****

**CE/CZ 4031:**

**Database System Principles**

***Project 1: Querying Databases Efficiently***

|  |  |
| --- | --- |
| THA TOE OO @JODI | U1621448G |
| LIM ZHI EN | U1622411D |
| ANDY NG CHIN KUAN | U1621910C |
| MUHAMMAD AMIRUL AFIQ | U1620684A |

**Contents**

[Introduction 3](#_Toc21356046)

[Schema Design and Data Acquisition 4](#_Toc21356047)

[Data Acquisition 5](#_Toc21356048)

[Queries and Optimizing Queries 6](#_Toc21356049)

[Effect of database size on the query time 6](#_Toc21356050)

[Building an Index and Studying the Effect of the Index 8](#_Toc21356051)

[Advanced Part: Study the Effect of Cache 21](#_Toc21356052)

[Appendix A: Table Schema 22](#_Toc21356053)

[Appendix B: Queries and Results 23](#_Toc21356054)

[Query 1 23](#_Toc21356055)

[Query 2 23](#_Toc21356056)

[Query 3 24](#_Toc21356057)

[Query 4 25](#_Toc21356058)

[Query 5 26](#_Toc21356059)

[Query 6 27](#_Toc21356060)

[Query 7a 28](#_Toc21356061)

[Query 7b 29](#_Toc21356062)

[Query 8 29](#_Toc21356063)

[Appendix C: SAX Parser Source Code 30](#_Toc21356064)

[Author.java 30](#_Toc21356065)

[Authored.java 31](#_Toc21356066)

[CSVWriter.java 32](#_Toc21356067)

[Main.java 35](#_Toc21356068)

[Parser.java 36](#_Toc21356069)

[ParserHandler.java 37](#_Toc21356070)

[Publication.java 41](#_Toc21356071)

[Appendix D: Halving Dataset Source Code 43](#_Toc21356072)

[ExtractSample.py 43](#_Toc21356073)

# Introduction

In this assignment, our team have decided to use PostgreSQL as the database of our choice. The database for this project was created on a Windows laptop as shown below.



Fig 1

# Schema Design and Data Acquisition

During our discussion, we have decided to design our schema based on the recommended sets and relationship given by the lab manual. Three tables are created as shown:

1. Author
2. Authored
3. Publications

The ER Diagram for the schema is shown below.

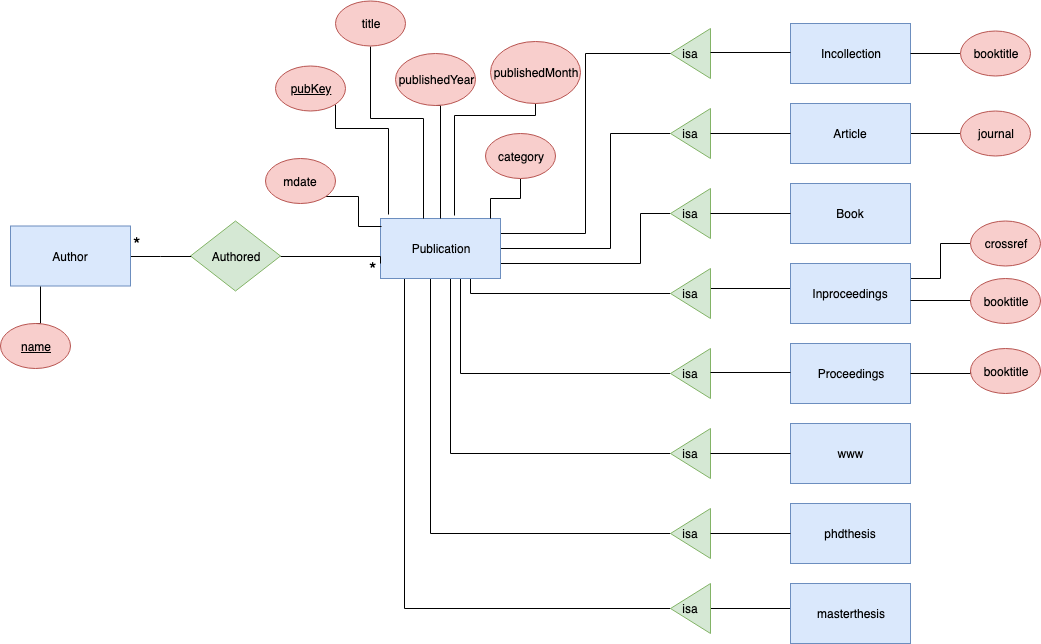


Fig 2

The commands used for creating the tables can be found in [Appendix A](#_Annex_A).

# Data Acquisition

In this section of the project, our team have used JAVA for the SAX Parser and to write the data into a CSV file. We have decided to follow the NULL approach to deal with the isa relationship in our relation schema.

During our design of the SAX Parser, we encountered various special cases that we had to deal with. One of the issues that we encountered was the selection of the right “CSV Separator” to distinguish the different columns while writing into a CSV file. In our experiment, we have found that common symbols like , or | is used within the file in the text and will disrupt the column segregation. As a result, we have decided to use the backtick symbol or ` as it is a unique symbol not found within the dblp file.

Another issue faced with the use of the backslash symbol and null values within the text of some titles. We have created a simple “hack” to deal with these cases and can be found in our "escapeSpecialCharacters” function.

Another special case that we found was some title uses tags such as "tt", "sub", "i", "sup" to represent inline teletype text, subscript, italics and superscript. In our experiment we realized that the titles will get shortened and ignored by the parser. To avoid such cases from happening, we detect those tags in the title and append any text found within them to the title string.

All the source codes can be found in [Appendix C](#_Appendix_C:_SAX) and [Appendix D](#_Appendix_D:_Halving). The github repository for our project is [here](https://github.com/akyulas/CE4031). It will be made public after the assignment deadline.

# Queries and Optimizing Queries

Code snippets and screen captures of the results for all the queries and can be found in [Appendix B](#_Appendix_B) of the report.

For our last query, we decided to find all the authors with the second highest number of publications.

Analysis:

Reducing the size of the database will reduce the query running time. Reducing the size of the database from full to half and from half to one-quarter will generally result in roughly similar decrease in query time in terms of percentage as can be seen from figure 2 and table 4. From these observations, we can conclude that it is possible that dataset can be split into smaller datasets and stored in separate bases and queried separately before the results are combined together for faster query time. It mainly applies to the queries that shows an improvement in query time of at least 50% if the database is reduced in half.

## Effect of database size on the query time

| **Queries** | **Before (s)** | **after ½ (s)** | **after ¼ (s)** |
| --- | --- | --- | --- |
| 1 | 6.763s | 3.457s | 1.789s |
| 2 | 12.764s | 6.645s | 3.251s |
| 3 | 2.656s | 1.455s | 0.826s |
| 4 | 66s | 26.649s | 12.196s |
| 5 | 17.955s | 8.423s | 4.046s |
| 6 | 2.527s | 1.369s | 0.714s |
| 7a | 32.664s | 9.54s | 2.521s |
| 7b | 19.295s | 8.351s | 2.495s |
| 8 | 178s | 46.457s | 12.967s |

Table 1

Fig 3

|  |  |  |
| --- | --- | --- |
| **Queries** | **Percentage Change (Full to Half)** | **Percentage Change (Half to Quarter)** |
| 1 | 48.88% | 48.25% |
| 2 | 47.94% | 51.08% |
| 3 | 45.22% | 43.23% |
| 4 | 59.62% | 54.23% |
| 5 | 53.09% | 51.96% |
| 6 | 45.83% | 47.85% |
| 7a | 70.79% | 73.57% |
| 7b | 56.72% | 70.12% |
| 8 | 73.90% | 72.09% |

Table 2

Fig 4

# Building an Index and Studying the Effect of the Index

During our experimentation with indexes, through the Explain Analyze keywords in PostgreSQL we found out that even with the creation of indexes, some of the queries were not making use of the indexes. This is because PostgreSQL decides to perform a sequential scan on any query that will hit a significant portion of a table. This is because, it is much faster to do so than to request for the index and then using the index for the query. To alleviate this, we decided to turn off sequential scan using the following command:

|  |
| --- |
| set enable\_seqscan = off; |

This forces PostgreSQL to use the indexes that we have created and thus shows the effect of index on the queries. On occasions where PostgreSQL uses the indexes naturally, we have kept the sequential scan settings on.

