How To Index Your Database

Baron Schwartz • PostgresConf 2018

Logistics and Stuff

- Ask questions anytime
- Write me baron@vividcortex.com
- Tweet me at @xaprb
- Slides at xaprb.com/talks/

Introduction and Agenda

The purpose of this talk is to organize and understand the principles of database indexing.

- What are indexes?
- What kinds are there?
- How do they work?
- What are the three purposes of an index?
- What are the six ways to best design and use indexes?

What Are Indexes?

Indexes Help Find Data

Indexes are fast-lookup structures for the data in a table.

They essentially do two things with the data:

1. Maintain a **search-optimized copy** of the data.

Indexes Help Find Data

Indexes are fast-lookup structures for the data in a table.

They essentially do two things with the data:

1. Maintain a **search-optimized copy** of the data.
2. Point to the data's **original location**.

How A Query Finds Rows In A Table

```
SELECT c FROM t WHERE b < 70;
```

This query must **examine every row** to find the right ones.

table t

a	b	c	d
33	83	84	11
29	46	81	37
11	46	30	71
59	46	9	43
18	84	52	74
91	30	0	33
61	16	91	94
3	17	87	72
30	97	71	64
70	66	82	8
86	80	68	1
64	86	88	49
23	46	67	15
26	77	72	8
46	56	10	29
9	57	45	43

How A Query Finds Rows With An Index

Indexes **help find rows** without full-scans.

- This is an index of columns (b, d).
- The index is a **sorted copy of those columns**.
- The query scans the index until `b >= 70`.

