Non-Equivalent Query Rewritings*

Amy J. LEE§, Andreas KOELLER‡, Anisoara NICA†, and Elke A. RUNDENSTEINER‡

(§)Dept. of Info. & Sys. Mgmt.
HKU of Science & Technology
Kowloon, Hong Kong
imalee@ust.hk

(‡)Dept. of Computer Science
Worcester Polytechnic Inst.
Worcester, MA 01609-2280
{koeller|rundenst}@cs.wpi.edu

(†) Dept. of EECS
Univ. of Michigan
Ann Arbor, MI 48109
anica@eecs.umich.edu

Abstract

Previous work in query rewriting has made the restricting assumption that the rewritten query must be *equivalent* to the initially given query. We now propose to relax this assumption to allow for query rewriting in situations where *equivalent* rewritings may not exist – yet alternate *not necessarily equivalent* query rewritings may still be preferable to users over not receiving any answers at all. In this paper, we apply this concept of non-equivalent query rewriting to the problem of maintaining view definitions (data warehouses), now allowing us to handle not only data but also schema-level changes of the underlying information sources. This relaxed query notion however raises the new issue of the evaluation of such queries. For this purpose, we introduce an analytical model of query rewritings that incorporates measures of information preservation (quality) of a query in addition to the commonly studied measures of view maintenance performance (query cost). Both quality and cost are integrated into one uniform model, called the Quality-Cost-Model or QC-Model, to allow for a trade-off among these two measures. This model can be used to select a good view maintenance solution from among alternate (even if not equivalent) rewritings. Experimental studies have been conducted that validate the utility and accuracy of the proposed QC-model.

1 Introduction

Traditional problems in query rewriting include query optimization [vdBK94] and rewriting queries using views [LMS95, CKP95]. Most of these works deal with the problem of maintaining the *exact original* interface and extent of a given query while optimizing performance. Recently, query rewriting with *relaxed semantics* has been proposed as a means of retaining the validity of a data warehouse (i.e., materialized queries) in situations where *equivalent* rewritings may not exist—yet alternate but *not necessarily equivalent* query rewritings may still be preferable to users over not receiving any answers at all [LNR97, NLR98]. Other scenarios that also motivate a relaxation of the "exact query" assumption include loosely-specified query paradigms as well as market-oriented environments in which very similar (but not equal) results in answering a query can lead to dramatically different query computation costs.

*This work was supported in part by the NSF NYI grant #IRI 94-57609. Dr. Rundensteiner would also like to thank IBM for the IBM Partnership Award as well as for IBM Fellowships for her students.

Generating non-equivalent query results raises a new problem in the context of query rewriting. Since results returned for a given query may now be quite distinct, it leads to the problem of having to *compare* "incomparable" query results for a given query. As one would expect the number of non-equivalent query rewritings is much larger than the number of equivalent query rewritings in general which raises the question of an automatic comparison of various rewritings. In this paper, we report the development of a measurement model for non-equivalent rewritings[1]. While the problem arises in many different environments in which queries are used, for the purpose of this paper we focus our attention on E-SQL [LNR97] as the relaxed query model and on the issue of view maintenance in data warehouses as motivation for establishing the model.

Information preservation (quality) and *view maintenance performance* (cost) of query rewritings are the two key components used for ranking in the proposed model, which we call *Quality-Cost-Model (QC-Model).* Quality is modeled as a function of the divergence from the intended view extent, both in terms of the preservation of the information amount (extent) and the information type (interface). Cost, on the other hand, refers to the commonly studied measures of performance for maintenance of the view specified by that given query over time (query cost). Given that the independent dimensions of quality and cost cannot be easily evaluated, we analyze the semantics of these dimensions and propose a model of assigning numerical values and trade-off parameters in order to achieve a quantifiable overall evaluation for rewritings. The resulting model, which we call *Quality-Cost-Model (QC-Model)*, combines these two dimensions into a single measure, while still allowing for trade-off among them if desired.

In summary, this paper first identifies the new problem of trade-offs of quality against cost for non-equivalent query rewritings and the need for a model for assessing these measures. Then, we introduce the measure of *quality* for a query, and establish techniques for determining the quality measure for a given query based on empirically supported findings. Lastly, in order to establish an integrated measure for both quality and cost, we adopt and extend an existing cost model for distributed view maintenance [ZGMHW95]. The resulting *QC-model* assigns numerical values to relaxed query rewritings, incorporating both quality and cost into one measure. We have developed a fully functional distributed data warehouse maintenance system for demonstrative and experimental purposes, which includes view synchronization algorithms and algorithms for incremental view maintenance. This EVE system will be formally demonstrated at ACM SIGMOD'99 [RKZ+99]. We have also conducted experimental studies using this EVE system that demonstrate the utility and soundness of the QC-Model. However, details of this study are beyond the scope of this current paper, and instead can be found in our technical report [LKNR98].

The remainder of this paper is organized as follows: Section 2 introduces background concepts necessary for the development of the QC-Model, whereas Sections 3 and 4 present a detailed analytic model of quality and cost trade-offs, respectively. Section 5 briefly summarizes our experimental results, Section 6 reviews related work, while Section 7 discusses our conclusions.

[1]Some of the initial ideas of the need for such non-equivalent rewritings have been indicated in a poster session at ICDE'99 [LKNR99].

2 Overview of a Relaxed Query Model – E-SQL

2.1 Query Relaxation via Relaxation Parameters

When the underlying base schema(s) upon which a query is specified undergoes changes, the query interface as well as the query extent may have to be relaxed if obtaining the original query result becomes impossible. E-SQL (or Evolvable-SQL), an extension of SQL, has been designed to allow for the specification of relaxed query semantics by query definers [2]. We take the stand that it is most appropriate for the view definers to specify the relaxed semantics at the time of query specification, as they are the ones that know the criticality and dispensability of the different components of their query. The main idea of E-SQL is to allow a query definer to specify as part of a query definition what information is indispensable, what information is replaceable by similar information from other ISs, and what relationship between original and new query result is desired. These evolution preferences are captured in seven *relaxation parameters*, see Table 1, associated with the different components of a query, such as the attributes in the SELECT clause, the conditions in the WHERE clause, and so on (Table 1). When the parameter setting is omitted from an E-SQL query, then the default value is assumed so an E-SQL query becomes an extension of SQL and pure SQL-queries are defined in an E-SQL system.

