriented Database Management Systems are Object Store, Ozone and EyeDB. Figure 2.3 has an OODM and object-oriented database example of the grade book system example [18].

```
Class Student {
   attribute integer SID;
   attribute string FNAME;
   attribute string LNAME;
   attribute string MINIT;
   relationship set<Course> enroll inverse Course::enrolledBy;
};
Class Course {
   attribute     string      TERM;
   attribute     integer     LINENO;
   attribute     string      CNO;
   attribute     integer     A;
   attribute     integer     B;
   attribute     integer     C;
   attribute     integer     D;
   relationship set<Student> enrolledBy inverse Students::enroll;
};
```

Figure 2.3 (a)

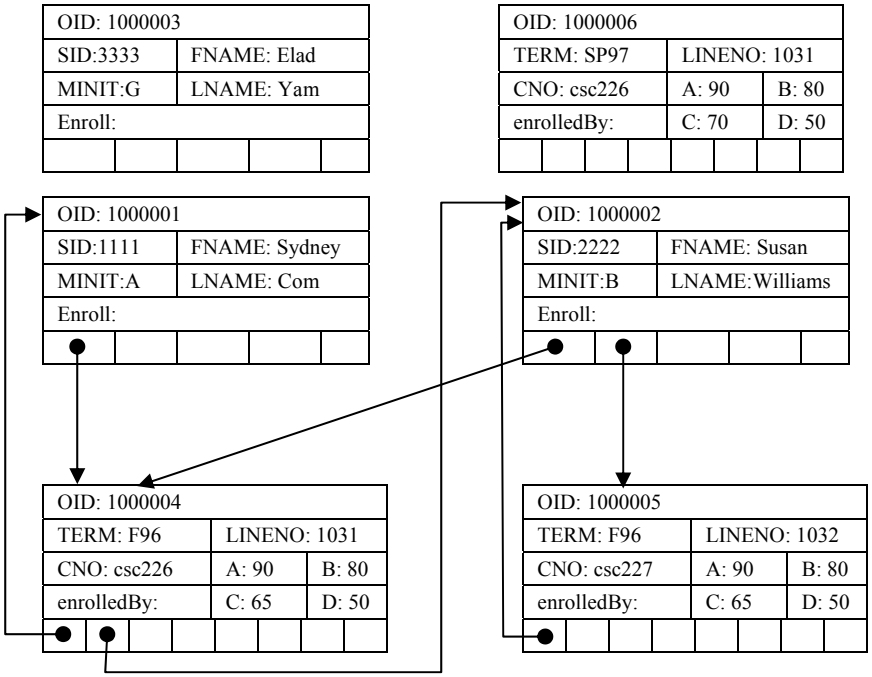

Figure 2.3 (b)

Figure 2.3 Object-Oriented data model and database example [18]

(a) The OODM; (b) an object-oriented database of a grade book system

2.1.4 Object-Role modeling

Object-Role Modeling (ORM) [24] is a powerful method used for designing and querying database models at the conceptual level. It is a generic term for a conceptual modeling approach which avoids the notion of attribute by picturing the application world in terms of objects that play roles individually or in relationships. ORM facilates detailed information modeling because it is sematically rich, it has rich graphic notion that often makes it eable to capture more business rules, and its notions are easily populated. Compared to other data models, ORM are also easier to validate and evolve.The use of ORM for conceptual and relational database design is becoming more popular.

2.1.5 Data models in XML

The term "Data models in XML" used in our research refers to models defined or expressed in XML. We talk about "Data models in XML" here because many life science data models such as CellML [54], BrainML [53] and SBML [55] are currently defined in a markup language which follows XML standards.

The Extensible Markup Language (XML) is general-purpose markup language which is recommended by W3C, and has recently emerged as a new standard for data representation and exchange on the internet. XML document has a tree-like structure and its specification defines a standard way to add markup to documents. In current database community, XML gains more and more attation and there are already several XML database management system developed. One significant advantage of XML document is that it is a simultaneously human- and machine-readable format. Figure 2.4 (next page) gives the XML schema and representation of grade book sample database.

```
<xsd:schema …>
  <xsd:element name="Course">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="TERM" type="xsd:string"
                                  use="required"/>
        <xsd:element name="LINENO" type="xsd:string"
                                  use="required"/>
        <xsd:element name="CNO" type="xsd:string"
                                  use="required"/>
        <xsd:element name="A" type="xsd:decimal"
                                  use="required"/>
        <xsd:element name="B" type="xsd:decimal"
                                  use="required"/>
        <xsd:element name="C" type="xsd:decimal"
                                  use="required"/>
        <xsd:element name="D" type="xsd:decimal"
                                  use="required"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Student">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="SID" type="xsd:string"
                                  use="required"/>
        <xsd:element name="FNAME" type="xsd:string"
                                  use="required"/>
        <xsd:element name="LNAME" type="xsd:string"
                                  use="required"/>
        <xsd:element name="MINIT" type="xsd:string"/>
        <xsd:sequence minOccurs="0" maxOccurs="5">
          <xsd:element ref="Course"/>
        </xsd:sequence>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="GradeBook">
    <xsd:complexType>
      <xsd:sequence minOccurs="1"
                    maxOccurs="unbounded">
        <xsd:element ref="Student"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Figure 2.4 (a)

