Semantic Integration and Querying of Heterogeneous Data Sources Using a Hypergraph Data Model

Dimitri Theodoratos*

Department of Computer Science
College of Computing Sciences
New Jersey Institute of Technology
USA
dth@cis.njit.edu

Abstract

Information integration in the World Wide Web has evolved to a new framework where the information is represented and manipulated using a wide range of modeling languages. Current approaches to data integration use wrappers to convert the different modeling languages into a common data model. In this work we use a nested hypergraph based data model (called HDM) as a common data model for integrating different structured or semi-structured data. We present a hypergraph query language (HQL) that allows the integration of the wrapped data sources through the creation of views for mediators, and the querying of the wrapped data sources and the mediator views by the end users. We also show that HQL queries (views) can be constructed from other views and/or source schemas using a set of primitive transformations. Our integration architecture is flexible and allows some (or all) of the views in a mediator to be materialized.

1 Introduction

The advent of the World Wide Web, has evolved information integration from a traditional multidatabase architecture to a new framework where the information is available in diverse formats and structures. The integration process involves data sources that are heterogeneous and range from object-oriented and traditional relational database management systems to semistructured data repositories. In this context, exploiting the semantics of the data sources, greatly improves the integration process.

A typical approach for dealing with the heterogeneity of the data sources chooses one model as a common data model [23] and converts the modeling languages of the data sources into this model. An integration architecture is shown in Figure 1. A wrapper [22, 10] is

*Research supported by EPSRC under the project "Automatic Generation of Mediator Tools for Heterogeneous Database Integration – AutoMed"

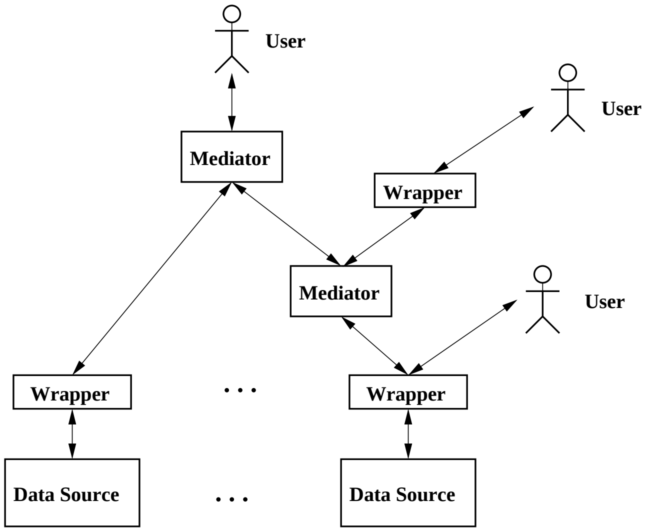


Figure 1: An integration architecture

software that translates between the source data models and local languages and the common data model and language. It transforms subqueries sent to the data sources and answers returned from the data sources. The mediators [29] are components that obtain information from wrapped data sources or other mediators. They provide information to other mediators above them or to the users of the system. The mediators export a mediator schema which is an integrated representation of data sources. The users can address their queries to the mediators or to the wrapped sources using the language of the common data model or using their own local language though a wrapper. A mediator can be seen as a view of the data found in one or more data sources [26]. Traditionally data are not stored in a mediator. The users query the mediator schema, and the views transform these queries between the mediator schema and the data sources [6]. More recently, the data warehousing approach for data source integration [27, 28] chooses to materialize the views in the mediators. In this case user queries are evaluated locally without accessing the data sources. A framework for supporting integrated views using a hybrid of these approaches is also possible [12, 11].

The common data model used in this paper for data source integration is based on the Hypergraph Data Model (HDM) [21]. An advantage of the HDM is that it is characterized by a conceptual simplicity. All conceptual objects are modeled using the notions of nodes and hyperedges. Another advantage of the HDM is that the definition of these constructs is separated by the definition of constraints on their values. In this sense, the HDM is a low level model that underpins other data models. Higher level models either structured (e.g. E/R, Relational, UML) or semi-structured (e.g. XML) can be defined in terms of the HDM [15, 16]. Thus, the HDM provides unifying semantics for higher level modeling constructs.

Contribution. The main contributions of the paper are the following.

- We present a variation of the HDM for data source integration. The new model distinguishes between two kinds of nodes that model entities and property values. Database schemas are hypergraphs. Instances are associated only with the hyperedges of a schema. Hyperedge instances may be either n-ary, $n \geq 2$, relations associating entities, or binary relations assigning property values to entities and to associations of entities. Relation attributes are hypergraph nodes. Node renaming functions are used to name the multiple occurrences of entity nodes in hyperedges. Node extensions are captured through the introduction of manifest constraints.

- We provide a hypergraph based query language (HQL). HQL is homogeneous with the HDM: queries in HQL are hypergraphs. In a query hypergarph, hyperedges are associated with relational algebra expressions. A query answer is a relation. Outer-join based semantics for queries are used to deal with missing data, a common situation when integrating heterogeneous distributed data sources.
- We offer an integration environment through the definition of HQL views (mediators) over multiple database schemas. The notion of schema edge instance is extended to view edges in order to allow the definition of views using other views. Our approach to data source integration is a hybrid one since views can be materialized or virtual.
- We show that an HQL view defined over an integration environment can be constructed from another view using view transformations. These transformations allow us also to deal with view evolution issues. We present a set of two primitive transformations based on which composite transformations can be defined. The primitive transformations are shown to be complete and reversible.

Outline. The next section presents related work. Section 3 introduces the HDM: hypergraphs, schemas, instances and databases. The Hypergraph query language HQL is presented in Section 4. Syntax, semantics, views and data source integration issues are discussed there. Section 5 deals with view transformations, primitive transformations, and their properties. Conclusions and directions for further work are provided in Section 6.

