Linking Stack Overflow and Github Public Data for Mining Purposes

Norbert Eke
Carleton University
Ottawa, ON
NorbertEke@cmail.carleton.ca

ABSTRACT

Developer expertise learning and recommendation is the task of defining and quantifying the expertise areas and levels of developers, then creating a top-*n* ranking for developers who are most qualified to perform a task. A software repository mining approach on this task would allow the creation of a developer expertise profile consisting of topical expertise and interest distributions learned from Stack Overflow and Github public data. This project addresses building a database consisting of Stack Overflow and Github public data, then linking them together based on a common attribute.

KEYWORDS

Data Mining, Mining Software Repositories, Big Data Management, Stack Overflow, GitHub, Data Mapping, Data Set Linkage, Data Exchange, Data Management, Database Design

ACM Reference Format:
Norbert Eke. . Linking Stack Overflow and Github Public Data for Mining Purposes. In *Proceedings of* . ACM, New York, NY, USA, 11 pages.

1 INTRODUCTION

The problem of developer expertise learning and recommendation is a well defined problem in software engineering. Project managers are often faced with the task of deciding who is the best developer to handle a certain bug fix, code review or pull request. A task at hand needs to be assigned to one or more developers within the company. The optimal task assignment would allow the developer(s) with the most amount of expertise and experience in the domain to complete the task at hand. To achieve such a task assignment, defining and predicting the expertise of a developer is needed.

This problem is worth looking at, as it is important for any software company to be able to define and quantify certain expertise levels for their developers. It is interesting to look at this problem from a data mining perspective. Software developer expertise can be learned from analyzing every interaction of developers with their software environments. Software repository mining data sources like question answering websites and source code repositories can be considered a rich public data gold mine for learning developer expertise. Stack Overflow is the most popular question answering website for software related questions. Github is the most popular code repository website hosting source code for millions of software projects, alongside their commits, pull requests and issues. These two data sets are very good data sources for a software repository mining project, assuming they can be linked together.

Developer expertise learning and recommendation is a difficult task for multiple reasons. Firstly, one needs to define what makes someone expert in a field or concept or programming paradigm. Secondly, there are too many different topics, concepts, programming languages and development areas/activities in computer science to classify expertise in. It is difficult even for a human to quantify and compare multiple developers across all possible ways of classifying expertise. Thirdly, considering the first and second point, it is that much harder to make an algorithm learn enough from data to perform a recommendation or ranking task, ranking developers based on a learned expertise score.

The Naive approach of counting up votes on each user's answers on Stack Overflow and associating them to tags on the questions would not work. Most experts in the community of software repository mining would agree that tags on Stack Overflow are too general to define expertise levels. For instance, the most frequently used tag on Stack Overflow is JS (JavaScript), but saying that some is in expert in JavaScript does not reveal what specific activities/tasks can they perform in that language. Disproving the Naive approach does not answer the question why this problem has not been solved before. There has been a lot of research done in this area of software engineering. For example, Matter et al. [7] built a vocabulary based expertise model for assigning bug reports to developers for bug fixing tasks. Wu et al. [10]'s approach performs developer recommendation with k-nearest-neighbor search using bug similarity and expertise ranking with various metrics. This problem has been previously explored, but a developer activity profiling approach via mining Stack Overflow and Github public data was not considered. A data mining approach would allow the creation of a developer expertise profile consisting of topical expertise and interest distributions learned from Stack Overflow and Github public data. One could argue that a topical distribution of expertise could be more convenient, more precise and possibly even more specific than a vocabulary based expertise model, for example.

This paper only addresses the first part of this project, which is building a database consisting of Stack Overflow and Github public data, then linking them together based on a common (linkable) attribute. Limitations of this project are the lack of large computational resources and data privacy concerns that do not allow access to Github and Stack Overflow user's email address (or email hash) data. Throughout the project only a shared server was available with limited disk space. In the future, a new server with more disk space available will be considered, since big data analysis projects need more computational resources.

1.1 Summary of Contributions

The rest of the paper moving forward will contain related works in section 2, then the whole process of the database creation in section 3, including data gathering (3.1), database design (3.2), database linkage (3.3) and data reduction/filtering (3.4). Results will be found in section 4, then conclusion and future work will follow in sections 5 and 6.

