Querying Complex Structured Databases *

Cong Yu H. V. Jagadish
Department of EECS, University of Michigan
{congy, jag}@eecs.umich.edu

ABSTRACT

Correctly generating a structured query (e.g., an XQuery or a SQL query) requires the user to have a full understanding of the database schema, which can be a daunting task. Alternative query models have been proposed to give users the ability to query the database without schema knowledge. Those models, including *simple keyword search* and *labeled keyword search*, aim to extract meaningful data fragments that match the structure-free query conditions (e.g., keywords) based on various *matching semantics*. Typically, the matching semantics are content-based: they are defined on data node inter-relationships and incur significant query evaluation cost. Our first contribution is a novel matching semantics based on analyzing the database schema. We show that query models employing a schema-based matching semantics can reduce query evaluation cost significantly while maintaining or even improving result quality.

The adoption of schema-based matching semantics does not change the nature of those query models: they are still *schema-ignorant*, i.e., users express no schema knowledge (except the labels in labeled keyword search) in the query. While those models work well for some queries on some databases, they often encounter problems when applied to complex queries on databases with complex schemas. Our second contribution is a novel query model that incorporates partial schema knowledge through the use of *schema summary*. This new *summary-aware* query model, called *Meaningful Summary Query* (MSQ), seamlessly integrates summary-based structural conditions and structure-free conditions, and enables ordinary users to query complex databases. We design algorithms for evaluating MSQ queries, and demonstrate that MSQ queries can produce better results against complex databases when compared with previous approaches, and that they can be efficiently evaluated.

1. INTRODUCTION

Establishing effective query mechanisms that are easily accessible to ordinary users is one of the most elusive goals of database research. Structured query models (e.g., SQL for relational databases and XQuery for XML databases) require the users to be knowledgable about the schema so that they can precisely specify both the locations of the entities and attributes they are searching for, and the relationships among those entities and attributes. However, understanding schema is a non-trivial task. Consider the example schema derived from the XMark Benchmark [18] as shown in Figure 1 (A). To answer a relatively simple query like the example Q1 in Figure 1, the user will have to examine the whole schema and (1) identify the location of `item`, (2) identify which element (`region/name`) represent the region of the item, and (3) identify which element (`description`) associated with the item should contain the keyword phrase "antiques," before correctly generating the following XQuery:

for $r in doc()/regions/region, $i in $r/item
where $r/name = "asia" and $i/description ∼ "antiques"
return $i

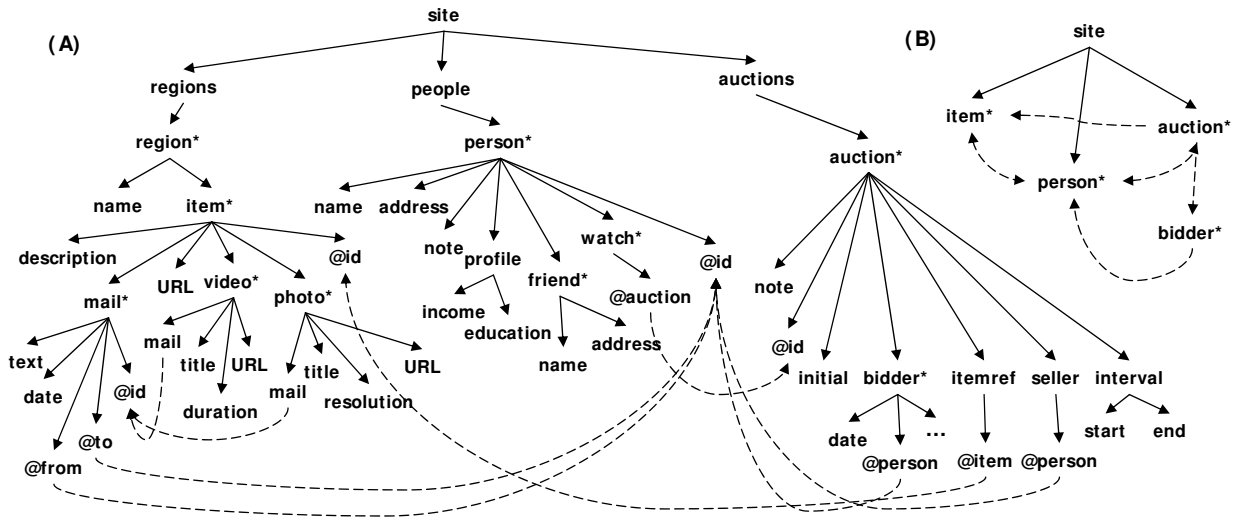
1.1 Structure-Free Query Models

Several alternative query models have been proposed to eliminate structures in the query, and they are often considered *structure-free* query models. One such model is *simple keyword search* [1, 2, 11, 10, 7, 21]. A simple keyword search query consists of a set of terms, $[t_1, t_2, ...]$. For example, Q1 can be expressed as `[item; region; asia; antiques]`. Data fragments (in the form of either a subtree of nodes or a group of tuples) matching those terms are returned as relevant results, often in some ranked order. *A fundamental concept of structure-free query models is the* matching semantics*, which determines which data fragments are* meaningful *and* relevant *to the structure-free query condition.* Several matching semantics for simple keyword search have been proposed: 1) Lowest Common Ancestor Subtree (LCAS) for XML data model—a subtree of nodes is considered a match if the nodes in the subtree collectively contain all the query terms, and the subtree root node is lowest in the document hierarchy; 2) Smallest Tuple Group (STG) for relational data model—a group of connected tuples is considered a match if tuples in the group collectively contain all the query terms, and no tuple can be removed without losing at least one of the query terms.

Another commonly adopted structure-free query model is *labeled keyword search* [9, 15]. This model trades some query simplicity with better query accuracy by allowing users to specify label terms and value terms separately. For example, in this model, Q1 can be expressed as [`region:`"asia"; `item:`"antiques"], where terms `region` and `item` are now identified as labels of schema elements. Because of this distinction, labeled keyword search can adopt stricter matching semantics to fetch "more meaningful" data fragments. For example, the *interconnection relationship semantics* in [9] excludes data fragments that contain multiple nodes of the same label; and the *meaningful LCAS (MLCAS) semantics* in [15] further excludes those data fragments whose roots are not the lowest ancestors of all the element labels, regardless of the keywords.

*Supported in part by NSF under grant IIS-0438909, and NIH under grant 1-U54-DA021519.

VLDB '07, September 23-28, 2007, Vienna, Austria.



Example Queries:

Q1: Find items that are "antiques" and registered in region "asia";

Q2: Retrieve auctions that are *sold* by the person named "peter" in "chicago" and *contain* items that are "antiques" in region "asia";

Q3: Identify all auctions that are *sold* by a person in "chicago", and *bid* upon by *the same* person more than 3 times;

Figure 1: An example database schema $S_{\texttt{site}}$ (A) that is derived from XMark, its schema summary (B), and example queries. Nodes, solid arrows, and dashed arrows in (A) represent schema elements, structural links, and value links, respectively. Elements with suffix '*' are repeatable, and those with prefix '@' are attributes.

In almost all previous proposals, matching semantics is defined solely based on the data fragments themselves. There are two drawbacks for this content-based semantics. First, the "meaningful-ness" of a data fragment can be affected by the presence of specific nodes in the data fragment or the choice of specific keywords in the user query, regardless of whether the data fragment is truly meaningful. Second, query evaluation using content-based semantics requires the system to examine all the data fragments in the database, which can have high query evaluation cost even with index support. In this paper, we advocate that the schema of a database provides important clues to the semantic meaning of the data fragments, and propose a novel schema-based matching semantics for determining meaningful data fragments. We show that this new matching semantics avoids certain pitfalls encountered by query mechanisms with content-based matching semantics, and at the same time significantly improves the query performance.

