Completeness of Integrated Information Sources

Felix Naumann, Johann-Christoph Freytag, Ulf Leser

Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, GERMANY

Abstract

For many information domains there are numerous World Wide Web data sources. The sources vary both in their extension and their intension: They represent different real world entities with possible overlap and provide different attributes of these entities. Mediator-based information systems allow integrated access to such sources by providing a common schema against which the user can pose queries. Given a query, the mediator must determine which participating sources to access and how to integrate the incoming results.

This article describes how to support mediators in their source selection and query planning process. We propose three new merge operators, which formalize the integration of multiple source responses. A completeness model describes the usefulness of a source to answer a query. The completeness measure incorporates both extensional value (called coverage) and intensional value (called density) of a source. We show how to determine the completeness of single sources and of combinations of sources under the new merge operators. Finally, we show how to use the measure for source selection and query planning.

Key words: Query planning, Coverage, Density, Information integration, Result size, Overlap

1 Introduction

The development of the World Wide Web has made it possible to access a multitude of data sources on almost any given topic. Most often, a user may

Email addresses: naumann@informatik.hu-berlin.de (Felix Naumann), freytag@dbis.informatik.hu-berlin.de (Johann-Christoph Freytag), leser@informatik.hu-berlin.de (Ulf Leser).

choose between many alternative sources to obtain the desired piece of information. In particular, the desired information may either reside at one source, on several alternative sources, or the desired information might be split across multiple sources. In the first case the user can choose to query only one (best) source or to query multiple sources and combine the results. In the second case, the user *must* query multiple sources and combine the results.

Cooperative information systems and integrated information systems in general are often based on a mediator-wrapper architecture [1]. Therein, each data source is wrapped by one or more source-specific modules, the *wrappers*, which offer an export schema and query interface hiding the particular data model, access path, and interface technology of the source. Wrappers are used by a *mediator* which offers an integrated access through its global schema.

In most projects for information integration, it is implicitly assumed that the mediator should always compute the complete answer. We argue that in many cases this assumption is wrong:

- Computing the complete answer is not always necessary. For instance, a meta-search engine does not need to download all hits from all search engines it uses; instead, taking the top ten hits usually suffices.
- Computing the complete answer may be too expensive if querying data sources costs money, or it may take too long.

On the other hand, computing *only* complete results is also not always sufficient. To see this, consider a query asking for books by Stephen King, together with titles, reviews, and prices. Imagine having two data sources, one being a large online bookseller that does not provide reviews, while the other is a small second-hand bookseller who can provide all the requested attributes. In this case, only the second source could provide answers with all attributes. But it is arguably better to query both sources even though the first source does not provide all requested attributes.

These examples show that integrated systems should also consider plans that produce answers lacking tuples and/or attribute values. We assume that a user is interested in getting the "best" answer, where the quality of a plan must be carefully weighted between the amount of data it produces, the time it takes to execute it, the money it costs, etc. In this article, we concentrate on the first issue, i.e., we present a general model for estimating the amount of data produced by a plan—its *completeness.*

The aim of this article is *not* to overcome the much lamented information overflow; we are guided by the assumption that the most complete response to the user is the best, given some cost limits. Filtering techniques of information retrieval may be used subsequently to reduce the result size. Such techniques benefit from a large amount of information to begin with.

The contributions of this article are twofold. First, we introduce an information model for WWW information systems including new operators to combine responses from multiple data sources, and a new model for query planning. Second, we define a comprehensive completeness measure that assesses the completeness of sources and of combinations of sources over the new operators. We show how to use this measure for effective information integration. These contributions combine three fields of research: information integration, query answering, and result size estimation. We regard related work on relational operators for information integration in Section 3.4, related work on query planning in Section 4.3, and related work on result size estimation in Section 6.6.

The results of our research are applicable not only to cooperative information systems as defined by De Michelis et al. in [2], but to integrated information systems in general. However, the completeness measures presented in this paper can be determined and predicted much more precisely for cooperating information sources than for integrated but autonomous sources: Completeness values of integrated results heavily rely on precise metadata about the underlying sources. For the cooperative integrated information systems under consideration, we assume the shared goal of providing answers to queries as completely and as efficiently as possible. In particular cooperative systems actively sharing and updating metadata about themselves (self-representation) are rewarded with precise completeness measures and in consequence with less redundant access to the data and less redundant network traffic.

In the first part of this article (Sections 2–4) we present the framework to which we apply our completeness model. In Section 2 we introduce a relational data model for modeling WWW data sources and describe how we integrate information from multiple sources. Section 3 defines three new merge operators performing integration. Section 4 describes user queries, source descriptions, and a method for using the new operators to construct plans across multiple sources to answer user queries. The second part of this article (Sections 5 and 6) covers our completeness model. In Section 5 we define "completeness" as a combination of the two measures "coverage" and "density". Section 6 shows how to predict completeness scores for combinations of data sources. Finally, Section 7 shows how to make use of our completeness model to find optimal plans to respond to user queries within a cost limit. We conclude in Section 8. Proofs of theorems and lemmata can be found in the appendix.

2 Integration Model of WWW Data Sources

We adopt the mediator-wrapper architecture as proposed by Wiederhold to allow integrated access to multiple, autonomous data sources (see Figure 1) [1].

The sources may be distributed across a network and heterogeneous in technical, syntactical, and semantical aspects. A mediator-based information system comprises wrappers, which hide technical and syntactical heterogeneity from the mediator, and a mediator, which additionally hides semantical heterogeneity from the user.

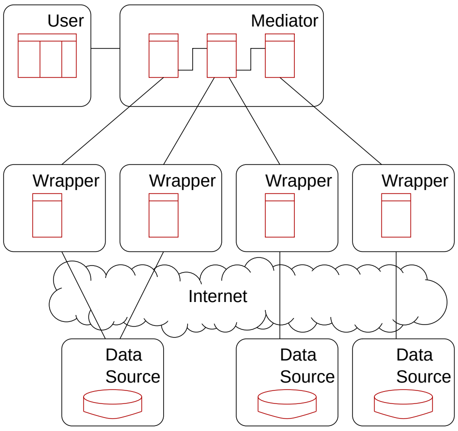

Fig. 1. The mediator-wrapper architecture

In this article we concentrate on the tasks of the mediator. Wrapper construction and deployment are discussed for instance in [3,4]. We base our approach on the relational model as the canonical data model, i.e., wrappers export a local relational schema, the mediator stores a given global relational schema.

2.1 Global Data Model

Users of a mediator-based information system communicate with the mediator by posing queries against its global relational schema. We call the set of tuples of a relation R the *extension* of R, and the set of attributes of R the *intension* of R. The extensions of relations of the global schema do not exist physically at the mediator, but reside at the participating data sources.

Relations may contain foreign ID-attributes, thus forming relationships between entities. Due to the autonomy of the sources, we can make no assumptions about referential integrity. For the same reason, we cannot require that ID-attributes are keys from the mediators point of view—the sources may store conflicting data about an entity. Together we have three disjoint types of attributes: primary IDs, foreign IDs, and ordinary attributes. To reflect the reality of many Web data sources, we allow all attribute values other than the ID to be `null`. A `null` value denotes missing or unknown data.

Example. Figure 2 shows an exemplary global schema at the mediator. It contains three relations, each with an underlined primary ID, and two relationships, which we identify by using the same attribute names. So for instance, attribute a_3 in relation R_1 is a foreign ID for the primary ID of R_2. This global schema serves as an example throughout the article. □

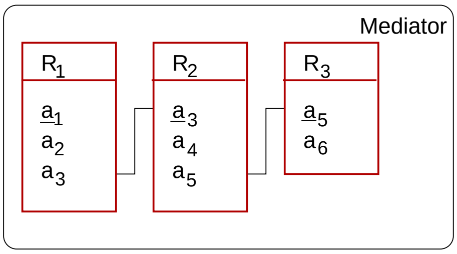

Fig. 2. The global schema of the mediator

2.1.1 Universal relation

We propose the universal relation as a means to describe data sources and user queries. The original universal relation model "aims at achieving complete access path independence in relational databases by relieving the user of the need for logical navigation among relations" [5]. Instead of revealing the global schema with its relations to the user, the relations are combined to a single, universal relation. The access path independence holds twofold for integrated information systems: First, the user must no longer formulate joins between relations, the joins are implicit in the universal relation. Second, the user need not specify which data sources are used to fill the global relations with data. The sources are selected automatically in the planning process.

Definition 1 (Universal Relation) *Let A be the set of all attributes of the global schema, i.e., the union of all attributes of global relations. Then the universal relation UR is the relation consisting of all attributes A.*

Following Maier et al., we make the usual *universal relation scheme assumption* and *unique role assumption* for the universal relation [5]. We use the concept of the universal relation (i) to describe sources and (ii) as a user interface to formulate queries. We define both aspects as subsets of the attributes of the universal relation in the following sections. This approach restricts our model to simple yet expressive source descriptions and user queries.

To represent the universal relation, the underlying relations, the data sources providing data for the relations, and user queries, we present the *UR-tableau.* A UR-tableau is a table similar to the table of the universal relation UR itself. Table 1 shows an example using the schema of Figure 2.

The columns of a UR-tableau represent the attributes of UR. The rows however do not represent tuples but rather entire tuple sets. The tuples represented by relation R_i are shown as a row R_i, where all attributes of the relations

Universal Relation $\longrightarrow$	*UR*	a_1	a_2	a_3	a_4	a_5	a_6
	R_1 :	$\underline{\checkmark}$	✓	✓			
Relations $\longrightarrow$	R_2 :			$\underline{\checkmark}$	✓	✓	
	R_3 :					$\underline{\checkmark}$	✓
	S_1 :	$\underline{\checkmark}$	✓	✓			
	S_2 :	$\underline{\checkmark}$		✓			
Sources $\longrightarrow$	S_3 :			$\underline{\checkmark}$	> 10		
	S_4 :			$\underline{\checkmark}$	> 50	✓	
	S_5 :					$\underline{\checkmark}$	✓
	Q_1 :		✓		✓		
User Queries $\longrightarrow$	Q_2 :	✓	< 10		✓		✓
	Q_3 :	✓	> 10		< 40		

Table 1
The UR-tableau

are marked '✓' in the corresponding column. ID-attributes are underlined. The tuples represented in a source S_i are shown as a row S_i, again with '✓' marks for the attributes that are exported by the source. We include predicates on attributes of the sources: Instead of marking the attribute in the UR-tableau, we note the predicate in the corresponding column. Finally, the tuples requested by a user query are represented in the same way as sources. We discuss these concepts in more detail and give examples.

2.1.2 Source description

The global schema exists only virtually; data is physically scattered over a set of heterogeneous data sources. We cannot expect that those sources conform to the structure of the global schema, nor is it feasible to adapt the global schema, whenever new sources are added or sources cease to exist. An easy plug-in and plug-out of data sources is of utmost importance in a highly dynamic environment, such as the Web. To enable this flexibility, we model sources as views on the global schema. This approach is called the local-as-view approach as opposed to the global-as-view approach, where relations of the global schema are modeled as views against the schemata of the sources [6]. For convenience we use the term 'source', but usually mean the corresponding source view on *UR* describing the source. We assume that sources export attributes of only one global relation. If a source provides data for different global relations, we model this source as two or more different views. It is not necessary that a source exports the entire set of attributes of a relation—we require only the ID-attribute. The view may contain selection conditions on the attribute values:

Definition 2 (Source view) *Let UR be the universal relation with attribute set A, and let R be a relation of the global schema with attribute set $A_R \subseteq A$. Then a source S for R exporting attribute set A_S with $A_S \subseteq A_R$ is described as*

$$S[A_S] \leftarrow R[A_R],\ C,$$

where C is a conjunction of selection conditions on attributes $a_i \in A_S$. Conditions have the form "$a_i\ \theta\ const$", where $\theta \in \{<, >, \leq, \geq, =\}$.

Of course, we do not require that different sources define views on different global relations. Instead, many sources may feed data for the same relation, i.e., we assume a one to many relationship between relations and sources.

Example. Table 1 shows five sources that provide data for *UR*. Sources have marks only for the attributes of one relation, and must have a mark for the ID-attribute, which is also ID for the source. We can see that sources S_1 and S_2 provide data for R_1, sources S_3 and S_4 for R_2, and source S_5 for R_3. The descriptions of S_3 and S_4 specify a selection on a_4. Note that some sources do not export all attributes of a relation. □

2.1.3 User Queries

A user query Q is a set of attributes from *UR*, where attributes possibly carry selection conditions.

Definition 3 (User Query) *Let A be the set of attributes of the universal relation UR. Then a user query is*

$$Q[A_Q] \leftarrow UR, C,$$

where $A_Q \subseteq A$ and C is a conjunction of selection conditions on attributes $a_i \in A_Q$. Conditions have the form "$a_i\ \theta\ const$", where $\theta \in \{<, >, \leq, \geq, =\}$.

When formulating a query, users need not perceive the underlying relations of *UR*. Note that attributes can be selected from the entire set and are not restricted to a certain relation like sources views. Users can arbitrarily select attributes of *UR* and add selection conditions. As Gupta pointed out, most users of commercial Web sites are unwilling or unable to formulate complex queries and joins [7]. The restrictions implied by the universal relation facilitate the explanation of the main ideas of this article, because they restrict the set of possible queries: No relation occurs more than once in a query, only predefined joins are used, there are no conditions on attributes that are not projected, etc. Even though such complex queries do occur, they are not common in real life situations.

