Ranked Relations: Concepts, Applications, and Query Languages for Multimedia

Sibel Adalı
Rensselaer Polytechnic Institute
sibel@cs.rpi.edu

Corey Bufi
GE Corp. Research & Development
bufi@crd.ge.com

Maria-Luisa Sapino
Universita di Torino
mlsapino@di.unito.it

Abstract

In this paper, we describe the notion of a ranked relation which incorporates to the relational data model the notion of rank, i.e. ordering among tuples or objects. The ordering of tuples may be based on a single rank information, or multiple ranks combined together. We show that such relations arise naturally in many applications. We introduce an algebra for querying ranked relations and show their various order preservation properties. We then introduce a declarative query language called ISEE-QL that extends SQL with the ability to query ranked relations, as well as data sources that return ranked relations. We show that ISEE-QL is able to incorporate all ranked relation algebra operations. If we use ISEE-QL to query multiple search engines, then it can be considered a meta-search engine. However, due to the ability to integrate many different rank implementations and rank merge methods, ISEE-QL is a much more general purpose query language.

1 Introduction

In this paper, we describe the notion of a ranked relation which incorporates to the relational data model the notion of rank, i.e. ordering among tuples or objects. The ordering of tuples may be based on a single rank information, or multiple ranks combined together. We show that such relations arise naturally in many applications. We introduce an algebra for querying ranked relations and show their various order preservation properties. We then introduce a declarative query language called ISEE-QL that extends SQL with the ability to query ranked relations, as well as data sources that return ranked relations. We show that ISEE-QL is able to incorporate all ranked relation algebra operations. If we use ISEE-QL to query multiple search engines, then it can be considered a meta-search engine. However, due to the ability to integrate many different rank implementations and rank merge methods, ISEE-QL is a much more general purpose query language.

Many query languages have been proposed to search the world-wide web [4]. These languages typically incorporate queries to search engines that return document identifiers as well as additional predicates that allow conditional path expressions that describe how documents are connected to each other. In general, ranking of objects is not explicitly handled in these languages. In fuzzy query processing [6, 11], the ordering of objects with respect to a final ranking formula has been discussed where each object matches the fuzzy condition to a different degree. This considers the “membership” a function of the condition and an object. The rank of an object (as in top-k objects) is a function of the degree of membership of all objects. In similarity based approaches [2], similarity is an abstract notion that is computed by some algorithm. In both cases, functions that operate on similarity or fuzzy values assume the values returned by

*This work was supported by the National Science Foundation under grants EIA-0091505 and IIS-9876932.
different algorithms and/or evaluations are comparable and available to the integrating system. However, both of these assumptions are not true in general. For this reason, we investigate the problem of using and integrating ranks where each rank constitutes the property of a tuple with respect to the whole relation. We investigate the types of properties that can hold for rank relations in the extended version of this paper. Ranked relations provide a basis for investigating when operations in a ranked relation algebra preserve these properties. These properties can be used to construct a query optimizer for a ranked relation database. We prove these properties in the extended version of this paper.

2 Ranked Relations and Ranked Relation Databases

In this section, we describe a framework for ranked relation databases that store ranked relations. We assume two types of attributes in this framework, property attributes and rank attributes. We assume that the property attributes carry information about the properties of an object (or a tuple) while the rank attributes carry information about the relative ordering of objects (or tuples) in the relation with respect to some criteria. We will use the term attribute to refer to either type of attribute unless we explicitly specify the attribute type. A tuple in our framework is defined as a set of (attribute, value) pairs as usual. An object is a tuple with a unique object identifier. We expect the object identifier to be a property attribute.

The schema of a relation in this framework contains both property and rank attributes. We will use the notation $A_1, A_2, \ldots$ to denote property attributes, $\text{Dom}(A)$ to denote the domain of a property attribute, $q_1, q_2, \ldots$ to denote the rank attributes, $B_1, B_2, \ldots$ to denote any type of attributes, $o_1, o_2, \ldots$ to denote objects, $t_1, t_2, \ldots$ to denote tuples, $o.A$ to denote the value of attribute $A$ for object $o$ (similarly for $t.A$). The domain of a rank column is the set $\mathbb{R} \cup \{null\}$ where null is treated as the top element ($\top$). The sets $\mathcal{U}_{\text{Dom}}$ and $\mathcal{U}_{\text{Obj}}$ denote the set of all possible attribute domains and all objects respectively.

**Definition 2.1** [Ranked Relation] A ranked relation is a relation with schema $\mathcal{S}$ such that there exists at least one rank column in $\mathcal{S}$. As usual, the notation $\text{RR}(A_1, \ldots, A_k, q_1, \ldots, q_m)$ is used to denote the schema of a ranked relation $\text{RR}$ with the set of attributes $\{A_1, \ldots, A_k, q_1, \ldots, q_m\}$. We will refer to any instance of this relation as $\text{RR}$.