1.2 Relaxed-Structure Query Model

Structure-free query models (regardless of the matching semantics they adopt) are also inherently limited in the query semantics that they can express. For example, queries requiring non-trivial value joins or aggregations like Q2 and Q3 in Figure 1 are almost impossible to express using the aforementioned structured-free query models. To address the issue, several relaxed structure query mechanisms for XML have been proposed [3, 15, 6, 4, 5] that implicitly or explicitly integrate structure-free components into structured queries. In particular, [6] proposed *Query Relaxation*, which converts the user provided query tree pattern into less restrictive patterns based on a set of relaxation rules. Those relaxed patterns are then used to match data fragments with a normal exact matching semantics. In [15], the authors proposed *Schema-Free XQuery*, which enables the users to embed structure-free conditions into a structured query, therefore allowing the users to retrieve data fragments that matches the structure-free conditions, as well as the structural condition in the rest of the query.

However, these relaxed-structure query mechanisms still have their limitations. In [6, 5], users are expected to generate an approximately correct query tree pattern in order for the system to retrieve reasonably accurate results. If the query tree pattern is wildly inaccurate, many retrieved results will likely to be inaccurate. On the other hand, in [15], users are required to manually identify and specify relationships between entities that are non-hierarchically and/or remotely related, because the matching semantics proposed is applicable only to hierarchically linked data nodes within close proximity. As an example, to express a schema-free query representing Q2 in Figure 1, user will need to explicitly specify how `auction` is associated with `person` and `item`. Simply listing all the keywords will retrieve the root of the entire database as the matching result instead.

We observe that the limitations of the aforementioned mechanisms become painfully obvious *when the schema is* extremely complex *and the query needs to retrieve data nodes that are remotely related.* Under those circumstances, an approximate query is difficult to express "correctly," and the results obtained by evaluating such a query using the above two mechanisms are often no better than the results obtained using simple keyword queries.

Understanding real databases with complex schemas is the subject of our recent work [22], where we proposed the novel concept of *Schema Summary.* A schema summary is, intuitively, a concise description of the underlying database that is easier for the user to understand than the original complex schema. It consists of representative elements that are the

```
Q1: set   @c1 = item[region:"asia", item:"antiques"]
    for   $i  in  doc()/site.item.MF(@c1)
    return $i

Q2: set   @c2 = item[region:"asia", item:"antiques"]
          @c3 = person[name:"peter", address:"chicago"]
    for   $a  in  doc()/site.auction.MF(),
          $i  in  doc()/site.item.MF(@c2),
          $p  in  doc()/site.person.MF(@c3)
    where MR($a, $i, -) and MR($a, $p, "sell")
    return $a

Q3: set   @c4 = person[address:"chicago"]
    for   $a  in  doc()/site.auction.MF(),
          $p  in  doc()/site.person.MF($c4)
    let   $b  in  $a.bidder.MF()
    where MR($a, $p, "sell") and MR($b, $p, "bid") and
          count($b) > 3
    return $a
```

Figure 2: Example Meaningful Summary queries.

most important in the schema and collectively have the most coverage of the underlying database content. Figure 1 (B) shows an example summary of the XMark schema. In [22], we demonstrated that a well-generated schema summary can help users more effectively understand the schema—as measured by the reduced schema exploration needed to locate desired elements by the user. For example, to locate elements (`region, name, item, description`) needed to construct Q1, a user only needs to explore the abstract element `item`, without being distracted by the irrelevant information presented in other parts of the schema.

The benefit of generating a fully structured query through schema summary exploration, on the other hand, is often limited. For example, Q2 requires the user to locate schema elements under three abstract elements (`auction, item, person`) in the summary, which means the user will have to explore almost the entire schema. However, we note that *the simplicity of a schema summary makes it an ideal starting point for users to formulate their queries.* The challenge is how to deal with the schema structures that are hidden inside the abstract elements. Fortunately, we have structure-free query models. Specifically, combining the strengths of schema summary and structure-free query models (labeled keyword search in particular), we propose a novel query paradigm called **Meaningful Summary Query (MSQ)**. Intuitively, an MSQ query is a structured query constructed using the schema summary and with structure-free conditions embedded. Figure 2 illustrates several MSQ queries that correspond to the example queries in Figure 1. Their detailed semantics and usage will be explained later. As seen from those examples, with the schema summary as the guide, users can construct a query that does not deviate too much from the desired structure, and therefore ensure the relative accuracy of the results. On the other hand, correct specification of an MSQ query only requires the user to examine the simple schema summary, instead of learning the whole complex schema. With the adoption of schema-based matching semantics, an MSQ query can be evaluated efficiently by rewriting it into a normal structured query, thereby leveraging the query optimization facilities that already exist in current DBMSs.

1.3 Main Contributions and Outline

We make two major contributions. First, we propose a novel schema-based matching semantics for the structure-free query model using labeled keyword conditions (Section 3). Second, we propose a new query paradigm called Meaningful Summary Query (Section 4), which enables a user to pose complex queries against complex databases with the knowledge of only schema summaries. We describe the MSQ query evaluation system in Section 5 and its evaluation in Section 6. We discuss related work in Section 7 and conclude in Section 8. Necessary background on *Schema Summary* [22] is presented in Section 2, along with definitions that we will use throughout the rest of the paper.

2. BACKGROUND

We consider both relational and XML data models.

Schema S : We consider a schema as a labeled directed graph, where the nodes represent *schema elements*, and the links represent *relationships* between schema elements, as shown is shown in Figure 1. There are two kinds of relationship links. The first is *structural links*, which are drawn as solid arrows with parent element (e.g., `region`) pointing to child element (e.g., `item`). Structural links represent the parent/child relationships in XML, or table/column relationships in the relational model. The second is *value links*, which are drawn as dashed arrows in Figure 1 with an attribute of the referrer element (e.g., `video`) pointing to an attribute of the referee element (e.g., `mail`). Value links represent the key/foreign-key (referential) constraints in both XML and relational models. We further enforce that every element in S is a child of another element, except for the *root* (e.g., `site`). For relational schemas, we introduce our own *root* and establish structural links between the *root* and every element in S that is not a child of another element.

While structural and value links are syntactically different, they in fact share the same fundamental semantics. For example, the value link between `video` and `mail` means potentially multiple `video`s are included in a single `mail`: the same semantics as if `video` were a child element of `mail`. Based on this observation, we define the concepts of *General Parent and Ancestor*, which become useful in Section 3.1.

Definition 1 (General Parent and Ancestor). *Given a schema S, an element e_A in S is a* general parent *of another element e_B in S if:*

- *e_A is the parent element of e_B, or,*
- *e_A is the referee element of e_B.*

Furthermore, e_A is a general ancestor *of element e'_B if:*

- *e_A is a general parent of e'_B, or,*
- *There is an element e_C in S, such that e_A is a general parent of e_C and e_C is a general ancestor of e'_B.*

We call e_B (e'_B) a general child (descendant) *of e_A.*

We ignore mixed-content elements in the XML data model: such an element can potentially be modeled using a special simple element to store all its atomic content. More importantly, a schema element is considered *repeatable* when it can have, in the database tree, multiple corresponding data nodes, all of which are under the same parent data node (e.g., `person`). Otherwise, it is considered *non-repeatable* (e.g., `profile`). In XML, repeatable elements can be identified as having `maxOccurs` greater than 1, while in the relational model, elements representing whole tuples are considered repeatable.