Analysis:

This shows that making use of the index or forcing the query to use an index will generally improve the query performance. However, this is not always the case as can be seen from query 3 and query 4, where the performance from using the index is much worse.

|  |  |  |  |
| --- | --- | --- | --- |
| **Queries** | **before** | **after index** | **seq scan** |
| 1 | 6.763s | 6.577s | off |
| 2 | 12.764s | 11.414s | off |
| 3 | 2.656s | 16.353 | off |
| 4 | 66s | 65s | on |
| 5 | 17.955s | 16.515s | on |
| 6 | 2.527s | 1.812s | off |
| 7a | 32.664s | 147s | off |
| 7b | 19.295s | 15.139s | on |
| 8 | 178s | 175s | off |

Table 3

Fig 5

Query 1)

Runtime without index: 6.763s

Runtime with index: 6.577s

Index:

create index publishedYear\_index

on publications(publishedyear);

Query Plan:

"HashAggregate  (cost=341547.15..341547.22 rows=7 width=16) (actual time=6329.914..6329.915 rows=8 loops=1)"

"  Group Key: category"

"  -> Bitmap Heap Scan on publications  (cost=82651.21..322072.62 rows=3894905 width=8) (actual time=2636.184..5482.216 rows=3922330 loops=1)"

"        Recheck Cond: ((publishedyear >= '2000'::text) AND (publishedyear <= '2018'::text))"

"        Rows Removed by Index Recheck: 561678"

"        Heap Blocks: exact=47792 lossy=66175"

"        -> Bitmap Index Scan on publishedyear\_index  (cost=0.00..81677.48 rows=3894905 width=0) (actual time=2627.110..2627.110 rows=3922330 loops=1)"

"              Index Cond: ((publishedyear >= '2000'::text) AND (publishedyear <= '2018'::text))"

Conclusion: Making use of the index is faster as this is a range query which is suitable for B-tree indexing.

Query 2)

Runtime without index: 12.764s

Runtime with index: 11.414s

Index:

create index category\_index

on publications(category);

Query Plan:

"Unique  (cost=614519.89..642932.31 rows=200 width=6) (actual time=9298.879..11566.899 rows=70 loops=1)"

"  -> Subquery Scan on conf  (cost=614519.89..642741.13 rows=76472 width=6) (actual time=9298.877..11566.710 rows=440 loops=1)"

"        -> GroupAggregate  (cost=614519.89..641976.41 rows=76472 width=19) (actual time=9298.876..11566.577 rows=440 loops=1)"

"              Group Key: publications.booktitle, publications.publishedyear"

"              Filter: (count(\*) > 500)"

"              Rows Removed by Filter: 41124"

"              -> Sort (cost=614519.89..620667.10 rows=2458882 width=11) (actual time=9289.252..11014.010 rows=2457260 loops=1)"

"                    Sort Key: publications.booktitle, publications.publishedyear"

"                    Sort Method: external merge  Disk: 58816kB"

"                    -> Bitmap Heap Scan on publications  (cost=50068.77..269465.84 rows=2458882 width=11) (actual time=309.351..1174.538 rows=2457260 loops=1)"

"                          Recheck Cond: (category = 'inproceedings'::text)"

"                          Heap Blocks: exact=62252"

"                          -> Bitmap Index Scan on category\_index  (cost=0.00..49454.05 rows=2458882 width=0) (actual time=297.634..297.634 rows=2457260 loops=1)"

"                                Index Cond: (category = 'inproceedings'::text)"

Conclusion: Making use of the index is faster as this makes use of bitmap index scan where PostgreSQL will first scan the index and compile the rows which are needed at the end of the scan. PostgreSQL will then use the result to fetch the rows from the table. This can result in a faster performance.

Query 3)

Runtime without index: 2.656s

Runtime with index: 16.353s

Index:

create index publishedYear\_index

on publications(publishedyear);

create index category\_index

on publications(category);

Query Plan:

"GroupAggregate  (cost=800780.57..800783.54 rows=66 width=68) (actual time=16473.405..16473.429 rows=10 loops=1)"

"  Group Key: t.yeardivision"

"  -> Sort  (cost=800780.57..800780.74 rows=66 width=12) (actual time=16473.390..16473.393 rows=85 loops=1)"

"        Sort Key: t.yeardivision"

"        Sort Method: quicksort  Memory: 28kB"

"        -> Subquery Scan on t  (cost=0.43..800778.58 rows=66 width=12) (actual time=0.467..16473.219 rows=85 loops=1)"

"              -> GroupAggregate  (cost=0.43..800777.92 rows=66 width=17) (actual time=0.465..16473.144 rows=85 loops=1)"

"                    Group Key: publications.publishedyear"

"                    -> Index Only Scan using publishedyear\_index on publications  (cost=0.43..777218.20 rows=4711713 width=5) (actual time=0.209..15812.216 rows=4751341 loops=1)"

"                          Filter: (publishedyear <> 'null'::text)"

"                          Rows Removed by Filter: 2353887"

"                          Heap Fetches: 7105228"

"Planning Time: 0.318 ms"

"Execution Time: 16473.520 ms"

Conclusion: We found that having an index has worsen the query time.

Query 4)

Runtime without index: 66s

Runtime with index: 65s

Index:

create index category\_index

on publications(category);

Query Plan:

"GroupAggregate  (cost=2863588.33..2934719.83 rows=1667 width=23) (actual time=64373.013..64739.813 rows=1 loops=1)"

"  Group Key: a1.author\_name"

"  Filter: (count(\*) = $1)"

"  Rows Removed by Filter: 162993"

"  InitPlan 2 (returns $1)"

"    -> Aggregate  (cost=2921311.91..2921311.92 rows=1 width=8) (actual time=32124.197..32124.197 rows=1 loops=1)"

"          -> GroupAggregate  (cost=2863588.33..2917145.28 rows=333330 width=23) (actual time=30953.034..32109.746 rows=162994 loops=1)"

"                Group Key: a1\_1.author\_name"

"                -> Sort (cost=2863588.33..2880329.55 rows=6696487 width=15) (actual time=30953.024..31871.704 rows=1308458 loops=1)"

"                      Sort Key: a1\_1.author\_name"

"                      Sort Method: external merge  Disk: 32008kB"

"                      -> Hash Join (cost=1092233.94..1875489.02 rows=6696487 width=15) (actual time=16039.934..25050.312 rows=1308458 loops=1)"

"                            Hash Cond: (a1\_1.publication\_key = publications\_2.pubkey)"

"                            Join Filter: (a1\_1.author\_name <> a2\_1.author\_name)"

"                            Rows Removed by Join Filter: 407156"

"                            -> Seq Scan on authored a1\_1  (cost=0.00..307811.61 rows=16491061 width=38) (actual time=0.065..2213.256 rows=16483353 loops=1)"

"                            -> Hash (cost=1025174.79..1025174.79 rows=2885212 width=59) (actual time=15905.485..15905.485 rows=407156 loops=1)"