Example. Table 1 shows three exemplary user queries. User queries can have marks on any attribute of the UR-tableau. □

2.2 Information Integration

Simultaneous gathering of data from multiple sources bears two advantages: We collect more data, and we collect more detailed data. Consider integrating data from several Web search engines. Search engines take keywords as a query and return URLs (links) together with some information about the linked Web page, such as title and size. Usage of more than one search engine covers larger parts of the Web and thus provides more links; for those links that are provided by more than one engine, we can hope for complementing attributes, i.e., one source returns the size of the linked Web page, another the language of the page, etc.

Generally speaking, data sources overlap in two ways: extensionally and intensionally. The extensional overlap between two sources is the set of real world entities that are represented in both sources. The intensional overlap between two sources is the set of attributes both sources provide.

Example. Consider the sources of Table 1. Sources S_1 and S_2 extensionally overlap in those tuples where the value for a_1 is the same. Intensionally they overlap in attributes a_1 and a_3. Data conflicts may arise for values of a_3 when the two sources store different values for the same real world entity identified by a_1. The extensional overlap of S_1 and S_3 are the tuples that have the same value for a_3, their intensional overlap is only a_3. □

To make use of overlap and to integrate data in a meaning- and useful way, we must recognize identical entities represented in different sources (object identification), and we must be able to resolve any data conflicts between values (conflict resolution).

2.2.1 Object Identification

Integrating data from different sources requires that different representations of identical real world entities are identified as such. This process is called object identification. Object identification is difficult, because the available knowledge about the objects under consideration is incomplete, inconsistent, and sparse. A particular problem occurs if no natural IDs exist. Object identification in the absence of IDs, which is essentially the same problem as duplicate detection [8], record linkage [9,10], or object fusion [11], is typically approached by statistical methods. To avoid the difficulties of object identification, we require that each tuple in a source has a unique ID-attribute, and that tuples gathered from different sources are identical if and only if their ID is identical. Although this requirement may seem strong, it is true for many domains: Web pages have the URL, stocks have a ticker symbol, books have an ISBN, persons have a passport number, etc.

2.2.2 Conflict Resolution

Once different tuples have been identified as representing the same entity, the data about them can be integrated. In general, a result integrated from tuples of different sources contains tuples where

(1) some attribute value is not provided by any of the sources,
(2) some attribute value is provided by exactly one source,
(3) some attribute value is provided by more than one source.

We define resolution functions to decide upon the value of the integrated result.

Definition 4 (Resolution function) *Let D be an attribute domain and $D^+ := D \cup \bot$, where $\bot$ represents the* **`null`** *value. A resolution function f is an associative function $f : D^+ \times D^+ \rightarrow D^+$ with*

$$f(x,y) := \begin{cases} \bot & \textit{if } x = \bot \textit{ and } y = \bot \\ x & \textit{if } y = \bot \textit{ and } x \neq \bot \\ y & \textit{if } x = \bot \textit{ and } y \neq \bot \\ g(x,y) & \textit{else} \end{cases}$$

where $g : D \times D \rightarrow D$. Function g is an internal associative resolution function.

Internal resolution functions may be of various types, depending on the type of attribute, the usage of the value, and many other aspects [12,13]. A simple resolution function might concatenate the values and annotate them with the source that provided the value. Especially conflicts in textual attributes may be resolved in this way. Here, the integration is not completely transparent, and users are given the opportunity to resolve the conflict by their own means. Notice that resolution functions need not depend only on the two conflicting attribute values. A resolution function could additionally depend on some date attribute of the sources and favor the most recent value.

2.3 Application Example – Meta-Search Engine

Meta-search engines, such as MetaCrawler [14], are systems that integrate existing search engines. The global schema of a meta-search engine has only one relation. We show the corresponding UR-tableau in Table 2. The global

ID of this relation is the URL; other attributes include title, description, size, etc. Search engines, such as AltaVista [15] or Google [16], are each represented by a view. National search engines, such as Web.de [17], are modeled as views with a selection on the language of a Web page.

UR	URL	title	descr.	date	size	language	category	ranking
R_1:	$\underline{\checkmark}$	✓	✓	✓	✓	✓	✓	✓
Google:	$\underline{\checkmark}$	✓	✓		✓		✓	
AltaVista:	$\underline{\checkmark}$	✓	✓			✓		
Web.de:	$\underline{\checkmark}$	✓	✓			'= german'		✓

Table 2
Universal relation for a meta-search engines

3 Merge Operators

In the presence of data overlap and value conflicts between sources, standard relational operators are not useful for integration: Overlap must be recognized, conflicts must be resolved, and multiple plans must be integrated. To formalize the integration of sources and plans we define four new operators: the join-merge operator "$\sqcap$", the left outerjoin-merge operator "$\sqsupset$" (and right outerjoin-merge operator "$\sqsubset$"), and the full outerjoin-merge operator "$\sqcup$". Figure 3 gives an intuition of the function of the operators. The figure represents the extensions of two sources corresponding to two different relations. The sources overlap intensionally in one attribute—a foreign ID of S_i which corresponds to the ID of S_j. This forms an ID–foreign ID relationship. The extensional overlap of the two sources is the part shaded by a grid. The figure shows the extensions produced by the different merge-operators.

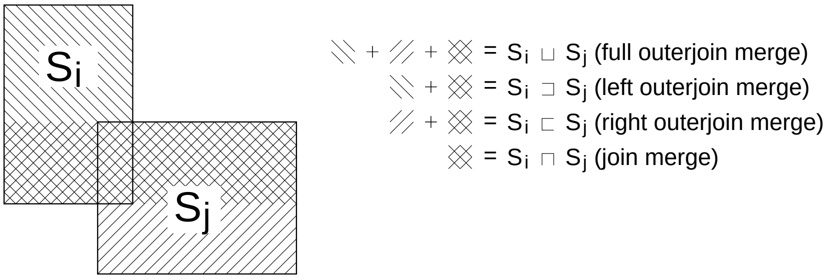
S_i
S_j
▧ + ▨ + ▩ = S_i ⊔ S_j (full outerjoin merge)
▧ + ▩ = S_i ⊐ S_j (left outerjoin merge)
▨ + ▩ = S_i ⊏ S_j (right outerjoin merge)
▩ = S_i ⊓ S_j (join merge)

Fig. 3. The four merge operators

We define the operators between sources (i.e., source views), and not between relations, because in a plan the operator is applied to sources, not to relations. The operators naturally extend to intermediate results.

We define the join-merge operator and show an example in Figure 4. The purpose of the join-merge operator is to perform join-like operations across sources with possibly conflicting data.

We restrict the use of the join-merge operator to two cases: Either the join attribute a_k is an ID for both sources ($S_1 \sqcap_{a_1} S_2$), or a_k is an ID for one source and a corresponding foreign ID for the other source ($S_1 \sqcap_{a_3} S_4$). We do not allow joins over other attributes than ID-attributes. This restriction avoids joining over conflicting data. By definition, IDs do not conflict—if they have the same value, we assume that they represent the same entity, if they differ, we assume that they represent different entities. The more general case, where tuples are merged based on the values of more than one ID attribute, would require only the generation of an artificial ID by concatenating those ID attributes and is therefore omitted. Wherever the join attribute is clear, we omit the operator index that specifies it.

Definition 5 (Join-Merge Operator $\sqcap$) *Let A be the set of attributes in the universal relation. Let $S_i = (A_i)$ and $S_j = (A_j)$ be two data sources with $A_i, A_j \subseteq A$, $A_i \cap A_j \neq \emptyset$, and let $a_k \in A_i \cap A_j$.*

$$
\begin{aligned}
S_i \sqcap_{a_k} S_j := \{\textit{tuple } t[A] \mid \exists r \in S_i,\ s \in S_j \textit{ with}\\
& t[a_k] = r[a_k] = s[a_k],\\
& t[a] = r[a],\ \forall a \in A_i \setminus A_j\\
& t[a] = s[a],\ \forall a \in A_j \setminus A_i\\
& t[a] = f(r[a], s[a]),\ \forall a \in A_i \cap A_j, a \neq a_k\\
& t[a] = \bot,\ \forall a \in A \setminus (A_i \cup A_j)\},
\end{aligned}
$$

where f is a resolution function as defined in Definition 4.

The join-merge operator returns tuples for *UR* that are joined from tuples in S_i and S_j, using a_k as join attribute. For all attributes exclusively provided by S_i, the values of S_i are used, for all attributes exclusively provided by S_j, the values of S_j are used. For common attributes, the join-merge operator applies the resolution function f to determine the final value. The values of all other attributes of *UR* are padded with `null` values.

Example. Assume sources S_1, S_2, and S_4 of Example 2.1.2 contain the tuples shown in Figure 4. Below the sources, we show the results of two join-merge operations. The two results represent the two cases to which we restrict the join-merge operator. □

The join-merge operator corresponds to a traditional inner join operation with

S_1 :	$\underline{a_1}$	a_2	a_3
	1	‘x’	15
	2	‘y’	$\perp$
	3	$\perp$	15
	4	‘z’	21

S_2 :	$\underline{a_1}$	a_3
	1	16
	4	$\perp$
	5	17

S_4 :	$\underline{a_3}$	a_4	a_5
	15	‘x’	‘g’
	16	‘y’	$\perp$
	17	‘x’	‘i’

$S_1 \sqcap_{a_1} S_2$:	a_1	a_2	a_3	a_4	a_5	a_6
	1	‘x’	$f(15, 16)$	$\perp$	$\perp$	$\perp$
	4	‘z’	$f(21, \perp)$	$\perp$	$\perp$	$\perp$

$S_1 \sqcap_{a_3} S_4$:	a_1	a_2	a_3	a_4	a_5	a_6
	1	‘x’	15	‘x’	‘g’	$\perp$
	3	$\perp$	15	‘x’	‘g’	$\perp$

Fig. 4. The join-merge operator

two exceptions: (i) We allow only one attribute as join attribute, even if the two sources have more than one attribute in common. This join-attribute is a_k in the definition. (ii) For all other common attributes data conflicts might arise, i.e., the sources provide different values for the attribute. Instead of creating a new tuple, a resolution function f resolves these conflicts and determines which value shall appear in the result.

3.2 The Left Outerjoin-Merge Operator

We define the left outerjoin-merge operator and continue the example in Figure 5. The definition is based on the join-merge operator of Definition 5. We also define the right outerjoin-merge, but shall use only the left outerjoin-merge operator in the following. We use the outer-union operator $\uplus$, which performs a union over relations with differing attribute sets [18]. The attribute set of the result is the union of the attribute sets of the two relations. In our case this is the entire attribute set A because the result of a join-merge operation has A as attribute set. $[A_i]$ denotes the projection on attributes A_i.

Definition 6 (Left/Right Outerjoin-Merge Operator $\sqsupset$ / $\sqsubset$) *Let A be the set of attributes in the universal relation. Let $S_i = (A_i)$ and $S_j = (A_j)$ be two data sources with $A_i, A_j \in A$, $A_i \cap A_j \neq \emptyset$, and let $a_k \in A_i \cap A_j$.*

$$\begin{aligned} S_i \sqsupset_{a_k} S_j &:= (S_i \sqcap_{a_k} S_j) \ \uplus \ (S_i \setminus (S_i \sqcap_{a_k} S_j)[A_i])\,, \\ S_i \sqsubset_{a_k} S_j &:= S_j \sqsupset_{a_k} S_i. \end{aligned}$$

The left outerjoin-merge corresponds to the classical left outerjoin applying the same restrictions as for the join-merge operator. The left outerjoin-merge

$S_1 \sqsupset_{a_1} S_2 :$ a_1	a_2	a_3	a_4	a_5	a_6
1	‘x’	$f(15, 16)$	⊥	⊥	⊥
2	‘y’	⊥	⊥	⊥	⊥
3	⊥	15	⊥	⊥	⊥
4	‘z’	$f(21, \perp)$	⊥	⊥	⊥

$S_1 \sqsupset_{a_3} S_4 :$ a_1	a_2	a_3	a_4	a_5	a_6
1	‘x’	15	‘x’	‘g’	⊥
2	‘y’	⊥	⊥	⊥	⊥
3	⊥	15	‘x’	‘g’	⊥
4	‘z’	21	⊥	⊥	⊥

Fig. 5. The left outerjoin-merge operator using the sources of Figure 4

guarantees that all tuples from one source (S_1 in Figure 5) appear in the result. Wherever possible, they are joined with tuples from the other source. If not possible, the missing values are padded with `null` (⊥). As for the join-merge operator, the left outerjoin-merge is restricted to the two cases shown in Figure 5. Because the right outerjoin-merge operator is basically the same as the left outerjoin-merge, we continue the discussion only with the latter.

3.3 The Full Outerjoin-Merge Operator

We define the full outerjoin-merge operator and continue with the example in Figure 6.

