F.33. pg_trgm

The `pg_trgm` module provides functions and operators for determining the similarity of alphanumeric text based on trigram matching, as well as index operator classes that support fast searching for similar strings.

This module is considered "trusted", that is, it can be installed by non-superusers who have `CREATE` privilege on the current database.

F.33.1. Trigram (or Trigraph) Concepts

A trigram is a group of three consecutive characters taken from a string. We can measure the similarity of two strings by counting the number of trigrams they share. This simple idea turns out to be very effective for measuring the similarity of words in many natural languages.

> **Note**
>
> `pg_trgm` ignores non-word characters (non-alphanumerics) when extracting trigrams from a string. Each word is considered to have two spaces prefixed and one space suffixed when determining the set of trigrams contained in the string. For example, the set of trigrams in the string "`cat`" is "` c`", "` ca`", "`cat`", and "`at `". The set of trigrams in the string "`foo|bar`" is "` f`", "` fo`", "`foo`", "`oo `", "` b`", "` ba`", "`bar`", and "`ar `".

F.33.2. Functions and Operators

The functions provided by the `pg_trgm` module are shown in Table F.25, the operators in Table F.26.

Table F.25. `pg_trgm` Functions

Function Description
`similarity ( text, text ) → real` Returns a number that indicates how similar the two arguments are. The range of the result is zero (indicating that the two strings are completely dissimilar) to one (indicating that the two strings are identical).
`show_trgm ( text ) → text[]` Returns an array of all the trigrams in the given string. (In practice this is seldom useful except for debugging.)
`word_similarity ( text, text ) → real` Returns a number that indicates the greatest similarity between the set of trigrams in the first string and any continuous extent of an ordered set of trigrams in the second string. For details, see the explanation below.
`strict_word_similarity ( text, text ) → real` Same as `word_similarity`, but forces extent boundaries to match word boundaries. Since we don't have cross-word trigrams, this function actually returns greatest similarity between first string and any continuous extent of words of the second string.
`show_limit () → real` Returns the current similarity threshold used by the `%` operator. This sets the minimum similarity between two words for them to be considered similar enough to be misspellings of each other, for example. (*Deprecated*; instead use `SHOW pg_trgm.similarity_threshold`.)
`set_limit ( real ) → real` Sets the current similarity threshold that is used by the `%` operator. The threshold must be between 0 and 1 (default is 0.3). Returns the same value passed in. (*Deprecated*; instead use `SET pg_trgm.similarity_threshold`.)

Consider the following example:

```
# SELECT word_similarity('word', 'two words');
 word_similarity
-----------------
             0.8
(1 row)
```

In the first string, the set of trigrams is `{" w"," wo","wor","ord","rd "}`. In the second string, the ordered set of trigrams is `{" t"," tw","two","wo "," w"," wo","wor","ord","rds","ds "}`. The most similar extent of an ordered set of trigrams in the second string is `{" w"," wo","wor","ord"}`, and the similarity is `0.8`.

This function returns a value that can be approximately understood as the greatest similarity between the first string and any substring of the second string. However, this function does not add padding to the boundaries of the extent. Thus, the number of additional characters present in the second string is not considered, except for the mismatched word boundaries.

At the same time, `strict_word_similarity` selects an extent of words in the second string. In the example above, `strict_word_similarity` would select the extent of a single word `'words'`, whose set of trigrams is `{" w"," wo","wor","ord","rds","ds "}`.

```
# SELECT strict_word_similarity('word', 'two words'), similarity('word', 'words');
 strict_word_similarity | similarity
------------------------+------------
               0.571429 |   0.571429
(1 row)
```

Thus, the `strict_word_similarity` function is useful for finding the similarity to whole words, while `word_similarity` is more suitable for finding the similarity for parts of words.

Table F.26. `pg_trgm` Operators

Operator Description
`text % text → boolean` Returns `true` if its arguments have a similarity that is greater than the current similarity threshold set by `pg_trgm.similarity_threshold`.
`text <% text → boolean` Returns `true` if the similarity between the trigram set in the first argument and a continuous extent of an ordered trigram set in the second argument is greater than the current word similarity threshold set by `pg_trgm.word_similarity_threshold` parameter.
`text %> text → boolean` Commutator of the <% operator.
`text <<% text → boolean` Returns `true` if its second argument has a continuous extent of an ordered trigram set that matches word boundaries, and its similarity to the trigram set of the first argument is greater than the current strict word similarity threshold set by the `pg_trgm.strict_word_similarity_threshold` parameter.
`text %>> text → boolean` Commutator of the <<% operator.
`text <-> text → real` Returns the "distance" between the arguments, that is one minus the `similarity()` value.
`text <<-> text → real` Returns the "distance" between the arguments, that is one minus the `word_similarity()` value.
`text <->> text → real` Commutator of the <<-> operator.
`text <<<-> text → real` Returns the "distance" between the arguments, that is one minus the `strict_word_similarity()` value.
`text <->>> text → real` Commutator of the <<<-> operator.

F.33.3. GUC Parameters

`pg_trgm.similarity_threshold (real)`

Sets the current similarity threshold that is used by the % operator. The threshold must be between 0 and 1 (default is 0.3).

`pg_trgm.word_similarity_threshold (real)`

Sets the current word similarity threshold that is used by the <% and %> operators. The threshold must be between 0 and 1 (default is 0.6).

`pg_trgm.strict_word_similarity_threshold (real)`

Sets the current strict word similarity threshold that is used by the <<% and %>> operators. The threshold must be between 0 and 1 (default is 0.5).

