Atlas: On the Expression of Spatial-Keyword Group Queries Using Extended Relational Constructs *

Ahmed R. Mahmood, Walid G. Aref, Ahmed M. Aly, Mingjie Tang
Purdue University, West Lafayette, IN
amahmoo, aref, aaly, tang49@cs.purdue.edu

ABSTRACT

The popularity of GPS-enabled cellular devices introduced numerous applications, e.g., social networks, micro-blogs, and crowd-powered reviews. These applications produce large amounts of geo-tagged textual data that need to be processed and queried. Nowadays, many complex spatio-textual operators and their matching complex indexing structures are being proposed in the literature to process this spatio-textual data. For example, there exist several complex variations of the spatio-textual group queries that retrieve groups of objects that collectively satisfy certain spatial and textual criteria. However, having complex operators is against the spirit of SQL and relational algebra. In contrast to these complex spatio-textual operators, in relational algebra, simple relational operators are offered, e.g., relational selects, projects, order by, and group by, that are composable to form more complex queries. In this paper, we introduce Atlas, an SQL extension to express complex spatial-keyword group queries. Atlas follows the philosophy of SQL and relational algebra in that it uses simple declarative spatial and textual building-block operators and predicates to extend SQL. Not only that Atlas can represent spatio-textual group queries from the literature, but also it can compose other important queries, e.g., retrieve spatio-textual groups from subsets of object datasets where the selected subset satisfies user-defined relational predicates and the groups of close-by objects contain miss-spelled keywords. We demonstrate that Atlas is able to represent a wide range of spatial-keyword queries that existing indexes and algorithms would not be able to address. The building-block paradigm adopted by Atlas creates room for query optimization, where multiple query execution plans can be formed.

*Walid G. Aref's research has been partially supported by the National Science Foundation under Grant III-1117766.

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA
 ISBN 978-1-4503-4589-7/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2996913.2996987

CCS Concepts

•Information systems → Query languages; Geographic information systems;

Keywords

Spatial-keyword; Query language; Relational Databases

1. INTRODUCTION

The popularity of GPS-enabled cellular devices has introduced a new platform to host numerous applications, e.g., social networks, micro-blogs, crowd-powered navigation [1], check-in, and crowd-powered reviews. These applications produce large amounts of geo-tagged textual data that need to be processed and queried in an efficient manner. Many of these queries involve textual keywords and spatial predicates. These queries are termed spatial-keyword queries. One popular spatial-keyword query in Twitter is to retrieve the most popular keywords of tweets near a specific location. Another query from Google Maps, is to identify the points of interest, e.g., hotels, in the vicinity of user's location from Google's database of textually-tagged locations.

To address the needs for processing spatio-textual data, many spatial-keyword queries have been proposed. One important class of spatial-keyword is the spatial-keyword group queries [7, 17]. In group queries, it is required to retrieve a group of objects that collectively satisfy specific spatial and textual criteria. In general, the proposed queries are considerably complex and target very specific use cases. Moreover, every type of a spatial-keyword query has a corresponding set of indexes that are fully optimized to address this specific type of query. However, an index that is efficient for one type of query is not as efficient or even not usable for another type of query. The survey of Chen et al. [10] contrasts fourteen different indexes to address only three types of spatial-keyword queries. The survey provides experimental evaluation of the performance of the indexes against the three query types. The survey also gives recommendations on what is the best type of index to use for a certain type of query under various workloads.

Using these query-specific tailored indexes to realize a spatial-keyword query service is a complex and unscalable task. Adopting this "complex-index-per-query-type" requires to disambiguate among all types of spatial-keyword queries. One will have to implement the most efficient index for every query type. This approach is not scalable. Moreover, in many cases, one needs to select a subset from a large spatial-keyword dataset, e.g., using relational predi-

cates (as in SQL), and then perform a spatial-keyword query on the selected subset. This can make using these specific and custom-tailored spatial keyword indexes not feasible, or at least sub-optimal (e.g., we may have to execute the spatial-keyword query on the entire data set in contrast to executing it only on the relevant subset).

Spatial-keyword queries retrieve geo-tagged textual data tuples that satisfy specific spatio-textual criteria. The spatial-keyword queries can be categorized into the following categories, namely: (1) *filter*, (2) *top-k*, (3) *group*, and (4) *other*. The *filter* query category filters data tuples that individually satisfy both the spatial and textual predicates of the filter query. The *top-k* query category ranks the tuples based on spatial and textual distance metrics. The *group* query category identifies groups of tuples that collectively satisfy specific spatial and textual criteria, e.g., a group of tuples of minimal spatial diameter that collectively contain all query keywords. The *other* category includes queries that do not fall in aforementioned categories, e.g., the *join* query category which joins two or more geo-tagged textual data sources. The join is based on both spatial relevance, e.g., spatial distance and textual relevance, e.g., textual overlap among tuples.

There exists several extensions to standard SQL to support spatial only [15, 37, 32, 16, 4] and textual only queries [32, 43]. However, these extensions are not expressive enough to represent complex spatial-keyword queries.

In this paper, we propose an SQL extension to support the class of spatial-keyword group queries. These extension follow the spirit of the SQL query language and relational algebra in the following sense. In relational algebra, simple relational operators are offered, e.g., relational selects, projects, joins, and group-by, that are composable to form more complex queries. Similarly, our SQL extension, termed Atlas, extends the SQL query language by offering simple declarative spatial, textual, and semantic building block operators and predicates. These extensions are composable along with the standard relational predicates (selects, joins, etc.) to form complex spatial-keyword queries.

Existing spatial-keyword queries consider either exact or approximate text matching. However, latent textual properties are of major importance, e.g., the semantics and the sentiment of text. In Atlas, we propose predicates and functions that consider latent textual properties.

In this paper, we demonstrate the expressiveness of Atlas by representing complex spatial-keyword group queries from the literature using Atlas's simple combinations of predicates. Also, we propose different query evaluation plans for these queries using Atlas's building-block operators.

The contributions of this paper are as follows:

- We introduce Atlas, an extension to the SQL query language to efficiently support spatial-keyword group queries.
- We demonstrate the expressiveness of Altas by representing various spatial-keyword search queries addressed in the literature.
- We present query evaluation plans for spatial-keyword queries using Atlas's building-block operators.
- We further introduce other predicates to support an interesting class of spatial-keyword queries that include text semantics and sentiment.