2 Related work

Although the World Wide Web has brought new dimensions to the problem of data source integration, this is an issue that has been addressed for many years and the bibliography is very extensive. There are several research projects and a considerable number of mediated query systems that assume a mediator environment based on a common data model. They adopt different modeling and querying languages [6]. The Tsimmis project [18] focuses on the integration of structured and unstructured data sources, techniques for the rapid prototyping of wrappers [10] and techniques for implementing mediators. Its common data model is a simple object-based information exchange model, while its query language uses a form of object-logic. The Garlic project implements a dynamic source capability solution [22]. The Information Manifold project [14] uses views to map from the source local schemas to the mediator schema. Its query language is a dialect of description logic. The evaluation of mediator queries is performed by rewriting queries using views [13]. The DISCO project [25] focuses on the problems encountered when the number of heterogeneous distributed data sources scales. Its mediator query language is a subset of ODMG corresponding to the relational algebra. Some data warehousing systems use the concept of mediation for the task of data integration. This is the case of the H2O project [30, 12]. However, even though data warehousing is also an approach for data source integration, the focus of the research in this area is on the materialized view maintenance problem [31], and the view selection problem [24].

A number of visual graphical query languages have been proposed in the past [5, 2, 9, 20, 3, 4, 19] some of them supporting also nesting [3, 4]. They aim at providing a uniform approach to representing, querying and, possibly, updating both schema and data. In contrast to these approaches, our goal here is to use a hypergraph data model as a low-level common data model for heterogeneous data sources and a hypergraph query language for integrating these data sources through the definition of views (mediators). [1] presents a graph model and shows how schemas expressed in different data models can be translated in terms of the graph model. It also provides a set of graphical primitives in terms of which query operations may be visually expressed. However, it does not provide a mechanism for defining and querying views.

3 The hyper-graph data model for heterogeneous data source integration

The Hypergraph Data Model [21] is based on the notion of a nested hypergraph. For the needs of this paper we present in this section a variation of the original Hypergraph Data Model. In the presentation below, we borrow terms from the E/R model [7] in order to provide some intuition to the reader.

We assume two infinite sets *Names* and *Values* whose elements are called names and values respectively.

Database schema. A *node* is an element of *Names*. With every node N a domain $dom(N) \subseteq Values$ is associated. We distinguish two kinds of nodes: *attribute nodes* and *entity nodes*. Entity nodes have a common domain. The domain of an attribute node is disjoint from the domain of the entity nodes. Intuitively, entity nodes represent collections of entities with common properties, and attribute nodes represent collections of property values.

A *hyper-edge* (henceforth simply *edge*) is a pair $\langle E, M \rangle$ where E is an element of *Names* and M is a finite multiset. E is the name of edge $\langle E, M \rangle$. Edges can be of two types: *relationship edges* and *attribute edges*. The multiset M of a relationship edge $\langle E, M \rangle$ contains entity nodes. The multiset M of an attribute edge $\langle E, M \rangle$ is a set of the form $\{C, A\}$, where C is an entity node or a relationship edge and A is an attribute node. An attribute edge between an entity and an attribute node or a relationship edge $\langle E, M \rangle$ is associated with a function that maps every occurrence of an entity node in M to a distinct name. This function is called *node renaming function*. Let f be the node renaming function of an edge $\langle E, M \rangle$. If $f(N)$ is N, name $f(N)$ is not mentioned explicitly. If an entity node N occurs more than once in M, $f(N)$ must be explicitly specified for at least one occurrence of N in M. Intuitively, relationship edges represent collections of associations among entities, while attribute edges represent collections of associations between entities and property values, or between associations of entities and property values. We call *construct* a node or an edge.

A *hypergraph* is a pair $\langle Nodes, Edges\rangle$, where $Nodes$ is a non-empty set of nodes, and $Edges$ is a non-empty set of edges such that:

(a) Every node in $Nodes$ has an incident edge in $Edges$.

(b) Set $Edges$ does not contain more than one attribute edge between an entity node (or a relationship edge) and an attribute node. $Edges$, however, can contain more than one relationship edge involving the same multiset of entity nodes.

(c) If $L \neq N$ is a name assigned by the node renaming function of a relationship edge to an entity node N in $Nodes$, there is a sequence of name pairs $(L_0, L_1), \ldots, (L_{n-1}, L_n)$, $n \geq 1$, such that $L_0 = N$, $L_n = L$ and L_k, L_{k+1}, $k \in [0, n-1]$, are assigned to N by the node renaming function a of relationship edge in $Edges$. One consequence of this is that if an entity node N in $Nodes$ has at least one incident relationship edge in $Edges$, N is assigned to at least one occurrence of N by the node renaming function of a relationship edge.

(d) If $L \neq N$ is a name assigned by the node renaming function of an attribute edge in $Edges$ to an entity node N in $Nodes$, L is also assigned to an occurrence of N by the node renaming function of a relationship edge in $Edges$. That is, new names for an entity node are created by the node renaming functions of relationship edges and are used by the node renaming functions of the attribute edges.

Such a hypergraph is nested because an attribute node can be linked not only to an entity node but also to a relationship edge (multiset of entity nodes). Note that a hypergraph is not necessarily a connected graph. Name E uniquely determines M in a hypergraph $\langle E, M\rangle$. Therefore, in the following, we identify an edge with its name.

Constraints are Boolean expressions involving constructs. Different kinds of constraints can be expressed in the generic hypergraph model as for instance cardinality constraints, ISA constraints, disjointness constraints, and completeness constraints [7]. Here we focus on a hypergraph query language and therefore we do not explicitly deal with constraints. We only mention *manifest constraints* and *uniqueness constraints*. Let E be the attribute edge between an entity node N and an attribute node A. A manifest constraint is denoted $manifest(E)$. Node A is called manifest attribute node of N. Intuitively, a manifest constraint states that every entity of N is associated in E with a property value from the domain of the manifest attribute node A (and thus every entity of N is manifested). The entities of node N are those entities of $dom(N)$ that are associated with an entity or a property value in an edge incident on N. Let now E be the attribute edge between an entity node or a relationship edge C and an attribute node A. A uniqueness constraint is denoted $unique(E)$. Intuitively, a uniqueness constraint states that an entity (if C is an entity node) or association of entities (if C is a relationship edge) is associated in E with at most one property value from the domain of attribute node A. Note that even though constraints are not involved in the definition of a query, their presence affects both the formulation of queries by the user, and the query optimization process.