Database D : We consider the database to be a rooted labeled tree. Figure 3 illustrates an example (partial) database

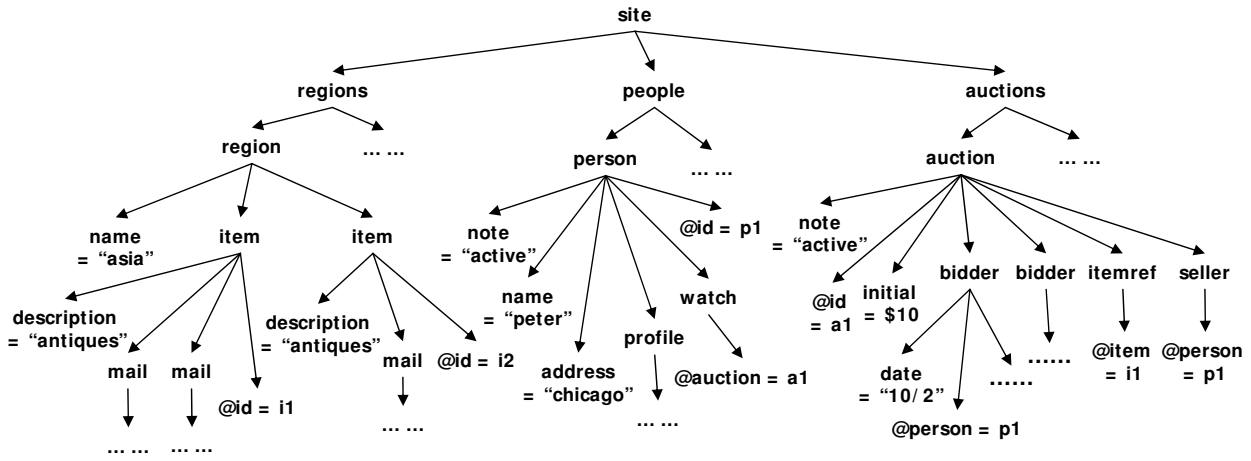


Figure 3: Example (partial) database conforming to the schema in Figure 1.

tree. The structural links are represented as solid links between data nodes in the database tree, with parent node pointing to child node. The value links in the schema are implicitly represented when two data nodes have the same atomic value (e.g., `@id=i1` under `/site/regions/region/item` and `@item=i1` under `/site/auctions/auction/itemref`). We assume all the database trees we consider conform to their associated schemas according to [19].

Schema Summary S_{sum} : A schema summary (e.g., Figure 1 (B)) is modeled as a labeled directed graph, where the nodes represent abstract elements and the links represent abstract links between those elements. A schema summary can not exist alone and is always associated with an underlying schema, upon which the summary is generated.

Definition 2 (Abstract Element). *An abstract element E^a, in a schema summary S_{sum} of the schema S, is a pair $\langle e_r, S_{sub}\rangle$, where S_{sub} is called the sub-schema associated with E^a, and contains a subset of schema elements and relationship links in S; and e_r, one of the elements in S_{sub} (not necessarily the root element), is the* representative element *of S_{sub}.*

No two abstract elements, E^a_1 and E^a_2, of S_{sum} have overlapping sub-schemas (i.e., $E^a_1.S_{sub}$ and $E^a_2.S_{sub}$ do not share any common schema element).

Intuitively, a schema summary decomposes the original schema into several non-overlapping sub-schemas, with each sub-schema represented by one of the abstract elements. For example, abstract element `item` represents the sub-schema $S_{\texttt{item}}$ as shown in Figure 4.

There are several things worth mentioning here. First, in the graphical representation of the schema summary, each abstract element is shown with only its representative element, and the sub-schema is "abstracted" away from the user. Second, we assume a *good* schema summary has been generated for the schema in this study. Several heuristic algorithms for automatic summary generation have been proposed in [22]. However, in this study, our MSQ query model does not distinguish between automatic summaries and summaries that are generated by users based on their knowledge of the schemas—it utilizes both in the same way. Third, a schema summary is called a *full summary* if all of its elements except the root element are abstract elements. In this study, we focus on full summaries for the purpose of query formulation. Abstract elements in the summary are connected through abstract links, which consolidates all the relationship links across two abstract elements.

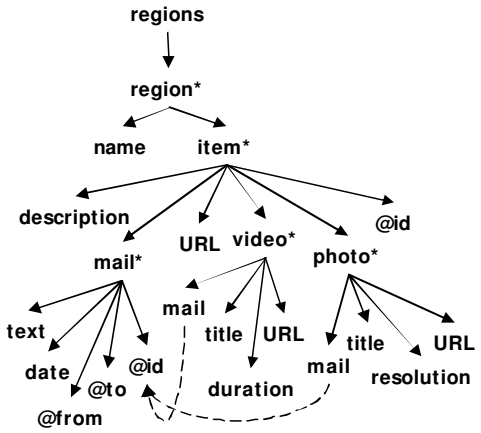


Figure 4: A sub-schema ($S_{\texttt{item}}$) of the XMark schema in Figure 1.

3. SCHEMA-BASED MATCHING SEMANTICS

In this section, we describe the schema-based matching semantics for identifying *meaningful* and *relevant* data fragments given the labeled keyword search condition. We assume that the user-provided labels are correct, or can be converted to the correct one through ontology-based normalization [14]. For the rest of this section, we focus on a sub-schema of the example XMark schema, $S_{\texttt{item}}$ (shown in Figure 4), for simplicity of discussion. We first formally define *Labeled Keyword Condition* in Definition 3.

Definition 3 (Labeled Keyword Condition). *A Labeled Keyword Condition $C = \{(l_i{:}v_i) \mid i \in [1,k]\}$, for a database D with schema S, is a set of pairs, where each pair contains a label (l_i), which corresponds to a schema element label in S, and a value (v_i), which corresponds to content of the schema element in D. We denote the set of labels $\{l_i \mid i \in [1,k]\}$ in C as L_C.*

Given C, we adopt a schema-based matching semantics to

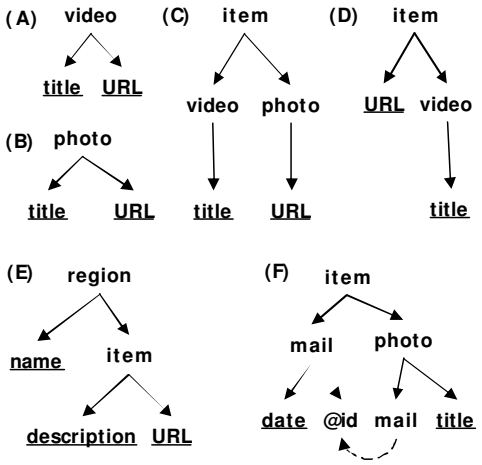


Figure 5: Potentially meaningful schema patterns: labels in the labeled keyword condition are underlined.

determine which data fragments in the database are matched to C. This matching semantics is defined in two phases. The first phase defines the semantics of *Meaningful Schema Pattern*s (MSPs) given the set of labels L_C. The second phase then defines the relevant data fragments given C and the MSPs obtained in the first phase.

3.1 Meaningful Schema Pattern

We define a *schema pattern* P as *a subgraph of the original schema* S: i.e., each schema element in P is a schema element in S and each relationship link in P is a relationship link in S. Definition 4 introduces *Meaningful Schema Pattern* (MSP) with the *Basic Semantics*.

DEFINITION 4 (MSP WITH BASIC SEMANTICS). *Given the schema S and the set of labels L_C in C, a schema pattern P is a Meaningful Schema Pattern (MSP) according to the* Basic Semantics *if the following conditions are satisfied.*

`i`. *For each $l \in L_C$, there exists a schema element e in P, such that $e.label = l$;*

`ii`. *For each schema element e or relationship link r in P, if e or r is removed from P, then either condition* (`i`) *is no longer satisfied, or P is no longer a connected graph.*

The Basic Semantics is, in many aspects, very similar to those adopted in previous content-based semantics, with the fundamental difference being that it is defined on schema graphs instead of data graphs. Condition (`i`) ensures that all the labels in C are covered by the MSP, while condition (`ii`) ensures that all schema elements in MSP are necessary. As an example, given $L_C = [\texttt{title}, \texttt{URL}]$, we can identify multiple MSPs, several of which are shown in Figure 5 (A-D), from $S_{\texttt{item}}$ (Figure 4) according to the Basic semantics. Among the MSPs being shown, the Basic semantics successfully identifies patterns (A) and (B) as being meaningful—they correspond to the `video` entities and the `photo` entities, respectively. However, it also identifies patterns (C) and (D), which are intuitively not as meaningful as (A) and (B).

Before addressing this problem, we first take a look at an important feature of schema elements. As mentioned in Section 2, a schema element can be considered as *repeatable* or *non-repeatable*, depending on how many data nodes that correspond to it can share a single parent data node. *Schema element repeatability is in fact a strong indicator of whether the schema element represent an entity or an attribute.* For example, repeatable elements in $S_{\texttt{item}}$, like `item` and `mail`, correspond to real world item and mail entities, while non-repeatable elements, like `description` and `date`, correspond to attributes of those entities[1]. Based on this observation, we refer to repeatable elements as *entity elements*, and non-repeatable elements as *attribute elements*. More importantly, we consider any attribute element (a) as an attribute of its closest ancestor entity element (e), and we state that *a belongs to e*. For convenience, we also state that e belongs to e itself.

