

Laboratory: Key Discovery

Pablo Mollá Chárlez

January 21, 2025

Contents

1	Hands-on Key Discovery	2
1.1	SAKey on a small dataset	2
1.2	SAKey on a bigger dataset from DBpedia	2
1.3	Use the keys for data linking	3
1.4	Going further	3
2	Solution	4
2.1	SAKey on a small dataset	4
2.2	SAKey on a bigger dataset from DBpedia	5
2.3	Use the keys for data linking	8
2.4	Going Further: Vickey & Rocker	12
2.4.1	Vickey	12
2.4.1.1	OAEL_2011_Restaurant_1.nt	12
2.4.1.2	Library-6k.nt	12
2.4.1.3	Library-196k.nt	12
2.4.2	Rocker	13
2.4.2.1	OAEL_2011_Restaurant_1.nt	13
2.4.2.2	Library-6k.nt	13
2.4.2.3	Library-196k.nt	14

1 Hands-on Key Discovery

This part is dedicated to the use of some existing key discovery tools. The idea is to run them, analyse the results and compare their results.

You can find at the following link the needed material for this hands-on session: [Ecampus](#)

You are asked to deliver your work on sub-sections 2.2, 2.3 and section 3 (as a bonus). The submission link is: [Submission Link](#).

We will use **SAKey** that discovers $n - \textit{almost}$ keys with $n - 1$ is the number of allowed exceptions for a set of properties to be a key. Bellow, is the readme file.

```
-----
SAKey takes as input two parameters:
  a) the file where you want the key discovery to take place
  b) the number of exceptions n.
Note that the n will lead to the discovery of
n-non keys and then the extraction of (n-1)-almost keys

To run SAKey use the following command:
java -jar "jar_fullpath"/SAKey.jar "file_fullpath" n

For example, in the following command:
java -jar /Users/danai/SAKey.jar /Users/danai/datasets/DB_Lake.nt 1

In this example, SAKey will discover first 1-non keys and then derive
0-almost keys for the file /Users/danai/datasets/DB_Lake.nt.

The file has to be in a n-triple format.
For example:
<http://www.okkam.org/oaie/restaurant1-Restaurant56>
<http://www.okkam.org/ontology_restaurant1.owl#phone_number> "718/522-5200" .
Or
http://www.okkam.org/oaie/restaurant1-Restaurant56
http://www.okkam.org/ontology_restaurant1.owl#phone_number
718/522-5200
-----
```

1.1 SAKey on a small dataset

1. Run **SAKey** on the dataset "OAEI_2011_Restaurant_1.nt" with $n = 1$, then copy the results that are displayed in the terminal and saved them in a text file **sakey-keys-restaurant-n1.txt**.
2. Run **SAKey** on the dataset "OAEI_2011_Restaurant_1.nt" with $n = 7$, then copy the results that are displayed in the terminal and saved them in a text file **sakey-keys-restaurant-n7.txt**.
3. Compare the obtained keys, in terms of number of keys and minimality (i.e. the number of properties involved in the keys).

1.2 SAKey on a bigger dataset from DBpedia

1. Run **SAKey** on the dataset "library-6k.nt" with $n = 1$, and redirect the results to a text file **sakey-keys-library-6k-n1.txt**.
2. Run **SAKey** on the dataset "library-196k.nt" with $n = 1$, and redirect the results to a text file **sakey-keys-library-196k-n1.txt**.
3. Compare the results in terms of number of keys and non-keys. If no keys found, try with a higher value of n .

1.3 Use the keys for data linking

Using the keys that you obtained on dbpedia **library-6k.nt** and by following the **SPARQL** query given as example bellow, write SPARQL queries for keys that you choose and run them on Yasgui. Bellow an example of a **SPARQL Query** for a key **hasKey(Book(rdfs:label)())**.

NB The limit 1 is important to have reasonable time execution of the query.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>

# Everything that starts with the # character is a comment
# INPUT-1: one key detected in DBpedia for the class (e.g Book)
# In this example query, the ranked key contains only the property rdfs:label

SELECT DISTINCT ?dbpediaID1 ?dbpediaID2
WHERE {
    # first book entity
    ?dbpediaID1 rdf:type dbo:Book .
    ?dbpediaID1 rdfs:label ?labelDBP1 .
    ?dbpediaID2 rdf:type dbo:Book .
    ?dbpediaID2 rdfs:label ?labelDBP2 .

