SOFTWARE ENGINEERING

Best practices to follow with database indexes [closed]

Asked 8 years, 9 months ago Active 2 years, 7 months ago Viewed 28k times

17

8

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 7 years ago.

What are some DOs and DONTs for improving database performance using index?

A DO would be a case in which an index should be created, or another indexes related tip that will improve performance.

A DONT will be a case when an index shouldn't be created, or another index related action that can hurt the performance.

database performance mysql database-design

edited May 23 '11 at 17:55
user8685

asked May 23 '11 at 17:19
Click Upvote
2,697 ● 3 ● 21 ● 33

3 profile, profile, profile – GrandmasterB May 23 '11 at 18:32

1 See stackoverflow.com/questions/6098616/dos-and-donts-for-indexes – Denis de Bernardy May 24 '11 at 17:21

7 Answers

15

This depends partly on what the database is to be used for, since in general indexes slow down inserts and updates and speed up queries. In a data warehouse, there are generally no updates and batched inserts, making it easier to create indexes, and lots and lots of queries, which get sped up with lots of indexes. In an on-line database for web sales and the like, there's lots of inserts and updates, so having more than a few carefully selected indexes will just slow it down.

If you get a lot of queries of one specific type, you could create an index for the query, although that's more for on-line processing than data warehouses. If certain columns come up a lot in queries, you might want an index on that column, and this is especially useful for data warehouses, which get queried in a lot of different and often unpredictable ways.

Whenever you add or remove an index, try to do a performance test to see what effect it has. Without that, you're shooting blind.

There are books on tuning queries and databases, often specific to one database system and using that RDBMS's tools. If you find yourself needing to optimize the database much, though, you're running a large operation and probably should hire a DBA with appropriate expertise.

answered May 23 '11 at 17:45

David Thornley
19.8k ● 49 ● 82

2

DO: Index the very few fields that you access the most through query and/or comparison.

DON'T: Index every field in the table thinking it will make it faster.

I don't have any statistics on it, but I try to keep no more than 4 indexed fields in a table if I can help it. Normalizing my databases usually helps to keep these numbers down since everything becomes searchable by numeric key (which is faster anyway). I try to stay away from full text fields for indexing. They're pretty heavy.

answered May 23 '11 at 17:23

Joel Etherton
11.6k ● 6 ● 42 ● 54

17

It highly depends on how you use your tables. There is no single and simple answer.

The best advice I can give you is: use a **tuning advisors**. They will analyze the database commands while you are using the application, then they will perform load tests against it to provide you with meaningful advices.

They exist for SQL Server & Oracle. I don't know if other DBMS have them, just I doubt they don't provide such basic tools.

Few random recommandations:

- Indexes provide high performance gains when applied on columns often included in WHERE clause
- Use Clustered index for the most used column in your queries.
- Don't forget that you can create multiple indexes with combination of columns (as they are used in your queries)
- Having many indexes will decrease performances of INSERT commands.

Last advice: if DB performances is really important to your project, hire a specialist. It's what I did.

edited May 23 '11 at 17:39

answered May 23 '11 at 17:32

user2567

2 +1 for indexes on combinations of columns. Indexes on columns `a` and `b` is **not** the same as an index on `(a, b)`. The latter is almost as good as the index on `a` for speeding up queries with a condition on `a`, is massively better for queries with conditions on `a` and `b`, and is not useful for queries on `b` alone. (Most databases won't use it. Oracle will, but doesn't get the mileage out of it that it regularly does.) – btilly May 23 '11 at 19:13

2 +1, would add "learn to read query plans so you'll know what to index" – Steven A. Lowe May 23 '11 at 20:43

1

DO: Try to keep the total size of the clustered index to a minimum. The clustered index entries will be included in other non-clustered indexes and from here comes a potential for wasting disk space.

answered May 23 '11 at 17:49

user8685

1

Think of a table as a lexicon, where the articles are sorted by order of appearance (or no helpful order at all), and of a table index as a book index to that lexicon.

You use an index to quickly find something in a book. Instead of scanning the whole book, you only need to find the key in the index (an index usually being somehow sorted (by category, by scientific field, by historical epoch, etc.), this also means you won't have to scan the whole index) and then jump to the right page.

Unlike a book however, a table is not once printed and then immutable. It gets updated all the time, and therefore every index must be updated with it. This of course comes at a space and time cost, that only can be justified by the usefulness of an index.

So do use an index for a column, if that column is used as a key in frequent search queries, and don't use one, if it's not. The word *frequent* is as good a quantifier as it gets, when generally speaking. In the end you will have to make a good estimate which ones are frequent, then simply benchmark performance with or without index in case of doubt.

answered May 23 '11 at 17:52

back2dos

28.9k ● 3 ● 69 ● 112

2

Basically, indices speed up searching but slow down writing, and they take up space. That's the trade-off being made.

Any field that is frequently used for joining on, searching/comparing on or ordering by is a candidate for an index. To know it is really benificial, measure. However, the foreign keys of heavily joined tables with lots (>1000s) of records and few inserts will pay off.

