Towards a Mediator Development Environment: The AURORA Approach

Ling Ling Yan, M. Tamer Özsu, and Ling Liu
Laboratory for Database Systems Research, Department of Computing Science
University of Alberta, Edmonton, Alberta, T6G 2H1
email: {ling, ozsu, lingliu}@cs.ualberta.ca

Abstract

The AURORA mediator architecture uses specialized mediators to provide integrated access to heterogeneous databases. It is an instance of the mediator framework in [Wied92]. A mediator development environment is designed to support construction of AURORA mediators by mediator authors. This environment consists of a collection of workbenches, each supporting construction of a specific type of mediator. We demonstrate the AURORA approach by presenting a specific suite of techniques for building high performance relational homogenization mediators. This includes a special mediation methodology, new view operators, a query rewriting algorithm and transformation rules for logical query optimization.

1 Introduction

Data from multiple information sources maybe heterogeneous in semantics and representation. This heterogeneity poses a major difficulty in providing integrated access to the data. The *mediator architecture* proposed in [Wied92] is widely accepted as a general framework for coping with this problem. The goal of the AURORA project is to develop environment tools to support construction of mediators, and to develop techniques for mediator view expression and efficient query processing. This project consists of a reference mediation framework and a mediator development environment.

1.1 The AURORA Reference Mediator Architecture

The AURORA reference mediator architecture, shown in Figure 1, is an instance of the general mediator framework [Wied92]. It has multiple incarnations depending on the canonical data model and query language assumed. Wrappers and mediators present their schemas in the canonical data model and entertain queries in the canonical query language. Wrappers handle idiosyncrasies of the data sources. If the source data model and query language are different from the canonical ones, the wrapper must translate queries from the latter to the former and also assemble query answers

using data retrieved from the source. Mediators are not concerned with such specifics; they focus on providing a desirable view that can be queried by applications. Mediators are application dependent while wrappers are not. There are two types of mediators in Figure 1, the homogenization mediators and the integration mediators.

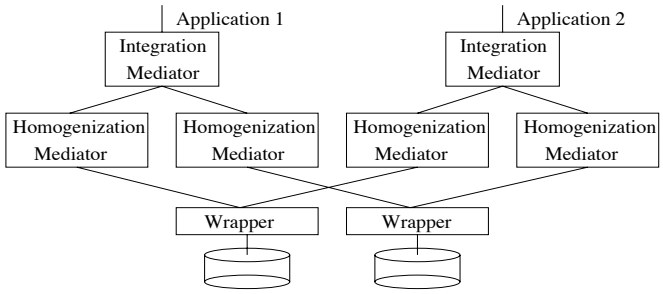


Figure 1: The AURORA Mediation Framework

Integrating multiple data sources is a complicated task when multiple types of heterogeneities [Kent91, Kim93] exist simultaneously. We distinguish between two categories of heterogeneities:

1. Schematic mismatches that arise when data sources model the same application differently.

2. Instance level conflicts that arise when data sources record inconsistent values/descriptions on the same real world object. These conflicts are relevant only if instance level object matching among multiple data sources is performed.

Schematic mismatches must be resolved first; otherwise, there is no basis for further integration since data from different databases may not be comparable. The process of resolving schematic mismatches is referred to as *homogenization*. In AURORA, specialized mediators, the *homogenization mediators*, support this process. The task of homogenization is to map a common application model into each participating database that models the same application (differently). Once this common model is defined, each database is homogenized independently. Homogenization mediators are application specific. In Figure 1, a data source is accessed by multiple homogenization mediators(multiple applications) but each such mediator accesses only one data source. Resolutions of instance level conflicts necessarily span all participating databases; these resolutions are also handled by specialized mediators, the *integration mediators*.

Rather than assuming a certain data model to be the most "suitable" for mediation, we intend to develop techniques for a variety of mediators, as discussed in the next section.

1.2 The AURORA Mediator Development Environment

The mediator development environment in AURORA consists of a collection of workbenches, each consisting of a mediator skeleton and a Mediator Author's Toolkit (MAT).

AURORA Mediator Skeletons. A mediator consists of an integrated view over (multiple) data sources and a query processor that answers queries posed against this view in a canonical query language. Based on the view definition, the query processor translates the view queries into queries over data sources and assemble query answers from the query results returned by the sources. Building a mediator means building the view and the query processor. In AURORA, mediators are constructed from *mediator skeletons* that have the following built-in capabilities:

1. A framework for defining views and a repository to maintain them; and

2. A query processor that entertains queries posed against views expressed in this framework.

Indeed, mediator skeletons are mediators with an empty view. Once a view is defined and stored in its repository, a mediator skeleton becomes a custom-made mediator that is able to process queries posed against this view. Different types of mediators requires skeletons that have different view expression capabilities and query processors.

AURORA Mediator Author's Toolkits (MATs). In AURORA, a *mediator author* chooses a mediator skeleton, identifies heterogeneities among the sources, and defines views into the mediator skeleton to resolve the heterogeneities. AURORA provides interactive tools, the Mediator Author's Toolkits (MATs), to assist the mediator authors in performing such tasks. This scenario is shown in Figure 2(a). A MAT has the following capabilities:

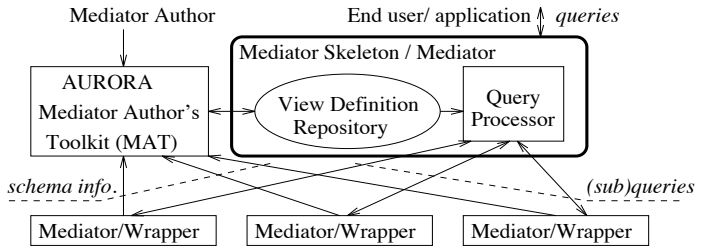
1. It mandates a *mediation methodology* for detecting and resolving heterogeneities; and

2. It provides view operators for expressing the resolutions.

Construction of different types of mediators require different methodologies and operators.

AURORA Mediator Development Environment. The AURORA mediator development environment consists of a collection of workbenches. A *workbench* consists of a mediator skeleton and a MAT. Figure 2(a) shows the general form of a workbench. Intuitively, a workbench is an environment where mediators of a particular type can be constructed by a mediator author. Construction of different mediators require different suites of MAT and skeleton. Rather than providing one workbench for all, AURORA provides a collection of workbenches. These workbenches are classified along two dimensions: 1) canonical data model and query language supported by the mediator; and 2) the specialty of the mediator, homogenization or integration. Figure 2(b) shows

this classification for two possible canonical data models: relational and object-oriented.



AURORA Workbenches

Mediator specialty \ Canonical Model	*Relational*	*Object-Oriented*
Homogenization	AURORA-RH	AURORA-OH
Integration	AURORA-RI	AURORA-OI

(b)

Figure 2: AURORA Workbench: General Form and Classification

1.3 The Scope and Contributions of This Paper

In this paper, we present detailed techniques used by AURORA-RH (Figure 2(b)), the workbench for building high performance relational homogenization mediators. It consists of a MAT and a mediator skeleton tightly coupled through a view mechanism. Our contributions include a homogenization methodology and mediator skeleton techniques that support powerful view mapping and efficient query processing.

Homogenization Methodology in AURORA-RH. Homogenization can be a complicated process when multiple types of domain and schema mismatches exist simultaneously. In practice, not only do we need language constructs and/or operators to express resolutions, we also need a methodology to ensure that the resolutions are correct and complete. Moreover, such methodologies should be supported by tools. Little previous research exists in this regard. In AURORA, MATs support mediation methodologies as well as powerful view operators. In particular, we establish a *homogenization methodology*, to be employed by mediator authors and mandated by the MAT in AURORA-RH, for homogenizing relational databases.

View Operators and Query Processing in AURORA-RH. A *homogenizing view* maps a common application model into a data source. This view is different from the usual relational views. First, it requires more powerful structural mapping. A relation in the homogenizing view may have an attribute whose values correspond to relation or attribute names in the underlying database. This is referred to as a *cross-over schema mismatch* [Kent91]. It has been argued that relational view mechanism can not express such mappings [KLK91]. Second, it requires more powerful value mapping. When defining homogenizing views, it may be necessary to use arbitrary functions/look-up tables to derive values of a data domain in this view from one or more domains

in the underlying database. The relational view does not allow user-defined functions. On the other hand, while not powerful enough for homogenization, the relational view supports view query optimization by modifying them into basic relational queries which are optimized by well-established techniques. More powerful frameworks tend to lose this property by focusing on expressive power and ignoring query optimization issues. For instance, [Kim93], [KLK91] and [Kent91] all provide powerful language constructs for resolving heterogeneities but none discusses query optimization issues in presence of the new constructs.

The AURORA-RH view framework allows both powerful view mapping and efficient query processing. We define a few operators that extend the power of relational algebra to express homogenizing views. The new operators compose with relational operators in a well-defined manner and are easy to implement. Arbitrary functions can be used for value mapping, however, characteristics of these functions, such as whether they are monotonic or have inverses, are used for query optimization. A view query rewriting algorithm and transformation rules involving the new operators are also developed to facilitate optimization of mediator queries.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 gives an overview of AURORA-RH. Section 4 presents AURORA-RH primitives that extend the power of relational algebra. Section 5 describes the MAT in AURORA-RH. Section 6 presents the query processing techniques in AURORA-RH. Section 7 contains conclusion and future work.

2 Related Work

There are many mediation frameworks in the literature, Multidatabase [DH84], Superviews [Mot87], TSIMMIS [PGU96], Garlic [Carey94], HERMES [Subr], DIOM [LPL96], to cite a few. The AURORA project is different from these frameworks in two ways: (1) AURORA defines a 2-tier mediation framework (Figure 1), homogenization followed by integration, that uses specialized mediators to resolve various types of heterogeneities. This specialization allows us to isolate a host of issues in mediator construction. No previous work makes this distinction; and (2) AURORA provides specialized tools, the mediator development workbenches, that assist mediator authors in building specific types of mediators. HERMES provides a mediator programming environment. Compared with AURORA workbenches, this environment is yet to be refined.

[Qian94] and [Goh95] present intelligent mediation techniques that detect and resolve semantic heterogeneities automatically by reasoning about semantics in a knowledge base or ontology. In

AURORA, such tasks are performed by a mediator author using MATs. Once established, intelligent mediation techniques can replace mediator authors. We plan to build APIs to allow such replacement. Until then, the AURORA approach is a practical solution. The AURORA project investigates a host of issues in mediator view expression and query processing that are essential no matter the heterogeneities are detected and resolved automatically or manually.

[Kent91] identifies domain mappings for resolving domain and schema mismatches. Resolutions for individual mismatches are demonstrated using an object-oriented database programming language. [Kent91] does not provide a mediation methodology, nor does it explore query optimization techniques in presence of the new language constructs. [Kim93] provides a comprehensive classification of mismatches and conflicts. Resolutions for individual conflicts are given. New language constructs are proposed but query rewriting and optimization methods for these constructs are not given. [Goh95] uses ontology to detect and resolve mismatches due to different units of measure. It is not clear how [Goh95] handles other types of schematic mismatches.

