Optimize MySQL search process

Asked 15 years, 10 months ago Modified 15 years, 10 months ago Viewed 2k times

Part of PHP Collective

0

Here is the **scenario 1.**

I have a table called "items", inside the table has 2 columns, e. g. `item_id` and `item_name`. I store my data in this way: item_id | item_name

```
Ss001   | Shirt1
Sb002   | Shirt2
Tb001   | TShirt1
Tm002   | TShirt2
```

... etc, i store in this way: first letter is the code for clothes, i.e S for shirt, T for tshirt second letter is size, i.e s for small, m for medium and b for big Lets say in my items table i got 10,000 items. I want to do fast retrieve, lets say I want to find a particular shirt, can I use:

Method1:

```
SELECT * from items WHERE item_id LIKE Sb99;
```

or should I do it like:

Method2:

```
SELECT * from items WHERE item_id LIKE S*;
```

*Store the result, then execute second search for the size, then third search for the id. Like the hash table concept. What I want to achieve is, instead of search all the data, I want to minimize the search by search the clothes code first, follow by size code and then id code. Which one is better in term of speed in mysql. And which one is better in long run. I want to reduce the traffic and not to disturb the database so often.

Thanks guys for solving my first scenario. But another scenario comes in:

Scenario 2:

I am using PHP and MySQL. Continue from the preivous story. If my users table structure is like this:

```
user_id | username | items_collected

U0001   | Alex     | Ss001;Tm002
U0002   | Daniel   | Tb001;Sb002
U0003   | Michael  | ...
U0004   | Thomas   | ...
```

I store the items_collected in id form because one day each user can collect up to hundreds items, if I store as string, i.e. Shirt1, pants2, ..., it would required a very large amount of database spaces (imagine if we have 1000 users and some items name are very long).

Would it be easier to maintain if I store in id form?

And if lets say, I want to display the image, and the image's name is the item's name + jpg. How to do that? Is it something like this:

$result = Select items_collected from users where userid= $userid

Using php explode:

$itemsCollected = explode($result, ";");

After that, matching each item in the items table, so it would like:

shirt1, pants2 etc

Den using loop function, loop each value and add ".jpg" to display the image?

PHP php mysql search methods

Share
Improve this question
Follow

edited Feb 16, 2009 at 3:25

asked Feb 12, 2009 at 11:05

roa3
901 ● 8 ● 15 ● 27

for an exact match you can: WHERE item_id='Sb99' (don't forget the quotes) – tehvan Feb 12, 2009 at 11:16

5 Answers

Sorted by: Highest score (default)

The first method will be faster - but IMO it's not the right way of doing it. I'm in agreement with tehvan about that.

3

I'd recommend keeping the item_id as is, but add two extra fields one for the code and one for the size, then you can do:

```
select * from items where item_code = 'S' and item_size = 'm'
```

With indexes the performance will be greatly increased, and you'll be able to easily match a range of sizes, or codes.

```
select * from items where item_code = 'S' and item_size IN ('m','s')
```

Migrate the db as follows:

```
alter table items add column item_code varchar(1) default '';
alter table items add column item_size varchar(1) default '';

update items set item_code = SUBSTRING(item_id, 1, 1);
update items set item_size = SUBSTRING(item_id, 2, 1);
```

The changes to the code should be equally simple to add. The long term benefit will be worth the effort.

For scenario 2 - that is not an efficient way of storing and retrieving data from a database. When used in this way the database is only acting as a storage engine, by encoding multiple data into fields you are precluding the relational part of the database from being useful.

What you should do in that circumstance is to have another table, call it 'items_collected'. The schema would be along the lines of

```
CREATE TABLE items_collected (
   id int(11) NOT NULL auto_increment KEY,
   userid int(11) NOT NULL,
   item_code varchar(10) NOT NULL,
   FOREIGN KEY (`userid`) REFERENCES `user`(`id`),
   FOREIGN KEY (`itemcode`) REFERENCES `items`(`item_code`)
 );
```

The foreign keys ensure that there is Referential integrity, it's essential to have referential integrity.

