Where’s Waldo?

Text Search and Pattern Matching in PostgreSQL

Joe Conway
conway@amazon.com
mail@joeconway.com

AWS
February 04, 2024

Where's Waldo?

- Many potential methods
- Usually best to use simplest method that fits use case
- Might need to combine more than one method

Agenda

- Summary of methods
- Overview by method
- Example use cases

Caveats

- Full text search could easily fill a tutorial
  $\Rightarrow$ this talk provides overview
- Even other methods cannot be covered exhaustively
  $\Rightarrow$ this talk provides overview
- citext not covered, should be considered

Text Search Methods

- Standard Pattern Matching
  - LIKE operator
  - SIMILAR TO operator
  - POSIX-style regular expressions
- PostgreSQL extensions
  - fuzzystrmatch
    - Soundex
    - Levenshtein
    - Metaphone
    - Double Metaphone
  - pg_trgm
- Full Text Search

Note on Extensions

- Extensions used are created as shown below

```
CREATE EXTENSION pg_trgm;
CREATE EXTENSION fuzzystrmatch;
```

Sample Data

```
-- make a table for the imported data
CREATE TABLE messages ([...] efrom text, eto text, subject text, body text, fti tsvector, [...]);
-- <import Postgres list mbox files>
-- populate the full text index vector column
UPDATE messages SET fti =
    setweight(to_tsvector(subject), 'A') || setweight(to_tsvector(body), 'D');
-- build the full text index
CREATE INDEX messages_fti_idx ON messages USING gin (fti);
Time: 322398.691 ms (05:22.399)
-- see what we got
select count(*) from messages;
  count
---------
 1533558
```

LIKE Syntax

- Expression returns TRUE if string matches pattern
- Typically string comes from relation in FROM clause
- Used as predicate in the WHERE clause to filter returned rows
- LIKE is case sensitive
- ILIKE is case insensitive

```
string LIKE pattern [ESCAPE escape-character]
string ~~ pattern [ESCAPE escape-character]
string ILIKE pattern [ESCAPE escape-character]
string ~~* pattern [ESCAPE escape-character]
lower(string) LIKE pattern [ESCAPE escape-character]
```

Negating LIKE

- To negate match, use the NOT keyword
- Appropriate operator also works

```
string NOT LIKE pattern [ESCAPE escape-character]
string !~~ pattern [ESCAPE escape-character]
string NOT ILIKE pattern [ESCAPE escape-character]
string !~~* pattern [ESCAPE escape-character]
NOT (string LIKE pattern [ESCAPE escape-character])
NOT (string ILIKE pattern [ESCAPE escape-character])
```

Wildcards

- Pattern can contain wildcard characters
  - Underscore ("_") matches any single character
  - Percent sign ("%") matches zero or more characters
- With no wildcards, expression acts like equals
- To match literal wildcard chars, they must be escaped
- Default escape char is backslash ("\")
  - May be changed using ESCAPE clause
  - Match the literal escape char by doubling up

Alternate Index Op Classes

- `varchar_pattern_ops`, `text_pattern_ops` and `bpchar_pattern_ops`
- Useful for anchored pattern matching, e.g. "`<pattern>`%"
- Used by LIKE, SIMILAR TO, or POSIX regex when not using "C" locale
- Also create "normal" index for queries with <, <=, >, or >=
- Does NOT work for ILIKE or ~~*
  - Expression index over lower(column)
  - pg_trgm index operator class

ESCAPE Example

```
SELECT 'A\b\C_%_dEf' LIKE 'A\b\C#_#%#_d%' ESCAPE '#';
?column?
----------
t
(1 row)
```

SIMILAR TO Syntax

- Equivalent to LIKE
- Interprets pattern using SQL definition of regex

```
string SIMILAR TO pattern [ESCAPE escape-character]
string NOT SIMILAR TO pattern [ESCAPE escape-character]
```

