

Database Indexing and Query Performance: How Indexes Make Searching Large Tables Scalable

Devvrat R. Tripathi¹, Shalini K. Dwivedi², Aniruddh V. Shukla³, Meghna M. Awasthi⁴, Yatharth T. Dixit⁵

ABSTRACT

Databases hold the data on which most information systems depend, and a common operation is to find particular records among many, but searching a large table by examining every record is slow, and becomes slower in proportion as the table grows, so that on a very large table such a search is impractical; an index makes this search fast by providing an ordered structure that locates records without scanning them all. This study examines database indexing and query performance and how an index makes searching a large table scalable. The time to find a record was examined as the size of the table grew, with and without an index, so that the difference between scanning every record and using an index, and how each scaled with the size of the table, could be seen. Without an index the search time grew in direct proportion to the size of the table, since every record had to be examined, so a table ten times larger took ten times as long to search, and a very large table took impractically long; with an index the search time grew only very slowly, far more slowly than the table, because the index located the record through an ordered structure in a number of steps that rose only logarithmically with the size of the table, so that even a very large table could be searched almost as quickly as a small one. The difference therefore widened enormously as the table grew, the indexed search remaining fast where the unindexed search became unusable. This is why indexes are essential to database performance, making searches that would otherwise be impractical on large tables fast and scalable. The study shows that indexing changes search time from growing in proportion to the table to growing only logarithmically, and it discusses the implications, and the costs, of indexing in information systems.

KEYWORDS: *Database indexing, Query performance, B-tree, Search, Scalability, Relational database, Full table scan, Logarithmic search, Information systems*

INTRODUCTION

Databases hold the data on which most information systems depend, the records of customers, transactions, products, and the rest, often in tables of very many rows. A fundamental operation on such a table is to find a particular record or records, those matching some condition, such as a customer with a given identifier. How quickly this search can be done largely determines how responsive the system is, since searches underlie almost everything a database is asked to do (1, 6).

The simplest way to search a table is to examine every record in turn, checking each against the condition, a full scan. This works, but it is slow on a large table, and crucially it becomes slower in proportion as the table grows: a table twice as large takes twice as long to scan, and a table of millions of records takes a long time to search record by record, so that on a very large table a full scan for every query is impractical and makes the system sluggish (5).

An index solves this. An index is an auxiliary ordered structure, kept alongside the table, that allows a record to be located by its key without examining every record, much as the index of a book lets one find a topic without reading every page; the most common form locates a record in a number of steps that grows only slowly with the size of the table (2, 3). This paper examines database indexing and query performance and how an index makes searching a large table scalable, examining the search time with and without an index as the table grows. The aim is to show how indexing changes the way search time grows with the table and to discuss the implications, and the costs, of indexing in information systems.

LITERATURE REVIEW

The organisation of data for efficient search is a foundational subject of database systems, which rest on the relational model of data and on structures for locating records quickly (1). Central among these is the balanced tree index, introduced to maintain large ordered indices efficiently, which locates a record in a number of steps growing only logarithmically with the

size of the data, and which, with its variants, remains the standard index structure, treated in all the texts (2, 3, 10).

Indexes and Query Processing

The literature describes how indexes speed search:

- Balanced tree indexes locate records in steps growing logarithmically with table size (2, 3).
- Searching without an index scans every record, taking time proportional to the table (5).
- The query optimiser chooses whether to use an index for a given query (4).
- Indexes speed search but cost storage and must be maintained as data changes (3, 6).

A recurring theme is that an index turns a search proportional to the table into one that is nearly independent of its size.

The literature establishes the contrast at the heart of indexing: a search without an index examines every record and so takes time proportional to the size of the table, while a search using a balanced tree index examines only a few records along a path through the structure, a number growing only logarithmically with the table, so the indexed search is vastly faster on large tables (2, 3, 5). It also describes how a database decides whether to use an index, through the query optimiser that selects an access path for each query, and notes the costs of indexes, the storage they occupy and the effort to keep them updated as the data changes, which mean indexes are chosen rather than created on everything (4, 6, 7). The structures, algorithms, and trade-offs of indexing are treated comprehensively in the standard references (6, 7, 8, 9).

RESEARCH GAP

Although indexing is foundational, a concise demonstration that contrasts how search time grows with the table size with and without an index, showing the proportional growth of the scan against the slow growth of the indexed search, remains useful as a clear illustration. There is therefore value in a study that displays both against table size and relates the difference to the scalability of database search.

