# Index Analysis

General indexes overview:

1. B-tree: B-tree is the default index type in PostgreSQL and handles equality and range queries efficiently. B-trees can handle all kinds of data: numbers, strings, and even complex types like arrays and JSON. They are useful when you need to perform sorting (ORDER BY), as well as for most common types of queries.
2. Hash: A hash index is used only for equality comparisons and can be faster than a B-tree for these types of operations. The drawback of hash indexes is that they only support equality (=) comparisons. Before PostgreSQL version 10, hash indexes were not write-ahead logged, meaning that they could not be recovered after a crash. This has changed in PostgreSQL 10 and later versions, where hash indexes are now logged and therefore safe to use in all situations.
3. Bitmap: A bitmap index uses bitmaps to answer queries quickly. In PostgreSQL, bitmap indexes are not explicitly created by the user. Instead, the PostgreSQL query planner can use a bitmap scan on B-tree indexes when it thinks that would be more efficient. This involves creating a temporary in-memory bitmap index for the duration of the query. It's important to note that bitmap scans are not a separate type of index, but a way to use a B-tree index.
4. Specialized Indexes: PostgreSQL has several specialized indexes for specific use-cases:

* GIN (Generalized Inverted Index): GIN indexes are used when you have composite values, such as arrays or JSONB data types, and you need to query on the elements within them. For example, a GIN index on an array can quickly answer "does this array contain this value?" type questions.
* GiST (Generalized Search Tree): GiST indexes provide a balanced, tree-structured access method, which can be used for different types of queries and data. For example, GiST indexes can be used for geometric data types or full-text search.

## Comparison of different indexes’ use cases:

1. Analysis index on field ‘*name’* (exact match)

**Hash:**Index Scan using idx\_students\_name on students (cost=0.00..8.02 rows=1 width=64) (actual time=0.022..0.024 rows=1 loops=1)

Index Cond: (name = 'Name 50000'::text)

Planning Time: 0.217 ms

Execution Time: 0.049 ms

**Btree:**  
Index Scan using idx\_students\_name on students (cost=0.42..8.44 rows=1 width=64) (actual time=0.044..0.046 rows=1 loops=1)

Index Cond: (name = 'Name 50000'::text)

Planning Time: 0.214 ms

Execution Time: 0.088 ms

**Gist:**  
Index Scan using idx\_students\_name on students (cost=0.28..8.30 rows=1 width=64) (actual time=1.786..2.421 rows=1 loops=1)

Index Cond: (name = 'Name 50000'::text)

Rows Removed by Index Recheck: 1

Planning Time: 0.152 ms

Execution Time: 2.452 ms

**Gin:**  
Bitmap Heap Scan on students (cost=84.01..88.02 rows=1 width=64) (actual time=7.024..7.025 rows=1 loops=1)

Recheck Cond: (name = 'Name 50000'::text)

Rows Removed by Index Recheck: 1

Heap Blocks: exact=2

-> Bitmap Index Scan on idx\_students\_name (cost=0.00..84.01 rows=1 width=0) (actual time=7.007..7.008 rows=2 loops=1)

Index Cond: (name = 'Name 50000'::text)

Planning Time: 0.162 ms

Execution Time: 7.140 ms

**Without index:**

Seq Scan on students (cost=0.00..2485.00 rows=1 width=64) (actual time=10.252..21.943 rows=1 loops=1)

Filter: (name = 'Name 50000'::text)

Rows Removed by Filter: 99999

Planning Time: 0.182 ms

Execution Time: 21.968 ms

**Conclusion:**   
Overall, for exact match queries on the 'name' field, Hash and B-Tree indexes are the most efficient, with Hash being the fastest. However, the choice between these two would depend on the specific requirements and constraints of the system (replication needs, transaction safety, etc.). For more complex data types and search conditions, I might want to consider using GiST or GIN indexes. Without an index, the performance is the worst as the size of the data grows.