Disco [TRV95] extends ODMG ODL for mediation and intends to use Volcano for query optimization. It introduces a logical operator *submit* and gives rules for exchanging relational operators with it. The physical cost model used is unclear. [DKS92, LOG92] describe approaches that collect/establish statistics to build mediator query cost models. In AURORA-RH, we handle a single data source. Currently we concentrate on query modification techniques to leverage the source query optimization capability; a mediator query cost model is not necessary. However, cost model for mediator queries is highly desirable. This is a topic of future research in AURORA.

3 Overview of the AURORA-RH Workbench

3.1 Homogenization Problem and Mediator Query Processing

Let B be a relational database. Let H be a homogenizing view consisting of relations $M_1, ..., M_n$. The problem of *homogenizing database B into H* is to specify procedures, $P_i(B)(1 \leq i \leq n)$, that construct relations $M_i(i = 1, n)$ from the relations in B. B is the *source database*; relations in B are *source relations*; $M_i(i = 1, n)$ are *target relations*. We also refer to H as the *target database*. Queries posed against H are referred to as *mediator queries*. Assume procedures $P_i(B)(1 \leq i \leq n)$ have been specified and consider a mediator query Q. The task of processing Q is to 1) translate Q into queries over B; this process intuitively "reverses" the P_i's specified earlier; and 2) send the queries to the source database and use the returned data to assemble the answer to Q in H.

Example Application. *Figure 3 depicts a homogenization problem. Besides the differences in schema, we also assume: 1) in the source database, the sales and salary data is recorded in Canadian dollars, while in the target database, the same data is to be reported in US dollars; 2) In the target database, Employee.salary includes bonus as well as base salary; and 3) The target database perceives the domain of* jobs *differently from the source database. Rather than having job titles from*

$\{SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector\}$

the target database assumes the job titles are from

$\{System\ Engineer, Development\ Engineer, Consultant, Research\ Scientist, Program\ Manager\}$

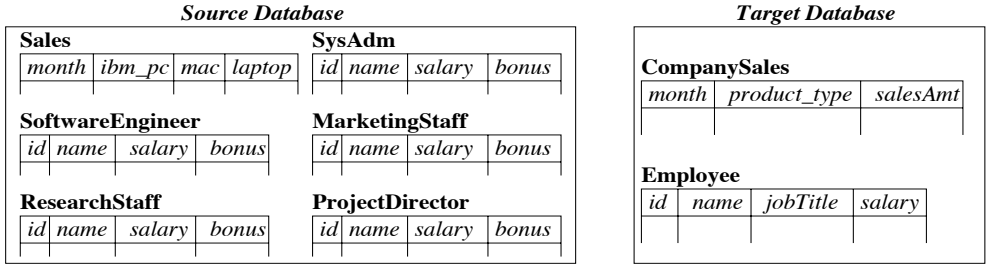


Figure 3: A Homogenization Problem

Each database defines *domains* that model *conceptual territories*. Domains can be a meta domain or a data domain. A domain is characterized by its conceptual territory and the unit, population, and data type of its elements. Domains from different databases that model the same concept are said to be *corresponding* domains (called a *domain group* in [Kent91]). These domains may be different from each other, giving rise to *domain mismatches*. A domain mismatch that involves a meta domain is referred to as a *schema mismatch*. Corresponding domains are converted to each other via *domain mappings*. To interpret the source database from the target database, the target must be mapped into the source, resolving domain and schema mismatches between the two. In AURORA, a mediator author defines this mapping using AURORA-RH.

3.2 Architecture of AURORA-RH

The architecture of AURORA-RH is shown in Figure 4. **MAT-RH** is a toolkit that assists a mediator author in constructing a homogenizing view, or a target database. It provides a set of view operators and mandates a homogenization methodology. The homogenization process is divided into 6 steps, each supported by a specialized tool (sections 5.3-5.8). In each tool, two types of information can be specified: transformation and domain mapping. Transformations are expressions

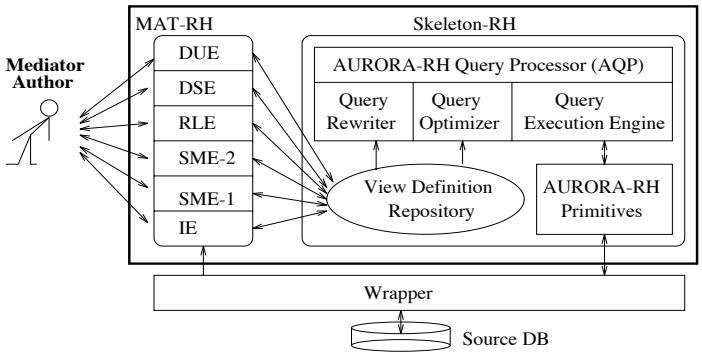


Figure 4: Architecture of AURORA-RH Workbench

consisting of the AURORA-RH view operators and the usual relational operators. Domain mappings are arbitrary mappings. Transformations and domain mappings are captured in the **View Definition Repository** and are used for query processing. **AURORA-RH Primitives** are new view operators that facilitate homogenization; they form an extension of the relational algebra. **AURORA-RH Query Processor (AQP)** processes mediator queries posed against the target database. It translates such a query into a (optimal) set of queries over the source database, sends these queries for execution and assembles the answer to the mediator query from the returned data.

The **Wrapper** is not part of the AURORA-RH. It translates queries from the canonical query language to the query language supported by the source. For AURORA-RH, the canonical query language is the relational algebra. If the source is relational, the translation performed by the wrappers is from a relational algebraic expression to an SQL query. This translation is simple because the relational model, algebra and SQL language are mostly standard among commonly used relational database systems. In general, wrappers vary in complexity depending on whether the source data model is different from the canonical one [Yu95, Meng93], and how standard the canonical data model is. Wrapper construction is beyond the scope of this paper.

4 Primitives and Transformations in AURORA-RH

AURORA-RH primitives extend the power of the relational algebra to enable the mediator author to express homogenizing views. All primitives take a relation as an argument and generate a relation; they compose with relational operators in a well-defined manner. The term "attribute" refers to both the name and the data type of the attribute. For simplicity, we do not discuss type checking or inferencing; the techniques established in this paper can be extended with these

features. Two attributes are often considered to be the "same" if they have the same name. We use $ATTR(R)$ to denote the set of attributes in relation R, $RELname(R)$ for the name of relation R, and $ATTRname(A)$ for the name of attribute A. Let B be the source database to be homogenized. AURORA-RH provides the following primitives:

retrieve. Let Q be an algebraic expression over the source relations in database B,

$$R' = retrieve(Q)$$

submits query Q to database B and returns the result table R'.

pad. Let R be a relation, A be an attribute, $A \notin ATTR(R)$, and c a constant,

$$R' = pad(R, A, c)$$

defines a relation R', $ATTR(R') = ATTR(R) \cup \{A\}$. The population of R' is defined by

$$R' = \{t' \mid t'[A] = c;\ t'[A'] = t[A'], t \in R, A' \in ATTR(R)\}$$

Intuitively, for each tuple $t \in R$, *pad* generates a R' tuple t' by "padding" t with a new field A with value c. *pad* is useful for restructuring relations. Consider the relation $SysAdm$ in Figure 3. Let $R' = pad(\ retrieve(SysAdm), jobTitle, \text{"}SysAdm\text{"})$. R' has scheme *(id, name, salary, bonus, jobTitle)* and a population consisting of all the *SysAdm* tuples each *tagged* with relation name *"SysAdm"* as attribute *jobTitle*. The scheme of R' is closer to that of *Employee* than *SysAdm*.

rename. Let R be a relation, $A \in ATTR(R)$, and n be an attribute name, such that no attribute in R has name n, then

$$R' = rename(R, A, n)$$

defines a relation R' with scheme identical to the scheme of R with attribute A renamed to n. The population of R' is defined by the following:

$$R' = \{t' \mid t'[n] = t[A], t'[A'] = t[A'], t \in R, A' \in ATTR(R) - \{A\}\}$$

deriveAttr. Let R be a relation. Let $L_i \subseteq ATTR(R)(i = 1, k)$ be a list of attributes in R. Let $N_i(i = 1, k)$ be attributes. Let f_i be functions of appropriate signatures.

$$R' = deriveAttr(R, L_1, N_1, f_1, ..., L_k, N_k, f_k)$$

defines a relation R', $ATTR(R') = ATTR(R) \cup \{N_1, ..., N_k\}$. The population of R' is defined by:

$$R' = \{t' \mid t'[N_i] = f_i(t[L_i]), 1 \leq i \leq k;\ t'[A] = t[A], A \in ATTR(R) - \{N_1, ..., N_k\}, t \in R\}$$

Intuitively, for each R tuple t, *deriveAttr* generates a R' tuple t' by adding fields N_i to t *(i=1,k)* and sets its value to be $f_i(t[L_i])$, where $t[L_i]$ is the list of values obtained by projecting t over L_i. If an attribute in R has the same name with some N_s $(1 \leq s \leq k)$, this attribute is replaced by N_s.

deriveAttr is used in resolving domain mismatches with arbitrary functions, as shown in sections 5.7 and 5.8. Notice that functions $f_i(1 \leq i \leq k)$ in *deriveAttr* are different from aggregates in

relational query languages; they apply to field(s) in a single tuple, while aggregates are applied to multiple tuples. For example, given a table containing student grades, *deriveAttr* can not be used to derive an attribute "*GradeAverage*"; it can only be used to derive the basic student-grade table. "*GradeAverage*" can then be derived using aggregates in relational views.

A **transformation expression** is a well-formed expression involving relational operators and AURORA-RH operators. It defines the derivation of the scheme and population of a relation from given relations and other arguments. A transformation expression T_E may define a relation, $R = T_E$. If T_E is of the form of $retrieve(Q)$, R is a *direct relation*, otherwise, R is a *derived relation*. Intuitively, a direct relation is the direct result of a query over the source database.

5 Mediator Author's Toolkit in AURORA-RH (MAT-RH)

5.1 Domain and Schema Mismatches Identified in MAT-RH

Consider a source relational database B and a target relation M. The following types of mismatches are identified in MAT-RH:

Cross-over schema mismatches. A **type 1** cross-over mismatch happens when a concept is represented as data in M but as relations in B. A **type 2** cross-over mismatch happens when a concept is represented as data in M but as attributes in B. There are two nonessential cases of cross-over schema mismatches. Case 1, when a concept is represented as relation or attribute in M but as data in B. Case 2, when a concept is represented as relations (attributes) in M but as attributes (relations) in B. These two cases are further discussed in Appendix A.

Domain structural mismatches. A domain structural mismatch happens when a domain in M corresponds to a domain with a different data type or several (related) data domains in B.

Domain unit mismatches. A domain unit mismatch happens when a domain in M assumes different unit of measurement from the corresponding domain(s) in the source database.

Domain population mismatches. A domain population mismatch happens when a domain in M assumes different population from the corresponding domain(s) in the source database.