A ranked relation instance $\text{RR}$ with schema $\text{RR}(A_1, \ldots, A_k, q_1, \ldots, q_m)$ is a bag of tuples such that if $t \in \text{RR}$ and $t = \langle a_1, \ldots, a_k, r_1, \ldots, r_m \rangle$ then $\forall i \in \{1, \ldots, k\}, a_i \in \text{Dom}(A_i)$, and $\forall j \in \{1, \ldots, k\}, r_j \in \mathbb{R} \cup \{null\}$. We will use the term $\mathcal{U}_{\text{RR}}$ to denote the set of all possible finite ranked relation instances.

Intuitively, rank columns represent the importance or desirability of objects with respect to some specific criteria. Below we give examples of possible applications of these. We note that it is not necessary for rank columns to induce a total ordering on the objects (or tuples). It is possible that two objects have the same value and hence are equally desirable with respect to some criteria. There are some applications however where the total ordering of objects is very important. We will refer to this property as the strictness of rank columns.

**Example 2.1** In this example, we describe various applications where ordering of objects is important:

**Meta-search.** Consider a relation containing images with attributes imageURL, name, author, Image-Context, CBquery, rank which corresponds to images from the web retrieved using a content based query (CBquery). As an example, a content based query can be based on the color distribution or texture of the object. The rank attribute in this case may correspond to the rank of each image with respect to this query. Finally, the attribute called ImageContext corresponds to the text that appears near this image either in a web page or in a description field. It is possible that the same object is ranked against different databases using different features for ranking such as color distribution, texture and shape of objects. In this case, the relation may contain multiple rank columns for each different database and algorithm that is available.
In this context, a ranked relation could be viewed as the intermediate product of a meta-search query. It is possible to get a different ranking of objects if the same query is executed twice against the same search engine at different times since the underlying database or the ranking method may have changed over time. In our model, the two independent executions of a query represent two ranking executions that produce two different rank columns as we will see in Section 4.1.

Preferences. In many push based e-commerce applications, databases maintain user profiles describing the interests of different users [8]. In this case, each object may be an item that the users might be interested in, for example a book, a music CD, a movie or a house for a real-estate application. Given a user may have multiple interests, each rank column may represent the importance of the object with respect to some interest in the user’s profile. Using this information, the application may decide how to prioritize the different interests to find the best items to send to a user.

Mediated Security. It is typical in many applications to use hierarchical security levels that determine which object belongs to which security level (i.e. how secret the object is) and who has access at which level. In a mediated system that integrates information from multiple databases, it might be necessary to operate under a multitude of security levels with possibly different meanings [5]. In such a mediated application, an object can be any item from the underlying databases, and the rank column may correspond to the security level of the object in a specific database. Then, it is possible to define in the mediated system different rules on how these security levels can be merged for different objects or for different security levels.

We now describe some properties that ranked relations may satisfy. Some of these properties are important and necessary for the applications we discussed above, and we will investigate them in conjunction with the algebra operators in the extended version of this paper.

**Definition 2.2** [Strictness] A rank relation instance \( RR \) with schema \( RR(A_1, \ldots, A_k, q_1, \ldots, q_m) \) is said to be strict iff it preserves the functional dependency \( q_1, \ldots, q_m \rightarrow A_1, \ldots, A_k \) for tuples \( t \) where \( \forall j \in \{1, \ldots, m\}, t.q_j \) is not null.

**Definition 2.3** [Coherence] Given two ranked relation instances \( RR, RR' \) with schema \( RR(A_1, \ldots, A_k, q_1, \ldots, q_m) \) and \( RR'(A_1, \ldots, A_k, q'_1, \ldots, q'_m) \), two rank columns \( q, q' \) where \( q \in \{q_1, \ldots, q_m\} \) and \( q' \in \{q'_1, \ldots, q'_m\} \) are said to be coherent with respect to \( RR, RR' \) iff for all pairs of tuples \( t_1, t_2 \in RR \) and \( t'_1, t'_2 \in RR' \) where \( t_1.A = t'_1.A \) and \( t_2.A = t'_2.A \) with respect to all property attributes \( A \in \{A_1, \ldots, A_k\} \) the following is true: (1) \( t_1.q \leq t_2.q \) if and only if \( t'_1.q' \leq t'_2.q' \), and (2) \( t_1.q = t_2.q \) if and only if \( t'_1.q' = t'_2.q' \) (note that null is treated as \( \top \) and for all \( r \in \mathbb{R}, r \leq \top \) and \( \top \leq r \)).

Two lists of rank columns \( \langle q^*_1, \ldots, q^*_k \rangle \) and \( \langle q^*_1, \ldots, q^*_k \rangle \) are said to be pairwise coherent with respect to ranked relation instances \( RR, RR' \) iff for all \( i \) such that \( 1 \leq i \leq k \), the rank columns \( q^*_i \) and \( q^*_i \) are coherent with respect to \( RR, RR' \). A ranked relation instance \( RR \) is said to be inherently coherent iff all pairs of rank columns in its schema are coherent with respect to \( RR, RR' \).

