Advantages and disadvantages to using indexes computer science

Put simply, database indexes help speed up retrieval of data. The other great benefit of indexes is that your server doesn’t have to work as hard to get the data. They are much the same as book indexes, providing the database with quick jump points on where to find the full reference (or to find the database row).

There are both advantages and disadvantages to using indexes, however.

One disadvantage is they can take up quite a bit of space ‘ check a textbook or reference guide and you’ll see it takes quite a few pages to include those page references.

Another disadvantage is using too many indexes can actually slow your database down. Thinking of a book again, imagine if every ‘ the’, ‘ and’ or ‘ at’ was included in the index. That would stop the index being useful ‘ the index becomes as big as the text! On top of that, each time a page or database row is updated or removed, the reference or index also has to be updated.

So indexes speed up finding data, but slow down inserting, updating or deleting data.

Some fields are automatically indexed. A primary key or a field marked as ‘ unique’ ‘ for example an email address, a userid or a social security number ‘ are automatically indexed so the database can quickly check to make sure that you’re not going to introduce bad data.

So when should a database field be indexed?

The general rule is anything that is used to limit the number of results you're trying to find.

It's hard to generalise so we'll look at some specific but common examples.

Note ' the database tables shown below are used as an example only and will not necessarily be the best setup for your particular needs.

In a database table that looks like this:

Note: The SQL code shown below works with both MySQL and PostgreSQL databases.

```
CREATE TABLE subscribers (

subscriberid INT PRIMARY KEY,

emailaddress VARCHAR(255),

firstname VARCHAR(255),

lastname VARCHAR(255)

);
```

if we want to quickly find an email address, we create an index on the emailaddress field:

```
CREATE INDEX subscriber_email ON subscribers(emailaddress);
```

' and any time we want to find an email address:

SELECT firstname, lastname FROM subscribers WHERE emailaddress='';

' it will be quite quick to find!

Another reason for creating indexes is for tables that reference other tables. For example, in a CMS you might have a news table that looks something like this:

CREATE TABLE newsitem (

newsid INT PRIMARY KEY,

newstitle VARCHAR(255),

newscontent TEXT,

authorid INT,

newsdate TIMESTAMP

);

and another table for authors:

CREATE TABLE authors (

authorid INT PRIMARY KEY,

username VARCHAR(255),

firstname VARCHAR(255),

lastname VARCHAR(255)

);

A query like this:

SELECT newstitle, firstname, lastname FROM newsitem n, authors a WHERE n. authorid= a. authorid;

‘ will be take advantage of an index on the newsitem authorid:

CREATE INDEX newsitem_authorid ON newsitem(authorid);

This allows the database to very quickly match the records from the ‘ newsitem’ table to the ‘ authors’ table. In database terminology this is called a table join ‘ you should index any fields involved in a table join like this.

Since the ‘ authorid’ in the authors table is a primary key, it is already indexed. The same goes for the ‘ newsid’ in the news table, so we don’t need to look at those cases.

On a side note, table aliases make things a lot easier to see what’s happening. Using ‘ newsitem n’ and ‘ authors a’ means we don’t have to write:

SELECT newstitle, firstname, lastname FROM newsitem, authors WHERE newsitem. authorid= authors. authorid;

for more complicated queries where more tables are referenced this can be extremely helpful and make things really easy to follow.

In a more complicated example, a news item could exist in multiple categories, so in a design like this:

```
CREATE TABLE newsitem (

newsid INT PRIMARY KEY,

newstitle VARCHAR(255),

newscontent TEXT,

authorid INT,

newsdate TIMESTAMP

);

CREATE TABLE newsitem_categories (

newsid INT,

categoryid INT

);

CREATE TABLE categories (

categoryid INT PRIMARY KEY,

categoryname VARCHAR(255)

);
```

This query:

SELECT n. newstitle, c. categoryname FROM categories c,

newsitem_categories nc, newsitem n WHERE c. categoryid= nc. categoryid

AND nc. newsid= n. newsid;

‘ will show all category names and newstitles for each category.

To make this particular query fast we need to check we have an index on:

newsitem newsid

newsitem_categories newsid

newsitem_categories categoryid

categories categoryid

Note: Because the newsitem newsid and the categories categoryid fields are primary keys, they already have indexes.

We need to check there are indexes on the ‘ join’ table ‘

newsitem_categories

This will do it:

CREATE INDEX newscat_news ON newsitem_categories(newsid);

CREATE INDEX newscat_cats ON newsitem_categories(categoryid);

We could create an index like this:

CREATE INDEX news_cats ON newsitem_categories(newsid, categoryid);

However, doing this limits some ways the index can be used. A query against the table that uses both ‘ newsid’ and ‘ categoryid’ will be able to use this index. A query against the table that only gets the ‘ newsid’ will be able to use the index.

A query against that table that only gets the ‘ categoryid’ will not be able to use the index.

For a table like this:

CREATE TABLE example (

a int,

b int,

c int

);

With this index:

CREATE INDEX example_index ON example(a, b, c);

It will be used when you check against ‘ a’.

It will be used when you check against ‘ a’ and ‘ b’.

It will be used when you check against ‘ a’, ‘ b’ and ‘ c’.

It will not be used if you check against ‘ b’ and ‘ c’, or if you only check ‘ b’ or you only check ‘ c’.

It will be used when you check against ‘ a’ and ‘ c’ but only for the ‘ a’ column ‘ it won’t be used to check the ‘ c’ column as well.

A query against ‘ a’ OR ‘ b’ like this:

SELECT a, b, c FROM example where a= 1 OR b= 2;

Will only be able to use the index to check the ‘ a’ column as well ‘ it won’t be able to use it to check the ‘ b’ column.