Example-1. The example shown in Figure 3 demonstrates all of the above mentioned mismatches:
- (cross-over schema mismatch, type 1.) In the target database, the concept of *jobs* is represented as data domain "*jobTitle*" in relation *Employee*, but is represented as relations in the source database.
- (cross-over schema mismatch, type 2.) In the target database, the concept of *product types* is represented as data domain *product_type*, but is represented as attributes in the source database.
- (domain structural mismatch.) In the target database, *salary* means the total income. The same

concept is represented by two data domains, *salary* and *bonus*, in the source database.

- (domain unit mismatch.) In the target database, all money amounts use US dollar as unit, while in the source database, all money amounts reported are in Canadian dollars.

- (domain population mismatch.) The two database schemas assume different domain populations of the concept *jobs*.

5.2 Homogenization Methodology Supported by MAT-RH

Consider deriving a target relation M from a source database B. When multiple mismatches exists between the target and the source, MAT-RH mandates that they be resolved in 6 steps:

1. construct an import schema;
2. resolve type 1 schema mismatches;
3. resolve type 2 schema mismatches;
4. link relations;
5. resolve domain structural mismatches; and,
6. resolve domain unit/population mismatches.

The above 6 steps consist the MAT-RH homogenization methodology. In each step, new view relations are defined over a set of (virtual) relations defined by earlier steps. MAT-RH supports each step by a specialized tool, or environment, that allows certain transformations and mappings to be specified. Some environments provide special transformations for resolving specific types of mismatches. In the following sections, each environment is described along 4 dimensions:

1) Input relations: relations that can be used in defining new view relations.

2) Transformation operators: operators and special transformations provided.

3) Output semantic information: domain mappings and other semantic information allowed.

4) Output relations: new relations (if any) that can be defined.

5.3 Import Environment (IE)

The input to IE (Figure 5) includes all the source relations exported by a source database B. IE produces a set of direct relations of the form $R = retrieve(Q)$, where Q is a relational algebraic expression over database B. No semantic mapping information is produced.

Example-2: Importing source database. The IE allows the mediator authors to choose relations and data of interest from the source database. In our example, all relations are of interest. The importing step produces a set of direct relations $R = retrieve(R)$, where $R \in$ { *Sales, SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector* }.

5.4 SME-1: Solving Type 1 Cross-over Schema Mismatches

Schema Mismatch Environment 1, SME-1 is depicted in Figure 5.

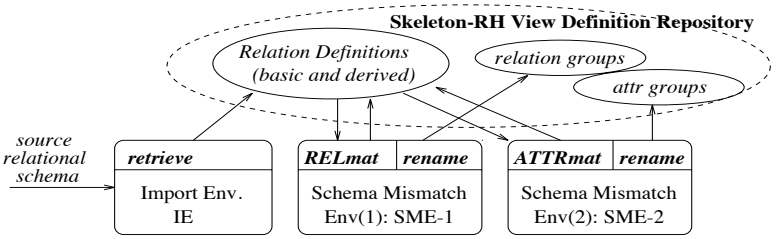


Figure 5: Environments supporting Step 1, 2 and 3 of Homogenization.

Input relations. All relations produced by IE for deriving M.

Transformation operators. $RELmat$, π, σ, $\bowtie$, $rename$, pad, $deriveAttr$.

Operator $RELmat$ (relation materialize) is defined as follows. Given $D^R = \{R_1, ..., R_n\}$, a group of relations with identical schemes, let A be an attribute, $A \notin ATTR(R_1)$, then:

$$RELmat(D^R, A) = \bigcup_{i=1}^{n} pad(R_i, A, RELname(R_i))$$

The result relation has attribute set $ATTR(R_1) \cup \{A\}$. It contains tuples from all the relations in D^R, each tagged with a new field A that contains the name of the relation it came from. This is illustrated in Figure 6. $RELmat$ transforms a meta domain, the relation group, into a data domain. The application of $RELmat$ is illustrated in Example-3.

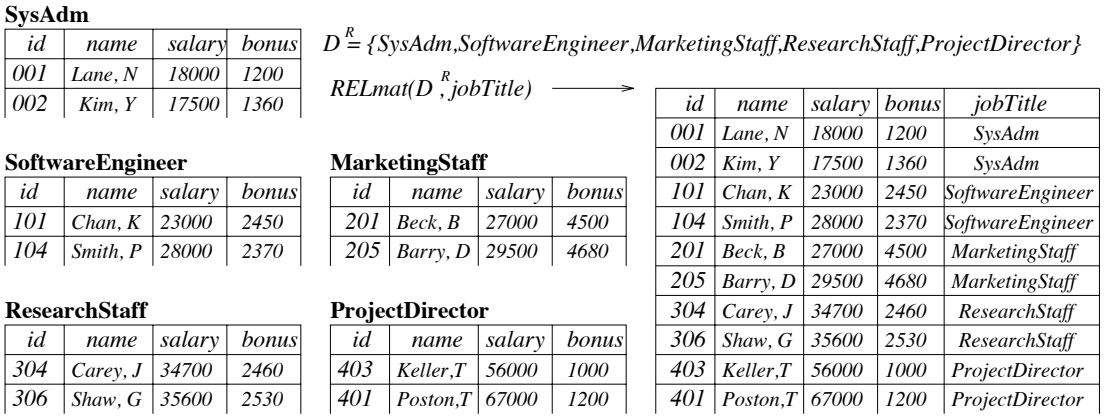


Figure 6: The $RELmat$ operator

Output semantic information. SME-1 allows specification of relation groups. In the target database, the names of the relations in a relation group form an enumerated data domain which represents a concept that is represented as relations in the source database.

Output relations. SME-1 allows multiple derived relations to be defined.

Example-3: solving type 1 cross-over schema mismatch. The source database models the concept of *jobs* as relations. The target database models *jobs* as a data domain *Employee.jobTitle*. The following is the resolution to this mismatch:

$D^R = \{SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector\}$

$S_{Employee} = RELmat(D^R, jobTitle)$

As shown in Figure 6, relation $S_{Employee}$ has scheme $S_{Employee}$*(id, name, salary, bonus, jobTitle)*. The data domain *jobTitle* has population D^R.

5.5 SME-2: Solving Type 2 Cross-over Schema Mismatches

Schema Mismatch Environment 2 (SME-2) is depicted in Figure 5.

Input relations. All relations defined in previous steps for deriving M.

Transformation operators: $ATTRmat$, π, σ, $\bowtie$, $rename$, pad, $deriveAttr$.

Operator $ATTRmat$ (attribute materialize) is defined as follows. Given $D^A = \{A_1, ..., A_n\}$, a group of attributes in a relation S that have identical data types, let N_A and N_V be attributes, $N_A, N_V \notin ATTR(S)$, then:

$$ATTRmat(S, D^A, N_A, N_V) = \bigcup_{i=1}^{n} pad(rename(\pi_{ATTR(S)-D^A \cup \{A_i\}}(S), A_i, ATTRname(N_V)), N_A, ATTRname(A_i))$$

The result relation has attribute set $ATTR(R) - D^A \cup \{N_A, N_V\}$. The effect of this operator is illustrated in Figure 7. $ATTRmat$ transforms a meta domain, the attribute group, into a data domain. Application of $ATTRmat$ is demonstrated in Example-4.

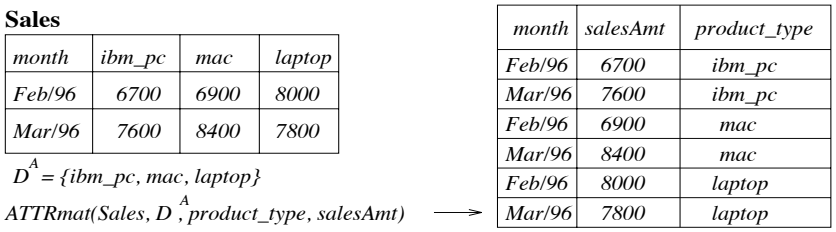


Figure 7: The $ATTRmat$ operator

Output semantic information. SME-2 allows specification of attribute groups. In the target database, the names of the attributes in an attribute group form an enumerated data domain which represents a concept that is represented as attributes in the source database.

Output relations. SME-2 allows multiple derived relations to be defined.

Example-4: Solving type 2 schema mismatch. The source database models *product_types* as attributes $ibm\_pc, mac, laptop$ in *Sales*, while the target database models it as a domain *CompanySales.product_type*. The following is the resolution for this mismatch:

$D^A = \{ibm\_pc, mac, laptop\}$, $S_{CompanySales} = ATTRmat(Sales, D^A, product\_type, salesAmt)$

As shown in Figure 7, relation $S_{CompanySales}$ has scheme *(month, salesAmt, product_type)*. The data domain *product_type* has population D^A.

5.6 RLE: Environment for Relation Linking

Relation Linking Environment, RLE, is depicted in Figure 8. The input includes relations previously defined. Derived relations can be defined using *rename*, π, σ, and $\bowtie$. A distinguished relation S_M, which contains all data domains(attributes) that corresponds to those in target relation M, must be defined. S_M is a "prototype" of M modulo data domain mismatches, and is the only relation referenced in future steps. Intuitively, relation linking explicitly "joins" relations to form a meaningful view. [Missi95] employs a universal relation approach to "complete" such joins. This technique can be "plugged" into RLE.

Example-5: relation linking. In Example-3 and 4, relations $S_{Employee}$ and $S_{CompanySales}$ are defined. Nothing is to be done in RLE in the example application.

5.7 DSE: Solving Domain Structural Mismatches

The mediator author resolves domain structural mismatches (section 5.1) in Domain Structure Environment, DSE (Figure 8).

Input relations. The distinguished output relation of RLE, S_M.

Transformation operators. None.

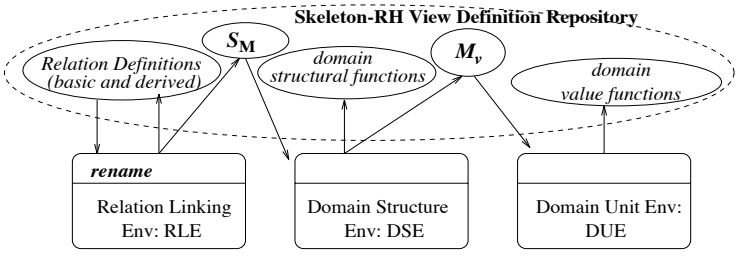


Figure 8: Environments supporting Step 4, 5 and 6 of Homogenization.

Output semantic information. DSE captures *domain structural functions*. Consider $A \in ATTR(M)$. The corresponding domain(s) of A in S_M might be (a) an attribute of the same data type as A; or (b) an attribute of a different data type from A; or (c) several related attributes that

together correspond to A. Let $ATTR(M)$=$\{A_1, ..., A_m\}$. To derive each of these attributes from attributes of S_M, the DSE requires the following to be specified for each $A_i (i = 1, m)$:

1. $L_i = \{A^S_{i,1}, ..., A^S_{i,k}\}$, attributes in S_M that correspond to A_i. By default $L_i = \{S_M.A_i\}$. If $A_i \notin ATTR(S_M)$, L_i must be given explicitly.

2. *domain structural function (DSF)*, f^s_i, that maps L_i to A_i. f^s_i is an identity function by default.