    FILTER(?labelDBP1 != ?labelDBP2)
}
limit 1
```

1.4 Going further

Redo the exercise 3.1 and 3.2 using the three tools **SAKey**, **Vickey** and **Rocker** and compare the results in terms of:

- Number of keys,
- Run-time,
- The average size of keys (if possible)

For more details on how to run **Vickey**, please use the following Github Link and for futher information on **Rocker**, use this Rocker Link. This is an example of how to use **Vickey**:

```
java -jar ./Lab/vickey.jar -mins 5 -p ./Sakey-handson-materials/datasets/DB Lake.nt.
```

```
----- READ ME -----
To use vickey.jar, run in a command line:

$ java -jar vickey.jar -mins <support-threshold> [-p] <input-file>

<support-threshold>: minimal number of entities that must be satisfied by the conditional
key in order to be reported.

-p : optional argument. If present, it makes VICKEY interpret the support threshold
as a percentage coverage threshold instead of an absolute support number. The formula
is (support / #subjects)100, that is, support 60 with the -p option enabled means that
a key will be reported if it covers AT LEAST 60% of all subjects in the dataset.

The list of unique conditional keys corresponds to the text output after the indicator
"==== Unique C-keys =====". VICKEY uses the non-keys reported by SAKey,
which are stored in a file reported by VICKEY.
-----
```

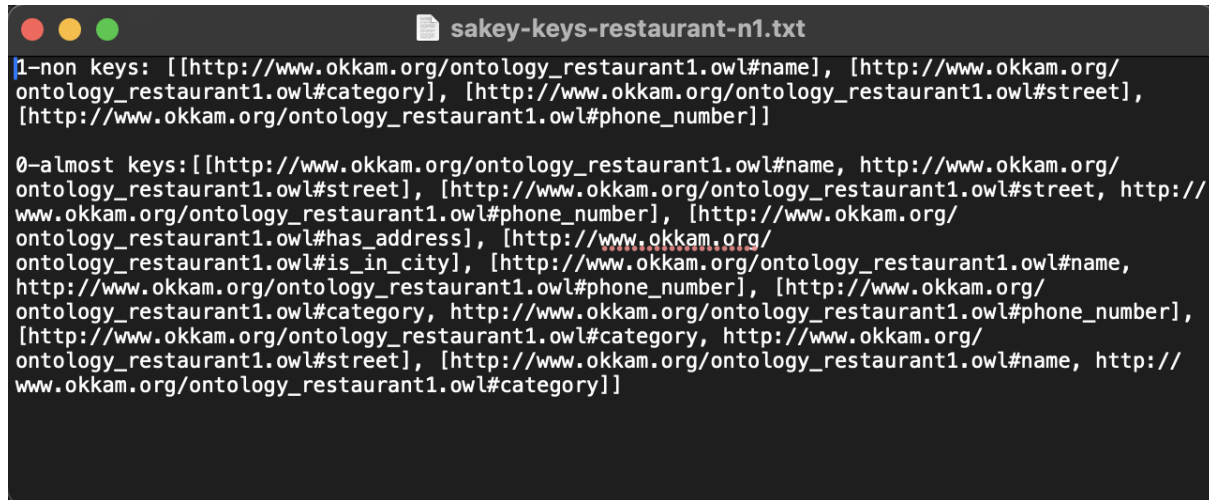
2 Solution

2.1 SAKey on a small dataset

- Run **SAKey** on the dataset "OAEI_2011_Restaurant_1.nt" with $n = 1$, then copy the results that are displayed in the terminal and saved them in a text file **sakey-keys-restaurant-n1.txt**.

Executing the following command produced the file in the figure 1:

```
java -jar ../Sakey-20250118/sakey.jar datasets/OAEI_2011_Restaurant_1.nt 1 >
sakey-keys-restaurant-n1.txt
```



```
sakey-keys-restaurant-n1.txt
1-non keys: [[http://www.okkam.org/ontology_restaurant1.owl#name], [http://www.okkam.org/ontology_restaurant1.owl#category], [http://www.okkam.org/ontology_restaurant1.owl#street], [http://www.okkam.org/ontology_restaurant1.owl#phone_number]]

