Index

Let's talk about indexes in databases in a simple way.

An index in a database is like a catalog in a library. Imagine you walk into a library filled with thousands of books. If you wanted to find a specific book, you could wander through the aisles, looking at each shelf one by one. That would take a long time! Instead, you can use the library's catalog, which lists all the books by title, author, and other details. This catalog helps you quickly find the book you want without searching through every shelf. Similarly, an index in a database helps you quickly find specific rows in a table without having to look through every single row.

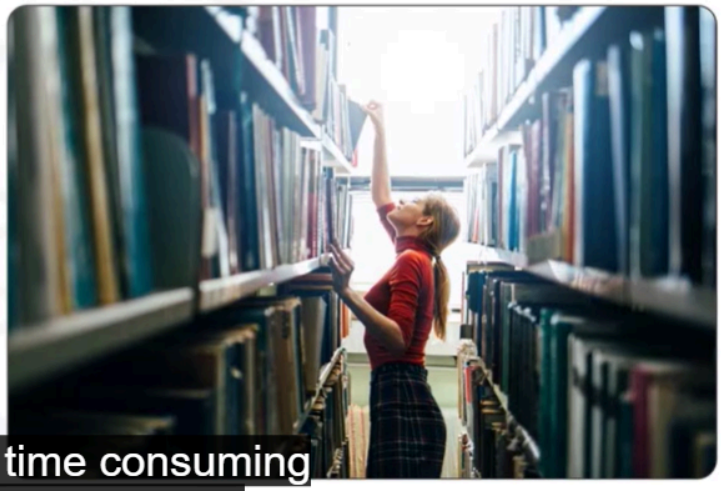
When you create an index on a database table, it organizes the data in a way that makes it faster to retrieve information. For example, if you have a table of customers and you create an index on their names, the database can quickly find all the information related to a specific customer without scanning the entire table. This makes searching much faster and more efficient!

Introduction

Library with an extensive collection of books

Go through each shelf one by one

Time-consuming and inefficient approach



- Imagine you have a library with a collection of books. If you want to find a specific book, you could go through each shelf individually until you find it. However, this approach can be quite time-consuming and inefficient.