For case (a) described above, the DSF is an identity function. For cases (b) and (c), the DSF is an arbitrary function. Inverses of DSFs, if they exist, must also be specified.

Output relations. No relation is explicitly derived. However, by defining DSFs for all attributes in M, the mediator author implicitly defines the following relation:

$$M_v = \pi_{A_1,...,A_m}(deriveAttr(S_M, L_1, A_1, f^s_1, ..., L_m, A_m, f^s_m))$$

The scheme of M_v is mostly identical to that of the target relation M except that an attribute in M_v may have a different unit of measure from the attribute in M that has the same name.

Example-6: solving domain structural mismatch. In relation $S_{Employee}$ defined in Example-3, there is (base)*salary* and *bonus*. In target relation *Employee*, we expect *salary* to include the total income of an employee. To resolve this mismatch, the following is specified:

$$L_{Employee.salary} = \{salary, bonus\},\ f^s_{Employee.salary}(s, b) = s + b$$

Relation $S_{CompanySales}$ defined in Example-4 is identical in scheme to the target relation *CompanySales*. All DSFs in *CompanySales* are identity functions. Relations $Employee_v$ and $CompanySales_v$ can be inferred as described earlier.

5.8 DUE: Solving Domain Unit/Population Mismatches

The mediator author resolves domain unit/population mismatches(section 5.1) in Domain Unit Environment, DUE (Figure 8).

Input relations. Relation M_v constructed by DSE.

Transformation operators. None.

Output semantic information. The DUE captures domain value functions. While M_v is mostly identical to the target relation M, the values for attribute $M_v.A$ may differ from that for $M.A$ due to (1) difference in unit of measurement; or (2) difference in domain population. DUE requires that for each attribute $A \in ATTR(M)$, a *domain value mapping* be specified to convert values in the domain $M_v.A$ to that in $M.A$. This mapping is by default an identity function but

can be an arbitrary function or a stored mapping table, as illustrated in Example-7 and Example-8. If a domain value mapping maps each $M_v.A$ value to a unique $M.A$ value, it is a *domain value function (DVF)*. Otherwise, there is *uncertainty* in the homogenization process. In this paper, we only consider DVFs. Inverses of DVFs, if they exist, must also be specified. They are used for efficient query processing in section 6.2.

Output relations. No relation is explicitly derived. However, by specifying DVFs for each attribute in M, the mediator author implicitly defines the following relation:

$$M = deriveAttr(M_v, M_v.A_1, M.A_1, f_1^v, ..., M_v.A_k, M.A_k, f_m^v)$$

where $ATTR(M) = \{A_1, ..., A_m\}$ and $f_i^v (i = 1, m)$ is the domain value function for attribute A_i.

Example-7: Solving domain unit mismatch. Consider relation $Employee_v$ constructed in Example-6. Attribute *salary* is derived, but its values are still in Canadian dollars. In the target database, we expect to see US dollars only. Assume 1 Canadian dollar worths r US dollars, a domain value function is defined for *Employee.salary*:

$$f^v_{Employee.salary}(s) = CNDtoUSD(s) = s \times r$$

Similarly, a domain value function can be defined for *CompanySales.salesAmt*:

$$f^v_{CompanySales.salesAmt}(s) = CNDtoUSD(s) = s \times r$$

$f^v_{salesAmt}()$ is the same as $f^v_{salary}()$; $CNDtoUSD()$ has an inverse, $USDtoCND()$. Domain value functions for attributes *id* and *name* are identity functions by default.

Example-8: Solving domain population mismatch. Consider relation $Employee_v$ in Example-6. The domain of *jobTitle* consists of {*SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector*}; while in *Employee*, this domain consists of {*System Engineer, Development Engineer, Consultant, Research Scientist, Program Manager*}. To resolve this mismatch, a domain value function must be specified for *Employee.jobTitle*. This function is a stored mapping table, *jobMap*, given in Table 1. $f^v_{Employee.jobTitle}(j) = jobMap(j)$. *jobMap* has an inverse.

source database	target database
SysAdm	System Engineer
SoftwareEngineer	Development Engineer
MarketingStaff	Consultant
ResearchStaff	Research Scientist
ProjectDirector	Program Manager

Table 1: *jobMap*: Domain Value Mapping for *Employee.jobTitle*

6 AURORA-RH Query Processor (AQP)

Assume that the source database B has been homogenized into the target database H. Let Q be a relational query against H. AQP translates this query into a set of queries over the source database, sends the queries for execution, and assembles the answer to Q using the data returned. As shown in Figure 4, AQP consists of a query execution engine, a query rewriter and a query optimizer.

6.1 AQP Query Execution Engine and QEPs

Query Execution Plans(QEPs) are expressions that involve only source relations. A QEP can be depicted as an *operation tree*. Each node in this tree is annotated with the operator name and a list of arguments. A non-leaf node of the tree is either an AURORA-RH primitive, *rename*, *pad* or *deriveAttr*, or a relational operator. The leaf nodes of the tree are *retrieve* primitives. Figures 9, 10, and 11 are QEP trees. The AQP query execution engine evaluates QEP trees bottom up.

6.2 AQP Query Rewriter

We consider mediator queries in the form of $\pi_L\sigma_p(M)$, where L is a list of attributes in M and p is a predicate. The rewriting algorithm given in this section can be adapted for join queries. Via MAT-RH, the derivation of M is captured as transformations and domain mappings in the View Definition Repository. These are used to modify Q and generate a QEP.

Algorithm. AQPrewriteQuery

Input: $Q = \pi_L\sigma_p(M)$ **Output:** A QEP for Q

1. Replace M in Q with S_M.
   Replace each $RELmat$ and $ATTRmat$ operation with its definition(section 5.4, 5.5)
   while (Q involves a *direct* or *derived* relation R)
   Replace R with its defining transformation expression.
   Replace each $RELmat$ and $ATTRmat$ operation with its definition.

2. Let $Q = \pi_L(Q')$, let $\{A_1, .., A_m\}$ be all the attributes in L whose domain value functions, $f_1^v, ..., f_m^v$, are *not* identity functions, rewrite Q as:
   $$Q = deriveAttr(\pi_L(Q'), A_1, A_1, f_1^v, ..., A_m, A_m, f_m^v)$$

3. For each attribute A involved in predicate p, if its domain value function, f^v, is not identity, replace it with $f^v(A)$.

4. Let $L = \{A_1, ..., A_n\}$. Let $f_1^s, ..., f_n^s$ be the domain structural functions for $\{A_1, ..., A_n\}$. Let $L_i(i = 1, n)$ be the list of attributes which are the arguments of f_i^s. Then do the following:

(a) Replace Q' by $deriveAttr(Q', L_1, A_1, f_1^s, ..., L_n, A_n, f_n^v)$

(b) For $i = 1$ to n do:

If f_i^s is an identity function, first, remove L_i, A_i, f_i^v from the argument list of *deriveAttr* function constructed above; second, if the argument attribute in L_i, A_i', has a different name from A_i, replace the first argument of *deriveAttr*, E' with $rename(E', A_i', ATTRname(A_i))$.

5. For each attribute A involved in predicate p, if A's domain structural function, f^s, is not an identity function, replace A with $f^s(A_1', ..., A_k')$, where $A_1', ..., A_k'$ is the argument list of f^s.

6. Repeat until no modification can be made:

For each subexpression in p that is in the form of $f(E_1)\theta f(E_2)$ or $f(E_3)\theta c$, where E_1, E_2 and E_3 are expressions, c is a constant, $\theta \in \{=, >, <\}$, and f is a function which has an inverse f^{-1}, if f is strictly monotonic or θ is "=", replace this predicate with $E_1\theta E_2$ or $E_3\theta c'$, respectively, where $c' = f^{-1}(c)$.

■

Example-9. Consider query: $\pi_{id,name,salary}\sigma_{salary>50000}\sigma_{jobTitle="DevelopmentEngineer"}(Employee)$ that retrieves the *id*, *name* and *salary* of development engineers who earn more than 50000. We rewrite this query using the above algorithm:

step 1. From the definitions of $S_{Employee}$ and $RELmat$ in Example-3 and section 5.4, we get:

$$\begin{aligned} Q = \pi_{id,name,salary}\sigma_{salary>50000}\sigma_{jobTitle="DevelopmentEngineer"}(\\ pad(retrieve(SysAdm), "jobTitle", "SysAdm") \cup \\ pad(retrieve(SoftwareEngineer), "jobTitle", "SoftwareEngineer") \cup \\ pad(retrieve(MarketingStaff), "jobTitle", "MarketingStaff") \cup \\ pad(retrieve(ResearchStaff), "jobTitle", "ResearchStaff") \cup \\ pad(retrieve(ProjectDirector), "jobTitle", "ProjectDirector")) \end{aligned}$$

steps 2 and 3. Domain value functions are defined for *salary* and *jobTitle*, as in Example-7 and Example-8. However, only *salary* is involved in a projection list. Performing steps 2 and 3 we get:

$$\begin{aligned} Q = deriveAttr(\pi_{id,name,salary}\sigma_{CNDtoUSD(salary)>50000} \\ \sigma_{jobMap(jobTitle)="DevelopmentEngineer"}(\bigcup(pad(...))), salary, "salary", CNDtoUSD()) \end{aligned}$$

steps 4 and 5. *salary* has a non-trivial domain structural function, as in Example-6. It is involved in the projection list and the predicate. Performing steps 4 and 5, we get the following:

$$\begin{aligned} Q = deriveAttr(\pi_{id,name,salary}(deriveAttr(\sigma_{CNDtoUSD(salary+bonus)>50000} \\ \sigma_{jobMap(jobTitle)="DevelopmentEngineer"}(\bigcup(pad(...))), \{salary, bonus\}, "salary", \\ f^s_{Employee.salary})), salary, "salary", CNDtoUSD()) \end{aligned}$$

step 6. $CNDtoUSD$ has an inverse $USDtoCND$, and *jobMap*(Table 1) also has an inverse. Let $C = USDtoCND(50000)$. Performing step 6, we get the final QEP given below and in Figure 9:

$$\begin{aligned} Q = deriveAttr(\pi_{id,name,salary}(deriveAttr(\sigma_{salary+bonus>C}\sigma_{jobTitle="SoftwareEngineer"} \\ (\bigcup(pad(...))), \{salary, bonus\}, "salary", f^s_{Employee.salary})), salary, "salary", CNDtoUSD()) \end{aligned}$$

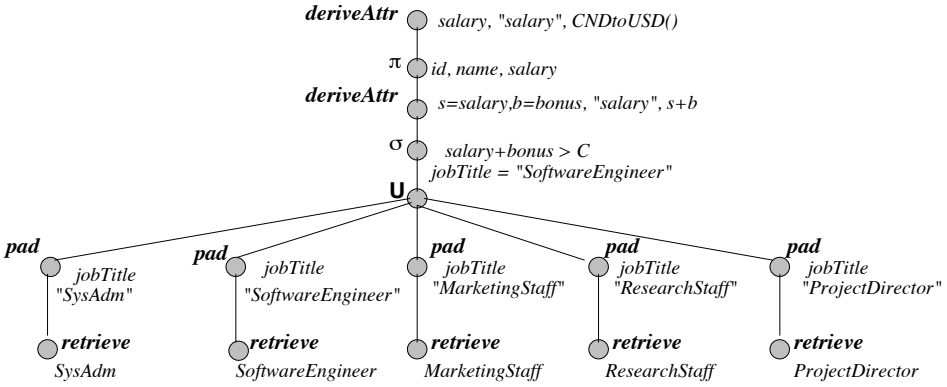


Figure 9: An QEP for Example-9.

