redmineorg-copy202205 - Vote #79404

Add missing index to custom_values

2022/05/09 18:33 - Admin Redmine

ステータス:	New	**開始日:**	2022/05/09
優先度:	通常	**期日:**	
担当者:		**進捗率:**	0%
カテゴリ:	Performance_53	**予定工数:**	0.00時間
対象バージョン:	Candidate for next major release_32	**作業時間:**	0.00時間
Redmineorg_URL:	https://www.redmine.org/issues/29171	**status_id:**	1
category_id:	53	**tracker_id:**	3
version_id:	32	**plus1:**	0
issue_org_id:	29171	**affected_version:**	
author_id:	123866	**closed_on:**	
assigned_to_id:	0	**affected_version_id:**	
comments:	9		

説明

On our Redmine installation, we have around 100000+ issues with lots of custom fields.
I ran into a bottleneck were some Issue Queries where very very slow when requesting criteria on multiple custom fields and grouping.

In the custom_values table, I noticed that there was no index for @[customized_type, customized_id, custom_field_id]@. Adding such index resulted in loading times much faster for those complex issue queries (From 60+ seconds down to 5 seconds, with DB caching disabled).

Here is the index I added:

```
class AddMissingIndexCustomValues < ActiveRecord::Migration
    def change
        add_index :custom_values, [:customized_type, :customized_id, :custom_field_id], name: "custom_values_customized_custom_field"
    end
end
```

journals

there're already two indexes

```
custom_values_customized                          [customized_type, customized_id]
index_custom_values_on_custom_field_id      [custom_field_id]
```

it would be better to replace custom_values_customized with [customized_type, customized_id, custom_field_id]. We don't need an extra index and I'll still be effective.

60s to 5s seems to be a lot. My db has 140000 issues and 10000000 custom values (mysql 5.7) and I can confirm grouping queries with custom fields are about 50% faster (without db caching). It's an improvement.

What is your db backend? Could you share query plans (explain) of the problematic query (with and without the index)?

https://dev.mysql.com/doc/refman/8.0/en/explain-extended.html

Here is my DB Version: @mysql Ver 15.1 Distrib 10.0.34-MariaDB, for debian-linux-gnu (x86_64) using readline 5.2@

Please find below an example of long SQL Query. The original query was much much bigger, but I have isolated a part which was taking a lot of time. For instance, sort the list of issues by a Custom Field:

```
SELECT issues.* FROM issues
LEFT OUTER JOIN custom_values cf_34
    ON cf_34.customized_type = 'Issue'
```

```
    AND  cf_34.customized_id  =  issues.id
    AND  cf_34.custom_field_id  =  34
    AND  cf_34.value  <>  ''
ORDER    BY  Coalesce(cf_34.value,  '')  DESC
LIMIT  25;
```

Here Custom Field id is 34, it is of type list in Redmine and its possible values are @['OK', 'KO', '']@ The DB contains: - 147262 Issues - 3318197 Custom Values - 51211 Custom Values associated with Custom Field 34 *Running the query without the added index Takes 13 seconds:*

```
...
25  rows  in  set  (13.64  sec)
```

And the EXPLAIN:

```
***************************  1.  row  ***************************
           id:  1
  select_type:  SIMPLE
        table:  issues
         type:  ALL
possible_keys:  NULL
          key:  NULL
      key_len:  NULL
          ref:  NULL
         rows:  147262
        Extra:  Using  temporary;  Using  filesort
***************************  2.  row  ***************************
           id:  1
  select_type:  SIMPLE
        table:  cf_34
         type:  ref
possible_keys:  custom_values_customized,index_custom_values_on_custom_field_id
          key:  custom_values_customized
      key_len:  96
          ref:  const,redmine_development.issues.id
         rows:  14
        Extra:  Using  where
```

Now with the added index, it takes 2 seconds:

```
...
25  rows  in  set  (1.91  sec)
```

And the EXPLAIN:

```
***************************  1.  row  ***************************
           id:  1
  select_type:  SIMPLE
        table:  issues
         type:  ALL
possible_keys:  NULL
          key:  NULL
      key_len:  NULL
          ref:  NULL
         rows:  147262
        Extra:  Using  temporary;  Using  filesort
***************************  2.  row  ***************************
           id:  1
  select_type:  SIMPLE
        table:  cf_34
         type:  ref
possible_keys:  custom_values_customized,index_custom_values_on_custom_field_id,custom_values_customized_custom_field
          key:  custom_values_customized_custom_field
      key_len:  100
          ref:  const,redmine_development.issues.id,const
         rows:  1
        Extra:  Using  where
```

Pavel Rosický wrote:

> it would be better to replace custom_values_customized with [customized_type, customized_id, custom_field_id]. We don't need an extra index and I'll still be effective.