Definition 7 (Full Outerjoin-Merge Operator $\sqcup$) *Let A be the set of attributes in the universal relation. Let $S_i = (A_i)$ and $S_j = (A_j)$ be two data sources with $A_i, A_j \in A$, $A_i \cap A_j \neq \emptyset$, and let $a_k \in A_i \cap A_j$.*

$$S_i \sqcup_{a_k} S_j := (S_i \sqcap_{a_k} S_j) \;\uplus\; (S_i \setminus (S_i \sqcap_{a_k} S_j)[A_i]) \;\uplus\; (S_j \setminus (S_i \sqcap_{a_k} S_j)[A_j])\,.$$

The full outerjoin-merge operator guarantees that every tuple from both sources enters the result. Missing values in attributes of tuples that do not have a matching tuple in the other source are padded with `null` values (⊥). Again, we restrict the operator to the two cases of Figure 6.

3.4 Related work on merge operators

Literature describes several operators similar to the full outerjoin-merge, however none of the authors have sources in mind that possibly are mutually inconsistent. For instance, LaCroix and Pirotte defined a similar operator, the “generalized natural join operator”, denoted $\overset{+}{\bowtie}$ [19]. Our full outerjoin-merge operator differs from their definition in two aspects: First, data conflicts are

$S_1 \sqcup_{a_1} S_2$:	a_1	a_2	a_3	a_4	a_5	a_6
	1	'x'	$f(15, 16)$	$\bot$	$\bot$	$\bot$
	2	'y'	$\bot$	$\bot$	$\bot$	$\bot$
	3	$\bot$	15	$\bot$	$\bot$	$\bot$
	4	'z'	$f(21, \bot)$	$\bot$	$\bot$	$\bot$
	5	$\bot$	17	$\bot$	$\bot$	$\bot$

$S_1 \sqcup_{a_3} S_4$:	a_1	a_2	a_3	a_4	a_5	a_6
	1	'x'	15	'x'	'g'	$\bot$
	2	'y'	$\bot$	$\bot$	$\bot$	$\bot$
	3	$\bot$	15	'x'	'g'	$\bot$
	4	'z'	21	$\bot$	$\bot$	$\bot$
	$\bot$	$\bot$	16	'y'	$\bot$	$\bot$
	$\bot$	$\bot$	17	'x'	'i'	$\bot$

Fig. 6. The union-merge operator using the sources of Figure 4

resolved with a resolution function f. Second, our join is not a natural join; rather, the join predicate contains only one join attribute—the ID. Other names for the full outerjoin are "two-sided outerjoin" (⟗) [20] or the "outer union" (⊐⋈⊏ or ⊎) [21,18]. There, the outer union operation is described as a union between relations that are not union compatible, i.e., that do not have identical attributes. The authors suggest to pad attributes for tuples that have no value with `null` values.

4 Query Answering across Multiple Sources

This section describes the process of query planning, i.e., finding answers to user queries. In a first step we move from user queries against the UR tableau to queries against the relations using the merge operator. In a next step we find a set of execution plans by replacing each relation with a combination of sources that provides data for these relations.

4.1 Translating User Queries

We use the universal relation *UR* for formulating queries against the global schema. Definition 3 defined a user query Q as a set of attributes over *UR*, with attributes possibly carrying selection conditions. When answering user queries, we make assumptions regarding the special semantics of the queries. We assume that users of integrated information systems have—among others—three requirements (R.1, R.2, R.3) and we present three corresponding concessions (C.1, C.2, C.3).

R.1 The user expects only *correct* results, i.e., only tuples where all selection predicates hold true. For example, a user of a search engine expects *only* such Web pages that contain the specified keywords.

C.1 The user accepts tuples with attribute values that are *close* to their selection condition. For example, a user querying for cars with a price lower then \$10,000 might also find agreeable cars for \$10,500 in the result.

R.2 The user expects the result to be *extensionally complete*, i.e., contain *all* correct tuples accessible by the integrated system. For example, A user of a stock information system asking for quotes with a day trade volume of more than 1 billion expects *all* quotes for which this is true.

C.2 The user accepts *extensionally incomplete* answers in the presence of constrained resources. If, for any reason, the extensionally complete answer cannot be returned, the best possible answer should be returned. We define the term "best" later. A user of a search engine usually does not demand the entire result set but is satisfied with, say, ten Web pages. However, the result should consist of those Web pages best matching the keywords of the query.

R.3 The user expects the result to be *intensionally complete*, i.e., contain all attributes of the query and contain non-`null` values in all the attributes. For example, a user of a stock information system asking for stock quotes of certain companies expects tuples where no data is missing.

C.3 The user accepts *intensionally incomplete* answers or answers with *missing values*—a partial answer is better than no answer. A user of a stock information service asking for companies whose stock quotes have risen more than 10 percent today along with a company profile is at least partially satisfied with tuples without the profile. Of course, those tuples for which the profile *is* available should be listed first, but others might still be a helpful part of the result.

For this article we ignore concession C.1 and only return tuples that are correct. Although the integration of fuzzy correctness as suggested by the concession is entirely possible, it is beyond the scope of this article. Our query planning mechanisms take the remaining requirements and concessions into account (i) when deciding which sources should participate in answering the query, and (ii) when deciding how to combine the participating sources.

To reflect these semantic requirements, we transform a query Q against the universal relation into a query Q' against the global schema according to the following definition:

Definition 8 (Translated Query) *Let $Q[A_Q] \leftarrow UR, C$ be a user query where A_Q is the set of requested attributes, and C the set of conditions on these attributes. We obtain the corresponding translated query $Q'(A_Q) \leftarrow body$ in the following way:*

(1) The body of Q' contains all relations of which A_Q contains an attribute.

Conditions C are included unchanged.

(2) The body of Q' additionally contains all necessary relations to make the graph of relations and joins connected.

(3) All ID–foreign ID relationships defined for the relations contained in the body of Q' are turned into one of the four new operators:
   - *full outerjoin-merge ($\sqcup$) if there are no selections on any attribute of either relation,*
   - *left/right outerjoin-merge ($\sqsupset$) if there is a selection on at least one attribute of exactly one relation. The relation with the selection condition is the "left" relation, i.e., unmatched tuples of this relation enter the result,*
   - *join-merge ($\sqcap$) if there are selections on attributes of both relations.*

(4) The precedence order of the operators is first $\sqcup$, then $\sqsupset$, then $\sqcap$. Parentheses are inserted accordingly.

The head of the translated query ensures the intensional completeness (R.3). The first translation rule ensures safety, i.e., all attributes to be returned to the user actually appear in one of the relations in the query. The rule also ensures correctness (R.1) by including all conditions specified by the user. Rule 2 catches cases where users specify attributes of relations that have no ID–foreign ID relationship. Rule 3 captures relationships among relations. Without Rule 3 translations would be possible, where there is no explicit join relationship between two relations of the query. In consequence, users would receive the cross product of the two relations, contradicting correctness R.1. Translation Rule 3 combines the relations of the global schema through merge operators. The relations are combined depending on whether there are selection conditions on them or not. Intuitively, the starting point is the extensional completeness requirement R.2. To include all accessible data, relations should be combined with the full outerjoin-merge operator ($\sqcup$). To ensure correctness (R.1) the operator possibly must be changed to a left outerjoin-merge ($\sqsupset$) or a join-merge ($\sqcap$), depending on the distribution of the conditions.

Note that if the selection condition were applied *after* a full outerjoin-merge, those tuples would be removed from the result anyway. However, with the help of the left outerjoin-merge we allow to push selection conditions to the source. Not only does this pushing improve response time of a plan (we do not have to retrieve the entire source), but typical Web data sources *require* some selection to be pushed. For instance, search engines require at least one keyword as input.

Finally, Rule 4 is necessary to ensure correctness, because a "late" outerjoin-merge could insert tuples conflicting with the conditions. The execution order can be changed later in the query plan, by repeating the selection operators that ensure the conditions.

Given a query translated according to Definition 8, the mediator tries to find plans that generate answers to the query. Executing a plan results in tuples being inserted into *UR*. Intuitively, the purpose of our completeness model of the following sections is to guide the mediator in finding plans that produce a maximal number of tuples, and a maximal number of non-`null` values in those tuples.

Example. Recall the global schema consisting of three relations R_1, R_2, and R_3 of Table 1 (page 6). Consider user queries Q_1 requesting a_2 and a_4 without any selection conditions, Q_2 requesting a_1, a_2, a_4, and a_6 with the condition $a_2 < 10$, and Q_3 requesting a_1, a_2, and a_4 with the conditions $a_2 > 10$ and $a_4 < 40$.

The three queries are translated according to Definition 8:

$$\begin{aligned}
Q_1'(a_2, a_4) &\leftarrow R_1(a_1, a_2, a_3) \sqcup_{a_3} R_2(a_3, a_4, a_5) \\
Q_2'(a_1, a_2, a_4, a_6) &\leftarrow R_1(a_1, a_2, a_3) \sqsupset_{a_3} (R_2(a_3, a_4, a_5) \sqcup_{a_5} R_3(a_5, a_6))\ , \ a_2 < 10 \\
Q_3'(a_1, a_2, a_4) &\leftarrow R_1(a_1, a_2, a_3) \sqcap_{a_3} R_2(a_3, a_4, a_5),\ a_2 > 10,\ a_4 < 40
\end{aligned}$$

□

4.2 Query Planning Across Sources

We describe how a translated query is transformed into a plan containing actual data sources.

Definition 9 (Query Plan) *Let Q' be a translated query against the global schema. We obtain a query plan P for Q' by replacing each relation of the body of Q' with a (possibly empty) set of sources. The sources in the plan are combined by full outerjoin-merge operators. Each source in the set must correspond to the relation the set replaces.*

The set that replaces a relation can include one or more sources or can be empty. In the latter case the result of the query plan is also empty, or at least misses some attributes. Our quality model will recognize these deficiencies and will assign low quality to such plans.

Example. Continuing the example, we regard the five sources $S_1, \ldots, S_5$ of the UR-tableau of Table 1, each providing a certain set of attributes of the universal relation *UR*, some having a selection condition on certain attributes. Consider the query $Q_1'(a_2, a_4) \leftarrow R_1(a_1, a_2, a_3) \sqcup_{a_3} R_2(a_3, a_4, a_5)$ translated from Q_1. The set of all possible query plans for Q_1' is

$$\begin{array}{ll}
P_1(a_2,a_4) \leftarrow \emptyset & P_9(a_2,a_4) \leftarrow S_1 \sqcup_{a_3} S_4 \\
P_2(a_2,a_4) \leftarrow S_1 & P_{10}(a_2,a_4) \leftarrow S_2 \sqcup_{a_3} S_3 \\
P_3(a_2,a_4) \leftarrow S_2 & P_{11}(a_2,a_4) \leftarrow S_2 \sqcup_{a_3} S_4 \\
P_4(a_2,a_4) \leftarrow S_1 \sqcup_{a_1} S_2 & P_{12}(a_2,a_4) \leftarrow S_1 \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4) \\
P_5(a_2,a_4) \leftarrow S_3 & P_{13}(a_2,a_4) \leftarrow S_2 \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4) \\
P_6(a_2,a_4) \leftarrow S_4 & P_{14}(a_2,a_4) \leftarrow (S_1 \sqcup_{a_1} S_2) \sqcup_{a_3} S_3 \\
P_7(a_2,a_4) \leftarrow S_3 \sqcup_{a_3} S_4 & P_{15}(a_2,a_4) \leftarrow (S_1 \sqcup_{a_1} S_2) \sqcup_{a_3} S_4 \\
P_8(a_2,a_4) \leftarrow S_1 \sqcup_{a_3} S_3 & P_{16}(a_2,a_4) \leftarrow (S_1 \sqcup_{a_1} S_2) \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4)
\end{array}$$

Several of these plans are useless. For instance, plans P_1 and P_3 do not return any results, because S_2 neither exports a_2 nor a_4. Many plans, such as P_5 or P_{13}, return values for only one of the two requested attributes. Only a few plans return values for both attributes and only P_{12} and P_{16} return *all* tuples given the available sources. The completeness model of the following sections recognizes these differences by calculating the "completeness" for each of these plans. In Table 3, we foreclose exemplary completeness scores $C(P)$ for the 16 plans. We observe the uselessness of P_1 and P_3 (completeness 0), and we observe that P_{12} and P_{16} have the same (maximal) completeness scores. □

plan	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$C(P)$	0	0.1	0	0.1	0.07	0.12	0.1732	0.17

plan	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}
$C(P)$	0.22	0.07	0.12	0.2732	0.1732	0.17	0.22	0.2732

Table 3
Completeness scores for translated queries

4.3 Related work on query planning

Query planning for information integration has received considerable attention in the database community during the last years. For a survey, see [22] or [23]. Query planning is highly dependent on the way how sources are modelled with respect to the global schema. In the so-called global-as-view (GAV) approach, global relations are modelled as views on information sources (e.g., [24]), while in the local-as-view (LAV) approach, source relations are modelled as views on the global schema (e.g., [25]). Recently, several groups suggested to use a hybrid approach, termed GLAV (e.g., [26]). GAV approaches are extensions of view expansion in centralized databases with the advantage of simple query planning algorithms. In contrast, LAV requires complex algorithms solving the problem of answering queries using views, which is shown to be NP-complete for conjunctive queries and conjunctive view definitions [27]. However, integration systems based on a LAV approach have advantages in terms of maintainability, since changes in sources only require the adaption of a precisely defined and limited set of views, while in a GAV Setting, such changes often require changes in the global schema with side-effects on all source descriptions.