1. Analysis index on field ‘*surname’* (partial match)

**Gin:**Bitmap Heap Scan on students (cost=178.11..1552.00 rows=11111 width=64) (actual time=6.561..10.450 rows=11111 loops=1)

Recheck Cond: (surname ~~ 'Surname 5%'::text)

Heap Blocks: exact=140

-> Bitmap Index Scan on idx\_students\_surname (cost=0.00..175.33 rows=11111 width=0) (actual time=6.521..6.521 rows=11111 loops=1)

Index Cond: (surname ~~ 'Surname 5%'::text)

Planning Time: 0.219 ms

Execution Time: 11.189 ms

**Gist:**  
Bitmap Heap Scan on students (cost=546.39..1920.28 rows=11111 width=64) (actual time=12.013..16.012 rows=11111 loops=1)

Recheck Cond: (surname ~~ 'Surname 5%'::text)

Heap Blocks: exact=140

-> Bitmap Index Scan on idx\_students\_surname (cost=0.00..543.61 rows=11111 width=0) (actual time=11.976..11.976 rows=11111 loops=1)

Index Cond: (surname ~~ 'Surname 5%'::text)

Planning Time: 0.248 ms

Execution Time: 16.590 ms

**Btree:**Seq Scan on students (cost=0.00..2485.00 rows=11111 width=64) (actual time=0.031..24.749 rows=11111 loops=1)

Filter: (surname ~~ 'Surname 5%'::text)

Rows Removed by Filter: 88889

Planning Time: 0.247 ms

Execution Time: 25.247 ms

**Without index:**

Seq Scan on students (cost=0.00..2485.00 rows=11111 width=64) (actual time=0.029..28.263 rows=11111 loops=1)

Filter: (surname ~~ 'Surname 5%'::text)

Rows Removed by Filter: 88889

Planning Time: 0.176 ms

Execution Time: 27.870 ms

**Hash:**Seq Scan on students (cost=0.00..2485.00 rows=11111 width=64) (actual time=0.038..33.048 rows=11111 loops=1)

Filter: (surname ~~ 'Surname 5%'::text)

Rows Removed by Filter: 88889

Planning Time: 0.222 ms

Execution Time: 33.659 ms

**Conclusion:**  
Overall, for partial match queries on the 'surname' field, GIN index provides the best performance followed by GiST. B-Tree indexes are not designed for partial text match searches and Hash indexes do not support such queries at all. Without an index, the performance is worse as the size of the data grows.

1. Analysis index on field ‘*phone\_number’* (partial match)

**Gin:**Bitmap Heap Scan on students (cost=52.08..89.50 rows=10 width=64) (actual time=0.612..0.663 rows=100 loops=1)

Recheck Cond: (phone\_number ~~ '555-05%'::text)

Heap Blocks: exact=2

-> Bitmap Index Scan on idx\_students\_phone\_number (cost=0.00..52.07 rows=10 width=0) (actual time=0.597..0.598 rows=100 loops=1)

Index Cond: (phone\_number ~~ '555-05%'::text)

Planning Time: 0.712 ms

Execution Time: 0.936 ms

**Gist:**  
Bitmap Heap Scan on students (cost=4.36..41.78 rows=10 width=64) (actual time=2.801..2.829 rows=100 loops=1)

Recheck Cond: (phone\_number ~~ '555-05%'::text)

Heap Blocks: exact=2

-> Bitmap Index Scan on idx\_students\_phone\_number (cost=0.00..4.36 rows=10 width=0) (actual time=2.769..2.769 rows=100 loops=1)

Index Cond: (phone\_number ~~ '555-05%'::text)

Planning Time: 0.767 ms

Execution Time: 2.889 ms

**Btree:**Seq Scan on students (cost=0.00..2485.00 rows=10 width=64) (actual time=0.182..27.928 rows=100 loops=1)

Filter: (phone\_number ~~ '555-05%'::text)

Rows Removed by Filter: 99900

Planning Time: 0.261 ms

Execution Time: 27.955 ms

**Without index:**

Seq Scan on students (cost=0.00..2485.00 rows=10 width=64) (actual time=0.195..28.224 rows=100 loops=1)

Filter: (phone\_number ~~ '555-05%'::text)

Rows Removed by Filter: 99900

Planning Time: 0.189 ms

Execution Time: 28.254 ms

**Hash:**Seq Scan on students (cost=0.00..2485.00 rows=10 width=64) (actual time=0.315..28.983 rows=100 loops=1)

Filter: (phone\_number ~~ '555-05%'::text)

Rows Removed by Filter: 99900

Planning Time: 0.193 ms

Execution Time: 29.010 ms

**Conclusion:**  
In conclusion, for queries involving partial matches on text fields like phone numbers, GIN indexes offer the best performance. GiST indexes also provide good performance but are slower than GIN. B-tree, hash indexes, and sequential scans without an index are significantly slower as they are not optimized for such queries. Therefore, for this particular type of query, using a GIN index would be most efficient.