The rest of this paper proceeds as follows. Section 2 formally defines the problem and class of group queries of interest. Section 3 introduces Atlas and its extensions over SQL in support of spatial-keyword group queries. In Section 4, we use Atlas to express example grouping queries from the literature, and provide the corresponding query evaluation plans using the Atlas building-blocks. Section 5 describes novel predicates that support interesting spatial-keyword queries. Section 6 discusses related work. Section 7 concludes the paper.

2. PRELIMINARIES

In this section, we present some definitions and notations that are used throughout the paper. A spatial-keyword tuple, say O, has the following format: $O = [oid,\ loc,\ text, ...]$, where *oid* is the tuple identifier, *loc* is the geo-location of the tuple and *text* is the set of keywords associated with the tuple. A data object may contain other relational attributes such as the timestamp of tuple. The other relational attributes are orthogonal to the spatial-keyword processing and do not require special spatial-keyword handling. However, these relational attributes may be considered in the spatial keyword query, e.g., the timestamp of a tuple may be part of the where clause in a spatial-keyword query.

2.1 Spatial-keyword queries

In this paper, we study the class of spatial-keyword queries. A spatial-keyword query retrieves a tuple or a set of tuples that satisfy specific spatial and textual criteria. A spatial-keyword group query retrieves a group of tuples that collectively satisfy a group spatio-textual criteria. Notice that a single tuple within the group may not satisfy the entire group criteria. An example group query is to retrieve the set of locations that are closest to a user's locations and collectively cover the keywords "food, coffee, cinema".

In the literature there exits three main types of group queries namely;

- The *clustered group query*, e.g., the mCK query [17], retrieves the group of tuples that are closest in space and collectively cover all query keywords.
- The *single ranked group query*, e.g., the collective keyword query [8], retrieves the group of tuples that contain all query keywords, close to each other in space and close to a specific query location
- The *multiple ranked groups query* , e.g., the top-k relevant groups query [40] is very similar to the *single ranked group query*. However, this query retrieves a ranked list of k groups. The groups of tuples are ranked based on the groups' spatio-textual relevance to the query.

3. LANGUAGE SPECIFICATION

In this section, we introduce Atlas, an extension to the SQL query language that supports the language requirements described in Section 2. Atlas extends SQL to support conditional LIMITs and the retrieval of groups of tuples. The extended syntax of the SELECT statement is as follows:

```
SELECT {*|attr1 [AS alias][,attr2,...]}
FROM source_name1 [,source_name2,...]
[WHERE condition]
```

```
[ORDER BY F(arg_list)]
[LIMIT {k|condition}]

SELECT grp_attr1 [AS alias][,grp_attr2,...],
AGGR_F [AS alias](attr_list)
FROM source_name1 [,source_name2,...]
[WHERE condition]
{PARTITION BY} grp_attr_list AS group_alias
[ORDER BY F(grp_arg_list)]
[LIMIT k]
[HAVING {condition}]
```

Atlas extends the standard SQL clauses and adds two new constructs; the PARTITION BY and the conditional LIMIT. These extended and new constructs support spatial and textual functions and predicates as we explain below. The **ORDER BY** clause specifies the ranking function for tuples or groups of tuples. Ranking can be based on a function of both the spatial and textual attributes of the tuples. In top-K queries, it is beneficial to retrieve a ranked list of tuples of size K. We use the **LIMIT** clause to support top-K queries. We extend **LIMIT** to support conditions. The conditional **LIMIT** adds ranked tuples into the query result-set until a specific group condition is satisfied, e.g., to report the nearest set of tuples that collectively contain all query keywords. The conditional **LIMIT** can only be used along with an **ORDER BY** clause.

The **PARTITION BY**[1] clause behaves similar to the traditional **GROUP BY** clause. However, **PARTITION BY** reports groups of tuples not the aggregates on groups. In the following sections, we present Atlas's extensions to the SQL query language.

3.1 Atlas Predicates and Functions

In this section, we introduce the main spatial and textual functions and predicates that are crucial to the evaluation of the spatial-keyword group queries. Some of these functions and predicates already exit in other spatial and textual SQL extensions. We list these predicates and functions here for completeness of the discussion.

DIST(type,geometry1,geometry2)
This *spatial* function returns the spatial distance between two tuples. The output of this function depends on the distance **type** argument. The distance type metric can either be Euclidean, Manhattan, or road-network distances. Road-network distance requires special support from the underlying system.

OVERLAP(text1,text2)
This *textual* function returns the number of keywords shared between text1 and text2. This function is meant for keyword exact-matching. OVERLAP identifies and ranks tuples based on textual relevance, e.g., OVERLAP("food, coffee, restaurant","restaurant, cafe, sale") is 1.

CONTAINS(text1,text2)
This *textual* predicate returns TRUE if text1 contains all keywords of text2. This functions is used to check if a tuple or a group of tuples satisfies all the textual requirement of a query, e.g., CONTAINS("food, coffee, restaurant","restaurant, cafe, sale")=FALSE. CONTAINS("food, coffee, restaurant","restaurant")=TRUE.

[1]The PARTITION BY clause already exists in the SQL extensions of ORACLE and SQLServer to retrieve groups of tuples

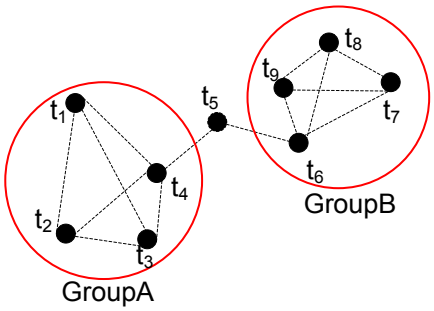


Figure 1: Example of WITHIN_DIST(Euclidean, source.loc, D)

3.2 GROUP BY/PARTITION BY Clause

In standard SQL, grouping is based on a set of relational attributes. Atlas introduces new spatial and textual grouping techniques that account for both the spatial and textual properties of the underlying data. Also, in standard SQL, only aggregates of groups are reported. This does not satisfy the needs of many spatial-keyword queries that require retrieving groups of tuples satisfying certain spatial/textual criteria. To support the retrieval of tuples within a specific group, Atlas utilizes the PARTITION BY. clause that retrieves groups of tuples in contrast to only aggregates of groups in traditional SQL. Atlas extensions to spatial and textual grouping are detailed in the following:

WITHIN_DIST(type,source.attr, D)
This operator, adopted from the similarity-group-by operator [33], identifies groups of tuples that satisfy a specific spatial/textual distance criteria. Every two tuples in a group have inter-distance that is upper-bounded by D. The input arguments are: (1) the type of distance metric used, e.g., Edit or Hamming for textual distance, and Euclidean or Manhattan for spatial distance, (2) the attribute of tuples to be grouped, and (3) the distance threshold between any pair of tuples. In contrast to the traditional group-by operator in which a tuple belongs to a single group, in this distance-based grouping, a tuple can belong to multiple groups. Figure 1 gives an example spatial grouping using the WITHIN_DIST operator. In this example, GroupA and GroupB contain the maximal set of tuples that have a maximum inter-distance of D.