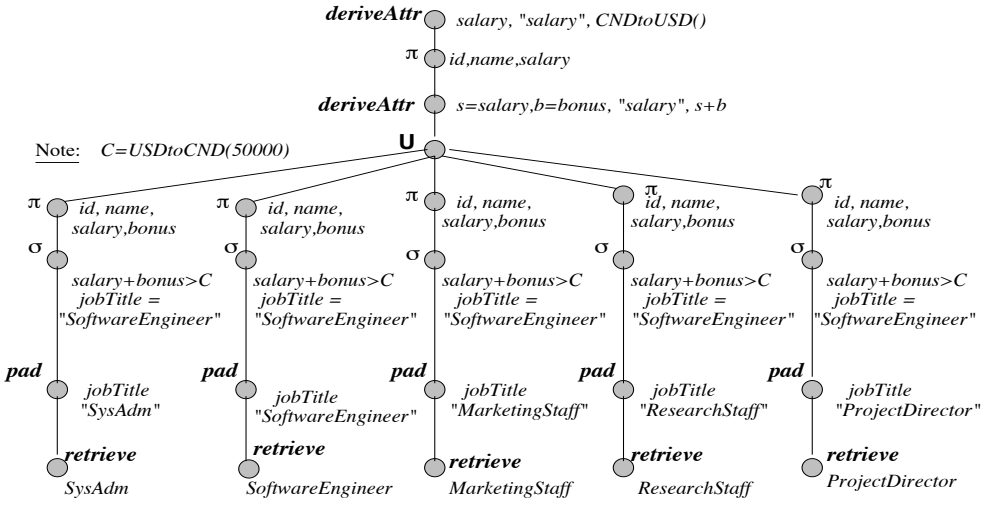


Figure 10: Transformed QEP

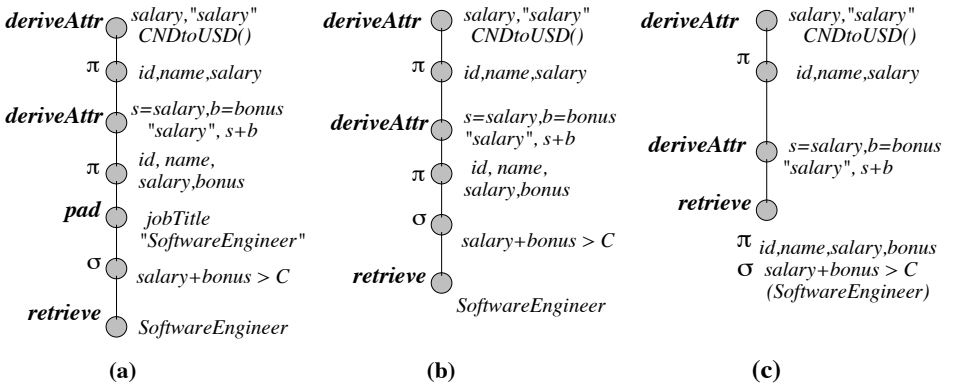


Figure 11: Transformed QEPs

6.3 AQP Query Optimization

The AQP query optimizer maximizes the number of relational operations performed by the source database so as to leverage the query optimization capability of the source and reduce the amount of data fetched. A QEP generated by the rewriter is transformed to *enlarge* the (sub)queries submitted to the source database. As *retrieve* is the only operator that submits queries, the optimizer pushes as many as possible relational operators into *retrieve*. Consider a QEP tree such as Figure 9. the query optimizer 1) pushes relational operators "across" *pad*, *rename*, and *deriveAttr* so that they move towards the leaves; and 2) pushes relational operators across *retrieve*, so that they become part of the argument (annotation) of the *retrieve* leaf. In this section, we discuss transformation rules and control strategies in AQP query optimizer.

	Transformation rules for *pad*
$T_{pad}[1]$.	$\pi_L(pad(R,N,s)) \equiv \pi_L(R)$, $L \subseteq ATTR(R)$, $N \notin ATTR(R)$.
$T_{pad}[2]$.	$\pi_L(pad(R,N,s)) \equiv pad(\pi_{L-\{N\}}(R),N,s)$, $L \subseteq \{N\} \cup ATTR(R)$, $N \in L$.
$T_{pad}[3]$.	$\sigma_p(pad(R,N,s)) \equiv pad(\sigma_{p^{N \leftarrow s}}(R),N,s)$.
$T_{pad}[4]$.	$R \bowtie_p pad(R_1,N_1,s_1) \equiv pad(R \bowtie_{p^{N_1 \leftarrow s_1}} R_1,N_1,s_1)$.
$T_{pad}[5]$.	$pad(R_1,N_1,s_1) \bowtie_p pad(R_2,N_2,s_2) \equiv pad(pad(R_1 \bowtie_{p^{N_1 \leftarrow s_1,N_2 \leftarrow s_2}} R_2,N_1,s_1),N_2,s_2)$.
	Transformation rules for *rename*
$T_{rename}[1]$.	$\pi_L(rename(R,A,N)) \equiv \pi_L(R)$, $L \subseteq ATTR(R)$, $N \notin ATTR(R)$.
$T_{rename}[2]$.	$\pi_L(rename(R,A,N)) \equiv rename(\pi_{L_{N \leftarrow A}}(R),A,N)$, $L \subseteq \{N\} \cup ATTR(R) - \{A\}$.
$T_{rename}[3]$.	$\sigma_p(rename(R,A,N)) \equiv rename(\sigma_{p^{N \leftarrow A}}(R),A,N)$.
$T_{rename}[4]$.	$R \bowtie_p rename(R_1,A_1,N_1) \equiv rename(R \bowtie_{p^{N_1 \leftarrow A_1}} R_1,A_1,N_1)$.
$T_{rename}[5]$.	$rename(R_1,A_1,N_1) \bowtie_p rename(R_2,A_2,N_2)$ $\equiv rename(rename(R_1 \bowtie_{p^{N_1 \leftarrow A_1,N_2 \leftarrow A_2}} R_2,A_1,N_1),A_2,N_2)$.
	Transformation rules for *deriveAttr*
$T_{deriveAttr}[1]$.	$\pi_L(deriveAttr(R,L_1,N,f)) \equiv \pi_L(R)$, $L \subseteq ATTR(R)$, $N \notin ATTR(R)$.
$T_{deriveAttr}[2]$.	$\pi_L(deriveAttr(R,L_1,N,f)) \equiv \pi_L(deriveAttr(\pi_{L-\{N\} \cup L_1}(R),L_1,N,f))$, $L \subseteq \{N\} \cup ATTR(R), N \in L$.
$T_{deriveAttr}[3]$.	$\sigma_p(deriveAttr(R,L,N,f)) \equiv deriveAttr(\sigma_{p^{N \leftarrow f(L)}}(R),L,N,f)$.
$T_{deriveAttr}[4]$.	$R \bowtie_p deriveAttr(R_1,L_1,N_1,f_1) \equiv deriveAttr(R \bowtie_{p^{N_1 \leftarrow f(L_1)}} R_1,L_1,N_1,f_1)$, $ATTR(R) \cap ATTR(R_1) = \phi$, $N_1 \notin ATTR(R)$.
$T_{deriveAttr}[5]$.	$deriveAttr(R_1,L_1,N_1,f_1) \bowtie_p deriveAttr(R_2,L_2,N_2,f_2)$ $\equiv deriveAttr(deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1),N_2 \leftarrow f_2(L_2)}} R_2,L_1,N_1,f_1),L_2,N_2,f_2)$, $ATTR(R_2) = \phi$, $N_1 \notin ATTR(R_2)$, $N_2 \notin ATTR(R_1)$, $N_1 \neq N_2$, $N_2 \notin L_2$.

Table 2: Transformation Rules for *pad, rename* and *deriveAttr*

Table 2 gives transformation rules for pushing relational operators across primitives *pad*, *rename* and *deriveAttr*, respectively. For simplicity, the rules for *deriveAttr* are given only for cases where there is one derived attribute. Extensions can be easily made to allow cases where multiple derived attributes are present. These rules are mostly self-explanatory. Proof of rules for *deriveAttr* is given in Appendix B. In Table 2, $p^{N \leftarrow X}$ denotes the predicate obtained from p by substituting all

appearances of N with X. If p does not involve N, $p^{N \leftarrow X} = p$. $L_{N \leftarrow A}$ denotes the list of attributes obtained from L by replacing attribute N with A. If L does not involve N, $L_{N \leftarrow A} = L$.

A relational operator can be pushed into *retrieve* if it is acceptable to the source query facility. As most relational query languages do not allow user-defined functions, selections whose predicates involve functions that are not built-in in the source query facility do not exchange with *retrieve*. This potentially increases the amount of data fetched from the source. In Algorithm **AQPrewrite-Query** step 6, inverses of domain mapping functions are used to eliminate such selection predicates.

A *control strategy* selects the next transformation rule to be applied. Currently AQP pushes relational operators across AURORA-RH primitives towards the leaves using the rules in Table 2, whenever and wherever applicable, in no particular sequence, until no rules are applicable. More sophisticated strategies to speed up optimization are a topic for future research.

Example-10: Optimization of Figure 9. Use rule $T_{deriveAttr}[2]$ to exchange π with the *deriveAttr* under it, the π argument list is now $id, name, salary, bonus$. Exchange $\cup$ with this π and σ, we get Figure 10. Push σ across *pad* operators using rule $T_{pad}[3]$, many of the *pad* subtrees become ϕ, e.g.

$$\sigma_{jobTitle=\text{“}SoftwareEngineer\text{”}}(pad(retrieve(SysAdm), jobTitle, \text{“}SysAdm\text{”}))$$

$$= \sigma_{\text{“}SysAdm\text{”}=\text{“}SoftwareEngineer\text{”}}(pad(retrieve(SysAdm), jobTitle, \text{“}SysAdm\text{”})) = \phi$$

Trim the empty branches from Figure 10 to get Figure 11 (a). Use rule $T_{pad}[1]$ to push π across *pad* to obtain Figure 11 (b). Finally, push the relational operators across *retrieve*. Since the selection predicate involves a function +, known to the source database, both π and σ exchange with *retrieve* and we obtain Figure 11 (c).