"                                  Buckets: 65536 Batches: 128 Memory Usage: 800kB"

"                                  -> Hash Join (cost=409116.12..1025174.79 rows=2885212 width=59) (actual time=6205.936..15721.777 rows=407156 loops=1)"

"                                        Hash Cond: (a2\_1.publication\_key = publications\_2.pubkey)"

"                                        -> Seq Scan on authored a2\_1 (cost=0.00..307811.61 rows=16491061 width=38) (actual time=0.022..2931.124 rows=16483353 loops=1)"

"                                        -> Hash (cost=386293.34..386293.34 rows=1243103 width=21) (actual time=6113.188..6113.188 rows=129731 loops=1)"

"                                              Buckets: 65536 Batches: 32 Memory Usage: 726kB"

"                                              -> Seq Scan on publications publications\_2 (cost=82610.71..386293.34 rows=1243103 width=21) (actual time=612.059..6070.031 rows=129731 loops=1)"

"                                                    Filter: (((category = 'inproceedings'::text) AND (hashed SubPlan 1)) OR ((category = 'article'::text) AND (lower(journal) ~~ '%data%'::text)))"

"                                                    Rows Removed by Filter: 6975497"

"                                                    SubPlan 1"

"                                                      -> Bitmap Heap Scan on publications publications\_1 (cost=707.39..82610.01 rows=280 width=21) (actual time=16.104..608.492 rows=2732 loops=1)"

"                                                            Recheck Cond: (category = 'proceedings'::text)"

"                                                            Filter: (lower(title) ~~ '%data%'::text)"

"                                                            Rows Removed by Filter: 39184"

"                                                            Heap Blocks: exact=26059"

"                                                            -> Bitmap Index Scan on category\_index (cost=0.00..707.32 rows=35052 width=0) (actual time=10.948..10.948 rows=41916 loops=1)"

"                                                                  Index Cond: (category = 'proceedings'::text)"

"  -> Sort  (cost=2863588.33..2880329.55 rows=6696487 width=15) (actual time=31479.997..32388.626 rows=1308458 loops=1)"

"        Sort Key: a1.author\_name"

"        Sort Method: external merge  Disk: 32008kB"

"        -> Hash Join  (cost=1092233.94..1875489.02 rows=6696487 width=15) (actual time=16645.657..25635.362 rows=1308458 loops=1)"

"              Hash Cond: (a1.publication\_key = publications.pubkey)"

"              Join Filter: (a1.author\_name <> a2.author\_name)"

"              Rows Removed by Join Filter: 407156"

"              -> Seq Scan on authored a1  (cost=0.00..307811.61 rows=16491061 width=38) (actual time=0.124..2221.754 rows=16483353 loops=1)"

"              -> Hash (cost=1025174.79..1025174.79 rows=2885212 width=59) (actual time=16511.741..16511.741 rows=407156 loops=1)"

"                    Buckets: 65536 Batches: 128  Memory Usage: 800kB"

"                    -> Hash Join (cost=409116.12..1025174.79 rows=2885212 width=59) (actual time=6332.529..16320.124 rows=407156 loops=1)"

"                          Hash Cond: (a2.publication\_key = publications.pubkey)"

"                          -> Seq Scan on authored a2  (cost=0.00..307811.61 rows=16491061 width=38) (actual time=0.040..3094.176 rows=16483353 loops=1)"

"                          -> Hash (cost=386293.34..386293.34 rows=1243103 width=21) (actual time=6236.276..6236.276 rows=129731 loops=1)"

"                                Buckets: 65536 Batches: 32 Memory Usage: 726kB"

"                                -> Seq Scan on publications (cost=82610.71..386293.34 rows=1243103 width=21) (actual time=674.924..6192.822 rows=129731 loops=1)"

"                                      Filter: (((category = 'inproceedings'::text) AND (hashed SubPlan 3)) OR ((category = 'article'::text) AND (lower(journal) ~~ '%data%'::text)))"

"                                      Rows Removed by Filter: 6975497"

"                                      SubPlan 3"

"                                        -> Bitmap Heap Scan on publications publications\_3  (cost=707.39..82610.01 rows=280 width=21) (actual time=13.253..671.192 rows=2732 loops=1)"

"                                              Recheck Cond: (category = 'proceedings'::text)"

"                                              Filter: (lower(title) ~~ '%data%'::text)"

"                                              Rows Removed by Filter: 39184"

"                                              Heap Blocks: exact=26059"

"                                              -> Bitmap Index Scan on category\_index (cost=0.00..707.32 rows=35052 width=0) (actual time=9.610..9.610 rows=41916 loops=1)"

"                                                    Index Cond: (category = 'proceedings'::text)"

Conclusion: The improvement in query time with an index created has not resulted in a significant gain in performance. The improvement could be a result of using Bitmap Index Scan instead of Sequential scan.

Query 5)

Runtime without index: 17.955s

Runtime with index: 16.515s

Index:

create index category\_index

on publications(category);

Query Plan:

"Limit  (cost=1468886.42..1468886.44 rows=10 width=23) (actual time=17444.463..17444.468 rows=10 loops=1)"

"  -> Sort  (cost=1468886.42..1469719.74 rows=333330 width=23) (actual time=17444.461..17444.461 rows=10 loops=1)"

"        Sort Key: (count(\*)) DESC"

"        Sort Method: top-N heapsort  Memory: 25kB"

"        -> GroupAggregate  (cost=1433377.59..1458349.98 rows=333330 width=23) (actual time=16906.018..17418.850 rows=167634 loops=1)"

"              Group Key: authored.author\_name"

"              -> Sort (cost=1433377.59..1440590.62 rows=2885212 width=15) (actual time=16906.008..17328.148 rows=407156 loops=1)"

"                    Sort Key: authored.author\_name"

"                    Sort Method: external merge  Disk: 10040kB"

"                    -> Hash Join (cost=409116.12..1025174.79 rows=2885212 width=15) (actual time=6567.944..14979.929 rows=407156 loops=1)"

"                          Hash Cond: (authored.publication\_key = publications.pubkey)"

"                          -> Seq Scan on authored (cost=0.00..307811.61 rows=16491061 width=38) (actual time=0.026..2237.760 rows=16483353 loops=1)"

"                          -> Hash (cost=386293.34..386293.34 rows=1243103 width=21) (actual time=6409.385..6409.386 rows=129731 loops=1)"

"                                Buckets: 65536 Batches: 32 Memory Usage: 726kB"

"                                -> Seq Scan on publications (cost=82610.71..386293.34 rows=1243103 width=21) (actual time=741.802..6360.925 rows=129731 loops=1)"

"                                      Filter: (((category = 'inproceedings'::text) AND (hashed SubPlan 1)) OR ((category = 'article'::text) AND (lower(journal) ~~ '%data%'::text)))"

"                                      Rows Removed by Filter: 6975497"

"                                      SubPlan 1"

"                                        -> Bitmap Heap Scan on publications publications\_1  (cost=707.39..82610.01 rows=280 width=21) (actual time=14.384..737.519 rows=2732 loops=1)"

"                                              Recheck Cond: (category = 'proceedings'::text)"

"                                              Filter: (lower(title) ~~ '%data%'::text)"

"                                              Rows Removed by Filter: 39184"

"                                              Heap Blocks: exact=26059"

"                                              -> Bitmap Index Scan on category\_index (cost=0.00..707.32 rows=35052 width=0) (actual time=9.679..9.679 rows=41916 loops=1)"

"                                                    Index Cond: (category = 'proceedings'::text)"

Conclusion: The improvement in query time with an index created has not resulted in a significant gain in performance. The improvement could be a result of using Bitmap Index Scan instead of Sequential scan.

