The PhenoDCC HeatMap and the Annotation Web Services

Gagarine Yaikhom

Last updated: 17 December 2013

Introduction

The annotation web services return significance values against a set of genotypes for a collection of ontology terms, or procedure/parameter. These values are then used by the client browser to display the PhenoDCC HeatMap. In this note, we wish to improve the design and implementation of the web services for better performance.

JSON data format for use by the PhenoDCC HeatMap

The PhenoDCC HeatMap web application expects the annotation web services to return data using the following JSON specification.

```
{
    "title": "Procedure HeatMap",
    "row_headers": <row entries>,
    "column_headers": <column entries>,
    "significance": <significance entries>
}
```

where, `<row entries>` and `<column entries>` are one-dimensional arrays of JSON objects that respectively contain row and column header information. The manner in which each of these entries are returned is left unspecified, which allows the user to re-use the PhenoDCC HeatMap for other purposes. For instance, the following JSON objects are both valid:

```
{
    "title": "HeatMap A",
    "row_headers": [
        { "id": 1, "title": "Alpha" },
        { "id": 2, "title": "Beta" }
    ],
    "column_headers": [
        { "id": 1, "title": "X" },
        { "id": 2, "title": "Y" },
        { "id": 3, "title": "Z" }
    ],
    "significance": [
        [1, 2, 3],
        [4, 5, 6]
    ]
}
```

```
{
    "title": "HeatMap B",
    "row_headers": [
        { "id": 1, "name": "Alpha" },
        { "id": 2, "name": "Beta" }
    ],
    "column_headers": [
        { "id": 1, "symbol": "X" },
        { "id": 2, "symbol": "Y" },
        { "id": 3, "symbol": "Z" }
    ],
    "significance": [
        [
            { "pvalue": 1, "link": "1" },
            { "pvalue": 2, "link": "2" },
            { "pvalue": 3, "link": "3" }
        ],
        [
            { "pvalue": 4, "link": "4" },
            { "pvalue": 5, "link": "5" },
            { "pvalue": 6, "link": "6" }
        ]
    ]
}
```

What really matters is that the PhenoDCC HeatMap object is supplied with the appropriate helper functions that can retrive the correct values from each of these entries.

Flexibility and helper function

An instance of the PhenoDCC HeatMap expects the users to pass during initialisation *content formatters* which use the entries returned by the annotation web services to produce HTML code segments that can be display on the HeatMap. For instance, the default row and column formatters that is used by the PhenoDCC HeatMap are as follows:

```
/**
 * Formats the row header entry datum object for display.
 *
 * @param {Object} datum Object that contains the data.
 * @returns {String} Formatted string header entry.
 */
function dcc_rowFormatter(datum) {
    return datum.name;
}

/**
 * Formats the column header entry datum object for display.
 *
 * @param {Object} datum Object that contains the data.
 * @returns {String} Formatted string header entry.
 */
function dcc_columnFormatter(datum) {
    return datum.symbol;
}
```

Hence, a JSON example for use with these default formatters is

```
{
    "title": "Procedure HeatMap",
    "row_headers": [
        {
            "id": "ESLIM_022_001_711",
            "name": "Acoustic Startle"
        },
        {
            "id": "ESLIM_022_001_001",
            "name": "Body Weight"
```

```
        }
    ],
    "column_headers": [
        {
            "id": 10035,
            "symbol": "Akt2"
        },
        {
            "id": 320,
            "symbol": "Aldh2"
        },
        {
            "id": 10538,
            "symbol": "Aldh2"
        }
    ],
    "significance": [
        [ -1, 0.0050, 0.0060 ],
        [ -1, 0.0015, 0.0055 ]
    ]
}
```

The `<significance entries>`, on the other hand, is a two dimensional object which contains the p-values. To extract the values, the user must again provide a Javascript function. The default p-value extractor used by the PhenoDCC HeatMap is:

```
/**
 * Extracts the p-value from a cell datum.
 *
 * @param {Object} datum Object that contains the data.
 * @returns {Real} Floating point value.
 */
function dcc_pvalueExtractor(datum) {
    return datum;
}
```

In other words, a user may choose to use the default helper functions provided by the PhenoDCC HeatMap object, in which case the data returned by the annotation web services must conform to the JSON format above, or define their own JSON format and also provide custom helper functions that understands this specialised format.

Hierarchical navigation

Where a column or row entry allows further inspection, essentially providing a *hierarchical navigation* facility, then we use the **unique** key of the column or row entry being inspected to specialise the request sent to the annotation web services. Hence, this key must be provided in the column or row entry object.