A *database schema* is a triple $\langle Nodes, Edges, Cons\rangle$, where $\langle Nodes, Edges\rangle$ is a hypergraph H and $Cons$ is a set of constraints over constructs in H. In particular, for every entity node N in $Nodes$, $Cons$ contains a manifest constraint involving an attribute edge incident on N.

Example 3.1 Figure 2 shows two database schemas S_1 and S_2 that describe two different sections of a company. Schema S_1 deals with employees, their supervisors – who are also employees –, the department where they work and their dependent persons. Schema S_2 deals with departments, their manager – who is an employee –, and the products the departments have created. Schema S_1 describes mainly employees, while schema S_2 describes mainly departments. In Figure 2, a big black circle denotes an entity node, while a smaller white one denotes an attribute node. The name of a construct is depicted close to the respective construct. An attribute edge between a relationship edge and an attribute node is distinguished by a black bullet in the intersection of the attribute edge and the relationship edge, e.g. attribute edge E_6 in S_1. The name of a manifest attribute node of an entity node is shown in bold, e.g. attribute nodes ENO and $DENAME$ in schema S_1. Unique constraints are denoted by underlying the name of the involved attribute node, e.g. attribute nodes DNO and $DNAME$ in scema S_1. An alternative name for an entity node provided by the renaming function of a relationship edge is shown in typewriter font by the node and the edge, e.g. name $SEPM$ (standing for supervisor employee) assigned to one occurrence of node EMP by the renaming function of the edge $SUPERVISES$ in schema S_1. The other occurrence of node EMP in $SUPERVISES$ is assigned the name EMP which is not displayed in the figure. □

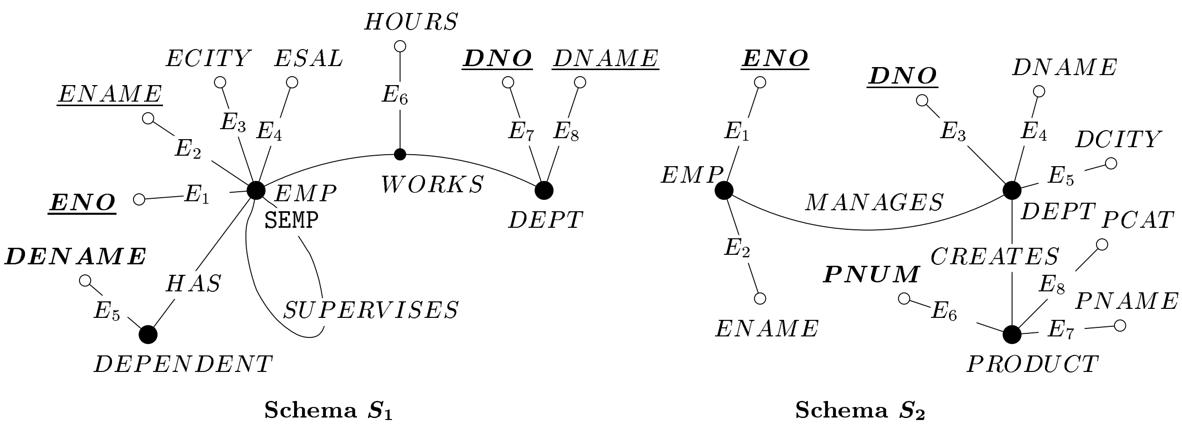


Figure 2: Two database schemas

Database instance. A database instance is defined using the notion of a relation. We follow the approach to the relational model that uses relation attributes. The attributes of the relations are node names.

We first define the *schema of an edge*. If E is an attribute edge between an entity node N and an attribute node A, and L is the name assigned to N by the node renaming function of E, the schema of E is the set $\{L, A\}$. The schema of a relationship edge is the range of the node renaming function associated with this edge (the set of node names assigned by the renaming function to the occurrences of nodes in the edge). The schema of an attribute edge between a relationship edge having schema $\{L_1, \ldots, L_k\}$, and an attribute node A is the set $\{L_1, \ldots, L_k, A\}$.

An *instance* $I(E)$ of an edge E is a relation such that:

1. The schema of $I(E)$ is the schema of E.
2. The content of $I(E)$ is a finite set of tuples. If L is an attribute in the schema of $I(E)$ and a name for a node N, and t is a tuple in the content of $I(E)$, the value in t for the attribute L belongs to the domain of N ($t[L] \in dom(N)$).

An *instance* I *of a hypergraph* $H = \langle Nodes, Edges \rangle$ is a function that maps each edge in $Edges$ to an edge instance such that if $\{L_1, \ldots, L_k, A\}$, $k \geq 2$, is the schema of the interpretation $I(E)$ of an attribute edge E between a relationship edge E' and an attribute node A then $\Pi_{L_1,\ldots,L_k}(I(E)) \subseteq I(E')$. ($\Pi_X$ denotes the relational projection operation over an attribute set X.) In other words, a property value is associated with an existing association of entities.

The association of the entities of an entity node with property values from the adjacent attribute nodes in a hypergraph instance need not be complete in the sense that an entity may be associated with a property value of one adjacent attribute node but with no property value of another adjacent attribute node.

Let E be the attribute edge in a hypergraph H between an entity node N and an attribute node, and L be the name assigned to N by the node renaming function of N. An instance I of H *satisfies* the manifest constraint $manifest(E)$, if for every edge E' incident on node N and for every name L' assigned to N by the renaming function of E', $\Pi_{L'}(I(E')) \subseteq \Pi_L(I(E))$. Let now E be the edge in H between an entity node or a relationship edge and an attribute node A. Let also $\{L_1, \ldots, L_k, A\}$, $k \geq 1$, be the schema of E. An instance I of H *satisfies* the uniqueness constraint $unique(E)$, if $I(E)$ satisfies the key constraint $L_1, \ldots, L_k$ (in the sense of the relational model) on the schema of $I(E)$.