Query 6)

Runtime without index: 2.527s

Runtime with index: 1.812s

Index:

create index category\_index

on publications(category);

Query Plan:

"Nested Loop  (cost=281760.81..308238.94 rows=1 width=53) (actual time=1603.767..1778.996 rows=204 loops=1)"

"  -> HashAggregate  (cost=281760.25..281877.54 rows=3128 width=17) (actual time=1603.080..1612.619 rows=4958 loops=1)"

"        Group Key: publications\_1.crossref"

"        Filter: (count(\*) > 100)"

"        Rows Removed by Filter: 36623"

"        -> Bitmap Heap Scan on publications publications\_1  (cost=50068.77..269465.84 rows=2458882 width=9) (actual time=272.224..943.786 rows=2457260 loops=1)"

"              Recheck Cond: (category = 'inproceedings'::text)"

"              Heap Blocks: exact=62252"

"              -> Bitmap Index Scan on category\_index  (cost=0.00..49454.05 rows=2458882 width=0) (actual time=261.110..261.110 rows=2457260 loops=1)"

"                    Index Cond: (category = 'inproceedings'::text)"

"  -> Index Scan using publications\_pkey on publications  (cost=0.56..8.42 rows=1 width=74) (actual time=0.033..0.033 rows=0 loops=4958)"

"        Index Cond: (pubkey = publications\_1.crossref)"

"        Filter: ((category = 'proceedings'::text) AND (lower(title) ~~ '%data%'::text))"

"        Rows Removed by Filter: 1"

Conclusion: The improvement in query time with an index created has not resulted in a significant gain in performance. The improvement could be a result of using Bitmap Index Scan instead of Sequential scan.

Query 7a)

Runtime without index: 32.664s

Runtime with index: 147s

Index:

create index publishedYear\_index

on publications(publishedyear);

Query Plan:

"GroupAggregate  (cost=10002730021.38..10002976308.38 rows=264 width=30) (actual time=127481.841..145699.926 rows=57 loops=1)"

"  Group Key: authored.author\_name, author.name"

"  Filter: (count(DISTINCT publications.publishedyear) = 30)"

"  Rows Removed by Filter: 131606"

"  -> Merge Join  (cost=10002730021.38..10002975253.50 rows=52744 width=35) (actual time=126902.976..144802.380 rows=733349 loops=1)"

"        Merge Cond: (authored.author\_name = author.name)"

"        -> Sort (cost=10002729059.44..10002755431.77 rows=10548932 width=20) (actual time=126902.345..136146.236 rows=13769430 loops=1)"

"              Sort Key: authored.author\_name"

"              Sort Method: external merge  Disk: 407984kB"

"              -> Hash Join (cost=10000425973.55..10001065818.22 rows=10548932 width=20) (actual time=8195.469..21367.773 rows=13769620 loops=1)"

"                    Hash Cond: (authored.publication\_key = publications.pubkey)"

"                    -> Seq Scan on authored  (cost=10000000000.00..10000307811.61 rows=16491061 width=38) (actual time=0.030..2406.679 rows=16483353 loops=1)"

"                    -> Hash (cost=338090.52..338090.52 rows=4545042 width=26) (actual time=8159.493..8159.494 rows=4567189 loops=1)"

"                          Buckets: 65536 Batches: 128  Memory Usage: 2611kB"

"                          -> Bitmap Heap Scan on publications  (cost=96447.11..338090.52 rows=4545042 width=26) (actual time=3194.138..6725.398 rows=4567189 loops=1)"

"                                Recheck Cond: ((publishedyear >= '1990'::text) AND (publishedyear <= '2019'::text))"

"                                Rows Removed by Index Recheck: 146646"

"                                Heap Blocks: exact=48415 lossy=66177"

"                                -> Bitmap Index Scan on publishedyear\_index  (cost=0.00..95310.85 rows=4545042 width=0) (actual time=3182.372..3182.372 rows=4567189 loops=1)"

"                                      Index Cond: ((publishedyear >= '1990'::text) AND (publishedyear <= '2019'::text))"

"        -> Index Only Scan using author\_pkey on author  (cost=0.43..192897.12 rows=12039 width=15) (actual time=0.413..3707.850 rows=136399 loops=1)"

"              Filter: ("substring"(name, ((length(name) - strpos(reverse(name), ' '::text)) + 2), length(name)) ~~ 'H%'::text)"

"              Rows Removed by Filter: 2271397"

"              Heap Fetches: 2407796"

Conclusion: We found that having an index has worsen the query time.

Query 7b)

Runtime without index: 19.295s

Runtime with index: 15.139s

Index:

create index publishedYear\_index

on publications(publishedyear);

Query Plan:

"GroupAggregate  (cost=1722706.75..1743153.37 rows=333330 width=38) (actual time=15139.336..15139.380 rows=8 loops=1)"

"  Group Key: author.name, authored.author\_name"

"  -> Sort  (cost=1722706.75..1726985.08 rows=1711332 width=30) (actual time=15139.311..15139.317 rows=101 loops=1)"

"        Sort Key: author.name"

"        Sort Method: quicksort  Memory: 32kB"

"        -> Hash Join  (cost=914112.25..1463633.66 rows=1711332 width=30) (actual time=7967.607..15139.120 rows=101 loops=1)"

"              Hash Cond: (authored.author\_name = author.name)"

"              -> Seq Scan on authored  (cost=0.00..307811.61 rows=16491061 width=15) (actual time=0.081..2616.267 rows=16483353 loops=1)"

"              -> Hash (cost=909279.94..909279.94 rows=249865 width=30) (actual time=7859.811..7859.811 rows=8 loops=1)"

"                    Buckets: 65536 Batches: 8  Memory Usage: 513kB"

"                    -> Hash Join (cost=840294.50..909279.94 rows=249865 width=30) (actual time=7394.285..7858.018 rows=8 loops=1)"

"                          Hash Cond: (author.name = authored\_1.author\_name)"

"                          -> Seq Scan on author (cost=0.00..37929.96 rows=2407796 width=15) (actual time=0.020..203.873 rows=2407796 loops=1)"

"                          -> Hash (cost=835950.19..835950.19 rows=249865 width=15) (actual time=6969.566..6969.566 rows=8 loops=1)"

"                                Buckets: 131072 Batches: 4 Memory Usage: 1025kB"

"                                -> Unique (cost=832202.21..833451.54 rows=249865 width=15) (actual time=6968.569..6968.580 rows=8 loops=1)"

"                                      InitPlan 2 (returns $1)"

"                                        -> Result (cost=0.54..0.55 rows=1 width=32) (actual time=0.108..0.108 rows=1 loops=1)"

"                                              InitPlan 1 (returns $0)"

"                                                -> Limit (cost=0.43..0.54 rows=1 width=5) (actual time=0.105..0.105 rows=1 loops=1)"

"                                                      -> Index Only Scan using publishedyear\_index on publications publications\_1  (cost=0.43..777218.20 rows=7105228 width=5) (actual time=0.103..0.104 rows=1 loops=1)"

"                                                            Index Cond: (publishedyear IS NOT NULL)"

"                                                            Heap Fetches: 1"

"                                      -> Sort (cost=832201.66..832826.32 rows=249865 width=15) (actual time=6968.567..6968.569 rows=12 loops=1)"

"                                            Sort Key: authored\_1.author\_name"

"                                            Sort Method: quicksort Memory: 25kB"

"                                            -> Hash Join (cost=196121.11..805526.78 rows=249865 width=15) (actual time=107.990..6968.520 rows=12 loops=1)"

"                                                  Hash Cond: (authored\_1.publication\_key = publications.pubkey)"

"                                                  -> Seq Scan on authored authored\_1 (cost=0.00..307811.61 rows=16491061 width=38) (actual time=0.018..2322.160 rows=16483353 loops=1)"

"                                                  -> Hash (cost=194144.42..194144.42 rows=107655 width=21) (actual time=0.803..0.803 rows=12 loops=1)"

"                                                        Buckets: 65536 Batches: 2 Memory Usage: 513kB"

"                                                        -> Bitmap Heap Scan on publications (cost=2018.76..194144.42 rows=107655 width=21) (actual time=0.150..0.334 rows=12 loops=1)"

"                                                              Recheck Cond: (publishedyear = $1)"

"                                                              Heap Blocks: exact=11"

"                                                              -> Bitmap Index Scan on publishedyear\_index (cost=0.00..1991.84 rows=107655 width=0) (actual time=0.144..0.144 rows=12 loops=1)"

"                                                                    Index Cond: (publishedyear = $1)"

Conclusion: The improvement in query time with an index created has not resulted in a significant gain in performance. The improvement could be a result of using Bitmap Index Scan instead of Sequential scan.