PARTITIONS(source.attr,{geometry1, geometry2, ...})
This spatial grouping technique has the following input arguments: (1) the spatial attribute of tuples to be grouped, and (2) the set of spatial partitions represented as a set of GEOMETRY objects, e.g., minimum bounding rectangles (MBRs) or polygons. A tuple, say t, belongs to a group if t is inside the group's partition. Sentiment analysis over tweets within different states in the United States [44] is an example of aggregates over partitions. The data source is tweets, the grouping partitions are the boundaries of the United States, and the aggregate function is the average of the tweets' sentiments.

3.3 Aggregates

In addition to the standard SQL aggregates, e.g., AVG, MAX, MIN, COUNT, Atlas supports aggregates that are specific to spatial and textual attributes. We introduce the

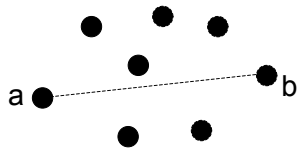


Figure 2: Diameter example.

proposed aggregates in the following sections:

CENTROID([source.attr|{loc1,loc2,...}])
This operator finds the centroid of a set of GEOMETRY points. The centroid point, say (x_c,y_c), of a set of points $(x_1,y_1),(x_2,y_2),...,(x_n,y_n)$ is calculated as follows:

$$x_c = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$y_c = \frac{\sum_{i=1}^{n} y_i}{n}$$

DIAMETER(type,[source.attr|{loc1,loc2,...}])
This operator finds the maximum distance between any pair of tuples in the input group. The first argument gives the type of the spatial distance to be used. The second argument specifies the group of tuples to be aggregated. Figure 2 gives an example of the diameter aggregate. In this figure, the aggregate value of the group is the distance between tuples a and b as it is the maximal distance between any pair of tuples.

3.4 The Conditional LIMIT

In standard SQL for relational data, TOP-K queries can be represented using the ORDER BY and LIMIT clauses. The following SQL statement resembles a TOP-K query, e.g., select the top three salaried employees:

```
SELECT * FROM Employee AS E
ORDER BY E.salary
LIMIT 3
```

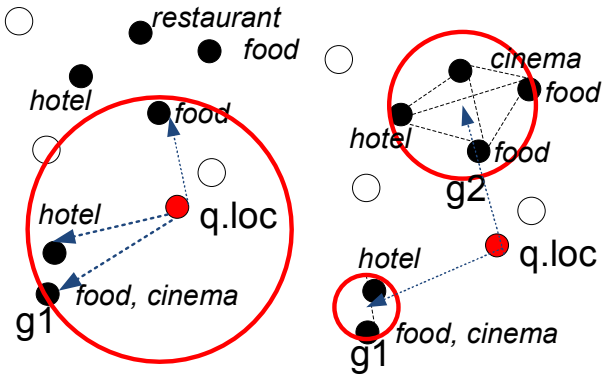
We extend this syntax to support **Conditional LIMIT**. We set the stopping criteria of the LIMIT clause to be either a number or a condition. This is useful in queries that return groups of objects ranked according to spatial/textual relevance and satisfy an overall group criteria. One example is to retrieve the nearest set of objects that contain all query keywords. In this example, the ORDER BY clause will be based on the spatial distance. The LIMIT clause will not report more tuples when the union of the keywords of the reported tuples contains all the query keywords. This is detailed in Section 4.

4. EXAMPLES

We use the following two example relations:

- Points of interest (POIs)
  This relation represents identifiable and interesting locations on the map such as restaurants and attractions. The relation is of the form <*oid, loc, text*>, where *oid*, *loc*, *text* are the identifier, spatial location, and textual description of the point of interest, respectively.

- Tweets
  Tweets are assumed to be of the form <*tid, loc, text, ts*>, where *tid*, *loc*, *text*, *ts* are the tweet identifier, spatial location, textual content, and the timestamp of an incoming tweet, respectively.



(a) Single ranked group (b) Multiple ranked groups

Figure 3: Ranked group examples.

In the following, we discuss the main types of group queries.

4.1 Clustered group

In this type of query [17], it is required to retrieve objects that are closest in space to each other and that collectively contain query keywords. As an example, we use Atlas to express the following query: find groups of POIs that collectively contain "food, cinema, hotel" and that are within 4 miles of each other. Ranking of groups is based on each group's diameter.

To find and exact answer to his query is NP-Hard [17], so we approximate this query using the WITHIN_DIST operator to identify groups of tuples that are close to each other in space. Then, these groups will be ranked by the DIAMETER of group.

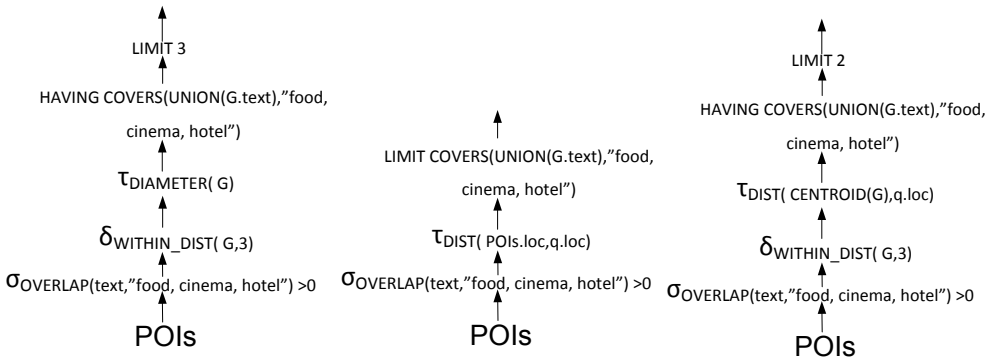
```
SELECT * FROM POIs AS p
WHERE OVERLAP("food, cinema, hotel",p.text) >0
PARTITION BY WITHIN_DIST(Euclidean,p.loc,4) AS G
ORDER BY DIAMETER (Euclidean,G)
HAVING CONTAINS(UNION(G.text),"food, cinema, hotel")
LIMIT 3
```