Catalog

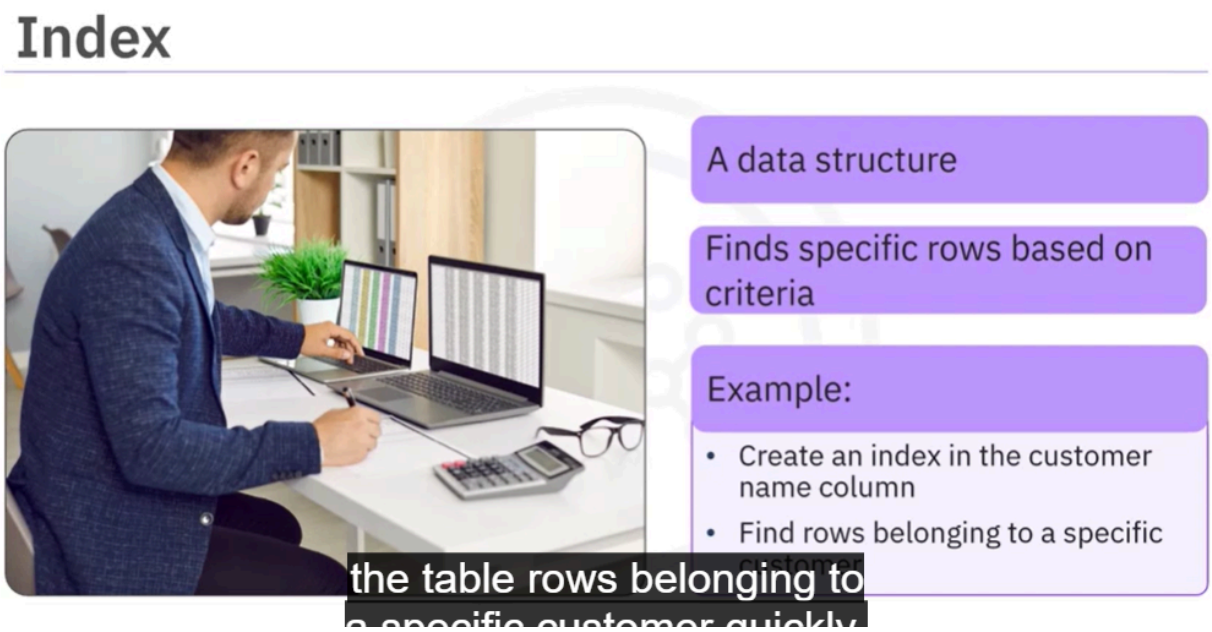
Offers an index listing all the books

Lists each book by relevant information

Helps locate the book

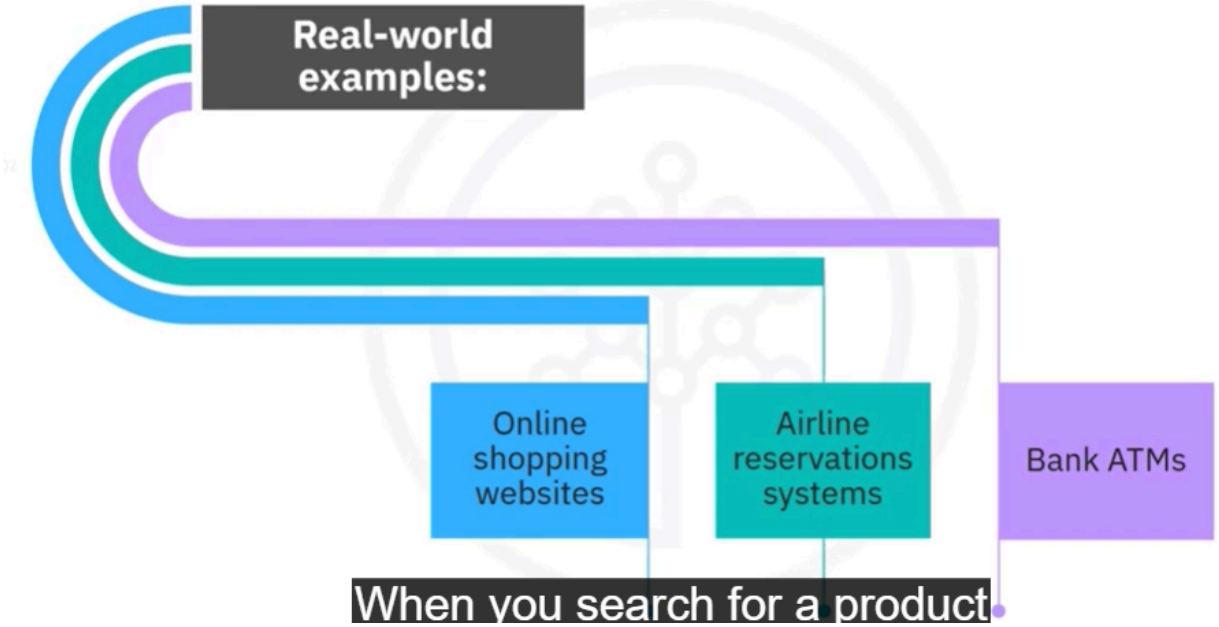
- A better way to locate the book would be to use the library's catalog. The catalog is an index of all the books in the library, and it lists each book by its

title, author, and other relevant information. When you use the catalog to find a book, you navigate through the index to locate its entry.



