SCHOOF OF COMPUTER SCIENCE AND ENGINEERING

SD6103 Data Systems

Project Report

Name	Matriculation No.	Contribution
Xu Baihui	G2303079C	Schema Design & Data Acquisition; Queries and Optimization; Indexing; Report
Eugene Ho Hong Zhuang	G2204889B	Schema Design & Data Acquisition; Queries and Optimization; Indexing; Report
Cheng Xinglan	G2304175J	Schema Design & Data Acquisition; Queries and Optimization; Indexing; Report
Duan Yuxuan	G2303761K	NO CONTRIBUTION AT ALL

1. INTRODUCTION

In this project, we designed and developed a relational database system utilizing MySQL, specifically targeting the DBLP dataset. This dataset comprises an extensive bibliography from prominent computer science journals and proceedings. After processing and integrating the DBLP dataset into MySQL, we created a series of queries. We then further explored and optimised these queries' efficiency through the use of relevant indexes.

The machine specifications used for developing the database, running queries, and performing respective tasks in this project are detailed in the following configuration:

Item	Value
OS Name	Microsoft Windows 10 Home
Version	10.0.19045 Build 19045
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	LAPTOP-P8D2FR88
System Manufacturer	Razer
System Model	Blade 14 - RZ09-0370
System Type	x64-based PC
System SKU	RZ09-0370CEA3
Processor	AMD Ryzen 9 5900HX with Radeon Graphics, 3301 Mhz, 8 Core(s), 16 Logical Processor(s)
BIOS Version/Date	Razer 1.01, 29/4/2021
SMBIOS Version	3.3
Embedded Controller V...	1.02
BIOS Mode	UEFI
BaseBoard Manufacturer	Razer
BaseBoard Product	PI411
BaseBoard Version	4
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\Windows
System Directory	C:\Windows\system32
Boot Device	\Device\HarddiskVolume2
Locale	United States
Hardware Abstraction L...	Version = "10.0.19041.3636"
User Name	LAPTOP-P8D2FR88\geneh
Time Zone	Malay Peninsula Standard Time
Installed Physical Mem...	16.0 GB
Total Physical Memory	15.4 GB
Available Physical Mem...	4.63 GB
Total Virtual Memory	29.4 GB
Available Virtual Memory	13.2 GB
Page File Space	14.0 GB
Page File	C:\pagefile.sys
Kernel DMA Protection	Off
Virtualization-based se...	Not enabled
Device Encryption Supp...	Elevation Required to View
Hyper-V - VM Monitor ...	Yes
Hyper-V - Second Level...	Yes
Hyper-V - Virtualizatio...	Yes
Hyper-V - Data Executi...	Yes

Figure 1. Machine Specification

2. Schema Design and Data Acquisition

The DBLP dataset (UNIVERSITAT TRIER, n.d.) serves as an extensive bibliographic resource in computer science, notable for its variety of publication types. It comprises ten unique publication elements: "article," "inproceedings," "proceedings," "book," "incollection," "phdthesis," "mastersthesis," "www," "person," and "data." Each element is further detailed with a range of bibliographic information, organized into attributes such as 'mdate', 'publtype', 'key', 'type', 'author', 'title', 'journal', 'year', 'ee', 'editor', 'publisher', 'isbn', 'volume', 'month', 'url', 'note', 'cdrom', 'booktitle', 'series', 'pages', 'crossref', 'school', 'cite', 'number', 'publnr', 'chapter', 'address' as described in Fig. 2. Additionally, DBLP entries often include cross-references ('crossref') to other works and external links ('ee') to their digital editions, creating a vast network of academic resources. This interconnectivity plays a crucial role in tracing the development of ideas and the dynamics of research collaborations within the field. An example of a DBLP XML element is showcased in Figure 2.

```
▼<proceedings key="conf/er/2019" mdate="2020-03-27">
    <editor>Alberto H. F. Laender</editor>
    <editor orcid="0000-0002-2034-9774">Barbara Pernici</editor>
    <editor orcid="0000-0003-0065-8665">Ee-Peng Lim</editor>
    <editor>José Palazzo M. de Oliveira</editor>
    <title>Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings</title>
    <booktitle>ER</booktitle>
    <publisher>Springer</publisher>
    <year>2019</year>
    <series href="db/series/lncs/index.html">Lecture Notes in Computer Science</series>
    <volume>11788</volume>
    <isbn>978-3-030-33222-8</isbn>
    <isbn>978-3-030-33223-5</isbn>
    <ee>https://doi.org/10.1007/978-3-030-33223-5</ee>
    <url>db/conf/er/er2019.html</url>
  </proceedings>
</dblp>
```

Figure 2. An example of a single publication elements

In the project, we have only utilised the eessential attributes/columns in the DBLP XML that are necessary for us to conduct the queries in "Queries and Optimizing Queries" section. The Entity-Relationship (ER) diagram used for this project is provided in . It depicts the authored table as maintaining a **many-to-many** relationship with both the publication table and the author table.

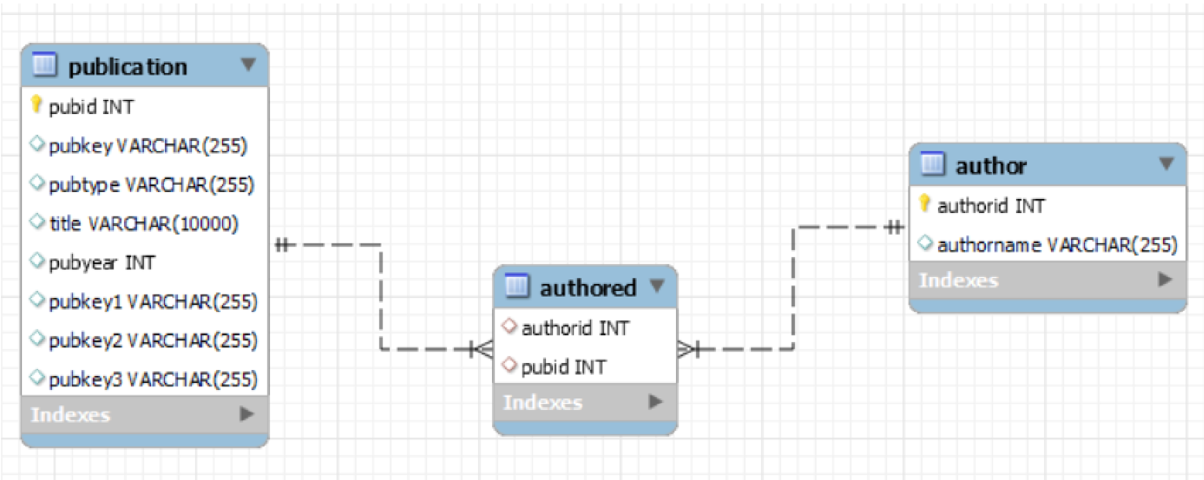

Figure 3. Entity Relational Diagram

The publication table would first contain the pubid(Primary Key, PK), pubkey (a unique identifier for each publication), pubtype (indicating the type of publication, such as article, inproceedings, proceedings, etc.), title (publication element title, inclusive of date of publication), pubyear (denoting the year of publication). We have also removed records that have pubtype = 'www', 'data', 'mastersthesis' and 'phdthesis', which are not essential for our queries. Subsequently in MYSQL, the pubkey is further segmented into three components: pubkey1 (identifying the publication's category, like 'conf' for conferences), pubkey2 (specifying the specific conference, e.g., 'ER' for Entity-Relationship), and pubkey3 (representing the publication year), which have been separated by '/'. The motivation of segmentating pubkey into pubkey1, pubkey2, and pubkey3 allows for direct access to specific segments of pubkey, catering to frequent queries that need only parts of this data, thereby bypassing the need to parse the entire string.

The database design further includes an author table, which is uniquely identified by authorID (Primary Key, PK) and the authorname to represent the first and last name of the author. Additionally, an authored table has been created to serve as a critical link between the publication and author tables, thereby preserving the relational structure of the database. This table would only contain both pubid and authorid (Foreign Keys, FK). The Entity-Relationship (ER) diagram, shown in Figure 3, illustrates that the authored table establishes a many-to-many relationship with both the publication table and the author table, facilitating a comprehensive and interconnected database schema.