An *instance of a database schema* $S = \langle N, E, C \rangle$ is an instance of the hypergraph $\langle N, E \rangle$ that satisfies all the constraints in C. In the following, edge names are used to denote both: edges and edge instances. The name of a schema can precede an edge name separated by a period to denote the schema the edge belongs to. □

Example 3.2 Figure 3 shows the instances of some edges in the instances I_1 and I_2 of the respective schemas S_1 and S_2 of Figure 2. Here we do not deal with inconsistency and conflict issues in heterogeneous distributed data sources. We assume that there is a mapping that identifies the same entities in the two instances, and distinguishes different entities. Thus, the same value in the instances of the two schemas correspond to the same entity, and conversely, and the same holds for property values. Further, we assume that I_1 has more information about employees than I_2. In this sense any property value assigned to an employee entity in S_2 is also assigned to the same employee entity in I_1. □

Database. A database is a pair $\langle S, I \rangle$, where S is a database schema and I is an instance of S.

$S_1.E_1$

EMP	ENO
e_1	1
e_2	2
e_3	3

$S_1.E_2$

EMP	$ENAME$
e_1	$Smith$
e_2	$Brown$
e_3	$Jones$

$S_1.E_3$

EMP	$ECITY$
e_1	NY
e_3	LA

$S_1.E_4$

EMP	$ESAL$
e_1	$10,000$
e_2	$50,000$

$S_1.SUPERVISES$

EMP	$SEMP$
e_2	e_1
e_3	e_1

$S_1.WORKS$

EMP	$DEPT$
e_1	d_1
e_2	d_2
e_3	d_2

$S_2.E_3$

$DEPT$	$CITY$
d_1	SF
d_2	LA

Figure 3: Edge instances

4 The hypergraph query language HQL

We introduce in this section the hypergraph query language HQL.

Syntax. A *query* over a database schema S is a hypergraph H such that:

1. An attribute node in H has at most one incident edge.
2. The nodes in H are marked as hidden or shown. In particular, entity nodes are marked as shown.
3. Every edge E in H is associated with a relational algebra expression that involves edge names from S. Given an instance for S, this expression evaluates to a relation R. The schema of R is the schema of E. If L is an attribute in the schema of R and a name for a node N, and t is a tuple in the content of R then $t[L] \in dom(N)$.

An edge expression is built using the usual relational algebra operators: selection (σ_c, where c is a Boolean combination of selection predicates), projection (Π_X, where X is a set of attributes), join ($\bowtie_c$, where c is a conjunction of join predicates), union ($\cup$), intersection ($\cap$), difference ($-$), and attribute renaming ($\rho_{A \rightarrow B}$, where A and B are node names). In particular, the expression of an attribute edge does not involve join operators, while the expression of a relationship edge does not involve selection operators.

Semantics. Let Q be a query over a database schema S. The *answer* of Q over an instance I of S is a relation resulting by evaluating over I an expression e defined as follows:

1. If Q contains only one edge E_1 associated with an expression e_1, e is $P_X(e_1)$, where P_X denotes the duplicate and null preserving projection operator, and X is the set of names (attributes) in the schema of E_1 that correspond to shown nodes. If E_1 is a relationship edge, all the names in the schema of E_1 correspond to shown nodes since the nodes of E_1 are entity nodes. If E_1 is an attribute edge, one name in the schema of E_1 correspond to an attribute node that can be a shown or a hidden node.

2. If Q contains only attribute edges $E_1, \dots, E_n$ associated with the expressions $e_1, \dots, e_n$, respectively, that involve only attribute renaming operators (if any), e is the expression $P_X(e_1 ⟗ \dots ⟗ e_n)$, where $⟗$ denotes the natural full outer join operation, and X is the set of names in the schemas of $E_1, \dots, E_n$ that correspond to shown nodes.
3. If Q does not fall in any of the previous two cases, it contains edges $E_1, \dots, E_n$, $n \geq 2$, associated with the expressions $e_1, \dots, e_n$, respectively, such that E_1 is a relationship edge, or e_1 involves operators other than attribute renaming operators. Then e is the expression $P_X(e_1\ \theta_1 \dots \theta_{n-1}\ e_n)$, where θ_i is: (a) the natural join operator, $\bowtie$, if E_{i+1} is a relationship edge or e_{i+1} involves operators other than attribute renaming operators, and (b) the natural left outer join operator, $⟕$, otherwise (in this case E_{i+1} is an attribute edge involving only attribute renaming operators, if any). X is again the set of names in the schemas of $E_1, \dots, E_n$ that correspond to shown nodes.

Notice that even though the instances of the edges of a schema are sets of tuples and do not contain null values, the answer of a query is a multiset of tuples and can contain null values. A null value can appear in a tuple in the answer of a query only for an attribute corresponding to an attribute node. Null values may occur in the answer of a query only as a result of the incompleteness of the association of the entities of an entity node with property values from an adjacent attribute in a database instance. These semantics are appropriate for queries because, in the framework of data source integration, we often deal with missing data.

Views. A *view* is a named query. When the answer of a view is stored, the view is called *materialized*. Otherwise, it is called virtual. In our approach, a mediator can contain a virtual or a materialized view.