Given a schema pattern with a set of elements, there are two scenarios. First, all the elements belong to the same entity element (e.g., Figure 5 (A) and (B)), in which case, the schema pattern is clearly an MSP. Second, the elements belong to different entity elements, in which case, we consider the schema pattern to be meaningful *if and only if every pair of entity elements in the pattern are meaningfully related*. The question is how do we determine whether two entity elements are meaningfully related.

Before answering that question, we first examine how two entity elements can be connected in the schema graph, and see if the schema structure can again provide us with indications. There are three basic relationships between two entity elements. First, the *ancestor-descendant (AD)* relationship, where one entity element is a *general ancestor* (defined in Definition 1 based on both structural and value links) of the other. Element pairs (`item`, `mail`) and (`mail`, `video`) are such examples. Second, *common ancestor sibling (SIB-A)* relationship, where two entity elements share at least one common general ancestor. Element pair (`video`, `photo`) is one such example. Third, *common descendant sibling (SIB-D)* relationship, where two entity elements share at least one common general descendant. Element pair (`mail`, `video`) has a SIB-D relationship by sharing `video/mail` as a common descendant. (Element `video/mail` is a general descendant of `mail` according to Definition 1.) We note here that SIB-D relationships occur only if there are value links in the schema graph.

We advocate that the AD and SIB-D relationships connect two entity elements in a meaningful way, while the SIB-A relationship does not. Intuitively, we observe that information regarding an entity always describes (i.e., is relevant to) its general descendant entities. For example, in Figure 5 (E), the `name` of a `region` essentially describes where all the descendant `item`s of that `region` are located. In fact, maintaining attributes at the ancestor can be viewed as an alternative way of maintaining the same attributes, redundantly, at each descendant. As a result, two entities with an AD relationship can be considered as meaningfully related because, essentially, the parent is also describing the descendant entity. Similarly, two entities with a SIB-D relationship can be considered as meaningfully related because both are describing their common descendant entity. The same argument, however, can not be made for two entity elements with a SIB-A relationship—no single entity is being described by both elements. Figure 5 (E) and (F) illustrate examples of schema patterns consisting of entities with AD and SIB-D relationships, respectively. Particularly in (F),

[1]This may not always be true, and we do occasionally find repeatable elements that are more like attributes—they tend to be at the leaf of the schema hierarchy and repeat very few times in the database, making them easy to identify.

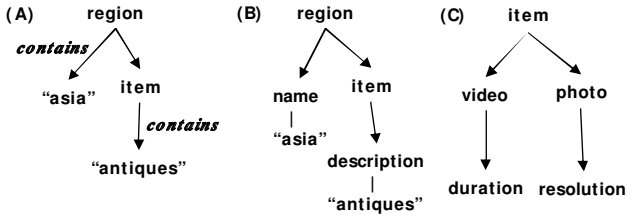


Figure 6: The Meaningful Query Pattern (A) for Q1, one of its Meaningful Data Fragments (B), and a data fragment (C) mistakenly identified as meaningful given keywords [duration:, resolution:] by content-based semantics.

where L_C = [date, title], mail and photo are meaningfully related because they share photo/mail as the common descendant. We now formally introduce the *Related-Entity (RE) Semantics* for MSP.

Definition 5 (MSP with Related-Entity Semantics). *Given the schema S and the set of labels L_C in C, a schema pattern P is a Meaningful Schema Pattern (MSP) according to the* Related-Entity (RE) Semantics *if the following conditions are satisfied.*

i. *P is considered meaningful under the Basic Semantics.*

ii. *Any two entity elements, e_1 and e_2, in P, must be connected in exactly one of the following two ways:*

− e_1 (or e_2) is a general ancestor of e_2 (or e_1) in P: AD relationship;

− both e_1 and e_2 are general ancestors of a third entity element e in P: SIB-D relationship.

iii. *For each schema element e or relationship link r in P, if e or r is removed from P, then condition* (i) *is no longer satisfied.*

Two entity elements may be connected in the original schema through multiple AD or SIB-D relationships, condition (ii) ensures that each MSP utilizes only one of those relationships. Condition (iii) ensures that the MSP keeps track of all the elements and links that make the pattern meaningful. Based on the RE Semantics, schema pattern (C) in Figure 5 can now be filtered out. However, schema pattern (D) still satisfies the RE Semantics. The problem here is not that pattern (D) is not meaningful by itself, but that (D) is not as meaningful as (A) or (B) because (D) properly contains (A) and (B). Formally, we introduce the *Non-Redundant (NR) Semantics* to address this issue.

Definition 6 (MSP with Non-Redundant Semantics). *Given the schema S and the set of labels L_C in C, a schema pattern P is a Meaningful Schema Pattern (MSP) according to the* Non-Redundant (NR) Semantics *if the following conditions are satisfied.*

i. *P is considered meaningful under the RE Semantics.*

ii. *There is no other schema pattern P', s.t. P' satisfies the RE Semantics for L_C, and the set of entity elements in P' is a strict subset of the set of entity elements in P.*

The NR Semantics in Definition 6 presents the most restrictive and accurate semantics for defining an MSP. Identified MSPs based on this semantics can then be used, along with the value terms in C, to identify Meaningful Data Fragments in the second phase, which we will describe next.

3.2 Meaningful Data Fragment

Identifying Meaningful Data Fragments (MDFs) based on extracted MSPs and the labeled keyword condition C involves two steps. The first step is converting each MSP into a *Meaningful Query Pattern* (MQP) by connecting each schema element in MSP with its corresponding value term in C (if there is one), through a "contains" link. The second step is evaluating the resulting MQPs against the database to retrieve the MDFs. Figure 6 illustrates the MQP for example query Q1, and one of the MDFs. The converting step is relatively straight-forward and we only discuss the issues involved in the evaluation step.

The key issue in evaluating a MQP is *determining the scope of "contains"*. Because of the limited knowledge the users have about the schema, they can not be expected to identify the perfect/closest label that should be associated with each value term. For example, in Q1, the user may not know that term "antique" appears directly inside the content of description, and simply associate it with item instead. If we naively evaluate the query pattern by simply looking for "antique" within the content of item nodes, no data fragment will be matched since item nodes have no content at all. As a result, the search scope of the value terms must be expanded beyond just the content of the node with the label. We perform two levels of expansion, *all-attributes* and *all-subtree*. In all-attribute scope expansion, we search for the value term within the content of all the attributes of the labeled node (i.e., the expansion stops when entity boundaries are encountered). For example, given [item:"antiques"], we will search for "antiques" under all attribute elements of item, namely description, URL, and @id. In all-subtree scope expansion, we search for the value term within the entire subtree of the labeled node. For example, given the same condition, any occurrence of "antiques" under an item node will return a positive match. All-attribute expansion is more likely to return only relevant data fragments, but may miss certain relevant ones; while all-subtree expansion will retrieve all relevant data fragments along with many irrelevant ones.

Definition 7 (Meaningful Data Fragment). *Given the database T, the schema S, and the labeled keyword condition C, a data fragment D is a Meaningful Data Fragment (MDF) if D is a subtree of T, and:*

i. *It conforms to at least one MSP that satisfies the Non-Redundant Semantics given S and C;*

ii. *For each label/value pair $(l{:}v) \in C$, there exists an data node n in D, such that $n.label = l$, and n contains v in its own content or the content of one of its attributes (with all-attribute scope), or one of the data nodes within the entire subtree rooted at n contains v (with all-subtree scope).*

4. MEANINGFUL SUMMARY QUERY

While we are able to achieve high accuracy and performance by adopting schema-based semantics to match structure-free query conditions for relatively simple schemas like S_{item}, directly applying the semantics to more complex schemas turns out to be very challenging. We briefly discuss the main problems[2] using the XMark schema $S_{\texttt{site}}$ and queries Q2 and Q3 in Figure 1 as examples. We first express Q2 and Q3 as labeled keyword conditions: $Q2$ = [auction:;

[2]Those problems are universal to all structure-free query mechanisms, regardless of the matching semantics (content-based or schema-based) being adopted.

person: "peter chicago"; item: "antiques"; region: "asia"], $Q3$ = [auction:; person: "chicago"].