Relaxation Parameter	Domain	Default
Attribute-dispensable ($\mathcal{AD}$)	*true/false* (attribute is dispensable/indispensable)	false
replaceable ($\mathcal{AR}$)	*true/false* (attribute is replaceable/non-replaceable)	false
Condition-dispensable ($\mathcal{CD}$)	*true/false* (condition is dispensable/indispensable)	false
replaceable ($\mathcal{CR}$)	*true/false* (condition is replaceable/non-replaceable)	false
Relation-dispensable ($\mathcal{RD}$)	*true/false* (relation is dispensable/indispensable)	false
replaceable ($\mathcal{RR}$)	*true/false* (relation is replaceable/non-replaceable)	false
View-extent ($\mathcal{VE}$)	$\approx$: no restriction on the new extent $\equiv$: new extent is equal to old extent $\supseteq$ / $\subseteq$: new extent is superset/subset of old extent	$\equiv$

Table 1: Relaxation Parameters (Preferences) of the E-SQL Query Language.

We now use an E-SQL example query (Equation 1) to illustrate the relaxation parameters.

$$
\begin{array}{ll}
\mathsf{CREATE} & \mathsf{VIEW}\ \textbf{Asia-Customer}\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \mathsf{AS} \\
\mathsf{SELECT} & \textbf{C.Name}, \textbf{C}.\mathit{Address}, \textbf{C}.\mathit{Phone}\ (\mathcal{AD} = \mathit{true}, \mathcal{AR} = \mathit{true}) \\
\mathsf{FROM} & \textbf{Customer C}, \textbf{FlightRes F}\ (\mathcal{RR} = \mathit{true}) \\
\mathsf{WHERE} & (\textbf{C.Name} = \textbf{F.PName})\ \text{AND}(\textbf{F.Dest} = \text{‘Asia’})\ (\mathcal{CD} = \mathit{true})
\end{array}
\quad (1)
$$

The semantics of this query are as follows. Any query rewriting is acceptable as long as the new view extent is a superset of the old one (expressed by $\mathcal{VE}$ = “$\supseteq$”); the attribute **Phone** is dispensable and can also be replaced from another source (expressed by $\mathcal{AD} = true$, $\mathcal{AR} = true$); the relation **FlightRes** (but not **Customer**) can be replaced with another relation ($\mathcal{RR} = true$); and the user will still have use for the view even if the second WHERE condition cannot be kept valid ($\mathcal{CD} = true$).

The choice of essentially two classes of relaxation parameters (dispensability, replaceability) in E-SQL is an approach at trading off the complexity of the system (i.e., the expressiveness

[2] E-SQL has been developed in the context of the *Evolvable View Environment* (*EVE*) project [RLN97, NLR98]

of the query model) against the ease of use by a user (i.e., the simplicity in specifying such relaxed query semantics). More complex extensions of such a relaxed query model, such as a replacement of the Boolean preference values by numeric "fuzzy" values, are of course possible, but are beyond the scope of the current paper.

3 Quality of a Query Rewriting

3.1 Information Preservation in Rewritings

The information returned by a query can be discussed in terms of two aspects, namely the query interface (i.e., the attributes in the SELECT clause) and the query extent. Ideally, a rewriting, as a replacement of a query affected by a schema change, is "equivalent" to the original query in terms of both aspects, although some information of the rewriting may be taken from other information sources. If preserving the original query is impossible, the next best alternative is having a rewriting that is the most informative one, with respect to the original query, among all the rewritings. We call the amount of information preserved by rewritings "quality" of the rewritings. Ranking non-equivalent rewritings, in terms of quality, is not a trivial task. This can be best demonstrated by the following example.

Example 1 *Let the view V be defined as follows:*

$$\begin{array}{ll} \text{CREATE} & \text{VIEW } V\ (\mathcal{VE} = \text{“}\approx\text{”})\ \text{AS} \\ \text{SELECT} & \textbf{Name}\ (\mathcal{AD} = false, \mathcal{AR} = true), \textbf{Address}\ (\mathcal{AD} = true, \mathcal{AR} = true), \\ & \textbf{City}\ (\mathcal{AD} = true, \mathcal{AR} = true), \textbf{Phone}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \text{FROM} & \textbf{Customer}\ (\mathcal{RD} = false, \mathcal{RR} = true) \end{array} \tag{2}$$

where **Customer** *keeps track of the customers who live in the Boston Metropolitan area.*

Deleting the relation **Customer** *from its site could now generate two possible rewritings, by replacing* **Customer** *with* **BackBay** *and* **MABranch**, *respectively* [3]. *Note that* **BackBay** *is a strict subset of* **Customer**, *and* **MABranch** *keeps track of the customers who live in MA but does not record the street information of the customers.*

$$\begin{array}{ll} \text{CREATE} & \text{VIEW } V_1\ (\mathcal{VE} = \text{“}\approx\text{”})\ \text{AS} \\ \text{SELECT} & \textbf{Name}(\mathcal{AD} = false, \mathcal{AR} = true), \\ & \textbf{Address}(\mathcal{AD} = true, \mathcal{AR} = true) \\ \text{FROM} & \textbf{BackBay}(\mathcal{RD} = false, \mathcal{RR} = true) \end{array} \tag{3}$$

$$\begin{array}{ll} \text{CREATE} & \text{VIEW } V_2(\mathcal{VE} = \text{“}\approx\text{”})\ \text{AS} \\ \text{SELECT} & \textbf{Name}(\mathcal{AD} = false, \mathcal{AR} = true), \\ & \textbf{City}(\mathcal{AD} = true, \mathcal{AR} = true), \\ & \textbf{Phone}(\mathcal{AD} = true, \mathcal{AR} = true) \\ \text{FROM} & \textbf{MABranch}(\mathcal{RD} = false, \mathcal{RR} = true) \end{array} \tag{4}$$

From the viewpoint of the query interface, V_1 and V_2 are able to preserve a different subset of attributes of the original interface, **Name** *and* **Address** *by V_1 and* **Name**, **City**, *and* **Phone** *by V_2. From the viewpoint of the query extent, by considering the common set of attributes between the original query and a query rewriting, V_1 is able to preserve two out of three tuples of the original query without introducing any extra tuples (i.e., precise but not total recall), while V_2 is able to preserve the original query with two surplus tuples (i.e., total recall but not precise). Obviously, we need a mechanism to decide which rewriting is more exact as a replacement for V and thus superior to the other.*

[3]We refer the reader to [RLN97, NLR98] for the description of different view synchronization algorithms.