Example 4.1 Consider the database schema S_1 of Figure 2. View V_1 (Figure 4) over S_1

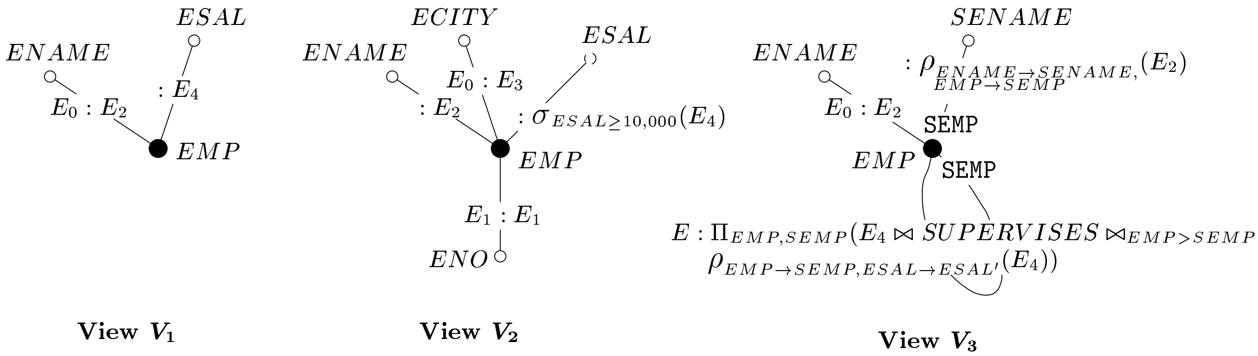


Figure 4: Views over schema S_1

retrieves the names of the employees and their salaries. The expression of an edge is shown in the Figures by the corresponding edge preceded by a colon. The name of an edge is often omitted in the Figures (e.g. the name of the edge between nodes EMP and $ESAL$ in view V_1). When it is not omitted, it precedes the colon (e.g. the name E_0 of the edge between

EMP	$ENAME$	$ESAL$
e_1	$Smith$	$10,000$
e_2	$Brown$	$50,000$
e_3	$Jones$	null

(a)

EMP	ENO	$ENAME$	$ECITY$
e_1	1	$Smith$	NY
e_2	2	$Brown$	null

(b)

EMP	$SEMP$	$ENAME$	$SENAME$
e_2	e_1	$Brown$	$Smith$

(c)

EMP	$ENAME$	$DEPT$
e_3	$Brown$	d_2

(d)

EMP	$ECITY$
e_1	NY
e_2	null

(e)

Figure 5: View answers

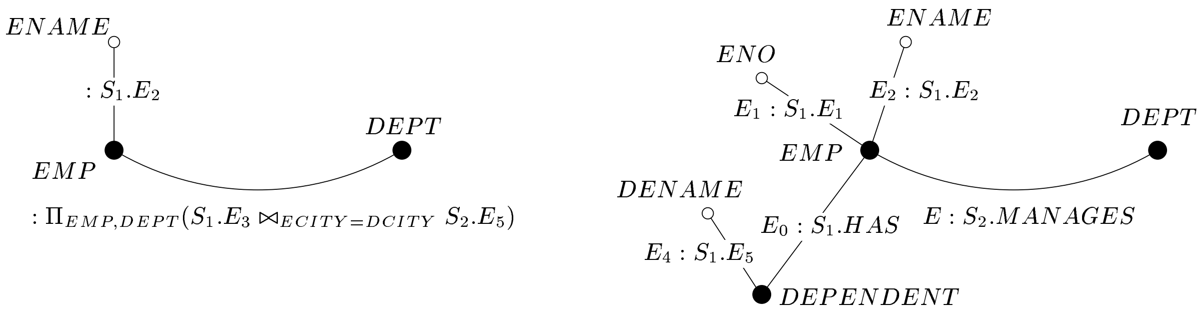
nodes EMP and $ENAME$ in view V_1). The expression that computes the answer of V_1 is $E_2 \text{ ⟗ } E_4$. The answer of V_1 over the instance I_1 of S_1, partially shown in Figure 3, is displayed in Figure 5(a).

View V_2 (Figure 4) over S_1 retrieves the numbers and the names of the employees and the cities where they live but only for the employees whose salary is greater than or equal to 10,000. Attribute node $ESAL$ is a hidden node in V_1. Hidden nodes in the Figures are shown with dashed lines. The answer of V_2 is computed by the expression $P_{EMP,ENO,ENAME,ECITY}(\sigma_{ESAL \geq 10,000}(E_4)$ $\text{ ⟕ } E_1 \text{ ⟕ } E_2 \text{ ⟕ } E_3)$. The result of the computation over the instance I_1 of S_1 is shown in Figure 5(b).

View V_3 (Figure 4) over S_1 retrieves the names of the employees whose salary is greater than the salary of their supervisor and the names of their supervisors. The supervisor's name is provided under the attribute $SENAME$. Notice that the node name $SEMP$ can be assigned to the node EMP by the node renaming function of the attribute edge between EMP and $SENAME$ because $SEMP$ has been assigned to EMP by the node renaming function of the relationship loop edge on EMP. The answer of V_3 is computed by the expression $E_2 \text{ ⟖ } \Pi_{EMP,SEMP}(E_4 \bowtie SUPERVISES \bowtie_{ESAL>ESAL'} \rho_{ESAL \rightarrow ESAL'}(E_4))$ $\text{ ⟕ } \rho_{ENAME \rightarrow SENAME}(E_2)$. The result of the computation over the instance I_1 of S_1 is shown in Figure 5(c). □

Data source integration. New edges that are not restrictions of existing schema edges can appear in a view. In other words, an edge among some nodes in a view may not be associated with an expression that involves an edge among the corresponding schema nodes in the schema. Views can also be defined over multiple database schemas. These views involve edge names from different schemas, and provide an integration of multiple data sources.

Example 4.2 View V_4 (Figure 6) retrieves the names of the employees who work for a department located in the city where they live. Schema names precede edge names in the view.



