In the DBLP dataset, particularly in the **proceedings** table, the columns **title**, **editor**, **booktitle**, **series**, and **publisher** represent specific attributes related to the proceedings of conferences or workshops. Here's an explanation of each of these attributes:

1. **title:**
   * **Meaning:** This column represents the title of the conference proceedings or the title of the entire publication.
   * **Explanation:** The **title** attribute contains the title of the entire set of conference proceedings or the publication as a whole. It provides information about the main theme or focus of the conference or workshop.
2. **editor:**
   * **Meaning:** This column represents the editor or editors responsible for compiling and editing the proceedings.
   * **Explanation:** The **editor** attribute contains the names of individuals who played a role in editing and organizing the conference proceedings. Editors are responsible for selecting, reviewing, and organizing the papers presented at the conference.
3. **booktitle:**
   * **Meaning:** This column represents the title of the conference or workshop where the proceedings were presented.
   * **Explanation:** The **booktitle** attribute specifies the title of the specific conference or workshop for which the proceedings were created. It provides information about the event for which the papers were originally presented.
4. **series:**
   * **Meaning:** This column represents the series or collection to which the proceedings belong.
   * **Explanation:** The **series** attribute indicates if the conference proceedings are part of a series or collection. Conferences or workshops may be organized as part of a series, and this attribute helps categorize the proceedings within that series.
5. **publisher:**
   * **Meaning:** This column represents the publisher or entity responsible for publishing the proceedings.
   * **Explanation:** The **publisher** attribute contains information about the entity or organization responsible for publishing and distributing the conference proceedings. It indicates the publisher's name.

In summary, when you look at a row in the **proceedings** table in the DBLP dataset:

* The **title** represents the title of the entire set of conference proceedings.
* The **editor** lists the individuals responsible for editing and organizing the proceedings.
* The **booktitle** specifies the title of the specific conference or workshop.
* The **series** indicates if the proceedings are part of a series or collection.
* The **publisher** provides information about the entity or organization responsible for publishing the proceedings.

n the DBLP dataset, particularly in the **inproceedings** table, the columns **title**, **author**, and **booktitle** represent specific attributes related to articles published in conference proceedings. Here's what each of these attributes means:

1. **title:**
   * **Meaning:** This column represents the title of the article or paper that was presented or published in the conference proceedings.
   * **Explanation:** The **title** attribute contains the title of the individual paper or article authored by one or more authors and presented at a conference.
2. **author:**
   * **Meaning:** This column represents the author or list of authors who contributed to the paper.
   * **Explanation:** The **author** attribute typically contains the names of the author(s) who wrote the paper. Multiple authors are often listed, and the names are usually separated by commas.
3. **booktitle:**
   * **Meaning:** This column represents the title of the conference proceedings in which the article was published.
   * **Explanation:** The **booktitle** attribute specifies the title of the overall conference proceedings or the book where the individual paper is included. It indicates the name of the conference or the broader publication event.

In summary when you look at a row in the **inproceedings** table in the DBLP dataset:

* The **title** represents the title of the specific article or paper.
* The **author** lists the author(s) who contributed to the paper.
* The **booktitle** indicates the title of the conference proceedings or the larger publication venue where the article is included.

These attributes provide valuable information about individual papers presented at conferences, including details about their titles, authors, and the conferences in which they were published.

**Query 1**

**Query plan without using indexing:**

* The query involves sorting operations, which are resource intensive.
* Parallel processing is utilized to speed up certain parts of the query. However, the total execution time is around 4312.883 milliseconds. The **planning time is** 1.587 ms.
* The 'Sort' operations indicate potential areas for optimization. Indexing can be used on certain frequently used columns, like the year column, booktitle on proceedings and the inproceedings table.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Query plan with using Indexing:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

* The execution time has increased compared to the previous query plan. It went from 4.313 seconds to 6.097 seconds.
* While indexing has been applied to the columns involved in the query, the execution time has increased.
* The number of workers launched has increased from 2 to 3. However, the nested loop operation, especially the bitmap heap scan on 'inproceedings', seems to be taking a significant amount of time.
* I also ran the ANALYZE command on both the proceedings and inproceedings tables to analyze and update the statistics for the 'inproceedings' and 'proceedings' tables, respectively. This, in turn, could contribute to better query performance by allowing the query planner to make more informed decisions based on the current state of the data.
* However, it did not help in boosting the query performance. In the contrary, the total execution time increased further from 6.097 seconds to 6.8663 seconds.
* This query plan gives an impression ‘Index’ usage might not always lead to performance improvement.