This query finds tuples that have overlap with query keywords. This uses the PARTITION BY and WITHIN_DIST clauses to build groups of tuples that overlap with query keywords. These groups of tuples are within a specific distance from each other, e.g., 4 miles. These groups are then filtered using the HAVING clause to only return groups that contain all query keywords. Then groups are ranked using the groups' diameter. The three groups with the least diameter are finally retrieved. Hence, this query approximately satisfies the requirements of clustered group queries [17] as it is able to identify groups of tuples within a specific maximum distance ranked based on the closeness of the groups' tuples. This query gives an approximate answer as we use a maximum distance threshold that does not exist in the original version of the query.

Figure 5 gives an example result-set for this query. Groups g1, g2 and g3 each contain all query keywords. Group g1 has the least diameter. Tuples belonging to any group are within 4 miles of each other. Figure 4(a) gives the query plan for this query.

4.2 Ranked groups

The difference between this type of query and the clus-



(a) Clustered group (b) Single ranked group (c) Multiple ranked groups

Figure 4: Query evaluation plans for group queries, τ is the ORDER BY symbol, and δ is the PARTITION BY symbol.

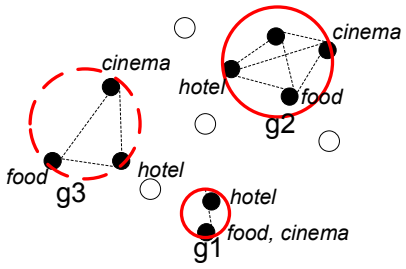


Figure 5: Clustered group example.

tered group query is that, in this type, we require the group of objects to be ranked by the distance to a specific query location. This is not the case in the clustered group query. In the ranked group query, it is required to find the set of tuples that collectively contain the a query keywords and that minimize the sum of distance to query location. Cao et al. show that this exact solution of this problem is NP-complete and provide approximations to answer such query.

We approximate this query by finding the closest group of tuples to the query location say "q.loc", such that the tuples in the group collectively contain the query keywords.

```
SELECT * FROM POIs AS p
WHERE OVERLAP("food, cinema, hotel",p.text)>0
ORDER BY DIST(Euclidean,q.loc,p.loc)
LIMIT CONTAINS(UNION(p.text,"food, cinema, ho-
tel"))
```

Figure 3(a) gives an example of the result-set of this query. Tuples in Group g1 are the closest to the query location q.loc and collectively contain the keywords of the query. Figure 4(b) gives the query plan for this query. This query plan can benefit from a textual index on the textual content of the POIs to select tuples that overlap the query keywords. This query plan orders tuples based on the distance between the tuple and the query focal point.

In traditional SQL query optimization, the query optimizer may choose a specific query plan if one step in the query plan produces data in an *interesting order* [36], e.g., having a clustered B^+-tree index on a specific attribute that is used in an ORDER BY clause. The B^+-tree can be used to retrieve tuples in the order required by the ORDER BY clause.

Similarly, if there exists a clustered spatial index on the spatial locations of POIs, an *interesting order* optimization would be to use the spatial index to fetch tuples sorted based on the spatial distance between the tuples' location and the query's focal point. This optimization fetches tuples in the order required by the ORDER BY clause and avoids the need of a separate sorting step.

Another approximation to this query is to find K groups of objects that have the following properties: (1) have their maximum inter-tuple distance be at most d, (2) collectively contain query keywords, and (3) are ranked according to the group's spatial distance to the query location.

```
SELECT * FROM POIs AS p
WHERE OVERLAP ("food, cinema, hotel", p.text) >0
PARTITION BY WITHIN_DIST(Euclidean,p.loc,3) AS G
ORDER BY DIST(Euclidean,CENTROID(G.loc),q.loc)
HAVING CONTAINS (UNION (P.text), "food, cinema,
hotel")
LIMIT 2
```

Figure 3(b) gives an example of the result-set of this query. Groups g1 and g2 have a maximum inter-tuple distance of 3. The two groups are ranked based on the distance between the centroid of the group and the query location. Figure 4(c) gives an example query plan for this query. This query plan can benefit from a textual index on the textual content of the POIs to select tuples that overlap with the query keywords. This query plan orders groups of tuples based on the distance between the centroid of the group and the query focal point. Similar to the plan in Figure 3(a), this plan in Figure 3(c) can benefit from an *interesting order* optimization if there exists a clustered spatial index on the spatial locations of POIs.

5. OTHER CONSTRUCTS

In addition to spatial-keyword group queries, there exist other important types of queries such as filter and top-k queries. These queries involve spatial predicates and textual predicates. In this section we list some relevant spatial and textual predicates that allow the representation of general spatial-keyword queries. Table 1 summarizes the Atlas predicates and functions.

5.1 Spatial Constructs

- **INSIDE(geometry1,geometry2)**
  This function determines whether geometry1 is inside geometry2. This function is useful in the filter query type.

Table 1: Atlas extensions to support spatial-keyword search queries.

