Design and Evaluation of an IR-Benchmark for SPARQL Fulltext Queries

Master’s Thesis in Computer Science
by
Arunav Mishra

Supervised by
Prof. Martin Theobald

Reviewed by
Prof. Gerhard Weikum
Overview

✓ Motivation

✓ Contributions

☑ Data collection and Query Benchmark.

☑ SPAR-Key: Rewriting SPARQL-FT to SQL

☑ Results

☑ Summary
Motivation

Keyword-based document retrieval (Fulltext):

- Returns ranked documents based on a notion of relevance.
- But: Offers no precise answer to the user
  - Fails to combine information that is located across two or more documents.
  - Challenges of exact user’s intent interpretation and disambiguation.

```
“What is common between Nelson Mandela and Mother Teresa”
```

```
“This is directed by Tim Burton”
```

Results
Motivation

Knowledge retrieval on structured (semantic) data:

- Queries to be formulated in a query language like SPARQL with complex join logics.
- Highly structured Queries are basic graph pattern matching thus returns results with high precision.
- But: Difficult for a user (or any automated system) without knowledge of underlying data schema.

“Unitary state”
= Unitary_state
Or Unitary_State

“Mountain range”
= MountainRange
Or mountainRange
Desiderata

Combine both the retrieval techniques where:

- Semantic
  - Captures, interprets and disambiguates user intention.
- Keyword
  - Adds to flexibility and incorporates the notion of relevance.
- New Technique
  - Answer Jeopardy-style NL questions.

**Motivation**
Contributions

1. Unique entity-centric data collection combining structured and unstructured data.

2. Query benchmark containing 90 manually translated Jeopardy-style NL questions into SPARQL-FT queries.

3. Query engine to process SPARQL-FT queries over data collection.

4. We organized and participated in the INEX’12 LOD track.
Overview

- Motivation
  - Contributions
  
  ✓ Data collection and Query Benchmark.

- SPAR-Key: Rewriting SPARQL-FT to SQL

- Results

- Summary
INEX’12 - Wikipedia-LOD Collection

Wikipedia Entity

XML Document

Meta Data  LOD Links  Wiki-Text (Unstructured Part)  Properties (Structured Part)

Data collection and Query Benchmark.
INEX’12 - Jeopardy Task (Benchmark)

90 Jeopardy-style Natural Language Questions

manually translated

SPARQL + Fulltext

SPARQL-FT Queries

SPARQL-FT: Extend SPARQL with **FTContains** Operator:

**FTContains**(<entity>, “<keyword(s)>”):
- **Argument 1:** <entity> occurs as a Subject or Object in the SPARQL part of the query.
- **Argument 2:** The set of keywords can be used for a fulltext search on the unstructured data of the collection.
- Binds an entity to a set of keywords.
- Represents a fulltext condition.

Data collection and Query Benchmark.
Two Classes of Benchmark Queries

**Type I: 50** Jeopardy-style NL questions target single entity.

Middle name of "Naked and the Dead" author Mailer or first name of "Lucky Jim" author Amis

```
SELECT ?s WHERE {
  FILTER FTContains (?x, "Lucky Jim").
}
```

**Type 2: 40** Natural-language question target ranked list of one or more entities.

These are famous couples of actors acting in crime movies

```
SELECT Distinct ?s ?o WHERE {
  FILTER FTContains (?m1, "crime movie") .
}
```

<table>
<thead>
<tr>
<th>Target Entity</th>
<th>One answer</th>
<th>Ranked list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kingsley Amis</td>
<td></td>
<td>Brad Pitt &amp; Angelina Jolie</td>
</tr>
<tr>
<td>Tom Cruise &amp; Nicole Kidman</td>
<td></td>
<td>Tom Cruise &amp; Nicole Kidman</td>
</tr>
<tr>
<td>Will Smith &amp; Jada Pinkett</td>
<td></td>
<td>Will Smith &amp; Jada Pinkett</td>
</tr>
</tbody>
</table>
Overview

- Motivation
  - Contributions

- Data collection and Query Benchmark.