The contribution of this project is two-fold:

- Linking public data from Stack Overflow and Github together in one data set
- Reducing the size of the joined data set significantly by keeping only linked data

2 RELATED WORK

In one of its latest public data dumps from December 2018 Stack Overflow listed over 42 million posts from almost 10 million registered users. To analyze how Stack Overflow posts evolve Baltes et al. [4] built SOTorrent [3], an open data set aggregating and connecting the official Stack Overflow data dump to other web sources, such as Github repositories. SOTorrent provides access to the version history of Stack Overflow content at two different levels: whole posts and individual text or code blocks. GHTorrent [6] is similar to SOTorrent, as the creators wanted to create an open data set mirroring the data present on Github.

In 2013, Vasilescu et al. [9] linked GitHub and Stack Overflow users based on a computed MD5 hash of the email attribute found in GHTorrent and SOTorrent's User tables. This linkage was possible in the past, but after 2016 both GHTorrent and SOTorrent were required to remove any personal data stored about users in order to follow GDPR compliances. Thus, since 2016 the linkage between the two open data sets has been questionable. In order to create an expertise profile consisting of topical expertise and interest distribution, it would be favorable to merge both data sources.

Topic models are a type of statistical model used in text mining to discover hidden semantic structures in textual data. Most of such models discover patterns of distributions of topics within the textual data. Tian et al. [8] used topic models in their work on predicting the best answerer for a new question on Stack Overflow. Their approach learns user topical expertise and interest levels by profiling each user's previous activity and reputation on Stack Overflow. Tian et al. [8] claim that the "semantic similarity between the user profile and the new question can be captured through topic models". For each potential answerer each user's expertise level can be learned through previous user activity data and up votes from the user profile. Arwan et al. [1] proposed a mechanism for source code retrieval on Stack Overflow by "inferencing concept location from source code" using a Latent Dirichlet Allocation (LDA) model, which is one of the most popular topic models. Topic modeling have been previously used to learn semantic similarity between user profiles, posts and even source codes on Stack Overflow. What my research is interested in creating is a developer expertise profile/ranking consisting of topical expertise and interest distributions learned from mining public software repository data like Stack Overflow and Github.

3 DATABASE CREATION

3.1 Data Gathering

All data used in this project is publicly available. The Stack Overflow data comes from Baltes et al. [3], who created SOTorrent for the purposes of mining and analyzing the evolution of Stack Overflow posts. SOTorrent releases a new version of processed Stack Overflow data dumps every 3 months. Each version of the SOTorrent data can be downloaded from their Zenodo download page [2]. SOTorrent data can also be looked at and queried online, as it is available as a BigQuery data set [1]. In this project the data dump from December 9th 2018 was used, as it was the latest version available at the start of the project. Not all data was downloaded, as certain tables containing data on Stack Overflow post evolution (e.g. PostHistory, PostVersion tables) are unrelated to future analysis, thus they are not needed. Compressed XML files of each table's raw data were downloaded from Zenodo's download page [2]. The tables downloaded included Users, Posts, PostLinks, PostType, PostReferenceGH, Comments, CommentUrl, GHMatches, Tags, Badges and Votes tables totaling 29 GB of compressed raw data. After the downloading process has ended, all ZIP files were unzipped, resulting in one XML file per table, totaling around 100 GB of data.

The Github data comes from Gousios' work named GHTorrent [6], a project that has been collecting data from all public projects available on Github. GHTorrent releases a new version of MySQL data dumps every month, while it also offers daily data dumps for MongoDB. Each version of GHTorrent MySQL data [5] can be downloaded from GHTorrent website's download page [3]. GHTorrent data can also be looked at and queried online, as it is available as a DBLite web data set [4]. In this project the data dump from March 1st 2019 was used, as it was the latest version available at the start of the project. GHTorrent's data dump comes in one large ZIP file, which needed to be uncompressed. The zipped version of the file is 96557 MB, and it took 4 days to download. Unzipping the main directory resulted in one CSV file for each table, for a total of 21 tables measuring over 400 GB of data.

[1] https://bigquery.cloud.google.com/dataset/sotorrent-org:2018_12_09
[2] https://zenodo.org/record/2273117#.XKVGXZhKhPY
[3]http://ghtorrent.org/downloads.html
[4]http://ghtorrent.org/dblite/

3.2 Database Design

Loading the CSV and XML files into the MySQL database was a very time consuming process. Some of the raw data files were over 100 GB. Loading such large files into tables could take days, as a few hundred million, or even billions of rows have too many foreign key constraints that needed to be checked by the MySQL engine in order to allow the storage of the data in the table. All SQL scripts performing table creations, data imports and general database manipulations can be found in SOTorrent's Github repository [5] and GHTorrent's Github repository [6].