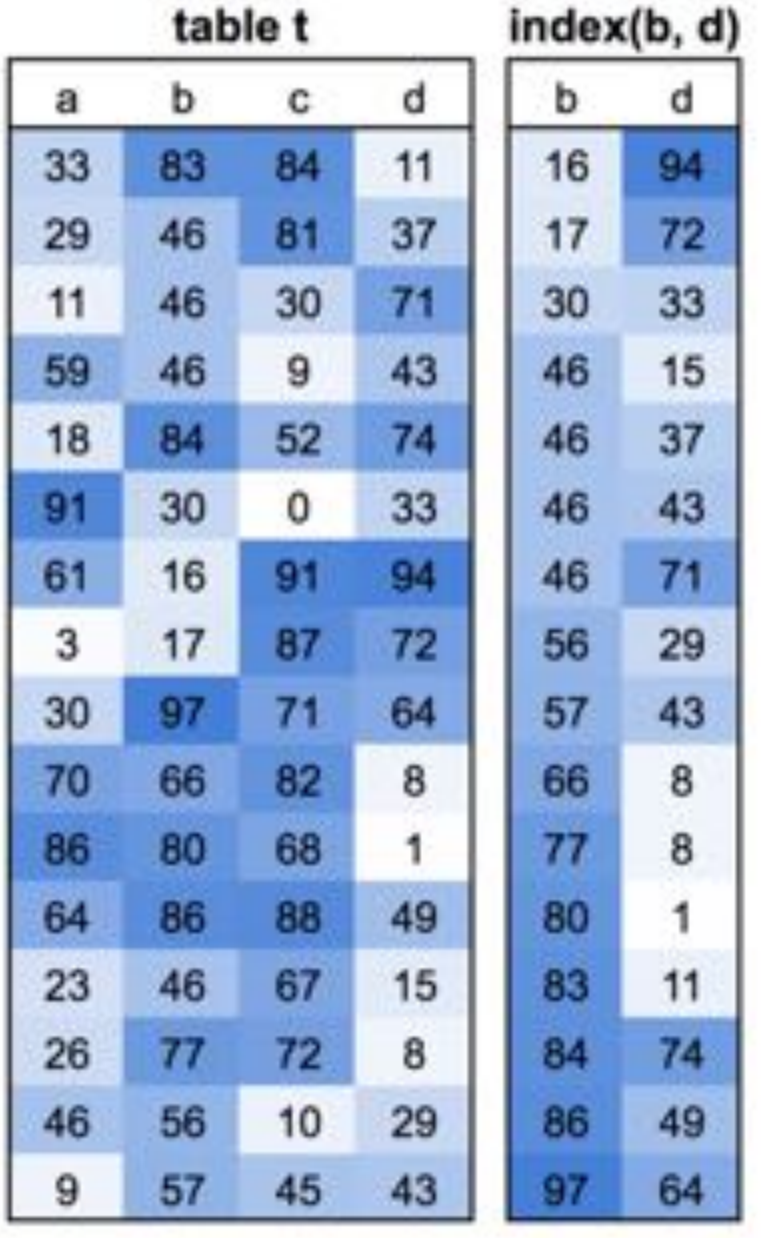


table t

a	b	c	d
33	83	84	11
29	46	81	37
11	46	30	71
59	46	9	43
18	84	52	74
91	30	0	33
61	16	91	94
3	17	87	72
30	97	71	64
70	66	82	8
86	80	68	1
64	86	88	49
23	46	67	15
26	77	72	8
46	56	10	29
9	57	45	43