View V_4 **View V_5**

Figure 6: Views over schemas S_1 and S_2

The answer is computed by the expression $S_1.E_2 \bowtie\!\sqsubset \Pi_{EMP,DEPT}(S_1.E_3 \bowtie_{ECITY=DCITY} S_2.E_5)$. The result of the computation over the instance I_1 of schema S_1 and the instance I_2 of schema S_2 (Figure 3) is shown in Figure 5(d). View V_5 (Figure 6) retrieves the numbers and the names of the managers who have at least one dependent and the names of their dependents. The answer is computed by the expression $S_1.E_5 \bowtie\!\sqsubset S_1.E_1 \bowtie\!\sqsubset S_1.HAS \bowtie S_2.MANAGES \sqsupset\!\bowtie S_1.E_2$. □

Queries and views can be defined using other views. A view can be seen as a schema. Given database instances, instances for the view edges are defined as follows: let $L_1, \ldots, L_n$ be the schema of an edge E in a view V, and R be the answer of V. The instance of E is the relation $\Pi_{L_1,\ldots,L_n}(R)$. Π is the relational algebra projection operator and it removes tuples that contain null values. As with schema edges, the name of a view edge is used to name its instance. Note the difference between the instance of a schema edge and the instance of a view edge. If E is a schema edge and also the expression associated with a view edge E', the instance of view edge E' is a subset of the instance of schema edge E. As with schemas, view names precede edge names separated by a period in a view to indicate the view they belong to. Conversely, a schema edge can be seen as a "view". The expression associated with this edge is a constant relation (the instance of the edge). This relation is also the answer of the view.

Example 4.3 View V_6 (Figure 7) retrieves the cities where the employees having a salary greater than or equal to $10,000$ live. It is defined using exclusively view V_2. The edge between node EMP and node ENO (the latter being the manifest attribute node of node EMP in the schema S_1) has to be included in the view if null values need to be retrieved. The answer of view V_6 over the instances I_1 and I_2 of schemas S_1 and S_2, respectively, is provided in Figure 5(e). View V_7 (Figure 7) retrieves the names of the managers who have no dependents. This view is defined using schemas S_1 and S_2 and view V_5. Its answer is computed by the expression $S_1.E_2 \bowtie\!\sqsubset (S_1.E_1 - V_5.E_1) \bowtie S_2.MANAGES$. View V_8 (Figure 7) retrieves the names of the employees who are managers and have dependents or whose salary is greater than the salary of their supervisor, but only of employees whose salary is known. View V_8 is

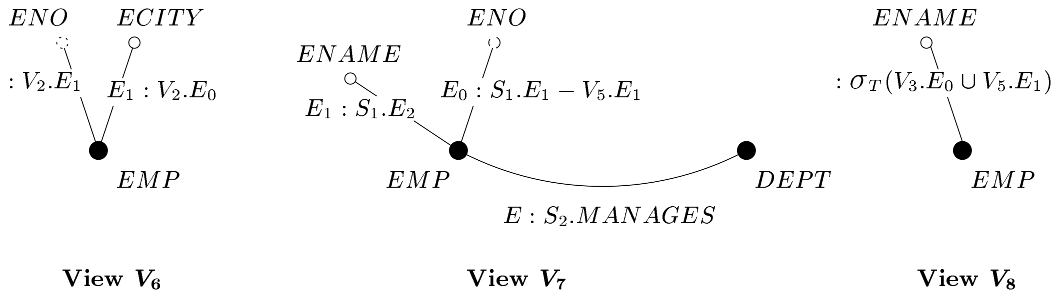


Figure 7: Views defined using other views

defined using exclusively views V_3 and V_5. The expression that computes the answer is the expression associated with the single edge of the view. T in the selection condition stands for a valid formula (that is, a formula that is true for any instance of the attributes that occur in it, e.g. $ENAME = ENAME$). □

HQL and relational languages. The expressions associated with the edges of an HQL query are relational algebra expressions, while those that compute the answer of an HQL query are extended relational algebra expressions. There are two advantages related to this feature: (a) HQL queries over HDM schemas can be easily translated to queries in a popular relational query language (e.g. SQL) over relational schemas and conversely, and (b) Well known query optimization techniques for centralized [8] and distributed [17] database systems can be employed for evaluating HQL queries.

5 View transformations

A view defined over database schemas and/or other views can also be constructed, in a step-by-step way, using view transformations. The transformations formalize changes to a view and thus, they allow us to deal with view evolution issues. A transformation is applied to a *source view* and returns a *target view*. The application of a transformation assumes, besides the source view, a set of database schemas and/or views over which the source view is defined. This set is called *integration environment*. Here we concentrate mainly on primitive (view) transformations.

Primitive transformations. We define two primitive transformations: *InsertEdge* and *DeleteEdge*. The primitive transformation *InsertEdge* inserts an edge in the source view. The source view can be absent. In this case a new view is created. If the source view is not absent, those of the new edge nodes that already exist in the source view are merged with the corresponding existing nodes of the source view. The rest of the nodes of the new edge are inserted as new nodes in the source view. *InsertEdge* has one of the following forms depending on whether the edge to be inserted is a relationship edge or an attribute edge:

$InsertEdge(R;\ View\_name;\ Edge\_name;\ Entity\_node\_1,\ Renamings\_1,\ \ldots\ ,$
$\quad Entity\_node\_n,\ Renamings\_n;\ Edge\_expression)$ or

$InsertEdge(A;\ View\_name;\ Edge\_name;\ Construct,\ Renaming,\ Attribute\_node;$
$\quad Attribute\_node\_status;\ Edge\_expression).$

The first parameter in both forms of the transformation determines the type of edge to be inserted. It is A, if the edge to be inserted is an attribute edge and R, if it is a relationship edge. $Edge\_name$ is the name of the edge to be inserted. $View\_name$ is the name of a view where $Edge\_name$ is to be inserted. If $View\_name$ is not specified, a new view is to be created. $Entity\_node\_1$, ... , $Entity\_node\_n$ are the nodes of the relationship edge $Edge\_name$, and $Renamings\_1$, ..., $Renamings\_n$ are sets of names assigned to these nodes, respectively, by the node renaming function of $Edge\_name$. There are as many distinct names in $Renamings\_i$ as are the occurrences of $Entity\_node\_i$ in $Edge\_name$. $Construct$ is the entity node or the relationship edge of the attribute edge $Edge\_name$. If $Edge\_name$ is an attribute edge between an entity node and an attribute node, $Renaming$ is a singleton containing the name assigned to node $Construct$ by the node renaming function of $Edge\_name$. If $Edge\_name$ is an attribute edge between a relationship edge and an attribute node, $Renaming$ is an empty set. $Attribute\_node$ is the attribute node of $Edge\_name$. Parameter $Attribute\_node\_status$ takes values H or S depending on whether $Attribute\_node$ is to be a hidden or a shown attribute node, respectively, in the target view. $Edge\_expression$ is the expression associated with $Edge\_name$, and involves edge names from the integration environment.