The DBLP dataset was initially obtained in XML format and then transformed into CSV format using a SAX XML parser (Leonswl, 2023) prior to the insertion into MySQL. However, irregularities were identified in the CSV file, notably in the 'year' field of publications, where entries appeared in formats like "2 015". Hence a separate python code was developed to convert "2 015" to "2015". Subsequently, the DBLP.CSV file was used to further generate two distinct CSV files: author.CSV, containing author names, and authored.CSV which has listed all the pubid that are linked to corresponding authorid. This linkage is vital for establishing connections between the publication and author tables. The SAX Parser's source code, which elaborates on this data transformation process, is provided in ANNEX section A.

After acquiring the three CSV files, we proceed with their bulkloading into MySQL. The publication table import data from DBLP.CSV, but only the selected attributes as outlined in the previous section. Similarly, the author table will load data from author.CSV, and the authored table will import from authored.CSV. The source code facilitating the creation, modification, and bulk loading/insertion of the DBLP dataset has been developed in MySQL. It is included in ANNEX Section B.

3. Queries and Optimizing Queries

3.1. Queries Answer

Referencing the machine specifications depicted in Figure 1, Queries 1 to 8 have been successfully executed. The source code for each query, along with their respective partial answers, is provided below due to space constraints. For a comprehensive view of the complete answers, please refer to the CSV files that are attached with this report.

Query 1: For each type of publication, count the total number of publications of that type between 2010- 2019. Your query should return a set of (publication-type, count) pairs. For example, (article, 20000), (inproceedings, 30000), ...

```
SELECT
   pubtype AS PublicationType,
   COUNT(DISTINCT pubkey) AS PublicationCount
FROM
   publication
WHERE pubyear > 2010
AND pubyear < 2019
GROUP BY pubtype
ORDER BY PublicationCount DESC;
```

PublicationType	PublicationCount
proceedings	21363
inproceedings	1257982
incollection	30511
book	5174
article	1059262

Query 1 Answer

Query 2: Find all the conferences that have ever published more than 500 papers in one year. Note that one conference may be held every year (e.g., KDD runs many years, and each year the conference has a number of papers).

```
SELECT DISTINCT ConfName
FROM(
  SELECT
    pubkey2 AS ConfName,
        pubyear,
        COUNT(*) as ConfCount
  FROM publication
  WHERE pubkey1 = 'conf'
  GROUP BY ConfName, pubyear
) tmp
WHERE tmp.ConfCount > 500;
```

	ConfName
▶	aaai
	acc
	acl
	ahfe
	aiam
	amcc
	amcis
	amia
	apccas
	ascc
	atal
	bibm
	bigdataconf

Query 2 Answer

Query 3: For each 10 consecutive years starting from 1970, i.e., [1970, 1979], [1980, 1989],…, [2010, 2019], compute the total number of conference publications in DBLP in that 10 years. Hint: for this query you may want to compute a temporary table with all distinct years.

```
WITH
    pubyear_range AS (
        SELECT DISTINCT pubyear
        FROM publication
        WHERE pubyear >= 1970
),

    pubyear_groups AS (
        SELECT
            pubyear,
            ((pubyear - 1970) DIV 10) AS group_num
        FROM pubyear_range
    ),

    publication_groups AS (
        SELECT
            pubyear_groups.group_num,
            COUNT(*) AS num_publications
        FROM pubyear_groups
        JOIN publication ON publication.pubyear = pubyear_groups.pubyear
        WHERE publication.PubKey1 = ’conf’
        GROUP BY pubyear_groups.group_num
    )

SELECT
    CONCAT(’[’, 1970 + group_num*10, ’, ’, 1979 + group_num*10, ’]’) AS pubyear_ra
nge,
    num_publications
FROM
```

```
    publication_groups
ORDER BY
    group_num;
```

	pubyear_range	num_publications
▶	[1970, 1979]	16135
	[1980, 1989]	60572
	[1990, 1999]	240948
	[2000, 2009]	852445
	[2010, 2019]	1592377
	[2020, 2029]	621158

Query 3 Answer

Query 4: Find the most collaborative authors who published in a conference or journal whose name contains "data" (e.g., ACM SIGKDD International Conference on Knowledge Discovery and Data Mining). That is, for each author determine its number of collaborators, and then find the author with the most number of collaborators. Hint: for this question you may want to compute a temporary table of coauthors.

```
SELECT
    a.authorname AS author,
    COUNT(DISTINCT pa2.authorid) AS collaboratorscount
FROM
    Author a
JOIN
    authored pa1 ON a.authorid = pa1.authorid
JOIN
    Publication p ON pa1.pubid = p.pubid
JOIN
    authored pa2 ON pa1.pubid = pa2.pubid AND pa1.authorid != pa2.authorid
WHERE
    (p.pubkey1 = 'journals' OR p.pubkey1 = 'conf')
    AND LOWER(p.title) LIKE '%data%'
GROUP BY
    a.authorid
ORDER BY
    collaboratorscount DESC
LIMIT 10;
```

author	collaboratorscount
Xin Li	738
Yang Liu	634
Wei Zhang	596
Chang Liu	573
Yang Zhang	568
Jing Li	543
Wei Liu	539
Yu Zhang	528
Wei Li	506
Tao Zhang	497

Query 4 Answer

Query 5: . Data analytics and data science are very popular topics. Find the top 10 authors with the largest number of publications that are published in conferences and journals whose titles contain word “Data” in the last 5 years.

```
SELECT
  author.authorname AS author,
    COUNT(*) AS num_publications
FROM publication
INNER JOIN authored ON publication.pubid = authored.pubid
INNER JOIN author ON authored.authorID = author.authorid
WHERE publication.pubyear >= YEAR(CURDATE()) - 5
AND publication.title LIKE '%Data%'
AND (publication.PubKey1 = 'conf' OR publication.PubKey1 = 'journals')
GROUP BY author.authorname
ORDER BY num_publications DESC
LIMIT 10;
```

author	num_publications
No Author	1865
Wei Zhang	118
Kim-Kwang Raymond Choo	118
Yang Liu	116
Mohsen Guizani	115
Alfredo Cuzzocrea	113
Chin-Chen Chang 0001	104
Hao Wang	101
Carson K. Leung	100
Witold Pedrycz	98

Query 5 Answer

Query 6: List the name of the conferences such that it has ever been held in June, and the corresponding proceedings (in the year where the conference was held in June) contain more than 100 publications.

```
SELECT pubkey2, pubyear, pubtype, COUNT(*)
FROM publication
WHERE pubkey2 IN (SELECT DISTINCT(pubkey2)
                  FROM publication
                  WHERE title LIKE '%June%'
                  AND pubtype = 'inproceedings'
                  AND pubkey1 LIKE 'conf%')
GROUP BY pubkey2, pubyear, pubtype
HAVING COUNT(*) > 100
ORDER BY COUNT(*) DESC;
```

pubkey2	pubyear	pubtype	COUNT(*)
igarss	2019	inproceedings	2274
igarss	2018	inproceedings	2208
igarss	2021	inproceedings	2152
igarss	2022	inproceedings	1936
igarss	2016	inproceedings	1905
igarss	2012	inproceedings	1902
igarss	2020	inproceedings	1641
igarss	2017	inproceedings	1569
igarss	2003	inproceedings	1518
chi	2023	inproceedings	1498
chi	2020	inproceedings	1367
igarss	2007	inproceedings	1354
igarss	2015	inproceedings	1353
igarss	2008	inproceedings	1337

Query 6 Answer

Query 7a: Find authors who have published at least 1 paper every year in the last 30 years, and whose family name start with ‘H’.

```
WITH RecentPublications AS (
    SELECT pubid, pubyear
    FROM publication
    WHERE pubyear >= (SELECT YEAR(CURDATE()) - 29)
),
FilteredAuthors AS (
    SELECT author.authorid, author.authorName
    FROM author
    WHERE SUBSTRING_INDEX(author.authorName, ' ', -1) LIKE 'H%'
)
SELECT
    fa.authorName AS author
FROM
    FilteredAuthors fa
INNER JOIN
    authored au ON fa.authorid = au.authorid
INNER JOIN
    RecentPublications rp ON rp.pubid = au.pubid
GROUP BY
    fa.authorName
HAVING
    COUNT(DISTINCT rp.pubyear) = 30;
```

author
Alain Hertz
Alan R. Hevner
Ali R. Hurson
Amir Herzberg
Andreas Henrich
Arthur H. M. ter Hofstede
Blake Hannaford
Boudewijn R. Haverkort
Chu-Ren Huang
Chung-Ming Huang
Dan Halperin
David C. Hogg
David Harel