Query 8)

Runtime without index: 178s

Runtime with index: 175s

Index:

create index category\_index

on publications(category);

Query Plan:

"Subquery Scan on inpro\_count  (cost=20003862526.74..20004005487.21 rows=28535 width=23) (actual time=172041.763..173097.188 rows=1 loops=1)"

"  InitPlan 1 (returns $0)"

"    -> Limit  (cost=10002045403.47..10002045403.47 rows=1 width=8) (actual time=85727.432..85727.436 rows=1 loops=1)"

"          -> Sort (cost=10002045403.47..10002059670.98 rows=5707005 width=8) (actual time=85727.430..85727.431 rows=2 loops=1)"

"                Sort Key: inpro\_count\_1.cnt DESC"

"                Sort Method: top-N heapsort  Memory: 25kB"

"                -> Subquery Scan on inpro\_count\_1  (cost=10001817123.27..10001988333.42 rows=5707005 width=8) (actual time=73310.455..85262.296 rows=4215549 loops=1)"

"                      -> GroupAggregate (cost=10001817123.27..10001931263.37 rows=5707005 width=28) (actual time=73310.453..84959.218 rows=4215549 loops=1)"

"                            Group Key: authored\_1.author\_name, publications\_1.publishedyear"

"                            -> Sort (cost=10001817123.27..10001831390.78 rows=5707005 width=20) (actual time=73310.443..82675.912 rows=7731253 loops=1)"

"                                  Sort Key: authored\_1.author\_name, publications\_1.publishedyear"

"                                  Sort Method: external merge Disk: 229272kB"

"                                  -> Hash Join (cost=10000317010.87..10000942594.54 rows=5707005 width=20) (actual time=2475.569..12090.161 rows=7731253 loops=1)"

"                                        Hash Cond: (authored\_1.publication\_key = publications\_1.pubkey)"

"                                        -> Seq Scan on authored authored\_1 (cost=10000000000.00..10000307811.61 rows=16491061 width=38) (actual time=0.179..2122.968 rows=16483353 loops=1)"

"                                        -> Hash (cost=269465.84..269465.84 rows=2458882 width=26) (actual time=1776.338..1776.338 rows=2457260 loops=1)"

"                                              Buckets: 65536 Batches: 64 Memory Usage: 2725kB"

"                                              -> Bitmap Heap Scan on publications publications\_1  (cost=50068.77..269465.84 rows=2458882 width=26) (actual time=238.507..1136.979 rows=2457260 loops=1)"

"                                                    Recheck Cond: (category = 'inproceedings'::text)"

"                                                    Heap Blocks: exact=62252"

"                                                    -> Bitmap Index Scan on category\_index (cost=0.00..49454.05 rows=2458882 width=0) (actual time=226.367..226.367 rows=2457260 loops=1)"

"                                                          Index Cond: (category = 'inproceedings'::text)"

"  -> GroupAggregate  (cost=10001817123.27..10001959798.39 rows=28535 width=28) (actual time=172041.761..173097.183 rows=1 loops=1)"

"        Group Key: authored.author\_name, publications.publishedyear"

"        Filter: (count(\*) = $0)"

"        Rows Removed by Filter: 4215548"

"        -> Sort (cost=10001817123.27..10001831390.78 rows=5707005 width=20) (actual time=75760.566..85185.412 rows=7731253 loops=1)"

"              Sort Key: authored.author\_name, publications.publishedyear"

"              Sort Method: external merge  Disk: 229272kB"

"              -> Hash Join (cost=10000317010.87..10000942594.54 rows=5707005 width=20) (actual time=2679.207..13441.611 rows=7731253 loops=1)"

"                    Hash Cond: (authored.publication\_key = publications.pubkey)"

"                    -> Seq Scan on authored  (cost=10000000000.00..10000307811.61 rows=16491061 width=38) (actual time=0.032..2446.565 rows=16483353 loops=1)"

"                    -> Hash (cost=269465.84..269465.84 rows=2458882 width=26) (actual time=1896.040..1896.040 rows=2457260 loops=1)"

"                          Buckets: 65536 Batches: 64  Memory Usage: 2725kB"

"                          -> Bitmap Heap Scan on publications  (cost=50068.77..269465.84 rows=2458882 width=26) (actual time=268.339..1214.491 rows=2457260 loops=1)"

"                                Recheck Cond: (category = 'inproceedings'::text)"

"                                Heap Blocks: exact=62252"

"                                -> Bitmap Index Scan on category\_index  (cost=0.00..49454.05 rows=2458882 width=0) (actual time=254.554..254.554 rows=2457260 loops=1)"

"                                      Index Cond: (category = 'inproceedings'::text)"

Conclusion: The improvement in query time with an index created has not resulted in a significant gain in performance. The improvement could be a result of using Bitmap Index Scan instead of Sequential scan.

# Advanced Part: Study the Effect of Cache

|  |  |  |  |
| --- | --- | --- | --- |
| **Queries** | **64MB (s)** | **128MB (s)** | **256MB (s)** |
| 1 | 6.649 | 6.763 | 6.314 |
| 2 | 12.146 | 12.764 | 12.148 |
| 3 | 2.854 | 2.656 | 2.931 |
| 4 | 65 | 66 | 69 |
| 5 | 18.742 | 17.955 | 18.119 |
| 6 | 2.571 | 2.527 | 2.483 |
| 7a | 32.834 | 32.664 | 32.683 |
| 7b | 19.453 | 19.295 | 18.71 |
| 8 | 178 | 178 | 178 |

Table 4

To complete this section of analysis, we adjusted the size of the Cache in PostgreSQL by changing the “shared-buffer” size to define the size of the cache. We did not see any considerable improvements in query time but also in some cases, query time has worsened. This can be seen from Table 4 where sometimes reducing the cache size improve performance instead of reducing it such as for query 1 where the caches size was reduced from 128MB to 64MB. The amount of time it takes for a query to run is about the same for different cache sizes as can be seen from Figure 6 where the line chart for different cache sizes looks identical and almost overlaps.