The primitive transformation $DeleteEdge$ removes an edge from the source view. Those of the nodes of the edge that do not have other incident edges besides this edge are also removed from the source view. The target view can be absent. In this case the source view is deleted. $DeleteEdge$ has the following form:

$DeleteEdge(View\_name;\ Edge\_name).$

$View\_name$ is the name of the view from which the edge is to be deleted and $Edge\_name$ is the name of the edge in view $View\_name$ to be deleted.

Example 5.1 Suppose that the integration environment comprises the database schemas S_1 and S_2 of Figure 2 and the view V_5 of Figure 7. A view identical to view V_5 of Figure 6 can be constructed by the following sequence of primitive transformation applications to V_7. We assume that when primitive transformations are applied in sequence to a view V_x, the resulting views are named $V_{x1}, V_{x2}, \ldots$.

$DeleteEdge(V_7; E_1)$
$InsertEdge(A;\ V_{71};\ E_2;\ EMP,\ \{EMP\},\ ENAME,\ S;\ S_1.E_2)$
$DeleteEdge(V_{72}; E_0)$
$InsertEdge(A;\ V_{73};\ E_1;\ EMP,\ \{EMP\},\ ENO,\ S;\ S_1.E_1)$
$InsertEdge(R;\ V_{74};\ E_0;\ EMP,\ \{EMP\},\ DEPENDENT,\ \{DEPENDENT\};\ S_1.HAS)$
$InsertEdge(A;\ V_{75};\ E_4;\ DEPENDENT,\ \{DEPENDENT\},\ DENAME,\ S;\ S_1.E_5).\Box$

Application constraints. The primitive transformation are not always applicable. Application constraints result by the fact that $edge\_name$ cannot be inserted in or deleted from the source view $view\_name$ if the graph resulting by this insertion or removal is not a view.

Example 5.2 Consider the sequence of transformation applications of Example 5.1. In general, the order of application of the transformations matters. For instance, the third and the fourth transformation applications cannot precede the first two because an edge named E_1 already exists in V_7.

Transformation $DeleteEdge(V_3;\ E)$ cannot be applied to view V_3 of Figure 4 to delete the loop edge on EMP: if the loop edge on EMP is removed from V_3, the name assigned to the entity node EMP by the node renaming function of the attribute edge between nodes EMP and $SENAME$ in the resulting graph has not been assigned to EMP by the node renaming function of a relationship edge in this graph. □

Composite transformations. A number of additional transformations are useful for constructing a view from a source view. These transformations modify any of the following elements of the source view: the name of an edge, the nodes of an edge, the names assigned to a node of an edge by the node renaming function of this edge, the status (hidden or shown) of an attribute node in an attribute edge, the expression associated with an edge. Other transformations combine multiple such modifications. For instance, the modification of the expression associated with an edge may require the modification of the nodes of the edge and the names assigned to them by the node renaming function of the edge. All these transformations are called *composite transformations.* A composite transformation application can be expressed as a sequence of primitive transformations applications.

Example 5.3 In Example 5.1, the sequence of the first two transformation applications deletes and inserts an attribute edge in order to modify its name (from E_1 to E_2). The sequence of the next two transformation applications inserts an attribute edge in order to modify its name (from E_0 to E_1), its associated expression (from $S_1.E_1 - V_5.E_1$ to $S_1.E_1$) and the status of its attribute node (from H to S). □

Completeness of the primitive transformations. A set of transformations is called *complete* if given any two views V_1 and V_2 defined over an integration environment, V_1 can be transformed to V_2 by applying the transformations in this set. Clearly, the set of the two primitive transformations presented in this section is complete.

Database schema transformations. Schema transformations are a special case of view transformations since, as mentioned above, a schema can be seen as a set of views. The syntax of the primitive schema transformations is similar to that of the primitive view transformations. By convention, $View\_name$ is the name of the schema. $Edge\_expression$ in the $InsertEdge$ transformation is a constant relation. The schema S of this relation depends on the type of the edge $Edge\_name$ to be inserted in the schema. If $Edge\_name$ is a relationship edge, S is the set $Renamings\_1\ \cup \ldots \cup\ Renamings\_n$. If $Edge\_name$ is an attribute edge between the entity node $Construct$ and the attribute node $Attribute\_node$, S is the set $Renaming\ \cup\ \{Attribute\_node\}$. If $Edge\_name$ is an attribute edge between the relationship edge $Construct$ and the attribute node $Attribute\_node$, S is the set $Schema\ \cup$ $\{Attribute\_node\}$, where $Schema$ is the schema of the relationship edge $Construct$. The primitive schema transformations are subject to a number of application constraints: a primitive schema transformation is not applicable if the graph resulting by the insertion or deletion of an edge violates the definition of a schema.

Example 5.4 The transformation $InsertEdge(A;\ S_1;\ E_8;\ WORKS,\ \emptyset,\ HOURS,\ s;\ \{EMP, DEPT, HOURS\})$ cannot be applied to the database schema S_1 of Figure 2 because: (a) E_8 names an existing edge in S_1, and (b) there is already an edge between the relationship edge $WORKS$ and the attribute node $HOURS$ in S_1. □

Transformation reversibility. A set of transformations is called *reversible* if whenever two views V_1 and V_2 defined over an integration environment are such that V_2 can be obtained from V_1 by applying the transformations of the set in sequence, V_1 can also be obtained from V_2 by a sequence of applications of transformations in this set. The set of the two primitive transformations presented here is reversible even in a stronger sense: each primitive transformation is the reverse of the other.