Query 7a Answer

Query 7b: Find the names and number of publications for authors who have the earliest publication record in DBLP.

```
SELECT A.authorid, A.authorname, COUNT(*)
FROM author A JOIN authored AP
ON A.authorid = AP.authorid
WHERE A.authorid IN (
SELECT DISTINCT AP.authorid
FROM authored AP JOIN publication P ON AP.pubid = P.pubid
WHERE P.pubyear = (SELECT MIN(pubyear) FROM publication)
)
GROUP BY A.authorid , A.authorname;
```

authorid	authorname	COUNT(*)
562355	J. Barkley Rosser	18
562477	Alonzo Church	6
562655	Arnold F. Emch	3
435189	Willard Van Orman Quine	29
563274	C. I. Lewis	1
562316	Frederic Brenton Fitch	30
562798	Curt John Ducasse	4
562775	Emil L. Post	3

Query 7b Answer

Query 8: Return the top 5 most common first name of the author that published in "US" in the last 2 year.

```
SELECT
  SUBSTRING_INDEX(a.authorname, ' ', 1) AS first_name,
  COUNT(*) AS publication_count
FROM
  author a
JOIN
  authored au ON a.authorid = au.authorid
JOIN
  publication p ON au.pubid = p.pubid
WHERE
  p.title LIKE '%US%'
  AND p.pubyear >= YEAR(CURDATE()) - 2
GROUP BY
  first_name
ORDER BY
  publication_count DESC
LIMIT 5;
```

first_name	publication_count
David	5619
Michael	5356
Wei	5287
Muhammad	4794
Daniel	4706

Query 8 Answer

3.2. Query Running Time on Halved and Quartered Database

We have created both half-size and quarter-size databases from the original full-size database in MySQL. This process began with the publication table, where we reduced its size by selecting only entries in odd positions. This approach ensures that the data points remain homogeneous and comparable to the full-size database. With the half size publication table, we reference back the records to the existing full size author and authored tables to generate corresponding half-size versions. This involved querying for the relevant author IDs and publication IDs from the half size publication table. The statistic count based on the record of the full size, half size and quarter size database is presented in Table 1. A similar strategy was applied to create the quarter-size database. The source code for these operations is provided in Annex Section C.

Table/Relation	Full Size Database	Half Size Database	Quarter Size Database
Publication Count	6,800,449	3,400,225	1,700,113
Author Count	3,425,121	2,421,974	1,659,728
Authorod Count	22,236,310	11,117,154	5,556,930

Table 1. Database Count for different sizes of database

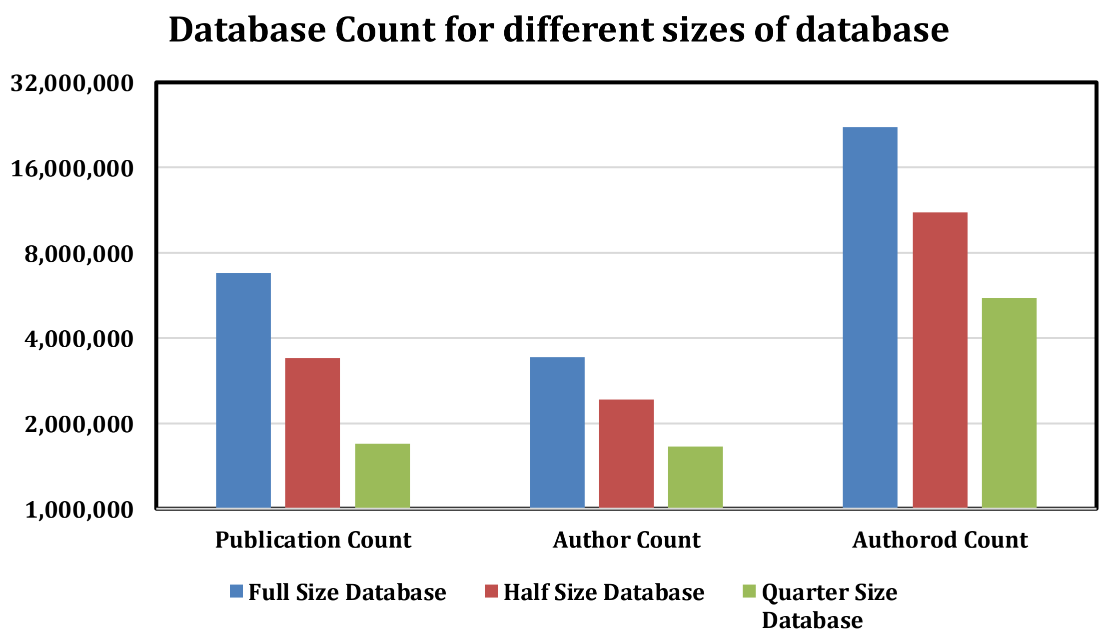

Figure 4. Database Count for different sizes of database and relations represented in Bar Charts

The running times for processing the full-size database, its half-size equivalent, and the quarter-size variant are detailed in Table 2 and illustrated in Figure 5. below. As the dataset was approximately halved, we observed that most queries performed up to twice as fast in the half-size database and showed a similar degree of improvement when the dataset was further reduced to a quarter of its original size. This enhancement in performance can likely be attributed to the diminished volume of data requiring processing. With fewer rows to scan, join, or sort, the system can execute operations more rapidly. Furthermore, the smaller dataset size enables more efficient utilization of the system's cache mechanisms, leading to expedited data access times.

Query	Full Size Database Runtime (ms)	Half Size Database Runtime (ms)	Quarter Size Database Runtime (ms)	Full/Half Database	Half/Quarter Database
1	28,484	7,750	4,672	3.7	1.7
2	33,062	8,813	3,172	3.8	2.8
3	44,921	9,000	5,250	4.9	1.7
4	112,500	52,547	26,219	2.1	2.0
5	57,890	24,000	12,672	2.4	1.9
6	48,250	9,938	5,265	4.9	1.9
7a	332,797	169,469	84,547	2.0	2.0
7b	44,047	7,250	5,171	6.1	1.4
8	64,594	36,406	18,281	1.8	2

Table 2. Queries runtime on different sizes of database

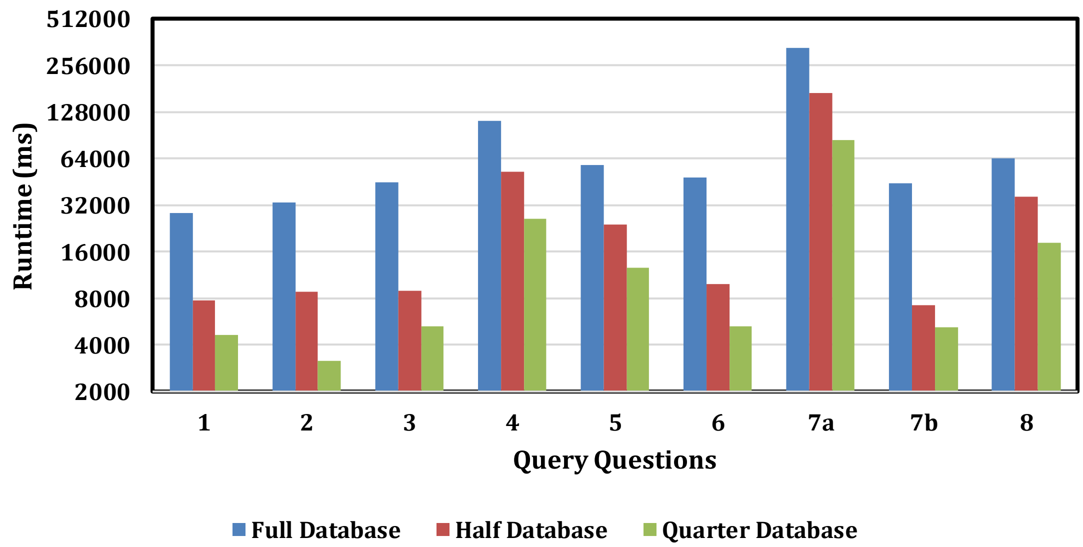

Figure 5. Runtime (ms) against Query Questions

4. Building an Index and Studying the Effect of the Index

In MySQL, primary keys and corresponding foreign keys are automatically indexed using the B+ Tree index structure. This section delves into the expansion of B+ Tree indexing to other attributes within the Publication and Author tables, to assess whether this can accelerate specific query performances. Our database uses the InnoDB storage engine, which supports B+ Trees, but do not support hashing index as it is used internally for its adaptive hash index feature.Additionally, we explore the use of composite indexes, also known as multi-column indexes. These indexes concatenate multiple columns into a single index, allowing for the storage of sorted pointers to other columns. This multi-column approach can significantly speed up data access, as it facilitates both sorting and searching across several columns, as depicted in Figure 6. Composite indexes, by leveraging multiple columns, can thus enhance the efficiency of B+ Tree indexing methods (Steven Li, 2017).