Fig 6

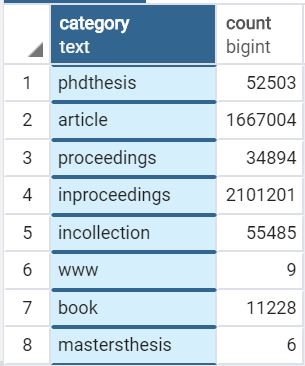
# Appendix A: Table Schema

|  |
| --- |
| CREATE TABLE Publications( |
| pubKey TEXT primary key, |
| mdate TEXT, |
| title TEXT, |
| category TEXT, |
| journal TEXT, |
| bookTitle TEXT, |
| publishedYear TEXT, |
| publishedMonth TEXT, |
| crossRef TEXT |
| ); |
|  |
| Drop Table IF EXISTS Author; |
|  |
| Create TABLE Author ( |
| name TEXT primary key |
| ); |
|  |
| DROP TABLE IF EXISTS authored; |
| Create TABLE authored ( |
| author\_name TEXT, |
| publication\_key TEXT |
| ); |
|  |
| copy author(name) |
| from '..\author.csv' |
| csv header;  delete from author where name = ‘name’; |
|  |
| copy authored |
| from '..\authored.csv' |
| delimiter '`'; |
| delete from authored where author\_name = 'author\_name'; |
|  |
| copy publications |
| from '..\publication.csv' |
| delimiter '`'; |
| delete from publications where pubKey = 'pubKey'; |
|  |
| ALTER TABLE authored |
| ADD CONSTRAINT verify\_authored FOREIGN KEY (author\_name) REFERENCES author (name) |
|  |
| ALTER TABLE authored |
| ADD CONSTRAINT verify\_authored FOREIGN KEY (author\_name) REFERENCES author (name) |

# Appendix B: Queries and Results

## Query 1

|  |
| --- |
| select category, count(\*) |
| from publications |
| where publishedyear between '2000' and '2018' |
| group by category |



Results of Query 1

## Query 2

|  |
| --- |
| select distinct booktitle |
| from ( |
| select booktitle, publishedyear, count(\*) as conf\_count |
| from publications |
| where category = 'inproceedings' |
| group by booktitle, publishedyear |
| ) conf |
| where conf\_count > 500 |



Results of Query 2

Total Number Result Rows : 70

## Query 3

|  |
| --- |
| SELECT concat(CAST(T.YearDivision \* 10 AS nchar(4)), N' - ', CAST(T.YearDivision \* 10 + 9 AS nchar(4))) AS YearRange, SUM(T.TotalCount) |
| FROM |
| ( |
| SELECT cast (publishedyear as int) / 10 AS YearDivision, COUNT(\*) AS TotalCount |
| FROM publications |
| where publishedyear <> 'null' |
| GROUP BY publishedyear |
| ) T |
| GROUP BY YearDivision |
|  |



Result of Query 3

## Query 4

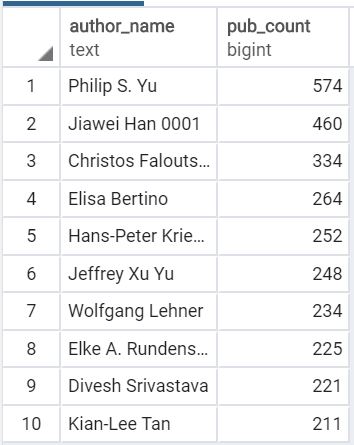
|  |
| --- |
| create view co\_count as( |
| select T3.author, T3.cnt from ( |
| select author, count(\*) as cnt from ( |
| select A1.author\_name as author, A2.author\_name as co\_author from authored A1 join ( |
| select \* from publications where (category = 'inproceedings' |
| and crossRef in (select pubkey from publications where category = 'proceedings' |
| and lower(title) like '%data%')) |
| or (category = 'article' and lower(journal) like '%data%') |
| )T1 on A1.publication\_key = T1.pubkey join authored A2 on A1.publication\_key = A2.publication\_key |
| and A1.author\_name <> A2.author\_name) T2 group by author) T3); |
|  |
| select author, cnt |
| from co\_count |
| where cnt = (select max(cnt) from co\_count); |



Result of Query 4

## Query 5

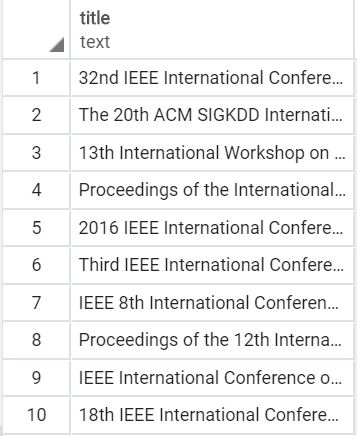
|  |
| --- |
| select \* |
| from ( |
| select author\_name, count(\*) as pub\_count |
| from authored join ( |
| select \* from publications where |
| (category = 'inproceedings' |
| and crossRef in (select pubkey from publications where category = 'proceedings' |
| and lower(title) like '%data%')) |
| or (category = 'article' and lower(journal) like '%data%') |
| ) T1 |
| on T1.pubkey = authored.publication\_key |
| group by author\_name |
| )T |
| order by pub\_count DESC |
| LIMIT 10; |
|  |



Result of Query 5

## Query 6

|  |
| --- |
| select title from publications |
| where category = 'proceedings' |
| and lower(title) like '%data%' |
| and pubkey in ( |
| select crossref from( |
| select crossref, count(\*) as cnt from publications where category = 'inproceedings' |
| group by crossref) T1 where T1.cnt > 100); |

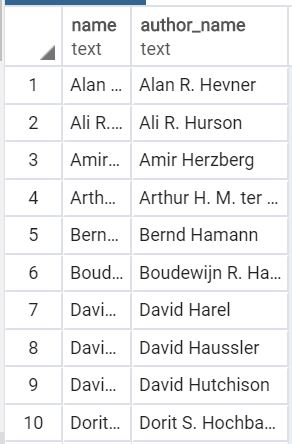


Result of Query 6

Total Number Result Rows : 204

## Query 7a

|  |
| --- |
| select author.name, authored.author\_name |
| from ( |
| author join authored on author.name = authored.author\_name |
| join publications on authored.publication\_key = publications.pubKey |
| ) |
| where publications.publishedYear Between '1990' and '2019' |
| and substring(author.name, length(author.name)-strpos(reverse(author.name),' ')+2, length(author.name)) like 'H%' |
| group by authored.author\_name, author.name |
| having count(distinct publications.publishedYear) = 30; |

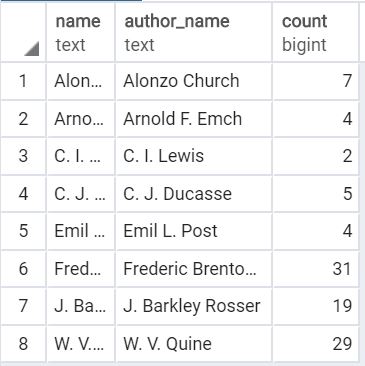


Result of Query 7a

Total Number Result Rows: 57

## Query 7b

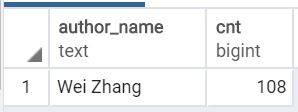
|  |
| --- |
| select author.name, authored.author\_name, count(\*) |
| from author join authored | |
| on author.name = authored.author\_name | |
| where author.name in ( | |
| select distinct authored.author\_name | |
| from authored join publications on authored.publication\_key = publications.pubKey | |
| where publications.publishedyear = (select min(publishedyear) from publications) | |
| ) | |
| group by author.name, authored.author\_name | |
|  |  | |



Result of Query 7b

## Query 8

|  |
| --- |
| create view inpro\_count as( |
| select author\_name, publishedyear, count(\*) as cnt from |
| authored join publications on authored.publication\_key = publications.pubKey |
| where publications.category = 'inproceedings' |
| group by author\_name, publishedyear |
| ); |
|  |
| select author\_name, cnt |
| from inpro\_count |
| where cnt = ( |
| select cnt |
| from inpro\_count |
| order by cnt desc limit 1 offset 1 |
| ) |



Result of Query 8

# Appendix C: SAX Parser Source Code

## Author.java

/\*\*

\* Created by jodiakyulas on 24/9/19.

