PostgreSQL Mini-Project Performance Tuning Report

Query 1: Publications by Decade (Excluding Seasonal/Special Conferences)

1. How did you improve performance?

The original query used a parallel sequential scan over the `inproceedings` table, scanning over 500K rows and applying `ILIKE` filters late in the plan. The filtering criteria included booktitles that should not contain seasonal terms (e.g., '%workshop%', '%symposium%'). I restructured the query to push filtering earlier using a CTE and cast the `year` field to integer properly, which reduced unnecessary computation. Additionally, I disabled sequential scan to encourage index usage.

2. Where did you create new indexes?

- CREATE INDEX idx_proceedings_covering ON public.proceedings (booktitle, year);

- CREATE INDEX idx_inproceedings_covering ON public.inproceedings (booktitle, year, "inproceedingsID");

3. What was the impact?

Execution time dropped significantly — from over 35 seconds to around 11.5 seconds. PostgreSQL began using an index-only scan on `idx_inproceedings_covering` with parallel workers, avoiding the need to read the entire heap.

4. Cache Effects

After the first execution, subsequent runs were faster due to data being held in memory buffers. Sorting and grouping operations were significantly faster on repeat runs.

Query 2: Summary of Publications per Conference

1. How did you improve performance?

This query performed acceptably from the outset, thanks to relatively small intermediate row sets and limited filtering.

2. Where did you create new indexes?

No indexes were created for this query.

3. What was the impact?

The query executed quickly from the start — under 20 ms. No further tuning was needed.

4. Cache Effects

Negligible, as the query returned quickly in both cold and warm cache conditions.

Query 3: Excluding Publications with Specific Booktitle Patterns

1. How did you improve performance?

The original query relied on sequential scans and filtered over 500K rows post-scan using `ILIKE '%workshop%'` and similar patterns. This delayed filtering caused high latency. I applied GIN indexes and used `LOWER(booktitle)` to better leverage text pattern matching with trigram support.

2. Where did you create new indexes?

- CREATE INDEX idx_booktitle_text_ops ON public.inproceedings (booktitle text_pattern_ops);

- CREATE INDEX idx_lower_booktitle ON public.inproceedings USING btree (LOWER(booktitle));

- CREATE INDEX idx_booktitle_gin ON public.inproceedings USING GIN (booktitle gin_trgm_ops);

3. What was the impact?

Execution time dropped from about 7.8 seconds to 1.6 seconds. The query plan shifted from sequential scan to a bitmap index scan using the GIN index. This improved filtering and reduced the number of rows processed during aggregation.

4. Cache Effects

Repeat executions further reduced latency due to cached index pages and warmed buffers. Bitmap scans benefitted significantly from warm cache.

Query 4: Top 10 Authors Publishing on "Data"

1. How did you improve performance?

Initial runs of this query took over 35 seconds using full sequential scans on `articles` and `inproceedings`, followed by large joins and aggregations. I introduced CTEs to filter on `LOWER(title) LIKE '%data%'` early and applied GIN indexes on those fields. I also disabled sequential scans and encouraged bitmap index scan usage.

2. Where did you create new indexes?

- CREATE INDEX idx_articles_title_gin ON articles USING gin (LOWER(title) gin_trgm_ops);

- CREATE INDEX idx_inproceedings_title_gin ON inproceedings USING gin (LOWER(title) gin_trgm_ops);

- CREATE INDEX idx_authors_name ON authors("authorName");

3. What was the impact?

Execution time dropped significantly — from around 16.7 seconds to under 8.9 seconds. The query plan used parallel bitmap heap scans, hash joins, and eliminated redundant nested loops. Memory usage dropped, and performance scaled well with parallel workers.

4. Cache Effects

Repeat execution showed gains of up to 30%, with key index pages being reused from memory.

Query 5: June Conferences with >100 Publications

1. How did you improve performance?

Although no new indexes were needed, I reused covering indexes created earlier (`idx_proceedings_covering`, `idx_inproceedings_covering`) to identify conference editions whose `title` field included 'June'. Using a CTE to pre-filter June-related conferences, then joining back to aggregate publication counts, allowed PostgreSQL to execute the query efficiently.

2. Where did you create new indexes?

None created specifically for this query; leveraged existing indexes from Queries 1 and 4.

3. What was the impact?

Performance was acceptable from the outset — no additional tuning was needed.

4. Cache Effects

Minor gains were observed on repeated runs, with consistent execution due to reuse of cached buffers and pre-sorted index pages.

Appendix: Query Planning and Execution Times (ms)

Worksheet	Planning Time	Execution Time
Query 4.1 Before	0.74 ms	35,631.7 ms
Query 4.1 After	1.03 ms	11,640.6 ms
Query 4.3 Before	1.11 ms	7,812.1 ms
Query 4.3 After	1.27 ms	1,638.1 ms
Query 4.4 Before	1.4 ms	16,743.1 ms
Query 4.4 After	66.61 ms	8,166.1 ms