F.33.4. Index Support

The `pg_trgm` module provides GiST and GIN index operator classes that allow you to create an index over a text column for the purpose of very fast similarity searches. These index types support the above-described similarity operators, and additionally support trigram-based index searches for `LIKE`, `ILIKE`, ~, ~* and = queries. Inequality operators are not supported. Note that those indexes may not be as efficient as regular B-tree indexes for equality operator.

Example:

```
CREATE TABLE test_trgm (t text);
CREATE INDEX trgm_idx ON test_trgm USING GIST (t gist_trgm_ops);
```

or

```
CREATE INDEX trgm_idx ON test_trgm USING GIN (t gin_trgm_ops);
```

`gist_trgm_ops` GiST opclass approximates a set of trigrams as a bitmap signature. Its optional integer parameter `siglen` determines the signature length in bytes. The default length is 12 bytes. Valid values of signature length are between 1 and 2024 bytes. Longer signatures lead to a more precise search (scanning a smaller fraction of the index and fewer heap pages), at the cost of a larger index.

Example of creating such an index with a signature length of 32 bytes:

```
CREATE INDEX trgm_idx ON test_trgm USING GIST (t gist_trgm_ops(siglen=32));
```

At this point, you will have an index on the `t` column that you can use for similarity searching. A typical query is

```
SELECT t, similarity(t, 'word') AS sml
  FROM test_trgm
  WHERE t % 'word'
  ORDER BY sml DESC, t;
```

This will return all values in the text column that are sufficiently similar to ***word***, sorted from best match to worst. The index will be used to make this a fast operation even over very large data sets.

A variant of the above query is

```
SELECT t, t <-> 'word' AS dist
  FROM test_trgm
  ORDER BY dist LIMIT 10;
```

This can be implemented quite efficiently by GiST indexes, but not by GIN indexes. It will usually beat the first formulation when only a small number of the closest matches is wanted.

Also you can use an index on the `t` column for word similarity or strict word similarity. Typical queries are:

```
SELECT t, word_similarity('word', t) AS sml
  FROM test_trgm
  WHERE 'word' <% t
  ORDER BY sml DESC, t;
```

and

```
SELECT t, strict_word_similarity('word', t) AS sml
  FROM test_trgm
  WHERE 'word' <<% t
  ORDER BY sml DESC, t;
```

This will return all values in the text column for which there is a continuous extent in the corresponding ordered trigram set that is sufficiently similar to the trigram set of ***word***, sorted from best match to worst. The index will be used to make this a fast operation even over very large data sets.

Possible variants of the above queries are:

```
SELECT t, 'word' <<-> t AS dist
  FROM test_trgm
  ORDER BY dist LIMIT 10;
```

and

```
SELECT t, 'word' <<<-> t AS dist
  FROM test_trgm
  ORDER BY dist LIMIT 10;
```

This can be implemented quite efficiently by GiST indexes, but not by GIN indexes.

Beginning in PostgreSQL 9.1, these index types also support index searches for `LIKE` and `ILIKE`, for example

```
SELECT * FROM test_trgm WHERE t LIKE '%foo%bar';
```

The index search works by extracting trigrams from the search string and then looking these up in the index. The more trigrams in the search string, the more effective the index search is. Unlike B-tree based searches, the search string need not be left-anchored.

Beginning in PostgreSQL 9.3, these index types also support index searches for regular-expression matches (~ and ~* operators), for example

```
SELECT * FROM test_trgm WHERE t ~ '(foo|bar)';
```

The index search works by extracting trigrams from the regular expression and then looking these up in the index. The more trigrams that can be extracted from the regular expression, the more effective the index search is. Unlike B-tree based searches, the search string need not be left-anchored.

For both `LIKE` and regular-expression searches, keep in mind that a pattern with no extractable trigrams will degenerate to a full-index scan.

The choice between GiST and GIN indexing depends on the relative performance characteristics of GiST and GIN, which are discussed elsewhere.

F.33.5. Text Search Integration

Trigram matching is a very useful tool when used in conjunction with a full text index. In particular it can help to recognize misspelled input words that will not be matched directly by the full text search mechanism.

The first step is to generate an auxiliary table containing all the unique words in the documents:

```
CREATE TABLE words AS SELECT word FROM
        ts_stat('SELECT to_tsvector(''simple'', bodytext) FROM documents');
```

where `documents` is a table that has a text field `bodytext` that we wish to search. The reason for using the `simple` configuration with the `to_tsvector` function, instead of using a language-specific configuration, is that we want a list of the original (unstemmed) words.

Next, create a trigram index on the word column:

```
CREATE INDEX words_idx ON words USING GIN (word gin_trgm_ops);
```

Now, a `SELECT` query similar to the previous example can be used to suggest spellings for misspelled words in user search terms. A useful extra test is to require that the selected words are also of similar length to the misspelled word.

> **Note**
>
> Since the `words` table has been generated as a separate, static table, it will need to be periodically regenerated so that it remains reasonably up-to-date with the document collection. Keeping it exactly current is usually unnecessary.

F.33.6. References

GiST Development Site http://www.sai.msu.su/~megera/postgres/gist/

Tsearch2 Development Site http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/

F.33.7. Authors

Oleg Bartunov <**oleg@sai.msu.su**>, Moscow, Moscow University, Russia

Teodor Sigaev <**teodor@sigaev.ru**>, Moscow, Delta-Soft Ltd.,Russia

Alexander Korotkov <**a.korotkov@postgrespro.ru**>, Moscow, Postgres Professional, Russia

Documentation: Christopher Kings-Lynne

This module is sponsored by Delta-Soft Ltd., Moscow, Russia.

Submit correction

If you see anything in the documentation that is not correct, does not match your experience with the particular feature or requires further clarification, please use this form to report a documentation issue.