Name	Address	City	Phone
Smith	1st St.	Somerville	617-123-4567
Baker	2nd Ave.	Boston	617-321-6547
Davis	Blue St.	Boston	617-987-6543

(a) Original Extent of V (Base Table **Customer**)

Name	Address
Baker	2nd Ave.
Davis	Blue St.

(b) Rewriting V_1
(Base Table **BackBay**)

Name	City	Phone
Smith	Somerville	617-123-4567
Baker	Boston	617-321-6547
Davis	Boston	617-987-6543
Adam	Worcester	508-123-4567
Jones	Worcester	508-321-6547

(c) Rewriting V_2
(Base Table **MABranch**)

Figure 1: Different Amounts of Information Are Preserved in Rewritings.

In short, our system must trade-off the pros and cons between the query interface and query extent preservation (and also between the two dimensions of query extent precision and recall) in order to rank these potential rewritings of V in some (linear) order so that one final solution can be selected.

3.2 Information Preservation on the Query Interface

There are several ways to numerically assess desirability of the attributes of a query interface, for example, assigning a number (weight) to each single attribute or to a subset of the attributes. To restrict the parameters that must be specified by the query definer to a reasonable number, we choose to classify the attributes in a query interface into four categories according to the combination of the Boolean values *attribute-dispensable* ($\mathcal{AD}$) and *attribute-replaceable* ($\mathcal{AR}$), and have a numerical weight assigned to each category by the query definer. Since any consistent rewriting must preserve all the indispensable attributes of the original query, independent of whether the attributes can be taken from other ISs, these attributes (in categories 3 and 4) will not be assigned weights and are not included in our further discussion. In short, each attribute that has ($\mathcal{AD}$ = true, $\mathcal{AR}$ = true), i.e., in category 1, contributes w_1 to the query's desirability, and w_2 for attributes which are ($\mathcal{AD}$ = true, $\mathcal{AR}$ = false), i.e., category 2, where $0 \leq w_1, w_2 \leq 1$. Since preserving replaceable attributes is intuitively preferable over preserving non-replaceable attributes, default values for those parameters would have $w_1 > w_2$ (further motivated in [LKNR98]).

3.3 Information Preservation on the Query Extent

In order to present our findings on the query extent, we now introduce a notation for *common set of attributes* and *common-set-of-attributes-equivalence.*

Let V and V_i be two query definitions, and $Attr(V)$ and $Attr(V_i)$ be their query interfaces. $Attr(V) \cap Attr(V_i)$ is the common set of attributes of V and V_i. We use $V^{\pi(V_i)}$ to denote the projection of relation V on the common attributes of V and V_i, i.e., $V^{\pi(V_i)} = \pi_{Attr(V) \cap Attr(V_i)} V$. Similarly, $V_i^{\pi(V)}$ is defined as $\pi_{Attr(V) \cap Attr(V_i)} V_i$. When V_i and V have different query interfaces, we refer to V_i as being *common-set-of-attributes-equal* to V, denoted by $V_i =_\pi V$, if their projections on the common set of attributes are equal. Other Boolean ($\subseteq_\pi$) and set ($\cap_\pi, \setminus_\pi$) operations between the view extents of V_i and V can be similarly defined on the common set of attributes of V_i and V. These operations and their semantics are summarized in Figure 2.

Operator	Semantics
$V =_\pi V_i$	$\forall\, t \in V, \exists\, t_i \in V_i$ s.t. $t[Attr(V) \cap Attr(V_i)] = t_i[Attr(V) \cap Attr(V_i)]$ and $\forall\, t_i \in V_i, \exists\, t \in V$ s.t. $t_i[Attr(V) \cap Attr(V_i)] = t[Attr(V) \cap Attr(V_i)]$
$V_i \subseteq_\pi V$	$\forall\, t_i \in V_i, \exists\, t \in V$ such that $t_i[Attr(V) \cap Attr(V_i)] = t[Attr(V) \cap Attr(V_i)]$
$V \cap_\pi V_i$	$\{z \mid \exists\, t \in V \wedge \exists\, t_i \in V_i,\ z = t[Attr(V) \cap Attr(V_i)] = t_i[Attr(V) \cap Attr(V_i)]\}$
$V \setminus_\pi V_i$	$\{z \mid \exists\, t \in V, z = t[Attr(V) \cap Attr(V_i)] \wedge \not\exists\, t_i \in V_i,\ z = t_i[Attr(V) \cap Attr(V_i)]\}$

Figure 2: Operations on the Common Set of Attributes of V and V_i.

Judging the importance of each single tuple in V is unrealistic, if not altogether impossible, therefore we choose to judge the quality of a rewriting on view extent on two aspects: first how many tuples of the original extent are preserved (i.e., recall), and second how many surplus tuples are introduced (i.e., precision), by comparing tuples on the common set of attributes only. Ideally, a good rewriting is able to maximize the former number while minimizing the latter.

3.4 Metric of Quality: Degree of Divergence ($\mathcal{DD}$)

Below, we discuss how we quantified the quality of query rewritings in terms of the query interface and the query extent, individually, and how to unify these two measures into one single magnitude – the *Degree of Divergence ($\mathcal{DD}$)* of a rewriting from the original query.