7 Conclusion and Future Work

We have described AURORA, a project that aims at developing practical techniques for building high performance mediators. Our contributions are summarized as follows. First, we have proposed a 2-tier, homogenization followed by integration, mediator architecture (Figure 1). The key feature of this architecture is to use specialized mediators to perform specific types of mediations. This specialization allows us to isolate a collection of technical issues in constructing mediators, such as mediation methodology, view expression, and query processing. Second, we have designed a mediator development environment, consisting of a collection of workbenches that assist the mediator authors in constructing various types of mediators. We believe that such environments are crucial for making the mediator framework practical. Third, we have developed a suite of techniques for AURORA-RH, a workbench for building relational homogenization mediators. These techniques

include: 1) a homogenization methodology, to be employed by the mediator author and mandated by the workbench; 2) a set of view operators that extend the power of relational algebra for expressing homogenizing views; 3) a query rewrite algorithm; and 4) a set of transformation rules that facilitate logical optimization of the mediator queries.

Currently we are implementing AURORA-RH. The major implementation issue is the design of an interactive user interface. To use AURORA-RH, the mediator author must have knowledge about the source and target schemas, the mismatches, and the resolutions. A friendly user interface should provide clear presentations of the schemas and guide the mediator author to perform correct and complete homogenization. In particular, this interface should help the user to: 1) follow the homogenization methodology (section 5.2); 2) correctly use the AURORA-RH transformations, such as *RELmat* and *ATTRmat*; 3) define domain structural/value functions with appropriate signatures and valid implementations; and 4) browse the source and target schema, and current transformations and mappings captured in the view definition repository. A properly designed GUI would make AURORA-RH a practical tool for constructing mediators.

Our ultimate goal is to build a collection of workbenches, AURORA-RI, AURORA-OH, and AURORA-OI (Figure 2 (b)). These workbenches will be of similar forms as AURORA-RH but require different suites of techniques. For instance, AURORA-RI may require different mediation methodology and view expression operators. Different query rewriting algorithm and transformation rules must be developed in similar fashion as in AURORA-RH.

References

[Carey94] M.J.Carey, *Towards Heterogeneous Multimedia Information Systems: the Garlic Approach*, Tech Report RJ 9911, IBM Almaden Research Center, 1994.

[DH84] U.Dayal, H-Y Hwang, *"View definition and generalization for database integration in a multidatabase system"*, IEEE Trans. on Software Engineering, vol SE-10, No.6, Nov 1984, pp 628-645.

[DKS92] W.Du, R.Krishnamurthy, M.Shan, *"Query Optimization in a Heterogeneous DBMS"*, VLDB 1992, pp277-291.

[Goh95] C.H.Goh, M.E.Madnick and M.D.Siegel, *"Ontologies, Context, and mediation: Representing and Reasoning about Semantic Conflicts in Heterogeneous and Autonomous Systems"*, MIT Sloan School of Management Working Paper 3848, 1995.

[KLK91] R.Krishnamurthy, W.Litwin, W.Kent, *"Language Features for Interoperability of Databases with Schematic Discrepancies"*, SIGMOD 1991, pp 40-49.

[Kent91] W.Kent, *"Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language"*, VLDB 1991, pp 147-160.

[Kim93] Won Kim et al, *"On Resolving Schematic Heterogeneity in Multidatabase Systems"*, *Distributed and Parallel Databases*, 1(3), 1993, pp 251-279.

[LOG92] H.Lu, B.C.Ooi, C.H.Goh, *"On Global Multidatabase Query Optimization"*, SIGMOD Record, 20, 4 1992, pp 6-11.

[LPL96] L.Liu, C.Pu, Y.Lee, *"An Adaptive Approach to Query Mediation Across Heterogeneous Information Sources"*, Int. Conf. on Cooperative Information Systems (CoopIS), June 1996, pp 144-156.

[Meng93] W.Meng et al, *"Construction of Relational Front-end for Object-Oriented Database Systems"*, IEEE International Conference on Data Engineering 1993, pp476-483.

[Mot87] A.Motro, *"Superviews: Virtual Integration of Multiple Databases"*, IEEE Trans. on Software Engineering, vol SE-13, No.7, July 1987, pp 785-798.

[Missi95] P.Missier and Mark Rusinkiewicz, *"Extending a multidatabase manipulation language to resolve schema and data conflicts"*, IFIP TC-2 Working Conference on Data Semantics (DS-6), May 1995, Stone Mountain, Georgia.

[PGU96] Y.Papakonstantinou, H.Garcia-Molina, J.Ullman, *"MedMaker: A Mediation System Based on Declarative Specifications"*, IEEE International Conference on Data Engineering 1996.

[Qian94] Xiaolei Qian and Teresa F.Lunt, *"Semantic Interoperation: A Query Mediation Approach*, Tech report SRI-CSL-94-02, Computer Science Laboratory, SRI International, April 1994.

[Subr] V.S.Subrahmanian et al, *"HERMES: Heterogeneous Reasoning and Mediator System"*, unpublished document, University of Maryland.

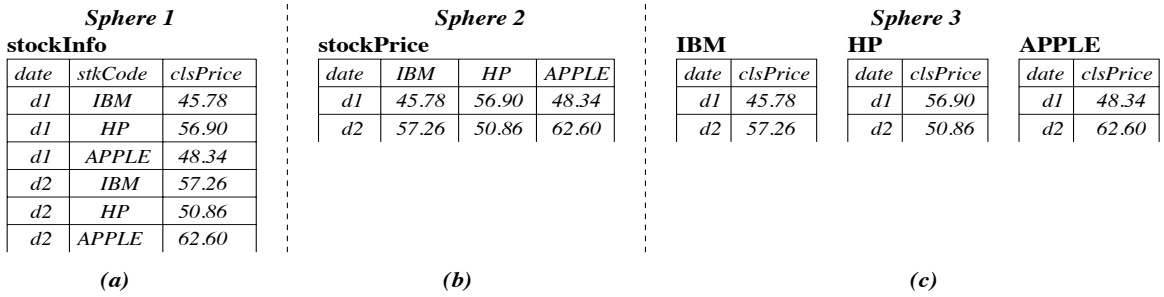
[TRV95] A.Tomasic, L.Raschid, P.Valduriez, *"Scaling Heterogeneous Databases and the Design of Disco"*, Tech Report 2704, November 1995, INRIA.

[Wied92] Gio Wiederhold, *"Mediator Architecture of Future Information Systems"*, IEEE Computer, pp 38-49. March 1992.

[Yu95] C.Yu et al, *"Translation of Object-Oriented Queries to Relational Queries"*, IEEE International Conference on Data Engineering 1995, pp90-97.

Appendix A: Cross-Over Schema Mismatches

Figure 12 shows 3 databases. To transform (b) and (c) to (a), we need to resolve type 2 and type 1 cross-over schema mismatches, respectively. However, if (b) or (c) is chosen to be the target database, one has to resolve other types of cross-over schema mismatches as pointed out in section 5.1. In this Appendix, we demonstrate resolutions for these mismatches using relational operators and AURORA operators.



Sphere 1

stockInfo

date	*stkCode*	*clsPrice*
d1	*IBM*	*45.78*
d1	*HP*	*56.90*
d1	*APPLE*	*48.34*
d2	*IBM*	*57.26*
d2	*HP*	*50.86*
d2	*APPLE*	*62.60*

(a)

Sphere 2

stockPrice

date	*IBM*	*HP*	*APPLE*
d1	*45.78*	*56.90*	*48.34*
d2	*57.26*	*50.86*	*62.60*

(b)

Sphere 3

IBM

date	*clsPrice*
d1	*45.78*
d2	*57.26*

HP

date	*clsPrice*
d1	*56.90*
d2	*50.86*

APPLE

date	*clsPrice*
d1	*48.34*
d2	*62.60*

(c)

Figure 12: Example *stocks*

(a) $\rightarrow$ (c): *stkCode* is represented as data domain in (a) but as relations in (c). This mismatch can be resolved as follows:

$$\begin{aligned} IBM &= \pi_{date,clsPrice}(\sigma_{stkCode=\text{“}IBM\text{”}}(retrieve(stockInfo))) \\ HP &= \pi_{date,clsPrice}(\sigma_{stkCode=\text{“}HP\text{”}}(retrieve(stockInfo))) \\ APPLE &= \pi_{date,clsPrice}(\sigma_{stkCode=\text{“}APPLE\text{”}}(retrieve(stockInfo))) \end{aligned}$$

(b) $\rightarrow$ (c): *stkCode* is represented as attributes in (b) but as relations in (c). This mismatch can be resolved as follows:

$$\begin{aligned} IBM &= rename(\pi_{date,IBM}(retrieve(stockPrice)), IBM, \text{“}clsPrice\text{”}) \\ HP &= rename(\pi_{date,HP}(retrieve(stockPrice)), HP, \text{“}clsPrice) \\ APPLE &= rename(\pi_{date,APPLE}(retrieve(stockPrice)), APPLE, \text{“}clsPrice) \end{aligned}$$

(c) $\rightarrow$ (b): *stkCode* is represented as relations in (c) but as attributes in (b). This mismatch can be resolved as follows

$$\begin{aligned} stockPrice = {} & rename(retrieve(IBM), clsPrice, IBM) \bowtie \\ & rename(retrieve(HP), clsPrice, HP) \bowtie \\ & rename(retrieve(APPLE), clsPrice, APPLE) \end{aligned}$$

(a) $\rightarrow$ (b): This case can be resolved by transforming (a) to (c), and (c) to (b).

Appendix B: Proof of AURORA-RH Transformation Rules

We prove transformation rules for operator *deriveAttr* given in Table 2. For the purpose of this section, we do not consider the data type of an attribute. We assume that two attributes are the same if and only if they have the same name. In the rest of this section, we use "LHS" to refer to the left hand side of an equation, and "RHS" for the right hand side. To prove the equivalence, we prove that for any tuple t, $t \in LHS$ if and only if $t \in RHS$.

Given a relation R, consider relation $R' = deriveAttr(R, L, N, f)$. Let $X = ATTR(R)$. By definition, for a tuple $t_r \in R$, there exists a tuple $t_d \in R'$, such that $t_d[X - \{N\}] = t_r[X - \{N\}]$, and $t_d[N] = f(t_r[L])$. We say that t_d is the **image** of t_r due to $deriveAttr(R, L, N, f)$, and t_r is the **origin** of t_d due to $deriveAttr(R, L, N, f)$.

Property 1. *Let $R' = deriveAttr(R, L, N, f)$. Each tuple in R' has an origin in R; each tuple in R has an image in R'.*

$\underline{\text{T}_{deriveAttr}[1]}$ $\pi_L(deriveAttr(R, L_1, N, f)) \equiv \pi_L(R)$, $L \subseteq ATTR(R)$, $N \notin ATTR(R)$.

Proof: Let $X = ATTR(R)$. Consider any tuple t in LHS, $t \in \pi_L(deriveAttr(R, L_1, N, f))$, by definition of π, there exists $t_1 \in deriveAttr(R, L_1, N, f)$, such that:

$$t = t_1[L] \tag{1}$$

Let t_2 be the origin of t_1 due to $deriveAttr(R, L_1, N, f)$. By definition, $t_2 \in R$ and $t_1[X - \{N\}] = t_2[X - \{N\}]$. Since $N \notin L$, $L \subseteq (X - \{N\})$, we get $t_1[L] = t_2[L]$. Combining this with 1, we get $t = t_2[L]$. Since $t_2[L] \in \pi_L(R)$, $t \in \pi_L(R)$, i.e. $t \in RHS$.