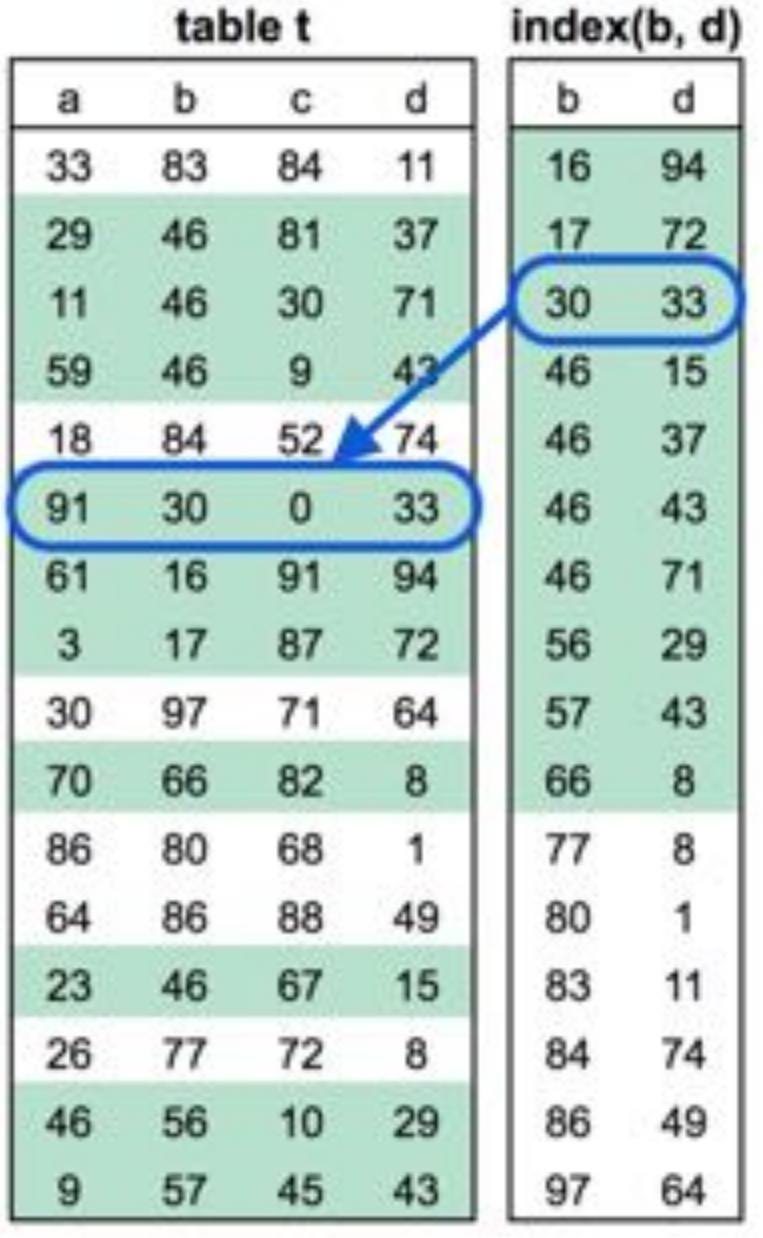
index(b, d)

b	d
16	94
17	72
30	33
46	15
46	37
46	43
46	71
56	29
57	43
66	8
77	8
80	1
83	11
84	74
86	49
97	64

How A Query Finds Rows With An Index

Indexes **help find rows** without full-scans.