Although indexes are able to speed up queries if used appropriately, the performance gains comes with additiinoal storage space to store these indexes and indexes have to been updated when records are inserted, deleted and modified.

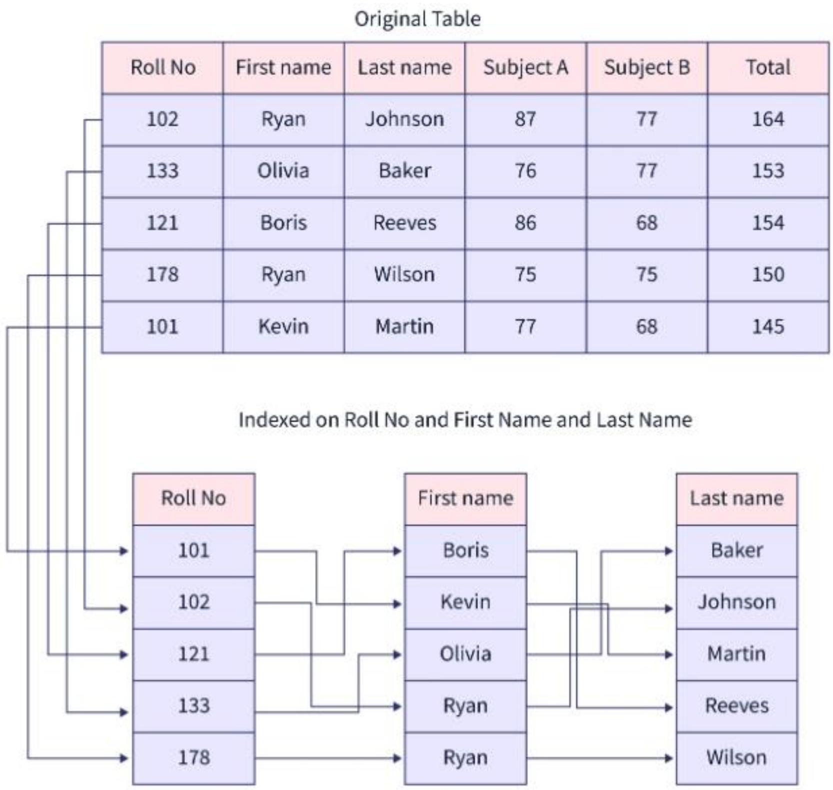

Figure 6. Composite Index referencing example (Santanu Baruah, 2023)

In this study, we evaluated the performance of database queries on a full-sized database comprises of Publications, Authors, and Authored tables based on the 8 querry questions we have addressed in the eariler section.Our approach is to use single column index and composite index. We evaluated the performance of the default index, which utilizes only the primary key as the index (in our case, ‘pubid’ for the publication table and ’authorid’ for the author table), and compared it with the runtime using single-column and composite indexes. In each query, We could also also use "EXPLAIN" statement to access the query plan, which provides insights into how the operations are executed and whether the indexes are being effectively used in the plan.

Query 1

Default Index (Publication and Author Relations Pri keys being Indexed)

Query Plan - Default Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	11.11	Using where; Using temporary; Using filesort

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx1_publication ON publication (pubkey);
CREATE INDEX idx1_publication ON publication (pubyear);
CREATE INDEX idx1_publication ON publication (pubtype);
```

Query Plan - Single column Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	index	idx11_publication,idx12_publication	idx12_publication	1023	NULL	6658672	50.00	Using where; Using temporary; Using filesort

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx1_publication ON publication (pubkey, pubtype, pubyear);
```

Query Plan - Composite Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	index	idx1_publication	idx1_publication	2051	NULL	6658672	11.11	Using where; Using index; Using temporary; Using filesort

In query 1, in addition to the default B+ tree index on primary keys, we have experimented with both single-column and composite indexes. The outcomes, as presented in Table 3, indicate a notable increase in query runtime when utilizing a single-column index, with 'pubtype' being the only index used. Conversely, the implementation of a composite index has led to a substantial decrease in query run time. This improvement is likely attributable to the composite index's ability to group relevant attributes together, thus enabling quicker searches and reducing I/O costs.

Query 2

Query Plan - Default Index (Publication and Author Relations Pri keys being Indexed)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	665867	100.00	Using temporary
2	DERIVED	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	10.00	Using where; Using temporary

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx2_publication ON publication (pubkey1);
CREATE INDEX idx21_publication ON publication (pubkey2);
CREATE INDEX idx22_publication ON publication (pubyear);
```

Query Plan – Single Column Index

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	3329336	100.00	Using temporary
	2	DERIVED	publication	NULL	ALL	idx2_publication	NULL	NULL	NULL	6658672	50.00	Using where; Using temporary

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx2_publication ON publication (pubkey1, pubkey2, pubyear);
```

Query Plan – Composite Index

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	3329336	100.00	Using temporary
	2	DERIVED	publication	NULL	ref	idx2_publication	idx2_publication	1023	const	3329336	100.00	Using index

In query 2, we see that both single-column and composite indexes have reduces the query runtime, as shown in Table 3. The 'filtered' metric in the query plan is higher for these indexes which imply that the indexes have help to narrow down the records tobe accessed, reducing the I/O cost. Especially the composite index, where the query runtime is reduced significantly.

Query 3

Query Plan – Default Index (Publication and Author Relations Pri keys being Indexed)

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	6658686	100.00	Using filesort
	2	DERIVED	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	10.00	Using where; Using temporary
	2	DERIVED	<derived4>	NULL	ref	<auto_key0>	<auto_key0>	5	publicationdb.publication.pubyear	10	100.00	Using index
	4	DERIVED	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	33.33	Using where; Using temporary

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx3_publication ON publication (pubkey1);
CREATE INDEX idx31_publication ON publication (pubyear);
```

Query Plan – Single Column Index

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	1687471	100.00	Using filesort
	2	DERIVED	<derived4>	NULL	ALL	NULL	NULL	NULL	NULL	37	100.00	Using where; Using temporary
	2	DERIVED	publication	NULL	ref	idx3_publication,idx31_publication	idx31_publication	5	pubyear_range.pubyear	91214	50.00	Using where
	4	DERIVED	publication	NULL	range	idx31_publication	idx31_publication	5	NULL	37	100.00	Using where; Using index for group-by

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx3_publication ON publication (pubkey1, pubyear);
```

Query Plan – Composite Index

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	33293435	100.00	Using filesort
	2	DERIVED	publication	NULL	ref	idx3_publication	idx3_publication	1023	const	3329336	100.00	Using where; Using index; Using temporary
	2	DERIVED	<derived4>	NULL	ref	<auto_key0>	<auto_key0>	5	publicationdb.publication.pubyear	10	100.00	Using index
	4	DERIVED	publication	NULL	index	idx3_publication	idx3_publication	1028	NULL	6658672	33.33	Using where; Using index; Using temporary

In query 3, the runtime with the default index is shorter than that with the single-column index but longer than with the composite index as shown in Table 3. While both the single-column and composite indexes use the 'pubkey' index in the 'where' clause and 'pubyear' in the 'group by', the runtime increases substantially with the single-column index, but decreases significantly with the composite index. This suggests that sometimes the complexity of a query can increase with more indexes, causing additional overhead and leading to longer runtimes.