\*/

public class Author {

@Override

public String toString() {

return "Author{" +

"name='" + name + '\'' +

'}';

}

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Author)) return false;

Author author = (Author) o;

return getName().equals(author.getName());

}

@Override

public int hashCode() {

return getName().hashCode();

}

}

## Authored.java

/\*\*

\* Created by jodiakyulas on 24/9/19.

\*/

public class Authored {

private String authorName;

private String publicationKey;

public String getAuthorName() {

return authorName;

}

public void setAuthorName(String authorName) {

this.authorName = authorName;

}

public String getPublicationKey() {

return publicationKey;

}

public void setPublicationKey(String publicationKey) {

this.publicationKey = publicationKey;

}

@Override

public String toString() {

return "Authored{" +

"authorName='" + authorName + '\'' +

", publicationKey='" + publicationKey + '\'' +

'}';

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Authored)) return false;

Authored authored = (Authored) o;

if (!getAuthorName().equals(authored.getAuthorName())) return false;

return getPublicationKey().equals(authored.getPublicationKey());

}

@Override

public int hashCode() {

int result = getAuthorName().hashCode();

result = 31 \* result + getPublicationKey().hashCode();

return result;

}

}

## **CSVWriter.java**

import java.io.\*;

import java.util.List;

import java.util.Set;

/\*\*

\* Created by jodiakyulas on 24/9/19.

\*/

public class CSVWriter {

private static final String publicationCSVName = "publication.csv";

private static final String authorCSVName = "author.csv";

private static final String authoredCSVName = "authored.csv";

private static final String CSV\_SEPARATOR = "`";

public static void writeToCSV(Set<String> authorSet, Set<Authored> authoredSet, List<Publication> publicationList) throws IOException {

writeToAuthorCSV(authorSet);

writeToAuthoredCSV(authoredSet);

writeToPublicationCSV(publicationList);

}

public static void createNewCSV() throws IOException {

createNewPublicationCSV();

createNewAuthorCSV();

createNewAuthoredCSV();

}

private static void createNewPublicationCSV() throws IOException {

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(publicationCSVName), "UTF-8"));

StringBuffer oneLine = new StringBuffer();

oneLine.append("pubKey");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("mdate");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("title");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("type");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("journal");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("bookTitle");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("publishedYear");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("publishedMonth");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("crossRef");

bw.write(oneLine.toString());

bw.newLine();

bw.flush();

bw.close();

}

private static void writeToPublicationCSV(List<Publication> publicationList) throws IOException {

if (publicationList.size() == 0) {

return;

}

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(publicationCSVName, true), "UTF-8"));

for (Publication publication: publicationList) {

StringBuffer oneLine = new StringBuffer();

oneLine.append(escapeSpecialCharacters(publication.getPubKey()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getMdate()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getTitle()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getType()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getJournal()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getBooktitle()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getYear()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getMonth()));

oneLine.append(CSV\_SEPARATOR);

oneLine.append(escapeSpecialCharacters(publication.getCrossRef()));

bw.write(oneLine.toString());

bw.newLine();

}

bw.flush();

bw.close();

}

private static void createNewAuthorCSV() throws IOException {

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(authorCSVName), "UTF-8"));

StringBuffer oneLine = new StringBuffer();

oneLine.append("name");

bw.write(oneLine.toString());

bw.newLine();

bw.flush();

bw.close();

}

private static void writeToAuthorCSV(Set<String> authorSet) throws IOException {

if (authorSet.size() == 0) {

return;

}

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(authorCSVName, true), "UTF-8"));

for (String author: authorSet) {

StringBuffer oneLine = new StringBuffer();

oneLine.append(author);

bw.write(oneLine.toString());

bw.newLine();

}

bw.flush();

bw.close();

}

private static void createNewAuthoredCSV() throws IOException {

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(authoredCSVName), "UTF-8"));

StringBuffer oneLine = new StringBuffer();

oneLine.append("author\_name");

oneLine.append(CSV\_SEPARATOR);

oneLine.append("publication\_key");

bw.write(oneLine.toString());

bw.newLine();

bw.flush();

bw.close();

}

private static void writeToAuthoredCSV(Set<Authored> authoredSet) throws IOException {

if (authoredSet.size() == 0) {

return;

}

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(authoredCSVName, true), "UTF-8"));

for (Authored authored: authoredSet) {

StringBuffer oneLine = new StringBuffer();

oneLine.append(authored.getAuthorName());

oneLine.append(CSV\_SEPARATOR);

oneLine.append(authored.getPublicationKey());

bw.write(oneLine.toString());

bw.newLine();

}

bw.flush();

bw.close();

}

private static String escapeSpecialCharacters(String string) {

if (string == null) {

return "null";

}

String result = string.replaceAll("\"", "\\\\\"");

result = result.replaceAll("\'", "\\\\'");

result = result.replaceAll("\\\\0", "\\\\\\\\0");

return result;

}

}

## **Main.java**

/\*\*

\* Created by jodiakyulas on 4/9/19.

\*/

public class Main {

public static void main(String[] args) {

if (args.length == 0) {

System.out.println("Please enter the file name as the first argument!");

System.exit(1);

}

try {

Parser parser = new Parser();

long startTime = System.nanoTime();

parser.parse(args[0]);

long endTime = System.nanoTime();

long duration = (endTime - startTime);

System.out.println(duration);

} catch (Exception e) {

e.printStackTrace();

System.err.println(e);

}

}

}

## P**arser.java**

/\*\*

\* Created by jodiakyulas on 4/9/19.

\*/

import org.xml.sax.SAXException;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import java.io.IOException;

import static com.sun.org.apache.xerces.internal.impl.Constants.JDK\_GENERAL\_ENTITY\_SIZE\_LIMIT;

import static com.sun.org.apache.xerces.internal.impl.Constants.JDK\_TOTAL\_ENTITY\_SIZE\_LIMIT;

public class Parser {

private SAXParser saxParser;

private ParserHandler parserHandler;

public Parser() throws ParserConfigurationException, SAXException {

parserHandler = new ParserHandler();

SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();

saxParserFactory.setNamespaceAware(true);

saxParserFactory.setValidating(true);

saxParser = saxParserFactory.newSAXParser();

saxParser.setProperty(JDK\_TOTAL\_ENTITY\_SIZE\_LIMIT, "0");

}

public void parse(String uri) throws ParserConfigurationException, SAXException, IOException {

saxParser.parse(uri, parserHandler);

}

}

## **ParserHandler.java**

import com.sun.xml.internal.ws.policy.privateutil.PolicyUtils;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

import java.io.IOException;

import java.util.\*;

/\*\*

\* Created by jodiakyulas on 6/9/19.