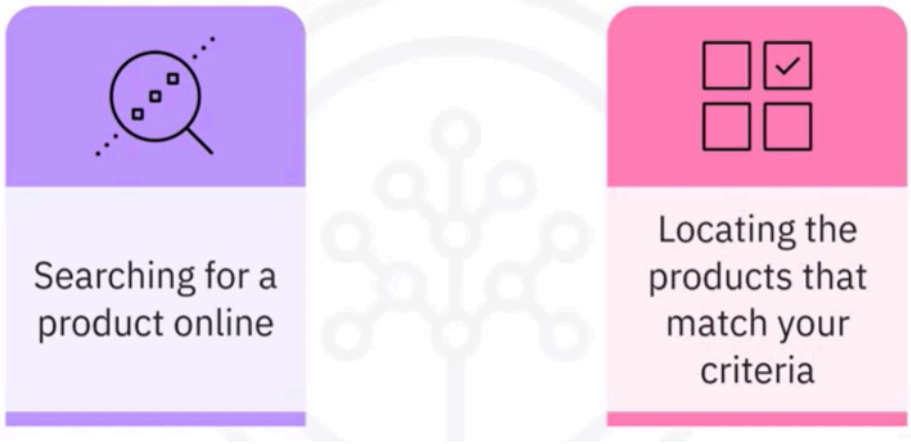
- Indexing a table in a database works similarly. An index is a data structure that allows you to find specific rows in a table based on certain criteria quickly. For example, if you have a customer data table, you could create an index in the customer's name. Column indexing would allow you to find all the table rows belonging to a specific customer quickly

Using Indexes



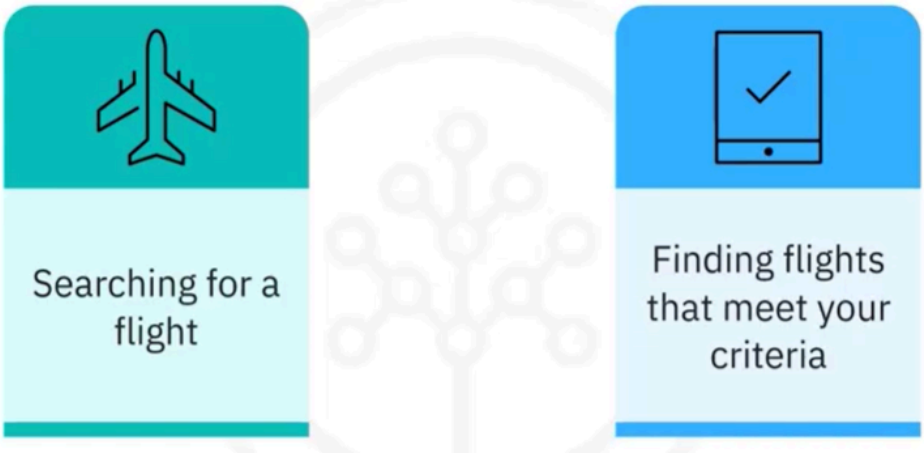
- Here are some real world examples of how databases use indexing. Some of these examples include online shopping websites, airline reservation systems, and bank ATM's.

Online shopping websites



- When you search for a product on an online shopping website, the website uses indexes to locate products that align with your search criteria efficiently.

Airline reservations systems



- When you search for a flight on an airline website, the website uses indexes to find the flights that meet your criteria quickly.

Bank ATMs

- Lastly, when you withdraw money from an ATM, the ATM uses indexes to retrieve your account information quickly

Headers use colons: not dashes

- Let us now explore how indexing is essential for optimizing database performance. Indexes can significantly improve the performance database queries. They allow the database to quickly find the relevant rows to the query without scanning the entire table.

Adding data to a table

book_id	title	total_pages	rating	isbn	pub_date	pub_id
101	Lean Software Development	240	4.17	978032	2003-05-18	5
201	Facing the Intelligence Explosion	91	3.87		2013-02-01	109
301	Scala in Action	419	3.74	978194	2013-04-10	111
401	Patterns of Software	256	3.84	978020	1996-08-15	150
250	Anatomy of LISP	446	4.43	978007	1978-01-01	119

Adding data to a table

book_id	title	total_pages	rating	isbn	pub_date	pub_id
101	Lean Software Development	240	4.17	978032	2003-05-18	5
201	Facing the Intelligence Explosion	91	3.87		2013-02-01	109
301	Scala in Action	419	3.74	978194	2013-04-10	111
401	Patterns of Software	256	3.84	978020	1996-08-15	150
250	Anatomy of LISP	446	4.43	978007	1978-01-01	119