Example 5.5 Consider the sequence of transformation applications of Example 5.1 to view V_7 resulting in view V_{75} (which is identical to view V_5). A view identical to view V_7 can be obtained by applying to V_5 the inverted sequence of the reverse of each transformation:

$DeleteEdge(V_5;\ E_4)$
$DeleteEdge(V_{51};\ E_0)$
$DeleteEdge(V_{52};\ E_1)$
$InsertEdge(A;\ V_{53};\ E_0;\ EMP,\ \{EMP\},\ ENO,\ S;\ S_1.E_1 - V_5.E_1)$
$DeleteEdge(V_{54};\ E_2)$
$InsertEdge(A;\ V_{55};\ E_1;\ EMP,\ \{EMP\},\ ENAME,\ H;\ S_1.E_2)$. □

6 Conclusion

A typical approach for integrating heterogeneous data sources involves translating their modeling languages into a common data model and defining mediators. Mediators can be seen as views of the data stored in the data sources. We have presented HDM, a hypergraph based data model, and HQL, a hypergraph based query language. HDM is used as a low-level common data model. HQL is used for integrating HDM schemas through the definition of views, and for querying HDM schemas and views. The semantics of HQL are provided by relational algebra expressions that allow us to deal with missing information and permit the use of well known query optimization techniques for evaluating queries. We have introduced a set of primitive transformations, which are complete and reversible. These transformations can be used for constructing a view from another view in an integration environment, and for dealing with view evolution issues. In our approach some or all of the mediator views can be materialized (hybrid data warehousing approach).

Our current research focuses on two issues: (a) Extending HQL with grouping/aggregation queries. This type of queries is extensively used in data warehousing applications. (b) Determining when a query defined using the source view of a primitive transformation can be rewritten using the target view. Answering this question allows us to avoid the expensive implementation of new mediators when the integration environment evolves.

References

[1] T. Catarci, G. Santucci, and J. Cardiff. Graphical interaction with heterogeneous databases. *VLDB Journal*, 6:97–120, 1997.

[2] M. P. Consens and A. O. Mendelzon. A Visual Formalism for Real Life Recursion. In *Proc. ACM Symp. Principles of Database Systems*, pages 404–416, 1990.

[3] M. P. Consens and A. O. Mendelzon. Hy+: A Hygraph-based Query and Visualization System. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 511–516, 1993.

[4] M. P. Cosens, F. C. Eigler, M. Z. Hasan, A. O. Mendelzon, E. G. Noik, A. G. Ryman, and D. Vista. Architecture and Applications of the Hy+ Visualization System. *IBM Systems J.*, 33(3):458–476, 1994.

[5] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G+: Recursive Queries Without Recursion. In *Proc. 2nd Intl. Conf. on Expert Database Systems*, pages 645–666, 1988.

[6] R. Domenig and K. Dittrich. An Overview and Classification of Mediated Query Systems. *SIGMOD Record*, 28(3), 1999.

[7] R. Elmasri and B. Navathe. *Fundamentals of Database Systems, Third Edition*. Addison-Wesley, 2000.

[8] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):183–236, 1994.

[9] M. Gyssens, J. Paredaens, and D. V. V. Gucht. A Graph-Oriented Object Model for Database End-User Interfaces. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 24–33, 1990.

[10] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based Wrappers in the TSIMMIS System. *SIGMOD Record*, 26(2):532–535, 1997.

[11] R. Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In *Proc. of the 16th ACM Symp. on Principles of Database Systems*, pages 51–61, 1997.

[12] R. Hull and G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 481–492, 1996.

[13] A. Levy, A. O. Mendelson, Y. Sagiv, and D. Srivastava. Answering Queries using Views. In *Proc. of the ACM Symp. on Principles of Database Systems*, pages 95–104, 1995.

[14] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of the 22nd Intl. Conf. on Very Large Data Bases*, pages 252–262, 1996.

[15] P. McBrien and A. Poulovassilis. A Uniform Approach to Inter-Model Transformations. In *Proc. of the 11th Intl. Conf. on Advanced Information Systems Engineering, Springer-Verlag, LNCS No 1626*, pages 333–348, 1999.

[16] P. McBrien and A. Poulovassilis. A Semantic Approach to Integrating XML and Structured Data Sources. In *Proc. of the 13th Intl. Conf. on Advanced Information Systems Engineering*, 2001.

[17] M. T. Oszu and P. Valduriez. *Priciples of Distributed Database Systems*. Prentice-Hall, 2nd edition, 1999.

[18] Y. Papakostantinou, H. Garcia-Molina, and J. Widom. Object Exchange across Heterogeneous Information Sources. In *Proc. of the 11th Intl. Conf. on Data Engineering*, 1995.

[19] A. Papantonakis and P. J. H. King. Syntax and Semantics of Gql, A Graphical Query Language. *J. Visual Languages and Computing*, 6(1):3–36, 1995.

[20] J. Paredaens, P. Peelman, and L. Tanca. G-Log: A Declarative Graphical Query Language. In *Proc. 2nd Intl. Conf. Deductive and Object-Oriented Databases*, pages 108–128, 1991.

[21] A. Poulovassilis and P. McBrien. A General Formal Framework for Schema Tranformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.

[22] M. T. Roth and P. Schwartz. Don't scrap it, wrap it! In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 266–275, 1997.

[23] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[24] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 126–135, 1997.

[25] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):808–823, 1998.

[26] J. Ullman. Information Integration Using Logical Views. In *Proc. of the 6th Intl. Conf. on Database Theory*, 1997.

[27] J. Widom, editor. *Data Engineering, Special Issue on Materialized Views and Data Warehousing*, volume 18(2). IEEE, 1995.

[28] J. Widom. Research Problems in Data Warehousing. In *Proc. of the 4th Intl. Conf. on Information and Knowledge Management*, pages 25–30, Nov. 1995.

[29] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, 1992.

[30] G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Supporting Data Integration and Warehousing Using H20. *Data Engineering*, 18(2):29–40, 1995.

[31] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 316–327, 1995.