Intuitively, strictness means that two objects cannot have the same non-null rank. This is definitely true for the meta-search example where objects are always returned by search engines in some strict rank ordering. Coherence is an important property in cases where two different rank columns have the same meaning. In this case, it is reasonable to assume that the same two objects will be ranked in the same order regardless of where they appear.

### 2.1 Rank Functions

In this section, we will introduce a family of functions called rank functions that calculate the rank of objects or tuples for different purposes such as estimating missing ranks of objects, combining ranks from
different columns or even assigning a rank that establishes a linear ordering over the tuples. An expression of the form $RF(RR, t, \langle q_1, \ldots, q_m \rangle)$ is interpreted as the new rank of tuple $t$ in ranked relation instance $RR$ with respect to the list of rank columns $\langle q_1, \ldots, q_m \rangle$. Hence, the rank function computes the rank of a tuple based on a ranked relation instance and specific ordering of rank columns in that relation. It is possible for example for certain columns to be more important than the others according to this denotation.

**Definition 2.4** [Rank Function (RF)] Suppose $RAttr$ is the set of all possible rank attributes, and the set $U_{RAttr}$ is the set of all possible enumerations of rank attributes given by $U_{RAttr} = \{ \{ X \} \mid X \in 2^{RAttr} \}$. A rank function $RF$ is a mapping $RF : (U_{RR} \times \bigcup_{RR \in U_{RR}} \{ t \mid t \in RR \}) \times U_{RAttr} \rightarrow R$ where $n = RF(RR, t, \langle q_1, \ldots, q_m \rangle)$ iff (i) $RR$ has schema $RR(X)$ and $\{ q_1, \ldots, q_m \} \subseteq X$, (ii) $t \in RR$, and (iii) If $t = t'$ then $RF(RR, t', \langle q_1, \ldots, q_m \rangle) = n$ (the bag semantics is preserved).

In this paper, we consider three special cases of rank functions. The first one called the rank assignment function $RaF$ is a simple lexicographic ordering of tuples with respect to a given set of rank columns, hence simulating the SQL ORDER BY clause. The second one is called a rank combination function $ReF$ which is used to combine multiple rank values into a final rank. Finally, rank estimation functions $ReF$ are used to guess missing ranks of tuples to fill in the null values. We will give an example of such a function for the meta-search example.

There are many different rank combination functions introduced in the literature. For merging similarity measures or scores, Fagin [11] used “aggregation functions” that are based on Boolean combinations of atomic formulas in fuzzy logic, such as fuzzy conjunction (e.g., as modeled by the min function) and disjunction (e.g., as modeled by the max function). Another group of rank combination methods based on voting methods is given in [10]. In this case, given two rank attributes corresponding to different ranking criteria $q_1$ and $q_2$, the Spearman footrule distance computes the absolute difference of tuples with respect to $q_1$ and $q_2$, i.e. $F(q_1, q_2, RR) = \sum_{t \in RR} |t.q_1 - t.q_2|$. The Kendall tau distance counts the number of pairwise disagreements between two lists, i.e. $K(q_1, q_2, RR) = card\{ (t_1, t_2) \mid t_1, t_2 \in RR, t_1.q_1 < t_2.q_1, \text{ but } t_1.q_2 > t_2.q_2 \}$. Given generalization of these two measures multiple rank columns, the rank combination function could be used to find a new ranking of tuples such that the overall distance of attributes $q_1, q_2$ are minimized with respect to one of the given distance measures. These rank combination functions satisfy various notions of fairness defined for voting systems.

It is also possible to consider the rank functions as a way to fill in missing ranks of objects. This is especially relevant for meta-search systems that query outside sources where a missing rank in a relation may mean that the rank of the object is not known yet or does not exist. If information is available about the type of ranking algorithm used in a specific system, then it might be possible to estimate the rank of an object [14]. Recent research [15] showed that good models of text databases can be constructed using sample queries and regression. Such a model also can be used to estimate the missing ranks of objects, based on a learnt model of the existing search systems.

### 3 Ranked Relation Algebra (RRA)

In this section, we introduce an algebra for querying ranked relations. Each operator in the algebra takes as input ranked relation instances and produces as output rank relation instances. We first introduce the elements of the query language. Given a ranked relation $RR(X)$ where $X$ is a set of attributes, the notation $RR(\langle X \rangle)$ is used to refer to a specific ordering of the attributes in $RR$. If $RR$ is a ranked relation instance and $A$ is an attribute (including rank), then $RR.A$ is a term in RRA. Similarly all ordinary constants are also terms. The notion of a boolean condition $C$ in the RRA algebra is defined in the usual way using the RRA terms and built-in selection conditions (such as $\langle, =, >, \leq, \geq \rangle$). If $C_1, C_2$ are boolean conditions, so
are $C_1 \land C_2$, $C_1 \lor C_2$, $\neg C_1$. We also assume the existence of a number of polymorphic functions $f_1, f_2, \ldots$ such as least, greatest, etc. that operate on rows of values and a number of aggregate functions $g_1, g_2, \ldots$ such as min, max, avg that operate on sets of values as usual. We assume the existence of fixed but arbitrary rank estimation (ReF) and combination (ReF) functions. Suppose $RR, RR_1, RR_2$ are ranked relation instances, $C$ is a boolean condition, $X = \{A_1, \ldots, A_k\}$ is a list of attributes.