- SPAR-Key: Rewriting SPARQL-FT to SQL

- Results

- Summary
Storing RDF and Textual Data.

### Structured Data

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SUBJECT</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>PREDICATE</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>OBJECT</td>
<td>VARCHAR2 (1024)</td>
</tr>
</tbody>
</table>

Schema of **DBpediaCore** table

<table>
<thead>
<tr>
<th>Index</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dbpedia_IDX_Obj</td>
<td>(Object, Subject, Predicate)</td>
</tr>
<tr>
<td>Dbpedia_IDX_Sub</td>
<td>(Subject, Predicate, Object)</td>
</tr>
<tr>
<td>Dbpedia_IDX_Prd</td>
<td>(Predicate, Object, Subject)</td>
</tr>
</tbody>
</table>

Indices over **DBpediaCore** table

### Unstructured Data

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY_ID</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>TERM</td>
<td>VARCHAR2 (1024)</td>
</tr>
<tr>
<td>SCORE</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

Schema of **Keywords** table

<table>
<thead>
<tr>
<th>Index</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords_Entity_IDX</td>
<td>(Entity ID, Term, Score)</td>
</tr>
<tr>
<td>Keywords_TERM_IDX</td>
<td>(Term, Entity ID, Score)</td>
</tr>
</tbody>
</table>

Indices over **Keywords** table
TopX Indexer

**TopX 2.0 indexer** to create per-term inverted lists from the plain (unstructured) text content of the Wikipedia documents.

- White-space tokenizer.
- Porter stemmer.
- Stopword removal.

<table>
<thead>
<tr>
<th>Doc ID</th>
<th>Tag</th>
<th>Term</th>
<th>Pre</th>
<th>Post</th>
<th>Score</th>
<th>Max Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>sec</td>
<td>xml</td>
<td>2</td>
<td>15</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>sec</td>
<td>xml</td>
<td>10</td>
<td>8</td>
<td>0.5</td>
<td>0.9</td>
</tr>
<tr>
<td>17</td>
<td>title</td>
<td>xml</td>
<td>5</td>
<td>3</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>par</td>
<td>xml</td>
<td>6</td>
<td>4</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Bulk loader** to create the KEYWORDS table.
- Wikipedia_ID → Dbpedia_Entity,
“This mountain range is bordered by another mountain range and is a popular sightseeing and sports destination?”

```
SELECT ?p WHERE {
  FILTER FTContains(?p, "popular sightseeing sports destination") .
}
```
SQL Conjunctive Query

Consider the Example:

```sql
SELECT ?p WHERE {
    FILTER FTContains(?p, "popular sightseeing and sports destination").
}
```

```sparql
SELECT ?p WHERE {
    FILTER FTContains(?p, "popular sightseeing and sports destination").
}
```
Keyword Scores to SPO Triples

SELECT ?p WHERE {
    FILTER FTContains(?p, "popular sightseeing sports destination").
}

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>&lt;border&gt;</td>
<td>O1</td>
<td>1.0</td>
</tr>
<tr>
<td>S2</td>
<td>&lt;border&gt;</td>
<td>O2</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>&lt;border&gt;</td>
<td>O3</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>&lt;border&gt;</td>
<td>O4</td>
<td></td>
</tr>
</tbody>
</table>

Keywords K1

<table>
<thead>
<tr>
<th>Entity_ID</th>
<th>Term</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>popular</td>
<td>1.0</td>
</tr>
<tr>
<td>O4</td>
<td>popular</td>
<td>0.99</td>
</tr>
<tr>
<td>S14</td>
<td>popular</td>
<td>0.98</td>
</tr>
<tr>
<td>O7</td>
<td>popular</td>
<td>0.97</td>
</tr>
<tr>
<td>S12</td>
<td>popular</td>
<td></td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>&lt;border&gt;</td>
<td>O1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

