PGQA Mode

GNU Emacs mode to parse, format and analyze SQL queries

Antonín Houska

This manual was generated from pgqa.texi, which is distributed with PGQA, or can be downloaded from `http://savannah.nongnu.org/projects/pgqa/`.

1 Introduction

PGQA (PostgreSQL Query Analyzer) is a major mode of GNU Emacs editor, designed to parse, format and analyze SQL queries for PostgreSQL database server.

Besides providing the user with particular functionality (accessible via menu and key sequences), the project aims to offer low-level functions to search in the query tree and to modify it.

2 Getting Started

First, make sure you have GNU Emacs 25 or later installed.

Second, copy the pgqa directory to the directory you usually install Elisp packages into, typically `~/.emacs.d/lisp/pgqa/`.

Then add the following lines to the init file of GNU Emacs (adjust *load-path* value if you decided to installed the package to a different directory):

```
;; Make GNU Emacs aware of PGQA mode.
(add-to-list 'load-path "~/.emacs.d/lisp/pgqa/")

;; Enable autoloading of the PGQA mode.
(autoload 'pgqa-mode "pgqa" "PGQA major mode function." t)

;; Make sure *.sql file suffix activates the PGQA mode automatically.
(add-to-list 'auto-mode-alist (cons "\\.sql\\'" 'pgqa-mode))

;; pgqa-format-query does not require the pgqa-mode. (Syntax
;; highlighting is not active w/o the pgqa-mode.)
(autoload 'pgqa-format-query "pgqa" "Format SQL query." t)
```

Finally restart the GNU Emacs editor. Once you open a file having `sql` suffix, major mode of the containing buffer should automatically become the PGQA mode. If the file has different suffix, or if you don't want to modify *auto-mode-alist*, you can use *`M-x pgqa-mode`* command to activate the mode.

Once the mode has been activated, "PGQA" string should appear in the mode line.

3 Query Parsing

If the PGQA mode is active, `C-c >` key sequence runs command `pgqa-parse`, which considers the current buffer to contain a single SQL query. The command parses the query string and puts the internal format (tree) to `pgqa-query-tree` buffer-local variable. The tree consists of nodes that represent query parts such as tables, joins, expressions, etc.

In addition, the `pgqa-parse` function creates an overlay for each node and sets `node` property of the overlay so it points back to the owning node object. Thus user can move point to arbitrary position of the query text and find out to which nodes of the query tree the position belongs — see Overlays in the Emacs Lisp documentation (`https://www.gnu.org/software/emacs/manual/html_node/elisp/Overlays.html#Overlays`). Some of the overlays can have text properties set, typically `font-lock-face`.

If you only want to parse part of the buffer (the current buffer may contain some other text besides the SQL query that PGQA parser does not recognize, e.g. constructs of PL/pgSQL language), or if the buffer contains multiple queries, user can mark the query (i.e. put it into "region"). Thus the `pgqa-parse` command only processes the region contents.

The region is remembered, so the there's no need to select the query again before the next run of `pgqa-parse`.

4 Query Formatting

4.1 Interactive Mode

If the PGQA mode is active, `C-c <` key sequence runs command `pgqa-format-query`, which considers the current buffer to contain a single SQL query. The command parses the query string, turns the internal format back into text and replaces the original query with it.

The command tries not to change position of the query within the buffer. In particular, the formatted query starts on the same line as the original did. If the first character of the query does not start at the beginning of a line, indentation of the whole query is adjusted so that the number of spaces in front of each line of the query is whole multiple of `tab-width`. (Spaces are added or removed so that the closest TAB position is reached.)

If the command is called with a prefix argument N, then N is considered the desired TAB position and no estimate is calculated.

If the region is used and the first line is preceded by at least one non-whitespace character, then the indentation is still estimated (or accepted as a prefix argument), but it's not applied to the first line. The idea is that user should know why the non-white character is there.

As for query selection (region), `pgqa-format-query` behaves in the same way as `pgqa-parse`, i.e. the last region is used for the next executions until user performs a new selection.

`pgqa-format-query` can (as long as autoloading is configured, see Chapter 2 [Getting Started], page 2) be used even in buffer whose major mode is not PGQA, but no syntax highlighting is active in that case. Also no key sequence is automatically bound to the command.

`fill-column` variable is honored during the formatting.

4.2 Batch Mode

PGQA can be used to format queries in batch mode. To ensure that the code gets loaded, add the containing directory to the value of `EMACSLOADPATH`. For example (the initial colon ensures that the existing value of *load-path* variable is not discarded):

```
export EMACSLOADPATH=:~/.emacs.d/lisp/pgqa/
```

Then run GNU Emacs this way (query.sql is the file containing your SQL query):

```
emacs -batch --insert query.sql -l pgqa.el -f pgqa-format-query-batch
```

The formatted query should be sent to the standard output.