The first problem is *lack of support for complex query semantics*. For example, Q2 involves four labels that correspond to elements across the entire schema and connected through complex join conditions; Q3 involves matching the label person twice, comparing the identity of both instances, and an aggregation function. Much of the semantics is lost when the queries are translated into the structure-free condition. Although this problem is inherent to the structure-free query model, it is exacerbated by the complexity of the schema, because the more complex the schema is, the more likely a user query will involve complex semantics.

The second problem is *increased evaluation cost*. For example, $Q2$ looks for MSPs that relate auction, person, item, and region elements. In the schema, auction is connected to person in three different ways, and item is connected to person in two different ways. This means there will be at least six schema patterns to be examined, and if we adopt the RE Semantics (instead of the NR Semantics), all will have to be evaluated against the database. The evaluation cost is further affected by redundant schema element labels (which is more likely to occur in complex schemas than in simple schemas) because the extra schema patterns need to be examined. More importantly, one element may block the matching of another element with the same label, and therefore cause relevant data fragments to be missing from the results. For example, in $S_{\texttt{site}}$, the label person is used for three different schema elements (under people, bidder, and seller), each corresponding to a different concept. Given C = [auction:; person:; note: "active"] and the XMark database shown in Figure 3, every matching semantics will match the label person to the person element under seller or bidder, instead of the one under people, because the first two are closer to the auction element, even though matching the label to the third one is likely to be relevant as well.

All those problems are caused by not allowing the users to introduce schema knowledge into the query. Structured query models allow (actually require) "full" schema knowledge in the query, but at a significantly high cost by forcing the users to study the schema. Relaxed-Structure query models, on the other hand, let users introduce "guessed" schema knowledge into the query. When the schema is complex, the guesses are often far from accurate, causing both the recall and the precision to drop significantly in the result. To address those problems, we propose the approach of *introducing "partial" schema knowledge into the query*, where the partial schema knowledge is conveyed to the user in the form of a schema summary [22], which is much easier to understand than the original schema. By allowing the users to pose queries against the schema summary and embed structure-free conditions into the query when needed, this novel query model is powerful as well as user-friendly.

We call this new query model *Meaningful Summary Query* (MSQ), and base its syntax on the existing XQuery language [20]. Figure 2 illustrates how the example queries in Figure 1 can be expressed as MSQ queries. As shown in the examples, the main innovation of the MSQ query model is that, instead of against the original schema, queries are formulated against the schema summary without the detailed knowledge of the underlying schema. This simplicity in querying is accomplished through two new language constructs, *Meaningful Fragment* (MF) function and *Meaningful Relationship* (MR) function. The MF function extracts meaningful data fragments that are within the scope of the sub-schema associated with the given abstract element, and that satisfy the structure-free condition provided by the user. The MR function, on the other hand, identifies pairs of meaningful fragments that are connected through relationship links matching user provided descriptions. We provide formal definitions of both functions in Sections 4.1 and 4.2, respectively. The basic clauses in the MSQ query model are briefly described below:

1) The set clause: A set clause of an MSQ query binds condition symbols (identified by the prefix '@') to *Entity-Matching Conditions* (see Definition 8). This clause is provided mainly for syntactic simplicity.

2) The for/let clauses: Unlike the for/let clauses in XQuery, which binds variables to specific data nodes in the database, the for/let clauses in an MSQ query bind variables (identified by the prefix '$') to meaningful data fragments in the database as returned by the MF function. We use symbol "." in two different ways. First, it is used to express the semantics of "directly reachable in summary" (e.g., the expression site.item matches the item elements directly connected with the site element). Second, it is used to connect an element with the MF function associated with the element (e.g., item.MF(@c2)).

3) The where clause: The where clauses in an MSQ query impose additional constraints on the meaningful fragments represented by the variables. In particular, the Meaningful Relationship (MR) function filters out fragment pairs that are not meaningfully related to each other.

4) The return clause: Similar to its counterpart in XQuery, the return clause in an MSQ query simply specifies the meaningful fragments or sub-fragments to be returned.

4.1 Meaningful Fragment Function

As shown in Figure 2, each MF function in an MSQ query takes as its argument an *Entity-Matching Condition* provided by the user. Intuitively, the semantics of the MF function is to extract meaningful data fragments that satisfy the Entity-Matching Condition within the scope of the sub-schema represented by the abstract element. Formally, we have the following definition.

DEFINITION 8 (ENTITY-MATCHING CONDITION). *An Entity-Matching Condition C_e is defined as a pair $\langle e, C\rangle$, where e, the focus element, can either be unspecified or be the label of an schema element, and $C = \{l_1{:}v_1, l_2{:}v_2, ...\}$ is a labeled keyword condition.*

In an MSQ query, an entity-matching condition is typically written as $e[l_1{:}v_1, l_2{:}v_2, ...]$, or simply $[l_1{:}v_1, l_2{:}v_2, ...]$ if e is unspecified.

The main difference between an Entity-Matching condition and a labeled keyword condition is the notion of *focus element*, which lets the user express the entity she would like to focus on when there are multiple entities in the meaningful schema pattern corresponding to the labeled keyword condition. As an example, consider the meaningful schema pattern in Figure 5(E), which contains two entity elements, region and item. The choice of focus entity will affect how the data fragments are organized. If region is the focus, then all relevant items of the same region will be grouped together and there will be one data fragment for each relevant region. However, if item is the focus, then each

relevant item will belong to its own data fragment. Since the users can not be expected to know the focus element all the time, it is assigned in the following order: 1) user provided label; 2) label of the abstract element if an entity element in the meaningful schema pattern has the same label; 3) label of the root element in the meaningful schema pattern. Formally, we have:

Definition 9 (Meaningful Fragment Function). *A* Meaningful Fragment *function takes two arguments: an abstract element E in the schema summary and an Entity-Matching condition C_e. It returns a set of data fragments from the database, each data fragment in the set is a meaningful data fragment matching the labeled keyword condition in C_e based on the sub-schema represented by E.*

4.2 Meaningful Relationship Function

Any relatively complex query invariably involves relating multiple entities. In a complex database where there are multiple value links connecting those entities, identifying the most likely link is a non-trivial task. For example, Q2 in Figure 1 attempts to retrieve a triple that contains meaningfully connected auction, item, and person elements. While there are multiple ways to connect them, only one connection is likely preferred by the user. Normally, join is used to identify pairs of related entities. However, correctly specifying a join requires the user to identify: 1) which attributes of the two entities should the join use; and 2) the exact schema location of these attributes. In the MSQ query model, users are shielded from such detailed schema information, specifying a join in an exact way is therefore impossible. Instead, the MSQ model adopts MR function to allow the users specify approximate join conditions. As shown in Figure 2, each MR function takes two meaningful data fragments and a keyword description, and returns true if the two fragments are meaningfully related: i.e., the two fragments are meaningfully connected and the link connecting the two fragments satisfies the keyword description. Similar to our approach in identifying meaningful data fragments, this meaningful relationship between data fragments can be defined based on meaningful relationships between schema patterns.

We first define the notion of *Property Links* of a meaningful schema pattern.

Definition 10 (Property Links of MSP). *Given a Meaningful Schema Pattern P, a value link L is a* property link *of P if it involves exactly one attribute element that belongs to an entity element in P. If L involves two such attribute elements, it is considered an* internal link *of P.*

For example, the MSP in Figure 5(E) contains the value link site/auctions/auction/itemref/@item → site/regions/region/item/@id as its property link because @id is an attribute of item. Intuitively, a property link can be considered as a "meaningful" link that connects the anchor element (e.g., item) of its own MSP with the anchor elements (e.g., auction) of other MSPs. As a result, it establishes a meaningful path between the anchor elements. Formally, we define *Meaningful Path* and MSP with *Join Semantics*.