**Query 2**

**Query plan without using indexing:**

* Rows Removed by Filter: 8025
* Sort Method: quicksort Memory: 362kB" OR (booktitle ~~ '%SIGMOD%'::text))
* Rows Removed by Filter: 1131932
* Planning Time: 0.173 ms
* Execution Time: 585.603 ms

**Query plan using indexing:**

* Rows Removed by Filter: 2395
* Index Scan using idx\_booktitle on inproceedings (cost=0.43..8.45 rows=246 width=61)
* Planning Time: 0.301 ms
* Execution Time: 6.719 ms

**Inferences**

* Query with indexing
  + The execution time was 6.719 ms.
* Query without utilizing indexing
  + The execution time was 585.603 ms.
* The query with indexing is much faster, completing in just 6.719 milliseconds, while the query without indexing takes significantly longer at 585.603 milliseconds.
* I implemented **indexing** on the “**booktitle**” column in the inproceedings table. There are notable differences in the query execution plan output.
* The index scan effectively filters rows based on the specified conditions of booktitle matching 'VLDB%' or 'SIGMOD%'.
* The query demonstrates a clear advantage in terms of efficiency when using indexing. The number of rows removed by the filter is significantly lower (2395 rows with indexing compared to 1131932 rows without indexing), indicating that the indexing effectively narrows down the result set.
* This query involves a HashAggregate operation with a cost of 10.30 and 13.37 rows. The width of the result is 101. The actual execution time is 4.402, and it returns 0 rows. The operation is performed on the Group Key, which consists of the author and booktitle of the inproceedings\_1 table. "Filter condition: (count(\*) is greater than or equal to 10)" "One batch was processed with a memory usage of 977kB." "A total of 4500 rows were removed due to filtering."
* The query with indexing demonstrates a reduction in memory usage for the hash aggregates, resulting in a more efficient utilization of system resources.
* This query optimizes performance by utilizing parallel index scans, which leverage multiple workers to process the data concurrently.
* This query's indexing offers numerous advantages, including improved retrieval speed, decreased filtered rows, shorter planning and execution times, optimized memory usage, and the ability to utilize parallel processing.

**Query 3: Conference Publications by Decade**

**Query without indexing- Query plan details**

* The total execution time taken to run the query is 4.645 seconds.
* The query involves sorting and grouping based on a calculated decade range.
* The hash join is performed but looks like it is taking relative high time
* Also, I can see that sequential scan is performed on inproceedings table as well as distinctyears temporary table

A screenshot of a computer

Description automatically generated

**Query with Indexing- Query plan details**

* Indexing was applied on DistinctYears “year” column and inproceedings “year” column as they are largely used to gather distinct years and then count all the conference papers within the given decade and arrange by year sequentially.
* Merge join between the indexed distinctyears table (dy) and the indexed inproceedings table (ip) based on the year column.
* Index Only Scan is used for both tables, which indicates that the necessary data is retrieved directly from the index without visiting the actual table.
* The dy table is scanned with Heap Fetches: 70.
* The ip table is scanned with a heap fetch of 0.
* Applying indexing to the year columns has significantly improved performance compared to the previous query plan without indexing.
* The merge join with index-only scan on both tables indicates that the database is retrieving data directly from the indexes, which generally leads to faster execution.
* While the sorting time and overall execution time are still noticeable, the improvement is evident.

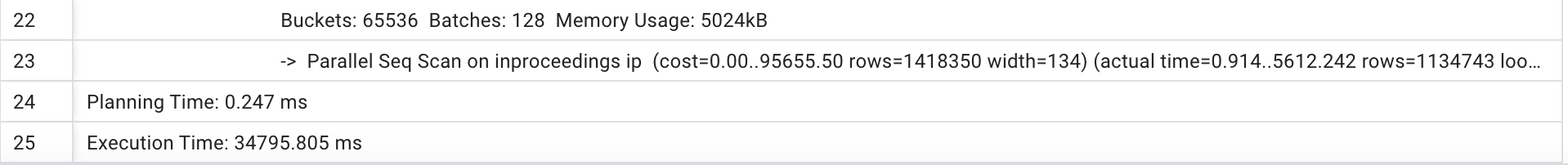
**Query 4: Highly Published Data Authors**

**Query without Indexing- Query plan details**

* The total execution time of the query without using any idexing technique is 34.8 seconds (34795.805 ms).
* Parallelism is used in various parts of the execution plan, including Parallel Hash Join, Parallel Hash, and Parallel Seq Scan.

