ChiliProject - Bug # 498: Wrong filters for int and float custom fields

Status:	Closed	**Priority:**	Normal
Author:	Felix Schäfer	**Category:**	Custom Fields
Created:	2011-06-30	**Assignee:**	
Updated:	2011-07-08	**Due date:**	
Remote issue URL:	http://www.redmine.org/issues/6180		
Affected version:	master		
Description:	int and float custom fields get the same filters as strings (default for custom fields in "unknown formats") instead as the ones for "integer custom fields".		

Associated revisions

2011-07-08 09:39 pm - Felix SchÃ¤fer

Correct filters for int,float custom fields. #498

History

2011-06-30 12:21 pm - Felix Schäfer

I have a fix for this, will post it shortly.

2011-07-04 02:04 pm - Felix Schäfer

- Target version set to 2.1.0

- Assignee deleted (Felix Schäfer)

- Status changed from Open to Ready for review

Ok, so this was a little longer in the making than I had thought because of how custom fields work (wouldn't it have been easier to have a text field for "character"-based custom fields and a float/numeric field for numeric custom fields?â€¦). Anyway, here's my proposed patch:

```
<pre><code class="diff">diff --git a/app/models/query.rb b/app/models/query.rb
index 1772d68..f7abb19 100644
--- a/app/models/query.rb
+++ b/app/models/query.rb
@@ -586,9 +586,17 @@ class Query < ActiveRecord::Base
       sql = "#{db_table}.#{db_field} IS NOT NULL"
       sql << " AND #{db_table}.#{db_field} <> ''" if is_custom_filter
     when ">="
-      sql = "#{db_table}.#{db_field} >= #{value.first.to_i}"
+      if is_custom_filter
+        sql = "#{db_table}.#{db_field} != '' AND CAST(#{db_table}.#{db_field} AS decimal(60,3)) >= #{value.first.to_f}"
+      else
+        sql = "#{db_table}.#{db_field} >= #{value.first.to_f}"
+      end
     when "<="
-      sql = "#{db_table}.#{db_field} <= #{value.first.to_i}"
+      if is_custom_filter
+        sql = "#{db_table}.#{db_field} != '' AND CAST(#{db_table}.#{db_field} AS decimal(60,3)) <= #{value.first.to_f}"
+      else
+        sql = "#{db_table}.#{db_field} <= #{value.first.to_f}"
+      end
     when "o"
       sql = "#{IssueStatus.table_name}.is_closed=#{connection.quoted_false}" if field == "status_id"
     when "c"
@@ -628,6 +636,8 @@ class Query < ActiveRecord::Base

     custom_fields.select(&:is_filter?).each do |field|
       case field.field_format
+      when "int", "float"
+        options = { :type => :integer, :order => 20 }
       when "text"
         options = { :type => :text, :order => 20 }
       when "list"</code></pre>
```

While we're at it, I'd like to change the @:integer@ filter type to @:numeric@ to make it clear it doesn't work only for integers.

2011-07-04 02:34 pm - Holger Just
Hmmm, this results in a full table scan as the casts can not use any index. This might be an issue if you have a couple of custom values. Also I'd use a scale of a least 4 digits in the decimal to accomodate common currency precision. Using this, I think this is a nice workaround

That said, I think a much cleaner variant would be to provide more columns on the custom fields to be used for different data types. Those columns should by correctly typed (and indexed) in the database. Thus, there could be the following columns:

* *:text* for actual strings and serialized complex values
* *:integer* for well,integers
* *:currency* which could be a @decimal(60,4)@ in the database
* *:float@, well floating point stuff. This one might not be necessary as the decimal could be sufficient

Additionally, a type field would select the actually value column to be used. There would only be one column used, the others would be force to contain null.

The slight storage overhead shouldn't really matter compared to the performance opportunitues of actually using indexes.

2011-07-04 10:20 pm - Felix Schäfer
Holger Just wrote:
> Hmmm, this results in a full table scan as the casts can not use any index. This might be an issue if you have a couple of custom values.

The current custom field structure doesn't allow anything better, more on that below.

> Also I'd use a scale of a least 4 digits in the decimal to accomodate common currency precision. Using this, I think this is a nice workaround

Well, I just copied the precision from source:/app/models/custom_field.rb#L116, we can bump both to @DECIMAL(60,4)@ or even @DECIMAL(65,10)@ though, I don't think the performance will be much different.

> That said, I think a much cleaner variant would be to provide more columns on the custom fields to be used for different data types. Those columns should by correctly typed (and indexed) in the database. Thus, there could be the following columns:
>
> * *:text* for actual strings and serialized complex values
> * *:integer* for well,integers
> * *:currency* which could be a @decimal(60,4)@ in the database
> * *:float@, well floating point stuff. This one might not be necessary as the decimal could be sufficient
>
> Additionally, a type field would select the actually value column to be used. There would only be one column used, the others would be force to contain null.
>
> The slight storage overhead shouldn't really matter compared to the performance opportunitues of actually using indexes.

Regarding that, I really (really really) dislike that the types are hardcoded and just differentiated by a bunch of case statements. I've spent a few minutes pondering this, and I think the nicest solution would be to have sort of a @CustomField@ namespace and have say @CustomField::String@, @CustomField::Text@, @CustomField::Integer@, @CustomField::Date@, @CustomField::Users@ and so on, and each could provide/decide how to store values (maybe @CustomValue::Text@, @CustomValue::Numeric@, @CustomValue::Association@ (for stuff referencing core objects and/or (multiselect) lists)), how to sort them, how to sort objects using them, and so on. This would also have the advantage of every field type being able to provide it's own form for the admin section instead of relying on (unextensible) js, that you wouldn't need to declare custom field types in @lib/redmine.rb@ as is done now (just scan for @CustomField::@ types), and all that combined would make adding new types even from a plugin a breeze.

The biggest snag I've hit though is that that would need some sort of doubly polymorphic many-to-many relation: @Customizable@ many-to-many @CustomField@, where @Customizable@ can be anything and sees custom fields as "@CustomizableCustomField@" (e.g. @IssueCustomField@), and @CustomField@ being a @CustomField::String@ or @CustomField::Users@ and so on. The backend shouldn't be too much of a problem, but I'm not sure how good rails will play with that.

Anyway, already much too much discussion about that for the scope of this ticket, I'll open a forum thread tomorrow to discuss all this, I'm also pretty sure there's a nicer way that what I have in mind currently, but I guess my ruby-foo isn't good enough for it yet :-)

2011-07-05 09:21 am - Felix Schäfer
Also see the "custom field rework proposal":/boards/2/topics/547 for more info.

2011-07-08 07:41 pm - Felix Schäfer
- Status changed from Ready for review to Closed

Committed the proposed patch with @DECIMAL(60,3)@ changed to @DECIMAL(60,4)@, this will at least get us working filters until we have a better solution.