Wildcards

- Same as `LIKE`
- Also supports meta-characters borrowed from POSIX REs
  - pipe ("`|`"): either of two alternatives
  - asterisk ("`*`"): repetition >= 0 times
  - plus ("+"): repetition >= 1 time
  - question mark ("`?`"): repetition 0 or 1 time
  - "`{m}`": repetition exactly m times
  - "`{m,}`": repetition >= m times
  - "`{m,n}`": repetition >= m and <= n times
  - parentheses ("`()`"): group items into a single logical item

SIMILAR TO Examples

```
SELECT 'AbCdEf' SIMILAR TO 'AbC%' AS true,
       'AbCdEf' SIMILAR TO 'Ab(C|c)%' AS true,
       'Abcccdef' SIMILAR TO 'Abc{4}%' AS false,
       'Abcccdef' SIMILAR TO 'Abc{3}%' AS true,
       'Abccdef' SIMILAR TO 'Abc?d?%' AS true;
 true | true | false | true | true
------+------+-------+------+------
 t    | t    | f     | t    | t
(1 row)
```

Regular Expression Syntax

- Similar to LIKE and ILIKE
- Allowed to match anywhere within string
  $\Rightarrow$ unless RE is explicitly anchored
- Interprets pattern using POSIX definition of regex

```
string ~ pattern -- matches RE, case sensitive
string ~* pattern -- matches RE, case insensitive
string !~ pattern -- not matches RE, case sensitive
string !~* pattern -- not matches RE, case insensitive
```

Regular Expression Syntax

- POSIX-style REs complex enough to deserve own talk
- See: www.postgresql.org/docs/9.5/static/functions-matching.html#FUNCTIONS-POSIX-REGEXP

```
SELECT 'AbCdefzzzzdef' ~* 'Ab((C|c).*)?z+def.*' AS true,
       'AbcabcAbc' ~ '^Ab.*bc$' AS true,
       'AbcabcAbc' ~ '^Ab' AS true,
       'AbcAbcAbc' ~* 'abc' AS true,
       'AbcAbcAbc' ~* '^abc$' AS false;
 true | true | true | true | false
------+------+------+------+-------
 t    | t    | t    | t    | f
(1 row)
```

Regular Expression Example

- Really slow without an index

```
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE body ~* 'multixact';
                   QUERY PLAN
-----------------------------------------------------
 Gather  (cost=1000.00..269313.46 rows=15486 width=8)
         (actual time=8.540..5373.761 rows=4809 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on messages
   [...]
 Planning Time: 0.514 ms
 JIT:
 [...]
 Execution Time: 5374.568 ms
```

Regular Expression Example

- Use trigram GIN index

```
CREATE INDEX trgm_gin_body_idx
ON messages USING gin (body gin_trgm_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE body ~* 'multixact';
                   QUERY PLAN
-----------------------------------------------------
 Bitmap Heap Scan on messages
   [...]
   ->  Bitmap Index Scan on trgm_gin_bodytxt_idx
       (cost=0.00..230.73 rows=15442 width=0)
       (actual time=11.648..11.648 rows=4819 loops=1)
         Index Cond: (body ~* 'multixact'::text)
 Planning Time: 0.564 ms
 Execution Time: 182.216 ms
(8 rows)
```

Regular Expression Example

- Or use trigram GiST index . . . oops

```
CREATE INDEX trgm_gist_body_idx
ON messages USING gist (body gist_trgm_ops);
ERROR:  index row requires 39296 bytes, maximum size is 8191
```

Regular Expression Compared to FTS

- For the sake of comparison - with full text search

```
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE fti @@ 'multixact:D';
          QUERY PLAN
-----------------------------------------------------------------
 Bitmap Heap Scan on messages
   [...]
   ->  Bitmap Index Scan on messages_fti_idx
         (cost=0.00..72.20 rows=7668 width=0)
         (actual time=0.401..0.402 rows=3382 loops=1)
         Index Cond: (fti @@ '''multixact'':D'::tsquery)
 Planning Time: 0.099 ms
 Execution Time: 21.924 ms
(8 rows)
```