Importing all data from both data sets took 2 weeks as downloading, unzipping and importing all data was not only time consuming, but also challenging in terms of disk storage. For this project a shared server was used with technical specifications of 32 GB RAM, 2 TB disk space and 30 CPU cores. Since this server is shared by 5 people, only about 400 GB disk space was available. The most challenging part was managing disk space during data imports, as the Github and Stack Overflow raw data files totaled up to 500 GB. The importing of the data was not achieved all at once, but rather through several iterations. Only a limited amount of raw data files were kept on server, and the rest of the files were compressed back to save disk space. After a table was successfully imported, its raw CSV or XML file was deleted to free up disk space.

Throughout the data import phase it was discovered that using *MyISAM* over *InnoDB* as database engine is more favorable, since there is a significant execution time difference between the two database engines when importing large amounts of data into MySQL. *MyISAM* database engine does not support foreign keys constraints, while it supports table-level locking. On the other hand *InnoDB* supports foreign key constraints and row-level locking. *MyISAM* is preferred when performing tasks that require fast import and querying speed, while *InnoDB* being the default database engine for MySQL, is more optimal for regular operations. For this particular reason all of the large raw data files were imported into the database using the *MyISAM* database engine. After all necessary database manipulations have been performed on these tables, the database engine was changed for each table back to the default engine, *InnoDB*, to allow foreign key constraints to be enforced.

Table 1 contains descriptions for each table's content. Figure 1 and 2 show the database schema and all attributes within each table of the SOTorrent and respectively the GHTorrent data set. Some descriptions come directly from Gousio's paper [6] describing the content of tables in his database. All tables from GHTorrent are shown in figure 2, while only the relevant tables from SOTorrent are shown on the diagram in figure 1.

3.3 Database Linkage

The most important task in this project is linking together the SOTorrent and GHTorrent databases. After a thorough investigation of both databases' main tables two candidate attributes within each database were identified to be potentially good attributes to link the two data sets on.

The first candidate attribute pair is *email hash* from SOTorrent's User table and *email* in one of GHTorrent's earlier versions (before March 2016) of the User table. This candidate attribute pair was used by Vasilescu et al. [9] in 2013 to "merge (i.e., link) a GitHub and a Stack Overflow user if the computed MD5 hash [of the email attribute in GHTorrent's User table] is identical to the MD5 email hash [stored in SOTorrent's User table]". This linkage was possible in 2013, but after 2016 neither GHTorrent, nor SOTorrent were allowed to store email addresses or email hashes in their databases, as it violated GDPR compliances. This attribute pair was the more favorable option, but unfortunately SOTorrent took out all email hash instances

[5] https://github.com/sotorrent/db-scripts/tree/master/sotorrent
[6] https://github.com/gousiosg/github-mirror/tree/master/sql

Table Name	Description	Table Name	Description
project	Github project repositories	pull request commits	Linkage between pull request and commits tables on Github.
GH_users	Github users data and metadata	repo_labels	Keyword labels attached to a project's repository on Github.
project members	Users with commit access to the referenced project	project languages	List of programming languages used in a project on Github.
organization members	List of members in an organization	project topics	Topical classification of a project on Github.
commits	A list of all commits on Github. The project_id field refers to the first project this commit has been added to	issue labels	Labels attached to each issue created for a Github project
project commits	List of all commits to a project.	Comments	Data related to comments on Stack Overflow posts
commit parents	Commits that are parents to a commit	CommentURL	Hyperlinks appearing on Stack Overflow posts
commit comments	Code review comments for a commit	Badges	Badge awards received by Stack Overflow users
watchers	users that have starred a project	SO_users	All metadata on Stack Overflow users
followers	users that are following another user	PostType	Type of post on Stack Overflow
issues	Issues that have been recorded for a project	PostLinks	Shows different linkages between related posts on Stack Overflow
issue events	Chronologically ordered list of events on an issue	Posts	All data and metadata related to a post on Stack Overflow
issue comments	Discussion comments on an issue	Votes	Keeps track of each user's votes on different Stack Overflow posts
pull requests	List of pull requests for base repo. Requests originated at head head_repo/ commit and are created by user_id	PostReference GH	Github hyperlink references found in Stack Overflow posts
pull request comments	Discussion comments on a pull request	Tags	Shows tag names and counts of tags in Stack Overflow posts
pull request history	Chronologically ordered list of events on a pull request	GHMatches	Helper table for Post ReferenceGH. Shows matching Github hyperlinks in Stack Overflow posts

Table 1: Short description of content in all tables within the GHTorrent and SOTorrent data sets.