3.4.1 Degree of Divergence on the Query Interface ($\mathcal{DD}_{attr}(V_i)$)

Let $|A_i^1|$ ($|A_i^2|$) be the number of dispensable attributes in the query interface of V_i that are replaceable (non-replaceable). The quality of the query interface of V_i can be defined as:

$$Q_{V_i} = |A_i^1| \cdot w_1 + |A_i^2| \cdot w_2 \tag{5}$$

The quality of the original view V is defined likewise and denoted by Q_V. The normalized degree of divergence of V_i from V in terms of the query interface, denoted by $DD_{attr}(V_i)$, can then be defined as:

$$DD_{attr}(V_i) = \begin{cases} 0 & \text{if } Q_V = 0 \\ \frac{Q_V - Q_{V_i}}{Q_V} & \text{otherwise} \end{cases}$$

This is a measure of *distance* of the interface of a rewriting from the original query interface. $Q_V = 0$ means that the attributes contained in the original query V are all indispensable. In this case, any legal rewriting V_i of V must preserve all the indispensable attributes. That is to say, V_i does not diverge from V on the query interface (i.e., $DD_{attr}(V_i) = 0$). When there are dispensable attributes in V, then $Q_V > 0$ and $DD_{attr}(V_i)$ is computed as defined above. If V_i does not preserve any of the dispensable attributes, then $Q_{V_i} = 0$ and $DD_{attr}(V_i) = 1$. So in terms of the query interface, V_i is preferred to V_j if $DD_{attr}(V_i) < DD_{attr}(V_j)$.

Example 2 *Let us look at the query and rewritings defined in Example 1. In that example,* $Attr(V) = \{\textbf{Name}, \textbf{Address}, \textbf{City}, \textbf{Phone}\}$, $A^1 = \{\textbf{Address}, \textbf{City}, \textbf{Phone}\}$, *and* $A^2 = \emptyset$. *Therefore,* $Q_V = 3 \cdot w_1$. *The rewriting* V_1 *preserves the attribute* **Address** *(besides the indispensable attribute* **Name***). Therefore,* $Q_{V_1} = 1 \cdot w_1$. *On the other hand, the rewriting* V_2 *preserves two of the dispensable attributes* **City** *and* **Phone**. *Therefore,* $Q_{V_2} = 2 \cdot w_1$. *Thus,* V_2 *is preferred to* V_1 *as indicated by* $1/3 = DD_{attr}(V_2) < DD_{attr}(V_1) = 2/3$.

3.4.2 Degree of Divergence on the Query Extent ($\mathcal{DD}_{ext}(V_i)$)

The divergence of a legal rewriting V_i from V is computed in two dimensions:

D1. The (relative) number of tuples in the original V that are not preserved in the new V_i, denoted by

$$DD_{ext\_D1}(V_i) = 1 \;-\; \frac{|V_i \cap_\pi V|}{|V^{\pi(V_i)}|} \tag{6}$$

D2. The (relative) number of surplus tuples in the new view V_i that are not in the original view V, denoted by

$$DD_{ext\_D2}(V_i) = \frac{|V_i \setminus_\pi V|}{|V_i^{\pi(V)}|} \;=\; \frac{|V_i^{\pi(V)}| - |V \cap_\pi V_i|}{|V_i^{\pi(V)}|} \tag{7}$$

Intuitively, the number of tuples that are *not preserved* (Case **D1**) has to be related to the size of the original view extent $|V^{\pi(V_i)}|$, whereas the number of extra tuples coming into the new view (Case **D2**) must be seen relative to the size of the *new* view extent $|V_i^{\pi(V)}|$. The total extent divergence of V_i from V is the weighed sum of $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D2}(V_i)$, denoted by $DD_{ext}(V_i)$, and defined as follows:

$$\begin{aligned} DD_{ext}(V_i) &= \varrho_1 \cdot DD_{ext\_D1}(V_i) + \varrho_2 \cdot DD_{ext\_D2}(V_i) \\ &= 1 - \frac{(\varrho_1 |V_i^{\pi(V)}| + \varrho_2 |V^{\pi(V_i)}|) \cdot |V_i \cap_\pi V|}{|V^{\pi(V_i)}||V_i^{\pi(V)}|} \end{aligned} \tag{8}$$

where ϱ_1 and ϱ_2 are the trade-off parameters between $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D2}(V_i)$ ($\varrho_1,\ \varrho_2 \geq 0$ and $\varrho_1 + \varrho_2 = 1$). Again, the view definer is given an opportunity to set the trade-off parameters, with the default setting being $(\varrho_1, \varrho_2) = (0.5, 0.5)$. That is because we assume recall and precision of a rewriting are of equal importance for end-users.

3.4.3 Estimation of Overlapping View Extents

In order to compute degrees of divergence using Equation (8), we need to be able to estimate how many tuples are common to both the view extents of V_i and V, i.e., the intersection $V_i \cap_\pi V$, without having to compute the actual overlap query. With this number and the estimated numbers of tuples in both the original view extent V and the new view extent V_i, we can then compute the desired degree of divergence $DD_{ext}(V_i)$ for the view V_i. We will now discuss how one could estimate the size of the intersection $V_i \cap_\pi V$. To help illustrate the process of estimating the size of the overlapping view extents, we consider the view from Example 1 (Equation (2)) with the legal rewritings defined there (Equations (3) and (4)).

Note that for the identification of such non-equivalent rewritings the system must have available some statistics and meta information about the information sources [CL95] (typically maintained in the data dictionary). One of the most important forms of meta information for our problem of view rewriting are *containment constraints.* A containment constraint ($\mathcal{CONTAIN}$) between two relations R_1 and R_2 states that a (horizontal and/or vertical) fragment of R_1 is semantically contained or equivalent to a (horizontal and/or vertical) fragment of R_2 at all times. Two $\mathcal{CONTAIN}$ constraints related to our example are given next:

$$\mathcal{CONTAIN}_{\textbf{Customer},\textbf{BackBay}} = (\pi_{\textbf{Name},\textbf{City},\textbf{Phone}}(\textbf{Customer}) \supseteq \pi_{\textbf{Name},\textbf{City},\textbf{Phone}}(\textbf{BackBay})) \tag{9}$$

$$\mathcal{CONTAIN}_{\mathbf{MABranch},\mathbf{Customer}} = (\pi_{\mathbf{Name,\ Address}}(\mathbf{MABranch}) \supseteq \pi_{\mathbf{Name,\ Address}}(\mathbf{Customer})) \tag{10}$$

The first constraint asserts that **BackBay** is a subset of **Customer** when comparing the projections of the common set of attributes. The second constraint states the containment relationship between **MABranch** and **Customer** likewise. The rewritings given in Equations (3) and (4) replace the dropped relation **Customer** with the relations **BackBay** and **MABranch**, respectively, thus using above constraints. Given this example, we now demonstrate how to estimate the view overlap $|V \cap_\pi V_i|$ needed for Equation (8).