- This is an index of columns (b, d).
- The index is a **sorted copy of those columns**.
- The query scans the index until `b >= 70`.
- The index has **pointers to the rows.**



a	b	c	d
33	83	84	11
29	46	81	37
11	46	30	71
59	46	9	42
18	84	52	74
91	30	0	33
61	16	91	94
3	17	87	72
30	97	71	64
70	66	82	8
86	80	68	1
64	86	88	49
23	46	67	15
26	77	72	8
46	56	10	29
9	57	45	43

b	d
16	94
17	72
30	33
46	15
46	37
46	43
46	71
56	29
57	43
66	8
77	8
80	1
83	11
84	74
86	49
97	64

What About Starting From 70?

The index is more than a plain copy. It’s organized in **seekable ranges.**

What About Starting From 70?

The index is more than a plain copy. It’s organized in **seekable ranges.**

That lets the database **seek to a starting point** in the index.

That’s All You Need To Know

Just remember an index is a **sorted, searchable** copy of data.

That’s All You Need To Know

Just remember an index is a **sorted, searchable** copy of data.

- You don’t need to know about **B-Tree Algorithms**.
- You don’t need to know **data structures**.
- You don’t need to understand $O(\log_{\frac{b}{k}}(n))$.

B-Trees For The Curious

Most databases default to B-tree indexes. B-trees have sorted leaf nodes. They let the database:

- Find single rows
- Find ranges of rows
- Retrieve rows in sort order.

Other Kinds Of Indexes

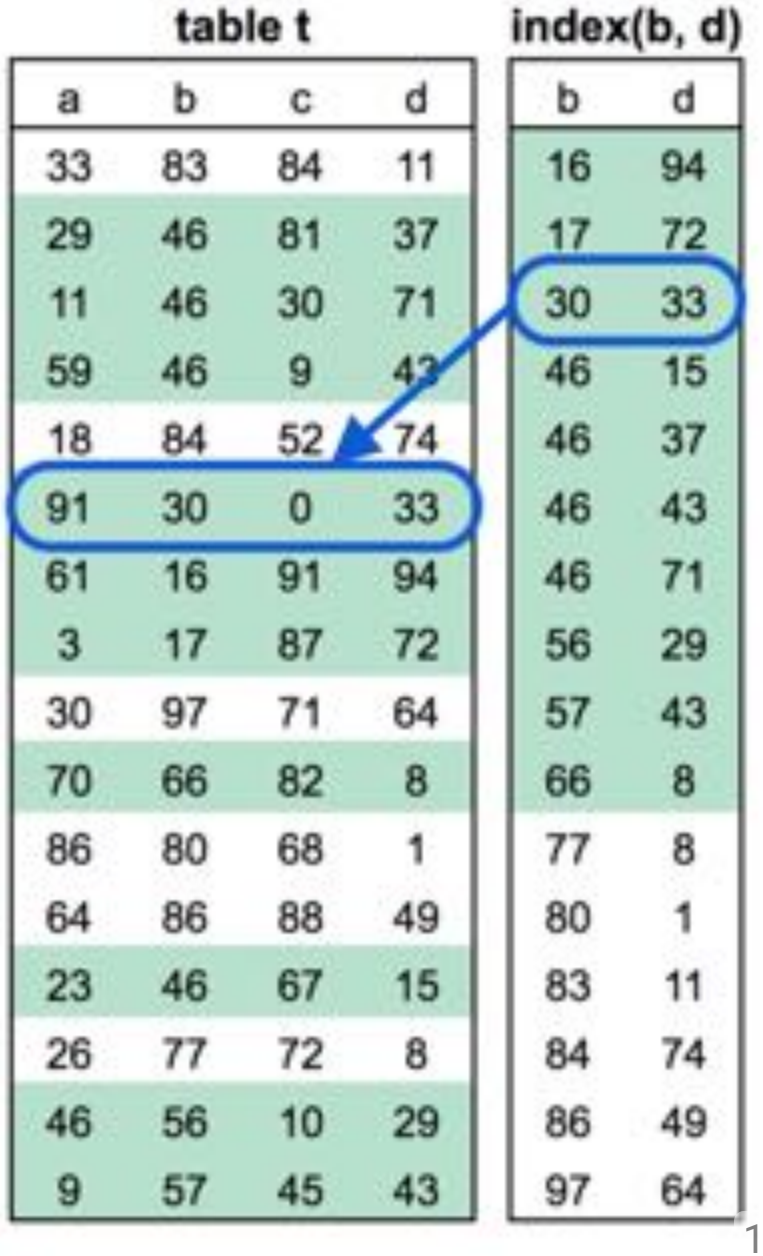
There are special-purpose indexes for special purposes. Study them when you need them.