```
<BradeBook>
  <Student>
    <SID>1111</SID>
    <FNAME>Sydney</FNAME>
    <LNAME>Corn</LNAME>
    <MINIT>A</MINIT>
    <Course>
      <TERM>F96</TERM>
      <LINENO>1031</LINENO>
      <CNO>csc226</CNO>
      <A>90</A>
      <B>80</B>
      <C>65</C>
      <D>50</D>
    </Course>
  </Student>
  <Student>
    <SID>2222</SID>
    <FNAME>Susan</FNAME>
    <LNAME>Williams</LNAME>
    <MINIT>B</MINIT>
    <Course>
      <TERM>F96</TERM>
      <LINENO>1031</LINENO>
      <CNO>csc226</CNO>
      <A>90</A>
      <B>80</B>
      <C>65</C>
      <D>50</D>
    </Course>
    <Course>
      <TERM>F96</TERM>
      <LINENO>1032</LINENO>
      <CNO>csc227</CNO>
      <A>90</A>
      <B>80</B>
      <C>65</C>
      <D>50</D>
    </Course>
  </Student>
  <Student>
    <SID>3333</SID>
    <FNAME>Elad</FNAME>
    <LNAME>Yam</LNAME>
    <MINIT>G</MINIT>
  </Student>
</GradeBook>
```

Figure 2.4 (b)

Figure 2.4 Grade book database in XML format
(a) XML schema; (b) XML document

2.1.6 Other data models

- UML Data Models

  The Unified Modeling Language (UML) [25] is becoming widely used for both database and software modeling; it is Object Management Group's (OMG) most-used specification. UML classifies instances into objects and data values, like other ER notations, UML allows relationships to be modeled as attributes. UML uses class diagrams for analysis when UML is used for data modeling purpose.

  Strictly speaking, UML is a language designed for the software engineering modeling, rather than the data modeling.

- Network Data Models

The network model is conceived as a flexible way of representing objects and their relationships. It was originally introduced by Charles Bachman, and developed into a standard specification published in 1969 by the CODASYL Consortium. Unlike in hierarchical data models where each object can only have one parent and many children, network data models allow each object to have multiple parent and child objects. Therefore, network data models form a lattice structure, while hierarchical data models form a tree structure. One example is the DBTG-network data model introduced by the Database Task Group (DBTG) [26, 27]. The DBTG-network data model uses two constructs: the record and the link. The record represents the entity types, and the link represents the relationship between the records. Although network data models were implemented and used, they failed to become dominant and were eventually displaced by relational data models.

- Hierarchical Data Models

Hierarchical data models define hierarchically-arranged data. They are similar to network data models except that they impose a further restriction on the relationship types. The arcs in the data structure diagram of a hierarchical data model must form an ordered tree. The placement of the nodes (such as, parent node or children node, left or right node of each other) in the tree is significant. Hierarchical data models were widely used in the first mainframe database management systems, for example the IBM's Information Management System (IMS) [28, 29] and Adabas [30]. However, this kind of models creates repetition of data and cannot handle many-to-many relationships. They often cannot model many kinds of information existing in the real world.

- Binary Data Models

A binary data model refers to (or can be) any graph data model in which the nodes represent simple and single attributes and the arcs represent binary relationship types between two attributes. It served as a basis for many proposed data models [31, 32].

2.2 Query Languages

After the data is stored in a database, how to efficiently retrieve them is another important issue that needs to be considered in a particular application. Our research focuses on three most commonly used data queries languages: Structure Query Language (SQL), Object Query Language (OQL), and XQuery.

Database query languages are computer languages used to make queries to the databases. They are normally DBMS-dependent, and designed for a certain type of DBMS. For example, SQL is a well known query language for relational databases, OQL is a query language for object-oriented databases, and XQuery is the query language for XML data sources.

Figure 2.5 shows the history of query languages [33].

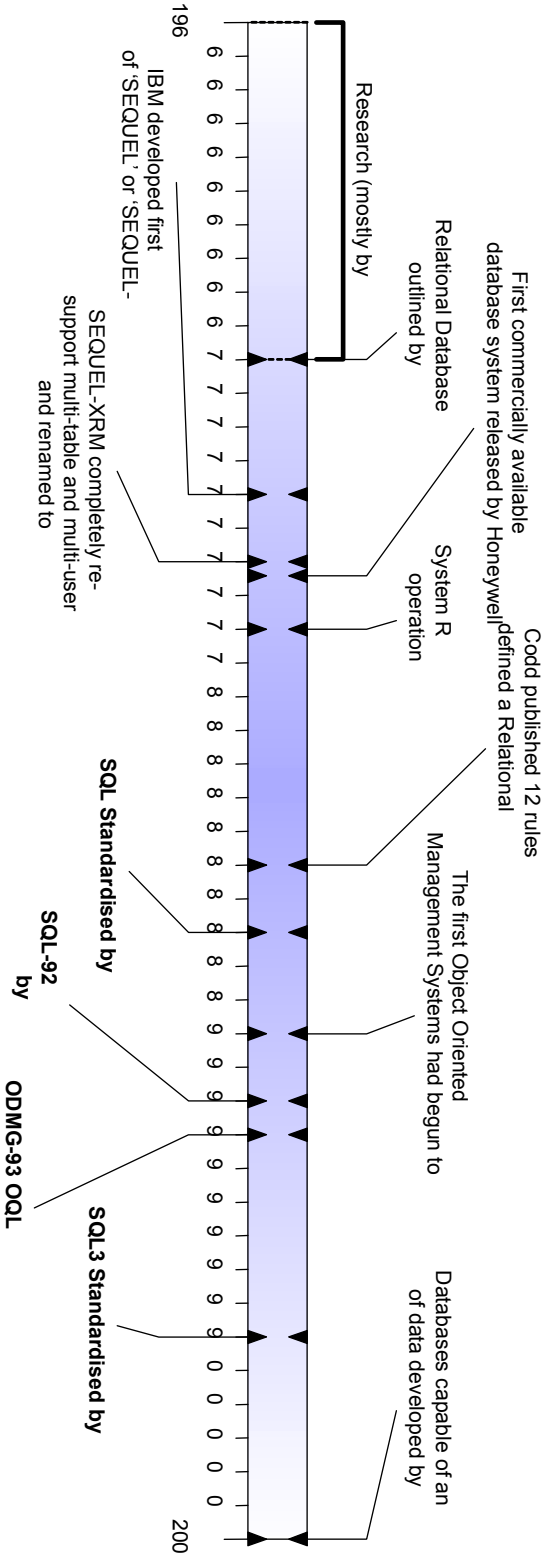

Figure 2.5 Query language history [33]

2.2.1 Traditional DBMS-supported query languages

Three major traditional DBMS-supported query languages are SQL, OQL, and XQuery.

Structured Query Language (SQL):

SQL (sometimes expanded as Structured Query Language) is a computer language used to create, retrieve, update and delete data from relational database management systems (DBMSs). SQL was adopted as a standard by ANSI (American National Standards Institute [34]) in 1986 and ISO (International Organization for Standardization [35]) in 1987. SQL can only be used to query data contained in a relational database. SQL is not an imperative language but a set-based, declarative programming language. It has gone through a number of revisions (see Table 2.1 [36]).

TABLE 2.1 SQL STANDARDS [36]

Year	Name	Alias	Comments
1986	SQL-86	SQL1	First published by ANSI. Ratified by6 ISO in 1987
1989	SQL-89		Minor revision
1992	SQL-92	SQL2	Major revision
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. (The last two are somewhat controversial and not yet widely supported.)
2003	SQL:2003		Introduced XML-related features, window functions, standardized sequences and column with auto-generated values (including identity-columns).
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it provides facilities that permit applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.

SQL statements are often divided into three categories:

DML (Data Manipulation Language): These SQL statements are used to retrieve and manipulate data. This category encompasses the most fundamental commands including DELETE, INSERT, SELECT, and UPDATE. DML SQL statements have only minor differences between SQL variations. DML SQL commands include the following:

- DELETE to remove rows.
- INSERT to add a row.
- SELECT to retrieve row.
- UPDATE to change data in specified columns.

DDL (Data Definition Language): These SQL statements define the structure of a database, including rows, columns; tables, indexes, and database specifics such as file locations. DDL SQL statements are more part of the DBMS and have large differences between the SQL variations. DML SQL commands include the following:

- CREATE to make a new database, table, index, or stored query.
- DROP to destroy an existing database, table, index, or view.
- DBCC (Database Console Commands) statements check the physical and logical consistency of a database.

DCL (Data Control Language): These SQL statements control the security and permissions of the objects or parts of the database(s). DCL SQL statements are also more part of the DBMS and have large differences between the SQL variations. DML SQL commands include the following:

- GRANT to allow specified users to perform specified tasks.

- DENY disallowing specified users from performing specified tasks.
- REVOKE to cancel previously granted or denied permissions.

Object Query Language (OQL):

OQL is the standard object-oriented SQL-like query language with special features dealing with complex objects, values and method for object-oriented database management systems modelled after SQL, which is specified by Object Data Management Group (ODMG) [37] as part of the ODMG2.0 standard. OQL can be used in two different ways, one is that it can be used as an embedded function in a programming language like C++ or Java to make programs simpler, as an embedded language, OQL allows querying denotable objects that are supported by the native language through expressions producing atoms, collections and literals; another is that it can be used interactively as an ad hoc query language allowing database queries from users. OQL uses ODL as its shema definition portion and OQL syntax has the SQL-like syntax. OQL concerns object-oriented notions like complex objects, object identity, path expression, polymorphism, and late binding.

XQuery:

XML is a versatile markup language; it can label the information content of diverse data sources including structured and semi-structured documents, relational databases, and object repositories. There are several query languages proposed for XML like XSL [39], XPath [40], XQL [41], XML-QL [42], Lorel [43], and YATL [44]. XQuery [38] is a new query language designed by W3C to be broadly applicable across many types of XML data sources.

XQuery aims to use the best features offered by previous query languages. That is, XQuery is derived from an XML query language called Quilt [45], which in turn borrowed features from several other languages including: XSL, XPath, XQL, XML-QL and traditional

database query languages. From XPath, XSL and XQL it took the path expression syntax suitable for hierarchical documents. From XML-QL it took the notion of binding variables and then using the bound variables to create new structures. From SQL it took the idea of a series of clauses based on keywords that provide a pattern for restructuring data (the SELECT-FROM-WHERE pattern in SQL). From OQL it took the notion of a functional language composed of several different kinds of expressions that can be nested with full generality. XQuery uses the document-oriented approach that forms a tree of nodes based on the DOM model.

2.2.2 Conceptual query languages

Traditional query languages are less than ideal for end user queries because they require user to know the underlying database structure rather than just conceptual schema. Compared to traditional query languages, conceptual query languages have some advantages: (1) conceptual query languages are usually DBMS-independent. (2) Conceptual query languages enable users to formulate queries directly on the conceptual schema without knowing much about underlying database.

Conceptual query languages are usually implemented using either a query generator to evaluate the query and then return the results to users or a query translator to translate the conceptual query into some DBMS-supported query statements. Many conceptual query languages have been proposed, some are based on ER model and some are not. ERQL [46] is a SQL-like conceptual query language based on ER or EER and it is texual. ERQL has the advantage over SQL in that relational details are hidden from users that make it very easier for user to write a query. CBQL [47] is a conceptual query language based on deductive models; it is a first-order like query language where users can specify the attributes and then

constrain the result set with logical constraints. ConQuer [48] is a conceptual query language built on ORM, which exposes semantic domains as conceptual object types, thus aloowing user to formulate the queries in terms of the relationships and operators such as “AND” and “NOT” .

2.2.3 Domain-Specific query languages

There are usually two approaches for any problem in the world, a generic approach or specific approach. Genetic approach usually provides a general solution for many problems in a certain area, but such a solution may be suboptimal. To a smaller set of problems, a specific approach will be good because it can provide a much better solution.

Domain-Specific query language is a query language works on a particular data domain, for some specific area like Web Search Engine [49], High Energy Physics [50], and Intrusion Analysis [51] Domain-Specific query language are very powerful to provide some advanced support for domain-specific data types and functions, but this kind of power usually is restricted to a particular data domain. Genomics Algebra [52] has been proposed based on the Genomics Ontology and is a new, integrating data model, language, and tool for processing and querying genomic information.

Because domain-specific query languages have more power when used just for a particular domain, they cannot be applied to other domains. So we believe that the general conceptual query language with the support for user-defined data types and functions will be the next generation of query languages.

CHAPTER 3 DOMAIN-SPECIFIC CONCEPTUAL QUERY SYSTEM

In this chapter, our Domain-Specific Conceptual Query System is introduced. In general, this system aims to provide a conceptual data model (DSC-DM) to capture more domain semantics and a user-oriented conceptual query language (DSC-QL) that uses only the abstract concepts and functions defined in DSC-DM. The methodology also enables end-users to define and add new domain-specific and application-specific functions into DSC-DM and DSC-QL at run time. It can be applied to any particular domain and work on any major type of database management system.

3.1 System Architecture

The architecture of the domain-specific conceptual query system is shown in Fig.3.1 [3]

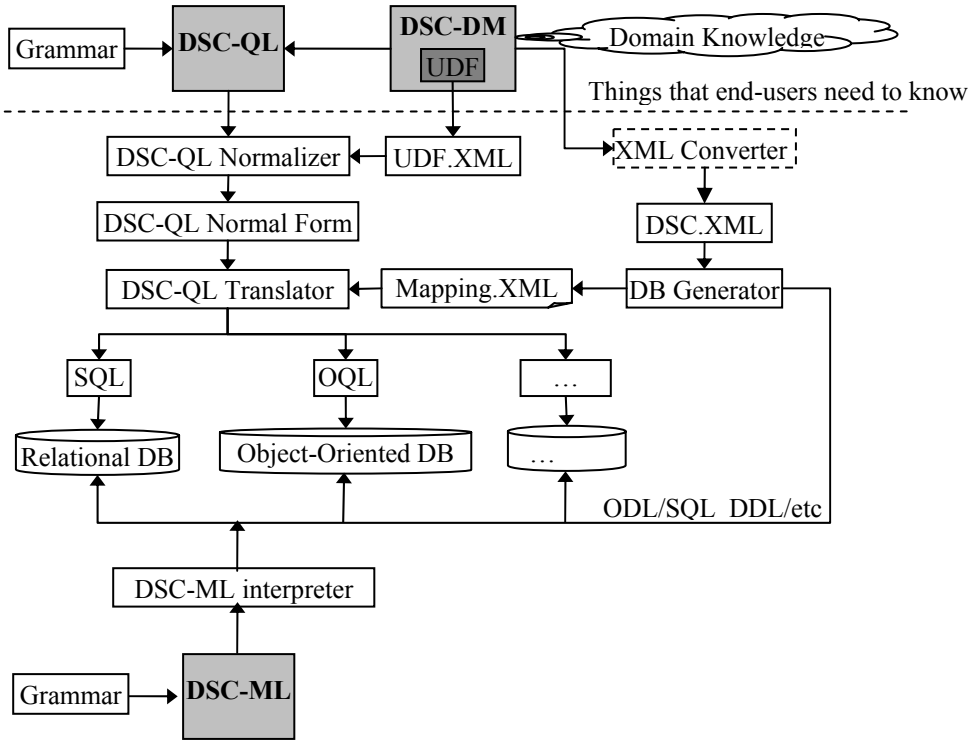

Figure 3.1 System Architecture of Query System

Our methodology includes DSC-DM, DSC-QL and DSC-ML; DSC-DM consists of a data structure diagram which is similar to Enhanced Entity-Relationship (EER [5]), definition of user-defined functions (UDFs), and two special tables for capturing additional domain semantics such as meta-information, annotation, and the relationship between entities and attributes. DSC-QL relies on DSC-DM and is a concept query language. It only uses the abstract concepts, relationships, and user-defined functions defined by domain experts and/or data model creators. So, the end-users can be easily write a query even they don't know the details of underlying database schemas. DSC-QL also integrates some features of OQL like dot-path expression and super-class; those features really make it much easier for end-users to write a query. DSC-QL queries can be translated into SQL, OQL or other query languages based on underlying database system. It is DBMS-independent and can be automatically updated when the underlying database schema evolves. DSC-ML is conceptual manipulation language. It is DBMS-independent and can be implemented as a DSC-ML interpreter which can be deployed to any database.

The right part of the architecture above is database creation. DSC-DM is converted into DSC.XML and UDF.XML firstly. DSC.XML stores all information in DSC-DM except definitions of UDF that are stored in UDF.XML. The conversion from DSC-DM to DSC.XML is done by XML Converter. DB Generator takes DSC.XML as input and generates SQL_DDL statements for relational DBMS or ODL for object-oriented (OO) DBMS. The database schema definition statements, in turn, are executed to create a database schema matching DSC-DM. Meanwhile, Mapping.XML is populated with the mapping information from DSC-DM to database schema. Like UDF.XML, DSC.XML can also be created manually so that the query system can be integrated into an existing application system.

The other part of the system is query evaluation. A DSC-QL query is firstly sent to DSC-QL Normalizer for parsing, validating, and normalizing. Any valid DSC-QL query can be transformed into a normal form [6], in which UDFs are replaced by corresponding DSC-QL sub-queries according to their definitions in UDF.XML. DSC-QL Translator will take as input the normalized DSC-QL query and output an equivalent SQL or OQL query according to the type of underlying database system. The mapping functions used in DSC-QL Translator are from Mapping.XML.For data manipulation(insert, updata, delete), the DSC-ML language is sent to DSC-ML Parser (using JFlex/JCUP) included in DSC-ML interpreter to check and verify the syntax and semantic, then the operations to the database will be implemented.

3.2 From DSC-DM to Database Schema

During the generation process of database schema, three XML files are created to store the corresponding information. The signature, semantics, and implementation of UDF are stored in UDF.XML file. Except UDF, other information in DSC-DM are converted into XML format and stored in DSC.XML. The mappings from concepts and relationships in DSC-DM to classes/tables in a database schema are stored in Mapping.XML.

3.2.1 User-Defined functions - UDF.XML

UDF includes domain-specific functions (DSFs) and application-specific functions (ASFs), which are defined by domain-experts and model creators. The signature, semantics, and implementation of UDF are stored in an XML file (i.e. UDF.xml) that is loaded by DSC-QL Normalizer at run time for query normalization. UDF.XML is part of DSC-DM; it is anually created by domain-experts and model creators. One UDF.XML example is given in section 3.2.4. Following is the XML schema of UDF.XML.