	Name	Description
Spatial	INSIDE(geometry1,geometry2)	Returns True if the first argument is inside the second argument.
	OVERLAP(geometry1,geometry2)	Returns the degree of overlap between input arguments.
	DIST(type, geometry1,geometry2)	Returns the spatial distance between the first argument and the second argument based on the spatial distance *type* argument, e.g., Euclidean and Manhattan distances.
	CENTROID([source.attr|{loc1,loc2,...}])	Returns the centroid point of the input arguments.
	DIAMETER(type,[source.attr|{loc1,loc2,...}])	Returns the largest distance between any pair of points in the input arguments. The *type* argument specifies the spatial distance metric used.
	PARTITIONS(source.attr,{geometry1, geometry2,...})	Returns groups of tuples using the spatial partition polygons given in the second argument.
Textual	OVERLAP(text1,text2)	Returns the number of shared keywords between texts lists text1 and text2.
	CONTAINS(text1,text2)	Returns True if text1 contain all keywords in text2.
	UNION([source.attr|{text1,text2,...}])	Returns the union of all keywords in the input arguments.
	INTERSECTION([source.attr|{text1,text2,...}])	Returns the intersection of all keywords in the input arguments.
	SEMANTIC_SIM(text1,text2)	Returns the semantic similarity score between the input text lists.
	SENTIMENT(text)	Returns the sentiment score of the input text list.
	FREQUENT([source.attr|{text1,text2,...}],k)	Returns the K most frequent keywords in the text of first argument.
	TEXT_DIST_ANY(type,text1,text2)	Returns the minimum text distance between any pair of keywords in the input arguments text1 and text2. The type of text distance depends on the *type* argument.
	TEXT_DIST_ALL(type,text1,text2)	Returns the minimum text distance between every keyword in text1 and any keyword in text2
Hybrid	LIMIT k|condition	Returns a ranked list of tuples of either size K or when a condition is satisfied.
	WITHIN_DIST(type,source.attr,D)	Returns groups of tuples, where every pair of tuple are within D distance of each other. The *type* argument specifies the distance metric used, the source.attr specifies which attribute to group based upon.

- **OVERLAP(geometry1,geometry2)**
  This function determines the degree of overlap between two geometry objects. The OVERLAP function is usually specified in the literature as a boolean function. However, we modify it to return the degree of overlap to allow ranking tuples according to the spatial relevance in top-k queries.

Other spatial functions, e.g., EQUAL, TOUCH, and DISJOINT, that exist in spatial extensions of SQL, e.g., in ORACLE [16] and SQL Server [2] spatial extensions. These functions can be directly applied in Atlas similar to the INSIDE, OVERLAP, and DIST functions.

5.2 Textual Constructs

- **SEMANTIC_SIM(text1,text2)**: Returns the semantic similarity score between text1 and text2. The semantic similarity score depends on the underlying semantic similarity measures used. Several other scoring mechanisms [13] have been developed to measure textual semantic similarity. Any of these scoring mechanisms can be applied in Atlas.

- **SENTIMENT(text)**: Returns the sentiment score of the input text. The sentiment score resembles how positive or negative the input text is. The returned sentiment score depends on the underlying sentiment measure used [25].

- **FREQUENT([source.attr|{text1,text2,...}],k)**
  This aggregate is a keyword-based aggregate. It identifies the K most-frequent keywords in the group.

- **UNION([source.attr|{text1,text2,...}])**
  Returns the union of all keywords of input text arguments. The input argument to this function is either a set of *TEXT* objects or the textual attribute of a relation to be aggregated. For example, UNION({“food, sale”,“cafe, sale”})=“food, sale, cafe”. Another example: assume you have a data-source $SRC1$. Tuples of $SRC1$ are of the form $< id, loc, text >$, where *id* is the tuple identifier, *loc* is the location of the tuple, and *text* is the textual content of the tuple. Assume that $SRC1$ has two tuples <id1, loc1, “food, sale”> and <id2, loc2, “cafe, sale”>. UNION($SRC1.text$) returns “food, sale, cafe”.

- **INTERSECTION([source.attr|text1, text2,...])**
  Returns the intersection of all keywords of input text arguments. The input argument to this function is either a set of *TEXT* objects or the textual attribute of a relation to be aggregated. For example, INTERSECTION({“food, sale”,“cafe, sale”})=“sale”. Another example: assume you have a data-source $SRC1$. Tuples of $SRC1$ are of the

form < *id*, *loc*, *text* >, where *id* is the tuple identifier, *loc* is the location of the tuple, and *text* is the textual content of the tuple. Assume that $SRC1$ has two tuples <id1, loc1, "food, sale"> and <id2, loc2, "cafe, sale">. INTERSECTION($SRC1.text$) returns "sale".

- **TEXT_DIST_ANY(type, text1, text2)**
  Returns the minimum text distance between any pair of keywords in text1 and text2. The *type* argument specifies the textual distance metric used, e.g., Edit- or Hamming-distance. This function measures the degree of overlap between keywords of text1 and text2. The degree of overlap is based on approximate string matching, e.g., TEXT_DIST_ANY(EDIT, "susi, sale, love", "sushi, home, coupon") =1 where text1="susi, sale, love" and text2 = "sushi, home, coupon". EDIT specifies that the text distance is the Edit distance. The minimum Edit distance between any pair of keywords in text1 and text2 is 1 and it is between keywords "susi" and "sushi".

- **TEXT_DIST_ALL(type, text1,text2)**
  Returns the maximum of all the minimum text distances between every keyword in text1 and any keyword in text2. The *type* argument specifies the textual distance metric used, e.g., Edit- or Hamming-distance. This function measures the degree of containment of keywords of text1 in text2. The degree of containment is based on approximate string matching, e.g., TEXT_DIST_ALL(EDIT,"suse, rstaurnt, home", "sushi, restaurant, home, coupon") =2, where text1="suse, rstaurnt, home" and text2="sushi, restaurant, home, coupon", and EDIT specifies that the text distance is the Edit distance. For every keyword in text1 the minimum Edit distances are as follows:

  - "suse": has a minimum Edit distance of 2 to "sushi" from text2.
  - "rstaurnt": has a minimum Edit distance of 2 to "restaurant" from text2.
  - "home": has a minimum Edit distance of 0 to "home" from text2.

  The overall maximum of all the minimum Edit distances is 2.

  The reason that we use the maximum of the minimum text distances is to set an upper bound on the error for every individual keyword. If we set the error threshold based on other criteria, e.g., the sum of the minimum text distances, this may result in retrieving tuples with keywords that have high text distance from query keywords.

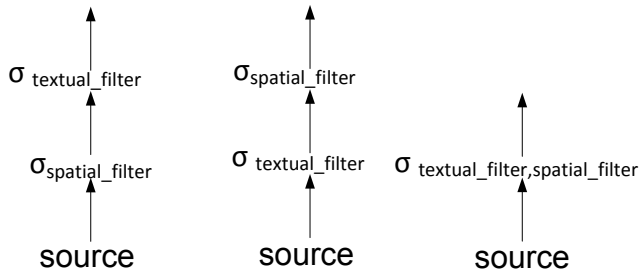
  Other textual functions, e.g., STEM, that finds the linguistic root of the keyword can also be applied in Atlas.