The PhenoDCC HeatMap uses the following function to extract a key from a column or row entry object.

```
/**
 * Extracts the key for a given datum.
 *
 * @param {Object} datum Object that contains the key.
 * @returns {String} The key as a string.
 */
function dcc_keyExtractor(datum) {
    return datum === undefined ? undefined : datum.id;
}
```

Again, the user can supply custom helper function to extract the key.

Where further inspection is not possible, it is sufficient to simply leave the key attribute `undefined`, as shown below:

```
{
    "title": "Procedure HeatMap",
    "row_headers": [
        {
            "name": "Acoustic Startle"
        },
```

```
        {
            "name": "Body Weight"
        },
        ...
    ],
    ...
}
```

This means that we cannot inspect "Acoustic Startle" and "Body Weight" any further, since their unique keys are undefined.

Various forms of the web service request

The annotation web services must accept two optional parameters:

1. `gid` - This is an integer that specifies the genotype we are interested in. This integer value is a primary key in the `genotype` table of the `phenodcc_overviews` database. The following is an example:

   `heatmap/rest/ontological/details?gid=377&type=MP:0005377&threshold=0.0001`

   which is requesting the heatmap details for the Cib2 gene.

2. `type` - This is the procedure type, which, if specified asks the annotation web service to only return items that are associated with parameters which belong to the supplied procedure type. The following is an example:

   `heatmap/rest/ontological/details?gid=377&type=MP:0005377&threshold=0.0001`

   which is requesting the heatmap details for the Cib2 gene for all of the parameters under the ontology term `MP:0005377`, which corresponds to 'Pigmentation'.

3. `mgiid` - It is also possible to identify a genotype using its MGI identifier. For instance, `MGI:1929293` corresponds to Cib2 gene.

   `heatmap/rest/ontological/heatmap?mgiid=MGI:1929293&type=MP:0001186`

4. `threshold` - This allows a user to specify a p-value threshold, so that only those items that are significant (i.e., p-values less than the supplied threshold) are returned.

When the heatmap makes web service requests, the value for the procedure `type` is retrieved from the row entry, which is the value of the *unique row entry key*. Hence, when the user clicks on the row header for further inspection, the query parameters are passed to the annotation web services. The annotation web services should then return another JSON object using the same JSON specification as before.

Please note that the PhenoDCC HeatMap maintains a history of the hierarchical navigation which is displayed to the user as a *breadcrum* user interface. Hence, the annotation web services need not worry about this navigation history. In other words, each request must be treated independently.

Improving performance

> The following analysis was written on 13 March 2013. Current deployment now uses the suggested improvements.

The current implementation of the annotation web services runs several SQL queries to fill in the JSON object as specified above. Unfortunately, it is currently very slow.

The following table summarises the improvement in performance, as reduction in query execution time, after applying the changes described below.

Query type	Original	New (without indexes)	New (with indexes)
Row entries	9.716	1.248	0.001
Column entries	0.004	-	-
p-values	2.615	0.736	0.036

- Measurements are in seconds.

Query to retrieve values for column entries: genotype

The first query retrieves the list of identifiers for all genotypes on which experimental procedures were carried out. The result set is then used to fill in the `<column entries>`:

```
select distinct
    genotype.genotype_id
from
    procedures_performed,
    genotype
where
    genotype.genotype_id != 0
    and procedures_performed.genotype_id = genotype.genotype_id
    and gene_symbol is not null
    and gene_symbol != 'null'
order by
    gene_symbol,
    (genotype.genotype_id + 0);
```

This query takes **~0.004** seconds to execute, so there is no need for optimisation.

Query to retrieve values for row entries: procedure/parameters

The second query is used for filling in the `<row entries>`. This retrieves all of the procedure/parameters group for which we wish to extract the p-values. The query is as follows:

```
select distinct
    parameter_key,
    impress.parameter.name
from
    impress.procedure,
    impress.procedure_type,
    impress.parameter,
    impress.procedure_has_parameters,
    parameters_performed
where
    impress.procedure.type = impress.procedure_type.id
    and impress.parameter.parameter_id = impress.procedure_has_parameters.parameter_id
    and impress.procedure_has_parameters.procedure_id = impress.procedure.procedure_id
    and procedure.type = 1
    and parameters_performed.parameter_id = parameter_key
    and is_annotation = 1
order by
    impress.parameter.name;
```

This query takes **~9.716** seconds to execute! It should therefore be optimised. But before we begin optimisation, we should check the remaining query that extracts the p-values, as we shall see in the following sections that we only need to optimise only a few of the tables and queries to obtain the required speedup.