1. Analysis index on search query ‘Find user with marks by user surname’ (partial match)

**Gist:**  
Hash Join (cost=2063.17..21058.28 rows=111110 width=27) (actual time=18.837..308.199 rows=111110 loops=1)

Hash Cond: (e.student\_id = s.id)

-> Seq Scan on exam\_results e (cost=0.00..16370.00 rows=1000000 width=8) (actual time=0.015..98.135 rows=1000000 loops=1)

-> Hash (cost=1924.28..1924.28 rows=11111 width=27) (actual time=18.716..18.718 rows=11111 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 819kB

-> Bitmap Heap Scan on students s (cost=550.39..1924.28 rows=11111 width=27) (actual time=9.748..14.489 rows=11111 loops=1)

Recheck Cond: (surname ~~ '%Surname 5%'::text)

Heap Blocks: exact=140

-> Bitmap Index Scan on idx\_students\_surname (cost=0.00..547.61 rows=11111 width=0) (actual time=9.718..9.718 rows=11111 loops=1)

Index Cond: (surname ~~ '%Surname 5%'::text)

Planning Time: 1.051 ms

Execution Time: 312.933 ms

**Gin:**Hash Join (cost=1674.89..20670.00 rows=111110 width=27) (actual time=14.550..358.426 rows=111110 loops=1)

Hash Cond: (e.student\_id = s.id)

-> Seq Scan on exam\_results e (cost=0.00..16370.00 rows=1000000 width=8) (actual time=0.018..112.525 rows=1000000 loops=1)

-> Hash (cost=1536.00..1536.00 rows=11111 width=27) (actual time=14.386..14.389 rows=11111 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 819kB

-> Bitmap Heap Scan on students s (cost=162.11..1536.00 rows=11111 width=27) (actual time=4.999..10.027 rows=11111 loops=1)

Recheck Cond: (surname ~~ '%Surname 5%'::text)

Heap Blocks: exact=140

-> Bitmap Index Scan on idx\_students\_surname (cost=0.00..159.33 rows=11111 width=0) (actual time=4.962..4.962 rows=11111 loops=1)

Index Cond: (surname ~~ '%Surname 5%'::text)

Planning Time: 0.490 ms

Execution Time: 363.649 ms

**Btree:**Hash Join (cost=2623.89..21619.00 rows=111110 width=27) (actual time=35.923..392.597 rows=111110 loops=1)

Hash Cond: (e.student\_id = s.id)

-> Seq Scan on exam\_results e (cost=0.00..16370.00 rows=1000000 width=8) (actual time=0.041..112.968 rows=1000000 loops=1)

-> Hash (cost=2485.00..2485.00 rows=11111 width=27) (actual time=35.851..35.853 rows=11111 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 819kB

-> Seq Scan on students s (cost=0.00..2485.00 rows=11111 width=27) (actual time=0.031..30.611 rows=11111 loops=1)

Filter: (surname ~~ '%Surname 5%'::text)

Rows Removed by Filter: 88889

Planning Time: 0.609 ms

Execution Time: 397.747 ms

**Without index:**

Hash Join (cost=2623.89..21619.00 rows=111110 width=27) (actual time=37.817..395.278 rows=111110 loops=1)

Hash Cond: (e.student\_id = s.id)

-> Seq Scan on exam\_results e (cost=0.00..16370.00 rows=1000000 width=8) (actual time=0.023..115.762 rows=1000000 loops=1)

-> Hash (cost=2485.00..2485.00 rows=11111 width=27) (actual time=37.769..37.771 rows=11111 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 819kB

-> Seq Scan on students s (cost=0.00..2485.00 rows=11111 width=27) (actual time=0.018..33.144 rows=11111 loops=1)