Each (old and new) view is based on a base relation R. The key to estimating intersection sizes is to determine how many tuples in a base relation R *derive* tuples in each view, i.e., how many tuples from a relation R participate in a view V derived from R. We will call this the *view selectivity* of R with regard to V and denote this relative number of tuples in relation R that derive tuples in the view V by $\sigma_{V,R}$. For our example, which is based on only one relation [4], this corresponds to the following measures:

$$\sigma_{V,\mathbf{Customer}} = \frac{|V|}{|\mathbf{Customer}|}; \quad \sigma_{V_1,\mathbf{BackBay}} = \frac{|V_1|}{|\mathbf{BackBay}|}; \quad \sigma_{V_2,\mathbf{MABranch}} = \frac{|V_2|}{|\mathbf{MABranch}|} \tag{11}$$

We make the assumption that the distribution of data within a relation, e.g., **Customer**, will roughly be maintained in any subset or superset of this table. Then, we can estimate that if a certain percentage of tuples in **Customer** contributes to the view V, the same percentage of tuples in **BackBay** or **MABranch** will contribute to a derived view V_i, i.e., we assume that if n tuples in a base relation derive $\sigma \cdot n$ tuples in a view, then m tuples in any subset of this relation will roughly derive $\sigma \cdot m$ tuples in the view. This implies that σ is a parameter of the information space and the current view and can be pre-computed prior to the estimation process [5]. We have found experimentally that this assumption can be supported by statistic means (Section 5). Assuming equality of the parameters in Equation (11), we need to measure or estimate only $\sigma_{V,R} = \sigma_{V,\mathbf{Customer}}$ and can then assume the same value of $\sigma_{V,R}$ for all rewritings of V and tables replacing R.

Given this parameter, we can now compute the size of an overlap between an original view and a view rewriting. For this computation, we need to determine the size of the overlaps of the original and replacing relations $|R \cap_\pi S|$ (through a query or stored system parameters). A complete discussion of how to determine overlaps between relations even in the case of the original relation being unavailable can be found in [Nic99].

Knowing the size of the overlaps between the base relations R and S, we can now obtain the size of the view overlap by calculating :

$$|V \cap_\pi V_i| = \sigma_{V,R} \cdot |R \cap_\pi S| \tag{12}$$

with $\sigma_{V,R}$ the parameter computed above. Here, we again assume that the percentage of tuples deriving tuples in the view, denoted by $\sigma_{V,R}$, will be constant.

3.4.4 Total Degree of Divergence on Information Preservation

With the above findings, we now define the total degree of divergence of V_i from Vas:

$$DD(V_i) = \varrho_{attr} \cdot DD_{attr}(V_i) + \varrho_{ext} \cdot DD_{ext}(V_i),\ where\ \varrho_{attr}, \varrho_{ext} \geq 0\ \ and\ \varrho_{attr} + \varrho_{ext} = 1. \tag{13}$$

[4]For views based on joins of more than one relation, this parameter can be determined by estimating the number of tuples in relation R that satisfy the WHERE conditions on relation R in the view definition.

[5]i.e.,$\sigma_{V,\mathbf{Customer}} \approx \sigma_{V_1,\mathbf{BackBay}} \approx \sigma_{V_2,\mathbf{MABranch}}$

A *degree of divergence* of 0 represents a "perfect" rewriting that preserves all tuples and all attributes of an original view extent. A *degree of divergence* of 1 means a view rewriting that does not preserve any of the dispensable attributes of a view and also preserves no tuples that appeared in the original view.

4 Cost of a Query Rewriting

4.1 View Maintenance Basics

We assume that data content updates on the base relations, e.g., inserts or deletes of tuples to/from the base relations, take place more frequently than schema changes. Therefore, we choose to rank the legal rewritings by their *long term* view maintenance costs [6]. A legal rewriting is considered to be *preferred* if its expected view maintenance costs are low compared to other legal rewritings.

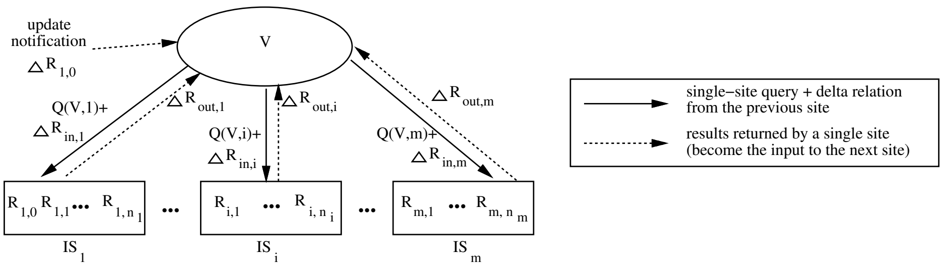

Figure 3: View Maintenance Process.

The conventional incremental view maintenance algorithm, similar to the one in [ZGMHW95] is reviewed next. We assume the IS data updates are sufficiently spaced from each other, so concurrent data updates are not considered in this paper. Considering concurrent updates would significantly complicate this portion of our analytical model, but we expect that it would not have a large enough impact on our findings and will confirm this assumption using our test bed. We assume a view V that references relations $R_{i,j}$ with $i = 1, ..., m$ and $j = 1, ..., n_i$ residing in m ISs (see Figure 3). Let $n = 1 + \sum_{j=1}^{m} n_j$ be the total number of relations that are referred to in V. Further assumptions are that ISs are cooperative and able to join their local relations with incoming delta-relations, hence allowing for the data warehouse not to have to perform any massive joins (e.g., through a wrapper). Partial results are sent along to the next IS (for all ISs), and maintenance is done in a non-parallel way. The relations referenced in the view are joined in an order that does not make it necessary to again query data from an IS already visited [7].

