Discovering Key Candidates

Milestone Report

Andrea Wang
Center for Data Science, NYU
ayw255@nyu.edu

Preet Gandhi
Center for Data Science, NYU
pg1690@nyu.edu

1 INTRODUCTION AND PROBLEM FORMULATION

Given a collection of datasets, if we want to merge or join between datasets, the process can be broken down into 2 steps. First of all, we need to find the unique identifiers of each datasets which can differentiate the tuple entries. We need an identifying column which can uniquely and logically identify a tuple. Secondly, we need to reason what are the appropriate keys that link data instances across related datasets. We need the foreign keys because for normalized tables, we need to have a unique and consistent identifier for matching the corresponding tuples which logically point to the same data entity. To illustrate, let's consider a toy example:
Consider we have the two tables listed below and we want to join them to get the height, weight, and age information of each person.

Height and Weight Table				
id	first name	last name	height (cm)	weight (kg)
1	Sam	Seaborn	180	75
2	Leo	McGarry	173	70
3	Josh	Lyman	175	68
4	C.J.	Cregg	168	55
5	Sam	Smith	185	77

Height and Age Table				
id	first name	last name	height (cm)	age
1	Leo	McGarry	173	62
2	C.J.	Cregg	168	37
3	Sam	Seaborn	180	35
4	Sam	Smith	185	27
5	Josh	Lyman	175	40

An intuitive first step is to find the columns or collections of columns that uniquely identify each row of the table. We call these columns 'candidate keys'. Each table may have one or more candidate keys, but one candidate key is unique, and it is called the primary key.The candidate keys for 'Height and Weight Table' is: *{'id'}, {'first name', 'last name'}, {'height'}, {'weight'}*. and the candidate keys for 'Height and Age Table' is *{'id'}, {'first name', 'last name'}, {'height'}, {'age'}*. Not every candidate keys can be used, or reasonably be used, to be the keys that link the data instances in the tables.Fore example, weight being a continuous quantity can have non-repeating values and hence unique. But it isnt logical to use weight as an identifying column nor can we use it to link to another data table. Hence, the next step is to determine which candidate keys we should use to merge the two tables. First of all, we can see that *{'id'}* is just row numbers, it does not carry any information regarding the data instances in both tables. Secondly, although *{'height'}, {'weight'},{'age'}* are unique across all rows, one can hardly argue that they are 'identifiers' because their uniqueness is based on the fact that they are continuous values and no two weights can be exactly same in terms of actual decimal values in thousands. The natural keys for these two tables are *{'first name', 'last name'}*.
In this project, we will try to tackle the following challenges illustrated in the toy example: 1. Identifying candidate keys. and subsequently, 2. Find the keys, among candidate keys, that reasonably identify each data instance.

2 RELATED WORKS AND REFERENCES

Many researchers have focused on the problem of automated meta data discovery, especially in the context of query optimization. The two main approaches can be characterized as either query driven or data driven. The query driven, or feedback, approach extracts information from the answers to user queries. An advantage of this approach is that it directs system resources toward the users needs and interests. Query-driven techniques scale well, and yield immediate gains by focusing on real production queries. These techniques, however, require a burnin period of initial learning. Moreover, these techniques may not be robust when faced with previously unseen queries or significant changes to the underlying data; in the latter case, the feedback from queries executed at different times can be mutually inconsistent. A variant type of query-driven technique uses information about a query workload, rather than the actual results of executing the queries . Data-driven techniques look directly at the base data, without reference to a query workload. These techniques form an important complement to query-driven methods: while perhaps less accurate, data-driven techniques tend to be more robust. Indeed, the two techniques can be fruitfully combined.Well known data-driven techniques include methods for producing summary or synopsis data structures such as histograms, wavelets and graphical statistical models . These techniques typically do not scale well to high-dimensional data (the so-called curse of dimensionality), and the user usually has to select which (few) dimensions to include in the summary.
In the paper, GORDIAN: Efficient and Scalable Discovery of Composite Keys[2], the problem of discovering (composite) keys can be formulated in terms of the cube operator which encapsulates all possible projections of a dataset while computing aggregate functions on the projected entities. It asserts that a projection corresponds to a key if and only if all the count aggregates for a projection are equal to 1. For example, [EmpNo] and [First Name, Phone] are keys, while [First Name, Last Name] is a non-key.