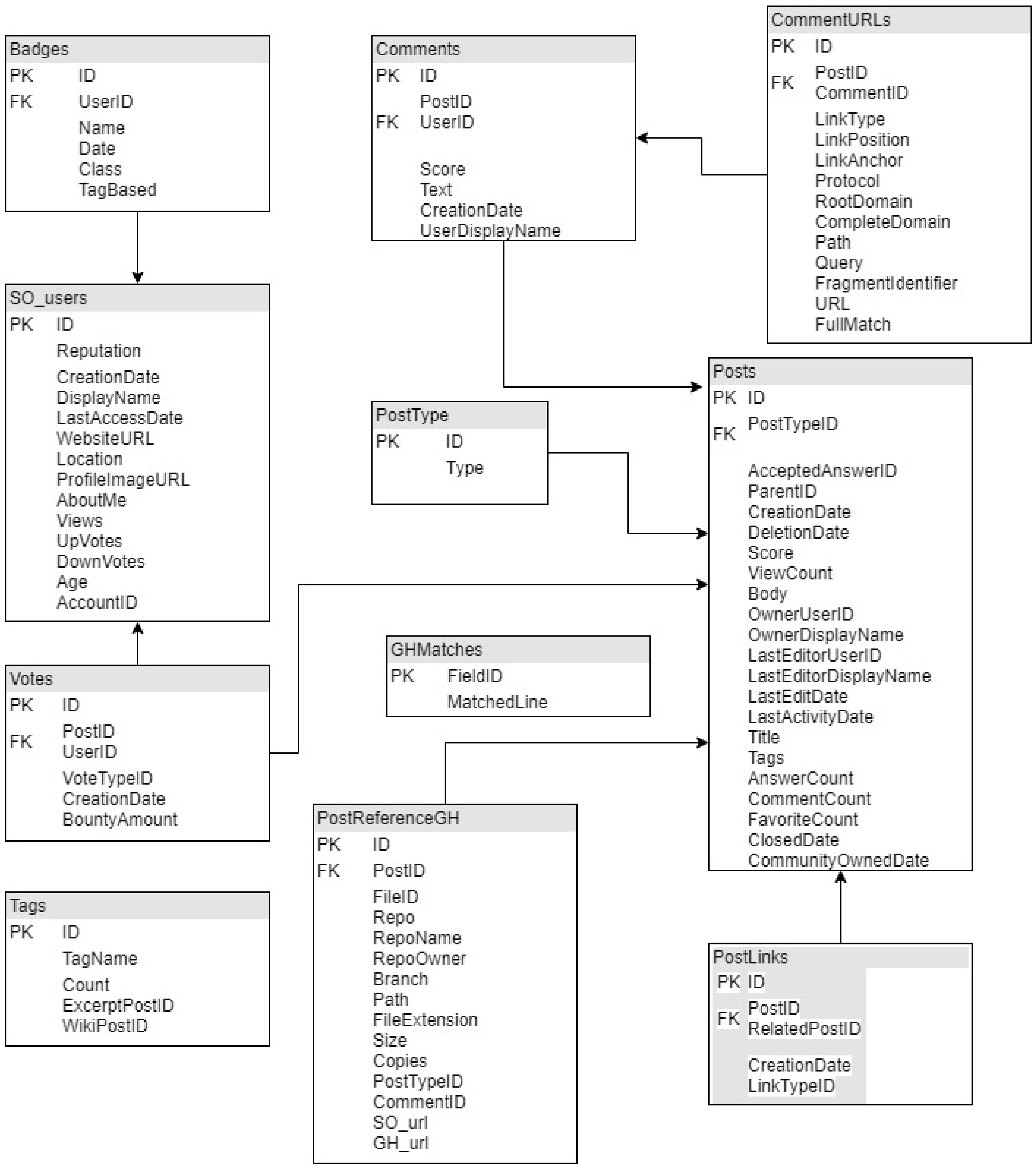

Figure 1: Redesigned database schema for Stack Overflow data.

from current and older versions of their database, thus currently no such linkage is possible.

The second candidate attribute pair was the *repo* attribute from SOTorrent's PostReferenceGH table, and a substring of *url* attribute in GHTorrent's Projects table. The *repo* attribute is a string consisting of the "*username/repo_name*" pattern.