In our model, query planning is simpler than in a most LAV approaches for several reasons. First, we restrict sources (more precisely: source descriptions) to views corresponding to exactly one relation of the global schema. Based on our assumptions of IDs and the way how joins are generated during query translation, problems related to distinguished or non-distinguished variables or contradicting conditions on the same attribute in different views of the same plan cannot occur. Second, translated queries never contain a relation twice. In [27] Levy at al. noticed that the problem of answering queries using views actually has two exponential problems in its core: First, there is an exponential number of potential plans, i.e., combinations of views. Second, testing a plan is itself an NP-complete problem in the length of the query, since it requires a test for query containment. In our model, the second problem is trivial. Clearly, we still have an exponential number of plans in the worst-case, but query containment is linear for queries without repeated predicates.

5 Completeness of Data Sources

For many data sources and many application domains, size is everything: The more tuples and the more attributes a source provides, the more attractive it is to users. For instance, users prefer large search engines, i.e., search engines that have indexed a large number of Web pages, over small search engines. The rationale is that the larger a search engine is, the higher the probability is, that the result the user is looking for has been indexed by the search engine (and therefore appears in the result). Furthermore, users prefer search engines that return more attributes than others, e.g., knowing the *byte size* of a Web page before clicking on the link is advantageous. To include all facets of completeness, we divide the measure into two sub-measures: *Coverage* is a measure for the relative number of tuples a source stores; *density* is a measure for how well the attributes stored at a source are filled with actual (non-`null`) values. We combine the two aspects to an overall *completeness* measure. Using this measure, we are able to assess the ‘usefulness’ of a source or a query plan to answer a query.

We define the coverage of a source as the ratio of the number of real world entities represented by the source, and the size of the universal relation UR as defined next. First, we define the extension of UR, then we define coverage of a data source.

Definition 10 (Extension of *UR*) *Let $S_1, \ldots, S_n$ be the set of all available data sources. Then the extension of UR is $ext(UR) := S_1 \sqcup \cdots \sqcup S_n$. The size of UR is $|UR| := |S_1 \sqcup \cdots \sqcup S_n|$.*

We call the extension of UR the *world.* The world is the set of all tuples that can be obtained through the sources at hand. Thus, we make the closed world assumption. Since $|UR|$ acts only as a normalizing factor, this assumption is not necessary, but simplifies definitions and calculation of the coverage criterion.

Definition 11 (Coverage) *Let S be a data source. We define the* coverage *of S as*

$$c(S) := \frac{|S|}{|UR|}.$$

The coverage scores are in $[0, 1]$ due to the closed world assumption. Intuitively, coverage can be regarded as the probability that an entity of the world is represented in the source.

To make use of coverage scores, one must be able to assess them. Sometimes the sources themselves publish their sizes as a means for advertising their service. However, not always can these figures be trusted. A further possibility is to sample or download the source. If these assessment methods fail, coverage scores can only be estimated.

Estimating the size of the universal relation proves more difficult because of its definition as the full outerjoin of all sources. The more relations are in the global schema, the more the sizes of the individual relation extensions and the degree of overlap play a role. However as explained earlier, $|UR|$ acts only as a normalizing factor, so an exact number is not necessary. The more precise the number is, the more meaningful the absolute coverage scores are, but the coverage score of a source in relation to other sources remains unchanged.

5.2 Density

Data sources often do not export all attributes of the corresponding relation in the global schema. For instance, only a few search engines return the size of Web pages. Also, sources provide attributes they do not completely cover. For instance, sources for company data often do not provide a company profile text for all companies listed. `Null` values are a common phenomenon in WWW data sources. Missing attributes and values result in incomplete results, i.e., tuples with `null` values. Density is a measure for the ratio of non-`null` values to all values provided by a source.

Definition 12 (Attribute density) *Let A_R be the set of attributes of relation R. The density of attribute $a \in A_R$ in source S providing data for R is*

$$d_S(a) := \frac{|\{t \in S | t[a] \neq \bot\}|}{|S|},$$

where t are tuples of S, and $t[a]$ is their attribute value of a.

With Definition 12, an attribute with a non-`null` value for every tuple of the source has an attribute density of 1. An attribute that is not provided by a source has attribute density 0. A source that provides actual values for a certain attribute in every second tuple, has a density score of 0.5 for that attribute.

Definition 13 (Source density) *The density of a source S is the average density over all attributes of the universal relation:*

$$d(S) := \frac{1}{|A|} \sum_{a \in A} d_S(a).$$

The source density of Definition 13 is query-independent. To measure the density of a source for a specific query, we introduce a query-dependent source density that takes into account the attributes of the user query.

Definition 14 (Query-Dependent Density) *The query-dependent density of a source S is the average density over all attributes of query $Q[A_Q]$:*

$$d_Q(S) := \frac{1}{|A_Q|} \sum_{a \in A_Q} d_S(a).$$

Query-dependent density is “fair”, because it counts only those attributes that appear in the query. For notational brevity, we continue with the query-independent density definition, but all results hold for the query-dependent definition as well. For the examples we use query-dependent scores.

The density measure is useful only if it is possible to assess the scores. Like coverage scores, density scores may be assessed in several different ways, depending on the ability and willingness of the data sources to cooperate. In some cases, data sources readily provide the scores. Statements like "We provide reviews for more than 10 percent of all available books" ($d(\text{review}) = 0.1$) or "All search results include a page size" ($d(\text{size}) = 1$) are not uncommon.

5.3 Completeness

Having defined coverage and completeness we combine the two to compute the overall completeness score. The completeness of a data source is the ratio of its amount of data and the potential amount of data of the real world. We understand the potential amount of data of the real world as the number of data items of the completely filled universal relation. This is the product of the number of tuples in the universal relation and the number of attributes of the universal relation.

Definition 15 (Completeness) *Let data source $S = (a_{ij})$ where a_{ij} is the value of the jth attribute of tuple t_i. Then the completeness of source S is defined by*

$$C(S) := \frac{|\{a_{ij} \neq \perp | a_{ij} \in S\}|}{|UR| \cdot |A|}.$$

Alternatively, we may calculate completeness of a data source without actually counting the number of non-`null` values, by using the coverage and density scores of the source:

Theorem 16 *Let S be a data source and let $c(S)$ and $d(S)$ be its coverage and density scores. Then $C(S) = c(S) \cdot d(S)$.*

PROOF. All proofs of theorems and lemmata can be found in the appendix.

Example. Table 4 shows the five sources of our running example with fictitious coverage scores, density scores for each attribute, the overall density score, and—derived from coverage and density—the completeness score. □

5.4 Extensions

Several extensions to our completeness model were omitted for simplicity. In the following we describe, which extensions are possible and how they can be included in the model.

	$c(S)$	$d_S(a_1)$	$d_S(a_2)$	$d_S(a_3)$	$d_S(a_4)$	$d_S(a_5)$	$d_S(a_6)$	$d(S)$	$C(S)$
S_1	0.4	1	0.5	1	0	0	0	2.5/6	1.00/6
S_2	0.8	1	0	1	0	0	0	2.0/6	1.60/6
S_3	0.7	0	0	1	0.2	0	0	1.2/6	0.84/6
S_4	0.3	0	0	1	0.8	1	0	2.8/6	0.84/6
S_5	0.2	0	0	0	0	1	0.9	1.9/6	0.38/6

Table 4
Sources with coverage, density, and completeness scores

Variable coverage scores: Until now, we assumed coverage to be an unchanging global score of a source. In reality, many sources may be partitioned into areas with high coverage and areas with low coverage, e.g., the Fireball search engine concentrates on german Web pages, but also indexes others. There are two approaches to such a situation: (i) The source is modeled by multiple views, one for each partition. The partitioning can be expressed as selection predicates of the view. Each partition receives an individual coverage score. (ii) Some component recognizes, which part of the source is addressed by the query and dynamically adapts coverage scores. For instance, Meng et al. present methods to estimate this number [28]. Both techniques can be combined.

Variable density scores: With the same arguments as for variable coverage scores, density scores can vary depending on the query. The same techniques as described before can be employed.

Attribute weightings: We defined user queries as a subset of the attributes of *UR*, together with selection predicates. Situations arise where one attribute is more important to the user than others. For instance, a user might look for a stock quote with a company profile, and if possible with the address of the company. An attribute weighting in the user query can express these preferences. Attribute weighting seamlessly integrates into our model, as we have shown in [29].

6 Completeness of Merged Results

A system that integrates multiple data sources distributes a user query to multiple data sources following a query plan. To reach our goal of finding the best query plan, we must be able to predict the coverage, density, and completeness scores of a plan. These scores depend on the degree of overlap between data sources. A full outerjoin-merge over two sources with a large overlap returns a smaller result than if the sources had only little overlap. Thus, before showing how to determine completeness of merged results, we analyze different overlap situations.

6.1 Overlap Situations

As discussed earlier, overlap between sources can be extensional (number of common objects) and intensional (number of common attributes). The degree of extensional overlap may vary from no overlap at all to a complete overlap of two sources:

Disjointness: Two sources are disjoint if the set of tuples of UR for which both sources provide data is empty, i.e., there is no overlap. Thus, $S_i \sqcap S_j = \{\}$.

Independence: Two sources are independent if there is no (known) dependency between the tuples of UR for which the sources provide data. That is, there is some coincidental overlap, determined by the size of the sources and the size of the real world they model. Whenever there is no concrete knowledge about overlap, we assume independence.

Quantified overlap: In some occasions the exact degree of overlap is known, i.e., $S_i \sqcap S_j = X$, where the size of X is known. The overlap can be specified by the number of tuples of UR for which both sources provide data.

Containment: One source is contained in another if every tuple of UR for which one source provides data, is also provided data from the other source. Thus, $S_i \sqcup S_j = S_i$ if S_j is contained in S_i. The actual data is not necessarily the same. For instance, one source may provide different attributes than the other. Also, the attribute values may conflict. A special case of containment is equality.

In the following sections we show coverage, density, and completeness calculation in depth for the case of mutually independent sources. We discuss the other cases and the case of mixed overlap situations between sources later. Because we explicitly describe which source exports which attribute, it is not necessary to consider different overlap situations for intensional overlap. Intensional overlap of two sources is the set of attributes both sources export.

At a finer level of granularity—at attribute value level—we make a simplifying assumption: We assume independence of `null` values between two sources, i.e., the probability that a source has a `null` value for an attribute of a certain tuple is independent of the probability that another source has a `null` value for the same attribute of the tuple representing the same real world entity. Also, we assume uniform distribution of attribute values for all attributes.

6.2 Merging Coverage Scores

For coverage calculation of merged results we are interested in the number of tuples in a merged result.

Lemma 17 *Let S_i and S_j be two independent sources. Then*

$$|S_i \sqcap S_j| = \frac{|S_i| \cdot |S_j|}{|UR|} \tag{1}$$

$$|S_i \sqsupset S_j| = |S_i| \tag{2}$$

$$|S_i \sqcup S_j| = |S_i| + |S_j| - |S_i \sqcap S_j| \tag{3}$$

$$|S_i \sqcap S_i| = |S_i| \tag{4}$$

$$|S_i \sqsupset S_i| = |S_i| \tag{5}$$

$$|S_i \sqcup S_i| = |S_i| \tag{6}$$

Notice that we require density of an ID to be 1, i.e, there are no missing values in the ID-attribute. However, because join-merge operators are performed only on ID–foreign ID relationships, the size of the result may be diminished by missing values in the foreign ID-attribute. If a_k is the foreign ID-attribute of S_i and the ID of S_j, the correct score for the result size e.g. of a join-merge operator is

$$|S_i \sqcap_{a_k} S_j| = \frac{|S_i| \cdot |S_j| \cdot d_{S_i}(a_k)}{|UR|}.$$

For simplicity, we require that density of foreign IDs is always 1. To relax this requirement, we can include the factor $d_{S_i}(a_k)$ wherever necessary without changing the properties of our measures.

Given the coverage scores of the individual data sources, we show how to determine the coverage of the merged result.

Theorem 18 *Let S_i and S_j be two independent sources. Then coverage of the merged sources is*

$$c(S_i \sqcap S_j) = c(S_i) \cdot c(S_j) \tag{7}$$

$$c(S_i \sqsupset S_j) = c(S_i) \tag{8}$$

$$c(S_i \sqcup S_j) = c(S_i) + c(S_j) - c(S_i \sqcap S_j) \tag{9}$$

Because the independence relationship is associative, Theorem 18 enables us to calculate coverage of any plan with any number of sources combined with any of the four merge operators.

Example. Consider query plan $P_{12}(a_2, a_4) \leftarrow S_1 \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4)$ of Example 4.2 (Page 17). With the source coverage scores of Table 4 (Page 23) we calculate

$$
\begin{aligned}
c(P_{12}) &= c(S_1 \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4)) \\
&= c(S_1) + c(S_3 \sqcup_{a_3} S_4) - c(S_1 \sqcap_{a_3} (S_3 \sqcup_{a_3} S_4)) \\
&= c(S_1) + c(S_3) + c(S_4) - c(S_3 \sqcap_{a_3} S_4) - c(S_1) \cdot c(S_3 \sqcup_{a_3} S_4) \\
&= c(S_1) + c(S_3) + c(S_4) - c(S_3) \cdot c(S_4) - c(S_1) \cdot (c(S_3) + c(S_4) - c(S_3 \sqcap_{a_3} S_4)) \\
&= c(S_1) + c(S_3) + c(S_4) - c(S_3) \cdot c(S_4) - c(S_1) \cdot c(S_3) - c(S_1) \cdot c(S_4) \\
&\quad + c(S_1) \cdot c(S_3) \cdot c(S_4) \\
&= 0.4 + 0.7 + 0.3 - 0.7 \cdot 0.3 - 0.4 \cdot 0.7 - 0.4 \cdot 0.3 + 0.4 \cdot 0.7 \cdot 0.3 \\
&= 0.874
\end{aligned}
$$

□

To lower response time or data transfer over the net it is often desirable to perform algebraic reordering of plans. The following properties show which reorderings are possible without changing the overall coverage of a plan. We propose to perform such a reordering *after* finding a plan with good completeness (see Section 7).