Query to retrieve the p-values

Finally, the following is the query for retrieving the p-values:

```
select
    procedure_type.id,
    user,
    min(significance + 0)
from
    annotation,
    impress.procedure,
    impress.procedure_type,
    impress.parameter,
    impress.procedure_has_parameters
where
    evidence_code = impress.parameter.parameter_key
    and procedure_type.id = 1
    and impress.procedure.type = impress.procedure_type.id
```

```
    and impress.parameter.parameter_id = impress.procedure_has_parameters.parameter_id
    and impress.procedure_has_parameters.procedure_id = impress.procedure.procedure_id
group by
    procedure_type.id,
    user
order by
    procedure_type.id,
    (user + 0);
```

This query takes ~**2.615** seconds, and must be optimised.

Analysis of the queries

If we analyse the last two queries above, we can see that most of the query time is spent on determining all of the parameters that belong to a procedure type for which there are experimental data. Since all of the parameters for a given procedure type does not change frequently, we should be able to build an overview table that contains this information. *We only need to update this overview table when the association between procedures and parameters changes*.

Overview table schema

The following defines the schema for an overview table that contains association between parameters and procedure types:

```
create database phenodcc_heatmap;
use phenodcc_heatmap;
create table parameters_for_procedure_type (
    id int not null auto_increment,
    procedure_type int(11) not null,
    parameter_key varchar(100) not null,
    parameter_name varchar(255) not null,
    primary key (id)
) engine = innodb;
```

We then run the following query everytime the parameters and procedures in IMPReSS changes.

```
use impress;
insert into
    phenodcc_heatmap.parameters_for_procedure_type
(
    procedure_type,
    parameter_key,
    parameter_name
)
select distinct
    procedure.type,
    parameter.parameter_key,
    parameter.name
from (((procedure join procedure_type) join parameter) join procedure_has_parameters)
where ((parameter.is_annotation = 1)
    and (procedure.type = procedure_type.id)
    and (parameter.parameter_id = procedure_has_parameters.parameter_id)
    and (procedure_has_parameters.procedure_id = procedure.procedure_id))
order by
    parameter.name,
    procedure.type;
```

This query builds the lookup table which significantly helps us reduce all of the joins with several tables in the IMPReSS database. Building this table takes ~**0.301** seconds, but it helps use reduce all of the expensive queries above.

Replacement query for retrieving row entries

First, we modify the query that fills in `<row entries>`. The query is changed from

```
select distinct
    parameter_key,
    impress.parameter.name
```

```
from
    impress.procedure,
    impress.procedure_type,
    impress.parameter,
    impress.procedure_has_parameters,
    parameters_performed
where
    impress.procedure.type = impress.procedure_type.id
    and impress.parameter.parameter_id = impress.procedure_has_parameters.parameter_id
    and impress.procedure_has_parameters.procedure_id = impress.procedure.procedure_id
    and procedure.type = 1
    and parameters_performed.parameter_id = parameter_key
    and is_annotation = 1
order by
    impress.parameter.name;
```

to

```
select distinct
    p.parameter_key,
    p.parameter_name
from
    (europhenome.parameters_performed e join phenodcc_heatmap.parameters_for_procedure_type p)
where
    p.procedure_type = 1
    and e.parameter_id = p.parameter_key
order by
    p.parameter_name;
```

This now takes ~**1.248** seconds, as compared to ~**9.716** seconds in the original version. We can improve this further by adding the following indexes:

- `europhenome.parameters_performed.parameter_id`
- `phenodcc_heatmap.parameters_for_procedure_type.procedure_type`
- `phenodcc_heatmap.parameters_for_procedure_type.parameter_key`

When we run the above query again, it now takes ~**0.001** seconds, which is a significant performance improvement.

Replacement query for retrieving p-values

Finally, if we replace

```
select
    procedure_type.id,
    user,
    min(significance + 0)
from
    annotation,
    impress.procedure,
    impress.procedure_type,
    impress.parameter,
    impress.procedure_has_parameters
where
    evidence_code = impress.parameter.parameter_key
    and procedure_type.id = 1
    and impress.procedure.type = impress.procedure_type.id
    and impress.parameter.parameter_id = impress.procedure_has_parameters.parameter_id
    and impress.procedure_has_parameters.procedure_id = impress.procedure.procedure_id
group by
    procedure_type.id,
    user
order by
    procedure_type.id,
    (user + 0);
```

with the equivalent query

```
select
    p.procedure_type,
    e.user,
    min(e.significance + 0)
from
    (phenodcc_annotations.annotation e join phenodcc_heatmap.parameters_for_procedure_type p)
where
    e.evidence_code = p.parameter_key
    and p.procedure_type = 1
group by
    p.procedure_type,
    e.user
order by
    p.procedure_type,
    (e.user + 0);
```

we find that the query execution time has reduced from **~2.615** seconds to **~0.736** seconds. By adding an index for `phenodcc_annotations.annotation.evidence_code`, we can reduce the query execution time to **~0.036** seconds.

We can make similar alterations for the remaining queries.