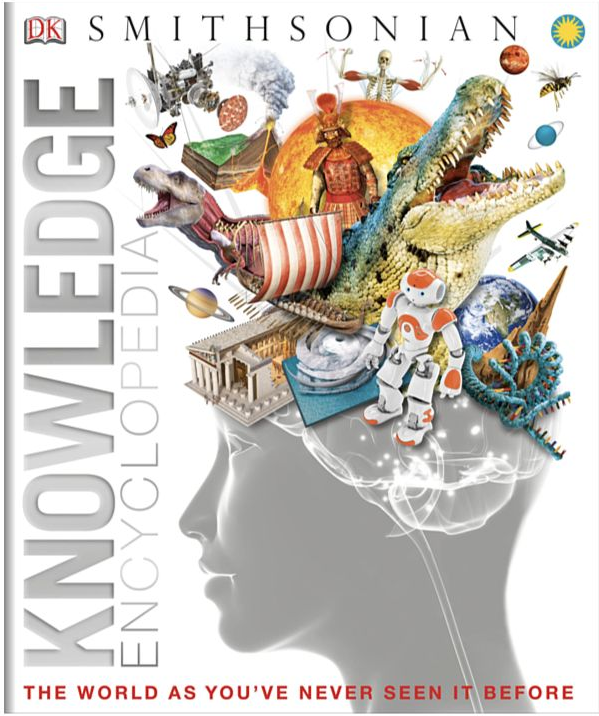
Indexes

Why use indexes?

Encyclopedias use index to refer keywords to pages with related content

Scanning for term without index requires scanning entire book

This is not an efficient task!



Why use indexes?

Encyclopedia uses index to refer keywords to pages with related content

Scanning is **O(n)** complexity (linear), where **n** = number of pages

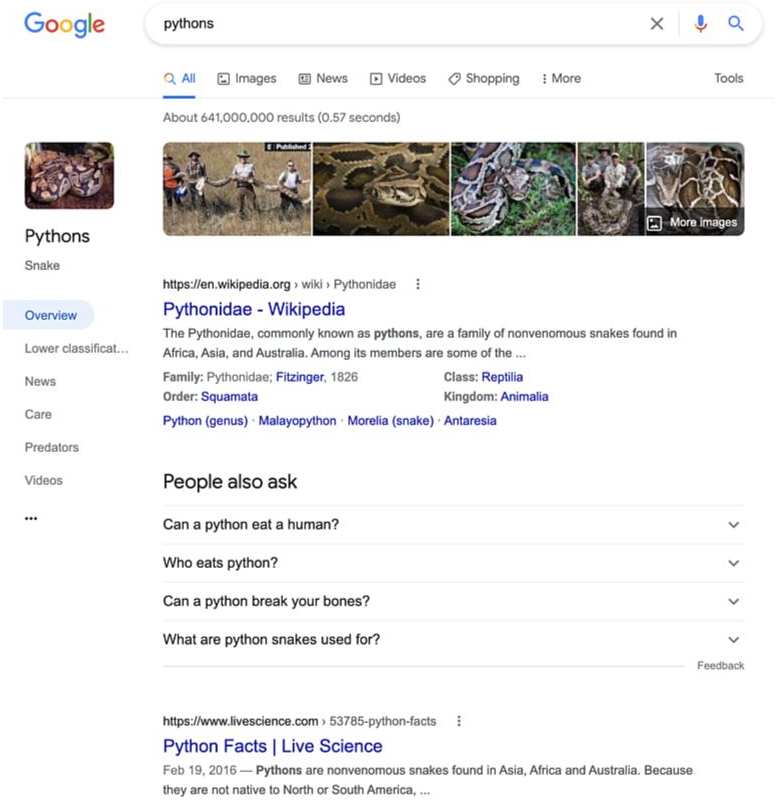
If the encyclopedia has 345 pages, and the index for "snakes" shows 5 pages, we've reduced **n** from 345 to 5!