A screenshot of a computer

Description automatically generated



To determine suitable columns for indexing, it's essential to consider the columns used in join conditions, filter conditions, and columns involved in grouping and sorting operations. In my query, here are the columns that could be candidates for indexing:

In my query, I have used articles and an in-proceedings table.

* The 'author' column in the 'articles' table is used for the join condition.sa.single\_author = ip.author), and it's part of the GROUP BY clause. This could be a candidate for indexing.
* Indexing on 'journal' may help as there are a significant number of rows and filtering based on the journal is frequent.
* The 'author' column in the 'inproceedings' table is used for the join condition (sa.single\_author = ip.author), and it's part of the GROUP BY clause.
* Indexing on 'year' may also help, as that column is used in join conditions.
* Indexing on 'title' may help, as we are counting the distinct articles or papers that contain the word data in them.

**Query with Indexing- Query plan details**

* I considered creating indexes on 'articles.author,' 'articles.year,' 'articles.journal,' 'inproceedings.author,' 'inproceedings.year,' and 'inproceedings.title.'
* I re-ran the query after creating the above indexes and compared the execution plan for improved performance.
* The updated query plan still shows that the indexes are not being utilized, as evidenced by the continued use of parallel sequential scans on both 'articles' and 'inproceedings.' The execution time is still relatively high.
* The total execution time of the query is approximately 32.1 seconds (32072.666 ms).