**RANK** $(\Upsilon_{\gamma(t_1, \ldots, t_m)}):$ Given $RR(X)$ such that $\{t_1, \ldots, t_m\} \subseteq X$, and $t \not\in X$, $RR'(X \cup \{t\}) = \Upsilon_{\gamma(t_1, \ldots, t_m)}(RR)$ iff $RR' = \{t' \mid \exists t \in RR, \forall A \in X, t'.A = t.A, t'.t = \gamma(\{t_1, \ldots, t_m\})\}.

**ESTIMATE** $(\nu_{\gamma(t_1, \ldots, t_m)}):$ Given $RR(X)$ where $\{t_1, \ldots, t_m\} \subseteq X$, $RR' = \nu_{\gamma(t_1, \ldots, t_m)}(RR)$ iff $RR' = \{t' \mid \exists t \in RR, \forall A \in X, t'.A = t.A, t'.t = \text{ReF}(RR, t, \{t_1, \ldots, t_m\})\}.

**NORMALIZE** $(\omega_{\gamma(t_1, \ldots, t_m)}):$ Given $RR(X)$ where $\{t_1, \ldots, t_m\} \subseteq X$ and $t \not\in X$ then $RR'(X \cup \{t\}) = \omega_{\gamma(t_1, \ldots, t_m)}(RR)$ iff $RR' = \{t' \mid t \in RR, \forall A \in X, t'.A = t.A, t'.t = \text{ReF}(RR, t, \{t_1, \ldots, t_m\})\}.

**PROJECTION** $(\Pi_X):$ Given $RR(Y)$, $X \subseteq Y$ then $RR'(X) = \Pi_X(RR)$ iff $RR' = \{t' \mid \exists t \in RR, \forall A \in X, t'.A = t.A\}.

**VALID PROJECTION** $(\Pi_X, \gamma(t_1, \ldots, t_m))$: Given $RR(Y)$, such that $\{t_1, \ldots, t_m\} \subseteq Y$, $X \subseteq Y$, and $t \not\in Y$, then $RR'(X \cup \{t\}) = \Pi_X, \gamma(t_1, \ldots, t_m)(RR)$ iff $RR' = \{t' \mid \exists t \in RR, \forall A \in X, t'.A = t.A, t'.t = \text{ReF}(RR, \{t_1, \ldots, t_m\})\}.

Note that $\Pi_X, \gamma(t_1, \ldots, t_m) = \Upsilon_{\gamma(t_1, \ldots, t_m)}(\Pi_X(RR))$.

**MERGE** $(\oplus):$ Given $RR_1(X), RR_2(Y)$ where $X \setminus Y$ and $Y \setminus X$ contain only rank columns, $RR'(X \cup (Y \setminus X)) = RR_1 \oplus RR_2$ iff

$$RR' = (RR_1 \ast RR_2) \cup \{t' \mid \exists t_1 \in RR_1, t_1 \not\in (\Pi_X(RR_1 \ast RR_2)), \forall A \in X, t_1.A = t'.A, \forall B \in (Y \setminus X), t'.B = \text{null} \} \cup \{t' \mid \exists t_2 \in RR_2, t_2 \not\in (\Pi_Y(RR_1 \ast RR_2)), \forall A \in Y, t_2.A = t'.A, \forall B \in (X \setminus Y), t'.B = \text{null} \}.$$

Similarly, we define the following operators that are identical to their counterparts in the relational algebra by treating rank columns as any arbitrary attributes: **SELECTION**, **UNION**, **INTERSECTION**, **DIFFERENCE**, **CARTESIAN PRODUCT**, **(NATURAL/INNER JOIN)**, **(LEFT/RIGHT/FULL OUTER JOIN)**, **(RENAME)**, **(DUPLICATE ELIMINATION)**, **GROUP BY**.

**Example 3.1** In this example, we will consider a real-estate example that contains ranked relations containing information about houses from the database of a specific broker. Let $RE_1, RE_2$ be two such relations with the same schema, containing the following columns:

- **Property Attributes**: Address contains the street address of the house which is the primary key, Price is the asking price of the house, LayoutDesc describes the number of rooms and extras such as fireplace, pool, etc., HouseType describes whether this is a town house, single family house, etc. including the style in which the house was built, Neighborhood describes the school district the house is in.