5.3 Examples of the Filter Queries

This category of spatial-keyword queries retrieves tuples that satisfy both a spatial and a textual selection criteria. The general form of this query is as follows:

```
SELECT * FROM source
WHERE spatial-filter-criteria
AND textual-filter criteria
```

Figure 6 gives three possible evaluation plans to process a filter query. Plan1 applies the spatial filter first. Plan1 is useful when there exists a spatial index and the selectivity



(a) Plan1 (b) Plan2 (c) Plan3

Figure 6: Query plans for a spatial-keyword filter query.

of the filter criteria is high. An analogous argument applies for Plan2. Plan3 is applicable when there exists a hybrid index that makes use of spatial and textual pruning simultaneously.

Consider the following query: Find all points of interest in MBR1 and that contain the following keywords "restaurant, deal". This type of query has been addressed in [19, 14, 10]. A variation of this query that filters according to a spatial direction [23] is also useful. We express this query as follows:

```
SELECT * FROM POIs AS p
WHERE INSIDE(p.loc, MBR1)
AND CONTAINS(p.text, "restaurant, deal")
```

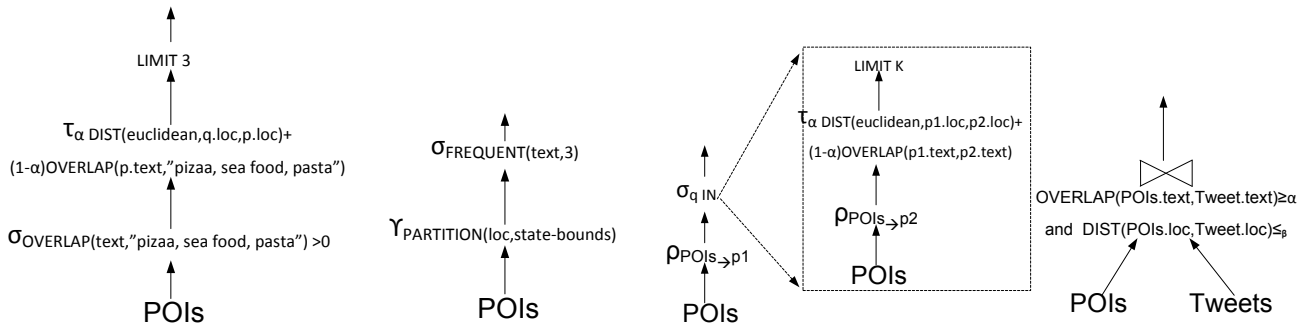
In this query, the spatial filter criterion is INSIDE(p.loc, MBR1). The textual filter criterion is CONTAINS(p.text, "restaurant, deal"). The textual matching in the previous query is containment with exact string matching. The filter query has the following flavors:

- **Overlap with exact string matching:**
  Retrieves tuples that have some overlap with the query keywords. The textual filter criterion of this query is: OVERLAP(p.text,"restaurant, deal")$\geq \epsilon$, where ϵ is the minimum number of overlapping keywords needed to qualify a tuple.

- **Containment with approximate string matching:**
  Retrieves tuples that contain all query keywords [46, 3]. The textual content of the retrieved tuples may have spelling errors, e.g., find all POIs within a specific spatial range r and have a textual description similar to all keyword "restaurant, deal". In this query, the user wants an object that is a "restaurant" with a "deal". However, there could be a typo in the query or in the textual description.

  In this case, the textual filter criterion would be: TEXT_DIST_ALL("restaurant, deal", p.text)$\leq \theta$, where θ is the maximum approximate string matching threshold. Using this textual filter criterion ensures that a retrieved object has all query keywords, or keywords that have at most θ typos. An object with a textual description of "restuarant, dael" can be retrieved if $\theta \geq 2$.

- **Overlap with approximate string matching:**
  Retrieves tuples that contain some query keywords. The textual content of the retrieved tuples may have spelling errors. The spelling error of a keyword in the retrieved tuple is bounded by a certain threshold. e.g., find all POIs within a specific spatial range r and have a textual description similar to some keyword in "restaurant, cafe, hotel".



(a) Top-K relevant tuples (b) Top-K frequent keywords (c) Reverse Top-K relevant tuples (d) Join plan

Figure 7: Query evaluation plans for TOP-K queries, τ is the ORDER BY symbol, , ρ is the rename operator, and γ is the GROUP BY symbol.

The textual filter criterion is:
TEXT_DIST_ANY(p.text, "restaurant, cafe, hotel")$\leq \theta$, where θ is the maximum approximate string matching threshold. An object with a textual description of "hotl" can be retrieved if $\theta \geq 1$.

- **Latent textual features:**
Textual semantic or sentiment functions can also be used as textual filter criteria. Consider the case when it is desired to retrieve tuples that are semantically similar [30] to query words. The textual filter criterion would be:
SEMANTIC_SIM(p.text, "restaurant, deal")$\geq \gamma$, where γ is the minimum semantic similarity score required. This query would retrieve tuples that not only contain "restaurant, deal" but also other tuples containing semantically similar keywords, e.g., "cafe , sale". Several text semantic similarity scores [13, 31] have a range between 0 and 1, where 1 stands for complete semantic similarity of text and 0 stands for total dissimilarity. In the example: SEMANTIC_SIM(p.text, "restaurant, deal")$\geq \gamma$, a high γ value, e.g., 0.9, retrieves tuples with high text semantic similarity and vice versa. A similar argument applies to the SENTIMENT textual predicate [25].

5.4 Examples of the Top-K Queries

5.4.1 Top-K relevant tuples

In this type of queries, it is required to retrieve a ranked list of the K-most relevant tuples [23]. Ranking is based on a function, say F, of both spatial and textual relevance.

```
SELECT * FROM source
[WHERE spatial-filter-criteria
AND textual-filter criteria]
ORDER BY F(spatial-relevance,textual-relevance)
LIMIT K
```

One example is to retrieve the three closest restaurants to the point $q.loc$ that offer the most from the following words: "pizza, sea food, pasta". The SQL statement for this query would be:

```
SELECT * FROM POIs p
WHERE OVERLAP(p.text,"pizza, sea food, pasta")>0
ORDER BY α * DIST(Euclidean,q.loc,p.loc)+
(1-α)*OVERLAP(p.text,"pizza, sea food, pasta")
LIMIT 3
```

where α is a query parameter to arbitrate between spatial and textual relevance. Figure 7(a) gives the query plan for this query.