Theorem 19 *Coverage is commutative, associative, and distributive for* $\sqcap$ *and* $\sqcup$*, and associative for* $\sqsupset$ *for independent sources:*

$$c(S_i \sqcap S_j) = c(S_j \sqcap S_i) \tag{10}$$

$$c(S_i \sqcup S_j) = c(S_j \sqcup S_i) \tag{11}$$

$$c(S_i \sqcap S_i) = c(S_i) \tag{12}$$

$$c(S_i \sqsupset S_i) = c(S_i) \tag{13}$$

$$c(S_i \sqcup S_i) = c(S_i) \tag{14}$$

$$c((S_i \sqcap S_j) \sqcap S_k) = c(S_i \sqcap (S_j \sqcap S_k)) \tag{15}$$

$$c((S_i \sqsupset S_j) \sqsupset S_k) = c(S_i \sqsupset (S_j \sqsupset S_k)) \tag{16}$$

$$c((S_i \sqcup S_j) \sqcup S_k) = c(S_i \sqcup (S_j \sqcup S_k)) \tag{17}$$

$$c(S_i \sqcap (S_j \sqcup S_k)) = c((S_i \sqcap S_j) \sqcup (S_i \sqcap S_k)) \tag{18}$$

$$c(S_i \sqcup (S_j \sqcap S_k)) = c((S_i \sqcup S_j) \sqcap (S_i \sqcup S_k)) \tag{19}$$

6.3 Merging Density Scores

As discussed in Section 2.2.2, a tuple in the combined result of two sources has a value in an attribute if either one or both sources provide some value. To compute the density of merged results we are first interested in the number of non-`null` values in a merged result.

Lemma 20 *Let* S_i *and* S_j *be two independent sources. We abbreviate* $|\{t \in$

$S_i | t[a] \neq \bot\}|$ with $|t_{S_i}[a] \neq \bot|$. Then for any attribute $a \in A$

$$|t_{S_i \sqcap S_j}[a] \neq \bot| = \frac{|t_{S_i}[a] \neq \bot| \cdot |S_j|}{|UR|} + \frac{|t_{S_j}[a] \neq \bot| \cdot |S_i|}{|UR|} - \frac{|t_{S_i}[a] \neq \bot| \cdot |t_{S_j}[a] \neq \bot|}{|UR|} \tag{20}$$

$$|t_{S_i \sqsupset S_j}[a] \neq \bot| = |t_{S_i}[a] \neq \bot| + \frac{|t_{S_j}[a] \neq \bot| \cdot |S_i|}{|UR|} - \frac{|t_{S_i}[a] \neq \bot| \cdot |t_{S_j}[a] \neq \bot|}{|UR|} \tag{21}$$

$$|t_{S_i \sqcup S_j}[a] \neq \bot| = |t_{S_i}[a] \neq \bot| + |t_{S_j}[a] \neq \bot| - \frac{|t_{S_i}[a] \neq \bot| \cdot |t_{S_j}[a] \neq \bot|}{|UR|} \tag{22}$$

As opposed to merging coverage scores, density scores are merged at attribute level. Theorem 21 shows how to calculate attribute density of merged attributes and Theorem 22 shows how to calculate merged source density.

Theorem 21 *Let S_i and S_j be two independent sources. Then the density of attribute a in the merged result is*

$$d_{S_i \sqcap S_j}(a) = d_{S_i}(a) + d_{S_j}(a) - d_{S_i}(a) \cdot d_{S_j}(a) \tag{23}$$

$$d_{S_i \sqsupset S_j}(a) = d_{S_i}(a) + d_{S_j}(a) c(S_j) - d_{S_i}(a) d_{S_j}(a) c(S_j) \tag{24}$$

$$d_{S_i \sqcup S_j}(a) = \frac{d_{S_i}(a) \cdot c(S_i)}{c(S_i \sqcup S_j)} + \frac{d_{S_j}(a) \cdot c(S_j)}{c(S_i \sqcup S_j)} - \frac{d_{S_i}(a) \cdot d_{S_j}(a) \cdot c(S_i \sqcap S_j)}{c(S_i \sqcup S_j)} \tag{25}$$

The density of an attribute of an outerjoin-merged result depends on coverage scores, while that of the join-merged result does not. Intuitively, this is true because in join-merge operations we regard only the tuples within the overlap. For density measurements we do not care how much overlap there is. For outerjoins, we regard all tuples and coverage scores are needed to determine how much overlap there is, i.e., how many non-`null` values appear in both sources.

Theorem 22 *Let S_i and S_j be two independent sources. Given the density scores for all attributes of a merged result, the density of the entire result is*

$$d(S_i \sqcap S_j) = \frac{1}{|A|} \sum_{a \in A} d_{S_i \sqcap S_j}(a) \tag{26}$$

$$d(S_i \sqsupset S_j) = \frac{1}{|A|} \sum_{a \in A} d_{S_i \sqsupset S_j}(a) \tag{27}$$

$$d(S_i \sqcup S_j) = \frac{1}{|A|} \sum_{a \in A} d_{S_i \sqcup S_j}(a) \tag{28}$$

Because the independence relationship is associative, Theorem 22 enables us to calculate the density of any plan with any number of sources combined with any of the four merge operators.

Lemma 23 *Attribute density is commutative, associative, and distributive for $\sqcap$ and $\sqcup$ for independent sources.*

$$d_{S_i \sqcap S_j}(a) = d_{S_j \sqcap S_i}(a) \quad (29)$$
$$d_{S_i \sqcup S_j}(a) = d_{S_j \sqcup S_i}(a) \quad (30)$$
$$d_{(S_i \sqcap S_j) \sqcap S_k}(a) = d_{S_i \sqcap (S_j \sqcap S_k)}(a) \quad (31)$$
$$d_{(S_i \sqcup S_j) \sqcup S_k}(a) = d_{S_i \sqcup (S_j \sqcup S_k)}(a) \quad (32)$$
$$d_{S_i \sqcap (S_j \sqcup S_k)}(a) = d_{(S_i \sqcap S_j) \sqcup (S_i \sqcap S_k)}(a) \quad (33)$$
$$d_{S_i \sqcup (S_j \sqcap S_k)}(a) = d_{(S_i \sqcup S_j) \sqcap (S_i \sqcup S_k)}(a) \quad (34)$$

Theorem 24 *Source density is commutative, associative, and distributive for $\sqcap$ and $\sqcup$ for independent sources.*

$$d(S_i \sqcap S_j) = d(S_j \sqcap S_i) \quad (35)$$
$$d(S_i \sqcup S_j) = d(S_j \sqcup S_i) \quad (36)$$
$$d((S_i \sqcap S_j) \sqcap S_k) = d(S_i \sqcap (S_j \sqcap S_k)) \quad (37)$$
$$d((S_i \sqcup S_j) \sqcup S_k) = d(S_i \sqcup (S_j \sqcup S_k)) \quad (38)$$
$$d(S_i \sqcap (S_j \sqcup S_k)) = d((S_i \sqcap S_j) \sqcup (S_i \sqcap S_k)) \quad (39)$$
$$d(S_i \sqcup (S_j \sqcap S_k)) = d((S_i \sqcup S_j) \sqcap (S_i \sqcup S_k)) \quad (40)$$

6.4 Merging Completeness Scores

As for single sources, we calculate completeness of a plan using the coverage and density scores of that plan:

Theorem 25 *Let P be a query plan. Then completeness of P is $C(P) = c(P) \cdot d(P)$.*

Example. Recall the 16 alternative plans for the translated user query

$$Q_1'(a_2, a_4) \leftarrow R_1(a_1, a_2, a_3) \sqcup_{a_3} R_2(a_3, a_4, a_5)$$

of the example on Page 17. Table 5 shows these plans with their merged scores with respect to coverage, attribute density, density, and completeness. To answer a user query with maximal completeness, an integrated system can construct all plans, calculate their completeness, and choose a plan with maximal completeness for execution.

A first observation is that source S_2 does not contribute to plan completeness in any way, because it does not export a_2. Comparing plans P_7 and P_{13}, we

Plan P	$c(P)$	$d_P(a_2)$	$d_P(a_4)$	$d_{Q'_1}(P)$	$C(P)$
$P_1(a_2, a_4) \leftarrow \emptyset$	0	0	0	0	0
$P_2(a_2, a_4) \leftarrow S_1$	0.4	0.5	0	0.25	0.1
$P_3(a_2, a_4) \leftarrow S_2$	0.8	0	0	0	0
$P_4(a_2, a_4) \leftarrow S_1 \sqcup_{a_1} S_2$	0.88	$0.2\overline{27}$	0	$0.1\overline{136}$	0.1
$P_5(a_2, a_4) \leftarrow S_3$	0.7	0	0.2	0.1	0.07
$P_6(a_2, a_4) \leftarrow S_4$	0.3	0	0.8	0.4	0.12
$P_7(a_2, a_4) \leftarrow S_3 \sqcup_{a_3} S_4$	0.79	0	0.4385	0.2192	0.1732
$P_8(a_2, a_4) \leftarrow S_1 \sqcup_{a_3} S_3$	0.82	0.2439	0.1707	0.2073	0.17
$P_9(a_2, a_4) \leftarrow S_1 \sqcup_{a_3} S_4$	0.58	0.3448	0.4138	0.3793	0.22
$P_{10}(a_2, a_4) \leftarrow S_2 \sqcup_{a_3} S_3$	0.94	0	0.1489	0.0745	0.07
$P_{11}(a_2, a_4) \leftarrow S_2 \sqcup_{a_3} S_4$	0.86	0	0.2791	0.1396	0.12
$P_{12}(a_2, a_4) \leftarrow S_1 \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4)$	0.874	0.2288	0.3964	0.3126	0.2732
$P_{13}(a_2, a_4) \leftarrow S_2 \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4)$	0.958	0	0.3616	0.1808	0.1732
$P_{14}(a_2, a_4) \leftarrow (S_1 \sqcup_{a_1} S_2) \sqcup_{a_3} S_3$	0.964	0.2075	0.1452	0.1764	0.17
$P_{15}(a_2, a_4) \leftarrow (S_1 \sqcup_{a_1} S_2) \sqcup_{a_3} S_4$	0.916	0.2183	0.262	0.2402	0.22
$P_{16}(a_2, a_4) \leftarrow (S_1 \sqcup_{a_1} S_2) \sqcup_{a_3} (S_3 \sqcup_{a_3} S_4)$	0.9748	0.2052	0.3554	0.2803	0.2732

Table 5
Plans for Q_1 with merged coverage, density, and completeness scores

recognize that P_{13} corresponds to P_7 with S_2 added. P_{13} has a higher coverage, because it involves more sources, but it has a lower density, because the additional source contributes only `null` values. Their overall completeness is the same, which is what we expected, because the actual number of non-`null` values is the same for both plans.

Another observation is that completeness increases monotonously with the number of sources for the full outerjoin-merge, i.e., if a source is added to a plan, the plan's completeness does not decrease. Consequently, plan P_{16} is the most complete, but potentially the most expensive, because it accesses the most number of sources. Coverage scores are also monotonous, however, density scores are not: Consider again plan P_7. Adding S_2 to P_7, we arrive at plan P_{13}, which has a lower density score. □

We now have a comprehensive model for source and plan completeness. We are able to compare sources with one another and we are able to compare combinations of sources (plans) with one another. Completeness is a measure for the amount of data to expect from a source or from a combination of sources. With the help of this model we can find an optimal set of sources to answer a user query. Before glancing at possible algorithms in Section 7 we describe the calculation of coverage and density under overlap situations other than independence.

6.5 Other Overlap Situations

Up until now, we assumed mutual independence of sources. In the following, we show the consequences of relaxing this assumption. In general, things are more complicated, because the formulas for computing the joint values must take the different overlaps into account. Recall the different overlap profiles: Sources may be disjoint, i.e., $S_i \sqcap S_j = \{\}$, contained, i.e., $S_i \sqcup S_j = S_i$, or there can be quantified overlap between sources, i.e., $S_i \sqcap S_j = X$. Tables 6 and 7 summarize coverage and density calculation for two merged sources with respect to all four alternatives.