SPAR-Key: Rewriting SPARQL-FT to SQL
# Bottlenecks and Solutions

<table>
<thead>
<tr>
<th>Bottlenecks</th>
<th>Solutions</th>
</tr>
</thead>
</table>
| The database optimizer frequently **fails** to find the best query plan.  | • Pre-decide the join ordering.  
  • Materialize temporary tables.                                          |
| **Empty Result** for large conjunctive Queries.                           | Allow keyword-match relaxation i.e., use full outer joins.                 |
| **Large cardinality** of result set from projection of Keywords table, caused by frequent terms. | Reduce the “search space” by adding more constraints: **Class Selection.** |
| **Large cardinality** of DBpediaCore table, caused due to common Predicates. |                                                                            |
| For sub queries containing only variables entire structured part of the collection is relevant. | Exploit query structure.                                                   |
| **Entity Disambiguation** and **False negatives**.                        | Perform **URI search.**                                                    |
Oracle Optimizer Hints: Ordered

```
SELECT /*+ORDERED*/ DISTINCT ENTITY_ID, MAX(SCORE) AS SCORE FROM ...
FROM
  (SELECT DISTINCT ENTITY_ID, SCORE AS SCORES FROM KEYWORDS WHERE TERM='sightse') K1
  FULL OUTER JOIN
  (SELECT DISTINCT ENTITY_ID, SCORE AS SCORES FROM KEYWORDS WHERE TERM='destin') K2
  ON K1.ENTITY_ID = K2.ENTITY_ID
  FULL OUTER JOIN
  (SELECT DISTINCT ENTITY_ID, SCORE AS SCORES FROM KEYWORDS WHERE TERM='popular') K3
  ON K1.ENTITY_ID = K3.ENTITY_ID
  FULL OUTER JOIN
  (SELECT DISTINCT ENTITY_ID, SCORE AS SCORES FROM KEYWORDS WHERE TERM='sport') K4
  ON K1.ENTITY_ID = K4.ENTITY_ID
...
```

Execution Time: From 2.237 seconds To 0.975 seconds
Class Selection & Query Structure Exploitation

- RDF data is already classified.
- Property Domain and Range, mark the required classes.
- For example,

```
SELECT ?s ?c WHERE {
?s rdf:type <http://dbpedia.org/ontology/Station>.
FILTER FTContains (?s, "most beautiful railway station").
}
```
- The entities in the marked classes are considered as candidate entities.
**URI Search**

- Titles or the entity URIs best summarizes the content and contains unique contextual information.
- In addition, entity description in a fulltext condition generally uses surface forms.
- Entity disambiguation and capture false negatives.

Create an additional URI index on tokens of the entity URIs.

<table>
<thead>
<tr>
<th>Index</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityURI_IDX</td>
<td>(Term, Entity ID, Score)</td>
</tr>
</tbody>
</table>
SELECT ?p WHERE {
  FILTER FTContains(?p, "popular sightseeing sports destination") .
}

Keywords K1
[Term= "sightseeing"]

Keywords K2
[Term= "destination"]

Keywords K3
[Term= "popular"]

Keywords K4
[Term= "sports"]

= Outer Join

= Inner Join

SPAR-Key : Rewriting SPARQL-FT to SQL
Overview

- Motivation
  - Contributions

- Data collection and Query Benchmark.

- SPAR-Key: Rewriting SPARQL-FT to SQL

✓ Results

- Summary
Experimental setup

- Dbpedia v 3.7 – created in July 2011
- YAGO2 core and full dump – created in January 2012