Consider any tuple t in RHS, $t \in \pi_L(R)$. By definition of π, there exists $t_1 \in R$, such that

$$t = t_1[L] \tag{2}$$

Let $t_2 \in deriveAttr(R, L_1, N, f)$ be the image of t_1 due to $deriveAttr(R, L_1, N, f)$. By definition, $t_1[X - \{N\}] = t_2[X - \{N\}]$. Since $N \notin L$, $L \subseteq (X - \{N\})$, we get $t_1[L] = t_2[L]$. Combining this with 2, we get $t = t_2[L]$. Since $t_2 \in deriveAttr(R, L_1, N, f)$, $t_2 \in \pi_L(deriveAttr(R, L_1, N, f))$, i.e. $t \in LHS$. ∎

$\underline{\text{T}_{deriveAttr}[2]}$ $\pi_L(deriveAttr(R, L_1, N, f)) \equiv \pi_L(deriveAttr(\pi_{L-\{N\} \cup L_1}(R), L_1, N, f))$,

$L \subseteq \{N\} \cup ATTR(R), N \in L.$

Proof: Let $X = ATTR(R)$. Consider any t in LHS, $t \in \pi_L(deriveAttr(R, L_1, N, f))$. By definition of π, there exists $t_1 \in deriveAttr(R, L_1, N, f)$, such that

$$t = t_1[L] \tag{3}$$

Let $t_2 \in R$ be the origin of t_1, by definition, we have:

$$t_1[X - \{N\}] = t_2[X - \{N\}], \;\; t_1[N] = f(t_2[L_1]) \tag{4}$$

Since $(L - \{N\}) \subseteq (X - \{N\})$, we also have

$$t_1[L - \{N\}] = t_2[L - \{N\}] \tag{5}$$

Let $t_3 = t_2[L - \{N\} \cup L_1]$, $t_3 \in \pi_{L-\{N\}\cup L_1}(R)$. Obviously we have the following:

$$t_2[L - \{N\}] = t_3[L - \{N\}], \;\; t_2[L_1] = t_3[L_1] \tag{6}$$

Let t_4 be the image of t_3 due to $deriveAttr(\pi_{L-\{N\}\cup L_1}(R), L_1, N, f)$. By definition, t_4 satisfies the following:

$$t_3[L - \{N\} \cup L_1 - \{N\}] = t_4[L - \{N\} \cup L_1 - \{N\}], \;\; t_4[N] = f(t_3[L_1]) \tag{7}$$

Since $(L - \{N\}) \subseteq (L - \{N\} \cup L_1 - \{N\})$, we have $t_3[L - \{N\}] = t_4[L - \{N\}]$. Combine this with 6 and 5, we get:

$$t_1[L - \{N\}] = t_4[L - \{N\}] \tag{8}$$

Combine 7 with 6 and 4, we get

$$t_1[N] = t_4[N] \tag{9}$$

From 8, 9 and 3, we get $t = t_1[L] = t_4[L]$. Since $t_4 \in deriveAttr(\pi_{L-\{N\}\cup L_1}(R), L_1, N, f)$, we have $t \in \pi_L(deriveAttr(\pi_{L-\{N\}\cup L_1}(R), L_1, N, f))$, i.e. $t \in RHS$.

Consider any $t \in RHS$, $t \in \pi_L(deriveAttr(\pi_{L-\{N\}\cup L_1}(R), L_1, N, f))$. By definition of π, there exists $t_1 \in deriveAttr(\pi_{L-\{N\}\cup L_1}(R), L_1, N, f)$, such that:

$$t = t_1[L] \tag{10}$$

Let $t_2 \in \pi_{L-\{N\}\cup L_1}(R)$ be the origin of t_1, then t_2 satisfies the following:

$$t_1[L - \{N\} \cup L_1 - \{N\}] = t_2[L - \{N\} \cup L_1 - \{N\}], \;\; t_1[N] = f(t_2[L_1]) \tag{11}$$

Since $(L - \{N\}) \subseteq (L - \{N\} \cup L_1 - \{N\})$, we also have:

$$t_1[L - \{N\}] = t_2[L - \{N\}] \tag{12}$$

By definition of π, there exists $t_3 \in R$, $t_2 = t_3[L - \{N\} \cup L_1]$. Obviously we have

$$t_2[L - \{N\}] = t_3[L - \{N\}], \;\; t_2[L_1] = t_3[L_1] \tag{13}$$

Let t_4 be the image of t_3 due to $deriveAttr(R, L_1, N, f)$, then t_4 satisfies the following:

$$t_3[X - \{N\}] = t_4[X - \{N\}], \;\; t_4[N] = f(t_3[L_1]) \tag{14}$$

Since $(L - \{N\}) \subseteq (X - \{N\})$, we have

$$t_3[L - \{N\}] = t_4[L - \{N\}] \tag{15}$$

From 15, 13 and 12, we have

$$t_1[L - \{N\}] = t_4[L - \{N\}] \tag{16}$$

From 14, 13 and 11, we have

$$t_1[N] = t_4[N] \tag{17}$$

From 10, 16 and 17, we have $t = t_1[L] = t_4[L]$. Since $t_4 \in deriveAttr(R, L_1, N, f)$, we have:

$$t \in \pi_L(deriveAttr(R, L_1, N, f))$$

that is, $t \in LHS$. ∎

T$_{deriveAttr}$[3] $\sigma_p(deriveAttr(R, L, N, f)) \equiv deriveAttr(\sigma_{p^{N \leftarrow f(L)}}(R), L, N, f)$.

Proof: Let $X = ATTR(R)$. Let p be in the form of $p(A_1, ..., A_k)$ where $A_i \in X \cup \{N\}$ $(1 \leq i \leq k)$. Consider any tuple t in LHS, $t \in \sigma_p(deriveAttr(R, L, N, f))$. By definition of σ, $t \in deriveAttr(R, L, N, f)$ and we also have:

$$p(t[A_1], ..., t[A_k]) = true \tag{18}$$

Let $t_1 \in R$ be the origin of t, by definition, we have

$$t[X - \{N\}] = t_1[X - \{N\}], \;\; t[N] = f(t_1[L]) \tag{19}$$

If there exists no i, $(1 \leq i \leq k)$, $A_i = N$, i.e. p does not involve attribute N, from 19 and 18, we have

$$p(t_1[A_1], ..., t_1[A_k]) = true$$

i.e. $t_1 \in \sigma_p(R)$. Otherwise, without loss of generality, assume $A_k = N$, from 19 we have:

$$p(t_1[A_1], ..., t_1[A_{k-1}], t[N]) = true$$

Since $t[N] = f(t_1[L])$, we have

$$p(t_1[A_1], ..., t_1[A_{k-1}], f(t[L])) = true$$

In both cases, we have $t_1 \in \sigma_{p^{N \leftarrow f(L)}}(R)$. Let $t_2 \in deriveAttr(\sigma_{p^{N \leftarrow f(L)}}(R), L, N, f)$ be the image of t_1, then we have

$$t_1[X - \{N\}] = t_2[X - \{N\}], \;\; t_2[N] = f(t_1[L]) \tag{20}$$

From 20 and 19, we have $t = t_2$, since $t_2 \in deriveAttr(\sigma_{p^{N \leftarrow f(L)}}(R), L, N, f)$, we have:

$$t \in deriveAttr(\sigma_{p^{N \leftarrow f(L)}}(R), L, N, f)$$

that is, $t \in RHS$.

Consider any tuple from the RHS, $t \in deriveAttr(\sigma_{p^{N \leftarrow f(L)}}(R), L, N, f)$. Let $t_1 \in \sigma_{p^{N \leftarrow f(L)}}(R)$ be the origin of t. t_1 satisfies the following:

$$t[X - \{N\}] = t_1[X - \{N\}], \;\; t[N] = f(t_1[L]) \tag{21}$$

Since $t_1 \in \sigma_{p^{N \leftarrow f(L)}}(R)$, $t_1 \in R$. Let t_2 be the image of t_1 due to $deriveAttr(R, L, N, f)$, then t_2 satisfies the following:

$$t_1[X - \{N\}] = t_2[X - \{N\}], \;\; t_2[N] = f(t_1[L]) \tag{22}$$

From 22 and 21 we have

$$t = t_2 \tag{23}$$

Now consider the fact that $t_1 \in \sigma_{p^{N \leftarrow f(L)}}(R)$. If there exists no i, $(1 \leq i \leq k)$, $A_i = N$, i.e. p does not involve attribute N, we have

$$p(t_1[A_1], ..., t_1[A_k]) = true$$

Otherwise, without loss of generality, assume $A_k = N$, we have:

$$p(t_1[A_1], ..., t_1[A_{k-1}], f(t_1[L])) = true$$

From 22, we can immediately derive for both cases:

$$p(t_2[A_1], ..., t_2[A_k]) = true$$

Combine the above with the fact that $t_2 \in deriveAttr(R, L, N, f)$, we have

$$t_2 \in \sigma_p(deriveAttr(R, L, N, f))$$

i.e. $t_2 \in LHS$. From 23, we get $t \in LHS$. ∎

T$_{deriveAttr}$[4] $R \bowtie_p deriveAttr(R_1, L_1, N_1, f_1) \equiv deriveAttr(R \bowtie_{p^{N_1 \leftarrow f(L_1)}} R_1, L_1, N_1, f_1)$,
$ATTR(R) \cap ATTR(R_1) = \phi$, $N_1 \notin ATTR(R)$.

Proof: Let $X = ATTR(R)$. Let $X_1 = ATTR(R_1)$. Let p be in the form of

$$p(B_1, ..., B_n, A_1, ..., A_k)$$

where $B_i \in X (1 \leq i \leq n)$, and $A_i \in X_1 \cup \{N_1\}$ $(1 \leq i \leq k)$.

Consider any t from the $LHS, t \in R \bowtie_p deriveAttr(R_1, L_1, N_1, f_1)$. Consider that $X \cap X_1 = \phi$, by definition of the $\bowtie$ operator, there exists $t_1 \in R$, $t_2 \in deriveAttr(R_1, L_1, N_1, f_1)$, such that:

$$t[X] = t_1[X], t[X_1 \cup \{N_1\}] = t_2[X_1 \cup \{N_1\}], p(t_1[B_1], ..., t_1[B_n], t_2[A_1], ..., t_2[A_k]) = true \quad (24)$$

Let $t_3 \in R_1$ be the origin of t_2 due to deriveAttr (R_1, L_1, N_1, f_1), then we have:

$$t_2[X_1 - \{N_1\}] = t_3[X_1 - \{N_1\}], \;\; t_2[N_1] = f_1(t_3[L_1]) \quad (25)$$

Since $N_1 \notin X$, there exists no j, $(1 \leq j \leq n)$, such that $B_j = N_1$. If there exists no i, $(1 \leq i \leq k)$ such that $A_i = N_1$, i.e. p does not involve attribute N_1, we have the following from 25 and 24:

$$p(t_1[B_1], ..., t_1[B_n], t_3[A_1], ..., t_3[A_k]) = true \quad (26)$$

Otherwise, without loss of generality, assume $A_k = N_1$, from 25 and 24, we have:

$$p(t_1[B_1], ..., t_1[B_n], t_3[A_1], ..., f_1(t_3[L_1])) = true \quad (27)$$

Let $t_4 \in R \times R_1$ be the concatenation of t_1 and t_3, i.e.

$$t_1[X] = t_4[X], t_3[X_1] = t_4[X_1] \quad (28)$$

From 27 and 26 we get:

$$t_4 \in R \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1$$

Let $t_5 \in deriveAttr(R \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1, L_1, N_1, f_1)$ be the image of t_4. By definition:

$$t_4[X \cup X_1 - \{N_1\}] = t_5[X \cup X_1 - \{N_1\}],\ t_5[N_1] = f_1(t_4[L_1]) \tag{29}$$

Consider $N_1 \notin X$, $L_1 \subseteq X_1$, from 29, 28, 25, and 24, we have:

$$t = t_5$$

Since $t_5 \in deriveAttr(R \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1, L_1, N_1, f_1)$, we have:

$$t \in deriveAttri \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1, L_1, N_1, f_1)$$

i.e. $t \in RHS$.