This type of queries has been addressed extensively in the literature. In various forms; the snapshot version of [14, 10], the continuous version [20, 45], and the road-network distance [38] version. Another version of this query treats objects as rectangles, not as points, and considers the spatial relevance based on the spatial overlap between the query MBR and objects MBR [21].

5.4.2 Top-K frequent keywords

In this type of queries, it is required to retrieve the K-most frequent keywords [28]. One example is to retrieve the top three popular keywords in tweets in the past hour for each USA state.

```
SELECT FREQUENT(w.text,3) FROM Tweets w
GROUP BY PARTITIONS(w.loc, state-bounds)
```

Figure 7(b) gives the query evaluation plan for this query.

5.4.3 Reverse Top-K relevant tuples

This type of queries [26, 27] finds tuples that have the query tuple in their top-k relevant spatial-keyword result-set. This type of query is motivated by the need to find the influence set of the query tuple. One example is to find the restaurants that can be affected by opening a new restaurant:

```
SELECT * FROM POIs AS p1
WHERE q IN
   [SELECT * FROM POIs p2
   ORDER BY (α DIST(Euclidean,p1.loc,p2.loc)
   +(1-α)OVERLAP(p1.text,p2.text))
   LIMIT k ]
```

Figure 7(c) gives the query evaluation plan for this query.

5.5 Join Query Example

The spatial-keyword join (ST-Join) query [6, 30] identifies pairs of objects with specific spatial relevance threshold α and textual similarity threshold β, e.g., find tweets near points of interest and that have shared keywords.

```
SELECT * FROM TWEETs AS w,POIs AS p
WHERE OVERLAP(p.text,w.text)≥ α
AND DIST(Euclidean, p.loc, w.loc)≤ β
```

, where α is the minimum textual overlap threshold, β is the maximum spatial distance threshold. Figure 4(d) gives an example join query plan.

6. RELATED WORK

In this section, we survey the literature related to spatial-keyword query processing. We discuss spatial-keyword queries and spatial/textual extensions to the SQL query language.

6.1 Spatial-keyword queries

There is a large body of work that addresses spatial-keyword queries. The survey [10] discusses the following three types of spatial-keyword queries and their corresponding indexes:

- *Boolean kNN spatial-keyword query* [9]: retrieves the k tuples nearest to the query's location such that each object's text description contains all the query keywords, e.g., find the nearest three *restaurants* to my location that offer *seafood*.
- *Top-k kNN spatial-keyword query* [24]: retrieves a set of K objects ranked by a combination of the distance to the query location and the relevance to the query keywords, e.g., retrieve the three closest restaurants to the point *q.loc* that offer the most from the following words: "pizza, sea food, pasta".
- *Boolean range spatial-keyword query* [11]: retrieves all the objects inside a specific spatial range and that contain all query keywords, e.g., find *hotels* in a specific area that have a *pool*.

. However, there exist other types of queries that are not discussed in the survey [10]. We classify spatial-keyword queries into the following categories: (1) *filter*, (2) *top-k*, (3) *group*, and (4) *other*. Queries in each category differ along the following dimensions:

- *Snapshot vs. continuous query:* A snapshot query is executed once against a certain snapshot of the state of the system. A continuous query is progressively executed on the system until revoked.
- *The spatial dimension:* i.e., assuming an underlying road-network or simply assuming an Euclidean space.
- *Ranking score:* i.e., whether ranking uses spatial relevance only, textual relevance only, or a combination of both.
- *Spatial geometry:* i.e., whether the spatial attribute of tuples is represented as a point, a rectangle, or a polygon.

The *filter* category contains queries that retrieve tuples based on spatial-keyword criteria, e.g., [19, 14, 27, 10, 30]. The *boolean range query* is an example of a spatial-keyword filter query. The *top-k* category, e.g., [20, 45, 21, 38, 26, 14], includes queries that retrieve a list of k tuples ranked according to their spatial and textual relevance to the query. Both the *top-k kNN query* and the *boolean kNN query* are examples of the top-k queries. The top-k frequent keywords query on micro-blogs [28] is another variation of the top-k spatial-keyword queries that finds the most popular keywords keywords of tweets in a specific spatial area. The *group* category describes queries that identify groups of tuples based on spatial and textual proximity criteria. The *m-closest keywords* query [47, 17] finds the spatially close groups of tuples that contain all query keywords, e.g., a find the groups of building that collectively provide dining, accommodations, and shopping with as small inter-building distance as possible. Cao et al. [8] present queries that find groups of tuples nearest to the query's location. Each group of tuples contains all the query's keywords and have the least inter-tuple distance, e.g., a tourist wants to find the best group of building that collectively provide dining, accommodations, and shopping. The tourist wants to minimize his walking distance and requires that the retrieved group of building is close to his location and has as least inter-building distance as possible. [40] This work addressed finding groups of objects ranked based on the some ranking functions The *other* category includes other types of queries such as the spatial-keyword join query that identifies pairs of tuples that are spatially and textually relevant, e.g., [6, 30], e.g., find tweets within a specific distance of attractions, such that there is overlap between the textual content of a joined tweet and an attraction. Several indexes exist to address spatial-keyword queries. To index data on only the spatial properties of data tuples, we can use any of the well known spatial indexes, e.g., the R-tree [18] or its variations [5], the kd-tree [34], or the quad-tree or any of its variations [39]. Inverted lists [42] and bitmaps [41] can be used to index text data. Hybrid indexes, e.g., the IR-tree [12], index data on both the spatial and textual properties.

6.2 Spatial-keyword query languages

The only proposal for a spatial-keyword query language is described in [29]. This language targets the processing of micro-blogs, e.g., tweets. This proposal is rather limited as it handles only top-k queries and does not address queries that retrieve groups of objects. There exist several spatial extensions to the SQL query language, e.g., [15, 37, 32, 16, 4]. These extensions provide two main components for the processing of spatial queries: (1) abstract data types to represent spatial data, e.g., point, rectangle and polygon, (2) spatial predicates and functions. These spatial extensions do not support spatial-keyword queries. Melton et al. [32] and Wang et al. [43] propose extensions to the SQL query language for text search. Park et al. [35] surveys keyword search in relational databases. Lee et al. [22] proposes building blocks for spatial-textual filter and top-k queries. However, this work does not consider spatial-keyword group queries.