Then for the example you give you would have multiple records.

```
user_id | username | items_collected
U0001   | Alex     | Ss001
U0001   | Alex     | Tm002
U0002   | Daniel   | Sb002
```

```
U0002   | Daniel   | Tb001
U0003   | Michael  | ...
U0004   | Thomas   | ...
```

Share

Improve this answer

Follow

edited Feb 18, 2009 at 10:05

answered Feb 12, 2009 at 11:28

Richard Harrison

19.4k ● 4 ● 42 ● 67

lets say, my data is not huge enough, should I still use indexes?? – roa3 Feb 16, 2009 at 9:12

For small values of huge you don't need indexes - however if the data is primarily static then even with a few (<100) rows it is worth it. Prepend "EXPLAIN" onto the select to see what is happening. – Richard Harrison Feb 18, 2009 at 9:50

1

The first optimization would be splitting the id into three different fields: one for type, one for size, one for the current id ending (whatever the ending means) If you really want to keep the current structure, go for the result straight away (option 1).

Share Improve this answer Follow

answered Feb 12, 2009 at 11:13

tehvan

10.4k ● 5 ● 28 ● 31

1

If you want to speed up for results you should split up the column into multiple columns, one for each property.

Step 2 is to create an index for each column. Remember that mysql only uses one index per table per query. So if you really want speedy queries and your queries vary a lot with these properties, then you might want to create an index on (type,size,ending), (type,ending,size) etc.

For example a query with

```
select * from items where type = s and size = s and ending = 001
```

Can benefit from the index (type,size,ending) but:

```
select * from items where  size = s and ending = 001
```

Can not, because the index will only be used in order, so it needs type, then size, then ending. This is why you might want multiple indexes if you really want fast searches.

One other note, generally it is not a good idea to use * in queries, but to select only the columns you need.

Share Improve this answer Follow

answered Feb 12, 2009 at 11:22

TomHastjarjanto
5,408 ● 1 ● 30 ● 41

1

You need to have three columns for the `model`, `size` and `id`, and index them this way:

```
CREATE INDEX ix_1 ON (model, size, id)
CREATE INDEX ix_2 ON (size, id)
CREATE INDEX ix_3 ON (id, model)
```

Then you'll be able to search efficiently on any subset of the parameters:

- `model-size-id`, `model-size` and `model` queries will use `ix_1`;
- `size-id` and `size` queries will use `ix_2`;
- `model-id` and `id` queries will use `ix_3`

Index on your column as it is now is equivalent to `ix_1`, and you can use this index to efficiently search on the appropriate conditions (`model-size-id`, `model-size` and `model`).

Actually, there is a certain access path called `INDEX SKIN SCAN` that may be used to search on non-first columns of a composite index, but `MySQL` does not support it AFAIK.

If you need to stick to your current design, you need to index the field and use queries like:

```
WHERE item_id LIKE @model || '%'
WHERE item_id LIKE @model || @size || '%'
WHERE item_id = @model || @size || @id
```

All these queries will use the index if any.

There is not need to put in into multiple queries.

Share

Improve this answer

Follow

edited Feb 16, 2009 at 9:17

answered Feb 12, 2009 at 12:11

Quassnoi
425k ● 93 ● 625 ● 619

I'm comfortable that you've designed your item_id to be searchable with a "Starts with" test. Indexes will solve that quickly for you.

I don't know MySQL, but in MSSQL having an index on a "Size" column that only has choices of S, M, L most probably won't achieve anything, the index won't be used because the values it contains are not sufficiently selective - i.e. its quicker to just go through all the data rather than "Find the first S entry in the index, now retrieve the data page for that row ..."

The exception is where the query is covered by the index - i.e. several parts of the WHERE clause (and indeed, all of them and also the SELECT columns) are included in the index. In this instance, however, the first field in the index (in MSSQL) needs to be selective. So put the column with the most distinct values first in the index.

Having said that if your application has a picklist for Size, Colour, etc. you should have those data attributes in separate columns in the record - and separate tables with lists of all the available Colours and Sizes, and then you can validate that the Colour / Size given to a Product is actually defined in the Colour / Size tables. Cuts down the Garbage-in / Garbage-out problem!

Your item_selected needs to be in a separate table so that it is "normalised". Don't store a delimited list in a single column, store it using individual rows in a separate table

Thus your USERS table will contain user_id & username

Your, new, items_collected table will contains user_id & item_id (and possibly also Date Purchased or Invoice Number)

You can then say "What did Alex buy" (your design has that) and also "Who bought Ss001" (which, in your design, would require ploughing through all the rows in your USERS table and splitting out the items_collected to find which ones contained Ss001 [1])

[1] Note that using LIKE wouldn't really be safe for that because you might have an item_id of "Ss001XXX" which would match WHERE items_collected LIKE '%Ss001%'

answered Feb 16, 2009 at 9:55

Kristen

4,281 ● 2 ● 30 ● 37