- Hash indexes
- Log-Structured Merge indexes
- Full-text search (inverted) indexes
- Geospatial indexes
- Block-range indexes
- Bitmap indexes

Three Data Access Rules

Three Data Access Rules

1. Reading a range of data in order is fast.
2. Reading a range out of order is slow.
3. A single-row retrieval or lookup is slow.

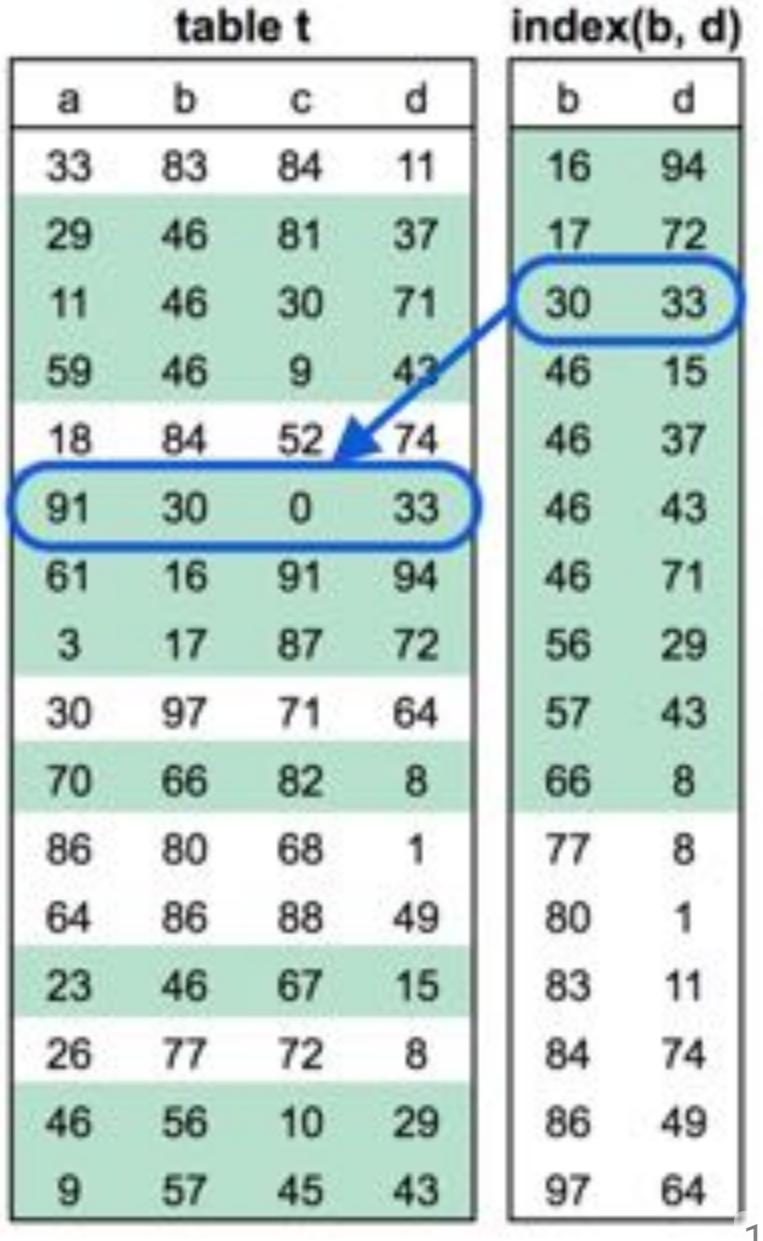


a	b	c	d
33	83	84	11
29	46	81	37
11	46	30	71
59	46	9	42
18	84	52	74
91	30	0	33
61	16	91	94
3	17	87	72
30	97	71	64
70	66	82	8
86	80	68	1
64	86	88	49
23	46	67	15
26	77	72	8
46	56	10	29
9	57	45	43

b	d
16	94
17	72
30	33
46	15
46	37
46	43
46	71
56	29
57	43
66	8
77	8
80	1
83	11
84	74
86	49
97	64

Three Data Access Rules

1. Reading a range of data in order is fast.
   - Scanning the index for `b < 70` is a range.
2. Reading a range out of order is slow.
3. A single-row retrieval or lookup is slow.
   - Finding each row in the table is a lookup.



Three Purposes Of An Index

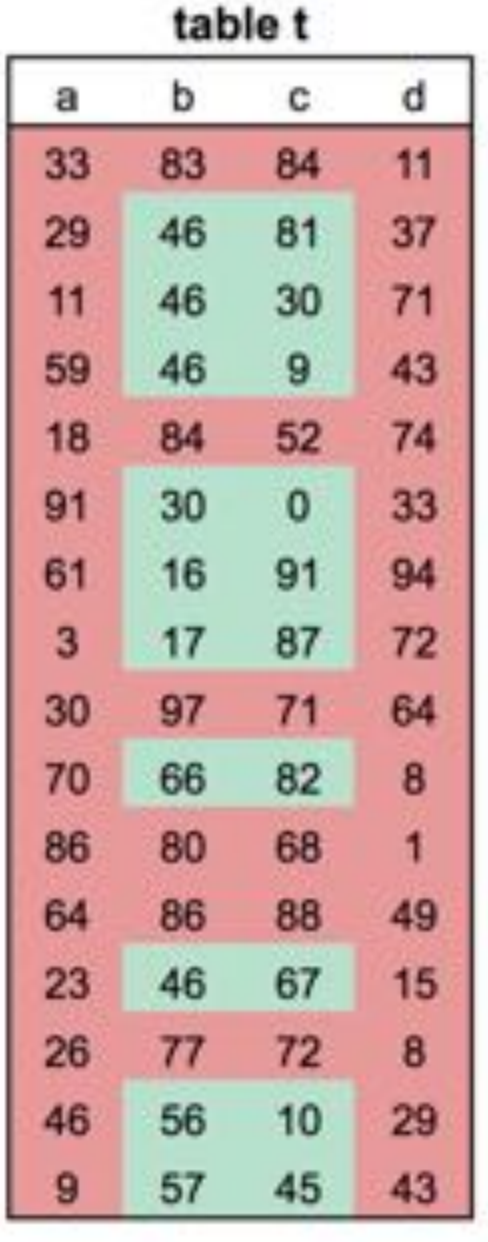
How Do Indexes Help?

1. Read less data.
2. Read data in bulk.
3. Read data presorted.

1. Read Less Data

```
SELECT c FROM t WHERE b < 70;
```

Without an index, this is a full table scan that reads **all rows and all columns**.



1. Read Less Data

```
SELECT c FROM t WHERE b < 70;
```

With an index, it reads only matching rows.