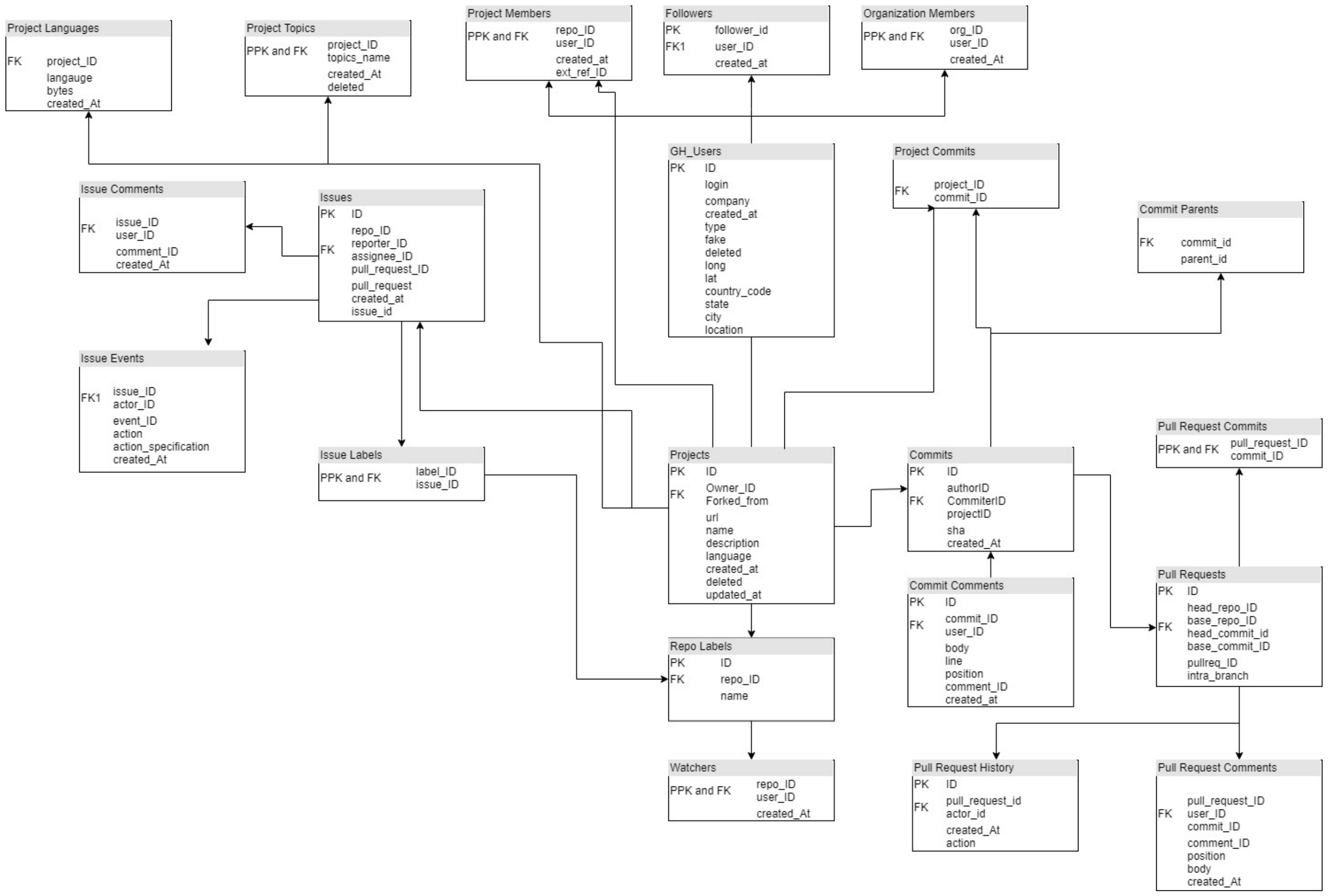
Project Languages
FK project_ID langauge bytes created_At
Project Topics
PPK and FK project_ID topics_name created_At deleted
Project Members
PPK and FK repo_ID user_ID created_at ext_ref_ID
Followers
PK follower_id
FK1 user_ID created_at
Organization Members
PPK and FK org_ID user_ID created_At
GH_Users
PK ID login company created_at type fake deleted long lat country_code state city location
Project Commits
FK project_ID commit_ID
Commit Parents
FK commit_id parent_id
Issue Comments
FK issue_ID user_ID comment_ID created_At
Issues
PK ID
FK repo_ID reporter_ID assignee_ID pull_request_ID pull_request created_at issue_id
Issue Events
FK1 issue_ID actor_ID event_ID action action_specification created_At
Issue Labels
PPK and FK label_ID issue_ID
Projects
PK ID
FK Owner_ID Forked_from url name description language created_at deleted updated_at
Commits
PK ID
FK authorID CommiterID projectID sha created_At
Pull Request Commits
PPK and FK pull_request_ID commit_ID
Commit Comments
PK ID
FK commit_ID user_ID body line position comment_ID created_at
Pull Requests
PK ID
FK head_repo_ID base_repo_ID head_commit_id base_commit_ID pullreq_ID intra_branch
Repo Labels
PK ID
FK repo_ID name
Watchers
PPK and FK repo_ID user_ID created_At
Pull Request History
PK ID
FK pull_request_id actor_id created_At action
Pull Request Comments
FK pull_request_ID user_ID commit_ID comment_ID position body created_At

Figure 2: Redesigned database schema for Github data.