Query 4

Query Plan - Default Index (Publication and Author Relations Pri keys being Indexed)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY	NULL	NULL	NULL	6658672	19.00	Using where; Using temporary; Using filesort
1	SIMPLE	pa1	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.p.pubid	3	100.00	Using where
1	SIMPLE	a	NULL	eq_ref	PRIMARY	PRIMARY	4	publicationdb.pa1.authorid	1	100.00	NULL
1	SIMPLE	pa2	NULL	ref	fk_authored_publication	fk_authored_publication	5	publicationdb.p.pubid	3	90.00	Using where

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx4_publication ON publication(pubkey1);
CREATE INDEX idx4_author ON author(authorname);
```

Query Plan - Single Column Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY,idx4_publication	NULL	NULL	NULL	6658672	100.00	Using where; Using temporary; Using filesort
1	SIMPLE	pa1	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.p.pubid	3	100.00	Using where
1	SIMPLE	a	NULL	eq_ref	PRIMARY,idx4_author	PRIMARY	4	publicationdb.pa1.authorid	1	100.00	NULL
1	SIMPLE	pa2	NULL	ref	fk_authored_publication	fk_authored_publication	5	publicationdb.p.pubid	3	90.00	Using where

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx4_publication ON publication(pubid, pubkey1);
CREATE INDEX idx4_author ON author(authorid, authorname);
```

Query Plan - Composite Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY,idx4_publication	NULL	NULL	NULL	6658672	19.00	Using where; Using temporary; Using filesort
1	SIMPLE	pa1	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.p.pubid	3	100.00	Using where
1	SIMPLE	a	NULL	eq_ref	PRIMARY,idx4_author	PRIMARY	4	publicationdb.pa1.authorid	1	100.00	NULL
1	SIMPLE	pa2	NULL	ref	fk_authored_publication	fk_authored_publication	5	publicationdb.p.pubid	3	90.00	Using where

In Query 4, we found that the default index outperformed both the single-column and composite indexes in terms of runtime, as shown in Table 3. According to the query plan, only 'pubid' and 'authorid' were used, whereas the attributes 'authorname' and 'pubkey1' remained unutilised, leading to higher overhead and memory waste, which can slow down the runtime of the query.

Query 5

Query Plan - Default Index (Publication and Author Relations Pri keys being Indexed)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	ALL	PRIMARY	NULL	NULL	NULL	6658672	0.70	Using where; Using temporary; Using filesort
1	SIMPLE	authored	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.publication.pubid	3	100.00	Using where
1	SIMPLE	author	NULL	eq_ref	PRIMARY	PRIMARY	4	publicationdb.authored.authorid	1	100.00	NULL

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx5_publication ON publication(pubkey1);
CREATE INDEX idx51_publication ON publication(pubyear);
CREATE INDEX idx5_author ON author(authorname);
```

Query Plan - Single Column Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	ALL	PRIMARY,idx5_publication,idx51_pu...	NULL	NULL	NULL	6658672	5.56	Using where; Using temporary; Using filesort
1	SIMPLE	authored	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.publication.pubid	3	100.00	Using where
1	SIMPLE	author	NULL	eq_ref	PRIMARY,idx5_author	PRIMARY	4	publicationdb.authored.authorid	1	100.00	NULL

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx5_publication ON publication(pubid, pubkey1, pubyear);
```

CREATE INDEX idx5_author ON author(authorID, authorname);

Query Plan - Composite Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	ALL	PRIMARY,idx5_publication	NULL	NULL	NULL	6658672	0.70	Using where; Using temporary; Using filesort
1	SIMPLE	authored	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.publication.pubid	3	100.00	Using where
1	SIMPLE	author	NULL	eq_ref	PRIMARY,idx5_author	PRIMARY	4	publicationdb.authored.authorid	1	100.00	NULL

In query 5, the situation were similar to query 4, where the query plan for the default, single-column index, and composite index only utilise the 'pubid' and 'authorid'. The indexes on other attributes remained unused. Consequently, the introduction of these additional indexes has led to an increase in the complexity of the queries without yielding a significant improvement in query runtime for both single-column and composite index scenarios.

Query 6

Query Plan - Default Index (Publication and Author Relations Pri keys being Indexed)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	100.00	Using where; Using temporary; Using filesort
1	SIMPLE	<subquery2>	NULL	eq_ref	<auto_distinct_key>	<auto_distinct_key>	1023	publicationdb.publication.pubkey2	1	100.00	NULL
2	MATERIALIZED	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	0.12	Using where

Single Column Index (inclusive of publication and author Pri keys being Index)

CREATE INDEX idx6_publication ON publication(pubyear);

CREATE INDEX idx61_publication ON publication(pubtype);

CREATE INDEX idx62_publication ON publication(pubkey2);

Query Plan - Single Column Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	ALL	idx62_publication	NULL	NULL	NULL	6658672	100.00	Using where; Using temporary; Using filesort
1	SIMPLE	<subquery2>	NULL	eq_ref	<auto_distinct_key>	<auto_distinct_key>	1023	publicationdb.publication.pubkey2	1	100.00	NULL
2	MATERIALIZED	publication	NULL	ALL	idx61_publication,idx62_publication	NULL	NULL	NULL	6658672	0.62	Using where

Composite Index (inclusive of publication and author Pri keys being Index)

CREATE INDEX idx6_publication ON publication(pubyear, pubtype, pubkey2);

Query Plan - Composite Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	index	idx6_publication	idx6_publication	2051	NULL	6658672	100.00	Using where; Using index; Using temporary; Using filesort
1	SIMPLE	<subquery2>	NULL	eq_ref	<auto_distinct_key>	<auto_distinct_key>	1023	publicationdb.publication.pubkey2	1	100.00	NULL
2	MATERIALIZED	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	0.12	Using where

In Query 6, as illustrated in Table 3, the runtime with the default index is slightly shorter than that with the single-column index since similar indexes are employed in both query plans. However, the composite index scenario reveals a different outcome. The composite index, 'idx6_publication,' is leveraged in the 'where' clause, which is not observed in the default or single-column index scenarios. Hence this contributes to a faster query runtime by streamlining the search process and thereby reducing the I/O cost.

Query 7a

Query Plan - Default Index (Publication and Author Relations Pri keys being Indexed)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	ALL	PRIMARY	NULL	NULL	NULL	6658672	33.33	Using where; Using temporary; Using filesort
1	SIMPLE	au	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.publication.pubid	3	100.00	Using where
1	SIMPLE	author	NULL	eq_ref	PRIMARY	PRIMARY	4	publicationdb.au.authorid	1	100.00	Using where

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx7a_publication ON publication(pubyear);
CREATE INDEX idx7a_author ON author(authorname);
```

Query Plan - Single Column Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	range	PRIMARY,idx7a_publication	idx7a_publication	5	NULL	3329336	100.00	Using where; Using index; Using temporary; Using filesort
1	SIMPLE	au	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.publication.pubid	3	100.00	Using where
1	SIMPLE	author	NULL	eq_ref	PRIMARY,idx7a_author	PRIMARY	4	publicationdb.au.authorid	1	100.00	Using where

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx7a_publication ON publication(pubid, pubyear);
CREATE INDEX idx7a_author ON author(authorid, authorname);
```

Query Plan - Composite Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	publication	NULL	index	PRIMARY,idx7a_publication	idx7a_publication	9	NULL	6658672	33.33	Using where; Using index; Using temporary; Using filesort
1	SIMPLE	au	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.publication.pubid	3	100.00	Using where
1	SIMPLE	author	NULL	eq_ref	PRIMARY,idx7a_author	PRIMARY	4	publicationdb.au.authorid	1	100.00	Using where

In Query 7a, the runtime using the default index is significantly lower than that of the single-column index but only slightly higher than the composite index. The query plan indicates that for the single-column index, a range type search occurs within the 'publication' table, which likely requires a sort on 'idx7a_publication' based on 'authorname', thereby increasing I/O costs and resulting in a higher runtime. In contrast, for the composite index, the query plan shows an additional index usage within the 'publication' table, leading to a more efficient search and a lower runtime compared to the default index.