\*/

public class ParserHandler extends DefaultHandler{

private int level = 0;

private int count = 0;

private List<String> publicationType = Arrays.asList("incollection", "book", "inproceedings", "article", "proceedings", "www", "phdthesis", "mastersthesis");

private List<String> titleModifiers = Arrays.asList("tt", "sub", "i", "sup");

private Set<String> authorSet = new HashSet<String>();

private List<Publication> publicationList = new ArrayList<Publication>();

private Set<Authored> authoredSet = new HashSet<Authored>();

// private Set<Author> currentAuthorSet = new HashSet<Author>();

private Publication currentPublication = new Publication();

private Set<Authored> currentAuthoredSet = new HashSet<Authored>();

private String currentTitle = "";

private String currentPublicationKey = "";

private String currentAuthorName = "";

private String currentYear = "";

private String currentBookTitle = "";

private String currentJournal = "";

private String currentMonth = "";

private String currentCrossRef = "";

private boolean publication = false;

private boolean author = false;

private boolean title = false;

private boolean year = false;

private boolean journal =false;

private boolean booktitle =false;

private boolean titleModifier = false;

private boolean month = false;

private boolean crossRef = false;

@Override

public void startDocument() throws SAXException {

try {

CSVWriter.createNewCSV();

} catch(IOException ioException) {

ioException.printStackTrace();

System.out.println("Can't create new csv");

}

}

@Override

public void startElement(String uri, String eleName, String raw, Attributes attributes) throws SAXException {

level++;

if (publicationType.contains(raw)) {

count++;

if (count % 100 == 0) {

try {

writeToCSV(new HashSet<String>(), authoredSet, publicationList);

// authorSet.clear();

authoredSet.clear();

publicationList = new ArrayList<Publication>();

} catch(IOException ioException) {

ioException.printStackTrace();

System.out.println("Can't write to csv");

}

}

publication = true;

String pubKey = attributes.getValue("key");

String mdate = attributes.getValue("mdate");

currentPublicationKey = pubKey;

currentPublication.setType(raw);

currentPublication.setPubKey(pubKey);

currentPublication.setMdate(mdate);

} else if (publication) {

if (raw.equalsIgnoreCase("author")) {

author = true;

} else if (raw.equalsIgnoreCase("title")) {

title = true;

} else if (raw.equalsIgnoreCase("year")) {

year = true;

} else if (raw.equalsIgnoreCase("journal")) {

journal = true;

} else if (raw.equalsIgnoreCase("booktitle")) {

booktitle = true;

} else if (titleModifiers.contains(raw)) {

titleModifier = true;

} else if (raw.equalsIgnoreCase("month")) {

month = true;

} else if (raw.equalsIgnoreCase("crossref")) {

crossRef = true;

}

}

}

@Override

public void characters(char[] ch, int start, int length) throws SAXException {

String string = new String(ch, start, length);

if (author) {

currentAuthorName += string;

} else if (title) {

currentTitle += string;

} else if (year) {

currentYear += string;

} else if (journal) {

currentJournal += string;

} else if (booktitle) {

currentBookTitle += string;

} else if (titleModifier) {

currentTitle += string;

} else if (month) {

currentMonth += string;

} else if (crossRef) {

currentCrossRef += string;

}

}

@Override

public void endElement(String uri, String localName, String qName) throws SAXException {

if (level == 2 && publication) {

// if (currentAuthoredSet.isEmpty()) {

// Authored tempAuthored = new Authored();

// tempAuthored.setAuthorName("N/A");

// tempAuthored.setPublicationKey(currentPublicationKey);

// currentAuthoredSet.add(tempAuthored);

// }

// authorSet.addAll(currentAuthorSet);

authoredSet.addAll(currentAuthoredSet);

publicationList.add(currentPublication);

// currentAuthorSet.clear();

currentAuthoredSet.clear();

currentPublication = new Publication();

currentPublicationKey = "";

publication = false;

} else if (level == 4 && titleModifier) {

titleModifier = false;

} else if (level == 3 && title) {

currentPublication.setTitle(currentTitle);

currentTitle = "";

title = false;

} else if (level == 3 && author) {

authorSet.add(currentAuthorName);

Authored tempAuthored = new Authored();

tempAuthored.setAuthorName(currentAuthorName);

tempAuthored.setPublicationKey(currentPublicationKey);

currentAuthoredSet.add(tempAuthored);

currentAuthorName = "";

author = false;

} else if (level == 3 && year) {

currentPublication.setYear(currentYear);

currentYear = "";

year = false;

} else if (level == 3 && journal) {

currentPublication.setJournal(currentJournal);

currentJournal = "";

journal =false;

} else if (level == 3 && booktitle) {

currentPublication.setBooktitle(currentBookTitle);

currentBookTitle = "";

booktitle = false;

} else if (level == 3 && month) {

currentPublication.setMonth(currentMonth);

currentMonth = "";

month = false;

} else if (level == 3 && crossRef) {

currentPublication.setCrossRef(currentCrossRef);

currentCrossRef = "";

crossRef = false;

}

level--;

}

@Override

public void endDocument() {

try {

writeToCSV(authorSet, authoredSet, publicationList);

} catch(IOException ioException) {

ioException.printStackTrace();

System.out.println("Can't write to csv");

}

}

private void writeToCSV(Set<String> authorSet, Set<Authored> authoredSet, List<Publication> publicationList) throws IOException {

CSVWriter.writeToCSV(authorSet, authoredSet, publicationList);

}

}

## **Publication.java**

/\*\*

\* Created by jodiakyulas on 24/9/19.

\*/

public class Publication {

private String mdate;

private String title;

private String type;

private String pubKey;

private String journal;

private String booktitle;

private String year;

private String month;

private String crossRef;

public String getMdate() {

return mdate;

}

public void setMdate(String mdate) {

this.mdate = mdate;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public String getPubKey() {

return pubKey;

}

public void setPubKey(String pubKey) {

this.pubKey = pubKey;

}

public String getJournal() {

return journal;

}

public void setJournal(String journal) {

this.journal = journal;

}

public String getBooktitle() {

return booktitle;

}

public void setBooktitle(String booktitle) {

this.booktitle = booktitle;

}

public String getYear() {

return year;

}

public void setYear(String year) {

this.year = year;

}

public String getMonth() {

return month;

}

public void setMonth(String month) {

this.month = month;

}

public String getCrossRef() {

return crossRef;

}

public void setCrossRef(String crossRef) {

this.crossRef = crossRef;

}

@Override

public String toString() {

return "Publication{" +

"mdate='" + mdate + '\'' +

", title='" + title + '\'' +

", type='" + type + '\'' +

", pubKey='" + pubKey + '\'' +

", journal='" + journal + '\'' +

", booktitle='" + booktitle + '\'' +

", year='" + year + '\'' +

'}';

}

}

# Appendix D: Halving Dataset Source Code

## **ExtractSample.py**

# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 12:20:12 2019

@author: Zhi En

"""

import os

def extractSample(original\_file\_dir, sample\_dir,part\_size,wanted\_part):

header = getHeader(original\_file\_dir)

total\_rows = getTotalRows(original\_file\_dir)

total\_parts = (total\_rows // part\_size) + 1

file = open(original\_file\_dir, 'r',encoding="utf8")

sample = open(sample\_dir,'w',encoding="utf8")

if(wanted\_part > total\_parts):

print("wanted part out of total number of part")

return -1;

if(wanted\_part != 1):

sample.write(header)

for a in range(1, total\_parts + 1):

if(a == wanted\_part):

for i in range (0,part\_size):

line = file.readlines(1)

if line:

sample.write(line[0])

else:

return 1;

break;

else:

for i in range(0,part\_size):

file.readlines(1)

os.chmod(sample\_dir, 0o777)

file.close()

sample.close()

def extractAltSample(original\_file\_dir, sample\_dir, size\_reduce\_factor):

file = open(original\_file\_dir,'r', encoding = 'utf-8')

line = file.readlines(1)

if(line == ''): #empty csv

file.close()

return -1

sample = open(sample\_dir,'w',encoding = 'utf-8')

counter = size\_reduce\_factor - 1 #track iteration to write to file

while(line):

if(counter % size\_reduce\_factor == 0):

sample.write(line[0])

counter += 1

line = file.readlines(1)

print("lines written to " + sample\_dir + ": " , counter // size\_reduce\_factor)

file.close()

sample.close()

return 1

def getTotalRows(file\_dir):

file = open(file\_dir,'r',encoding="utf8")

rows = 0

for line in file:

rows += 1

file.close()

return rows

def getHeader(file\_dir):

file = open(file\_dir,encoding="utf8");

return file.readlines(1)[0];