bogdan@j3e
Web, Java, J2EE, SaaS, Tips&Tricks

MySQL Index Performance

When you create an index at design time, you only can guess what would happen with you application in production. After you are live, the real hunt for creating indexes is just begining. Very often we tend to choose bad indexes ignoring basic rational thinking. I've seen a lot indexes that were slowing the application instead of increasing speed.

Right now, the current databases advanced so far that the differences between indexing a number column and indexing a varchar column are not so obvious anymore. Either you think to create an index on a varchar or on a number column, first you need to lay down the selects that you are going to run against that table. Not doing so is a waste of time. Next, think at the distribution.

For example let's assume that we store 11,000,000 users in a table called my_users. We would have the following fields: username, email, first name, last name, nickname, birthday, age, gender, country. Our application will search for users using the following fields:

1. username
2. first name, last name, gender
3. email

For 1 and 3 the indexes are obvious, we could make an index on username and an index on email. If the username is unique their selectivity will be equal with the primary key selectivity. But what is selectivity? A simple definition is that the selectivity of a column is the ratio between # of distinct values and # of total values. A primary key has selectivity 1.

So coming back to case number 2, what would be the best indexes? Let's take the gender column first. Here we have only two possible values M and F. This means a selectivity of 2/11,000,000 which is 0, an awful index. If you have such an index you may well drop it because a full table scan could be more efficient than using this index.

How about first name and last name? This is a little more complicated, it differs on what names are you storing. If you have 30,000 of Johns and 50,000 of Smiths the index is useless, a simple select distinct on each column will give you the number to calculate the selectivity. If it is above 15% then the index is good, otherwise drop it. But one great thing is that in this case you could create a composed index on first

name+last name which it will give you a higher selectivity for sure . Don't forget that in MySQL an index on string columns allows only 1000 characters, which means when using UTF-8 you can only inlcude 333 characters.

Select the worst performing indexes by using the following sql. Note that composed indexes will apear multiple times, in the result, for each column so you need to pick the last apearance. Everything is below 15% it needs to be analyzed.

view plain copy to clipboard print ?

```
/*
SQL script to grab the worst performing indexes
in the whole server
*/
SELECT
t.TABLE_SCHEMA AS `db`
, t.TABLE_NAME AS `table`
, s.INDEX_NAME AS `inde name`
, s.COLUMN_NAME AS `field name`
, s.SEQ_IN_INDEX `seq in index`
, s2.max_columns AS `# cols`
, s.CARDINALITY AS `card`
, t.TABLE_ROWS AS `est rows`
, ROUND(((s.CARDINALITY / IFNULL(t.TABLE_ROWS, 0.01)) * 100), 2) AS `sel %`
FROM INFORMATION_SCHEMA.STATISTICS s
INNER JOIN INFORMATION_SCHEMA.TABLES t
ON s.TABLE_SCHEMA = t.TABLE_SCHEMA
AND s.TABLE_NAME = t.TABLE_NAME
INNER JOIN (
SELECT
TABLE_SCHEMA
, TABLE_NAME
, INDEX_NAME
, MAX(SEQ_IN_INDEX) AS max_columns
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA != 'mysql'
GROUP BY TABLE_SCHEMA, TABLE_NAME, INDEX_NAME
) AS s2
ON s.TABLE_SCHEMA = s2.TABLE_SCHEMA
AND s.TABLE_NAME = s2.TABLE_NAME
AND s.INDEX_NAME = s2.INDEX_NAME
WHERE t.TABLE_SCHEMA != 'mysql'                          /* Filter out the mysql system DB
AND t.TABLE_ROWS > 10                                    /* Only tables with some rows */
AND s.CARDINALITY IS NOT NULL                            /* Need at least one non-NULL val
AND (s.CARDINALITY / IFNULL(t.TABLE_ROWS, 0.01)) < 1.00 /* Selectivity < 1.0 b/c unique i
ORDER BY `sel %`, s.TABLE_SCHEMA, s.TABLE_NAME           /* Switch to `sel %` DESC for bes
```

This is taken from MySQL forge

February 8, 2009 | Filed Under MySql

Comments

3 Responses to "MySQL Index Performance"

1. Tom on February 27th, 2010 9:12 pm

   Thanks for this very clear explanation, really useful.

2. vikixp on March 24th, 2010 12:53 am

   Thanks..so helpful.....

3. Kelsi Foston on January 30th, 2011 9:43 am

   I'm really enjoying the theme/design of your web site. Do you ever run into any browser compatibility problems? A few of my blog visitors have complained about my website not working correctly in Explorer but looks great in Opera. Do you have any tips to help fix this issue?

- To search, type and hit enter

- RSS for Articles
- RSS for Comments
- My Twitter
- Bucharest daily photos

- **About me**

  This blog is about building web applications. The views expressed on this site are personally.More about me.

- **Recently Written**

  - Key takeaways from my TDD training sessions
  - How a DNS problem can put your Mysql server down
  - Creating a secure JMX Agent in JDK 1.5
  - Configure Apache and Tomcat severs together
  - Tomcat Clustering & Java Servlet Specification
  - Tomcat clustering configuration
  - Put together Struts2, JPA, Hibernate and Spring
  - Key-Value Storage using MemcacheDB
  - Facebook temporarily lost data.
  - Simple MVC for Flex and AIR

- **Categories**

  - AIR
  - Architecture
  - clustering
  - Coding
  - Design

- Flex
- Hibernate
- Infrastructure
- j2EE news
- JMX
- JPA
- Memcachedb
- mod_proxy_ajp
- Monitoring
- MVC
- MySql
- News
- SOA
- Spring
- Struts2
- Tomcat
- Web API

Archives

- February 2010
- January 2010
- October 2009
- August 2009
- June 2009
- May 2009
- March 2009
- February 2009

Blogroll

- Alexandru Nedelcu
- Ben Forta
- BileBlog
- Bitesize Bugs
- Cameron Purdy /dev/null
- Cornel the Godfather
- Dikran Seropian
- James Ward
- macaroni – Ted Husted's blog
- Mihai Corlan Flex,RIA&PHP
- myadobe.ro
- Rod Johnson
- Ted Neward
- Toby Segaran

created by Brian Gardner