Query 7b

Query Plan - Default Index (Publication and Author Relations Pri keys being Indexed)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<subquery2>	NULL	ALL	NULL	NULL	NULL	NULL	NULL	100.00	Using where; Using temporary
1	PRIMARY	A	NULL	eq_ref	PRIMARY	PRIMARY	4	<subquery2>.authorid	1	100.00	NULL
1	PRIMARY	AP	NULL	ref	fk_authored_author	fk_authored_author	5	<subquery2>.authorid	7	100.00	Using index
2	MATERIALIZED	P	NULL	ALL	PRIMARY	NULL	NULL	NULL	6658672	10.00	Using where
2	MATERIALIZED	AP	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.P.pubid	3	100.00	NULL
3	SUBQUERY	publication	NULL	ALL	NULL	NULL	NULL	NULL	6658672	100.00	NULL

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx7b_publication ON publication(pubyear);
CREATE INDEX idx7b_author ON author(authorname);
```

Query Plan - Single Column Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<subquery2>	NULL	ALL	NULL	NULL	NULL	NULL	NULL	100.00	Using where; Using temporary
1	PRIMARY	A	NULL	eq_ref	PRIMARY,idx7b_author	PRIMARY	4	<subquery2>.authorid	1	100.00	NULL
1	PRIMARY	AP	NULL	ref	fk_authored_author	fk_authored_author	5	<subquery2>.authorid	7	100.00	Using index
2	MATERIALIZED	P	NULL	ref	PRIMARY,idx7b_publication	idx7b_publication	5	const	12	100.00	Using where; Using index
2	MATERIALIZED	AP	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.P.pubid	3	100.00	NULL
3	SUBQUERY	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Select tables optimized away

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx7b_publication ON publication(pubid, pubyear);
CREATE INDEX idx7b_author ON author(authorid, authorname);
```

Query Plan - Composite Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<subquery2>	NULL	ALL	NULL	NULL	NULL	NULL	NULL	100.00	Using where; Using temporary
1	PRIMARY	A	NULL	eq_ref	PRIMARY,idx7b_author	PRIMARY	4	<subquery2>.authorid	1	100.00	NULL
1	PRIMARY	AP	NULL	ref	fk_authored_author	fk_authored_author	5	<subquery2>.authorid	7	100.00	Using index
2	MATERIALIZED	P	NULL	index	PRIMARY,idx7b_publication	idx7b_publication	9	NULL	6658672	10.00	Using where; Using index
2	MATERIALIZED	AP	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.P.pubid	3	100.00	NULL
3	SUBQUERY	publication	NULL	index	NULL	idx7b_publication	9	NULL	6658672	100.00	Using index

In Query 7, the runtime when using the default index is significantly higher as compared to both the single-column and composite index. The query plans for the single-column and composite indexes demonstrate that indexes on 'pubyear' and 'authorname' are being utilised, thus reduces the I/O cost. The composite index, which combines multiple attributes, may introduce an unnecessary level of complexity for this particular query, leading to a runtime that is longer than that of the single-column index.

Query 8

Query Plan - Default Index (Publication and Author Relations Pri keys being Indexed)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY	NULL	NULL	NULL	6658672	3.70	Using where; Using temporary; Using filesort
1	SIMPLE	au	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.p.pubid	3	100.00	Using where
1	SIMPLE	a	NULL	eq_ref	PRIMARY	PRIMARY	4	publicationdb.au.authorid	1	100.00	NULL

Single Column Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx8_publication ON publication(pubyear);
CREATE INDEX idx8_author ON author(authorname);
```

Query Plan - Single Column Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY,idx8_publication	NULL	NULL	NULL	6658672	4.32	Using where; Using temporary; Using filesort
1	SIMPLE	au	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.p.pubid	3	100.00	Using where
1	SIMPLE	a	NULL	eq_ref	PRIMARY,idx8_author	PRIMARY	4	publicationdb.au.authorid	1	100.00	NULL

Composite Index (inclusive of publication and author Pri keys being Index)

```
CREATE INDEX idx8_publication ON publication(pubid, pubyear);
CREATE INDEX idx8_author ON author(authorid, authorname);
```

Query Plan - Composite Index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY,idx8_publication	NULL	NULL	NULL	6658672	3.70	Using where; Using temporary; Using filesort
1	SIMPLE	au	NULL	ref	fk_authored_author,fk_authored_p...	fk_authored_publication	5	publicationdb.p.pubid	3	100.00	Using where
1	SIMPLE	a	NULL	eq_ref	PRIMARY,idx8_author	PRIMARY	4	publicationdb.au.authorid	1	100.00	NULL

In Query 8, similar to the findings from Query 4, the runtime with the default index is shorter than the runtimes when using either the single-column or composite indexes, as shown in Table 3. This outcome is attributed to the fact that the imposed indexes are not leveraged in the query plan, rendering them redundant. The inclusion of these non-essential indexes thus only adds unnecessary complexity and results in higher I/O costs.

Queries	Default (With Primary Key as Index) (ms)	Single Column Index (ms)	With Composite Index (ms)
1	28,484	51,922	12,031
2	33,062	26,203	1,578
3	44,921	293,610	6,281
4	112,500	>200,000	119,750
5	57,890	57,718	58,812
6	33,672	35,734	27,172
7a	316,250	826,985	300,250
7b	44,047	16	5,953
8	72,562	73,719	79,594

Table 3. Queries runtime on full sized database (a) without index, (b) with single column index, (c) with composite index

5. Conclusion

In this project, we have successfully extracted, cleaned, and filtered the DBLP dataset, resolving any data anomalies and seamlessly integrating it into a SQL server specifically using MySQL. We have adeptly tackled all eight query questions posed. Furthermore, we divided the database into two parts: half and one quarter by selecting records at odd positions to ensure data homogeneity. We also explored indexing on the DBLP dataset in MySQL, focusing on B+ Trees. Additionally, we exam ined the implementation of single-column indexes based on individual attributes an d composite indexes that encompass multiple attributes through concatenation. The performance implications of each indexing strategy were analysed by reviewing their respective query plans.

6. Reference

Leonswl. (2023). *DBLP-SAX-PARSER* . Github. https://github.com/leonswl/dblp-sax-parser

Santanu Baruah. (2023). *Composite Index in SQL*. Scaler Topics.

Steven Li. (2017). *Single vs Composite Indexes in Relational Databases*. Medium. https://user3141592.medium.com/single-vs-composite-indexes-in-relational-databases-58d0eb045cbe

UNIVERSITAT TRIER, S. D. (n.d.). *Computer Science Bibliography, DBLP*. Retrieved November 24, 2023, from https://dblp.org/

ANNEX

Section A: Source Code for DBLP SAX PARSER

```python
# Import the necessary packages
import os
import xml.sax
import gzip
import re
import pandas as pd

# Set the working directory to where the DBLP dataset is located
os.chdir('C:/Users/geneh/Desktop/DBLP')

# Define a SAX parser class for the DBLP dataset
class DBLP_Parser(xml.sax.ContentHandler):

    def __init__(self):
        # Initialize variables for XML parsing
        self.path = []  # Stack to track the current XML path
        self.text = []  # Accumulate text data within XML tags
        self.row = {}   # Temporary storage for data of a single entry
        self.articles = [] # List to store all parsed articles
        self.record_count = 0 # Counter for processed records

        # Define valid elements and their respective data types
        self.valid_elements = {"article", "inproceedings", "proceedings", "book", "incolle
ction", "phdthesis", "mastersthesis", "www", "person", "data"}
        self.element_features = {
            "address": "str", "author": "list", "booktitle": "str", "cdrom": "str",
            "chapter": "str", "cite": "list", "crossref": "str", "editor": "list",
            "ee": "list", "isbn": "str", "journal": "str", "month": "str",
            "note": "str", "number": "str", "pages": "str", "publisher": "str",
            "publnr": "str", "school": "str", "series": "str", "title": "str",
            "url": "str", "volume": "str", "year": "str"
        }

    # Function to calculate the total number of pages from a page range
    def page_counter(self, page_info):
        total_pages = 0
        for part in re.split(r",", page_info):
            sections = re.split(r"-", part)
            if len(sections) > 2:
                continue
            try:
                sections = [int(re.findall(r"[\d]+", sec)[-1]) for sec in sections]
            except IndexError:
                continue
            total_pages += 1 if len(sections) == 1 else sections[1] - sections[0] + 1
        return str(total_pages) if total_pages != 0 else ""
```

```
    # Called by SAX parser when it encounters the start of an element
    def startElement(self, tag, attributes):
        self.path.append(tag)

        # Initialize a new row when encountering a valid publication element
        if tag in self.valid_elements:
            self.row['mdate'] = attributes.get('mdate')
            self.row['publtype'] = attributes.get('publtype')
            self.row['key'] = attributes.get('key')
            self.row['type'] = tag  # Add publication type

        # Print progress every 10 million records
        self.record_count += 1
        if self.record_count % 10000000 == 0:
            print(f'Processed {self.record_count} records so far.')