Definition 11 (Meaningful Path). *A path between two anchor entity elements of two MSPs, P_1 and P_2, is a* Meaningful Path *if it contains a value link L such that:*

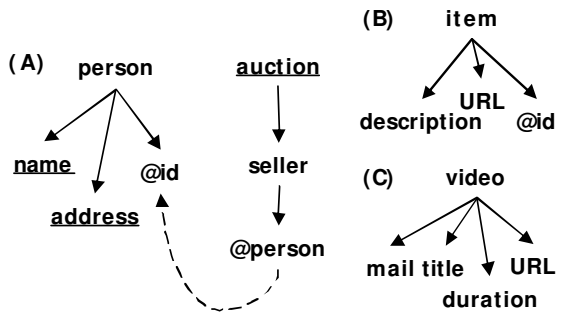


Figure 7: (A) Meaningful Schema Pattern with Join Semantics. (B) and (C) Two Basic Schema Patterns $P_{\texttt{item}}$ and $P_{\texttt{video}}$.

– L is a property link of both P_1 and P_2, or
– L is a property link of either P_1 or P_2, and no other link L' satisfy the first condition.

Definition 12 (MSP with Join Semantics). *Given two sub-schemas S_1 and S_2, two labeled keyword conditions C_1 and C_2 on S_1 and S_2 respectively, and a join keyword description V. A schema pattern P is a Meaningful Schema Pattern (MSP) according to the* Join Semantics *if P can be divided into two sub-patterns P_1 and P_2, and:*
i. *P_1 is an MSP given S_1 and C_1, P_2 is an MSP given S_2 and C_2;*
ii. *P_1 and P_2 are joined through a Meaningful Path L;*
iii. *For each value term $v \in V$, there exist $e \in L$ such that the label of e is similar to v.*

Figure 7 illustrates an MSP satisfying the Join Semantics, where the labeled keyword conditions are [name:, address:] and [auction:] respectively, and the join keyword description is "sell," which matches the label seller. There are many ways to measure the similarity between two labels, and we currently adopt an edit-distance based approach[3]. The MSP in Figure 7 (A) allows us to evaluate the MR function and identify which pairs of data fragments, one from person and the other from auction, are meaningfully related. Formally, we have:

Definition 13 (Meaningful Relationship Function). *A* Meaningful Relationship *function takes three arguments: two meaningful data fragments D_1 and D_2 (satisfying MSPs P_1 and P_2 respectively), and a keyword description V. The function returns true if:*
– there is an MSP P that satisfies the Join Semantics and connects P_1 and P_2 with meaningful link L;
– D_1 and D_2 conform to P (i.e., the value link in L is satisfied between D_1 and D_2).
The function returns false if no such MSP P exists or D_1 and D_2 do not conform to P.

5. EVALUATING MEANINGFUL SUMMARY QUERIES

The adoption of schema-based semantics in the MSQ query model suggests a novel query evaluation mechanism: an MSQ query can be converted into multiple structured queries,

[3]Other approaches include ontology-based similarity detection, etc.

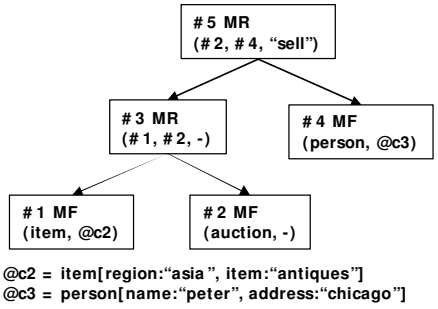


Figure 8: Step 1: Initial Evaluation Plan for Q2.

which can then be evaluated with any traditional query evaluation engine. Traditional structure-free query models have to examine all matching data fragments and filter out the ones that are not meaningful. Special access methods are often defined and integrating these into a query pipeline is a challenge. In contrast, out schema-based semantics allows us to fetch only meaningful data fragments and integrates better with the standard database query evaluation pipeline. In this section, we describe the MSQ query evaluation system, which consists of four main components: *Schema and Summary Analyzer*, *Query Analyzer*, *Query Rewriter*, *Query Evaluator*.

5.1 Schema and Summary Analyzer

As described in Section 3.1 and Section 4.2, each MSP can be considered as a set of meaningfully connected entity elements, each with its own associated attribute elements. This association between attribute and entity elements is fixed regardless of which MSQ query the MSP corresponds to. We consider each entity element, along with all its associated attribute elements, as a *Basic Schema Pattern* (BSP). Formally, we have:

DEFINITION 14 (BASIC SCHEMA PATTERN). *A* Basic Schema Pattern *(BSP) P of a given schema S is a schema pattern where: 1) there is only one entity element*[4] *e, called core element, in P; 2) all the attribute elements that belong to e are in P. We denote this BSP as P_e.*

Figure 7 (B) illustrates two example BSPs based on the sub-schema S_{item} in Figure 4. The goal of the Schema and Summary Analyzer is to identify BSPs associated with each abstract element in the summary. It accomplishes this goal by identifying each repeatable element in the sub-schema as a core element for a BSP, and traversing the subtree rooted at the core element to identify all attribute elements that belong to the core element and the BSP. A straightforward top-down algorithm is adopted and all BSPs for a sub-schema can be identified in time linear in the number of schema elements in the sub-schema. The details of the algorithm are straight-forward and not presented here.

5.2 Query Analyzer

The Query Analyzer parses the user provided MSQ query and converts it into a query evaluation logical plan based on the TLC algebra [17]. We adopt this logical representation for the following reasons. First, an evaluation plan gets rid of syntactic sugar and is conceptually simpler to transform.

[4]As mentioned in Section 3, entity elements are elements that are considered repeatable in S.

Algorithm DetermineMSP:
Input: An Entity-Matching Condition C,
a set $\mathcal{B}$ of BSPs associated with the sub-schema S
1. Initialize $M = \emptyset$;
2. **let** E be the (alphabetically) ordered list of labels in C;
3. **foreach** $p \in \mathcal{B}$:
4. Attach label bit array L to p, each bit represents an $e \in E$;
5. **foreach** $e \in E$:
6. **if** $e \in p$: $p.L[e] = 1$; // set the bit representing e
7. **if** no bit is set in $p.L$: remove p from $\mathcal{B}$;
8. **foreach** $B \subseteq \mathcal{B}$:
9. // iterate over all subsets in the order of number of BSPs
10. **if** $\exists p_1, p_2 \in B$, s.t. p_1, p_2 are not AD or SIB-D related:
11. **continue**; // Related-Entity Semantics
12. **if** $\exists B' \in M$, s.t. $B' \subset B$:
13. **continue**; // Non-Redundant Semantics
14. **initialize** label bit array L';
15. **foreach** $p \in B$: $L' = L' \mid p.L$;
16. **if** all bits are set in L':
17. add B to M;
18. **foreach** $m \in M$: attach values terms in C to m;
Output: M, the set of MSPs from S satisfying C

Figure 9: Algorithm DetermineMSP.

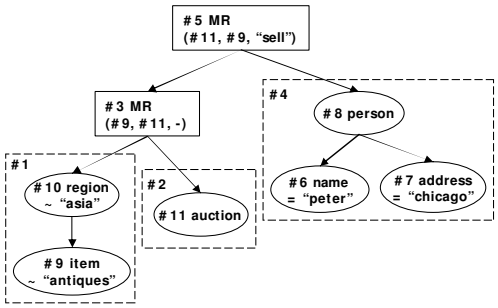


Figure 10: Step 2: Evaluation Plan for Q2 with all MF nodes substituted. Symbol $\sim$ represents the "contains" operator.

Second, the TLC algebra is well studied and its evaluation plan can be optimized and executed directly in our Timber system [12]. Figure 8 illustrates the initial evaluation plan for the example query Q2 in Figure 2. Note here that we use square boxes to represent non-executable nodes: i.e., the MF and MR nodes that can not be directly evaluated inside a query evaluation engine.

5.3 Query Rewriter

An initial plan for an MSQ query always contains some non-executable nodes, and the goal of the Query Rewriter is to substitute all such nodes with executable nodes based on the identified MSP. This rewriting is done in two stages: *substituting MF nodes* and *substituting MR nodes*.

5.3.1 Substituting MF Nodes

To substitute an MF node, we first identify the MSP from the sub-schema associated with the abstract element, based on the Entity-Matching Condition provided. This is accomplished by Algorithm DetermineMSP shown in Figure 9. The algorithm iterates through all combinations of BSPs and retains those combinations that are meaningful according to the semantics defined in Section 3.1 and satisfy all the labels in the condition. For efficiency, we alphabetically sort the element labels in the condition, and adopt a bit array data structure to indicate the presence and absence of an element label.