For text fields, you can index on a part of the field (for example, the first 6 chars) which would speed up your query but lighten the load on the indices. Full text searches (search on `like %substring%`) requires different techniques, which I am not familiar with, so I can't give you advise there.

An important situation where indices are not going to help: you can't use the index of complete date or datetime fields when you search (/join/order) on part of the date. An index on `date_created` will not help you with a query like `select * from t where year(date_created) = 2011`. In mysql you can't create an index on part of the date. (When you use '`between`' rather than `year()` it can use the index on the date field.)

More info on MYSQL in the manual: http://dev.mysql.com/doc/refman/5.6/en/optimization-indexes.html

answered May 23 '11 at 19:24

Inca
1,514 ● 9 ● 11

4

@Pierre 303 already said it, but I'll say it again. **DO** use indexes on combinations of columns. A combined index on `(a, b)` is only slightly slower for queries on `a` than an index on `a` alone, and is massively better if your query combines both columns. Some databases can join indexes on `a` and `b` before hitting the table, but this is not nearly as good as having a combined index. When you create a combined index you should put the column that is most likely to be searched first in the combined index.

If your database supports it, *DO* put indexes on functions that show up in queries rather than columns. (If you're calling a function on a column, indexes on that column are useless.)

If you're using a database with true temporary tables that you can create and destroy on the fly (eg PostgreSQL, MySQL, but *not* Oracle), then **DO** create indexes on temporary tables.

If you're using a database that allows it (eg Oracle), **DO** lock in good query plans. Query optimizers over time will change query plans. They usually improve the plan. But sometimes they make it dramatically worse. You generally won't really notice plan improvements - the query wasn't a bottleneck. But a single bad plan can take down a busy site.

DON'T have indexes on tables you're about to do a large data load on. It is much, much faster to drop indexes, load the data, then rebuild the indexes than to maintain them as you load the table.

DON'T use indexes on queries that have to access more than a small fraction of a large table. (How small depends on hardware. 5% is a decent rule of thumb.) For example, if you have data with names and gender, names are a good candidate for indexing since any given name represents a small fraction of the total rows. It would not be helpful to index on gender since you'll still have to access 50% of the rows. You really want to use a full table scan instead. The reason is that indexes wind up accessing a large file randomly, causing you to need disk seeks. Disk seeks are slow. As a case in point I recently managed to speed up an hour long query that looked like:

```
SELECT small_table.id, SUM(big_table.some_value)
FROM small_table
  JOIN big_table
    ON big_table.small_table_id = small_table.id
GROUP BY small_table.id
```

to under 3 minutes by rewriting it as follows:

```
SELECT small_table.id, big_table_summary.summed_value
FROM small_table
  JOIN (
      SELECT small_table_id, SUM(some_value) as summed_value
      FROM big_table
      GROUP BY small_table_id
    ) big_table_summary
    ON big_table_summary.small_table_id =  small_table.id
```

which forced the database to understand that it shouldn't attempt to use the tempting index on `big_table.small_table_id`. (A good database, such as Oracle, should figure that out on its own. This query was running on MySQL.)

Update: Here is an explanation of the disk seek point that I made. An index gives a quick lookup to say where the data is in the table. This is usually a win since you will only look at the data you need to look at. But not always, particularly if you will eventually look at a lot of data. Disks stream data well, but make lookups slow. A random lookup to data on disk takes 1/200th of a second. The slow version of the query wound up doing something like 600,000 of those and took close to an hour. (It did more lookups than that, but caching caught some of those.) By contrast the fast version knew it had to read everything and streamed data at something like 70 MB/second. It got through an 11 GB table in under 3 minutes.

edited Jul 10 '17 at 22:21

answered May 23 '11 at 19:35

btilly

17.8k ● 1 ● 44 ● 72

Hi, I'm confused by your example. I would've thought that using the index would've made things faster, isn't that the point of indexes? Are you saying that if a query would access > 5% of a table, then having an index on the column you're searching by would make things slower? – Click Upvote May 24 '11 at 16:30

@Click Upvote: If a query accesses more than 5% (exact fraction highly dependent on hardware and data) of a table, it is faster to not use an index for that query. Having an index doesn't hurt as long as you don't use it. I'll update with more detail as to why that is. – btilly May 24 '11 at 16:52

Useful information. More on this for example mysqlperformanceblog.com/2007/08/28/... But I was wondering, was 'ignore key' not up to this that you need to make it a subquery? – Inca May 25 '11 at 10:34

@Inca: I was not aware of 'ignore key'. I switch databases enough that there are often database specific things that I am not aware of. From the sounds of it that would work, but significantly less efficiently than my eventual solution. The difference being that that would join, then group, while mine grouped, then joined. This saves work on the join because fewer records need to be joined. – btilly May 25 '11 at 17:05

"A good database (eg Oracle, but not MySQL)": please, avoid stupid promotional stuff like that, especially when you ignore the fact that MySQL can perfectly use multiple index at the same time (noted "INDEX MERGE" in query plans). – Patrick Allaert Jul 10 '17 at 15:22