<table>
<thead>
<tr>
<th>Property</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Documents</td>
<td>3,164,041</td>
</tr>
<tr>
<td>XML Elements</td>
<td>1,173,255,397</td>
</tr>
<tr>
<td>Wikipedia Category Articles</td>
<td>266,134</td>
</tr>
<tr>
<td>Wikipedia Entity Articles</td>
<td>2,053,050</td>
</tr>
<tr>
<td>Wikipedia Entity Articles with Infoboxes</td>
<td>907,304</td>
</tr>
<tr>
<td>Other Wikipedia Articles</td>
<td>844,857</td>
</tr>
<tr>
<td>Resolved DBpedia Links</td>
<td>36,941,795</td>
</tr>
<tr>
<td>Resolved YAGO2 Links</td>
<td>32,941,667</td>
</tr>
<tr>
<td>Intra-Wiki Links</td>
<td>22,235,753</td>
</tr>
<tr>
<td>External Web Links</td>
<td>7,214,827</td>
</tr>
<tr>
<td>Imported DBpedia Properties</td>
<td>168,374,863</td>
</tr>
<tr>
<td>Imported YAGO2 Properties</td>
<td>23,634,511</td>
</tr>
</tbody>
</table>

**Data Collection Statistics**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Of Rows</td>
<td>20187000</td>
</tr>
<tr>
<td>Blocks</td>
<td>4288182</td>
</tr>
<tr>
<td>Distinct Rows Subject</td>
<td>18272256</td>
</tr>
<tr>
<td>Distinct Rows Object</td>
<td>26873856</td>
</tr>
<tr>
<td>Distinct Rows Predicate</td>
<td>18310</td>
</tr>
</tbody>
</table>

**DBpediaCore Table Statistics**

| Number Of Rows                                | 715609000      |
| Blocks                                        | 6415636        |
| Distinct Rows Entity_ID                       | 135082         |
| Distinct Rows Term                            | 16353141       |
## Translation Algorithms

<table>
<thead>
<tr>
<th>Components</th>
<th>SPAR-Key Identity</th>
<th>SPAR-Key Supremacy</th>
<th>SPAR-Key Ultimatum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjunctive Query</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Optimizer Hints</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Materialization Temporary tables</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Class Selection</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Query Structure Exploitation</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>URI Search</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

Results
Official INEX Results

Official INEX evaluation on 50 Queries
Gold Result Set

For Example,
<topic id="2012305">
North Dakota’s highest point is White Butte; its lowest is on this river of another colour.

Correct Answer= “Red river of the north”
“Red river of the north” <maps to>

However, a Ad-hoc search style relevance assessment marks the following entities as the relevant entities :
http://dbpedia.org/resource/geography_of_north_dakota
http://dbpedia.org/resource/pembina_county,_north_dakota
http://dbpedia.org/resource/north_dakota
http://dbpedia.org/resource/1997_red_river_flood_in_the_united_states
Re-evaluations

Jeopardy Task re-evaluations with gold result set with 26 queries

<table>
<thead>
<tr>
<th>Runs</th>
<th>I/R</th>
<th>MAP</th>
<th>P@5</th>
<th>P@10</th>
<th>P@15</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
<th>NDCG@15</th>
</tr>
</thead>
<tbody>
<tr>
<td>kas16-EXT-10</td>
<td>0.0916</td>
<td>0.0249</td>
<td>0.0353</td>
<td>0.0294</td>
<td>0.0314</td>
<td>0.0376</td>
<td>0.0336</td>
<td>0.0543</td>
</tr>
<tr>
<td>LDT2012-ruc-comb07</td>
<td>0.2484</td>
<td>0.1744</td>
<td>0.1077</td>
<td>0.0808</td>
<td>0.0692</td>
<td>0.1994</td>
<td>0.2029</td>
<td>0.2176</td>
</tr>
<tr>
<td>SPAR-Key Ultimatum</td>
<td>0.5135</td>
<td>0.3835</td>
<td>0.1826</td>
<td>0.1087</td>
<td>0.0783</td>
<td>0.4337</td>
<td>0.435</td>
<td>0.4331</td>
</tr>
</tbody>
</table>
Effect of merging Supremacy with URI search for 26 queries