Three inefficiencies:

- It reads extra columns
- It reads from the table in random order
- It reads from the table row-by-row

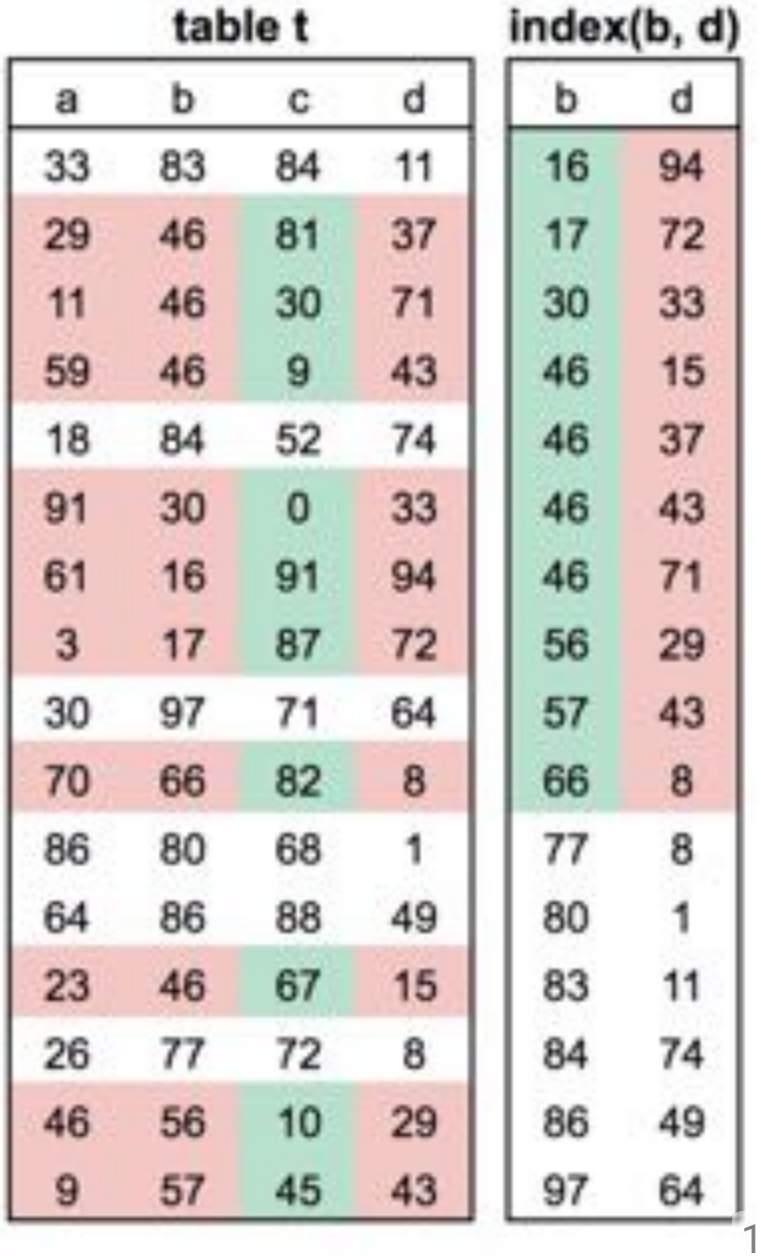


table t

a	b	c	d
33	83	84	11
29	46	81	37
11	46	30	71
59	46	9	43
18	84	52	74
91	30	0	33
61	16	91	94
3	17	87	72
30	97	71	64
70	66	82	8
86	80	68	1
64	86	88	49
23	46	67	15
26	77	72	8
46	56	10	29
9	57	45	43

index(b, d)

b	d
16	94
17	72
30	33
46	15
46	37
46	43
46	71
56	29
57	43
66	8
77	8
80	1
83	11
84	74
86	49
97	64

1. Read Less Data

Create an index with **all columns mentioned**:

```
index(b,c)
```

```
SELECT c FROM t WHERE b > 70;
```

Now the index “covers” the query and it **doesn’t access** the table at all!

- No row-by-row lookups
- No randomly ordered access

table t				index(b, c)	
a	b	c	d	b	c
33	83	84	11	16	91
29	46	81	37	17	87
11	46	30	71	30	0
59	46	9	43	46	9
18	84	52	74	46	30
91	30	0	33	46	67
61	16	91	94	46	81
3	17	87	72	56	10
30	97	71	64	57	45
70	66	82	8	66	82
86	80	68	1	77	72
64	86	88	49	80	68
23	46	67	15	83	84
26	77	72	8	84	52
46	56	10	29	86	88
9	57	45	43	97	71

2. Read Data In Bulk

Indexes are sorted, so logically nearby rows are physically nearby.