Consider any $t \in RHS, t \in deriveAttr(R \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1, L_1, N_1, f_1)$. Let $t_1 \in R \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1$ be the origin of t due to $deriveAttr(R \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1, L_1, N_1, f_1)$. Then we have:

$$t[X \cup X_1 - \{N_1\}] = t_1[X \cup X_1 - \{N_1\}],\ t[N_1] = f_1(t_1[L_1]) \tag{30}$$

Since $t_1 \in R \bowtie_{p^{N_1} \leftarrow f_1(L_1)} R_1$, there exist $t_2 \in R$ and $t_3 \in R_1$, such that

$$t_1[X] = t_2[X], t_1[X_1] = t_3[X_1] \tag{31}$$

Since $N_1 \notin X$, there exists no j, $(1 \leq j \leq n)$, such that $B_j = N_1$. If there exists no i, $(1 \leq i \leq k)$ and $A_i = N_1$, i.e. p does not involve attribute N_1, we have

$$p(t_2[B_1], ..., t_2[B_n], t_3[A_1], ..., t_3[A_k]) = true \tag{32}$$

Otherwise, without loss of generality, assume $A_k = N_1$, then we have:

$$p(t_2[B_1], ..., t_2[B_n], t_3[A_1], ..., f(t_3[L_1])) = true \tag{33}$$

Let $t_4 \in deriveAttr(R_1, L_1, N_1, f_1)$ be the image of t_3 due to $deriveAttr(R_1, L_1, N_1, f_1)$, then we have:

$$t_3[X_1 - \{N_1\}] = t_4[X_1 - \{N_1\}],\ t_4[N_1] = f_1(t_3[L_1]) \tag{34}$$

Let $t_5 \in R \times deriveAttr(R_1, L_1, N_1, f_1)$ be the concatenation of t_2 and t_4, then we have:

$$t_5[X] = t_2[X],\ t_5[X_1 \cup \{N_1\}] = t_4[X_1 \cup \{N_1\}] \tag{35}$$

From 35, 34, 33 and 32, we have

$$t_5 \in R \bowtie_p deriveAttr(R_1, L_1, N_1, f_1)$$

From 35, 31, 34 and 30, we have:

$$t = t_5$$

i.e. $t \in R \bowtie_p deriveAttr(R_1, L_1, N_1, f_1)$, i.e. $t \in LHS$. ∎

T$_{deriveAttr}$[5] $deriveAttr(R_1, L_1, N_1, f_1) \bowtie_p deriveAttr(R_2, L_2, N_2, f_2) \equiv$
$deriveAttr(deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2, L_1, N_1, f_1), L_2, N_2, f_2),$
$ATTR(R_1) \cap ATTR(R_2) = \phi$, $N_1 \notin ATTR(R_2)$, $N_2 \notin ATTR(R_1)$,
$N_1 \neq N_2$, $N_2 \notin L_2$.

Proof: Let $X_1 = ATTR(R_1)$, $X_2 = ATTR(R_2)$. Also let p be in the form of:

$$p(A_1, ..., A_k, B_1, ..., B_l)$$

where $A_i \in X_1 \cup \{N_1\}$ $(1 \leq i \leq k)$, $B_j \in X_2 \cup \{N_2\}$ $(1 \leq j \leq l)$.

Consider any $t \in deriveAttr(R_1, L_1, N_1, f_1) \bowtie_p deriveAttr(R_2, L_2, N_2, f_2)$. There exists $t_1 \in deriveAttr(R_1, L_1, N_1, f_1)$, $t_2 \in deriveAttr(R_2, L_2, N_2, f_2)$, such that:

$$t[X_1 \cup \{N_1\}] = t_1[X_1 \cup \{N_1\}], t[X_2 \cup \{N_2\}] = t_2[X_2 \cup \{N_2\}] \tag{36}$$

Moreover, we have:

$$p(t_1[A_1], ..., t_1[A_k], t_2[B_1], ..., t_2[B_l]) = true \tag{37}$$

Let $t_3 \in R_1$ be the origin of t_1 due to $deriveAttr(R_1, L_1, N_1, f_1)$, then we have:

$$t_1[X_1 - \{N_1\}] = t_3[X_1 - \{N_1\}], t_1[N_1] = f_1(t_3[L_1]) \tag{38}$$

Let $t_4 \in R_2$ be the origin of t_2 due to $deriveAttr(R_2, L_2, N_2, f_2)$, then we have:

$$t_2[X_2 - \{N_2\}] = t_4[X_2 - \{N_2\}], t_2[N_2] = f_2(t_4[L_2]) \tag{39}$$

Let $t_5 \in R_1 \times R_2$ be the concatenation of t_3 and t_4, i.e:

$$t_5[X_1] = t_3[X_1], t_5[X_2] = t_4[X_2] \tag{40}$$

From 40, 39, and 38, we get:

$$t_1[N_1] = f_1(t_5[L_1]), t_2[N_2] = f_2(t_5[L_2]) \tag{41}$$

From 41, 40, 39, 38 and 37, we can easily infer:

$$t_5 \in R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2 \tag{42}$$

Let t_6 be the image of t_5 due to $deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2, L_1, N_1, f_1)$, we have:

$$t_5[X_1 \cup X_2 - \{N_1\}] = t_6[X_1 \cup X_2 - \{N_1\}], t_6[N_1] = f_1(t_5[L_1]) \tag{43}$$

Let $t_7 \in deriveAttr(deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2, L_1, N_1, f_1), L_2, N_2, f_2)$ be the image of t_6. Noticing that $N_1 \neq N_2$, we have:

$$t_6[X_1 \cup X_2 - \{N_1\} - \{N_2\}] = t_7[X_1 \cup X_2 - \{N_1\} - \{N_2\}], t_6[N_1] = t_7[N_1] \tag{44}$$

And

$$t_7[N_2] = f_2(t_6[L_2]) \tag{45}$$

From 43, and $N_1 \notin X_2$, we get:

$$t_5[L_2] = t_6[L_2] \tag{46}$$

From 45, 46, 40, 39, and 36, we have

$$t[N_2] = t_7[N_2]$$

From 44, 43, 40, 38, and 36, we have

$$t[N_1] = t_7[N_1]$$

From 44, 43, 40, 39, 38, and 36, we have

$$t[X_1 \cup X_2 - \{N_1\} - \{N_2\}] = t_7[X_1 \cup X_2 - \{N_1\} - \{N_2\}]$$

i.e.

$$t = t_7$$

Since $t_7 \in deriveAttr(deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2, L_1, N_1, f_1), L_2, N_2, f_2)$, we have:

$$t \in deriveAttr(deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2, L_1, N_1, f_1), L_2, N_2, f_2)$$

i.e. $t \in RHS$.

Consider any tuple t in the LHS, i.e.

$$t \in deriveAttr(deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2, L_1, N_1, f_1), L_2, N_2, f_2)$$

Let $t_1 \in deriveAttr(R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2, L_1, N_1, f_1)$ be the origin of t. We have:

$$t[X_1 \cup X_2 \cup \{N_1\} \cup \{N_2\} - \{N_2\}] = t_1[X_1 \cup X_2 \cup \{N_1\} \cup \{N_2\} - \{N_2\}] \tag{47}$$

and

$$t[N_2] = f_2(t_1[L_2]) \tag{48}$$

Let $t_2 \in R_1 \bowtie_{p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}} R_2$ be the origin of t_1. We have:

$$t_1[X_1 \cup X_2 \cup \{N_1\} \cup \{N_2\} - \{N_2\} - \{N_1\}] = t_2[X_1 \cup X_2 \cup \{N_1\} \cup \{N_2\} - \{N_2\} - \{N_1\}] \tag{49}$$

and

$$t_1[N_1] = f_1(t_2[L_1]) \tag{50}$$

By definition of $\bowtie$, there exists $t_3 \in R_1$ and $t_4 \in R_2$, such that:

$$t_3[X_1] = t_2[X_1], t_4[X_2] = t_2[X_2] \tag{51}$$

and

$$p^{N_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)}(t_3, t_4) = true \tag{52}$$

Let $t_5 \in deriveAttr(R_1, L_1, N_1, f_1)$ be the image of t_3, we have:

$$t_3[X_1 - \{N_1\}] = t_5[X_1 - \{N_1\}], t_5[N_1] = f_1(t_3[L_1]) \tag{53}$$

Let $t_6 \in deriveAttr(R_2, L_2, N_2, f_2)$ be the image of t_4, we have:

$$t_4[X_2 - \{N_2\}] = t_6[X_2 - \{N_2\}], t_6[N_2] = f_2(t_4[L_2]) \tag{54}$$

From 54, 53, 52, we have:

$$p(t_5, t_6) = true \tag{55}$$

Let $t_7 \in deriveAttr(R_1, L_1, N_1, f_1) \times deriveAttr(R_2, L_2, N_2, f_2)$ be the concatenation of t_5 and t_6,

$$t_7[X_1 \cup \{N_1\}] = t_5, t_7[X_2 \cup \{N_2\}] = t_6 \tag{56}$$

From 56 and 55, we have:

$$t_7 \in deriveAttr(R_1, L_1, N_1, f_1) \bowtie_p deriveAttr(R_2, L_2, N_2, f_2) \tag{57}$$

i.e. $t_7 \in LHS$. Since $L_2 \subseteq X_2$, $N_1, N_2 \notin L_2$, from 56, 54, 51, 49, and 48, we have

$$t[N_2] = t_7[N_2] \tag{58}$$

From 56, 53, 51, 50, and 47, we have:

$$t[N_1] = t_7[N_1] \tag{59}$$

Easily we can also get:

$$t[X_1 \cup X_2 - \{N_1\} - \{N_2\}] = t_7[X_1 \cup X_2 - \{N_1\} - \{N_2\}] \tag{60}$$

From 60, 59, and 58, we have:

$$t = t_7$$

From 57, we get:

$$t \in deriveAttr(R_1, L_1, N_1, f_1) \bowtie_p deriveAttr(R_2, L_2, N_2, f_2)$$

i.e. $t \in LHS$. ■