Without loss of generality, we assume $IS_1.R_{1,0}$ changes its data content and IS_1 notifies the system about the data change by sending the update information $\Delta R_{1,0}$ to the view site. Upon receiving this notification, the system decides which views are affected by it. Then our view maintainer brings the view extents of these affected views up-to-date by iterating through the

[6] The cost for recomputing the original view extent after a view re-definition is a one-time cost. Thus we do not judge the legal rewriting on this one-time view update cost.

[7] If this is the case, we can treat such a physical IS as two or more logical ones and count it more than once in our computation, which would then generate the correct results for the cost estimates.

information sources (with the possible exception of IS_1 if that had only one relation) and send single site queries one-by-one (Figure 3).

The delta relation ($\Delta R_{out,i}$) sent back by IS_i to the warehouse becomes the delta relation ($\Delta R_{in,i+1}$) sent along with the single site query Q(V,i+1) to IS_{i+1}. Similar to prior work, we assume the following for estimation purposes:

1. The cardinality (number of tuples) of each relation R is known and denoted as $|R|$.
2. The size of each attribute $R.A$ is known as $s_{R.A}$ (and registered in the system dictionary).
3. The join selectivity (js) is the percentage of tuples in a relation that would join (with an equijoin condition [8]) with tuples in the other relation.
4. We assume there is a local selection condition σ_{R_i} for each relation involved in a view definition. Similar to joins, all operators of local conditions are equality-based (to have a constant local selectivity σ_{IS_i}).
5. $|R|$ and js do not change significantly as updates occur.
6. bfr_{R_i} is the number of bytes per physical block in the information source for relation R_i.

Like previous work [ZGMHW95], we consider three cost factors (for a single data content update) for a particular legal rewriting, namely, the number of messages exchanged, the number of bytes transferred, and the I/O-cost at the local ISs as further elaborated below.

4.2 Cost Factor Based on Number of Messages Exchanged (CF_M)

The number of messages exchanged between the information space and the view site for a single base data update, denoted as CF_M, is in the range $[0, 2m]$ (with m denoting the number of information sources involved in the view). To be more specific:

$$CF_M = \begin{cases} 0 & \text{if } m = 1 \text{ and } n_1 = 0 \\ 2 & \text{if } m = 1 \text{ and } n_1 > 0 \\ 2 \cdot (m-1) & \text{if } m > 1 \text{ and } n_1 = 0 \\ 2 \cdot m & \text{otherwise} \end{cases}$$

n_1 is the number of relations in the update-generating IS *besides* the relation where the update occured. Note that when there is only one relation in IS_1 referred to in V ($n_1 = 0$), then no query needs to be sent to IS_1.

4.3 Cost Factor Based on Bytes of Data Transferred (CF_T)

Considering an information space consisting of n relations $R_1, \ldots, R_n$ in m information sources $IS_1, \ldots, IS_m$, we can derive a general computation for the number of bytes transferred (assuming the algorithm described earlier in Section 4.1). This computation assumes that one inserted/deleted tuple is sent from IS_1 to the view site, which is the initial delta relation (the first line in Equation 14). Then this delta relation is sent down to IS_1 to join with other relations in IS_1 referred to in the view query (the first term in the second line), and the resulting new delta relation (the second term in the second line) is sent back to the view site. The same process iterates through all ISs referred to in the view to build up the delta relation that contains the tuples affected by the data update. In summary, the number of bytes transferred can thus

[8] To simplify we assume all the relations are joined by some equijoin conditions.

be approximated by:

$$
\begin{aligned}
CF_T \quad &= \underbrace{s_{R_{1,0}}}_{update\ notification} \\
+ \quad & \underbrace{s_{R_{1,0}}}_{\Delta R_{in,IS_1}} + \underbrace{\sigma_{IS_1} \cdot J_{IS_1} \cdot s_{\Delta R_{out,IS_1}}}_{\Delta R_{out,IS_1}} \\
+ \quad & \underbrace{\sigma_{IS_1} \cdot J_{IS_1} \cdot s_{\Delta R_{out,IS_1}}}_{\Delta R_{in,IS_2}} + \underbrace{\sigma_{IS_1}\sigma_{IS_2} \cdot J_{IS_1}J_{IS_2} \cdot s_{\Delta R_{out,IS_2}}}_{\Delta R_{out,IS_2}} + \cdots \\
+ \quad & \underbrace{(\sigma_{IS_1} \cdot \ldots \cdot \sigma_{IS_{m-1}})(J_{IS_1} \cdot \ldots \cdot J_{IS_{m-1}}) s_{\Delta R_{out,IS_{m-1}}}}_{\Delta R_{in,IS_m}} \\
& + \underbrace{(\sigma_{IS_1} \cdot \ldots \cdot \sigma_{IS_m})(J_{IS_1} \cdot \ldots \cdot J_{IS_m}) s_{\Delta R_{out,IS_m}}}_{\Delta R_{out,IS_m}}
\end{aligned}
\tag{14}
$$

where s_R is the size (sum of the length of attributes in bytes) of the relation R or intermediate query result R, σ_{IS_i} is the average selectivity for the selection conditions used in the single-site query to source IS_i , and J_{IS_i} is the estimated size of the resulting join relation $R_{i,1} \bowtie R_{i,2} \bowtie \cdots \bowtie R_{i,n_i}$ returned by the source IS_i. That is, $J_{IS_i} \approx js^{n_i} \cdot |R_{i,1}| \cdot |R_{i,2}| \cdot \ldots \cdot |R_{i,n_i}|$, with js^{n_i} being the average join selectivity for this IS, raised to the power of the number of relations in this IS.

4.4 Cost Factor Based on I/O ($CF_{I/O}$)

We use the total number of estimated input/output operations (block accesses) performed by local ISs in order to process incremental view maintenance for each legal rewriting as a criterion to rank the legal rewritings. Let $CF_{I/O}(IS_i)$ be the number of estimated I/Os at the information source IS_i. $CF_{I/O}(IS_i)$ is the sum of the I/Os of the relations that reside at source IS_i, i.e., incorporating the I/O-costs of all relations at IS_i. Then the total number of I/Os, denoted as $CF_{I/O}$, is the sum of the I/Os at all m sources, i.e.,

$$CF_{I/O} = \sum_{i=1}^{m} CF_{I/O}(IS_i) \tag{15}$$

Ways to estimate the number of blocks accessed for a relational DB query are available in the literature (see also our technical report [LKNR98]).