- **Rank Attributes**: RPrice contains a ranking on the desirability of the price, RLayout contains a ranking on the overall importance of the various features of the layout, Rdistance contains a ranking on the address with respect to its distance various locations, RNeighborhood, RHouseType contains the compatibility of the values of the corresponding property attributes, RFireplace, RPool contains a ranking based on the existence and the type of fireplace and pool the house contains (and has null value if the house does not contain any). In all rank attributes, we assume a smaller rank means a more preferable house.
Based on this information, we compute the following ranked relation:

$\text{RR}_0 = \text{RE}_1 \cup \text{RE}_2$.

$\text{RR}_1 = \sigma_{(\text{LayoutDesc like '3br') or (LayoutDesc like '4br')}} \text{RR}_0$.

$\text{RR}_2 = \gamma_{\text{Neighborhood,HouseType,\text{min}(\text{Rnew}_1 \rightarrow \text{Rnew}_2)} ((\text{T}_{\text{Rnew}_1}(\text{Layout,RFireplace,Rdistance}) \rightarrow \text{RR}_1))}$.

$\text{RR}_3 = \Pi_{\text{Neighborhood,Rnew}_4} (\sigma_{\text{Rnew}_4 \leq 10}(\omega_{\text{Rnew}_4}(\text{Rnew}_2)(\gamma_{\text{Neighborhood},\text{avg}(\text{Rnew}_2)} \rightarrow \text{Rnew}_3 \rightarrow \text{RR}_2))$).

$\text{RR}_4 = \Pi_{\text{Address,Neighborhood,Rnew}_4} (\text{RR}_3 \bowtie \text{RR}_0)$.

In this example, we combine houses from two different databases and select only 3 or 4 bedroom houses, rank them with respect to a combination of RLayout, RFireplace, Rdistance ranking measures. Then, we consider the best house in each neighborhood with respect to this combined measure for each different house type. Then, we combine the best rank of each house type using an average for each neighborhood. We finally select the best 10 neighborhoods and the houses in these neighborhoods with respect to this criteria assuming we will visit these houses first. The normalize operator above is used to assign increasing ranks to objects after the group by operations.

**Example 3.2** Suppose, we are querying different search engines for complex information needs. The sources we query for images based on image content return tuples that contain property attributes such as the IURL containing the URL of the image, size, author, date of the image, and an attribute OType that determines the type of object the image is, i.e. painting or picture. This assumes that the sources return the URL of the images as their unique identifier. In addition, we have sources that we query for text based queries and they return similar attributes as image queries such as TURL containing the URL of the text. For text queries, the OType attribute is set to value “text”.

Suppose we issue a multi-part query MPQ which contains two independent parts, search engine queries and a combination part. We assume the search engine queries are executed first and the results are then combined using the ranked relation algebra. Query MPQ contains three independent search engine queries: (1) IQ1 is an image query that asks for the images that resemble an image image1 with respect to its shape features, (2) IQ2 is another image query that asks for the images that resemble another image image2 with respect to its color distribution, (3) TQ1 is a text query that returns all URLs that contain specific keywords. We assume the result of these three queries are independent ranked relations IQ1, IQ2, TQ1 each with a single rank column given by RIQ1, RIQ2, RTQ1 respectively.

The following relational algebra expression combines the results of IQ1, IQ2, TQ1 to return a single ranked list of most relevant objects as the result of query MPQ.

$\text{RR}_0 = (\gamma_{\text{Rnew}_1(\text{RIQ1,RIQ2}) (\text{IQ1} \bowtie \text{IQ2})}) \bowtie \text{TURL}\text{.contains} \text{IURL} (\Pi_{\text{TURL,Abstract,RTQ1}} \text{TQ1})$.

$\text{RR}_1 = \Pi_{\text{TURL,Rnew}_4} (\sigma_{\text{Rnew}_4 \leq 10}(\omega_{\text{Rnew}_4(\text{Rnew}_2,\text{RTQ1},\text{Rnew}_3)}(\gamma_{\text{TURL,\text{count}()} \rightarrow \text{Rnew}_2,\text{min}(\text{Rnew}_1) \rightarrow \text{Rnew}_3 \rightarrow \text{RR}_0)))$.

In this example, we merge the two image relations into one, putting the ranks of images in the two relations side by side and we combine the ranks. We then join the text files in TQ1 with the images they contain. We assume that this can be done by downloading the text URLs and checking the included images. Finally, we group by for each text and find the total rank of all the images they contain and rank by this figure. As a result, texts that contain many images with high rank will rank higher than the others and return the top 10 text document URLs according to this criteria.