Soundex

- soundex: converts string to four character code
- difference: converts two strings, reports # matching positions
- Generally finds similarity of English names
- Part of fuzzystrmatch extension

```
SELECT soundex('Joseph'), soundex('Josef'),
       difference('Joseph', 'Josef');
 soundex | soundex | difference
---------+---------+------------
 J210    | J210    |          4
```

Levenshtein

- Calculates Levenshtein distance between two strings
- Comparisons case sensitive
- Strings non-null, maximum 255 bytes
- Part of fuzzystrmatch extension

```
SELECT levenshtein('Joseph','Josef') AS two,
       levenshtein('John','Joan') AS one,
       levenshtein('foo','foo') AS zero;
 two | one | zero
-----+-----+------
   2 |   1 |    0
```

Metaphone

- Constructs code for an input string
- Comparisons case in-sensitive
- Strings non-null, maximum 255 bytes
- `max_output_length` arg sets max length of code
- Part of fuzzystrmatch extension

```
SELECT metaphone('extensive',6) AS "EKSTNS",
       metaphone('exhaustive',6) AS "EKSHST",
       metaphone('ExTensive',3) AS "EKS",
       metaphone('eXhaustivE',3) AS "EKS";
 EKSTNS | EKSHST | EKS | EKS
--------+--------+-----+-----
 EKSTNS | EKSHST | EKS | EKS
```

Double Metaphone

- Computes primary and alternate codes for string
- Non-English names especially, can be different
- Comparisons case in-sensitive
- No length limit on the input strings
- Part of fuzzystrmatch extension

```
SELECT dmetaphone('extensive') AS "AKST",
       dmetaphone('exhaustive') AS "AKSS",
       dmetaphone('Magnus') AS "MNS",
       dmetaphone_alt('Magnus') AS "MKNS";
 AKST | AKSS | MNS | MKNS
------+------+-----+------
 AKST | AKSS | MNS | MKNS
```

Trigram Matching

- Functions and operators for determining similarity
- Trigram is group of three consecutive characters from string
- Similarity of two strings - count number of trigrams shared
- Index operator classes supporting fast similar strings search
- Support indexed searches for LIKE and ILIKE queries
- Comparisons case in-sensitive
- Part of pg_trgm extension

Trigram Matching Example

```
\timing
SELECT set_limit(0.6); -- defaults to 0.3
SELECT DISTINCT efrom, -- uses trgm_gist_idx
similarity(efrom, 'Josef Konway <mail@joeconway.com>') AS sml
FROM messages WHERE efrom % 'Josef Konway <mail@joeconway.com>'
ORDER BY sml DESC, efrom;
                efrom                 |    sml
--------------------------------------+-----------
 Joseph Conway <mail@joeconway.com> | 0.7241379
 Joe Conway <mail@joeconway.com>    | 0.7037037
 jconway <mail@joeconway.com>       | 0.6785714
 "Joe Conway" <joe.conway@mail.com> | 0.6296296
(4 rows)

Time: 419.809 ms
```

Overview

- Searches documents with potentially complex criteria
- Superior to other methods in many cases because:
  - Offers linguistic support for derived words
  - Ignores stop words
  - Ranks results by relevance
  - Very flexibly uses indexes
- Topic very complex - see:

```
http://www.postgresql.org/docs/current/static/textsearch.html
http://www.postgresql.org/docs/current/static/datatype-textsearch.html
http://www.postgresql.org/docs/current/static/functions-textsearch.html
http://www.postgresql.org/docs/current/static/textsearch-indexes.html
```

Preprocessing

- Convert text to tsvector
- Store tsvector
- Index tsvector

```
CREATE FUNCTION messages_fti_trigger_func()
RETURNS trigger LANGUAGE plpgsql AS $$
BEGIN NEW.fti =
  setweight(to_tsvector(coalesce(NEW.subject, '')), 'A') ||
  setweight(to_tsvector(coalesce(NEW.body, '')), 'D');
RETURN NEW; END $$;

CREATE TRIGGER messages_fti_trigger BEFORE INSERT OR UPDATE
OF subject, body ON messages FOR EACH ROW
EXECUTE PROCEDURE messages_fti_trigger_func();

CREATE INDEX messages_fti_idx ON messages USING gin (fti);
```