<table>
<thead>
<tr>
<th>Recall Levels</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>0.55</td>
</tr>
<tr>
<td>10%</td>
<td>0.48</td>
</tr>
<tr>
<td>20%</td>
<td>0.40</td>
</tr>
<tr>
<td>30%</td>
<td>0.33</td>
</tr>
<tr>
<td>40%</td>
<td>0.26</td>
</tr>
<tr>
<td>50%</td>
<td>0.20</td>
</tr>
<tr>
<td>60%</td>
<td>0.14</td>
</tr>
<tr>
<td>70%</td>
<td>0.08</td>
</tr>
<tr>
<td>80%</td>
<td>0.03</td>
</tr>
<tr>
<td>90%</td>
<td>0.01</td>
</tr>
<tr>
<td>100%</td>
<td>0.00</td>
</tr>
</tbody>
</table>

### Results

<table>
<thead>
<tr>
<th>Runs</th>
<th>I/R</th>
<th>MAP</th>
<th>P@5</th>
<th>P@10</th>
<th>P@15</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
<th>NDCG@15</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPAR-Key Supremacy</td>
<td>0.5135</td>
<td>0.3835</td>
<td>0.1826</td>
<td>0.1087</td>
<td>0.0783</td>
<td>0.4337</td>
<td>0.435</td>
<td>0.4331</td>
</tr>
<tr>
<td>SPAR-Key Supremacy by adding entities direct matches on URIs</td>
<td>0.4923</td>
<td>0.3618</td>
<td>0.1667</td>
<td>0.0958</td>
<td>0.0667</td>
<td>0.4086</td>
<td>0.4079</td>
<td>0.4034</td>
</tr>
</tbody>
</table>
Comparison Supremacy and Ultimatum for 11 queries

<table>
<thead>
<tr>
<th>Runs</th>
<th>I/R</th>
<th>MAP</th>
<th>P@5</th>
<th>P@10</th>
<th>P@15</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
<th>NDCG@15</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPAR-Key Supremacy</td>
<td>0.608</td>
<td>0.3427</td>
<td>0.2889</td>
<td>0.1889</td>
<td>0.1407</td>
<td>0.4358</td>
<td>0.4391</td>
<td>0.4343</td>
</tr>
<tr>
<td>SPAR-Key Ultimatum</td>
<td>0.7035</td>
<td>0.4219</td>
<td>0.26</td>
<td>0.15</td>
<td>0.1067</td>
<td>0.5081</td>
<td>0.4764</td>
<td>0.4655</td>
</tr>
</tbody>
</table>
Extended Evaluation of Ultimatum Components on 11 queries

### Results

<table>
<thead>
<tr>
<th>Runs</th>
<th>I/R</th>
<th>MAP</th>
<th>P@5</th>
<th>P@10</th>
<th>P@15</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
<th>NDCG@15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploiting Dbpedia class information</td>
<td>0.6017</td>
<td>0.3377</td>
<td>0.26</td>
<td>0.16</td>
<td>0.12</td>
<td>0.4251</td>
<td>0.3965</td>
<td>0.3922</td>
</tr>
<tr>
<td>Boosting direct matches on entity URIs</td>
<td>0.6365</td>
<td>0.3836</td>
<td>0.26</td>
<td>0.15</td>
<td>0.1067</td>
<td>0.4753</td>
<td>0.4435</td>
<td>0.4327</td>
</tr>
<tr>
<td>Merging both the approaches</td>
<td>0.7035</td>
<td>0.4219</td>
<td>0.26</td>
<td>0.15</td>
<td>0.1067</td>
<td>0.5081</td>
<td>0.4764</td>
<td>0.4655</td>
</tr>
</tbody>
</table>
Overview

- Motivation
  - Contributions

- Data collection and Query Benchmark.