	$S_i \sqcap S_j = \{\}$	$S_i \sqcap S_j = X$
$c(S_i \sqcap S_j)$	0	$\lvert X\rvert/\lvert UR\rvert$
$c(S_i \sqsupset S_j)$	$c(S_i)$	$c(S_i)$
$c(S_i \sqcup S_j)$	$c(S_i) + c(S_j)$	$c(S_i) + c(S_j) - \lvert X\rvert/\lvert UR\rvert$
$d_{S_i \sqcap S_j}(a)$	undefined	$d_{S_i}(a)c(S_i) + d_{S_j}(a)c(S_j)$ $-\frac{d_{S_i}(a)c(S_i)d_{S_j}(a)c(S_j)}{c(S_i \sqcap S_j)}$
$d_{S_i \sqsupset S_j}(a)$	$d_{S_i}(a)$	$d_{S_i}(a) + \frac{d_{S_j}(a)c(S_j)c(S_i \sqcap S_j)}{c(S_i \sqsupset S_j)}$ $-d_{S_i}(a)d_{S_j}(a)c(S_i \sqcap S_j)$
$d_{S_i \sqcup S_j}(a)$	$\frac{d_{S_i}(a)c(S_i)+d_{S_j}(a)c(S_j)}{c(S_i \sqcup S_j)}$	$\frac{d_{S_i}(a)\cdot c(S_i)}{c(S_i \sqcup S_j)} + \frac{d_{S_j}(a)\cdot c(S_j)}{c(S_i \sqcup S_j)}$ $-\frac{d_{S_i}(a)\cdot d_{S_j}(a)\cdot c(S_i \sqcap S_j)}{c(S_i \sqcup S_j)}$

Table 6
Coverage and density measures for different overlap situations

6.6 Related work on completeness

The information quality literature has defined completeness in general, non-quantified terms, such as "the extent to which data is not missing and is of sufficient breadth and depth for the task at hand" [30]. Further related work on our completeness measure draws from two areas: Determining the completeness of single data sources, and determining the completeness of combined sources, in particular, join size estimation.

Determining the "size" of a data source has become a problem only recently,

	$S_i \sqcup S_j = S_i$	$S_i \sqcup S_j = S_j$
$c(S_i \sqcap S_j)$	$c(S_j)$	$c(S_i)$
$c(S_i \sqsupset S_j)$	$c(S_i)$	$c(S_i)$
$c(S_i \sqcup S_j)$	$c(S_i)$	$c(S_j)$
$d_{S_i \sqcap S_j}(a)$	$c(S_i)d_{S_i}(a) + d_{S_j}(a)$ $-c(S_i)d_{S_i}(a)d_{S_j}(a)$	$d_{S_i}(a) + c(S_j)d_{S_j}(a)$ $-c(S_j)d_{S_i}(a)d_{S_j}(a)$
$d_{S_i \sqsupset S_j}(a)$	$d_{S_i}(a) + d_{S_j}(a)$ $-c(S_i)d_{S_i}(a)d_{S_j}(a)$	$d_{S_i}(a) + c(S_j)d_{S_j}(a)$ $-c(S_j)d_{S_i}(a)d_{S_j}(a)$
$d_{S_i \sqcup S_j}(a)$	$d_{S_i}(a) + d_{S_j}(a)$ $-c(S_i)d_{S_i}(a)d_{S_j}(a)$	$d_{S_i}(a) + d_{S_j}(a)$ $-c(S_j)d_{S_i}(a)d_{S_j}(a)$

Table 7
Coverage and density measures for different overlap situations

when such metadata was desired for autonomous sources of unknown size, such as typical WWW data sources. There are yet few projects striving to model or determine the size of Web data sources. Chen et al., who address query processing in the WWW, mention the quality criteria “size of result” and “number of documents accessed”, but they neither define them, nor point out the difference between the two [31]. Also, the authors do not integrate the two criteria into a general value model as we do. Motro and Rakov define a “completeness” criterion, which matches our coverage criterion [32]. Motro suggests to add “completeness assertions” as accompanying information in the query result, adding more meaning to the result [33]. Completeness assertions are statements, such as “the data contains *all* recordings on the CBS label”. These assertions are aggregated along query plans in a similar fashion to our coverage and density scores. Thus, the author can give qualitative statements about the completeness of results, but no quantitative statements as we do. Apart from a note on *column completeness* in [30], to the best of our knowledge, the density criterion as we define it, has never been addressed in literature before, even though missing attribute values are all too common.

Calculating or predicting join result sizes is an important technique for cost-based query optimization in DBMS. In general terms, the size of a join is the size of the cross-product of the two relations in the join, multiplied with a *selectivity factor*. Selectivity factors are statistical values stored in the data dictionary of the DBMS. Many research efforts tackled the problem of join size estimation [34–36]; Mannino et al. give a survey on the suggested statistical values to store, how to maintain them, and how to use them to predict the result sizes of various database operations [37]. Most projects make the same simplifying assumptions as we do: uniformity of attribute values and independence of attribute values [38].

Florescu et al. attempt to describe quantitatively the content of distributed autonomous document sources using probabilistic measures [39]. In their model, the authors calculate two values: "Coverage" of data sources, determining the probability that a matching document is found in the source, and "overlap" between two data sources, determining the probability that an arbitrary document is found in both sources. These probabilities are calculated with the help of word-count statistics. Their coverage measure is similar to the *precision* measure of the information retrieval field and determines the query dependent usefulness of a source. Their overlap measure expresses ideas similar to ours, but the authors do not consider different types of overlap, such as independence or disjointness. Rather, it is a measure solely based on probability.

7 Information Integration with Completeness Measures

After presenting the completeness model we now discuss its usage in integrating information systems: Completeness determines the usefulness of a source or a plan to answer a user query. Therefore, the completeness measure is a valuable tool to guide query planning in the same way as a cost model guides query optimization in traditional databases.

7.1 Finding Best Plans

Any system integrating data sources answers user queries by searching for one or more query plans for the query, executing the plans, and integrating the results for the user. In general, more than one plan may contribute to the overall response in some way. Common approaches to query planning search for all those plans, execute them all, and provide a complete response to the user. Such an approach does not consider any cost involved in accessing the different sources.

To reflect reality, we must restrict plan execution to only a few, "best" plans. In [40] and [41] we presented two algorithms to find the top N plans in the search space. If we assign some cost—monetary cost, response time, or any other punishment function—to each source, the problem is to find the set of best plans whose combined cost does not exceed a cost limit given by the user.

However, such an approach yields only locally optimal solutions: Given the set of best plans, their combination is not necessarily the best. For instance, the plans can all be quite similar and not complement each other. To obtain globally optimal solutions, one must search for the best set of plans instead of for the set of best plans. Our outerjoin operators and our completeness model aid in finding the best combination. In fact, plans as we defined them in Definition 9 already present combinations of conventional plans. The search space of our approach is the power set of the set of available sources. It can be traversed using any known search algorithm such as exhaustive search, greedy search, etc. each with the known advantages and disadvantages in runtime complexity and optimality guarantees.

7.2 Algebraic Reordering

After finding the optimal subset of sources to query we now move to the conventional paradigm of optimizing a plan: Using traditional optimization techniques we can perform algebraic transformations of the plan to improve response time without changing the result.

Because the result of query plans remains the same after reordering, we require the completeness score of a plan to remain the same as well. Completeness is a measure for the plan result and not for how the result is obtained. The properties of $\sqcap$ and $\sqcup$ proven in Theorems 19 and 24 guarantee unchanging plan completeness under a number of reorderings. The properties for $\sqsupset$ are more limited; Galindo-Legaria and Rosenthal discuss outerjoin reordering in greater detail [20].

Example. Post-optimizing a plan may have great effect on plan cost. Consider plan P_{30} for query Q'_3 of the example:

$$P_{30}(a_1, a_2, a_4) \leftarrow (S_1 \sqcup_{a_1} S_2) \sqcap_{a_3} S_3,\ a_2 > 10,\ a_4 < 40.$$

Assume that all selections can be pushed to the sources, i.e., we can retrieve from S_3 only those tuples where $a_4 < 40$, etc. Assume further, that the selection condition $a_2 > 10$ is not very selective and that the selection condition $a_4 < 40$ is very selective. If the plan were executed as is, a huge intermediate result of $(S_1 \sqcup_{a_1} S_2)$ would be created, of which only few tuples enter the final result after the join-merge with the small intermediate result of S_3. Using Theorem 19 we can transform plan P_{30} to

$$P'_{30}(a_1, a_2, a_4) \leftarrow (S_3 \sqcap_{a_3} S_1) \sqcup_{a_1} (S_3 \sqcap_{a_3} S_2),\ a_2 > 10,\ a_4 < 40.$$

Using this plan, we can first retrieve from S_3 only those tuples where $a_4 < 40$ and then push their IDs (attribute a_3) as an additional selection condition to sources S_1 and S_2. The overall number of tuples retrieved over the net decreases dramatically, saving time and possibly money. □

8 Conclusion

There is a new quality to the planning of user queries against multiple, autonomous, integrated Web data sources. Among many other difficulties such as schema conflicts and heterogeneous interfaces, the Web sources store overlapping data, have different sizes, and have many missing values. Also, user-demands towards Web sources are different than towards traditional databases. In this article, we have addressed these new aspects of query answering. We presented a data model for Web data sources based on the universal relation. Our completeness model measures coverage and density of the sources to determine which ones are most appropriate to answer queries. Since multiple sources are combined to a common response, we presented new merge operators to perform this combination and extended the completeness model to measure the completeness of combinations of sources. Finally, we used the model to evaluate the completeness of query plans and showed how the measure can guide query planning.

Future work will touch several aspects. We have already sketched several possible extensions to both our query model and our completeness model. Further research will be spent with the development of the algorithms, of which we plan to implement and test different variations. Finally we plan to extend our model to criteria beyond completeness. In other work, we have argued that completeness is just one aspect among many other information quality criteria like availability or timeliness. We plan to define measures for these criteria in the same way as we did for coverage and density. The ideas presented in this article have been implemented within a meta-search engine prototype. We are continuing this effort and, apart from the completeness measure, we are including several other quality dimensions, such as timeliness and availability.

Acknowledgements. This research was partly supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316) and the German Ministry for Education and Research (BMBF grant).

References

[1] G. Wiederhold, Mediators in the architecture of future information systems, IEEE Computer 25(3) (1992) 38–49.

[2] G. D. Michelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopoulos, M. Papazoglou, K.Pohl, J.Schmidt, C.Woo, E.Yu, Cooperative information systems: A manifesto, in: M. P. Papazoglou, G. Schlageter (Eds.), Cooperative Information Systems: Trends and Directions, Academic Press, 1997.

[3] M. T. Roth, P. M. Schwarz, Don't scrap it, wrap it! A wrapper architecture for legacy data sources, in: Proceedings of the International Conference on Very Large Databases (VLDB), 1997, pp. 266–275.

[4] A. Sahuguet, F. Azavant, Building light-weight wrappers for legacy Web datasources using W4F, in: Proceedings of the International Conference on Very Large Databases (VLDB), 1999, pp. 738–741.

[5] D. Maier, J. D. Ullman, M. Y. Vardi, On the foundations of the universal relation model, ACM Transactions on Database Systems (TODS) 9(2) (1984) 283–308.

[6] R. Hull, Managing semantic heterogeneity in databases: A theoretical perspective, in: Proceedings of the Symposium on Principles of Database Systems (PODS), Tuscon, Arizona, 1997, pp. 51–61.

[7] A. Gupta, Some data integration and database issues in e-commerce (and world peace), Invited talk at the International Conference on Extending Database Technology (EDBT) (2000).

[8] D. Bitton, D. J. DeWitt, Duplicate record elimination in large data files, ACM Transactions on Database Systems (TODS) 8(2) (1983) 255–265.

[9] H. Newcombe, Handbook of Record Linkage, Oxford University Press, Oxford, UK, 1988.

[10] M. Neiling, H.-J. Lenz, Data integration by means of object identification in information systems, in: Proceedings of European Conference on Information Systems, Vienna, Austria, 2000.

[11] Y. Papakonstantinou, S. Abiteboul, H. Garcia-Molina, Object fusion in mediator systems, in: Proceedings of the International Conference on Very Large Databases (VLDB), Bombay, India, 1996, pp. 413–424.

[12] C. Yu, W. Meng, Principles of database query processing for advanced applications, Morgan Kaufmann, San Francisco, CA, USA, 1998.

[13] F. Naumann, M. Häussler, Declarative data merging with conflict resolution, in: Proceedings of the International Conference on Information Quality (IQ), Cambridge, MA, 2002.

[14] The MetaCrawler meta-search engine, `www.metacrawler.com`.

[15] The AltaVista search engine, `www.altavista.com`.

[16] The Google search engine, `www.google.com`.

[17] Web.de, a german search engine, `www.web.de`.

[18] E. Codd, Extending the relational database model to capture more meaning, ACM Transactions on Database Systems (TODS) 4(4) (1979) 397–434.

[19] M. LaCroix, A. Pirotte, Generalized joins, SIGMOD Record 8(3) (1976) 14–15.

[20] C. Galindo-Legaria, A. Rosenthal, Outerjoin simplification and reordering for query optimization, ACM Transactions on Database Systems (TODS) 22(1) (1997) 43–74.

[21] R. Elmasri, S. B. Navathe, Fundamentals of Database Systems, 2nd Edition, Benjamin/Cummings Publishing Company, Redwood City, 1994.

[22] J. D. Ullman, Information integration using logical views, in: Proceedings of the International Conference on Database Theory (ICDT), Delphi, Greece, 1997, pp. 19–40.

[23] M. Lenzerini, Data integration: A theoretical perspective, in: Proceedings of the Symposium on Principles of Database Systems (PODS), Madison, WN, 2002, pp. 233–246.

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

[25] A. Y. Levy, A. Rajaraman, J. J. Ordille, Querying heterogeneous information sources using source descriptions, in: Proceedings of the International Conference on Very Large Databases (VLDB), Bombay, India, 1996, pp. 251–262.