     # Called by SAX parser for character data within elements
    def characters(self, content):
        self.text.append(content.strip())

    # Called by SAX parser when it encounters the end of an element
    def endElement(self, tag):
        full_text = " ".join(self.text).strip()  # Combine accumulated text

        # Process and store the text data based on the tag type
        if tag in self.element_features:
            if self.element_features[tag] == "str":
                if tag == "pages":
                    full_text = self.page_counter(full_text)
                self.row[tag] = full_text
            elif self.element_features[tag] == "list":
                self.row.setdefault(tag, []).append(full_text)

        # Finalise and store the row when the end of a valid element is reached
        if tag in self.valid_elements:
            self.articles.append(self.row.copy())
            self.row.clear()

        # Announce completion when the end of the document is reached
        if tag == 'dblp':
            print("Completed parsing XML. All entries processed.")

        # Reset the text accumulator and update the path
        self.text.clear()
        self.path.pop()

# Instantiate the parser and set the content handler
parser = xml.sax.make_parser()
handler = DBLP_Parser()
parser.setContentHandler(handler)
with gzip.open('dblp.xml.gz', 'rt') as f:
    parser.parse(f)
```

```
# Convert parsed data into a DataFrame and save it as CSV
dblp = pd.DataFrame(handler.articles)
dblp.to_csv('dblp.csv')

#Handle irregularies in dblp.csv and removal of type = 'data' , 'mastersthesis' , phdthes
is' , 'www' which care not relevant to our queries
dblp = pd.read_csv('dblp.csv',index_col=0,header=0)
types_to_remove = ['data', 'mastersthesis', 'phdthesis', 'www']
dblp = dblp[~dblp['type'].isin(types_to_remove)]

years = dblp['year']
for i in range(len(years)):
    tempyear = years[i]
    if isinstance(tempyear,str):
        dblp.loc[i,'year'] = tempyear.replace(' ','') # for issues like '2 015'
    else:
        if np.isnan(tempyear):
            dblp.loc[i,'year'] = 'NaN' # for values with 'np.nan'
del years
del tempyear
dblp.to_csv('dblp.csv',header=True,index=True)

#Creation of Author Table
# Assume author_df is your original DataFrame with an 'author' column containing lists of
authors
df.reset_index(inplace=True)
df.rename(columns={'index': 'pubid'}, inplace=True)
author_df = df[['pubid', 'author', 'title']]
author_df_exploded = author_df.explode('author').reset_index(drop=True)
# Now, since 'author' contains individual authors per row, we can get unique authors
authors = author_df_exploded['author'].unique()
# Create a mapping of authors to unique IDs
author_mapping = {author: i+1 for i, author in enumerate(authors)}
# Map the authors in the exploded DataFrame to their unique IDs
author_df_exploded['AuthorID'] = author_df_exploded['author'].map(author_mapping)
author_df_exploded.to_csv('author.csv')


#Creation of Authored Table - Linking Author Table to Publication Table

# Loading the previously saved author data
author = pd.read_csv('author.csv',index_col=0,header=0)
# Sorting by 'AuthorID' and removing any duplicate entries
author.sort_values('AuthorID',inplace=True)
author.drop_duplicates(subset='AuthorID',inplace=True)
# Extracting a table of unique authors and their corresponding IDs
unique_author = author.loc[:,['author','AuthorID']]
# Saving this unique author table, which can be used to link authors to publications
unique_author.to_csv('unique_author.csv',index=True,header=True)
```

Section B: Source Code for Table Creation, Modification and Bulkloading

```
-- Create a new database named publicationDB
CREATE DATABASE publicationDB;
-- Select the newly created database for use
USE publicationDB;

-- Set the GLOBAL local_infile variable to true to allow for local data loading
SET GLOBAL local_infile = true;

-- Drop any existing tables named authored, publication, and author if they exist to
avoid conflicts
DROP TABLE IF EXISTS authored;
DROP TABLE IF EXISTS publication;
DROP TABLE IF EXISTS author;

-- Create a new table named Publication with specified columns and data types
CREATE TABLE Publication(
	pubid INT NOT NULL,
	pubkey VARCHAR(255),
	pubtype VARCHAR(255),
	title VARCHAR(10000),
	pubyear INT,
	PRIMARY KEY (pubid) -- Sets pubid as the primary key for the table
);

-- Load data into the Publication table from a CSV file located at the given path
-- Note: The LOAD DATA INFILE path should be changed to the actual path where the dbl
p.csv file is located.
LOAD DATA INFILE 'D://developer_tools//MySQL//MySQL Server 8.0//Uploads//dblp.csv'
INTO TABLE Publication
CHARACTER SET latin1 -- Specifies the character set for the data
FIELDS TERMINATED BY ',' -- Specifies that fields are terminated by commas
OPTIONALLY ENCLOSED BY '"' -- Fields are optionally enclosed by double quotes
LINES TERMINATED BY '\n' -- Specifies that lines are terminated by newlines
IGNORE 1 ROWS — Ignores the first row, which often contains column headers
(
	-- Temporary variables for each column in the CSV file
	-- The number of @col variables should match the number of columns in your CSV
file
	@col1, @col2, @col3, @col4, @col5, @col6, @col7, @col8, @col9, @col10,
	@col11, @col12, @col13, @col14, @col15, @col16, @col17, @col18, @col19,
	@col20, @col21, @col22, @col23, @col24, @col25, @col26, @col27,@col28
)
-- Set the actual columns in Publication table based on the temporary variables from
the CSV file
set pubid=@col1,pubkey=@col4,pubtype=@col5,title=@col7,pubyear=IF(@col9 = 'NaN',NULL,
REPLACE(@col9,' ',''));
```

```sql
-- Modify the Publication table to add additional columns for split pubkey values
ALTER TABLE publication
ADD COLUMN pubkey1 VARCHAR(255),
ADD COLUMN pubkey2 VARCHAR(255),
ADD COLUMN pubkey3 VARCHAR(255);

-- Disable safe updates to allow updates without specifying a WHERE clause
SET sql_safe_updates=0;
-- Update the Publication table to split pubkey into three separate columns based on
'/'
UPDATE publication
SET
     pubkey1 = SUBSTRING_INDEX(pubkey, '/', 1),
     pubkey2 = SUBSTRING_INDEX(SUBSTRING_INDEX(pubkey, '/', 2), '/', -1),
     pubkey3 = SUBSTRING_INDEX(pubkey, '/', -1);

-- Create a new Author table with authorid and authorname as columns
CREATE TABLE Author(
      authorid INT,
      authorname VARCHAR(255),
      PRIMARY KEY (authorid) -- Sets authorid as the primary key for the table
);

-- Load data into the Author table from a CSV file located at the given path
-- Note: The LOAD DATA INFILE path should be changed to the actual path where the uni
que_author.csv file is located.
LOAD DATA INFILE 'D://developer_tools//MySQL//MySQL Server 8.0//Uploads//unique_autho
r.csv'
INTO TABLE Author
CHARACTER SET latin1 -- Specifies the character set for the data
FIELDS TERMINATED BY ',' -- Specifies that fields are terminated by commas
OPTIONALLY ENCLOSED BY '"' -- Fields are optionally enclosed by double quotes
LINES TERMINATED BY '\n' -- Specifies that lines are terminated by newlines
IGNORE 1 ROWS -- Ignores the first row as it often contains column headers
(
      -- Temporary variables for each column in the CSV file
      @col1, @col2, @col3
)
-- Set the actual columns in Author table based on the temporary variables from the C
SV file
SET authorid=@col3,authorname=@col2;