After an MSP is identified, we replace the MF node in the initial plan with an evaluation plan that corresponds to the

Algorithm DetermineJoinMSP:
Input: Two MSPs, M_1 and M_2, and a join description V
1. Initialize $M = \emptyset$, foundClose = **false**;
2. **foreach** property link l of M_1:
3. **let** $e_1 \in M_1$ be the entity element for l;
4. **if** l is also a property link of M_2:
5. **let** $e_2 \in M_2$ be the entity element for l;
6. **if** the path between e_1 and e_2 matches V:
7. Construct MSP m from M_1, M_2, l; add m to M;
8. foundClose = **true**;
9. **if** foundClose: **return**
10. **foreach** property link l of M_1: // vice versa for M_2
11. **let** B be the BSP, whose core element is not in M_1,
12. with l as its property link;
13. remove keywords that are already matched in M_1 from V;
14. M' = DetermineJoinMSP(B, M_2, V);
15. **foreach** MSP $m' \in M'$:
16. Construct MSP m from M_1, m', l; add m to M;
Output: M, the set of Join MSPs from S

Figure 11: Algorithm DetermineJoinMSP.

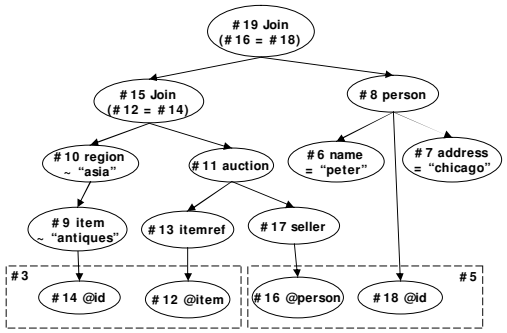


Figure 12: Step 3: Evaluation Plan for Q2 with all MR nodes substituted.

identified MSP. If there are multiple such MSPs, multiple evaluation plans are generated accordingly. Figure 10 shows the evaluation plan for Q2 after all MF nodes have been substituted.

5.3.2 *Substituting MR Nodes*

To substitute an MR node, we need to identify MSPs based on the Join Semantics as defined in Definition 12. Figure 11 illustrates the DetermineJoinMSP algorithm that accomplishes this task. Intuitively, the algorithm first tries to discover any property link that is shared by the two MSPs (lines 2-9). If unsuccessful, the algorithm continues on to discover links that are property links of only one of the two MSPs in a recursive way (lines 10-16).

Based on the identified Join MSP, we can replace the MR nodes in the evaluation plan with appropriate value joins, as shown in Figure 12. After this step, we have obtained the final evaluation plan corresponding to the user provided MSQ query.

5.4 Query Evaluator

The *Query Evaluator* component evaluates the final query plan after the rewriting. Because the final query plan is a regular structured query, the Query Evaluator can be any database engine with keyword search capability (for the "contains" operator). In our system, we use the Timber Native XML DBMS as the Query Evaluator and users are referred to [12] for more details.

6. EXPERIMENTAL EVALUATION

We evaluate our MSQ query evaluation system based on three aspects: *human cost*, *result quality*, and *performance*. We show that adopting the MSQ query model to query complex databases can reduce the human cost of query construction when compared against structured query models, while increasing the result quality and significantly reducing the query evaluation cost when compared against other alternative query models, namely simple keyword searches and Schema-Free XQuery.

	XMark	MiMI
# schema elements	327	289
# summary elements	10	10
# simple queries	14	36
# complex queries	6	16

Table 1: Dataset statistics.

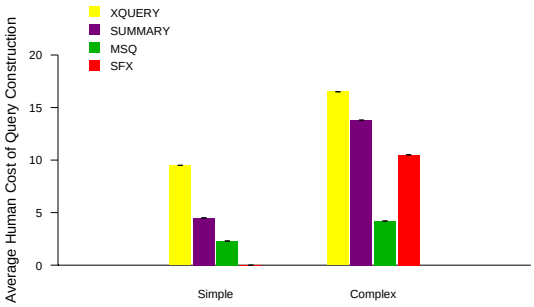


Figure 13: Human Cost of Query Construction: average number of extra schema elements visited.

The statistics of the two datasets used in the experiments, **XMark** and **MiMI**, are shown in Table 1. XMark [18] is a synthetic XML benchmark dataset about auctions, while MiMI [13] is a real world scientific dataset on protein interaction information. The summaries we use for both datasets are generated based on the methods described in [22]. (Our approach, however, are not limited to automatically generated summaries.) We divide the queries for each dataset into two categories: *simple queries*, where each query involves only one entity element, and *complex queries*, where each query involves more than one entity element. For example, in Figure 2, Q1 is a simple query, while both Q2 and Q3 are complex queries. We compare our MSQ query evaluation mechanism against the following four alternative query mechanisms: XQuery through schema exploration (XQUERY), XQuery with schema summary support (SUMMARY), Schema-Free XQuery [15] (SFX), and XSEarch [9] for labeled keyword search (XSEarch). The queries generated through XQUERY and SUMMARY are the same (i.e., standard XQueries)—the difference is that less human effort is required for the latter. We generate all other queries (i.e., MSQ queries, Schema-Free XQueries, and labeled keyword queries) using a systematic approach. In particular, labels and values for each query are extracted according to the structured query in the dataset (i.e., tag names translate to labels, keyword conditions translate to values). For MSQ, one entity-matching condition is created for all entity elements covered by the same summary element to form an MF construct, and multiple MF constructs are connected via MR constructs. For SFX, one MLCAS condition is created for all entity elements within a single ancestor descendant hierarchy, and multiple MLCAS conditions are connected through explicit value joins. The XMark database is constructed with scale factor 1.

Human Cost of Query Construction: We calculate

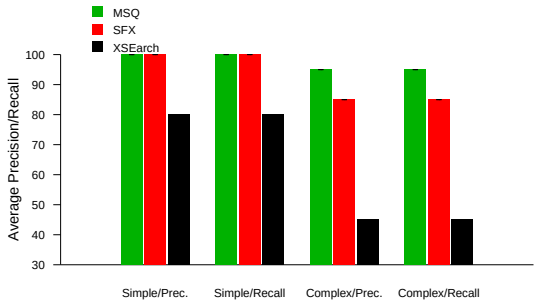


Figure 14: Result Quality: average precision and recall.

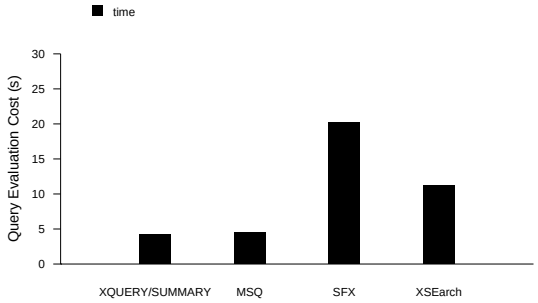


Figure 15: Average Query Evaluation Cost (s).

the human cost of query in the same way as in [22]: the cost of query construction is the number of extra schema elements a user needs to visit in order to locate all the necessary schema elements[5]. For XSEarch, the human cost is always zero since the user does not need to locate the schema elements. As shown in Figure 13, for simple queries, all alternative query mechanisms can significantly reduce the human cost of query construction. SFX even achieves zero human cost because it behaves just like XSEarch for those simple queries. For complex queries, however, both SUMMARY, which needs to explore multiple sub-schemas, and SFX, which needs to locate necessary join attributes in the schema, can incur significant human cost. Only MSQ can still maintain over 70% savings on the human cost.

Result Quality: We evaluate the result quality in terms of precision and recall. For each query, the standard set of results are obtained by evaluating the XQuery on the database, and the number of results (say N) is noted. For alternative query mechanisms that do not generate XQuery (i.e., MSQ, SFX, and XSEarch), we retrieve *up to* N results, and calculate the precision and recall based on those results. (Alternative mechanisms in fact return N results for most queries, and as a result, the precision and recall numbers coincide.) The result is shown in Figure 14 (note that XQUERY and SUMMARY always achieve 100% precision and recall and are therefore omitted from the comparison). Again, SFX and XSEarch achieve quite good result quality when the queries are simple. While both MSQ and SFX maintain reasonably good result quality when dealing with complex queries, the quality of XSEarch results drops significantly. This confirms our intuition that pure keyword search can not handle queries with complex semantics.