The *url* attribute consists of a string with pattern "*https://api.github.com/username/repo_name/*". Taking a substring of GHTorrent's url attribute gives the same string pattern as the one in SOTorrent's repo attribute, thus the two databases can be linked using a query joining the attributes' two tables.

After considering what kind of join should be performed, the conclusion was reached that a left join between SOTorrent's PostReferenceGH and GHTorrent's Projects table is needed, as it would make more sense to keep everything in SOTorrent data, and only join GHTorrent's Projects that are a match. Completing this left join could be problematic, as SOTorrent's PostReferenceGH table has over 6 million rows, and GHTorrent's Projects table contains over 116 million rows. Performing a left join on two such tables is time consuming in best case scenario, while the server could time out or run out of computational resources in worst case scenario.

A much more fail-safe, well thought-out, computationally less expensive and less time consuming way needed to be used in order to avoid a potentially risky left join. This alternative, and better solution consisted of two main tasks: candidate attribute pair matching, and table stitching.

The following is the algorithm for candidate attribute pair matching:

(1) Create indices on the two target attributes that will be used for the linkage (repo and substring of url)
(2) Create master table containing partial primary keys (PPK) created from the two tables'

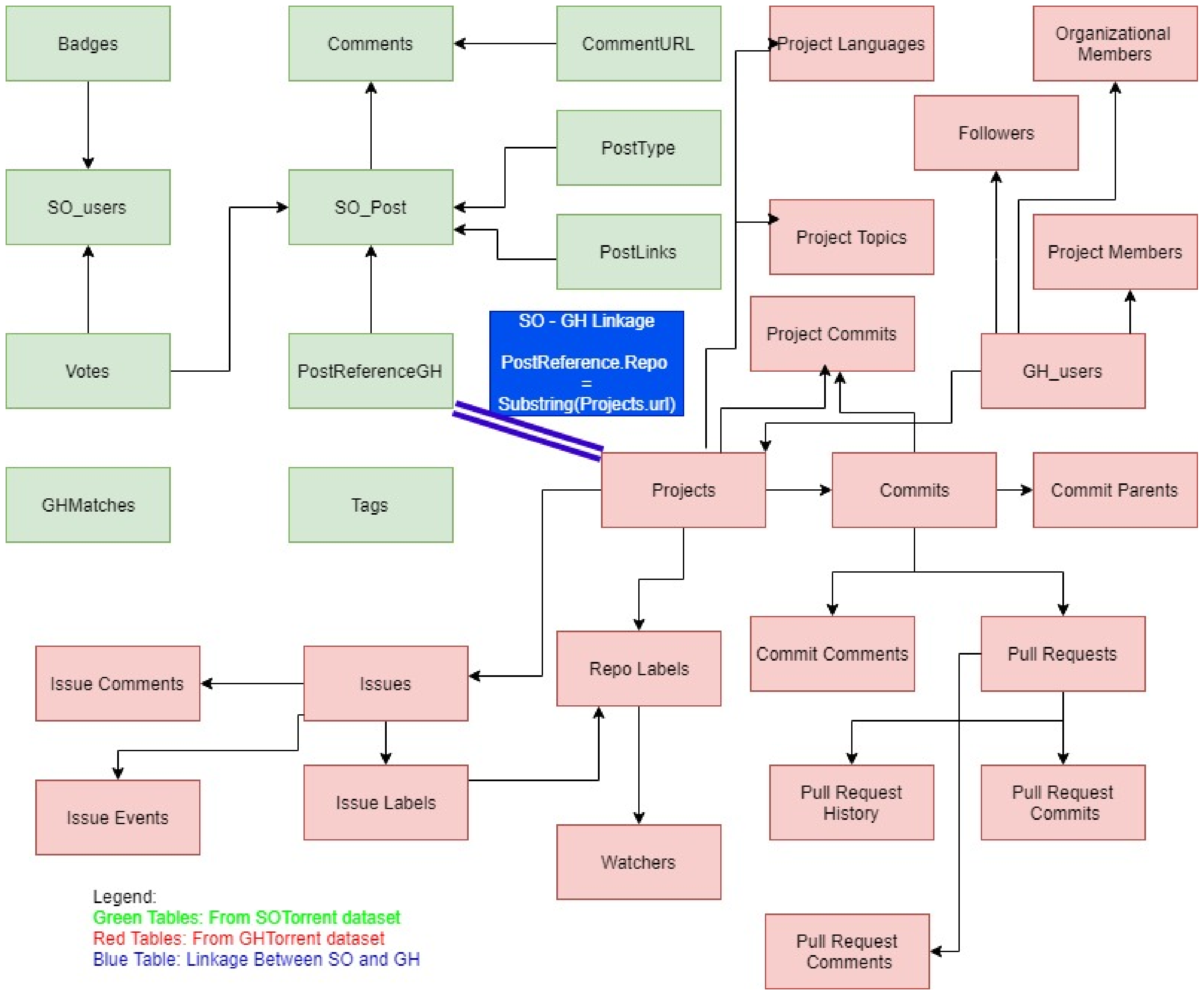

Figure 3: Stack Overflow and Github linked Database overview with linkage shown.

primary keys. Store important foreign keys (FK) of the two tables to be linked

(3) Run select statement for linking the two tables, then take the results and insert them into a master table
(4) Create a secondary linkage table that will not contain PostReferenceGH IDs, but rather just create linkage between GHTorrent's Projects and SOTorrent's Posts tables
(5) Finally, create indices on the PPKs of the master linkage table so the next task, table stitching will run faster

Note that step 3 runs very fast, as in step 1 indices are created on the attributes to be joined. Another note that in step 3 the WHERE clause has "PostReferenceGH.Repo = projects.url" as filtering condition. Step 4 is done just for convenience; it is more useful to have the SOTorrent's Posts table directly connected to GHTorrent's Project table, since they belong to the main tables of each database.

The following is the table stitching algorithm, which is much simpler:

(1) Create table GH_projects_linked and insert into it the result of a join between Master Linkage and GHTorrent's projects table
(2) Alter GH_projects_linked table to add the necessary PPKs and FKs
(3) Create table SO_Posts_linked and insert into it the result of a join between Master Linkage and SOTorrent's Posts table
(4) Alter SO_Posts_linked table to add the necessary PPKs and FKs
(5) Finally, create indices for the PPKs of the newly "stitched" tables

The result of candidate attribute pair matching and table stitching algorithms can be called the database linkage between SOTorrent and GHTorrent, and it can be visualized in figure 3.

3.4 Database Reduction

As previously mentioned, there was over 400 GB of GHTorrent data present in the database. It makes sense to try to reduce the amount of data to be mined. All Github data would represent too much data that can not be looked at all at once. The idea behind the database reduction was to only keep Github data that was successfully linked to Stack Overflow data. The linkage from section 3.3 occurred on the Projects table, which is the main table in GHTorrent with all other tables branching out it. Thus, the database reduction consisted of running *CREATE TABLE AS* statements with selecting every attribute of a particular table, and including a filtering condition in the *WHERE* clause to only keep rows which are connected or related to project IDs that exist in the projects_linked table obtained during table stitching. This database reduction routine was executed on all 21 tables of the GHTorrent database, then *ALTER TABLE* statements were executed to properly point the foreign keys to the newly created "reduced" tables, thus completing the table reduction. After this process the original table was dropped, thus reducing the size of the data set by allowing only linked, relevant data to be mined and analyzed in the future.

Relation Name	**Original Size (# of Rows)**	**Reduced Size (# of Rows)**
projects	116 217 069	399 262
commits	1 276 402 890	63 846 528
repo_labels	353 235 838	1 715 494
watchers	138 749 808	17 804 316
commit_ comments	5 489 034	699 044
commit_ parents	1 275 331 898	479 483 406
project_ commits	6 096 603 791	63 846 528
pull_requests	47 844 942	3 078 255
pull_request_ comments	32 182 723	4 371 393
pull_request_ history	124 096 184	8 208 929
pull_request_ commits	243 579 978	19 747 343
issues	91 057 372	3 119 766
issue_labels	25 710 804	6 718
issue_comments	139 761 328	5 962 468
issue_events	127 166 727	101 602
project_languages	128 592 248	2 260 577

Table 2: Table reduction results showing the original and reduced sizes of each GHTorrent table