Filter: (surname ~~ '%Surname 5%'::text)

Rows Removed by Filter: 88889

Planning Time: 0.398 ms

Execution Time: 400.174 ms

**Hash:**Hash Join (cost=2623.89..21619.00 rows=111110 width=27) (actual time=51.418..397.221 rows=111110 loops=1)

Hash Cond: (e.student\_id = s.id)

-> Seq Scan on exam\_results e (cost=0.00..16370.00 rows=1000000 width=8) (actual time=0.036..112.978 rows=1000000 loops=1)

-> Hash (cost=2485.00..2485.00 rows=11111 width=27) (actual time=51.352..51.354 rows=11111 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 819kB

-> Seq Scan on students s (cost=0.00..2485.00 rows=11111 width=27) (actual time=0.024..46.344 rows=11111 loops=1)

Filter: (surname ~~ '%Surname 5%'::text)

Rows Removed by Filter: 88889

Planning Time: 0.518 ms

Execution Time: 402.179 ms

**Conclusion:**  
In conclusion, for queries that involve a partial match on a text field like a surname and fetching related data from another table (in this case, exam results), using a GiST or GIN index would yield better performance. While B-tree, no index, and hash indexes are less efficient for such operations. The specific dataset, system configuration, and the PostgreSQL version could influence these results, so it's always a good idea to benchmark indexes with the data and workload.

1. Analysis on btree and hash indexes (exact match)

Hash:

Index Scan using idx\_students\_name on students (cost=0.00..8.02 rows=1 width=64) (actual time=0.022..0.024 rows=1 loops=1)

Index Cond: (name = 'Name 50000'::text)

Planning Time: 0.217 ms

Execution Time: 0.049 ms

Like  
Seq Scan on students (cost=0.00..2485.00 rows=10 width=64) (actual time=0.215..26.163 rows=111 loops=1)

Filter: (name ~~ 'Name 500%'::text)

Rows Removed by Filter: 99889

Planning Time: 0.216 ms

Execution Time: 26.194 ms

Btree:  
Index Scan using idx\_students\_name on students (cost=0.42..8.44 rows=1 width=64) (actual time=0.044..0.046 rows=1 loops=1)

Index Cond: (name = 'Name 50000'::text)

Planning Time: 0.214 ms

Execution Time: 0.088 ms

Without index:

Seq Scan on students (cost=0.00..2485.00 rows=10 width=64) (actual time=0.103..21.233 rows=111 loops=1)

Filter: (name ~~ 'Name 500%'::text)

Rows Removed by Filter: 99889

Planning Time: 0.118 ms

Execution Time: 21.255 ms

Like  
Seq Scan on students (cost=0.00..2485.00 rows=10 width=64) (actual time=0.215..25.900 rows=111 loops=1)

Filter: (name ~~ 'Name 500%'::text)

Rows Removed by Filter: 99889

Planning Time: 0.213 ms

Execution Time: 25.931 ms

1. Analysis of btree and gin indexes (partial match)

**Btree:**

Seq Scan on students (cost=0.00..2485.00 rows=11111 width=64) (actual time=0.031..24.749 rows=11111 loops=1)

Filter: (surname ~~ 'Surname 5%'::text)

Rows Removed by Filter: 88889

Planning Time: 0.247 ms

Execution Time: 25.247 ms

**Gin:**

Bitmap Heap Scan on students (cost=178.11..1552.00 rows=11111 width=64) (actual time=6.561..10.450 rows=11111 loops=1)

Recheck Cond: (surname ~~ 'Surname 5%'::text)

Heap Blocks: exact=140

-> Bitmap Index Scan on idx\_students\_surname (cost=0.00..175.33 rows=11111 width=0) (actual time=6.521..6.521 rows=11111 loops=1)

Index Cond: (surname ~~ 'Surname 5%'::text)

Planning Time: 0.219 ms

Execution Time: 11.189 ms

Gist:  
equal:

Index Scan using idx\_students\_surname on students (cost=0.28..8.30 rows=1 width=64) (actual time=0.457..1.885 rows=1 loops=1)

Index Cond: (surname = 'Surname 50000'::text)

Rows Removed by Index Recheck: 1

Planning Time: 0.161 ms

Execution Time: 1.908 ms