[26] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa, Data exchange: Semantics and query answering, in: Proceedings of the International Conference on Database Theory (ICDT), Siena, Italy, 2003.

[27] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava, Answering queries using views, in: Proceedings of the Symposium on Principles of Database Systems (PODS), San Jose, CA, 1995, pp. 95–104.

[28] W. Meng, K.-L. Liu, C. T. Yu, W. Wu, N. Rishe, Estimating the usefulness of search engines, in: Proceedings of the International Conference on Data Engineering (ICDE), Sydney, Australia, 1999, pp. 146–153.

[29] F. Naumann, U. Leser, Cooperative query answering with density scores, in: Proceedings of the International Conference on Management of Data (COMAD), Pune, India, 2000.

[30] L. Pipino, Y. Lee, R. Wang, Data quality assessment, Communications of the ACM 4 (2002) 211–218.

[31] Y. Chen, Q. Zhu, N. Wang, Query processing with quality control in the World Wide Web, World Wide Web 1(4) (1998) 241–255.

[32] A. Motro, I. Rakov, Estimating the quality of databases, in: Proceedings of the International Conference on Flexible Query Answering Systems (FQAS), Springer Verlag, Roskilde, Denmark, 1998, pp. 298–307.

[33] A. Motro, Completeness information and its application to query processing, in: Proceedings of the International Conference on Very Large Databases (VLDB), Kyoto, 1986, pp. 170–178.

[34] A. Rosenthal, Note on the expected size of a join, SIGMOD Record 11(4) (1981) 19–25.

[35] D. Gardy, C. Puech, On the effects of join operations on relation sizes, ACM Transactions on Database Systems (TODS) 14(4) (1989) 574–603.

[36] A. Swami, K. B. Schiefer, On the estimation of join result sizes, in: Proceedings of the International Conference on Extending Database Technology (EDBT), Vol. 779 of LNCS, Springer Verlag, Cambridge, UK, 1994, pp. 287–300.

[37] M. V. Mannino, P. Chu, T. Sager, Statistical profile estimation in database systems, ACM Computing Surveys 20(3) (1988) 191–221.

[38] S. Christodoulakis, Implications of certain assumptions in database performance evaluation, ACM Transactions on Database Systems (TODS) 9(2) (1984) 163–186.

[39] D. Florescu, D. Koller, A. Levy, Using probabilistic information in data integration, in: Proceedings of the International Conference on Very Large Databases (VLDB), Athens, Greece, 1997, pp. 216–225.

[40] F. Naumann, U. Leser, J.-C. Freytag, Quality-driven integration of heterogenous information systems, in: Proceedings of the International Conference on Very Large Databases (VLDB), Edinburgh, UK, 1999, pp. 447–458.

[41] U. Leser, F. Naumann, Query planning with information quality bounds, in: Proceedings of the International Conference on Flexible Query Answering Systems (FQAS), Advances in Soft Computing, Springer Verlag, Warsaw, Poland, 2000.

A Proofs of Lemmata and Theorems

Theorem 16. Let S be a data source and let $c(S)$ and $d(S)$ be its coverage and density scores, respectively. Then $C(S) = c(S) \cdot d(S)$.

PROOF.

$$\begin{aligned}
C(S) &:= \frac{|\{a_{ij} \neq \bot | a_{ij} \in S\}|}{|UR| \cdot |A|} && \text{(Definition 15)} \\
&= \frac{|S| \cdot \sum_{a \in A} d_S(a)}{|UR| \cdot |A|} && \text{(Definition 12)} \\
&= \frac{|S|}{|UR|} \cdot \frac{1}{|A|} \cdot \sum_{a \in A} d_S(a) \\
&= c(S) \cdot d(S) && \text{(Definitions 11 and 13)}
\end{aligned}$$

Lemma 17. Let S_i and S_j be two independent sources. Then

$$|S_i \sqcap S_j| = \frac{|S_i| \cdot |S_j|}{|UR|} \qquad \text{(A.1)} \qquad\qquad |S_i \sqcap S_i| = |S_i| \qquad \text{(A.4)}$$

$$|S_i \sqsupset S_j| = |S_i| \qquad \text{(A.2)} \qquad\qquad |S_i \sqsupset S_i| = |S_i| \qquad \text{(A.5)}$$

$$|S_i \sqcup S_j| = |S_i| + |S_j| - |S_i \sqcap S_j| \qquad \text{(A.3)} \qquad\qquad |S_i \sqcup S_i| = |S_i| \qquad \text{(A.6)}$$

PROOF. (A.1) According to Definition 5 of the join-merge operator, $S_i \sqcap S_j$ contains only tuples that are represented both in S_i and S_j. The other conditions in Definition 5 only concern construction of the tuple in the result and do not influence the number of tuples. For each real world entity the probability that it is represented in S_i is $\frac{|S_i|}{|UR|}$ and that it is represented in S_j is $\frac{|S_j|}{|UR|}$. Because of independence, the probability that it is represented in both is $\frac{|S_i|}{|UR|} \cdot \frac{|S_j|}{|UR|}$. Because there are $|UR|$ possible entities, we have an overall number of $|UR| \cdot \frac{|S_i|}{|UR|} \cdot \frac{|S_j|}{|UR|}$ tuples.

(A.2) follows from (A.1) and Definition 6.

(A.3) follows from (A.1) and Definition 7.

(A.4) follows from Definition 5 because $S \sqcap S = S$.

(A.5) follows from Definition 6 and (A.4).

(A.6) follows from Definition 7 and (A.4).

Theorem 18. Let S_i and S_j be two independent sources. Then coverage of the merged sources is

$$c(S_i \sqcap S_j) = c(S_i) \cdot c(S_j) \tag{A.7}$$

$$c(S_i \sqsupset S_j) = c(S_i) \tag{A.8}$$

$$c(S_i \sqcup S_j) = c(S_i) + c(S_j) - c(S_i \sqcap S_j) \tag{A.9}$$

PROOF.

$$(A.7) \qquad c(S_i \sqcap S_j) = \frac{|S_i \sqcap S_j|}{|UR|} \underset{\text{(Lemma 17)}}{=} \frac{|S_i| \cdot |S_j|}{|UR| \cdot |UR|} = c(S_i) \cdot c(S_j)$$

$(A.8)$ follows immediately from Lemma 17.

$$\begin{aligned}(A.9) \qquad c(S_i \sqcup S_j) &= \frac{|S_i \sqcup S_j|}{|UR|} \underset{\text{(Lemma 17)}}{=} \frac{|S_i| + |S_j| - |S_i \sqcap S_j|}{|UR|} \\ &= \frac{|S_i|}{|UR|} + \frac{|S_j|}{|UR|} - \frac{|S_i \sqcap S_j|}{|UR|} = c(S_i) + c(S_j) - c(S_i \sqcap S_j)\end{aligned}$$

Theorem 19. Coverage is commutative, associative, and distributive for $\sqcap$ and $\sqcup$, and associative for $\sqsupset$ for independent sources:

$$c(S_i \sqcap S_j) = c(S_j \sqcap S_i) \tag{A.10}$$

$$c(S_i \sqcup S_j) = c(S_j \sqcup S_i) \tag{A.11}$$

$$c(S_i \sqcap S_i) = c(S_i) \tag{A.12}$$

$$c(S_i \sqsupset S_i) = c(S_i) \tag{A.13}$$

$$c(S_i \sqcup S_i) = c(S_i) \tag{A.14}$$

$$c((S_i \sqcap S_j) \sqcap S_k) = c(S_i \sqcap (S_j \sqcap S_k)) \tag{A.15}$$

$$c((S_i \sqsupset S_j) \sqsupset S_k) = c(S_i \sqsupset (S_j \sqsupset S_k)) \tag{A.16}$$

$$c((S_i \sqcup S_j) \sqcup S_k) = c(S_i \sqcup (S_j \sqcup S_k)) \tag{A.17}$$

$$c(S_i \sqcap (S_j \sqcup S_k)) = c((S_i \sqcap S_j) \sqcup (S_i \sqcap S_k)) \tag{A.18}$$

$$c(S_i \sqcup (S_j \sqcap S_k)) = c((S_i \sqcup S_j) \sqcap (S_i \sqcup S_k)) \tag{A.19}$$

PROOF. The proofs of (A.10) – (A.14) are trivial given Lemma 17. For the rest, we apply the definitions and algebraic transformations:

$$(A.15): \quad \begin{aligned} c((S_i \sqcap S_j) \sqcap S_k) &= (c(S_i) \cdot c(S_j)) \cdot c(S_k) = c(S_i) \cdot (c(S_j) \cdot c(S_k)) \\ &= c(S_i \sqcap (S_j \sqcap S_k)) \end{aligned}$$

$$(A.16): \quad c((S_i \sqsupset S_j) \sqsupset S_k) = c(S_i \sqsupset S_j) = c(S_i) = c(S_i \sqsupset (S_j \sqsupset S_k))$$

$$(A.17): \quad \begin{aligned} c((S_i \sqcup S_j) \sqcup S_k) &= c(S_i) + c(S_j) + c(S_k) - c(S_j)c(S_k) - c(S_i)c(S_k) \\ &\quad - c(S_i)c(S_j) + c(S_i)c(S_j)c(S_k) \\ &= c(S_i \sqcup (S_j \sqcup S_k)) \end{aligned}$$

$$(A.18): \quad \begin{aligned} c(S_i \sqcap (S_j \sqcup S_k)) &= c(S_i) \cdot (c(S_j) + c(S_k) - c(S_j)c(S_k)) \\ &= c(S_i)c(S_j) + c(S_i)c(S_k) - c(S_i)c(S_j)c(S_k) \\ &= c(S_i)c(S_j) + c(S_i)c(S_k) - c(S_i \sqcap S_i)c(S_j)c(S_k) \\ &= c(S_i)c(S_j) + c(S_i)c(S_k) - c((S_i \sqcap S_j) \sqcap (S_i \sqcap S_k)) \\ &= c((S_i \sqcap S_j) \sqcup (S_i \sqcap S_k)) \end{aligned}$$

$$(A.19): \quad \begin{aligned} c(S_i \sqcup (S_j \sqcap S_k)) &= c(S_i) + c(S_j \sqcap S_k) - c(S_i \sqcap (S_j \sqcap S_k)) \\ &= c(S_i \sqcap (S_i \sqcup S_j \sqcup S_k)) + c(S_j)c(S_k) - c(S_i)c(S_j)c(S_k) \\ &= c(S_i)\Big(c(S_i) + c(S_k) - c(S_i)c(S_k) + c(S_j) - c(S_i)c(S_j) \\ &\quad - c(S_j)c(S_k) + c(S_j)c(S_j)c(S_k)\Big) \\ &\quad + c(S_j)c(S_k) - c(S_i)c(S_j)c(S_k) \\ &= c(S_i)c(S_i) + c(S_i)c(S_k) - c(S_i)c(S_i)c(S_k) \\ &\quad + c(S_j)c(S_i) + c(S_j)c(S_k) - c(S_j)c(S_i)c(S_k) \\ &\quad - c(S_i)c(S_j)c(S_i) - c(S_i)c(S_j)c(S_k) + c(S_i)c(S_j)c(S_i)c(S_k) \\ &= (c(S_i) + c(S_j) - c(S_i)c(S_j)) \cdot (c(S_i) + c(S_k) - c(S_i)c(S_k)) \\ &= c(S_i \sqcup S_j) \cdot c(S_i \sqcup S_k) \\ &= c((S_i \sqcup S_j) \sqcap (S_i \sqcup S_k)) \end{aligned}$$

Lemma 20. Let S_i and S be two independent sources. We abbreviate $|\{t \in S_i | t[a] \neq \bot\}|$ with $|t_{S_i}[a] \neq \bot|$. Then, for any attribute a

$$|t_{S_i \sqcap S_j}[a] \neq \bot| = \frac{|t_{S_i}[a] \neq \bot||S_j|}{|UR|} + \frac{|t_{S_j}[a] \neq \bot||S_i|}{|UR|} - \frac{|t_{S_i}[a] \neq \bot||t_{S_j}[a] \neq \bot|}{|UR|} \tag{A.20}$$

$$|t_{S_i \sqsupset S_j}[a] \neq \bot| = |t_{S_i}[a] \neq \bot| + \frac{|t_{S_j}[a] \neq \bot||S_i|}{|UR|} - \frac{|t_{S_i}[a] \neq \bot||t_{S_j}[a] \neq \bot|}{|UR|} \tag{A.21}$$

$$|t_{S_i \sqcup S_j}[a] \neq \bot| = |t_{S_i}[a] \neq \bot| + |t_{S_j}[a] \neq \bot| - \frac{|t_{S_i}[a] \neq \bot||t_{S_j}[a] \neq \bot|}{|UR|} \tag{A.22}$$

PROOF.

(A.20) According to Lemma 17, $\frac{|S_i| \cdot |S_j|}{|UR|}$ is the number of tuples in S_j having a matching tuple in S_i. Thus, $\frac{|t_{S_i}[a] \neq \bot| \cdot |S_j|}{|UR|}$ is the number of tuples in S_j having a matching tuple in S_i with a non-`null` value in attribute a. Analogously, $\frac{|t_{S_j}[a] \neq \bot| \cdot |S_i|}{|UR|}$ is the number of tuples in S_i having a matching tuple in S_j with a non-`null` value in attribute a. Finally we subtract the number of those tuples counted twice, i.e., those that match and have a non-`null` value both in S_i and S_j.