- Specify sort order in the SELECT statement
- Return rows in a particular order
- Create an index on a table

Adding data to a table

book_id	title	total_pages	rating	isbn	pub_date	pub_id
101	Lean Software Development	240	4.17	978032	2003-05-18	5
201	Facing the Intelligence Explosion	91	3.87		2013-02-01	109
301	Scala in Action	419	3.74	978194	2013-04-10	111
401	Patterns of Software	256	3.84	978020	1996-08-15	150
250	Anatomy of LISP	446	4.43	978007	1978-01-01	119

Adding data to a table

book_id	title	total_pages	rating	isbn	pub_date	pub_id
101	Lean Software Development	240	4.17	978032	2003-05-18	5
201	Facing the Intelligence Explosion	91	3.87		2013-02-01	109
301	Scala in Action	419	3.74	978194	2013-04-10	111
401	Patterns of Software	256	3.84	978020	1996-08-15	150
250	Anatomy of LISP	446	4.43	978007	1978-01-01	119

- Appended to the end
- No inherent order to data

Adding data to a table

book_id	title	total_pages	rating	isbn	pub_date	pub_id
101	Lean Software Development	240	4.17	978032	2003-05-18	5
201	Facing the Intelligence Explosion	91	3.87		2013-02-01	109
301	Scala in Action	419	3.74	978194	2013-04-10	111
401	Patterns of Software	256	3.84	978020	1996-08-15	150
250	Anatomy of LISP	446	4.43	978007	1978-01-01	119

- Usually, when you add data to a table, it is added to the end of the table. However, this action has no guarantee or inherent order to the data. When you select a particular row from that table, the processor must check each row until it finds the one you want. On a large table, it can significantly slow down the process of locating a specific row. Also, when you select multiple rows, the result may lack a specific order unless you specify a sort order in your select statement. As you might want to return the rows in a particular order or select a subset of sequential rows, you can create an index on a table to locate the row or set of rows that you need quickly.

Functioning of an index

book_id	pointer
101	p1
201	p2
205	p5
301	p4
401	p3

pointer	book_id	title	total_pages	...
p1	101	...	...	...
p2	201	...	...	...
p3	301	...	...	...
p4	401	...	...	...
p5	250	...	...	...