```
<xs:schema>
 <xs:element name="udfs">
```

```
  <xs:complexType>
   <xs:sequence>
      <xs:element name="udf" minOccurs="0"
               maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
          <xs:element ref="signature" minOccurs="1"
                                   maxOccurs="1"/>
          <xs:element ref="semantics" minOccurs="1"
                                   maxOccurs="1"/>
          <xs:element ref="implementation" minOccurs="1"
                                          maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
   </element>
  </xs:sequence>
 </xs:complexType>
  </xs:element>
  <xs:element name="semantics" type="xsd:string"
             minOccurs="1" maxOccurs="1"/>
 <xs:element name = "signature">
  <xs:complexType>
   <xs:sequence>
      <xs:element name="name" type="xs:string"
                minOccurs="1" maxOccurs="1"/>
      <xs:element name="parameter" type="xs:string"
                minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name = "implementation">
  <xs:complextType>
   <xs:sequence>
      <xs:element name="operator" type="xs:string"
                minOccurs="1" maxOccurs="1"/>
      <xs:element name="result" type="xs:string"
                minOccurs="1" maxOccurs="unbounded"/>
      <xs:element name="declaration" type="xs:string"
                minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="comparison" type="xs:string"
                minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
  </xs:schema>
```

UDF can be defined either by domain experts and/or data model creators at the time of system creation or by end-users at run time. After UDF.xml and the data structure for UDF in the system memory get updated, new functions are available for use.

3.2.2 Other information - DSC.XML

Except UDF, other information in DSC-DM are converted into XML format and stored in DSC.XML. This file is typically created by XML Converter and sent to DB Generator to generate the database schema definition statements (e.g. SQL CREATE statements or ODL statements). Following are the XML schemas of property, entity and relationship elements in DSC.XML and one example is given in section 3.2.4.

```
<xs:element name = "property">
  <xs:complexType><xs:sequence>
      <xs:element name="name" type="xs:string"
                minOccurs="1" maxOccurs="1"/>
      <xs:element name="primary_key" type="xs:boolean"
                minOccurs="1" maxOccurs="1"/>
      <xs:element name="structure" type="xs:string"
                minOccurs="1" maxOccurs="1"/>
      <xs:element name="data_type" type="xs:string"
                minOccurs="1" maxOccurs="1"/>
   </xs:sequence></xs:complexType>
 </xs:element>
 <xs:element name = "entity">
  <xs:complexType><xs:sequence>
     <xs:element name="name" type="xs:string"
                minOccurs="1" maxOccurs="1"/>
     <xs:element ref="property" type="xs:string"
                minOccurs="1" maxOccurs="unbounded"/>
     <xs:element name= "superClass" type= "xs:string"
                minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence></xs:complexType>
  </xs:element>
<xs:element name = "relationship">
 <xs:complexType><xs:sequence>
     <xs:element name="name" type="xs:string"
                minOccurs="1" maxOccurs="1"/>
     <xs:element ref="property" type="xs:string"
                minOccurs="1" maxOccurs="unbounded"/>
     <xs:element ref="participants" minOccurs="2"
                maxOccurs="n"/>
  </xs:sequence></xs:complexType>
</xs:element>
```

3.2.3 Mapping from DSC-DM to database schema - Mapping.XML

The mappings from concepts and relationships in DSC-DM to classes/tables in a database schema are stored in Mapping.XML. It is created with DSC.XML together by DB Generator

or model creators. The purpose of this file is to provide the mapping functions to DSC-QL Translator to translate the normalized DSC-QL expression into an equivalent SQL or OQL query.

A normalized DSC-DM schema *S* (defined in Chapter 4) is used to define the mappings. *S* is represented by a five-tuple *S* = (C, R, P, T, L), where

- C is a set of non-super class entity types in DSC-DM
- R is a set of non-inheritance relationships, where the relationship involving a super-class is distributed to all its sub-classes at the leaf level.
- $P = P_S \cup P_P$, where P_S is a set of set-attributes and P_P is a set of primitive attribute with a single value including meta-attributes. For each $p \in P$, p must belong to an owner $o \in C \cup R$ (represented as *owner*(p) = o). The attributes of a super class are copied to all its subclasses at the leaf level. The component attributes of a composite attributes are bound to the owner entity/relationship directly.
- T includes annotation-table, meta-attribute-table, and UDFs.
- L is set of names of entity types, relationship types, attributes, and predefined names for set-attributes.

The mapping function Δ: *S* = (C, R, P, T, L) → RDBS (relational database schema)

- $(\forall e \in C \cup R \cup P_S)$ (Δ(e) = relation r in RDBS, where *name(r) = name(e)*)
- $(\forall p \in P_P)$ (if p is not meta-attribute, then Δ(p) = a column c of relation r in RDS, where *name(c) = name(p)* and *name(r) = name(owner(p))*. The data type of p is specified in *Meta-attribute-table*).
- (Δ(*Annotation-table*) = relation *Annotation* in RDBS)
- (Δ(*Meta-attribute-table*) = relation *Meta-attribute* in RDBS)

- (Δ(constraint in *Meta-attribute-table*) = constraint(s) in RDBS)

The mapping function Δ: *S* = (C, R, P, T, L) → OODBS (object-oriented database schema)

- ($\forall e \in$ C $\cup$ R) (Δ(e) = class c in OODBS, where *name(c)* = *name(e)*)
- ($\forall p \in$ P) (if p is not meta-attribute, then Δ(p) = an attribute a of class c in OODBS, where *name(a)* = *name(p)* and *name(c)* = *name(owner(p))*. The data type of p is specified in *Meta-attribute-table.*)
- (Δ(*Annotation-table*) = class *Annotation* in OODBS)
- (Δ(Meta-attribute-table) = class Meta-attribute in OODBS)

3.2.4 Examples

Figure3.2 is a part of DSC-DM for neuroscience domain. We use this as an example to illustrate how our query system generates a database schema from DSC-DM. The details of the data model and query language created for neuroscience domain are described in [3, 7].

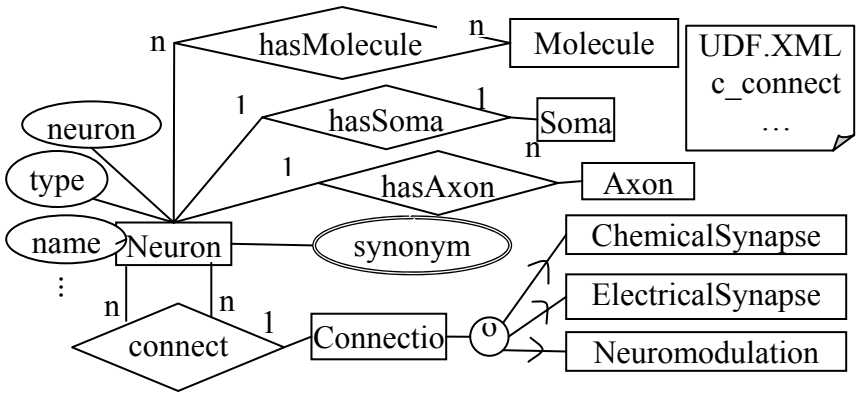

Figure 3.2 a part of DSC-DM for neuroscience domain

UDF example:

Signature:

c_connect(Neuron n, String s)

Semantics:

exist a chemical synapse from input neuron to another neuron whose name is the input string

Implementation:

```
EXISTS(n2.neuronid)[Neuron n2,connect c,
          ChemicalSynapse cs,
          c.pre_neuronid= n.neuronid,
          c.post_neuronid= n2.neuronid,
          n2.name=s, c.conid=cs.conid]
```

Part of UDF.XML example is shown below:

```
<udfs>
<udf>
 <signature>
  <name>c_connect</name>
  <parameter>Neuron n</parameter>
  <parameter>String s</parameter>
 </signature>
   <semantics>exist a chemical synapse from input neuron to
      another neuron whose name is the input string
 </semantics>
 <implementation>
  <operator>EXISTS</operator>
   <result>Neuron.neuronid</result>
   <declaration>Neuron n2</declaration>
   <declaration>Connect c</declaration>
   <declaration>ChemicalSynapse cs</declaration>
     <comparison>c.pre_neuronid=n.neuronid</comparison>
     <comparison>c.post_neuronid=n2.neuronid</comparison>
   <comparison>c.conid=cs.conid</comparison>
   <comparison>n2.name = String</comparison>
 </implementation>
</udf>
  ...
</udfs>
```

Part of DSC.XML example is shown below:

```
<entity>
 <name>Neuron</name>
 <property>
  <name>neuronid</name>
  <primary_key>True</primary_key>
  <structure>Simple</structure>
  <data_type>String</data_type>
 </property>
 <property>
```

```
  <name>synonym</name>
  <primary_key>False</primary_key>
  <structure>Set</structure>
  <data_type>String</data_type>
 </property>
 ...
 <superClass>none</superClass>
</entity>
...
<relationship>
 <name>hasMolecule</name>
 <participants>
  <entityName cardinality="n">Neuron</entityName>
  <entityName cardinality="n">Molecule</entityName>
 </participants>
</relationship>
<relationship>
 <name>connect</name>
 <participants>
  <entityName cardinality="n">Neuron</entityName>
  <entityName cardinality="n">Neuron</entityName>
  <entityName cardinality="1">Connection</entityName>
 </participants>
</relationship>
...
```

Corresponding ODL and SQL statements for Neuron table/class are shown below:

```
class Neuron {
  string neuronid;
  string name;
  set<string> synonym;
    ...;
  hasSoma *itsSoma inverse hasSoma::theNeuron;
  hasAxon *itsAxon inverse hasAxon::theNeuron;
  set<hasMolecule*> itsMolecule inverse
       hasMolecule::theNeuron;
  set<connect *> input inverse connect:: preNeuron;
  set<connect *> output inverse connect:: postNeuron;
};

 create table Neuron (
  neuronid varchar2(20) primary key;
  name varchar2(30);
  ...;
   };
```

It should be reasonably easy to figure out the DB-schema-generation algorithm from examples above (i.e. the mapping functions in Mapping.XML). The things that might not be clear from the examples are mappings of relationships, set-attributes, and inheritances. Each relationship in DSC-DM is mapped to a class/table in the schema that has references/foreign-keys to every participant. In relational database schema, the set-attribute is converted into a table, and inheritance is replaced by the distribution of all attributes and relationships of a super-class to all its subclasses at the leaf level. In the object-oriented database schema, the set-attribute and inheritance are handled in the normal fashion.

3.3 From DSC-QL to SQL/OQL

DSC-QL is designed as a high level conceptual language. It is DBMS-independent and can be translated into SQL or OQL. Firstly, the query is sent to DSC-QL Parser (using JFlex/JCUP) included in DSC-QL Normalizer to check and verify the syntax and semantic. Then it is normalized into DSC-QL normal form, in which all UDFs are replaced by predefined sub-queries in DSF.XML. DSC-QL Translator will translate normalized DSC-QL expressions into SQL or OQL according to the mapping functions in Mapping.XML.

3.3.1 DSC-QL syntax

The syntax of DSC-QL is defined in [3], which has a very simple structure like *(result-list)[[query-criteria] OR [query-criteria]…]*. The result-list can have declarations and attribute references. The query-criteria is composed of a list of condition-terms such as declaration, comparison-term, UDF, and sub-query. All terms in DSC-QL are the abstract concepts or functions defined in DSC-DM. We give three examples to illustrate the general syntax of DSC-QL.

Following is the grammar of DSC-QL in Backus-Naur Form (BNF) notation:

```
//general structure of DSC-QL
<compound-dscql> ::= <dscql> <set-operator> <dscql>
<dscql> ::= "(" <result-list>  ")" [ <query-criteria> ]
<query-criteria> ::= "["<condition-list>"]"|
                     "["<condition-list>"]" OR <query-criteria>
//syntax of result-list
<result-list> ::=  [DISTINCT] <result-term> {"," <result-term> }
<result-term> ::= <declaration> | <attr-ref> | <meta-attr-ref>
//syntax of condition-list
<condition-list> ::= <condition-term> { "," <condition-term> }
<condition-term> ::= <declaration> | [NOT] <comparison>
                     | [NOT] <udf> | [NOT] <sub-query>
//syntax of dot-path expression
<path-exp> ::= <data-sort> | <identifier> "." <path-exp-trail>
<path-exp-trail> ::= <data-sort> {"." <data-sort> }
//syntax of attribute reference and meta-attribute reference
<attr-ref> ::= <path-exp> "." <attribute>
<meta-attr-ref> ::= <attr-ref> "." <meta-attribute>
//syntax of declaration
<declaration> ::= <path-exp> " " <identifier>
//syntax of <comparison>
<comparison> ::= <operand> <logic-operator> <operand>
<operand> ::=<constant>|<attr-ref>|<operand><arithmetic-operator><operand>
//syntax of <sub-query>
<sub-query> ::= [<attr-ref>] <operator> <sub-query>
//syntax of <udf>
<udf> ::= <udf-name> "(" <parameters> ")"
<parameters> ::=  /*empty*/ | <parameter> {"," <parameter>}
<parameter> ::= <declaration> | <constant> | <attr-ref> | <identifier>
//syntax of operators and other terms
<set-operator> ::= UNION | INTERSECT | MINUS
<logic-operator> ::= > | < | = | ≥| ≤| <> | LIKE
<arithmetic-operator> ::= + | - | * | /
<operator> ::= EXISTS | IN
<attribute> ::=  <identifier> /*predefined in DSC-DM*/
<meta-attribute> ::= <identifier> /*predefined in DSC-DM*/
<data-sort> ::= <identifier> /*predefined in DSC-DM*/
<udf_name> ::= <identifier> /*predefined in DSC-DM*/
<identifier> ::= [A-Za-z_]{A-Za-z_0-9}
<constant> ::= 0 | [1-9]{0-9} | [A-Za-z_]{A-Za-z_0-9}
```

The following are some DSC-QL query examples:

Example 1: Find all neurons have molecule '5HT' and connect to another neuron, whose

connid is ‘1’.

(Neuron n)[n.hasMolecule.Molecule.name=‘5HT’, n.connect.Connection.connid=’1’]

Example 2: Find the names of neurons whose soma’s coloration is ‘Orange Circle’ or ’Yellow Circle’.

(Neuron n, n.name)[[n.hasSoma.Soma.coloration = ‘Orange Circle’] OR [n.hasSoma.Soma.coloration = ‘Yellow Circle’]]

Example 3: Find neurons which have molecule ‘5HT’ and connect to neuron ‘R3-13’ through a chemical synapse.

(Neuron n)[EXISTS (m)[n.hasMolecule.Molecule m, m.name = ‘5HT’], c_connect(n, ‘R3-13’)]

Example 1-3 also shows the usage of super-class (*Connection*), dot-path expression (*n.hasSoma.Soma.coloration)*, user-defined functions (*c_connect (n, ‘R3-13’)*), and sub-query (*EXISTS (m)[n.hasMolecule.Molecule m, m.name=‘5HT’]*).

3.3.2 Normal form of DSC-QL

After syntax and semantics checking, DSC-QL query will be normalized to the normal form. The definition of normal form and normalization algorithm are introduced in [6]. The goal of normalization is to remove the mismatches between DSC-QL and DBMS-supported query languages, especially SQL such as user-defined functions, inheritance, super-class, and set and composite attributes. As a result, the translation function is simplified significantly and its dependence on the flexible and extensible DSC-QL syntax is minimized. That is, as long as DSC-QL is compatible to the normal form, DSC-QL translation function does not have to be reprogrammed when DSC-QL is upgraded to a new version. Although the normalization procedure adds some burden to the performance and might need to be modified

when DSC-QL is upgraded, the total cost should be considerably lower than a complicated direct-translation process.

3.3.3 Translation of DSC-QL

DSC-QL is a high level conceptual language and can be translated into SQL and OQL. We can divide the whole translation process into three steps, firstly, the query is sent to DSC-QL Parser (using JFlex/JCUP) included in DSC-QL Normalizer to check and verify the syntax and semantic. Secondly, it is normalized into DSC-QL normal form, in which all UDFs are replaced by predefined sub-queries in DSF.XML. Lastly, DSC-QL Translator will translate normalized DSC-QL expressions into SQL or OQL according to the mapping functions in Mapping.XML. the translation algorithms from DSC-QL to SQL and DSC-QL to OQL are presented below:

Algorithm: DSC-QL to SQL [6]

Input: A DSC-QL query in normal form
Output: A SQL query statement

Method:

Initialize a SQL query statement with empty SELECT, FROM, and WHERE clauses
Initialize *tid_i* and increase *i* whenever it is used
Scan the DSC-QL query from left to right

For each result-term enclosed in ( ) (i.e. Q_R)
Case 1: attribute reference e.g. *v.p*
Append "*v.p*" into SELECT clause
Case 2: meta-attribute reference e.g. $v.p.p_m$, where *v* is of type E_1
Append "*Meta-attribute tid_i*" to FROM clause
Append "$tid_i.p_m$" to SELECT clause
Append "*tid_i.class* = E_1 *and tid_i.attribute = 'p'*" to WHERE clause

For each declaration enclosed in [ ] (i.e. Q_D)
Case 1: simple declaration
Append the declaration into FROM clause
Case 2: path declaration e.g. $c.E_2.E_3\ v$, where *c* is of type E_1
Append following string into FROM clause
"*(select tid_1.pk as pk, tid_3.* from*E_1 *tid_1,* E_2 *tid_2,* E_3 *tid_3*

where tid_1.pk = tid_2.fkToE$_1$ AND tid_2.pk = tid_3.fkTo E$_2$) as v"
Append "*c.pk = v.pk*" into WHERE clause //JOIN condition

For each comparison in condition list (i.e. Q_C)
Append it into WHERE clause

For each sub-query in condition list (i.e. Q_Q)
Recursively call normal translation algorithm to transform sub-query into a SQL query
Append sub-query's SQL translation into WHERE clause

The translation algorithm above scans the input normal DSC-QL query from left to right, and outputs an equivalent SQL query. In general the *SELECT* clause of SQL is resulted from Q_R, *FROM* clause is resulted from Q_D, and the *WHERE* clause is from Q_C and Q_Q. The dot-path expression will be translated into a sub-query with a series of JOINs in the *FROM* clause and a JOIN condition in *WHERE* clause.

Algorithm: DSC-QL → OQL

Input: A DSC-QL query in normal form
Output: A OQL query statement

Method:

Initialize a OQL query with empty SELECT, FROM, and WHERE clauses
Initialize cid_i and increase i whenever it is used
Scan the DSC-QL query from left to right

For each result-term enclosed in ( )
Case 1: attribute reference e.g. v.p
Append "v.p" into SELECT clause
Case 2: meta-attribute reference e.g. v.p.p$_m$, where v is of type E$_1$
Append "Meta-attribute cid_i" to FROM clause
Append "cid_i.p$_m$" to SELECT clause
Append "cid_i.class = E$_1$ and cid_i.attribute = 'p'" to WHERE clause

For each declaration enclosed in [ ]
Case 1: simple declaration
Append the declaration into FROM clause
Case 2: path declaration e.g. c.E$_2$.E$_3$ v, where c is of type E$_1$, and assume c.E$_2$ returns a set
Append "cid_i in c.E$_2$, cid_i.E$_3$ v" into FROM clause

For each comparison in condition list

Append it into WHERE clause

For each sub-query in condition list
Recursively transform sub-query into a OQL query
Append sub-query's OQL translation into WHERE clause

The translation algorithm above scans the input normal DSC-QL query from left to right, and outputs an equivalent OQL query. In general, the *SELECT* clause of OQL is resulted from result-terms, *FROM* clause is resulted from declarations, and the *WHERE* clause is from comparisons and sub-queries. During the translation, terms in DSC-QL are replaced according to mapping functions in Mapping.XML. Therefore, domain-experts and data model creators can define meaningful terminologies for end-users at the conceptual level and use different ones at database leve.

3.3.4 DSC-QL to SQL/OQL example

DSC-QL is normalized to DSC-QL normal form by DSC-QL Normalizer and then the normal form can be translated to SQL or OQL by DSC-QL translator. The following is an example to show how the DSC-QL is translated to SQL or OQL.

Example:

Consider the query: *find all neurons having molecule "5HT" and a chemical synapse connection with neuron "R3-13".*

This query can be expressed as follows:

```
(Neuron n)[c_connect(n, 'R3-13'), n.hasMolecule.Molecule.name=
'5HT']
```

The query's normal form after normalization:

```
(n.name)[Neuron n, n.hasMolecule.Molecule sid_0,
        sid_0.name='5HT',
        EXISTS(sid_1.neuronid)[Neuron sid_1,connect sid_2,
              ChemicalSynapse sid_3,
              sid_2.pre_neuronid=sid_1.neuronid,
              sid_2.post_neuronid=n.neuronid,
              sid_1.name='R3-13', sid_2.conid=sid_3.conid]]
```

SQL query is presented below after DSC-QL to SQL translation:

```
SELECT n.name
FROM Neuron n,(SELECT tid_0.neuronid AS pk, tid_2.*
               FROM Neuron tid_0, hasMolecule tid_1,
                    Molecule tid_2
               WHERE tid_0.neuronid=tid_1.neuronid AND
                     tid_1.moleculeid=tid_2.moleculeid)
               AS sid_0
WHERE n.neuronid = sid_0.pk AND
      sid_0.name='5HT' AND
      EXISTS
       (SELECT sid_1.neuronid
        FROM Neuron sid_1, connect sid_2,
             ChemicalSynapse sid_3
        WHERE sid_2.conid=sid_3.conid AND

              sid_2.pre_neuronid=sid_1.neuronid AND
              sysid_1.name='R3-13'AND
              sid_2.post_neuronid=n.neuronid)
```

OQL query is presented below after DSC-QL to OQL translation:

```
SELECT n.name
FROM Neuron n, cid_0 IN n.itsMolecule,
     cid_0.theMolecule sid_0
WHERE sysid_0.name='5HT' AND
      EXISTS
         (SELECT sid_1.neuronid
          FROM Neuron sid_1, connect sid_2,
               ChemicalSynapse sid_3
          WHERE sid_2.preNeuron=sid_1 AND
                sid_2.postNeuron=n AND
               (ChemicalSynapse)sid_2.theConnection=sid_3
                AND sid_1.name='R3-13')
```

3.4 DSC-ML

DSC-ML is designed as a high level conceptual manipulation language. It is DBMS-independent and can be implemented as a DSC-MLinterpreter which can be deployed to any database. Because ODMG standard does not provide an OML specification so we don't translate the DSC-ML to SQL/OQL. Firstly, the DSC-ML is sent to DSC-ML Parser (using

JFlex/JCUP) included in DSC-ML interpreter to check and verify the syntax and semantic. Then it is implemented on related database.

3.4.1 DSC-ML syntax

Following is the grammar of DSC-ML in Backus-Naur Form (BNF) notation:

<dscml>::=”INSERT” <identifier> “(“ <property-values> “)”

| “DELETE” <identifier>

| “UPDATE” <identifier> “(“ <property-values> “)”

<property-values>::=<property-value> | <property-value> “,” <peoperty-value>

<property-value>::= <identifier> “=” <identifier> | <identifier> “=” <constant>

| *<identifier> “=” “[“ <values> “]”*
<values>::=<value> | <value> “,” <value>
<value>::= | <identifier>| <constant>

3.4.2 DSC-ML examples

The following is some examples of DSC-ML (use enrollment system shown in Figure 3.3 next page):

Consider the INSERT operation*:* *insert a student with sid 111, one course with cno 222 and this student enroll this course all those information to database*.

Insert operations can be expressed as follows:

```
DSC-Int> X:=INSERT student
               (sid='111', name='tom', phone=['404-231-
                5678','678-211-9000'],advisor='Dr.Li').
DSC-Int> Y:=INSERT course(cno='222',ctitle='database').
DSC-Int> INSERT enroll(student=X,course=Y,grade=90).
```

Consider the DELETE operation: *delete all students whose advisor is Dr.Li from the database.*

Delete operations can be expressed as follows:

```
DSC-Int> S:= (student s)[advisor='Dr.Li'].
```

```
DSC-Int> DELETE S.
```

Consider the UPDATE operation: *Change student's advisor to Dr.Zhou and this student sid is 111.*

Update operations can be expressed as follows:

```
DSC-Int> T:=(student s)[sid='111'].
DSC-Int> UPDATE T (advisor='Dr.Zhou').
```

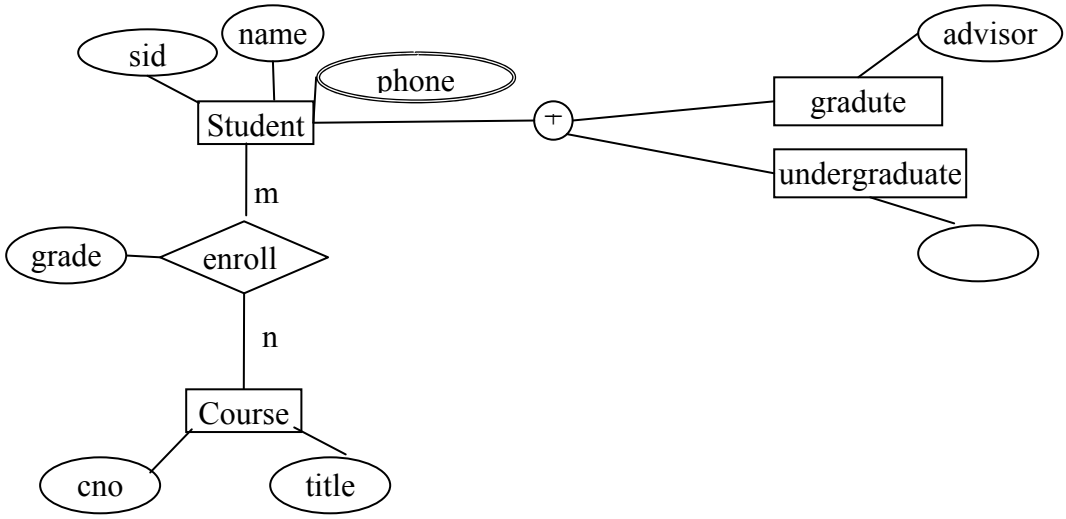

Figure 3.3 Data structure diagram of DSC-DM for an enrollment system

3.5 Source Code

The query system has been partially implemented and tested on a relational DBMS, Oracle 10g and an object-oriented DBMS, EyeDB 2.7.9. Current version is a standalone Java application with a command line input query interface. A web-based query interface is under development now and will be released very soon. The source code can be downloaded at http://tinman.cs.gsu.edu/~xshen2/DSC_QL.zip.

CHAPTER 4 CONCLUSION AND FUTURE WORK

This research presents the architecture of a domain-specific conceptual data model and query system and its implementation details. Domain-specific conceptual data model (DSC-DM) consists of three components, a data diagram, definitions of user-defined functions (UDF), and two tables for meta-information and annotation. The data diagram in DSC-DM is similar to Enhanced Entity-Relationship (EER) data model because EER data model is very simple and easy-to-understand for naïve users. The two tables for meta-information and annotation enable DSC-DM to capture more domain semantics than EER. User-defined functions (UDFs) include domain-specific functions (DSF) and application-specific functions (ASF) which are defined by domain experts and/or data model creators during the system creation. The signature, semantics, and implementation of UDF are stored in an XML file (i.e. UDF.xml) that is loaded by DSC-QL Normalizer at run time for query normalization. Therefore, UDF can be easily updated at any time without changing rest parts of the system.

DSC-QL is another important component of domain-specific conceptual query system. It is based on DSC-DM; DSC-QL only uses the abstract concepts, relationships, and functions in DSC-DM and is designed to be flexible, extensible, and readily usable to end-users. Comparing to SQL or OQL queries, DSC-QL queries are very simple and eas to write based on its specific features. It also integrates dot-path expression and super-class those object-oriented concepts. DSC-QL is DBMS-independent so that end-users don't need to know details of underlying database schemas when they write a query. It can be translated into SQL or OQL. Firstly, the query is sent to DSC-QL Parser (using JFlex/JCUP) included in DSC-QL Normalizer to check and verify the syntax and semantic. Then it is normalized into DSC-QL normal form, in which all UDFs are replaced by predefined sub-queries in DSF.XML. DSC-

QL Translator will translate normalized DSC-QL expressions into SQL or OQL according to the mapping functions in Mapping.XML.

Compared to traditional query language like SQL, OQL etc. traditional query system are too complicated and difficult for end-users because they all need end-users know the database knowledge and details. Except traditional query language, form-based query system also requires end-users to have a lot of knowledge about underlying database. Even some form-based interfaces have no requirement to end-users for having knowledge of database details, they still have a lot of limitation like limited expressive power, re-programming required when the underlying database schema evolves. Although many conceptual query languages have been proposed so far [1, 8-11], most of them do not have implementation reports and those ones being implemented are often designed as a part of some software or combined with fixed query interface. As a consequence, they cannot exist independently and be applied to other existing applications.

. ConQuer [12] is another conceptual query language based on Object-Role Modeling. It has been commercially released with specific query interface. Compared to ConQuer and other conceptual query languages, our query system has following unique features:

- It supports domain-specific and application-specific functions that are defined by domain-experts and data model creators. The query system can dynamically load these user-defined functions without reprogramming and recompiling. As a result, user-defined functions can be easily upgraded at any time.
- It adopts the normal form to solve the mismatches between the high level conceptual query language DSC-QL and DBMS-supported query languages like SQL and OQL. It also minimizes the dependence of translation function on the

flexible and extensible syntax of DSC-QL. Thus, it is easier to extend or upgrade the query language in the future.

- The underlying data model (DSC-DM) can capture more domain semantics such as meta-attributes and annotation relationship between a reference entity and an attribute.
- The conceptual query system is designed to be general enough to be applied to any domain and any type of DBMS, and even existing applications.
- DSC-QL has very simple structure and is easy to write for end-users. It also dynamically supports new user-defined functions without reprogramming the application system.

The future work of this research is as following:

- The translation from DSC-QL to XQuery will be implemented in order to support the use of our query system on a native XML database system.
- The query interface and the result report interface need to be improved as well. Develop a graphical user interface (GUI) or web-based interface to replace current command-line interface in prototypes.
- Evaluate the usability of DSC-QL
- Simplify the integration process of our query system to existing database application system.
- Design a graphical query language based on current text-based DSC-QL.
- Add DSC-ML into DSC-QL so that it can be used as DML as well.

BIBLIOGRAPHY

[1] Lawey, M., Topor, R.: A Query Language for EER Schemas. ADC'94 Proceedings of the 5th Australian Database Conference, Global Publications Service, (1994) 292-304.

[2] Owei, V.: Development of a Conceptual Query Language: Adopting the User-Centered Methodology. The Computer Journal, 46(6), (2003) 602-624.

[3] Hao T., Sunderraman R., Hong Y.: A Domain-Specific Conceptual Data Modeling and Querying Methodology. 1st International Conference on Information Systems, Technology and Management, March 2007, New Delhi, India