(A.21) The proof is similar to that of (A.20), only the first term is different. In (A.20) only those non-`null` values of S_i entered the sum whose tuple had a matching tuple in S_j. Here, all non-`null` values of S_i enter the sum, because with the left outerjoin-merge all tuples of S_i enter the result.

(A.22) follows from (A.20) and Definition 7.

Theorem 21. Let S_i and S_j be two independent sources. Then the density of the merged sources is

$$d_{S_i \sqcap S_j}(a) = d_{S_i}(a) + d_{S_j}(a) - d_{S_i}(a) \cdot d_{S_j}(a) \tag{A.23}$$

$$d_{S_i \sqsupset S_j}(a) = d_{S_i}(a) + d_{S_j}(a) c(S_j) - d_{S_i}(a) d_{S_j}(a) c(S_j) \tag{A.24}$$

$$d_{S_i \sqcup S_j}(a) = \frac{d_{S_i}(a) \cdot c(S_i)}{c(S_i \sqcup S_j)} + \frac{d_{S_j}(a) \cdot c(S_j)}{c(S_i \sqcup S_j)} - \frac{d_{S_i \sqcap S_j}(a) \cdot c(S_i \sqcap S_j)}{c(S_i \sqcup S_j)} \tag{A.25}$$

PROOF.

(A.23) We apply Definition 12 and transform

$$\begin{aligned}
d_{S_i \sqcap S_j}(a) &= \frac{1}{|S_i \sqcap S_j|} \cdot |t_{S_i \sqcap S_j}[a] \neq \bot| \\
\text{(Lemma 17)} &= \frac{|UR|}{|S_i| \cdot |S_j|} \cdot |t_{S_i \sqcap S_j}[a] \neq \bot| \\
&= \frac{|UR|}{|UR|^2 \cdot c(S_i) \cdot c(S_j)} \cdot |t_{S_i \sqcap S_j}[a] \neq \bot| \\
\text{(Lemma 20)} &= \frac{|t_{S_i}[a] \neq \bot| \cdot |S_j|}{|UR|^2 \cdot c(S_i) \cdot c(S_j)} + \frac{|t_{S_j}[a] \neq \bot| \cdot |S_i|}{|UR|^2 \cdot c(S_i) \cdot c(S_j)} - \frac{|t_{S_i}[a] \neq \bot| \cdot |t_{S_j}[a] \neq \bot|}{|UR|^2 \cdot c(S_i) \cdot c(S_j)} \\
&= \frac{|t_{S_i}[a] \neq \bot| \cdot |S_j|}{|S_i| \cdot |S_j|} + \frac{|t_{S_j}[a] \neq \bot| \cdot |S_i|}{|S_i| \cdot |S_j|} - \frac{|t_{S_i}[a] \neq \bot| \cdot |t_{S_j}[a] \neq \bot|}{|S_i| \cdot |S_j|} \\
&= d_{S_i}(a) + d_{S_j}(a) - d_{S_i}(a) \cdot d_{S_j}(a)
\end{aligned}$$

(A.24) We apply Definition 12 and transform

$$\begin{aligned}
d_{S_i \sqsupset S_j}(a) &= \frac{|t_{S_i \sqsupset S_j}[a] \neq \bot|}{|S_i \sqsupset S_j|} \\
\text{(Lemma 17)} &= \frac{|t_{S_i \sqsupset S_j}[a] \neq \bot|}{|S_i|} \\
\text{(Lemma 20)} &= \frac{|t_{S_i}[a] \neq \bot|}{|S_i|} + \frac{|t_{S_j}[a] \neq \bot||S_i|}{|UR||S_i|} - \frac{|t_{S_i}[a] \neq \bot||t_{S_j}[a] \neq \bot|}{|UR||S_i|} \\
&= d_{S_i}(a) + \frac{|t_{S_j}[a] \neq \bot|}{|UR|} - d_{S_i}(a) \cdot \frac{|t_{S_j}[a] \neq \bot|}{|UR|} \\
&= d_{S_i}(a) + \frac{|t_{S_j}[a] \neq \bot| c(S_j)}{|S_j|} - d_{S_i}(a) \cdot \frac{|t_{S_j}[a] \neq \bot| c(S_j)}{|S_j|} \\
&= d_{S_i}(a) + d_{S_j}(a) c(S_j) - d_{S_i}(a) d_{S_j}(a) c(S_j)
\end{aligned}$$

(A.25) We apply Definition 12 and transform

$$\begin{aligned}
d_{S_i \sqcup S_j}(a) &= \frac{|t_{S_i \sqcup S_j}[a] \neq \bot|}{|S_i \sqcup S_j|} \\
\text{(Lemma 17)} &= \frac{|t_{S_i \sqcup S_j}[a] \neq \bot|}{|S_i| + |S_j| - |S_i \sqcap S_j|} \\
\text{(Lemma 20)} &= \frac{|t_{S_i}[a] \neq \bot| + |t_{S_j}[a] \neq \bot| - \frac{1}{|UR|}|t_{S_i}[a] \neq \bot||t_{S_j}[a] \neq \bot|}{|UR| \cdot c(S_i \sqcup S_j)} \\
&= \frac{|S_i| d_{S_i}(a) + |S_j| d_{S_j}(a) - \frac{1}{|UR|}|S_i||S_j| d_{S_i}(a) d_{S_j}(a)}{|UR| \cdot c(S_i \sqcup S_j)} \\
&= \frac{|UR|[c(S_i) d_{S_i}(a) + c(S_j) d_{S_j}(a) - c(S_i) c(S_j) d_{S_i}(a) d_{S_j}(a)]}{|UR| \cdot c(S_i \sqcup S_j)} \\
&= \frac{d_{S_i}(a) \cdot c(S_i)}{c(S_i \sqcup S_j)} + \frac{d_{S_j}(a) \cdot c(S_j)}{c(S_i \sqcup S_j)} - \frac{d_{S_i}(a) \cdot d_{S_j}(a) \cdot c(S_i \sqcap S_j)}{c(S_i \sqcup S_j)}
\end{aligned}$$

Theorem 22. Let S_i and S_j be two independent sources. Given the density scores for all attributes of a merged result, the density of the entire result is

$$d(S_i \sqcap S_j) = \frac{1}{|A|} \sum_{a \in A} d_{S_i \sqcap S_j}(a) \tag{A.26}$$

$$d(S_i \sqsupset S_j) = \frac{1}{|A|} \sum_{a \in A} d_{S_i \sqsupset S_j}(a) \tag{A.27}$$

$$d(S_i \sqcup S_j) = \frac{1}{|A|} \sum_{a \in A} d_{S_i \sqcup S_j}(a) \tag{A.28}$$

PROOF. The proof follows immediately from Definition 13.

Lemma 23. Source density is commutative, associative, and distributive for $\sqcap$ and $\sqcup$ for independent sources:

$$d_{S_i \sqcap S_j}(a) = d_{S_j \sqcap S_i}(a) \tag{A.29}$$

$$d_{S_i \sqcup S_j}(a) = d_{S_j \sqcup S_i}(a) \tag{A.30}$$

$$d_{(S_i \sqcap S_j) \sqcap S_k}(a) = d_{S_i \sqcap (S_j \sqcap S_k)}(a) \tag{A.31}$$

$$d_{(S_i \sqcup S_j) \sqcup S_k}(a) = d_{S_i \sqcup (S_j \sqcup S_k)}(a) \tag{A.32}$$

$$d_{S_i \sqcap (S_j \sqcup S_k)}(a) = d_{(S_i \sqcap S_j) \sqcup (S_i \sqcap S_k)}(a) \tag{A.33}$$

$$d_{S_i \sqcup (S_j \sqcap S_k)}(a) = d_{(S_i \sqcup S_j) \sqcap (S_i \sqcup S_k)}(a) \tag{A.34}$$

PROOF. The proofs of (A.29) and (A.30) are trivial. The proof of (A.31) is analog to that of (A.17) in Theorem 19. Due to the length of the following expressions, we use a shorthand notation to prove (A.32) and abbreviate $c(S_i)$ with c_i and $d_{S_i}(a)$ with d_i. Also, we treat numerator and denominator separately. The denominator is transformed with Theorem 19. We transform the numerator:

$$\begin{aligned}
d_{(S_i \sqcup S_j) \sqcup S_k}(a) &= \frac{d_{S_i \sqcup S_j}(a) \cdot c(S_i \sqcup S_j) + d_{S_k}(a) \cdot c(S_k) - d_{(S_i \sqcup S_j) \sqcap S_k}(a) \cdot c((S_i \sqcup S_j) \sqcap S_k)}{c((S_i \sqcup S_j) \sqcup S_k)} \\
\text{(only numerator)} &= d_i c_i + d_j c_j - d_{i \sqcap j} c_{i \sqcap j} + d_k c_k - [(d_{i \sqcup j} + d_k - d_{i \sqcap j} d_k) c_{i \sqcup j} c_k] \\
&= d_i c_i + d_j c_j + d_k c_k - [d_i + d_j - d_i d_j] c_i c_j - d_{i \sqcup j} c_{i \sqcup j} c_k - d_k c_{i \sqcup j} c_k + d_{i \sqcap j} d_k c_{i \sqcup j} c_k \\
&= d_i c_i + d_j c_j + d_k c_k - d_i c_i c_j - d_j c_i c_j + d_i d_j c_i c_j - [d_i c_i + d_j c_j - d_{i \sqcap j} c_{i \sqcap j}] c_k \\
&\quad - d_k [c_i + c_j - c_i c_j] c_k + [d_i + d_j - d_i d_j] d_k [c_i + c_j - c_i c_j] c_k \\
&= d_i c_i + d_j c_j + d_k c_k - d_i c_i c_j - d_j c_i c_j + d_i d_j c_i c_j - d_i c_i c_k - d_j c_j c_k \\
&\quad + [d_i + d_j - d_i d_j] c_i c_j c_k - d_k c_i c_k - d_k c_j c_k + d_k c_i c_j c_k \\
&\quad + [d_i d_k + d_j d_k - d_i d_j d_k][c_i c_k + c_j c_k - c_i c_j c_k]
\end{aligned}$$

$$\begin{aligned}
&= d_ic_i + d_jc_j + d_kc_k - d_ic_ic_j - d_jc_ic_j + d_id_jc_ic_j - d_ic_ic_k - d_jc_jc_k \\
&\quad + d_ic_ic_jc_k + d_jc_ic_jc_k - d_id_jc_ic_jc_k - d_kc_ic_k - d_kc_jc_k + d_kc_ic_jc_k \\
&\quad + d_id_kc_ic_k + d_jd_kc_ic_k - d_id_jd_kc_ic_k + d_id_kc_jc_k + d_jd_kc_jc_k \\
&\quad - d_id_jd_kc_jc_k - d_id_kc_ic_jc_k - d_jd_kc_ic_jc_k + d_id_jd_kc_ic_jc_k \\
&= d_ic_i + d_jc_j + d_kc_k - d_ic_ic_j - d_ic_ic_k - d_jc_ic_j - d_jc_jc_k - d_kc_ic_k - d_kc_jc_k \\
&\quad + d_ic_ic_jc_k + d_jc_ic_jc_k + d_kc_ic_jc_k + d_id_jc_ic_j + d_id_kc_jc_k + d_id_kc_ic_k + d_jd_kc_jc_k \\
&\quad + d_jd_kc_ic_k - d_id_jc_ic_jc_k - d_id_kc_ic_jc_k - d_jd_kc_ic_jc_k \\
&\quad - d_id_jd_kc_ic_k - d_id_jd_kc_jc_k + d_id_jd_kc_ic_jc_k
\end{aligned}$$

The final expression is symmetric with respect to j, i.e., wherever j appears in combination with i, it also appears in combination with k in the same manner. Hence, we can deduce associativity.

For brevity we omit the tedious proofs of (A.33) and (A.34). Their structure is similar to the proofs above.

Theorem 24. Source density is commutative, associative, and distributive for $\sqcap$ and $\sqcup$ for independent sources:

$$d(S_i \sqcap S_j) = d(S_j \sqcap S_i) \tag{A.35}$$
$$d(S_i \sqcup S_j) = d(S_j \sqcup S_i) \tag{A.36}$$
$$d((S_i \sqcap S_j) \sqcap S_k) = d(S_i \sqcap (S_j \sqcap S_k)) \tag{A.37}$$
$$d((S_i \sqcup S_j) \sqcup S_k) = d(S_i \sqcup (S_j \sqcup S_k)) \tag{A.38}$$
$$d(S_i \sqcap (S_j \sqcup S_k)) = d((S_i \sqcap S_j) \sqcup (S_i \sqcap S_k)) \tag{A.39}$$
$$d(S_i \sqcup (S_j \sqcap S_k)) = d((S_i \sqcup S_j) \sqcap (S_i \sqcup S_k)) \tag{A.40}$$

PROOF. All properties follow trivially from Definition 13 with Lemma 23.

Theorem 25. Let $P = S_1, \ldots, S_n$ be a set of sources. Then completeness of P is $C(P) = c(P) \cdot d(P)$.

PROOF. The proof follows from Theorem 16, because a plan P represents a set of tuples, just as a source S does.