7. CONCLUSIONS

In this paper, we introduce Atlas as an extension to SQL to represent spatial-keyword queries. We demonstrate that Atlas can express a wide range of spatial-keyword queries using *spatial distance grouping*, *conditional limit*, and traditional spatial/textual predicates. Also, we propose to use functions for latent textual properties such as sentiment and semantic similarity to compose novel types of spatial-keyword queries. We express several complex queries from the literature to demonstrate the power of Atlas. We also propose query pipelines to evaluate spatial-keyword queries. The specification of Atlas is applicable to any spatial-keyword processing system. We plan to realize Atlas in the real-time spatial-keyword system Tornado [30]. We

also plan to study the completeness of the Atlas specifications.

8. REFERENCES

[1] Waze. https://www.waze.com, 2015.

[2] A. Aitchison. *Beginning spatial with SQL Server 2008*. Apress, 2009.

[3] S. Alsubaiee, A. Behm, and C. Li. Supporting location-based approximate-keyword queries. In *SIGSPATIAL*, pages 61–70, 2010.

[4] W. G. Aref and H. Samet. Extending a dbms with spatial operations. In *Advances in Spatial Databases*, pages 297–318, 1991.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.

[6] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *VLDB*, 6(1):1–12, 2012.

[7] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *TODS*, 40(2):13, 2015.

[8] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011.

[9] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *Scientific and Statistical Database Management*, pages 87–95, 2010.

[10] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. In *VLDB*, volume 6, pages 217–228, 2013.

[11] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *The international conference on Information and knowledge management*, pages 423–432, 2011.

[12] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *VLDB*, 2(1):337–348, 2009.

[13] C. Corley and R. Mihalcea. Measuring the semantic similarity of texts. In *ACL workshop on empirical modeling of semantic equivalence and entailment*, pages 13–18, 2005.

[14] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[15] M. J. Egenhofer. Spatial sql: A query and presentation language. *TKDE*, 6(1):86–95, 1994.

[16] S. G. Greener and S. Ravada. *Applying and Extending Oracle Spatial*. 2013.

[17] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418, 2015.

[18] A. Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[19] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *SSBDM*, pages 16–16, 2007.

[20] W. Huang, G. Li, K.-L. Tan, and J. Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *International conference on Information and knowledge management*, pages 932–941, 2012.

[21] A. Khodaei, C. Shahabi, and C. Li. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *Database and Expert Systems Applications*, pages 450–466, 2010.

[22] T. Lee, J.-w. Park, S. Lee, S.-w. Hwang, S. Elnikety, and Y. He. Processing and optimizing main memory spatial-keyword queries. *VLDB*, 9(3):132–143, 2015.

[23] G. Li, J. Feng, and J. Xu. Desks: Direction-aware spatial keyword search. In *ICDE*, pages 474–485, 2012.

[24] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.

[25] B. Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, 2012.

[26] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011.

[27] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi. Efficient algorithms and cost models for reverse spatial-keyword k-nearest neighbor search. *TODS*, 39(2):13, 2014.

[28] A. Magdy, L. Alarabi, S. Al-Harthi, M. Musleh, T. M. Ghanem, S. Ghani, and M. F. Mokbel. Taghreed: a system for querying, analyzing, and visualizing geotagged microblogs. In *SIGSPATIAL*, pages 163–172, 2014.

[29] A. Magdy and M. F. Mokbel. Towards a microblogs data management system. In *MDM*, volume 1, pages 271–278, 2015.

[30] A. R. Mahmood, A. M. Aly, T. Qadah, E. K. Rezig, A. Daghistani, A. Madkour, A. S. Abdelhamid, M. S. Hassan, W. G. Aref, and S. Basalamah. Tornado: A distributed spatio-textual stream processing system. *PVLDB*, 8(12):2020–2023, 2015.

[31] J. Martinez-Gil. An overview of textual semantic similarity measures based on web intelligence. *Artificial Intelligence Review*, 42(4):935–943, 2014.

[32] J. Melton and A. Eisenberg. Sql multimedia and application packages (sql/mm). *Sigmod Record*, 30(4):97–102, 2001.

[33] W. G. A. M. J. A. Q. M. M. M. O. Y. N. S. MingJie Tang, Ruby Y. Tahboub. Similarity group-by operators for multi-dimensional relational data. *TKDE.*, 28(2):510–523, 2016.

[34] B. C. Ooi, K. J. McDonell, and R. Sacks-Davis. Spatial kd-tree: An indexing mechanism for spatial databases. In *IEEE COMPSAC*, volume 87, page 85, 1987.

[35] J. Park and S.-G. Lee. Keyword search in relational databases. *Knowledge and Information Systems*, 26(2):175–193, 2011.

[36] R. Ramakrishnan and J. Gehrke. Database management systems. 2000.

[37] P. Rigaux, M. Scholl, and A. Voisard. *Spatial databases: with application to GIS*. Morgan Kaufmann, 2001.

[38] J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. In *The international conference on extending database technology*, pages 168–179, 2012.

[39] H. Samet. *The design and analysis of spatial data structures*, volume 85. Addison-Wesley Reading, MA, 1990.

[40] A. Skovsgaard and C. S. Jensen. Finding top-k relevant groups of spatial web objects. *The VLDB Journal*, 24(4):537–555, 2015.

[41] K. Stockinger, J. Cieslewicz, K. Wu, D. Rotem, and A. Shoshani. Using bitmap index for joint queries on structured and text data. In *New Trends in Data Warehousing and Data Analysis*, pages 1–23. 2009.

[42] A. Tomasic, H. Garcia-Molina, and K. Shoens. *Incremental updates of inverted lists for text document retrieval*, volume 23. ACM, 1994.

[43] S. Wang and K.-L. Zhang. Searching databases with keywords. *Journal of Computer Science and Technology*, 20(1):55–62, 2005.

[44] C. A. Wong, M. Sap, A. Schwartz, R. Town, T. Baker, L. Ungar, and R. M. Merchant. Twitter sentiment predicts affordable care act marketplace enrollment. *Journal of medical Internet research*, 17(2), 2015.

[45] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[46] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou. Approximate string search in spatial databases. In *ICDE*, pages 545–556, 2010.

[47] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.