4.5 Total View Maintenance Cost

The total view maintenance cost of a view V with respect to a data update can now be defined as:

$$Cost(V) = CF_M \cdot cost_M + CF_T \cdot cost_T + CF_{I/O} \cdot cost_{I/O} \tag{16}$$

where $cost_M, cost_T$, and $cost_{I/O}$ are unit costs for sending a message, transferring a byte of data, and performing a disk I/O, respectively. We have conducted experiments that empirically obtain useful values for these costs (for a brief summary see Section 5, for details see our technical report [LKNR98]).

We normalize the cost for k legal rewritings $V_i (i = 1 \ldots k)$ of a view V_i as follows:

$$COST^*(V_i) = \frac{COST(V_i) - min_{1 \le j \le k}(COST(V_j))}{max_{1 \le j \le k}(COST(V_j)) - min_{1 \le j \le k}(COST(V_j))} \tag{17}$$

4.6 Overall QC_Value of a Rewriting

The *overall QC_value* of a legal rewriting can now be computed as

$$QC(V_i) = 1 - (\varrho_{quality} \cdot \mathcal{DD}(V_i) + \varrho_{cost} \cdot COST^*(V_i)) \tag{18}$$

with $0 \leq \varrho_{quality}, \varrho_{cost} \leq 1$ and $\varrho_{quality} + \varrho_{cost} = 1$. With both quality and cost normalized, this number will be between 0 and 1. A *QC_value* of 0 means a legal rewriting that preserves no information (which renders it "useless"). A *QC_value* of 1 would identify a "perfect" legal rewriting preserving the complete view interface and all tuples at no cost.

5 Experimental Evaluation

We have conducted a series of experiments to support our proposed QC-Model. Due to space considerations, we summarize only a small subset of our results below while detailed experiments and their results are reported in our technical report [LKNR98]. In the context of the EVE-project [9], we have implemented a Java-based experimental data warehouse maintenance system that is able to maintain a data warehouse over distributed sources handling both schema and data source changes and which we use for this study. The schema and view for the experiments were taken from the TPC/D benchmark test provided by the Transaction Processing Performance Council (TPC).

We ran studies to validate the estimation function we proposed in Section 3.4.3 for predicting the size of overlaps between views ($V \cap V_i$), as the latter is a parameter needed for the computation of the Degree of Divergence between two views. For this, we measured the size of a view which was based on a number of base tables and varied one of its base tables R in size. The replacements were different-sized randomly selected subsets R_i of the original table R. We found a high correlation between the size of actual view overlaps and the estimated size obtained in Section 3.4.3. For details, please refer to our technical report [LKNR98].

Our experiments supporting the soundness and correctness of the cost part of our QC-Model focused on determining how well the estimation by our cost model predicts the actual cost of maintenance after data updates (Section 4). While the cost part of QC incorporates aspects such as data transported, I/O-cost at the ISs, etc., for these experiments, we assess cost as the (real) time it takes our data warehouse to update its extent after a data update in an underlying IS has been detected. An important result of this experimental study is that we can not only support the usefulness of our cost model, but we can empirically compute the unit costs $cost_M, cost_T$, and $cost_{I/O}$ (Equation 18) for our experimental data warehouse setup by measuring execution on example views. With carefully selected views and classes of updates, it is possible to separate the measurement of the three cost factors and determine a value for each with high statistical confidence. For different setups, these parameters will be different but generally constant for a given data warehouse implementation.

Postulating the correctness of our cost model and linear correlation between the number of messages, transferred bytes, and I/O-operations, respectively, with query execution time, we measured the following cost on the reference system (given with their correlation coefficients r in Table 2).

The high correlation factors in Table 2 suggest that there is a correlation between our cost factors and the actual view maintenance cost. A linear dependency between a tradeoff of these three cost factors (using the unit costs obtained above) and the actual measured execution time

[9] Please also refer to our project home page `http://davis.wpi.edu/dsrg/EVE`.

Parameter	Value	Correlation r
$cost_{I/O}$	$1.96 \cdot 10^{-3} \frac{\text{sec}}{\text{I/O-operation}}$	0.98
$cost_T$	$6.23 \cdot 10^{-4} \frac{\text{sec}}{\text{byte}}$	0.97
$cost_M$	$0.53 \frac{\text{sec}}{\text{message}}$	0.91

Table 2: Cost Factors Measured for the Reference System

can also be statistically supported. In our experiments (comparing measured and predicted query execution times on a variety of views), we found that the correlation coefficient between the predicted and measured cost is a very high 0.96 (a small constant additional overhead time that we measured can be attributed to the system).

6 Related Work

While most prior work on views in distributed database systems has focused on equivalent view maintenance [GM95, vdBK94, AAS97], we focus on the relaxation of the equivalence assumption for views before and after maintenance. We introduced the notion of *non-equivalent query rewriting* in [RLN97, LNR97], in particular the concept of associating evolution preferences with view specifications (E-SQL). Based on this, we also developed algorithms for finding rewritings that satisfy all E-SQL relaxation constraints, referred to as the (*view synchronization*) process. [NR98, NLR98, KRH98]. This current paper now addresses the need for a measurement model that arises with the establishment of a ranking among several possible rewritings which are non-equivalent to the original one.

The problem of *query optimization* has been addressed for instance by Agrawal et al. [AAS97] or van den Berg et al. [vdBK94]. They are concerned with optimizing a given query for efficient execution. Another group of researchers answer queries using materialized views. For example, Levy et al. [LMS95] replace an original query with a new expression containing materialized view definitions such that the new query is *equivalent* to the old one. To the best of our knowledge, generating queries without *equivalence* has not been studied in this context, and view quality is a novel aspect that must be assessed as one of the key components of the desirability of a rewrting.