Range queries will read pages that are **densely packed** with desired rows.

Densely packed pages let the database:

- Read fewer pages (goal #1)
- Read pages in sequential order, avoiding random reads

2. Read Data In Bulk

To achieve this you can use *index-organized tables* or *clustered indexes*.

- The table itself is **sorted** in physical order
- The search-and-seek structures are built on the table, not separately

table t

a	b	c	d
61	16	91	94
3	17	87	72
91	30	0	33
59	46	9	43
11	46	30	71
23	46	67	15
29	46	81	37
46	56	10	29
9	57	45	43
70	66	82	8
26	77	72	8
86	80	68	1
33	83	84	11
18	84	52	74
64	86	88	49
30	97	71	64

2. Read Data In Bulk

```
SELECT c FROM t WHERE b < 70;
```

The query scans a range from the table only.

- Bulk access
- Ordered access
- Some superfluous columns

table t

a	b	c	d
61	16	91	94
3	17	87	72
91	30	0	33
59	46	9	43
11	46	30	71
23	46	67	15
29	46	81	37
46	56	10	29
9	57	45	43
70	66	82	8
26	77	72	8
86	80	68	1
33	83	84	11
18	84	52	74
64	86	88	49
30	97	71	64

3. Read Data Presorted

Indexes are sorted, so the database doesn’t need to sort.

This helps optimize queries such as:

- `ORDER BY`
- `GROUP BY`
- `DISTINCT`
- `MIN` and `MAX`

The Three-Star System

Grade an index with three stars, one for each:

- A star if the rows are densely packed.
- A star if the rows are sorted.
- A star if the query doesn’t access the table.

The **third star** is often much more important!

Six Ways To Optimize Index Usage

1. Don’t Defeat Indexes

This query **defeats** an index on column `a`:

```
... WHERE NOW() > a + INTERVAL 30 DAY;
```

This query can use the index:

```
... WHERE a > NOW() - INTERVAL 30 DAY;
```

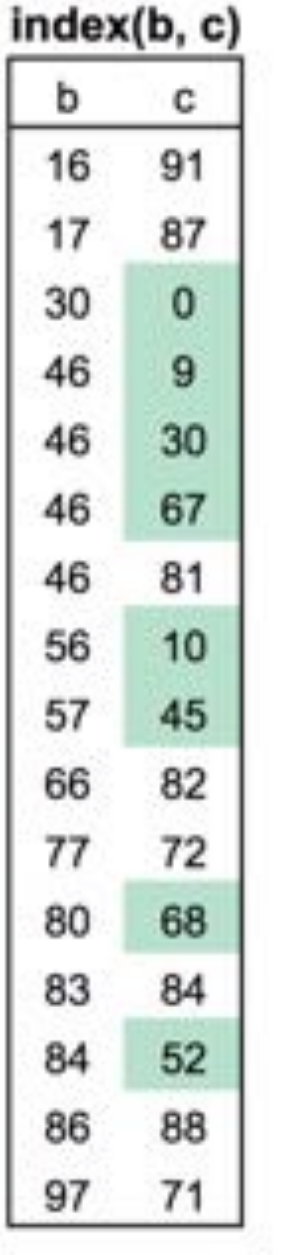
You can seek/search for a value in an index, but not an expression.

2. The Left-Prefix Rule

Multi-column indexes are sorted by column 1, then column 2, etc.

- Queries can use a **prefix of an index**.
- They generally **can't use a suffix**.

`WHERE c < 70` won't work, it will be a full index scan.



2. The Left-Prefix Rule

A prefix specifies a range up to and including the **first inequality**.

```
WHERE b < 70 AND c < 50
```

Anything past the first inequality is sub-filtering the range: no longer bulk access.