OBJECTIVES

The objectives of this study are:

- To examine the time to find a record as the size of the table grows, with and without an index.
- To show how the unindexed search time grows in proportion to the table.
- To show how the indexed search time grows only logarithmically with the table.
- To relate the difference to the scalability of database search and the costs of indexing.

METHODOLOGY

Search With and Without an Index

The time to find a record was examined as the size of the table grew, in two cases: without an index, where every record is examined in turn, so the time grows in proportion to the table; and with a balanced tree index, where the record is located through an ordered structure in a number of steps growing only logarithmically with the table (3, 5). The elements are summarised in Table 1.

Table 1: Elements of database search and indexing

Element	Description
Full scan	Examining every record in turn
Index	Ordered structure locating records by key
Table size	Number of records in the table
Search time	Time to find a matching record

Scaling With Table Size

The search time was examined as the table grew from small to very large, for the scan, growing in proportion to the table, and for the indexed search, growing only logarithmically (2, 3). The widening difference between them, and what it means for searching large tables, were considered (5).

RESULTS AND FINDINGS

Without an index the search time grew in direct proportion to the size of the table, while with an index it grew only very slowly, as shown in Figure 1, which plots the search time against the table size for a full scan and an indexed lookup. The unindexed scan had to examine every record, so a table ten times larger took ten times as long to search, and a very large table took impractically long (5).

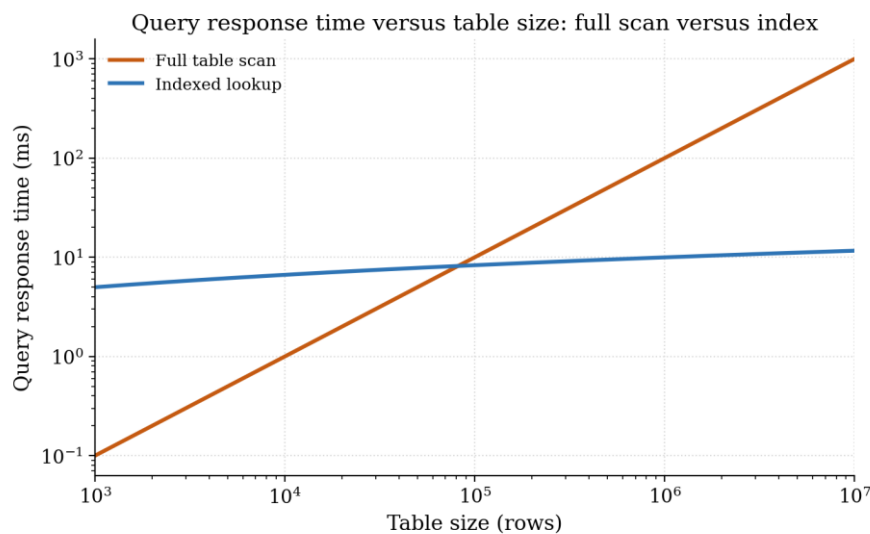


Figure 1: Query response time versus table size for a full scan and an index

The indexed search, by contrast, grew far more slowly than the table, because the index located the record through its ordered structure in a number of steps that rose only logarithmically with the size of the table, so that even a very large table could be searched almost as quickly as a small one (2, 3). The difference therefore widened enormously as the table grew, the indexed search remaining fast where the unindexed search became unusable. The findings are summarised in Table 2.

Table 2: Search time with table size

Table size	Full scan	Indexed lookup
Small	Fast	Fast
Large	Slow	Still fast

Table size	Full scan	Indexed lookup
Very large	Impractical	Fast
As table grows	Grows in proportion	Grows logarithmically

DISCUSSION

The results show that indexing changes search time from growing in proportion to the table to growing only logarithmically, and the difference is one of fundamentally different scaling. A full scan must look at every record, so the work it does is proportional to the number of records, and as the table grows the search slows in step with it without limit; this is why scanning is acceptable on a small table but ruinous on a large one (5). An index, by locating a record through an ordered structure, examines only a path of records whose length grows logarithmically with the table, so that multiplying the table many times over adds only a little to the search; the indexed search is therefore nearly insensitive to the size of the table, staying fast as the table grows (2, 3).