I agree we could just replace the existing index with the new one, though I don't know if this may slow down things somewhere else.

ok, the real problem is elsewhere

```
SELECT  issues.*  FROM  issues
LEFT  OUTER  JOIN  custom_values  cf_34
    ON  cf_34.customized_type  =  'Issue'
    AND  cf_34.customized_id  =  issues.id
    AND  cf_34.custom_field_id  =  34
    AND  cf_34.value  <>  ''
ORDER    BY  Coalesce(cf_34.value,  '')  DESC
LIMIT  25;
```

id: 1
select_type: SIMPLE
table: issues
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 147262
Extra: Using temporary; Using filesort

it means that your db has to load 147262 issues to execute your query. The query is also ordered

```
ORDER    BY  Coalesce(cf_34.value,  '')  DESC
```

because it's ordered by a join statement it has to process
14 * 147262 rows
if you add an index the locality is better, then we have to process only
1 * 147262 rows

Maybe we can use inner join for IS/ALL filters (give it a try), but it would require major changes how redmine stores custom values right now. There always has to be a custom value and it also won't work for NULL values. Let's discuss about it in a new ticket if you're interested.

```
SELECT  issues.*  FROM  issues
INNER  JOIN  custom_values  cf_34
    ON  cf_34.customized_type  =  'Issue'
    AND  cf_34.customized_id  =  issues.id
    AND  cf_34.custom_field_id  =  34
    AND  cf_34.value  <>  ''
ORDER    BY  Coalesce(cf_34.value,  '')  DESC
LIMIT  25;
```

Stephane Evr wrote:

> Pavel Rosický wrote:
>
> > it would be better to replace custom_values_customized with [customized_type, customized_id, custom_field_id]. We don't need an extra index and I'll still be effective.
>
> I agree we could just replace the existing index with the new one, though I don't know if this may slow down things somewhere else.
> writes could be slower because [customized_type, customized_id, custom_field_id] is more complicated than [customized_type, customized_id]

> it won't slowdown existing read queries because the order is the same as the previous index, for instance
>
> SELECT custom_values WHERE customized_type = 'Issue' AND customized_id = 1 can use [customized_type, custo
> , custom_field_id] index
>
> but
>
> SELECT custom_values WHERE customized_id = 1 AND custom_field_id = 1 can't (not a real case)

Pavel Rosický wrote:

> because it's ordered by a join statement it has to process
> 14 * 147262 rows
> if you add an index the locality is better, then we have to process only
> 1 * 147262 rows
>
> Maybe we can use inner join for IS/ALL filters (give it a try), but it would require major changes how redmine stores custom values right now. There always has to be a custom value and it also won't work for NULL values. Let's discuss about it in a new ticket if you're interested.

Okay, thanks for your comments! One thing I am missing is why 14 rows? I tried with a completely different custom field and there were 14 rows to process as well.

Anyway I will keep this index as it has really decreased the response time of Issue#index by a lot in complex projects. For sure there is a small overhead but only as new issues are created (existing custom values will not be reindexed when their value changes).

Stephane Evr wrote:

> Okay, thanks for your comments! One thing I am missing is why 14 rows? I tried with a completely different custom field and there were 14 rows to process as well.
> It is showing how many rows it ran through to get result (it's just an estimate, not an exact number). It depends on many factors, but if the number of rows is too high the query is probably too complex or indexes are missing.
>
> Anyway I will keep this index as it has really decreased the response time of Issue#index by a lot in complex projects. For sure there is a small overhead but only as new issues are created (existing custom values will not be reindexed when their value changes).

I don't see any downsides about this change, so +1

> Stephane Evr and Pavel Rosický, thank you for the detailed investigation.

My understanding is that the conclusion is that Redmine should have an index for @[:customized_type, :customized_id, :custom_field_id]@ instead of @[customized_type, customized_id]@. Is it correct?

Go MAEDA wrote:

> Stephane Evr and Pavel Rosický, thank you for the detailed investigation.
>
> My understanding is that the conclusion is that Redmine should have an index for @[:customized_type, :customized_id, :custom_field_id]@ instead of @[customized_type, customized_id]@. Is it correct?

Yes

履歴

#1 - 2022/05/10 17:04 - Admin Redmine

- カテゴリ を Performance_53 にセット

- 対象バージョン を Candidate for next major release_32 にセット