FirstName	*LastName*	*Phone*	*EmpNo*	*COUNT*
Michael	Thompson	3478	10	1
Sally	Kwan	3478	20	1
Michael	Spencer	5237	90	1
Michael	Thompson	6791	50	1

FirstName	*LastName*	*COUNT*
Michael	Thompson	2
Sally	Kwan	1
Michael	Spencer	1

FirstName	*COUNT*
Michael	3
Sally	1

Phone	*COUNT*
3478	2
5237	1
6791	1

FirstName	*Phone*	*COUNT*
Michael	3478	1
Sally	3478	1
Michael	5237	1
Michael	6791	1

LastName	*COUNT*
Thompson	2
Kwan	1
Spencer	1

EmpNo	*COUNT*
10	1
20	1
90	1
50	1

Figure 3: A subset of the cube operator for the Dataset in Fig. 1

So overall the GORDIAN allows the discovery of composite keys while avoiding the exponential processing and memory requirements that have limited the applicability of previous brute- force methods to very small data sets.

3 METHODS, ARCHITECTURE AND DESIGN

3.1 Identify Candidate Keys

1. First we aim to find columns which have all unique values (i.e. non-repeating) and no null as this is what qualifies as a candidate key. For single column, we make use of spark and pandas DataFrame to count the unique values in each column and compare it to number of total rows. Considering that the dataset may not be very clean, we examine the 'uniqueness' of a column,

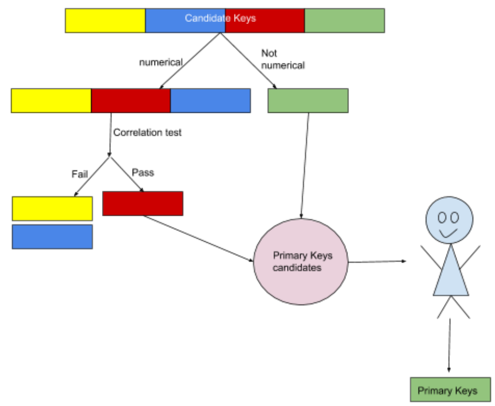
$$\text{uniqueness} = \frac{\text{number of unique values in the column}}{\text{number of rows}}\%$$

, and list the column that has has very high uniqueness, if not 100%, as candidate keys. This is to tolerate data processing errors (say, null values or data entry errors).Ideally the candidate keys should be unique and not null but we have taken percentage because big data can be dirty and due to a small percentage of error we don't want a very hard limit. We will set up a threshold to determine what level of uniqueness we consider to be appropriate when tagging it to be the composite keys.

2. Discovering composite keys is harder. The basic ideas is to examine the combinations of all the columns that are not unique by itself. We can implement the algorithms given by [1] or [2]. The crude idea is that if we have individual columns that we know are unique, we can create combinations of them that can make logical sense when we use it to link to another table. A composite key is preferable as if can make more logical sense and prevent overlap with other tables while joining.

3.2 Identify Primary Keys

As illustrated in the toy example presented in the Introduction section, identify primary keys solely by considering the uniqueness of columns can be tricky. However, by closer inspection, we believe there are several tricks to further narrow down the set of possible primary keys. First of all, candidate keys that are numerical have high possibility of being continuous and thus unfit for primary keys. Therefore, we should first separate out numerical candidate keys. All the keys that are not numerical can be considered candidates for primary keys. For the numerical keys, we can look at the correlations of those keys with other numerical columns. If the keys have high correlations with one column, then we should consider it not a possible primary key. But those with really low correlation can be a unique number id which can be a candidate key.