This difference in scaling is why the gap between the two widens so enormously and why indexing matters. On a small table both are quick and the difference hardly shows, which is why indexing seems unnecessary at small scale; but as the table grows, the scan slows in proportion while the indexed search barely changes, so the gap grows without bound, and on a very large table the indexed search is faster by an enormous factor, the difference between a query that returns at once and one that does not return usefully at all (3). Indexing is thus not a minor optimisation but the difference between a database that scales to large data and one that does not, which is why indexes are fundamental to database performance.

Indexes are not free, however, and this shapes how they are used. An index occupies storage, and, more importantly, it must be kept up to date as records are inserted, deleted, or changed, which adds work to every such change, since the index too must be updated; so an index speeds searches but slows updates and consumes space (2, 6). For this reason indexes are created selectively, on the columns by which records are frequently searched, where the gain to queries outweighs the cost to updates and storage, rather than on everything; and the database's query

optimiser decides, for each query, whether using an available index is actually faster than scanning, since for retrieving a large fraction of a table a scan can be better (4, 7). The practical conclusion is that indexing is essential to making database search scale to large tables, applied judiciously where its benefit to queries justifies its cost to updates and storage, the main caveats being that the right indexes depend on the workload and that the figure illustrates the scaling rather than the performance of a specific database.

LIMITATIONS

The following limitations apply to this study:

- The right indexes depend on the workload, the queries, and the pattern of updates.
- The figure illustrates the scaling rather than the performance of a specific database.
- Indexes cost storage and slow updates, which were discussed but not quantified.
- For retrieving a large fraction of a table, a scan can be faster than using an index.

FUTURE SCOPE

Future work could extend the present study as follows:

- Measurement of search time with and without indexes on real databases and workloads.
- Study of the cost of index maintenance under heavy updates.
- Comparison of index structures, such as balanced trees and hash indexes.
- Selection of indexes to suit a given query workload.
- Examination of query optimisation and when a scan is preferred to an index.

CONCLUSION

Database indexing and query performance and how an index makes searching a large table scalable were examined by examining the search time with and without an index as the table grew. Without an index the search time grew in proportion to the table, since every record was examined, becoming impractical on a very large table, while with an index it grew only logarithmically, staying fast however large the table, so the difference widened enormously as the table grew. The study shows that indexing changes search time from growing in proportion to the table to growing only logarithmically, which is why indexes are essential to database performance and scalability, applied selectively where their benefit to queries outweighs their

cost to updates and storage, with the caveats that the right indexes depend on the workload and that the figure illustrates the scaling rather than the performance of a specific database.

REFERENCES

1. Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377–387.
2. Bayer, R., & McCreight, E. (1972). Organization and maintenance of large ordered indices. *Acta Informatica*, 1(3), 173–189.
3. Comer, D. (1979). The ubiquitous B-tree. *ACM Computing Surveys*, 11(2), 121–137.
4. Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., & Price, T. G. (1979). Access path selection in a relational database management system. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 23–34.
5. Knuth, D. E. (1998). *The art of computer programming, volume 3: Sorting and searching* (2nd ed.). Addison-Wesley.
6. Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems* (3rd ed.). McGraw-Hill.
7. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database system concepts* (7th ed.). McGraw-Hill.
8. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson.
9. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database systems: The complete book* (2nd ed.). Pearson.
10. Graefe, G. (2011). Modern B-tree techniques. *Foundations and Trends in Databases*, 3(4), 203–402.

***Author for Correspondence**

Devvrat R. Tripathi

E-mail: devvrat.tripathi555@gmail.com

¹Associate Professor, Department of Information Technology, Rohilkhand Institute of Technology, Bareilly, Uttar Pradesh

²Assistant Professor, Department of Computer Science and Engineering, Rohilkhand Institute of Technology, Bareilly, Uttar Pradesh

³Research Scholar, Department of Information Technology, Dr. Rammanohar Lohia Avadh University, Ayodhya

⁴Research Scholar, Department of Computer Applications, Dr. Rammanohar Lohia Avadh University, Ayodhya

⁵Lecturer, Department of Computer Science and Engineering, Dr. Rammanohar Lohia Avadh University, Ayodhya

Received Date: December 08, 2025

Accepted Date: January 10, 2026

Published Date: Janaury 25, 2026

Citation: Devvrat R. Tripathi, Shalini K. Dwivedi, Aniruddh V. Shukla, Meghna M. Awasthi, Yatharth T. Dixit. Database Indexing and Query Performance: How Indexes Make Searching Large Tables Scalable. Journal of Research in Information Technology and Its Applications. 2026; 11(1): 1-9p.