Google search

Google does not scan entire internet for every search request

Instead, builds index ahead of time

Index points keywords to relevant webpages



Indexing in Postgres

Improves query time for specific columns

Each index entry contains column value that can be thought of as "search term" (e.g. "snakes")

Also contains reference to its corresponding row (usually by primary key)

Indexing in Postgres

magic_markers table

id	color	...	...	...	...	...	...	...	...	...
1	blue	...	...	...	...	...	...	...	...	...
2	green	...	...	...	...	...	...	...	...	...
3	blue	...	...	...	...	...	...	...	...	...
4	red	...	...	...	...	...	...	...	...	...
5	blue	...	...	...	...	...	...	...	...	...
6	red	...	...	...	...	...	...	...	...	...
7	yellow	...	...	...	...	...	...	...	...	...

Indexing in Postgres

Index on magic_markers color column

Column Value	Record ID
blue	3
blue	1
blue	5
green	2
red	4
red	6
yellow	7

Index example use case

```
CREATE TABLE people (
    id SERIAL PRIMARY KEY,
    full_name TEXT NOT NULL,
    birthdate DATE DEFAULT now()
);
```

Assume 100,000,000 rows

Index example use case


```sql
SELECT full_name FROM people WHERE birthdate = '1912-06-23';
SELECT full_name FROM people WHERE birthdate = '1903-12-28';
SELECT full_name FROM people WHERE birthdate = '1916-04-30';
```

Scans 300,000,000 rows (no index)

Index syntax


```sql
CREATE INDEX index_name ON table_name (column_name);
```

Postgres starts building index structure

While building index, table is in read-only mode (SELECT only)

Once index is built, read-only mode is turned off

Postgres updates index as table values change

Index example use case


```sql
CREATE TABLE people (
    id SERIAL PRIMARY KEY,
    full_name TEXT NOT NULL,
    birthdate DATE DEFAULT now()
);
```

```sql
CREATE INDEX people_birthdate_index ON people (birthdate);
```

```sql
SELECT full_name FROM people WHERE birthdate = '1912-06-23';
SELECT full_name FROM people WHERE birthdate = '1903-12-28';
SELECT full_name FROM people WHERE birthdate = '1916-04-30';
```

Index data structures

Q: Why do indexes have same size as table but faster lookup time?

A: Data structures optimized for lookup

Index on magic_markers color column

O(log n)

Column Value	Record ID
blue	3
blue	1
blue	5
green	2
red	4
red	6
yellow	7

magic_markers table

O(n)

id	color	...	...	...	...	...	...	...	...	...
1	blue	...	...	...	...	...	...	...	...	...
2	green	...	...	...	...	...	...	...	...	...
3	blue	...	...	...	...	...	...	...	...	...
4	red	...	...	...	...	...	...	...	...	...
5	blue	...	...	...	...	...	...	...	...	...
6	red	...	...	...	...	...	...	...	...	...
7	yellow	...	...	...	...	...	...	...	...	...

Index data structures

B-Tree Index	Hash Index
Ordered	Not ordered
Used for matching with <, <=, >=, >, or =	Used only for matching with =
O(log n) lookup	O(1) lookup
O(log n) insertion	O(1) insertion
O(log n) deletion	O(1) deletion

Q: How can we index on birthdate to optimize this query?

```
SELECT full_name FROM people WHERE birthdate = '1916-04-30';
```

Index data structures

B-Tree Index	Hash Index
Ordered	Not ordered
Used for matching with <, <=, >=, >, or =	Used only for matching with =
O(log n) lookup	O(1) lookup
O(log n) insertion	O(1) insertion
O(log n) deletion	O(1) deletion

Q: How can we index on birthdate to optimize this query?

```
SELECT full_name FROM people WHERE birthdate = '1916-04-30';
```

A: Either, but Hash index is preferred

Index data structures

B-Tree Index	Hash Index
Ordered	Not ordered
Used for matching with <, <=, >=, >, or =	Used only for matching with =
O(log n) lookup	O(1) lookup
O(log n) insertion	O(1) insertion
O(log n) deletion	O(1) deletion

Q: How can we index on birthdate to optimize this query?

```
SELECT full_name FROM people WHERE birthdate > '1916-04-30';
```

A: Must use B-Tree index

Indexes: Final tips

Common way to enhance database performance

Tradeoffs:

1. Indexes take up a lot of space
2. Indexes must stay up to date

Rule of thumb: Only index columns that are queried frequently

PostgreSQL automatically creates an index for each table's primary key