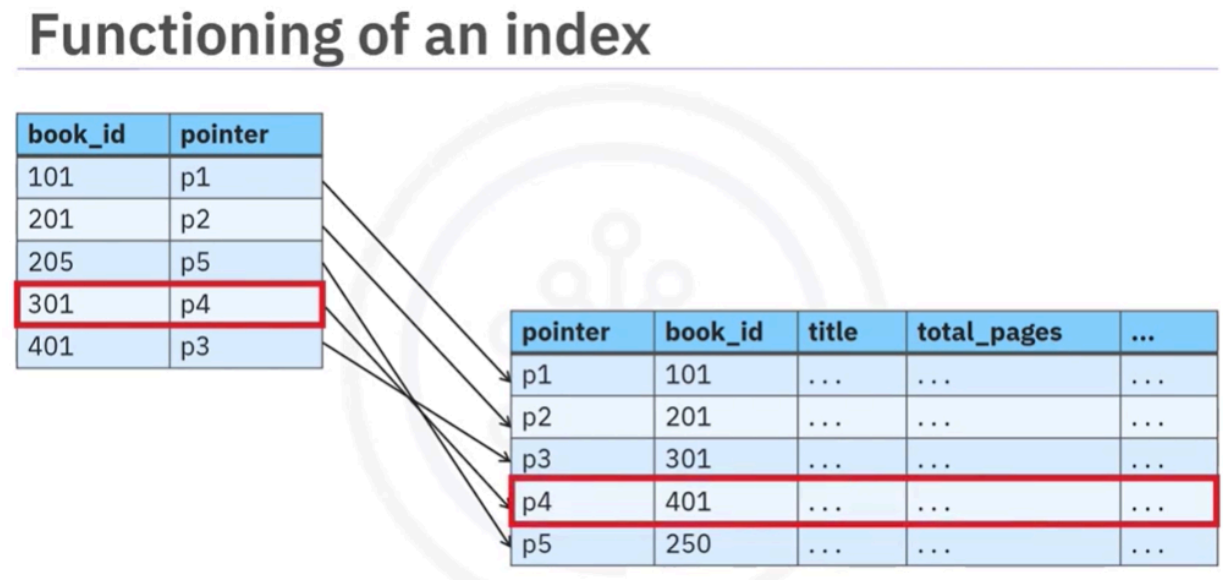
Let us now explore

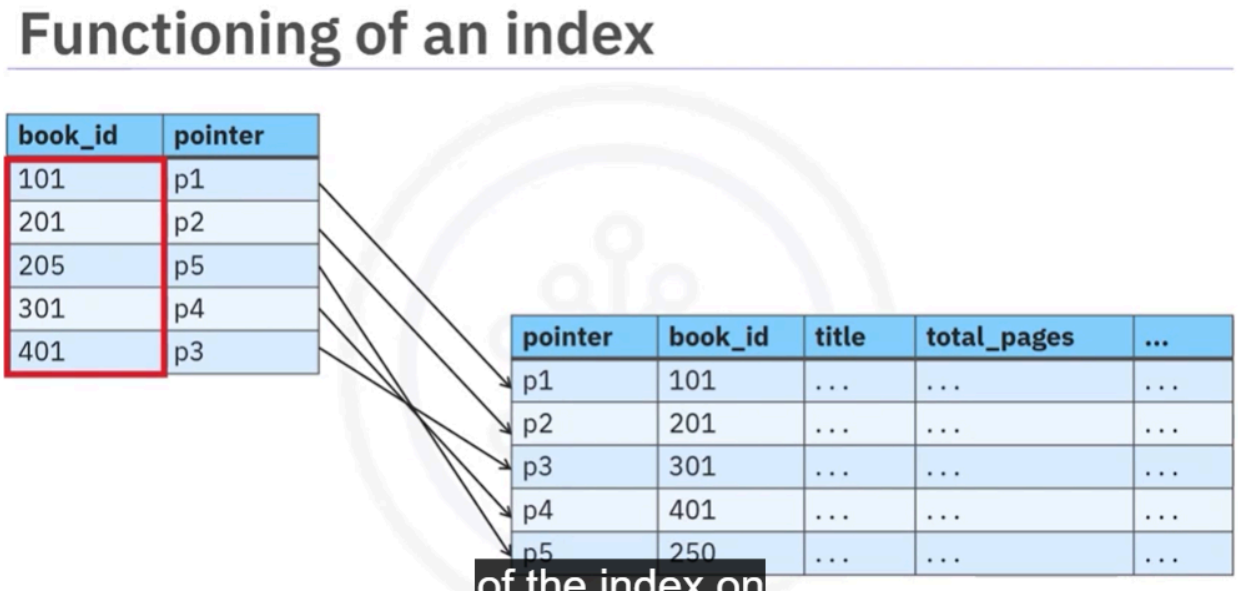
Functioning of an index

book_id	pointer
101	p1
201	p2
205	p5
301	p4
401	p3

pointer	book_id	title	total_pages	...
p1	101	...	...	...
p2	201	...	...	...
p3	301	...	...	...
p4	401	...	...	...
p5	250	...	...	...

pointers to each





- Let us now explore how an index works. An index works by storing pointers to each row in the table, so that when you request a particular row, the SQL processor can use the index to locate the row quickly. It is similar to how you use the index in a book to find a particular section quickly. The unique key dictates the order of the index on which values rely.