- SPAR-Key: Rewriting SPARQL-FT to SQL

- Results

✓ Summary
Summary

- A fast shot approach towards processing and evaluating SPARQL queries with full-text conditions.
- Implemented using relational databases to show proof of concept and retrieve answers to the NL questions (there is no publicly available engine).
- Improved result quality compared to full-text query processing engines.
- As future work we would work on getting better results by utilizing dictionaries and automatic query expansion.
- The long-term goal is indeed to provide a custom engine that processes these queries.
Demonstration
THANK YOU
SELECT ?p WHERE {
  ?p <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> 
  <http://dbpedia.org/ontology/MountainRange>.
  FILTER FTContains(?p, "popular sightseeing and sports destination") .
}

= Outer Join
= Inner Join

DBPEDIACORE = An instance of DBPEDIACORE table
Keywords = An instance of KEYWORDS Table

Select Query

TAB1

DBPEDIACORE

T1

KEYWORDS

KEYS0

TAB2

DBPEDIACORE

T2

Adding constraint on Predicate

Keywords K1

Keywords K2

Keywords K3

Keywords K4

Adding constraint on Predicate and Object

Results
Exploiting the structure of query for class identification

**Star Structure**

```
SELECT ?c WHERE {
  <museum> <located> ?a.
  FILTER FTContains(?a,"...").
}
```

**Chain Structure**

```
Station <type> ?s <type> ?c
```

Results
Backup:

Creating temporary table for fulltext condition.

CREATE TABLE KEYS0 AS SELECT /*+ORDERED*/ * FROM
(SELECT /*+ORDERED*/ * FROM
(SELECT /*+ORDERED*/ DISTINCT ENTITY_ID , MAX(SCORESS) AS SCORE FROM
(SELECT DISTINCT K1.ENTITY_ID AS ENTITY_ID ,
(NVL(K1.SCORES,0)+1 + NVL(K2.SCORES,0) + NVL(K3.SCORES,0) + NVL(K4.SCORES,0) ) AS SCORESS FROM
(SELECT DISTINCT ENTITY_ID , SCORE AS SCORES FROM KEYWORDS WHERE TERM='sightseeing') K1
FULL OUTER JOIN
(SELECT DISTINCT ENTITY_ID ,SCORE AS SCORES FROM KEYWORDS WHERE TERM='destination') K2
ON K1.ENTITY_ID = K2.ENTITY_ID
FULL OUTER JOIN
(SELECT DISTINCT ENTITY_ID ,SCORE AS SCORES FROM KEYWORDS WHERE TERM='popular') K3
ON K1.ENTITY_ID = K3.ENTITY_ID
FULL OUTER JOIN
(SELECT DISTINCT ENTITY_ID ,SCORE AS SCORES FROM KEYWORDS WHERE TERM='sport') K4
ON K1.ENTITY_ID = K4.ENTITY_ID
ORDER BY SCORESS DESC) GROUP BY ENTITY_ID ORDER BY SCORE DESC )
Temporary table for triple patterns.

CREATE TABLE TAB2 AS
SELECT SUBJECT , PREDICATE , OBJECT , ( NVL(KEYS0.score,0) ) AS SCORE
FROM
(SELECT * FROM dbpediacore3 t1
WHERE
  t1.PREDICATE='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
AND  t1.OBJECT='http://dbpedia.org/ontology/MountainRange') TEMP
INNER JOIN KEYS0
ON TEMP.SUBJECT=KEYS0.ENTITY_ID

CREATE TABLE TAB1 AS
SELECT SUBJECT , PREDICATE , OBJECT , ( NVL(KEYS0.score,0) ) AS SCORE
FROM
(SELECT * FROM dbpediacore3 t2
WHERE
  t2.PREDICATE='http://dbpedia.org/ontology/border') TEMP
INNER JOIN KEYS0
ON TEMP.OBJECT=KEYS0.ENTITY_ID
Final Select Query

```
SELECT /*+Ordered*/ p, MAX(SCORE) AS SCORE_MAX FROM (SELECT /*+ORDERED*/ DISTINCT TAB2SUBJECT AS p, ( NVL(TAB2.score,0 ) + NVL(TAB1.score,0 )) AS score FROM TAB2, TAB1 WHERE TAB1.OBJECT = TAB2SUBJECT ) GROUP BY p ORDER BY SCORE_MAX DESC
```

Dropping the Intermediate Tables

```
DROP TABLE KEYS0
DROP TABLE TAB2
DROP TABLE TAB1
```