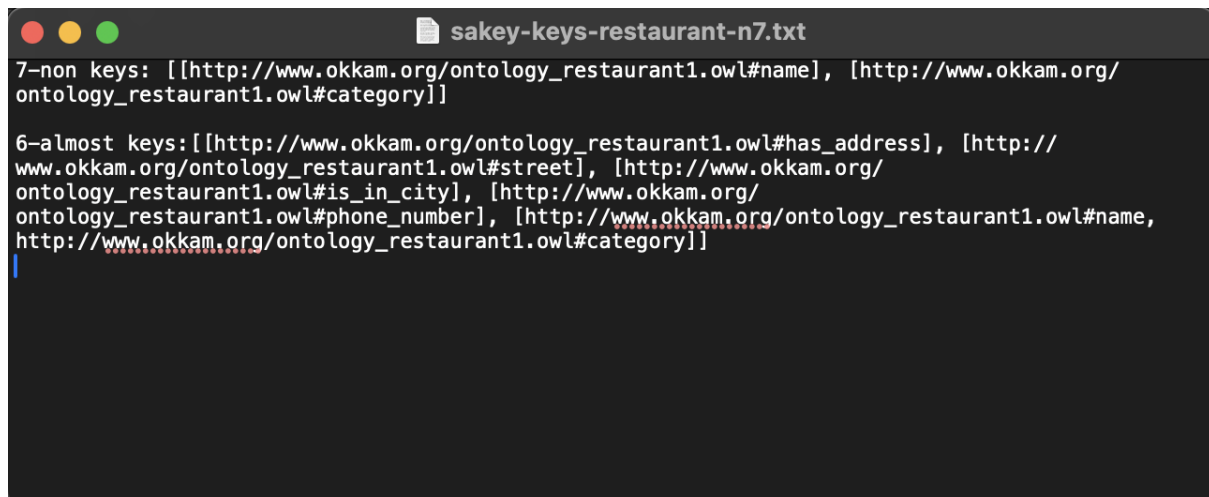
0-almost keys: [[http://www.okkam.org/ontology_restaurant1.owl#name, http://www.okkam.org/ontology_restaurant1.owl#street], [http://www.okkam.org/ontology_restaurant1.owl#street, http://www.okkam.org/ontology_restaurant1.owl#phone_number], [http://www.okkam.org/ontology_restaurant1.owl#has_address], [http://www.okkam.org/ontology_restaurant1.owl#is_in_city], [http://www.okkam.org/ontology_restaurant1.owl#name, http://www.okkam.org/ontology_restaurant1.owl#phone_number], [http://www.okkam.org/ontology_restaurant1.owl#category, http://www.okkam.org/ontology_restaurant1.owl#phone_number], [http://www.okkam.org/ontology_restaurant1.owl#category, http://www.okkam.org/ontology_restaurant1.owl#street], [http://www.okkam.org/ontology_restaurant1.owl#name, http://www.okkam.org/ontology_restaurant1.owl#category]]
```

Figure 1: sakey-keys-restaurant-n1.txt

- Run **SAKey** on the dataset "OAEI_2011_Restaurant_1.nt" with $n = 7$, then copy the results that are displayed in the terminal and saved them in a text file **sakey-keys-restaurant-n7.txt**.

Executing the following command produced the file in the figure 2:

```
java -jar ../Sakey-20250118/sakey.jar datasets/OAEI_2011_Restaurant_1.nt 7 >
sakey-keys-restaurant-n7.txt
```



```
sakey-keys-restaurant-n7.txt
7-non keys: [[http://www.okkam.org/ontology_restaurant1.owl#name], [http://www.okkam.org/ontology_restaurant1.owl#category]]

6-almost keys: [[http://www.okkam.org/ontology_restaurant1.owl#has_address], [http://www.okkam.org/ontology_restaurant1.owl#street], [http://www.okkam.org/ontology_restaurant1.owl#is_in_city], [http://www.okkam.org/ontology_restaurant1.owl#phone_number], [http://www.okkam.org/ontology_restaurant1.owl#name, http://www.okkam.org/ontology_restaurant1.owl#category]]
```

Figure 2: sakey-keys-restaurant-n7.txt

- Compare the obtained keys, in terms of number of keys and minimality (i.e. the number of properties involved in the keys).

Let's first analyze the first file [sakey-keys-restaurant-n1.txt](#). According to [SAKey](#), we have obtained the following 4 single sets of properties which are not keys:

[name] [category] [street] [phone_number]

On the other hand, the minimal sets of properties obtained that are almost keys (0 – *almost* keys) are:

[has_address] [is_in_city] [name, street] [street, phone_number]
[name, phone_number] [category, phone_number] [category, street] [name, category]

The total count for 0 – *almost* keys adds up to 2 single-property keys and 6 two-property key sets.

In the second file [sakey-keys-restaurant-n7.txt](#), we have obtained 2 non-keys:

[name] [category]

On the other hand, the minimal sets of properties obtained that are almost keys (6 – *almost*) are:

[has_address] [is_in_city] [street] [phone_number] [name, category]

The total count for 6 – *almost* keys adds up to 4 single-properties and 1 two-property set. Now, let's compare both results. The number of discovered keys for [sakey-keys-restaurant-n1.txt](#) is 8 keys (0 – *almost* keys) and for [sakey-keys-restaurant-n7.txt](#) is 5 keys (6 – *almost* keys). In terms of minimality (size of the key sets), we have that:

- [sakey-keys-restaurant-n1.txt](#): Although there are more keys in total, only 2 of them are single-property ([has_address], [is_in_city]); the rest are pairs.
- [sakey-keys-restaurant-n7.txt](#): Fewer total keys (only 5), but 4 of them are single-property.

2.2 SAKey on a bigger dataset from DBpedia

1. Run [SAKey](#) on the dataset "library-6k.nt" with $n = 1$, and redirect the results to a text file [sakey-keys-library-6k-n1.txt](#).

Executing the following command produced the file in the figure 3:

```
java -jar ../Sakey-20250118/sakey.jar library-6k.nt 1 > sakey-keys-library-6k-n1.txt
```

The screenshot shows the output of the SAKey tool for the dataset library-6k.nt with n=1. The output is saved in the file sakey-keys-library-6k-n1.txt. The file contains two sections: '1-non keys' and '0-almost keys'. Each section lists a set of URIs representing properties that are either not keys or almost keys. The URIs are formatted as [http://...].

Figure 3: sakey-keys-library-6k-n1.txt

- Run **SAKey** on the dataset "library-196k.nt" with $n = 1$, and redirect the results to a text file **sakey-keys-library-196k-n1.txt**.

Executing the following command produced the file in the figure 4:

```
java -jar ../Sakey-20250118/sakey.jar library-196k.nt 1 > sakey-keys-library-196k-n1.txt
```

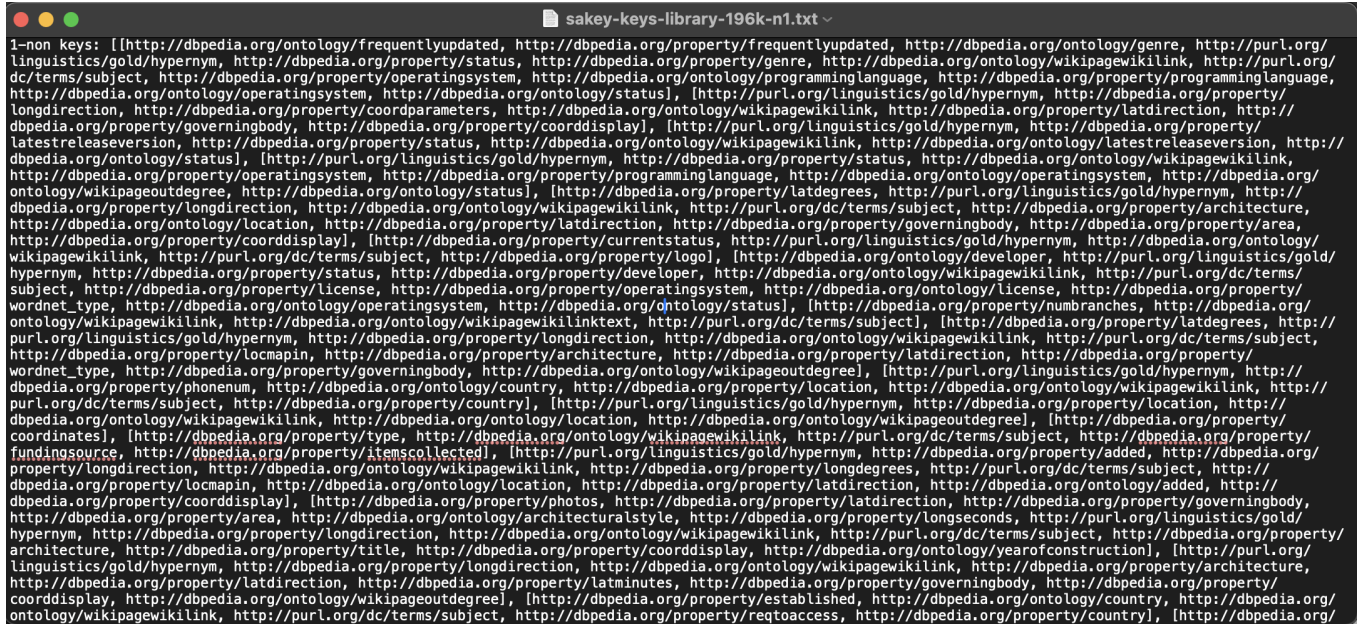


Figure 4: sakey-keys-library-196k-n1.txt

- Compare the results in terms of number of keys and non-keys. If no keys found, try with a higher value of n .

Let's first analyze the first file **sakey-keys-library-6k-n1.txt**. According to **SAKey**, we have obtained the following distribution of **sets of properties which are not keys** which corresponds to a total of **23 non-keys**:

- **1-property:** 3 occurrences
- **2-properties:** 4 occurrences
- **3-properties:** 4 occurrences
- **4-properties:** 1 occurrence
- **5-properties:** 1 occurrence
- **6-properties:** 6 occurrences
- **7-properties:** 3 occurrences
- **9-properties:** 1 occurrence

On the other hand, the minimal sets of properties obtained that are almost keys (0 – *almost* keys) are:

- **1-property:** 17 keys
- **2-properties:** 43 keys
- **3-properties:** 2 keys

The total count for 0 – *almost* keys adds up to **62 keys properties**.

In the second file [sakey-keys-library-196k-n1.txt](#), we have obtained 522 non-keys, following the distribution described:

- **1-property:** 3 occurrences
- **2-properties:** 12 occurrences
- **3-properties:** 24 occurrences
- **4-properties:** 38 occurrence
- **5-properties:** 55 occurrence
- **6-properties:** 55 occurrences
- **7-properties:** 69 occurrences
- **8-properties:** 74 occurrence
- **9-properties:** 39 occurrence
- **10 properties:** 35 occurrences
- **11 properties:** 33 occurrences
- **12 properties:** 20 occurrences
- **13 properties:** 24 occurrences
- **14 properties:** 14 occurrences
- **15 properties:** 11 occurrences
- **16 properties:** 13 occurrences
- **17 properties:** 2 occurrences
- **18 properties:** 1 occurrences

To find the previous distribution, the following Python code was implemented:

```
# Re-read the content and directly process the keys
with open(file_path, 'r') as file:
    data = file.read()

# Split the data to identify 1-non keys and count their occurrences
# First, we remove the part before the first key list
keys_section = data.split("1-non keys:")[1]

# Extract the 1-non keys by capturing all occurrences of the pattern '[]'
matches = re.findall(r'\[.*?\]', keys_section)

# Now, count the distribution of 1-non keys based on the number of properties
distribution = {}

for match in matches:
    # Clean the match and convert it to a Python list using eval
    match_cleaned = match.strip()[1:-1] # Remove leading and trailing brackets
    keys = match_cleaned.split(', ') # Split the keys by the comma and space
    num_properties = len(keys) # Number of properties
    if num_properties in distribution:
        distribution[num_properties] += 1
    else:
        distribution[num_properties] = 1

# Total number of 1-non keys
total_1non_keys = len(matches)

# Display the results
total_1non_keys, distribution
```

On the other hand, the minimal sets of properties obtained that are almost keys (*0-almost*) are distributed as follows:

- **1-properties:** 540 occurrences
- **2-properties:** 15798 occurrences
- **3-properties:** 2709 occurrences
- **4-properties:** 965 occurrences
- **5-properties:** 133 occurrences
- **6-properties:** 2 occurrences

The previous distribution adds up to **20147** which are *0-almost keys*. To find the previous distribution, the following Python code was implemented:

```
from collections import Counter

# Load the file content
file_path = '/mnt/data/zeroalmostkeys196.txt'
with open(file_path, 'r') as file:
    data = file.read()

# Parse the data to find sets of properties (each set is surrounded by square brackets [])
import re

# Extracting sets of properties
property_sets = re.findall(r'\[[^\]]+\]', data)
# Count the number of properties in each set
property_count = [len(props.split(',')) for props in property_sets]

# Calculate the distribution of property counts
property_distribution = Counter(property_count)

# Format the results as LaTeX
latex_distribution = "\n".join([
    f"\\item \\textbf{{{count}}-properties:}} \\$\\{{occurrences}\\}\\$ occurrences"
    for count, occurrences in sorted(property_distribution.items())
])

latex_distribution
```

Now, let's compare both results. The number of discovered keys for **sakey-keys-library-6k-n1.txt** is **62 keys** (*0-almost keys*) and for **sakey-keys-library-196k-n1.txt** is **20147 keys** (*0-almost keys*). In terms of minimality (size of the key sets), we have that:

- **sakey-keys-library-6k-n1.txt:** Most keys consist of 1 or 2 properties, with very few requiring 3 properties. This suggests high minimality.
- **sakey-keys-library-196k-n1.txt:** While the majority of keys have 2 or 3 properties, the presence of keys requiring up to 6 properties indicates a broader distribution and less minimality compared to the 6k dataset.

2.3 Use the keys for data linking

The following **SPARQL** queries have been considered to study the retrieved information from **SAKey**:

- **Query 1**

To check if **dbp:etymology** is a valid *0-almost key* (meaning it can uniquely identify library entities with no exceptions), we can create a **SPARQL** query that retrieves pairs of libraries that share the same **etymology**

value but are distinct entities. If no such pairs exist, it would suggest that **dbp:etymology** is functioning as a valid key. The considered **SPARQL** query is defined as follows:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT DISTINCT ?library1 ?library2
WHERE {
    ?library1 rdf:type dbo:Library .
    ?library1 dbp:etymology ?etymology1 .

    ?library2 rdf:type dbo:Library .
    ?library2 dbp:etymology ?etymology2 .

    # Ensure the two libraries are distinct
    FILTER (?library1 != ?library2)

    # Ensure they share the same etymology value
    FILTER (?etymology1 = ?etymology2)
}
LIMIT 10
```