4 RESULTS

There were no experiments conducted during the creation of the database. The only measurable results are table reduction results showing how much data got eliminated from the data set. Table 2 shows the original and reduced size of each table in GHTorrent, size being measured in number of rows. As one can see, the original size of most tables is hundreds of millions of rows, and in some cases even over a billion rows. To make the mining process easier, and more manageable, most tables within the GHTorrent data set got reduced to a few million, or couple of dozen million rows, which is more desirable. Most importantly the projects table got reduced from over 116 million projects to almost 400 000 projects. Mining data from over 100 million projects is way too large of a task, but even a sample size of 400 000 distinct projects will probably represent a challenge in the future. Two more tables that needed to be reduced to a smaller size were commits and project_commits. These two tables' original size is 1.276 billion rows and 6.096 billion rows. It became difficult to query these two tables. After the database reduction step the reduced size is 63.846 million rows for both of them, which allows faster querying and more reasonable analysis in the future.

Table 3 shows the reduction rate results from the database reduction task. Most tables get reduced by a rate of over 90 %, which is impressive. The average reduction rate is 92.8539 %. The only outlier is the commit_parents table, which stores inheritance relationships within a project's commits. This table only gets reduced by 62.4032 %. It is worth mentioning that after database reduction the total amount of data within the linked database (estimated 500 GB) got reduced to an estimated amount of 150-200 GB.

Relation Name	**Table Reduction Rate**
projects	0.996565
commits	0.949979
repo_labels	0.995143
watchers	0.871680
commit_ comments	0.872647
commit_ parents	0.624032
project_ commits	0.989528
pull_requests	0.935662
pull_request_ comments	0.864170
pull_request_ history	0.933850
pull_request_ commits	0.918929
issues	0.965738
issue_labels	0.999739
issue_comments	0.957338
issue_events	0.999201
project_languages	0.982421
average reduction	**0.928539**

Table 3: Reduction rate results from the database reduction task described in section 3.4

5 CONCLUSION

This project successfully linked two open data sets, SOTorrent and GHTorrent, which is all the more significant, since the only previous linkage (email hash) is not available anymore due to privacy concerns. The linkage of these two open data sets will allow more fruitful research to be carried out in the field of software repository mining, where especially SOTorrent is playing an important role, being the current mining challenge for MSR 2019 conference. As a result of this project a new database has been created, and irrelevant (not linked) data has been removed, making the average reduction rate for GHTorrent data to be 92.8539 %.

6 FUTURE WORK

The next important step in this project is to start the data cleaning process. All textual data needs to go through some text pre-processing, while some numerical and categorical data needs further processing, and formatting, for example counting how frequently an action (for instance, creating a pull request) happens. Once the data cleaning tasks are finished, the next steps will include applying a number of feature engineering techniques for all numerical data and fitting topic models for all textual data.

7 ACKNOWLEDGEMENTS

I would like to acknowledge the great advice received from my supervisor, Prof. Olga Baysal. I would like to thank Prof. Ahmed El-Roby for pointing out the gaps in my research and guiding me to a better path. Also I need to thank Saraj Manes, Diana Lucaci and Vasileios Lioutas for helping with technical details.

REFERENCES

[1] Arwan, A., Rochimah, S., and Akbar, R. J. Source code retrieval on stackoverflow using lda. In *2015 3rd International Conference on Information and Communication Technology (ICoICT)* (2015), IEEE, pp. 295–299.

[2] Baltes, S., and Dumani, L. Sotorrent dataset, Dec 2018.

[3] Baltes, S., Dumani, L., Treude, C., and Diehl, S. Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018* (2018), pp. 319–330.

[4] Baltes, S., Treude, C., and Diehl, S. Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets. *arXiv preprint arXiv:1809.02814* (2018).

[5] Gousios, G. The ghtorrent project.

[6] Gousios, G. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (Piscataway, NJ, USA, 2013), MSR '13, IEEE Press, pp. 233–236.

[7] Matter, D., Kuhn, A., and Nierstrasz, O. Assigning bug reports using a vocabulary-based expertise model of developers. In *2009 6th IEEE international working conference on mining software repositories* (2009), IEEE, pp. 131–140.

[8] Tian, Y., Kochhar, P. S., Lim, E.-P., Zhu, F., and Lo, D. Predicting best answerers for new questions: An approach leveraging topic modeling and collaborative voting. In *International Conference on Social Informatics* (2013), Springer, pp. 55–68.

[9] Vasilescu, B., Filkov, V., and Serebrenik, A. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing* (2013), IEEE, pp. 188–195.

[10] Wu, W., Zhang, W., Yang, Y., and Wang, Q. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-Pacific Software Engineering Conference* (2011), IEEE, pp. 389–396.