-- Update a specific record in Author table where authorid is 11 to 'No Author'
UPDATE author
SET authorname = 'No Author'
WHERE authorid = 11;
```

```sql
-- Create a new Authored table to represent a many-to-many relationship between Authors and Publications
CREATE TABLE Authored(
	authorid INT,
	pubid INT,
	FOREIGN KEY (authorid) REFERENCES Author(authorid), -- Defines a foreign key relationship with Author table
	FOREIGN KEY (pubid) REFERENCES Publication(pubid) -- Defines a foreign key relationship with Publication table
);

-- Temporarily disable foreign key checks to allow loading data without constraint checks
SET FOREIGN_KEY_CHECKS=0;

-- Load data into the Authored table from a CSV file located at the given path
-- Note: The LOAD DATA INFILE path should be changed to the actual path where the author.csv file is located.
LOAD DATA INFILE 'D://developer_tools//MySQL//MySQL Server 8.0//Uploads//author.csv'
INTO TABLE Authored
CHARACTER SET latin1 -- Specifies the character set for the data
FIELDS TERMINATED BY ',' -- Specifies that fields are terminated by commas
OPTIONALLY ENCLOSED BY '"' -- Fields are optionally enclosed by double quotes
LINES TERMINATED BY '\n' -- Specifies that lines are terminated by newlines
IGNORE 1 ROWS — Ignores the first row as it often contains column headers
(
	-- Temporary variables for each column in the CSV file
	@col1, @col2, @col3,@col4,@col5
)
-- Set the actual columns in Authored table based on the temporary variables from the CSV file
set authorid=@col5,pubid=@col2;

-- Re-enable foreign key checks after loading data into Authored table
SET FOREIGN_KEY_CHECKS=1;
```

Section C: Source Code for Half Size and Quarter Size Publication, Author and Authored Tables

```
-- Half Publication, Author and Authored Tables
-- Switch to the publicationdb database
USE publicationdb;

-- Create a new database named halfDB
CREATE DATABASE halfDB;

-- Create a view that selects only the odd rows from the publication table
CREATE VIEW half_publication_view AS
SELECT pubid, pubkey, pubtype, title, pubyear, pubkey1, pubkey2, pubkey3 FROM (
      SELECT *, row_number() OVER (ORDER BY pubid) AS row_num
      FROM publication
) AS temp
WHERE row_num % 2 = 1;

-- Create a new table in halfDB database using the data from the half_publication_vie
w
CREATE TABLE halfDB.half_publication AS
SELECT *
FROM publicationDB.half_publication_view;

-- Create a view that joins the authored table with the half_publication table based
on pubid
CREATE VIEW half_authored_view AS
SELECT au.*
FROM authored au
INNER JOIN halfDB.half_publication p ON au.pubid = p.pubid;

-- Create a new table in halfDB database using the data from the half_authored_view
CREATE TABLE halfDB.half_authored AS
SELECT *
FROM publicationDB.half_authored_view;

-- Create an index on the half_authored table for the authorid column in halfDB to im
prove query performance
CREATE INDEX idx_halfauthored_authorid ON halfDB.half_authored(authorid);

-- Create a view that selects distinct authors who have authored publications in the
half_authored table
CREATE VIEW half_author_view AS
SELECT DISTINCT a.*
FROM author a
INNER JOIN halfDB.half_authored r ON a.authorid = r.authorid;

-- Create a new table in halfDB database using the data from the half_author_view
```

```sql
CREATE TABLE halfDB.half_author AS
SELECT *
FROM publicationDB.half_author_view;

-- Switch to the halfDB database for subsequent operations
USE halfDB;

-- Temporarily disable foreign key checks to alter table constraints without errors
SET FOREIGN_KEY_CHECKS=0;

-- Add a foreign key constraint to the pubid column of the half_authored table refere
ncing the half_publication table's pubid column
ALTER TABLE half_authored
ADD CONSTRAINT fk_pubid
FOREIGN KEY (pubid)
REFERENCES half_publication (pubid);

-- Add a foreign key constraint to the authorid column of the half_authored table ref
erencing the half_author table's authorid column
ALTER TABLE half_authored
ADD CONSTRAINT fk_authorid
FOREIGN KEY (authorid)
REFERENCES half_author (authorid);

-- Re-enable foreign key checks after altering table constraints
SET FOREIGN_KEY_CHECKS=1;

-- Set the primary key for the half_publication and half_author tables to ensure uniq
ueness and improve query performance
ALTER TABLE half_publication ADD PRIMARY KEY (pubid);
ALTER TABLE half_author ADD PRIMARY KEY (authorid);

-- The following queries are for verification and data integrity checks
-- Find any records in HALF_AUTHORED that do not have a corresponding record in HALF_
PUBLICATION based on pubid
SELECT half_authored.*
FROM half_authored
LEFT JOIN half_publication ON half_authored.pubid = half_publication.pubid
WHERE half_publication.pubid IS NULL; -- No records returned

-- Quarter the publication, author and authored tables
-- Change the current database to halfDB
USE halfDB;

-- Create a new database named quarterDB
CREATE DATABASE quarterDB;

-- Create a view in halfDB that selects only odd rows from the half_publication table
```

```sql
CREATE VIEW quarter_publication_view AS
SELECT pubid, pubkey, pubtype, title, pubyear, pubkey1, pubkey2, pubkey3 FROM (
	SELECT *, row_number() OVER (ORDER BY pubid) AS row_num
	FROM half_publication
) AS temp
WHERE row_num % 2 = 1;

-- Create a new table in quarterDB database using the data from the quarter_publicati
on_view
CREATE TABLE quarterDB.quarter_publication AS
SELECT *
FROM halfDB.quarter_publication_view;

-- Create a view in halfDB that joins the half_authored table with the quarter_public
ation table based on pubid
CREATE VIEW quarter_authored_view AS
SELECT au.*
FROM half_authored au
INNER JOIN quarterDB.quarter_publication p ON au.pubid = p.pubid;

-- Create a new table in quarterDB database using the data from the quarter_authored_
view
CREATE TABLE quarterDB.quarter_authored AS
SELECT *
FROM halfDB.quarter_authored_view;

-- Create a view in halfDB that selects distinct authors who have authored publicatio
ns in the quarter_authored table
CREATE VIEW quarter_author_view AS
SELECT DISTINCT a.*
FROM half_author a
INNER JOIN quarterDB.quarter_authored r ON a.authorid = r.authorid;

-- Create a new table in quarterDB database using the data from the quarter_author_vi
ew
CREATE TABLE quarterDB.quarter_author AS
SELECT *
FROM halfDB.quarter_author_view;

-- Switch to the quarterDB database for subsequent operations
USE quarterDB;

-- Disable foreign key checks before altering tables to add foreign key constraints
SET FOREIGN_KEY_CHECKS=0;

-- Add a foreign key constraint to the quarter_authored table referencing the pubid c
olumn of the quarter_publication table
ALTER TABLE quarter_authored
```

```sql
ADD CONSTRAINT fk_quarter_authored_pubid
FOREIGN KEY (pubid)
REFERENCES quarter_publication (pubid);

-- Add a foreign key constraint to the quarter_authored table referencing the authori
d column of the quarter_author table
ALTER TABLE quarter_authored
ADD CONSTRAINT fk_quarter_authored_authorid
FOREIGN KEY (authorid)
REFERENCES quarter_author (authorid);

-- Re-enable foreign key checks after adding constraints
SET FOREIGN_KEY_CHECKS=1;

-- Set the pubid column as the primary key of the quarter_publication table
ALTER TABLE quarter_publication
ADD PRIMARY KEY (pubid);

-- Set the authorid column as the primary key of the quarter_author table
ALTER TABLE quarter_author
ADD PRIMARY KEY (authorid);

-- Query to find any records in the quarter_authored table that do not have a corresp
onding record in the quarter_publication table
SELECT quarter_authored.*
FROM quarter_authored
LEFT JOIN quarter_publication ON quarter_authored.pubid = quarter_publication.pubid
WHERE quarter_publication.pubid IS NULL;
```