Creating an index

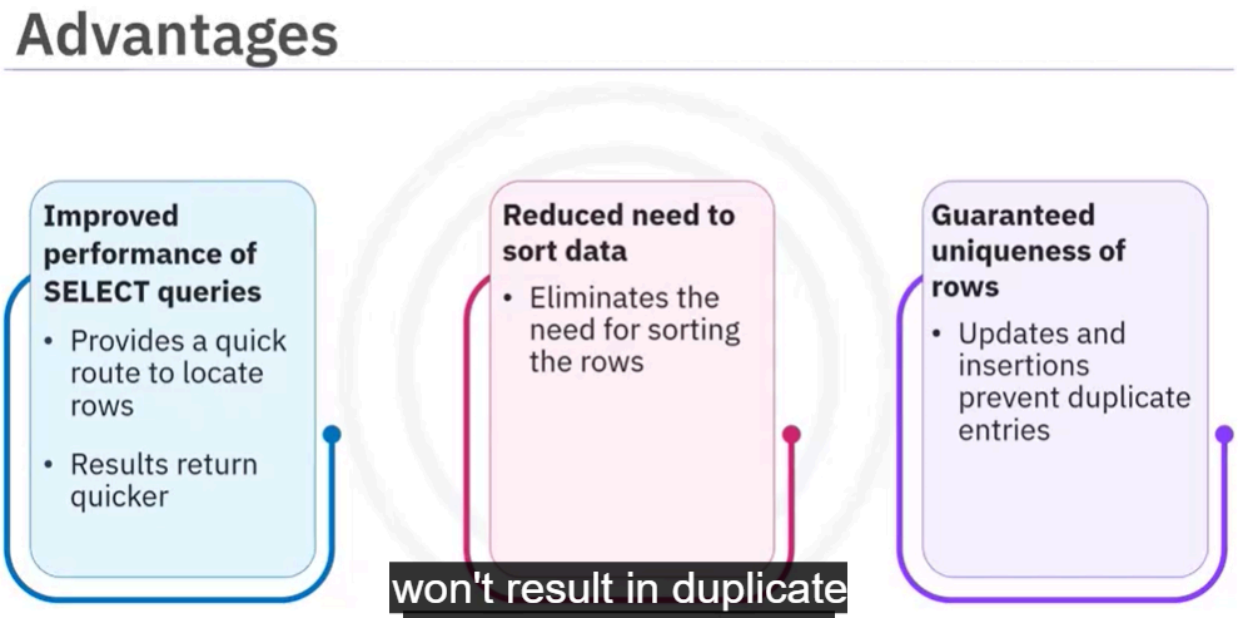
```
CREATE TABLE book(
   . . .
   PRIMARY KEY(book_id));
```

```
CREATE UNIQUE INDEX unique_book_id
   ON book(book_id);
```

- By default, when you create a primary key on a table, the system automatically generates an index on that key. However, you can also create your indexes on regularly searched columns. Use the create index statement to define the index, specifying the index name, its uniqueness, and the table and column on which to base it.

Advantages

- Indexes provide the database user with many benefits. These include improved performance of select, reduced need to sort data, and guaranteed uniqueness of rows.



- Improved performance of select queries when searching on an indexed column. Because the index offers a quick route to locate rows matching the search term, it returns results faster than when it needs to check every row in the table. Reduced need to sort data. If you constantly need rows in a specific order, employing an index can eliminate the necessity to sort the rows after their retrieval. Guaranteed uniqueness of rows. If you use the unique clause when you create the index, you ensure that updates and insertions won't result in duplicate entries in that column

Disadvantages

- There are, however, a few disadvantages of indexes as well. These include use of disk space. Each index you create uses disc space in the same way that adding indexes increases the number of pages in a book. Decreased performance of insert, update, and delete queries. Because the rows in an indexed table are sorted according to the index, adding or removing rows can take longer than in a non-indexed table.

Recap

In this video, you learned that:

- An index is a data structure that allows you to quickly find specific rows in a table based on certain criteria
- Indexing is an essential tool for optimizing database performance
- An index works by storing pointers to each row in the table so that when you request a particular row, the SQL processor can use the index to locate the row quickly
- You should create an index only when you stand to gain more from the advantages than you might lose from the disadvantages