4 PRELIMINARY RESULTS

While testing our code and methodology on yu9n-iqyk.json, which is New York City Results on the New York State English Language Arts (ELA) Tests. We found that dirtykeys.txt, which are columns that are unique after disregarding null values, holds *sid, id* and *position*. We have a howunique.txt which has the column name and uniqueness percentage. Of course the dirty keys *sid, id* and *position* have the highest ratio. The original json file can be found at our repository.
Moreover, after taking a closer look at our numerical attributes, we found that the numerical columns like createdat and updatedat have vastly varying mean, median, mode and the percentiles that don't make any logical sense. Same goes for position. But as we consider sid as a dirty column, we can see in our analysis that it can potentially be a key if we set a uniqueness threshold.

```
>>> df.describe()
                  sid          position    created_at    updated_at
count  5966887.000000  5966887.000000  5.966887e+06  5.966887e+06
mean   2983444.070385  2983444.070385  1.487089e+09  1.487089e+09
std    1722492.165505  1722492.165505  5.186605e+06  5.186605e+06
min          1.000000        1.000000  1.485889e+09  1.485889e+09
25%    1491722.500000  1491722.500000  1.485889e+09  1.485889e+09
50%    2983444.000000  2983444.000000  1.485889e+09  1.485889e+09
75%    4475165.500000  4475165.500000  1.485889e+09  1.485889e+09
max    5966888.000000  5966888.000000  1.520542e+09  1.520542e+09
```

When we try to find the correlation between the numerical attributes, we can see that the results turn out to be like we had assumed in our methodology. For example createdat and updatedat are completely correlated meaning they cant possible identify tuples uniquely as some dependencies exist between the columns. Same goes for sid and position. Position cant definitely identify a unique column and is probably related to sid as sid is dirty.

```
>>> df.corr()
                 sid  position  created_at  updated_at
sid         1.000000  1.000000    0.382703    0.382703
position    1.000000  1.000000    0.382703    0.382703
created_at  0.382703  0.382703    1.000000    1.000000
updated_at  0.382703  0.382703    1.000000    1.000000
```

A snapshot of the discussed columns can be found here:

sid	id	position	created_at
1	545A09A6-7D75-4664-8046-8BC3EB8B0333	1	1317822619
2	8ABC04DD-23D0-4D10-8E32-C798F79DA50B	2	1317822619
3	F860C6DE-2277-4A42-9F36-6FAE588BCAD0	3	1317822619
4	D217B95D-88F0-447A-866B-E771371A2891	4	1317822619
5	9131D906-7558-4E7E-8687-B6BB1658F871	5	1317822619

created_meta	updated_at	updated_meta	meta	DBN	Grade
396547	1317822619	396547	{\n}	01M015	3
396547	1317822619	396547	{\n}	01M015	4
396547	1317822619	396547	{\n}	01M015	5
396547	1317822619	396547	{\n}	01M015	6
396547	1317822619	396547	{\n}	01M015	All Grades

.

The ideal primary key for this dataset is *'id'*.

5 CODE REPOSITORY

The code and result files from our preliminary results as well as basic data exploration can be found at our repository :
https://github.com/preetgandhi95/BigData18
firstcode.py goes through the methodology proposed in 3.1. It reads data and keeps track of columns which are both unique and not null. It also looks for columns which are unique but disregarding unique values. Last part deals with finding the percentage of unique values in a column if we are interested in setting up a threshold. The outputs are stored in 3 files corresponding to each of the three tasks. It can be found at the link shared.

REFERENCES

[1] Ziawasch Abedjan and Felix Naumann. 2011. Advancing the discovery of unique column combinations. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 1565–1570.

[2] Yannis Sismanis, Paul Brown, Peter J Haas, and Berthold Reinwald. 2006. GORDIAN: efficient and scalable discovery of composite keys. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 691–702.