A screenshot of a computer program

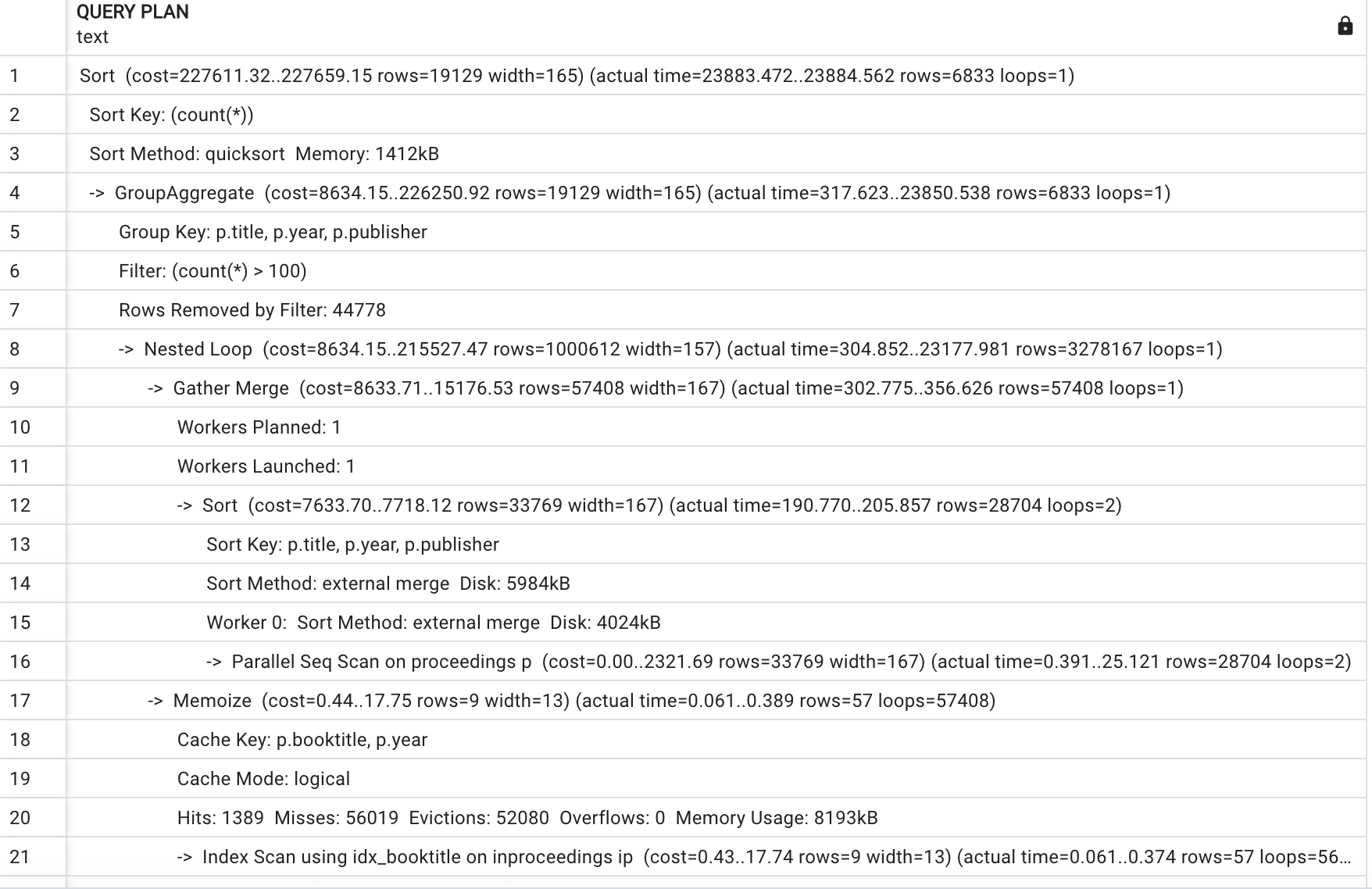
Description automatically generated

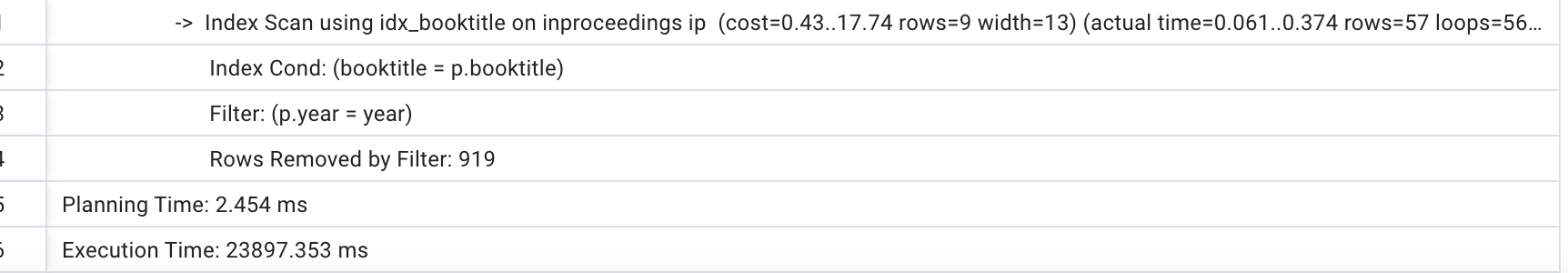
A screenshot of a computer

Description automatically generated

**Query 5: Highly Published June Conferences**

**Query Without Indexing- Query plan details**





* The total execution time for the query is 23.89 seconds.
* The first part of the query was executed in approximately 304 milliseconds, and it processed nearly 3.2 million rows. This section of the query plan illustrates the following.
  + Nested Loop (cost=8634.15..215527.47 rows=1000612 width=157) (actual time=304.852..23177.981 rows=3278167 loops=1). The sorting method utilized for the search is quick sort and it consumed 1412 KB of memory.
* The processing of the proceeding tables is taking a significant amount of time due to a parallel sequential scan on the proceedings table, which involves going through nearly 28k rows. This is despite the use of index scanning on the booktitle column of the inproceeding table.

**Query with indexing and other optimization techniques- Query plan details**

**First run:**

* Execution time: 2490 milliseconds = 2.4 seconds
* The number of rows processed in group aggregates was reduced to ~1 million rows.
* The initial run with EXPLAIN ANALYZE showed that the total execution time was less than the one where indexing was not used on the proceedings table. However, after creating indexes on the "booktitle" and "year" columns in the proceedings table, it appears to be still taking longer to execute the query. The query doesn’t appear to be using indexing in the proceedings table.
* One other tuning option that I tried was to increase the “max\_parallel\_workers\_per\_gather” by 4 (initially set to 2 by default). This did decrease the total amount of execution time it took to execute the query; however, there isn’t any significant change in the query performance.
* After adding indexing to the inproceeding and proceeding table columns (booktitle, year) and changing the “max\_parallel\_workers\_per\_gather” by 4, here’s how the query plan looks.

A screenshot of a computer

Description automatically generated

**Second Run:**

A screenshot of a computer

Description automatically generated

Conclusion:

* I ensured that statistics on tables were up-to-date by running the following commands:
  + ANALYZE articles;
  + ANALYZE inproceedings;
* I confirmed that the optimizer is aware of the indexes. I checked the pg\_indexes system catalog to verify the existence and visibility of the created indexes.
* Based on my research, in some complex queries involving multiple joins, conditions, or aggregations, the query planner might find a sequential scan to be more efficient.
* If the conditions in the query are not selective (i.e., they match a large portion of the table), PostgreSQL might choose a sequential scan over an index scan.
* A query might perform better without using indexes. There could be scenarios where the PostgreSQL query planner might decide that a sequential scan (full table scan) is more efficient based on factors such as the size of the table, the selectivity of the conditions, and the distribution of data.