Query Evaluation Cost: Finally, we examine the time cost to evaluate the same query using different query mechanisms. To achieve maximum performance, various indices are built on the database (including inverted index, tagname index, join index, etc.). As shown in Figure 15, evaluating an MSQ query incurs, on average, very little overhead compared with evaluating the standard XQuery. This limited overhead is mostly associated with the query rewriting and the need to executed multiple queries when an MSQ query is translated into multiple XQueries. SFX and XSEarch, however, incur significant overhead because they can not utilize the standard query evaluation engine and have to examine many non-meaningful data fragments.

[5]In reality, we also need to account for cost associated with designing the query logic, which we ignore in this study.

7. DISCUSSION AND RELATED WORK

Our work is built upon many previously proposed structure-free query mechanisms [1, 2, 11, 10, 7, 21, 9, 15, 16] and relaxed query mechanisms [3, 15, 6, 4, 5] for both relational and XML databases. However, to the best of our knowledge, schema-based matching semantics for structure-free query mechanism is a novel concept first described here. A closely related work is [8], in which the authors proposed the interconnection semantics using query patterns. Although they do not leverage schema information (e.g., element repeatability) directly, those query patterns resemble the schema patterns described in this paper.

Advantages of schema-based semantics: Comparing with content-based matching semantics employed by previous query mechanisms like BANKS [1] and XRANK [10], schema-based matching semantics has two main advantages. First, it promises to identify data fragments that are fundamentally meaningful instead of being incidentally meaningful due to missing data nodes or poor choice of keywords by the users. For example, a query with keyword condition [`resolution:`, `duration:`] on the database conforming to the schema in Figure 4 will erroneously consider data fragments connecting `item, video, photo` as meaningful, as shown in Figure 6 (C). This is due to the fact that no content-based semantics understands that `video` and `photo` are unrelated entities. The lack of `duration` under `photo` and the lack of `resolution` under `video` lead most content-based semantics to believe that both are attributes of `item`. Second, as shown in our experimental evaluation, schema-based matching semantics allows more efficient detection of meaningful data fragments: instead of scanning the database, (a set of) structured queries can be generated and only data fragments that are meaningful are fetched and examined for their relevance.

Ranking extension: While result ranking is not the focus of this paper, we briefly describe here how ranking can be introduced into the process. There are several ranking opportunities. First, the RE Semantics and NR Semantics can be too restrictive—some schema patterns satisfying RE Semantics may still be relevant even if they do not satisfy the NR Semantics, and some schema patterns may be relevant even they do not satisfy the RE Semantics. For example, if two sibling entity elements share the same parent, and both have very low cardinality in the database, they might be considered closely related to each other. Second, even when multiple meaningful schema patterns satisfy the same semantics (NR or RE), they can be of different sizes (i.e., containing different numbers of elements) and therefore have different cohesiveness. Third, evaluation of the "contains" operator in the meaningful query pattern allows many in-

formation retrieval style ranking mechanisms to be adopted. We note here that any data fragment returned is a result of being both *meaningful* and *relevant*. The first two ranking opportunities mentioned above rank the "meaningful-ness" of the data fragments while the last one ranks their relevance. How to integrate these two aspects is an interesting research issue that we will explore in the future.

Keyword conditions: In this study, we adopt labeled keyword, instead of simple keyword, search condition for several reasons. First, several previous studies [9, 15] have shown that queries separating labels and values generally produce better result precision, which is more desired in database query tasks. Second, the labeled keyword condition enables a richer query semantics without introducing too much complexity into the query. Third, the labels in the condition enable our schema-based matching semantics. When the users can not separate labels and values, it is necessary to design methods for converting a simple keyword condition into a labeled keyword condition. This is an interesting problem, but beyond the scope of this paper.

Independence from summary generation: The utility of our MSQ query model relies heavily on the quality of schema summaries. However, it is not tied to any particular summary generation mechanism. A schema summary can either be generated automatically according to [22] or be generated by a human user. Furthermore, in the case where users are interested in only a portion of the database, a summary can even be generated for that particular part of the schema only. As a result, different users may prefer different schema summaries on which they will be posing their MSQ queries. Deciding which summary or summaries are the best for a particular group of users will be an interesting research problem.

User semantics: The user semantics of the MSQ query model is the same as Schema-Free XQuery [15] and FleXPath [6]: users get back a set of data fragments, which are extracted from the database based on a set of (labeled) keywords, and those data fragments satisfy additional structure conditions as specified in the rest of the MSQ query. As a result, we expect traditional users of both keyword search and structured search to find the MSQ query model appealing. For the former, they can impose a little bit more constraints in their queries to improve the result precision; for the latter, they can be fuzzy about some of the constraints while still maintaining high result quality.

Other Related Work: Generating good schema summaries is the topic of [22], in which we studied what defines a good schema summary and how to efficiently generate good schema summaries. Another interesting study in [14] describes a natural language interface for database querying. The authors in that study translate an NLP query into a Schema-Free XQuery [15], which is then executed by the underlying database engine. We note here that an NLP query can similarly be translated into an MSQ query, and take advantages of the many benefits the MSQ query model has over Schema-Free XQuery.

8. CONCLUSION

Correctly generating a complex structured query against a complex database is a daunting task, and current structure-free query mechanisms are inadequate in either result quality or query performance. We introduce various schema-based matching semantics and a novel summary-based query model, called Meaningful Summary Query (MSQ), to address those inadequacies. We show that MSQ queries, coupled with schema-based matching semantics, can be evaluated much more efficiently when compared with structure-free queries, and can achieve almost the same result quality as structured queries. Furthermore, the MSQ query model enables ordinary users to query on schema summary directly, without knowing the details of the underlying complex schema, thereby saving significant human cost of query construction, as demonstrated in our experimental results.

9. REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and P. Sudarshan. BANKS: Browsing and Keyword Searching in Relational Databases. In *VLDB*, 2002.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*, 2002.

[3] S. Al-Khalifa, C. Yu, and H. V. Jagadish. Querying Structured Text in an XML Database. In *SIGMOD*, 2003.

[4] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. In *WWW*, 2004.

[5] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and Content Scoring for XML. In *VLDB*, 2005.

[6] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *SIGMOD*, 2004.

[7] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-Based Keyword Search in Databases. In *VLDB*, 2004.

[8] S. Cohen, Y. Kanza, B. Kimelfeld, and Y. Sagiv. Interconnection Semantics for Keyword Search in XML. In *CIKM*, 2005.

[9] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In *VLDB*, 2003.

[10] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *SIGMOD*, 2003.

[11] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *VLDB*, 2002.

[12] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A Native XML Database. *The VLDB Journal*, 11:274–291, 2002.

[13] M. Jayapandian, A. Chapman, V. G. Tarcea, C. Yu, A. Elkiss, A. Ianni, B. Liu, A. Nandi, C. Santos, P. Andrews, B. Athey, D. States, and H. Jagadish. Michigan Molecular Interactions (MiMI): Putting the Jigsaw Puzzle Together. *Nucl. Acids Res.*, 34, 2006.

[14] Y. Li, H. Yang, and H. V. Jagadish. Constructing a Generic Natural Language Interface for an XML Database. In *EDBT*, 2006.

[15] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB*, 2004.

[16] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive Processing of Top-k Queries in XML. In *ICDE*, 2005.

[17] S. Paparizos, Y. Wu, L. V. Lakshmanan, and H. Jagadish. Tree Logical Classes for Efficient Evaluation of XQuery. In *SIGMOD*, 2004.

[18] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project, 2001.

[19] W3C. XML Schema. http://www.w3.org/TR/xmlschema-0/.

[20] W3C. XQuery 1.0: An XML Query Language.

[21] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, 2005.

[22] C. Yu and H. V. Jagadish. Schema Summarization. In *VLDB*, 2006.