Incremental view maintenance has been an active area of research [CGL$^+$96, GMS93]. Recently, Zhuge et al. introduce the ECA algorithm [ZGMHW95] for incremental view maintenance in a single IS environment and establish some factors influencing cost, and they present the Strobe family [ZGMW96] for multi-source information spaces. Agrawal et al. [AAS97] propose the SWEEP-algorithm, which can ensure consistency of the data warehouse in a larger number of cases compared to Strobe. In a related effort, we have also explored maintenance under both data and schema changes, which appears to be the first effort handling both changes within one framework [ZR99b, ZR99a]. All the work mentioned above does not address quality factor of rewritings.

We propose to combine quality and cost factors and combine them into a single quantifiable cost measure. To determine the *quality* of a view rewriting, we need to estimate sizes of overlapping view extents in order to determine how much information is retained by a new query and how much meaningless new data is introduced [LKNR98]. This parallels the concept of *precision* and *recall* from information retrieval [RJB89] – though we now apply these concepts to the relational database context by addressing for example extent subset estimation.

7 Conclusion

The concept of relaxed query semantics creates a new problem of *non-equivalent query rewritings* that have to be compared to each other. While previous approaches to comparing (equivalent) query rewritings have focussed on establishing measures of query costs [ZGMHW95], in this paper we now show that the introduction of the concept of quality as a novel measure of query rewriting evaluation complimentary to cost factors is needed in the context of non-equivalent rewritings. Our measure of *divergence* expresses a well-defined semantic difference between a query and its rewriting in terms of the quality (i.e., view interface and view extent). In order to evaluate our *Quality-Cost-Model (QC-Model)*, we have implemented a test bed and performed experiments to verify the prediction accuracy of the QC-Model, varying different parameters of the information space [LKNR98]. While we developed this measure of quality and conducted the experiments in the context of EVE [LNR97], we expect that both will also contribute to problem areas beyond view synchronization whenever query reformulation techniques are applied and hence evaluation of such rewritings is required [LMS95, AAS97].

Acknowledgments. The authors would like to thank students at the Database Systems Research Group at WPI for their interactions and feedback on this research. In particular, we are grateful to Xin Zhang and Yong Li for implementing several modules of the EVE system, including the MKB, the VKB, and the SVS algorithm.

References

[AAS97] D. Agrawal, A. El Abbadi, and A. Singh. Efficient View Maintenance at Data Warehouses. In *Proc. of SIGMOD*, pages 417–427, 1997.

[CGL+96] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for Deferred View Maintenance. In *Proc. of SIGMOD*, pages 469–480, 1996.

[CKP95] S. Chaudhuri, R. Krishnamurthy, and S. Potamianos. Optimizing Query with Materialized Views. In *Proceedings of IEEE International Conference on Data Engineering*, 1995.

[CL95] T. Catarci and M. Lenzerini. Representing and Using Interschema Knowledge in Cooperative Information Systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1995.

[GM95] A. Gupta and I.S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing*, 18(2):3–19, 1995.

[GMS93] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. In *Proc. of SIGMOD*, pages 157–166, 1993.

[KRH98] A. Koeller, E. A. Rundensteiner, and N. Hachem. Integrating the Rewriting and Ranking Phases of View Synchronization. In *Proc. of ACM First Int. Workshop on Data Warehousing and OLAP (DOLAP'98)*, Nov. 1998.

[LKNR98] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings. Technical Report WPI-CS-TR-98-2, Worcester Polytechnic Inst., Computer Science, 1998.

[LKNR99] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings. In *Proceedings of IEEE International Conference on Data Engineering*, Poster Session, 1999.

[LMS95] A.Y. Levy, A.O. Mendelzon, and Y. Sagiv. Answering Queries Using Views. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 95–104, May 1995.

[LNR97] A. J. Lee, A. Nica, and E. A. Rundensteiner. Keeping Virtual Information Resources Up and Running. In *Proceedings of IBM Centre for Advanced Studies Conference (CASCON'97), Best Paper Award*, pages 1–14, Nov. 1997.

[Nic99] A. Nica. *View Evolution Support for Information Integration Systems over Dynamic Distributed Information Spaces.* PhD thesis, University of Michigan in Ann Arbor, May 1999.

[NLR98] A. Nica, A. J. Lee, and E. A. Rundensteiner. The CVS Algorithm for View Synchronization in Evolvable Large-Scale Information Systems. In *Proceedings of International Conference on Extending Database Technology (EDBT'98)*, pages 359–373, Valencia, Spain, March 1998.

[NR98] A. Nica and E. A. Rundensteiner. Using Containment Information for View Evolution in Dynamic Dist. Environments. In *Int. Workshop on Data Warehouse Design and OLAP Technology (DWDOT'98)*, Vienna, Austria, August 1998.

[RJB89] V. V. Raghavan, G. S. Jung, and P. Bollmann. A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Office Information Systems*, pages 205–229, July 1989.

[RKZ+99] E. A. Rundensteiner, A. Koeller, X. Zhang, A. Lee, and A. Nica. Evolvable View Environment. In *Demo Session Proceedings of SIGMOD'99*, 1999. to appear.

[RLN97] E. A. Rundensteiner, A. J. Lee, and A. Nica. On Preserving Views in Evolving Environments. In *Proceedings of 4th Int. Workshop on Knowledge Representation Meets Databases (KRDB'97): Intelligent Access to Heterogeneous Information*, pages 13.1–13.11, Athens, Greece, August 1997.

[vdBK94] C. A. van den Berg and M.L. Kersten. An Analysis of a Dynamic Query Optimization Schema for Different Data Distributions. In J. C. Freytag, D. Maier, and G. Vossen, editors, *Query Processing for Advanced Database Systems*, chapter 15, pages 449–473. Morgan Kaufmann Pub., 1994.

[ZGMHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proc. of SIGMOD*, pages 316–327, May 1995.

[ZGMW96] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *Int. Conference on Parallel and Distributed Information Systems*, Dec. 1996.

[ZR99a] X. Zhang and E. A. Rundensteiner. A Framework Enabling Data Warehouse Maintenance Under Concurrent Schema and Data Updates. In *Int. Database Eng. and Appl. Symp. (IDEAS'99),Montreal, Canada*, Aug. 1999.

[ZR99b] X. Zhang and E. A. Rundensteiner. Data Warehouse Maintenance Under Concurrent Schema and Data Updates. In *Proceedings of IEEE International Conference on Data Engineering*, Poster Session, Mar. 1999.