**Example 3.3** In this example, we consider again the meta-search example by executing the same keyword search against two different search engines. The result of this search is two ranked lists TQ1, TQ2 with
associated rank columns RTQ1, RTQ2. In this example, suppose we have higher confidence in the ranks given in column RTQ1 and we are using the other search engine to increase the number of text documents that we gather. Below is another example of a combination policy that can be used to combine these two ranked relations.

\[ RR_0 = \nu_{\text{RTQ1}}(\langle \text{RTQ1}, \text{RTQ2} \rangle \ominus \text{TQ1} \oplus \text{TQ2}) \].

\[ RR_1 = \rho_{\text{TURL} \leftarrow \text{URL1}, \text{RTQ1} \leftarrow \text{RTQ11}}(\Pi_{\text{TURL, RTQ1}}(RR_0)). \]

\[ RR_2 = \varpi_{\text{New2}, \langle \text{New1}, \text{RTQ1} \rangle \ominus \gamma_{\text{TURL}}(\text{median}(\text{RTQ11}) \rightarrow \text{New1}(RR_0 \leftarrow \text{TURL hasLinkTo TURL1 RR}_1)). \]

In this example, we first combine the two relations and use both ranks to estimate the missing ranks of the tuples that are only returned by the second engine. We then join the relation by itself to find if any of the documents link to each other. In fact, we find the median rank of such documents through a group by and use this median as the primary rank parameter in the query. This in essence implements a sort of hub query that finds documents with lots of links that point to other relevant documents.

4 ISEE-QL: A Query Language for Multimedia Repositories

In this section, we introduce a query language that extends SQL with the ability to query external sources and combine the resulting ranked relations. The language incorporates operators from the ranked relation algebra in different places. We first describe how the querying of external sources is modeled in the language in Section 4.1. Then, we introduce the basic template of queries in ISEE-QL and give some examples in Section 4.2. We finish this section by some equivalence results between our algebra and this SQL based language.

4.1 Querying Existing Multimedia Repositories

In this paper, we refer to any object repository that can be queried with respect to a ranking algorithm as a multimedia repository. When a query is executed in such a repository, instead of determining if an object is an answer to the query or not, we can only determine how suitable an object is to a query given by the rank, i.e. a positional ordering of objects. We use text and image databases as motivating examples for our discussion since ranking methods have been used extensively in the information retrieval systems. However, we note that many other types of objects can be ranked using different criteria, such as when shopping for different items, it is common to specify different criteria and obtain a ranked list of results.

We first note that each repository has a predefined query language for specifying the information needs of users. We will use the set \( U_Q \) to denote the set of all possible queries that can be specified for the given repository [1, 9].

**Definition 4.1** [Multimedia Repository] A multimedia repository is given by a 4-tuple of the form \( \text{MMRep} = \langle \mathcal{D}, \mathcal{S}, U_Q, \text{RImp} \rangle \) where (i) \( \mathcal{D} \) is the multimedia database containing a set of objects, (ii) \( \mathcal{S} \) is the schema of \( \mathcal{D} \), i.e. set of all possible property attributes for objects, (iii) \( U_Q \) is the set of all valid retrieval queries for this repository representing the formulation of an information problem, (iv) \( \text{RImp} \) is a ranking implementation where \( \text{RImp} : 2^{U_Q} \times U_Q \rightarrow \{ \{ \mathcal{D} \} \ | \ \mathcal{D} \in 2^{U_Q} \} \) such that if \( \{ \mathcal{D}' \} = \text{RImp}(\mathcal{D}, RQ) \) for some database \( \mathcal{D} \) and valid retrieval query then \( \mathcal{D}' \subseteq \mathcal{D} \). In other words, each retrieval implementation maps a database to the enumeration of a subset of that database.

Intuitively, a ranking implementation returns an enumeration of the database of objects based on a given retrieval query. The position of objects in this set are assumed to correspond to their rank. In
other words, the first object in the enumeration has rank 1 with respect to the given retrieval query, the second object has rank 2, etc. It is expected that retrieval queries specify both conditions that must hold for all objects (for example each object should be an image and have size at most 2K) as well as ranking conditions (for example rank images in terms of their similarity to an input image with respect to the color saturation). Below, we define the notion of a retrieval query abstraction as a function that maps valid retrieval queries to a ranked relation with specific properties [7, 2].

**Definition 4.2** [Multimedia Query Abstraction] A multimedia query abstraction $\text{MMAbs}$ is given by a 5-tuple $\text{MMAbs} = (\text{MMRep}, RQ, S', N, RCS)$ where $\text{MMRep} = \langle D, S, UQ, \text{RImp} \rangle$ is a multimedia repository, $RQ$ is a retrieval query such that $RQ \in UQ$, $S'$ is a set of property attributes such that $S' \subseteq S$, $RCS$ is a boolean condition over the schema attributes $S'$, and $N$ is an integer (called rank upper-bound).

The result of a multimedia query abstraction $\text{MMAbs}$ is a ranked relation instance $RR$ with schema $S' \cup \{OID, \varrho\}$ where $OID$ is a unique object identifier and $\varrho \notin S'$ is a new rank column, $\text{card}(RR) \leq N$ such that for every tuple $t \in RR$ the following is true: given $\langle D' \rangle = \text{RImp}(D, RQ)$, there exists an object $o \in D'$ such that $o$ satisfies $RCS$, $o.A = t.A$ for all $A \in S'$ and $\text{pos}(o, \langle D' \rangle) = t.\varrho$.