The results from the query are in the following figure 5:



Figure 5: Query 1 - Results

• Query 2

For this next **SPARQL** query, we considered the 0 – *almost* keys **location** and **established**, which appear on the results from **SAKey**.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT DISTINCT ?library1 ?library2
WHERE {
    ?library1 rdf:type dbo:Library .
    ?library1 dbp:established ?established1 .
    OPTIONAL { ?library1 dbp:location ?location1 . }
```

```

?library2 rdf:type dbo:Library .
?library2 dbp:established ?established2 .
OPTIONAL { ?library2 dbp:location ?location2 . }

FILTER(
  (?established1 != ?established2) ||
  (BOUND(?location1) && BOUND(?location2) && ?location1 != ?location2)
)
}
LIMIT 10

```

However, the results of the **SPARQL** query in figure 6 indicate that the combination of **location** and **established** is not strictly unique for identifying libraries, as multiple libraries are associated with the same library. This doesn't align with the concept of a 0 – *almost* key in **SAKey**.

Query: <http://dbpedia.org/sparql>

```

1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4 PREFIX dbo: <http://dbpedia.org/ontology/>
5 PREFIX dbp: <http://dbpedia.org/property/>
6
7 SELECT DISTINCT ?library1 ?library2
8 WHERE {
9   ?library1 rdf:type dbo:Library .
10  ?library1 dbp:established ?established1 .
11  OPTIONAL { ?library1 dbp:location ?location1 . }
12
13  ?library2 rdf:type dbo:Library .
14  ?library2 dbp:established ?established2 .
15  OPTIONAL { ?library2 dbp:location ?location2 . }
16
17  FILTER(
18    (?established1 != ?established2) ||
19    (BOUND(?location1) && BOUND(?location2) && ?location1 != ?location2)
20  )
21 }
22 LIMIT 10
23

```

Showing 1 to 10 of 10 entries (in 0.045 seconds)

library1	library2
http://dbpedia.org/resource/Cabarnus_County_Public_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Cat_Poly_Pomona_University_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Catala_Free_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Calgary_Tool_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/California_Area_Public_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Cambridge_University_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Camden_Public_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Cantonment_Public_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Capitol_View_Neighborhood_Library	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives
http://dbpedia.org/resource/Cat_Albert_Center	http://dbpedia.org/resource/California_Ethnic_and_Multicultural_Archives

Showing 1 to 10 of 10 entries (in 0.045 seconds)

Figure 6: Query 2 - Results

In particular, the results highlight potential issues in the dataset. Specifically, the repetition of **California Ethnic and Multicultural Archives** suggests either incomplete or inconsistent data for the **location** or **established** properties. This could be due to shared or overlapping values for these attributes, or errors in the data.

Further investigation into the dataset is necessary to understand and possibly address these anomalies, ensuring that the combination of properties achieves a higher degree of uniqueness.

• Query 3

Finally, we consider a third **SPARQL** query, which retrieves pairs of distinct library entities (of type **dbo:Library**) that share the same **location** and **country** values, and once again, in theory such pair of properties is an 0 – *almost* key as shown in figure 7. The **SPARQL** query is the following:

```

PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?dbpediaID1 ?dbpediaID2
WHERE {
  # First entity

```

```

?dbpediaID1 rdf:type dbo:Library .
?dbpediaID1 <http://dbpedia.org/property/location> ?location1 .
?dbpediaID1 <http://dbpedia.org/property/country> ?country1 .

# Second entity
?dbpediaID2 rdf:type dbo:Library .
?dbpediaID2 <http://dbpedia.org/property/location> ?location2 .
?dbpediaID2 <http://dbpedia.org/property/country> ?country2 .

# Ensure the keys are not the same for two different entities
FILTER (?dbpediaID1 != ?dbpediaID2)
FILTER (?location1 = ?location2 && ?country1 = ?country2)