Multi-column indexes have quite specific uses, so check their use carefully.

Now that we’ve seen when we should use indexes, let’s look at when we shouldn’t use them. They can actually slow down your database (some databases may actually choose to ignore the index if there’s no reason to use it).

A table like this:

CREATE TABLE news (

newsid INT PRIMARY KEY,

newstitle VARCHAR(255),

newscontent TEXT,

active CHAR(1),

featured CHAR(1),

newsdate TIMESTAMP

);

‘ looks pretty standard. The ‘ active’ field tells us whether the news item is active and ready to be viewed on the site.

So’ should we should create an index on this field for a query like this?

SELECT newsid, newstitle FROM news WHERE active=’1?;

No, we shouldn’t.

If most of your content is live, this index will take up extra space and slow the query down because almost all of the fields match this criteria. Imagine 500 news items in the database with 495 being active. It’s quicker to eliminate the ones that aren’t active than it is to list all of the active ones (if you do have an index on the ‘ active’ field, some databases will choose to ignore it anyway because it will slow the query down).

The featured field tells us whether the news item should feature on the front page. S

hould we index this field? Yes. Most of our content is not featured, so an index on the ‘ featured’ column will be quite useful.

Other examples of when to index a field include if you’re going to order by it in a query. To get the most recent news items, we do a query like this:

SELECT newtitle, newscontent FROM news ORDER BY newsdate DESC;

Creating an index on ‘ newsdate’ will allow the database to quickly sort the results so it can fetch the items in the right order. Indexing can be a bit tricky to get right, however there are tools available for each database to help you work out if it’s working as it should.

Well there you have it ‘ my introduction to database indexes. Hopefully you’ve learned something from this article and can apply what you’ve learned to your own databases.

This entry was posted in Programming. Bookmark the permalink.

22 Responses to ‘ Introduction to Database Indexes’

Jim says:

February 17, 2006 at 7: 13 am

I think you need to be a bit more “ the reader knows absolutly nothing” when describing the table joins. You lost me for a bit there. Perhaps a better step by step hand holding example would be better.

[Editors note: Sure thing. I’ll see what I can come up with for next month! If you’re desperate for information and can’t wait – drop me a line – chris at interspire dot com and I’ll explain it further]

Reply

khani says:

May 14, 2006 at 3: 55 pm

Good effort chris,

You ‘ ve described Indexes in a simple way.

Reply

VRS says:

May 24, 2006 at 1: 32 pm

Good article. Do include some explanation on clustered and non clustered indexes.

Reply

Vivek says:

July 13, 2006 at 3: 25 am

Good article. Helped a lot in understading the basics of indexing. Thanks

Reply

Unknown says:

October 11, 2006 at 8: 43 pm

Good article man. I really appretiate your effort.

Reply

Ayaz says:

November 14, 2006 at 9: 22 am

Good article to understand indexes for a beginner.

Reply

Debiz says:

November 27, 2006 at 5: 21 pm

Very well written and simply explained for those looking for a basic overview

Reply

Nand says:

December 14, 2006 at 11: 46 am

Good article, felt like walking over the bridge on a gorge. Can u pl. explain drawbacks of using index also.

[Chris’ note – The main drawback is that every insert, update or delete has to change the index as well. If you have a lot of indexes, that adds a lot of overhead to the operation.]

Reply

Myo says:

December 19, 2006 at 11: 56 pm

Very easy to understand and gives examples with different

situations to demonstrate when and where we should use indexes and why.

Thanks man!

Reply

John Lowe says:

March 14, 2007 at 2: 57 am

A quick a useful reminder to what idexes are all about, thanks.

Reply

Shravanti says:

June 26, 2007 at 3: 11 am

Good Introduction to Indexes. It would also be valuable to have information on how do indexes work on OLAP side of a Data Warehouse.

Reply

Harsha says:

August 13, 2007 at 11: 21 pm

crisp tutorial.. good work

Reply

krish says:

September 24, 2007 at 2: 44 am

Really very nice explanation

Reply

Alagesan says:

October 10, 2007 at 11: 33 pm

This is a great article to learn indexing for beginners I really appreciate your efforts and good will in explaining them in words here. Thanks!

Reply

Heather says:

October 12, 2007 at 8: 23 am

This was a great explanation of indexes for me ‘ I am self-taught when it comes to databases so the language in this tutorial was very easy for me to understand. Also, you used great examples to help explain your information. THANKS!

Reply

Jess Duckin says:

October 28, 2007 at 4: 58 am

The explaination on the usage of indexing is very helpful

Reply

Mayur says:

October 29, 2007 at 1: 56 pm

Thank you very much, a really informative tutorial’for me it was a 100% match to what I was looking for. Thanks’

Reply

satish soni says:

January 11, 2008 at 7: 17 am

Great article on indexes even oracle has not provided that much knowledge about indexes

Reply

Shweta says:

January 11, 2008 at 4: 25 pm

Good. Just the overview i needed.

Reply

Hemant Jirange says:

January 17, 2008 at 3: 39 am

Great article’this is very simple to understand whole disadvantages about index’

Reply

ramesh says:

January 18, 2008 at 2: 26 am

impossible’. even wikipedi couldnt match your tutorial on this topic’thank uuuuuuuuuuuuuu very much

Reply

Ravi says:

September 12, 2008 at 5: 57 am

thanks Chris, was an easy read for a database novice. I look forward to seeing the next chapter

Reply

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Home | Email Marketing | Shopping Cart | Knowledge Management Software | Content Management Software | Ecommerce Software | Sell Products Online | Our Guarantee | Privacy Policy

‘ Copyright 1999-2010 Interspire Pty. Ltd. ACN: 107 422 631