**Example 4.1** Various components of an image query abstraction can be defined using expressions of the form: $\text{IMAGE}\_\text{LIKE}(RQ) \text{MMRep}(A_1, \ldots, A_k)[N] \text{WHERE} \text{RSC}$ where $\text{MMRep}$ is an identifier of an image repository with its underlying database and rank implementation, $RQ$ is the retrieval query, $S' = \{A_1, \ldots, A_n\}$ is the schema of the output, the top $N$ documents are returned and the relational selection condition is $RSC$.

The wrapper application implemented by a multimedia query abstraction performs the following important jobs. As an input to each abstraction, a schema of property attributes are provided. These correspond to the attributes the user is interested in. First, the abstractions executes the input retrieval query on top of the existing repository. The result is a ranked list of objects with a new rank column. Some of the objects in the database may not be returned in the enumeration since they clearly do not satisfy the relational selection condition. Then, the abstraction generates new objects by selecting from each object only the property attributes specified in the input, attaches a unique object identifier (if one is needed) and a rank column. It maps the position of each object in the list to the value of the rank attribute for the corresponding tuple. Finally, it returns the first $K$ objects from this list as a result. However, if the search engine does not have the ability to filter objects with respect to the relational search condition, then the wrapper will keep filtering the objects and retrieving new objects until a total of $K$ objects that satisfy the given selection criteria are found [13, 16].

### 4.2 Overview of the ISEE-QL Language

In this section we describe a query language called ISEE-QL that is used to formulate queries on top of existing multimedia repositories and if necessary, to combine and rank the results returned by these sources. ISEE-QL is an extension of the Structured Query Language, or SQL. The basic form of an ISEE-QL statement is defined by two required clauses SELECT, FROM, with an optional WITH QUERY clause. In addition to this, there are four optional clauses MERGE, WHERE, GROUP BY, RANK BY and has the following form:

\[
\begin{align*}
\text{SELECT} & \quad \text{select-list} \\
\text{FROM} & \quad \text{from-list} \\
\text{[ MERGE} & \quad \text{merge-expression-list }] \\
\text{[ WHERE} & \quad \text{relational-query-condition }] \\
\text{[ GROUP BY} & \quad \text{grouping-specification }]
\end{align*}
\]
Before we explain the meaning of each sentence, we note that the FROM clause in ISEE-QL queries has two main forms given with the below excerpt of the grammar:

\[
\text{from-list} \quad : \quad \text{ranked-relation-list} \quad | \quad \text{repository-schema-list WITH QUERY content-based-query-formulation-list}
\]

which means that it is possible to specify in the form clause either a list of ranked relations or a set of multimedia abstractions using the FROM ... WITH QUERY ... construct which is explained below. In the following discussion, we concentrate mostly on this new construct for incorporating queries to outside sources in an SQL type language since the use of the regular FROM clause is well understood in the literature. Below, we explain the main differences between SQL and ISEE-QL step by step.

**Rank column**: We assume all ranked relations in the ISEE-QL language have a pseudo column called rank that contains the “main” rank of the relation. As we will see shortly, the rank columns are obtained from multimedia abstractions. As different relations (say RR1, RR2) are combined using merge and join, then the resulting relation has two rank columns, namely RR1.rank, RR2.rank.

**FROM ... WITH QUERY ...** The main distinction between an ISEE-QL expression and an SQL expression is the FROM multimedia repositories WITH QUERY combination of retrieval queries construct, which replaces the usual FROM clause in SQL. Using these constructs, the user specifies pairs corresponding to multimedia query abstractions. The expression: IMAGE\_LIKE(shape(‘image1’)) ImageSource1(imageURL, context)\[10\] could be expressed in ISEE-QL as:

\[
\text{FROM ImageSource1(imageURL, context) IS1 WITH QUERY (IMAGE\_LIKE(shape(‘image1’)) IS1[10]) F1.}
\]

Furthermore, it is possible to combine multiple multimedia abstractions in a single query formulation in the WITH QUERY clause, and give a unique identifier to a single query formulation as the identifier F1 above. Intuitively, the result of a FROM ... WITH QUERY ... expression is a set of ranked relations containing the results of all multimedia abstractions contained in the expression.

**MERGE**. The merge clause makes it possible the combine the ranked relations resulting from the previous clause using one of the ranked relation algebra operations, namely MERGE, JOIN, OUTER JOIN. This is similar to the join operations that can be specified in the FROM clause of SQL. The main difference is the result of a merge clause of the form MERGE (F1.IS1, F2.IS1) M1 is a single ranked relation M1 that contains the result of \( \gamma_{M1\_rank}(F1.IS1.rank,F2.IS1.rank) \) (F1.IS1 \( \oplus \) F2.IS1). Hence, the resulting ranked relation M1 contains three rank columns, M1.rank, F1.IS1.rank, and F2.IS1.rank. Intuitively, anytime a new merge and join operation is performed, a rank combination operation is performed on the available columns of the merged/joined relations.