- Place equalities first, ranges last
- Try supplying missing equalities or converting ranges to lists of values

index(b, c)

b	c
16	91
17	87
30	0
46	9
46	30
46	67
46	81
56	10
57	45
66	82
77	72
80	68
83	84
84	52
86	88
97	71

3. Exploit Index-Only Queries

Create *covering indexes* for important queries (index-only queries).

Remember: it works only if the index has **all** the columns the query mentions.

It’s often possible to union the indexes needed by several important queries.

4. Exploit Clustered Indexes

The benefit of a clustered index is the table is physically organized in PK order.

- Supported in Oracle, MySQL, SQL Server, IBM DB2 LUW
- Not supported in PostgreSQL, despite `CLUSTER` command (~ `DEFRAG`)

There can be **only one** clustered index per table.

5. Consider Column Selectivity And Order

Column order in compound indexes matters a lot!

- For queries:
  - The leftmost prefix rule applies
  - Makes indexes useful for queries, or not
- For data characteristics:
  - Organize by most-selective or least-selective
  - The best order depends on use-case: least for bulk reads, most for single-rows.

6. Avoid Over-Indexing...

Indexes add cost to writes and complicate the planner’s job

- Avoid duplicates
- Analyze redundant indexes with a common prefix
- Analyze unused indexes

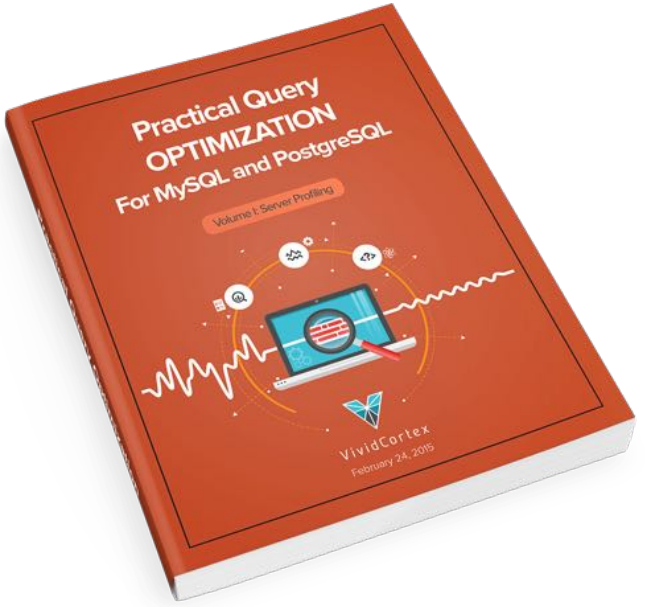
6. But Don’t Fear Indexes

Caution: “unused” indexes often really aren’t!

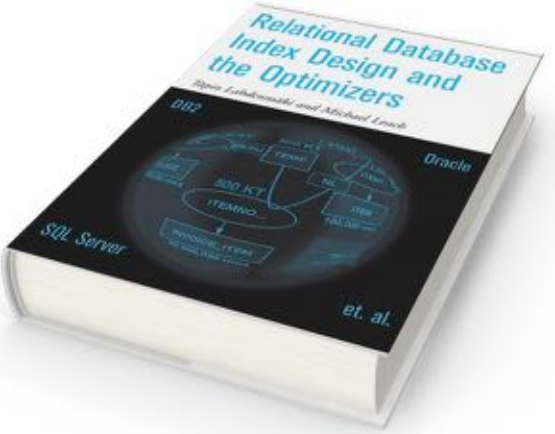
Indexes are a lot less expensive than you’d guess. Cost/benefit tradeoffs usually weigh in favor of indexes.

For a rigorous analysis, see Lahdenmaki and Leach’s book.

Resources


Practical Query
OPTIMIZATION
For MySQL and PostgreSQL
Volume I: Server Profiling
VividCortex
February 24, 2015


SQL
PERFORMANCE
EXPLAINED


Relational Database
Index Design and
the Optimizers
DB2
Oracle
SQL Server
et. al.

Slides and Contact Information

Slides are at https://www.xaprb.com/talks/ or you can scan the QR code.

Contact: @xaprb and baron@vividcortex.com

Photo Credits

- Dragon’s Blood Tree
- Keys