}
LIMIT 10

```

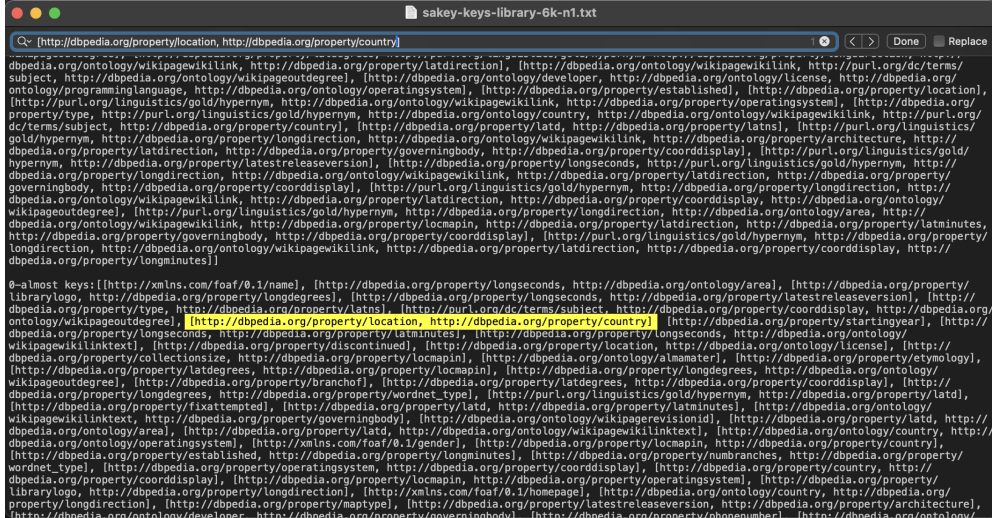
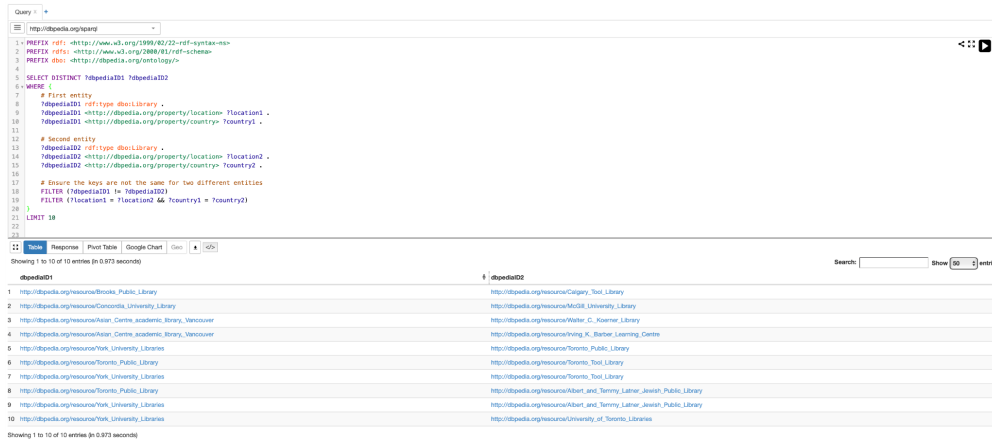


Figure 7: Keys Location & Country

The results obtained are the following:



dbpediaID1	dbpediaID2
http://dbpedia.org/resource/Brooks_Public_Library	http://dbpedia.org/resource/Calgary_Public_Library
http://dbpedia.org/resource/Concordia_University_Library	http://dbpedia.org/resource/McGill_University_Library
http://dbpedia.org/resource/Asian_Centre_academic_library_Vancouver	http://dbpedia.org/resource/Walter_C._Rosenberg_Library
http://dbpedia.org/resource/Asian_Centre_academic_library_Vancouver	http://dbpedia.org/resource/Wing_K._Barber_Learning_Centre
http://dbpedia.org/resource/York_University_Libraries	http://dbpedia.org/resource/Toronto_Public_Library
http://dbpedia.org/resource/Toronto_Public_Library	http://dbpedia.org/resource/Toronto_Public_Library
http://dbpedia.org/resource/York_University_Libraries	http://dbpedia.org/resource/Toronto_Public_Library
http://dbpedia.org/resource/Toronto_Public_Library	http://dbpedia.org/resource/Lester_B. and Terry Lerner Jewish Public Library
http://dbpedia.org/resource/York_University_Libraries	http://dbpedia.org/resource/Jacob and Tanya Lerner Jewish Public Library
http://dbpedia.org/resource/York_University_Libraries	http://dbpedia.org/resource/University_of_Toronto_Libraries

Figure 8: Query 3 - Results

The query results reveal cases where libraries, such as **"Brooks Public Library"** and **"Calgary Tool Library"** or **"Concordia University Library"** and **"McGill University Library"**, share the same **location** and **country**, indicating that this combination is not always unique. According to **SAKey**, **location** and **country** were identified as an 0 – *almost* key in the **library-6k.nt** dataset, allowing for a small number of exceptions. These results confirm that while **location** and **country** are useful for distinguishing most libraries, they are not sufficiently distinctive to uniquely identify every library entity. This aligns with the 0 – *almost* key definition, where some violations are permitted.

2.4 Going Further: Vickey & Rocker

2.4.1 Vickey

2.4.1.1 OAEI_2011_Restaurant_1.nt

Executing the following **command** produced the following results:

```
java -jar Vickey-20250118/vickey.jar -mins 5 -p datasets-20250118/datasets/OAEI_2011_Restaurant_1.nt > vickey-restaurant_1.nt.txt
```

Results:

- Computing the non-keys: 4 non-keys found.
- Facts loaded: 1130.
- Minimum support: 5.0%.
- Instances found: 339.
- Unique Conditional Keys: 0 found in 1 ms.

2.4.1.2 Library-6k.nt

Executing the following **command** produced the following results:

```
java -jar Vickey-20250118/vickey.jar -mins 5 -p datasets-20250118/library-6k.nt > vickey-keys-library-6k-n1.txt
```

Results:

- Computing the non-keys: 23 non-keys found.
- Facts loaded: 6098.
- Minimum support: 5.0%.
- Instances found: 96.
- Unique Conditional Keys: 0 found in 2 ms.

2.4.1.3 Library-196k.nt

Executing the following **command** produced the following results:

```
java -jar Vickey-20250118/vickey.jar -mins 5 -p datasets-20250118/library-196k.nt > vickey-keys-library-196k.nt.txt
```

Results:

- Computing the non-keys: 522 non-keys found.
- Facts loaded: 196,501.
- Minimum support: 5.0%.
- Instances found: 2687.
- Unique Conditional Keys: 0 found in 5 ms.

2.4.2 Rocker

2.4.2.1 OAEI_2011_Restaurant_1.nt

Executing the following **command** produced the following results:

```
java -Xmx8g -jar Vickey-20250118/rocker-1.2.1-full.jar "OAEI_2011_Restaurant_1"
"file:///Users/pablomollacharlez/Downloads/Key_Discovery/datasets-
20250118/datasets/OAEI_2011_Restaurant_1.nt"
"http://www.okkam.org/ontology_restaurant1.owl#Restaurant" false true 10.0 >
rocker-keys-OAEI_2011_Restaurant_1.txt
```

Results:

- Instances: 113.
- Properties: 4 properties analyzed.
- Parsing Errors: 0.
- Exact Keys:
 - http://www.okkam.org/ontology_restaurant1.owl#has_address (Score: 1.0).
 - http://www.okkam.org/ontology_restaurant1.owl#name (Score: 1.0).
- Composite Key:
 - [http://www.okkam.org/ontology_restaurant1.owl#category,
http://www.okkam.org/ontology_restaurant1.owl#phone_number] (Score: 1.0).
- High-Scoring 0-Almost Key:
 - http://www.okkam.org/ontology_restaurant1.owl#phone_number: 0.991.
- Low-Scoring Property:
 - http://www.okkam.org/ontology_restaurant1.owl#category: 0.168.

2.4.2.2 Library-6k.nt

Executing the following **command** produced the following results:

```
java -Xmx8g -jar Vickey-20250118/rocker-1.2.1-full.jar "library-6k.nt"
"file:///Users/pablomollacharlez/Downloads/Key_Discovery/datasets-20250118/library-6k.nt"
"http://dbpedia.org/ontology/Library" false true 10.0 > rocker-keys-library-6k.txt
```

Results:

- Instances: 95.
- Properties: 221 unique properties analyzed.
- Parsing Errors: 1 (unclosed triple).
- Exact Keys:
 - <http://www.w3.org/ns/prov#wasDerivedFrom>.
 - <http://dbpedia.org/ontology/wikiPageRevisionID>.
 - <http://dbpedia.org/ontology/wikiPageLength>.

- <http://www.w3.org/2000/01/rdf-schema#label>.
- High-Scoring 0-Almost Keys:
 - <http://xmlns.com/foaf/0.1/homepage>: 0.505.
 - <http://dbpedia.org/property/location>: 0.452.
 - <http://dbpedia.org/ontology/wikiPageOutDegree>: 0.421.

2.4.2.3 Library-196k.nt

Executing the following **command** produced the following results:

```
java -Xmx8g -jar Vickey-20250118/rocker-1.2.1-full.jar "library-196k.nt"
"file:///Users/pablomollacharlez/Downloads/Key_Discovery/datasets-20250118/library-196k.nt"
"http://dbpedia.org/ontology/Library" false true 10.0 > rocker-keys-library-196k.txt
```

Results:

- Instances: 2687.
- Properties: 744 unique properties analyzed.
- Parsing Errors: 0.
- High-Scoring 0-Almost Keys:
 - <http://dbpedia.org/property/website>: 0.362.
 - <http://dbpedia.org/property/type>: 0.073.
 - <http://dbpedia.org/property/itemsCollected>: 0.064.
- Many properties had low scores (e.g., 0.001 or lower), reflecting weak potential as 0-almost keys.