**WHERE**. The WHERE clause is identical to the where clause in SQL. If the multimedia abstractions used in any of the query formulations have relational selection conditions, they are placed in the WHERE clause. It is the duty of the optimizer to determine if these conditions can be isolated and pushed to the source.
GROUP BY ⟨STEP ROLLUP⟩. The grouping constructs are again very similar to their SQL equivalents. We introduce a new syntactic construct called STEP ROLLUP which is equivalent to a series of group by, aggregate operations that allow the user to feed the output of one aggregation operations to the next operation as input. The main difference between the SQL rollup is the ability to introduce new columns in one step and use them in the following steps.

RANK BY ⟨COMBINE(...)⟩. We assume all ISEE-QL expressions return a ranked relation that contains a single rank column, referred to as rank as the result of the select clause. By default, this is done by normalization based on all the rank columns in the resulting expression. This default behavior can be overwritten by a RANK BY clause which specifies explicitly which rank columns the final normalization should be based on. It is also possible to combine ranks at this final step to compute the final ranking based on a rank (combination) operation.

SELECT ... ⟨ESTIMATE(ID.rank,...)⟩. The final step of the ISEE-QL is the projection of columns that are to be included in the result. Optionally, an estimate operation can be introduced to estimate the missing ranks in an existing column, say ID.rank and use this column as the final rank column. We note that a single ISEE-QL expression may contain only one of COMBINE or ESTIMATE options for the computation of the final rank. We assume that more complex expressions can be built on top of these simpler expressions with the use of views and usual set theoretic operations in SQL, i.e. UNION, INTERSECTION and EXCEPT.

Example 4.2 Recall the ranked relation algebra expression given in Example 3.2. Below, we show how this query can be written in ISEE-QL using two different image search engines and a text search engine:

```sql
SELECT F2.TS1.pageURL, rank
FROM I:ImageSource1(imageURL, context) IS1, I:ImageSource2(imageURL, context) IS2
T:TextSource1(textURL) TS1
WITH QUERY (IMAGE LIKE(shape('image1')) IS1[10]; IMAGE LIKE(color('image2')) IS2[10]) F1
(TEXT LIKE(phrase('Pure Abstraction')) TS1[20]) F2
MERGE (F1.IS1, F1.IS2) M1 FULL JOIN F2.TS1 ON (hasLinkTo(F2.TS1.pageURL, M1.imageURL))
WHERE M1.context like '%painting%'
GROUP BY F2.TS1.pageURL
RANK BY count(DISTINCT imageURL), F2.TS1.rank, min(M1.rank)
TOP 10
```

This query contains all the elements of the ranked relational algebra expression, in addition it shows additional selection conditions on the “context” attribute which represents the text surrounding an image for images that appear in HTML documents. Each image or text source is marked in the language with the prefix I,T respectively for the query parser. In addition, the condition that a page (represented by a URL) has a link to another object (in this case an object) is implemented as a wrapper function called hasLinkTo.

It is apparent from the previous example that ISEE-QL expressions can get quite complicated easily. We do not expect that end users will be writing queries in ISEE-QL. In general, end users rarely write queries in pure SQL. The problem is even more complicated in our case due to the types of input that might be required to execute queries. For example, it is not possible to input an image in a text-based query interface. We expect that end-user implementations of ISEE-QL will involve preset query templates with parameters marked for input, such as input images. However, the ISEE-QL language and query execution engine (described in the next section) makes it possible to develop and test such interfaces with little programming effort.
Theorem 4.1 [Equivalence of Ranked Relation Algebra and ISEE-QL] Given any expression in ranked relation algebra, there exists an equivalent expression in ISEE-QL. Given any ISEE-QL expression that only involves ranked relations, there exists an equivalent ranked relation algebra expression.

Proof. The proof can be found in the extended version of this paper where a constructive algorithm to translate an expression to its equivalent form is given in both cases. □

5 Conclusions

In this paper, we introduced the notion of a ranked relation, two equivalent query languages, one based on the relational algebra and the other one on SQL that are natural extensions of their counterparts. Our notion of rank differs from the approaches in the literature where either measures that model similarity measurements between objects or boolean conditions that model fuzzy membership are used to derive ranks. In meta-search literature, the notion of rank aggregation is studied without an explicit framework for storing such ranks. We have shown the use of our language for multimedia information integration type queries. The ranked relation query languages can be used to develop many other decision support applications that integrate outside sources for ranking information and also allows the storage and reuse of ranks for various applications. In the extended version of this paper, we also investigate when various algebra operators may preserve strictness, coherence of ranked relations as well as more generic properties. This provides us with an important step towards the development of query optimization methods for ranked relations.

References