[4] EyeDB: website – www.eyedb.org

[5] Elmasri, R. and Navathe, S.B.: Fundamentals of database systems, third Edition. Addison Wesley (2000).

[6] Tian, H., Sunderraman, R.: A Query Normal Form and Graph-Based Semantics for Conceptual to Relational Mappings, Unpublished manuscript

[7] Tian, H., Sunderraman, R., Calin-Jageman, R., Yang, H., Zhu Y., and Katz, P.S.: A Domain-Specific Query Language for Neuroscience Data. 11th International Workshop on Foundations of Models and Languages for Data and Objects: Query Languages and Query Processing, (as EDBT2006 Workshop), (2006) 613-624.

[8] Amaral, V., Helmer, S., Moerkotte, G.: A Visual Query Language for HEP Analysis. Nuclear Science Symposium Conference Record, IEEE, 2, (2003) 829-833.

[9] Aslam, J., Bratus, S., Kotz, D., Peterson, R., Tofel, B., Rus, D.: The Kerf Toolkit for Intrusion Analysis. IEEE Security and Privacy, 2(6), (2004) 42-52.

[10] Collberg, C.,: A Fuzzy Visual Query Language for a Domain-Specific Web Search Engine. Proceedings of Second International Conference on Diagrammatic Representation and Inference, (2002) 176-190.

[11] Hammer, J., Schneider, M.: The GenAlg Project: Developing a new integrating data model, language, and tool for managing and querying genomic information. SIGMOD Record, ACM, 33(2) (2004).

[12] Bloesch, A.C., Halpin, T.A.: ConQuer: a conceptual query language. Proc. ER'96: 15th Int. Conf. on conceptual modeling, Springer LNCS, 1157, (1996) 121-33.

[13] Dionysios C. Tsichritzis and Frederick H. Lochovsky, "Data Models," Prentice-Hall, 1982.

[14] The Data Management Association: http://www.dama.org

[15] PC Magazine: http://www.pcmag.com

[16] Langefors, B., “Information systems theory,” Inf. Syst. 2, pp. 207-219, 1977.

[17] E. F. Codd, “A relational model of data for large shared data banks,” Commun. ACM 13, 1970, pp, 377-387.

[18] Rajshekhar Sunderraman, Oracle 9i Programming: A Primer, Addison-Wesley, 2004.

[19] P. P. Chen, “The entity-relationship model: Toward a unified view of data,” ACM Trans. Database Syst. 1, 1976, pp. 9-36.

[20] T. Teorey, D. Yang, and J. Fry, “A logical design methodology for relational databases using the extended entity-relationship model,” ACM Computing Surveys, 18:2, June 1986.

[21] M. Gogolla, and U. Hohenstein, “Towards a semantic view of an extended entity-relationship model,” TODS, 16:3, September 1991.

[22] R. Elmasri, J. Weeldreyer, and A. Hevner, “The category concept: an extension to the entity-relationship model,” International Journal on Data and Knowledge Engineering, 1:1, May 1985.

[23] A. Badia, “Entity-relationship modeling revisited,” SIGMOD Record, Vol. 33, No. 1, March 2004.

[24] Object Role Modeling (ORM): http://www.orm.net/

[25] Unified Modeling Language (UML): http://www.uml.org/

[26] CODASYL Data Base Task Group Report. Conf. On Data System Language, ACM, New York, 1971.

[27] Taylor, R.W., and Frank, R. L., “CODASYL data-base management systems.” ACM Comput. Surv. 8, pp.67-103, 1976.

[28] Information Management System/Virtual Storage (IMS/VS) publications: General Information Manual, GH20-1260-3; System/Application Design Guide, SH20-9025-2; Application Programming Reference Manual, SH20-9026-2; System Programming Reference Manual, SH20-9027-2; Operator’s Reference Manual, SH20-9028-1; Utilities Reference Manual, SH20-9030-2. IBM Corp., White Plains, NY.

[29] McGee, W.C., “The information management system IMS/VS,” IBM Syst. J. 16, pp.84-168, 1977.

[30] Software AG: http://www.softwareag.com

[31] Senko, M.E., “Information systems: records, relations, set, entities, and things,” Inf. Syst. 1, pp 3-13, 1975.

[32] Bracchi, G., Paolini, P., and Pelagatti, G., "Binary logical associations in data modeling," in Modelling in Data Base Management Systems (Nijssen, G. M., ed.), pp.125-148. North-Holland, Amsterdam, 1979.

[33] OQL vs. SQL report, http://www.soc.napier.ac.uk/module/op/resources/moduleid/CO42009

[34] American National Standards Institute (ANSI): http://www.ansi.org/

[35] International Organization for Standardization (ISO): http://www.iso.org/

[36] SQL history: http://en.wikipedia.org/wiki/Sql

[37] Object Data Management Group (ODMG) website: http://www.odmg.org/

[38] XQuery 1.0: An XML Query Language: http://www.w3.org/TR/xquery

[39] XSL Transformations (XSLT) Version 1.0: http://www.w3.org/TR/xslt

[40] XML Path Language (XPath) 2.0: http://www.w3.org/TR/xpath20

[41] Hiroshi Ishikawa, Kazumi Kubota, and Yasuhiko Kanemasa, "XQL: A Query Language for XML Data," http://www.w3.org/TandS/QL/QL98/pp/flab.txt

[42] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu, "XML-QL: A Query Language for XML," http://www.w3.org/TR/1998/NOTE-xml-ql-19980819

[43] Roy Goldman, Jason McHugh, and Jennifer Widom, "Lore: A Database Management System for XML, A SQL-like system for XML data," http://www.ddj.com/documents/s=886/ddj0004i/0004i.htm

[44] Sophie Cluet, and Jérôme Siméon, "YATL: a Functional and Declarative Language for XML," draft manuscript, 2000.

[45] Jonathan Robie, Don Chamberlin, and Daniela Florescu, "A Quilt: an XML Query Language for Heterogeneous Data Sources," Lecture Notes in Computer Science, vol. 1997, 2001.

[46] Lawley, M. and Topor, R., "A query language for EER schemas", Proceedings of the 5th Australian Database Conference, Global Publications Service, 1994, pp. 292-304.

[47] Staudt, M., ., Nissen, H.W., Jeusfeld, M.A. 1994, *Query by Class, Rule and Concept.* Applied Intelligence, Special Issue on Knowledge Base Management, vol. 4, no. 2, pp. 133-157.

[48] Bloesch, A.C. and Halpin, T.A., "ConQuer: a conceptual query language", 15th International Conference on Conceptual Modeling, Springer LNCS, No. 1157, 1996, pp.121-33.

[49] Collberg, C., "A fuzzy visual query language for a domain-specific web search engine", Proceedings of Second International Conference on Diagrammatic Representation and Inference, 2000, pp.176-190.

[50] Auddino, A., Amiel, E., and Bhargava, B., "Experiences with SUPER, a database visual environment," DEXA'91 Database and Expert System Applications, 1991, pp.172-178.

[51] Aslam, J., Bratus, S., Kotz, D., Peterson, R., Tofel, B., and Rus, D., "The kerf toolkit for intrusion analysis", IEEE Security & Privacy, Vol. 2, No. 6, 2004, pp. 42-52.

[52] Hammer, J. and Schneider, M., "Genomics algebra: a new, integrating data model, language, and tool for managing and querying genomic information", 1st Biennial Conference on Innovative Data Systems Research (CIDR), 2003, pp.176-187.

[53] BrainML.org: http://www.brainml.org

[54] CellML.org: http://www.cellml.org

[55] SBML: Systems biology markup language: http://www.sbml.org