4.3 Customization

This section summarizes the settings that affect SQL query formatting. `C-c +` key sequence runs `pgqa-customize` command which opens a window containing the related customization settings.

`pgqa-multiline-query` setting can effectively disable formatting if it's value is `nil`. In that case line is only broken if the column of the next character is greater than the value of `fill-column` setting.

For example, if `fill-column` is set to 40 (just to demonstrate how lines are broken, without using too complex query), `pgqa-format-query` will produce this:

```
SELECT p.name, l.name, r.version, r.date
FROM projects AS p JOIN licenses AS l ON
l.id = p.license_id JOIN releases AS r
ON r.project_id = p.id WHERE r.date <=
'2012-06-30' AND p.name LIKE '%GNU%'
ORDER BY r.date, p.name;
```

(Note that all the following formatting settings must be set to `nil` in this case, otherwise the formatting will result in error message.)

If you set `pgqa-multiline-query` option, each query "clause keyword" (`SELECT`, `FROM`, etc.) of the formatted query will start on a new line:

```
SELECT      p.name, l.name, r.version,
            r.date
FROM        projects AS p JOIN licenses
            AS l ON l.id = p.license_id
            JOIN releases AS r ON
            r.project_id = p.id
WHERE       r.date <= '2012-06-30' AND
            p.name LIKE '%GNU%'
ORDER BY    r.date, p.name;
```

`pgqa-clause-newline` can be set in addition to ensure that the actual "top-level clause" will start on a new line. The clause will be given extra indentation relative to the "clause keyword". For example, if the value of `tab-width` setting is equal to 4, the query will looke like this:

```
SELECT
    p.name, l.name, r.version, r.date
FROM
    projects AS p JOIN licenses AS l ON l.id =
    p.license_id JOIN releases AS r ON
    r.project_id = p.id
WHERE
    r.date <= '2012-06-30' AND p.name LIKE '%GNU%'
ORDER BY
    r.date, p.name;
```

`pgqa-clause-newline` requires `pgqa-multiline-query` to be set.

Furthermore, `pgqa-clause-item-newline` setting ensures that comma in the "top-level" clause is always followed by a new line:

```
SELECT
    p.name,
    l.name,
    r.version,
    r.date
FROM
    projects AS p JOIN licenses AS l ON l.id =
    p.license_id JOIN releases AS r ON
    r.project_id = p.id
WHERE
    r.date <= '2012-06-30' AND p.name LIKE '%GNU%'
ORDER BY
    r.date,
    p.name;
```

`pgqa-clause-item-newline` requires `pgqa-clause-newline` to be set.

`pgqa-multiline-join` setting ensures that `JOIN` keyword is always printed on a new line, following the appropriate indentation:

```
SELECT
    p.name,
    l.name,
    r.version,
    r.date
FROM
    projects AS p
    JOIN licenses AS l ON l.id = p.license_id
    JOIN releases AS r ON r.project_id = p.id
WHERE
    r.date <= '2012-06-30' AND p.name LIKE '%GNU%'
ORDER BY
    r.date,
    p.name;
```

`pgqa-multiline-join` requires `pgqa-multiline-query` to be set.

If `pgqa-join-newline` is enabled, line delimiter and indentation are also printed out in front of the right side of the join:

```
SELECT
    p.name,
    l.name,
    r.version,
    r.date
FROM
    projects AS p
    JOIN
    licenses AS l ON l.id = p.license_id
    JOIN
    releases AS r ON r.project_id = p.id
WHERE
    r.date <= '2012-06-30' AND p.name LIKE '%GNU%'
ORDER BY
    r.date,
    p.name;
```

`pgqa-join-newline` requires `pgqa-multiline-join` to be set.

If `pgqa-clause-keyword-right` is enabled, the clause keywords are right-aligned:

```
             SELECT p.name, l.name, r.version, r.date
               FROM projects AS p
                    JOIN
                    licenses AS l ON l.id = p.license_id
                    JOIN
                    releases AS r ON r.project_id = p.id
              WHERE r.date <= '2012-06-30' AND p.name LIKE '%GNU%'
           ORDER BY r.date, p.name
```

`pgqa-clause-keyword-right` requires `pgqa-multiline-query` to be set.

`pgqa-multiline-operator` setting can help you understand complex expressions. If this is set, operator expressions are printed out in structured way — exactly the way PGQA understands them:

```
SELECT
    p.name,
    l.name,
    r.version,
    r.date
FROM
    projects AS p
    JOIN
    licenses AS l ON
            l.id
        =
            p.license_id
    JOIN
    releases AS r ON
            r.project_id
        =
            p.id
WHERE
            r.date
        <=
            '2012-06-30'
    AND
            p.name
        LIKE
            '%GNU%'
ORDER BY
    r.date,
    p.name;
```

`pgqa-multiline-operator` requires `pgqa-multiline-join` to be set.

`pgqa-print-as` setting determines whether expression or table alias will be denoted by the `AS` keyword.