Weighting

- Weights used in relevance ranking
- Array specifies how heavily to weigh each category
- `{D-weight, C-weight, B-weight, A-weight}`
- defaults: `{0.1, 0.2, 0.4, 1.0}`

Creating tsvector

- Parse into tokens
  - Classes of tokens can be processed differently
  - Postgres has standard parser and predefined set of classes
  - Custom parsers can be created
- Convert tokens into lexemes
  - Dictionaries used for this step
    ⇒ standard dictionaries provided
    ⇒ custom ones can be created
  - Normalized: different forms of same word made alike
    ⇒ fold upper-case letters to lower-case
    ⇒ removal of suffixes
    ⇒ elimination of stop words

Writing tsquery

- The pattern to be matched
- Lexemes combined with boolean operators
  - `&` (AND)
  - `|` (OR)
  - `!` (NOT)
- `!` (NOT) binds most tightly
  `&` (AND) binds more tightly than `|` (OR)
- Parentheses used to enforce grouping
- Label with * to specify prefix matching
- Supports weight labels

Writing tsquery

- `to_tsquery()` to create
  - Normalizes tokens lexemes
  - Discards stop words
- Can also cast to tsquery
  - Tokens taken at face value
  - No weight labels

```
SELECT to_tsquery('(hello:A & the:A) | writing:D') AS to_tsq,
       '(hello & the) | writing'::tsquery AS cast_tsq
UNION ALL
SELECT to_tsquery('postgres:*'),
       'postgres:*'::tsquery;
        to_tsq         |          cast_tsq
-----------------------+----------------------------
 'hello':A | 'write':D | 'hello' & 'the' | 'writing'
 'postgr':*            | 'postgres':*
```

Writing tsquery

- Alternative function plainto_tsquery()
  - Text parsed and normalized
  - & (AND) operator inserted between surviving words
  - Should use simple strings only
  - No boolean operators,
    No weight labels,
    No prefix-match labels

```
SELECT plainto_tsquery('(hello:B & the:B) | writing:D') AS tsq1,
       plainto_tsquery('postgres:*') AS tsq2;
                tsq1                 |   tsq2
-------------------------------------+----------
 'hello' & 'b' & 'b' & 'write' & 'd' | 'postgr'
```

Match Operator

- Text search match operator `@@`
- Returns true if tsvector (preprocessed document) matches tsquery (search pattern)
- Either maybe be written first

```
SELECT split_part(efrom, '<', 1) AS name, ts FROM messages
WHERE fti @@ 'multixact:A & race:D & bug:D' ORDER BY 2;
      name       |           ts
-----------------+------------------------
 Alvaro Herrera  | 2013-11-25 10:36:19-05
 Andres Freund   | 2013-11-25 11:26:55-05
 Andres Freund   | 2013-11-29 14:58:06-05
 [...]
 Robert Haas     | 2015-07-02 13:58:45-04
(10 rows)
```

Relevance Ranking

- ts_rank(): based on frequency of matching lexemes
- ts_rank_cd(): lexeme proximity taken into consideration

```
WITH ts(q) AS
(SELECT to_tsquery('multixact:A & (crash:D | (data:D & loss:D))'))
SELECT ts_rank(m.fti, ts.q) as tsrank
FROM messages m, ts WHERE m.fti @@ ts.q
ORDER BY tsrank DESC LIMIT 4;
  tsrank
-----------
 0.9999975
 0.9999971
 0.9999968
 0.9999968
```

Highlighting

- ts_headline(): returns excerpt with query terms highlighted
- Apply in an outer query, after inner query LIMIT
  ⇒ avoids ts_headline() overhead on eliminated rows

```
SELECT subject, tsrank, ts_headline(format('%s: %s', subject, body), q)
FROM (WITH ts(q) AS
      (SELECT to_tsquery('multixact:A & (crash:D | (data:D & loss:D))'))
      SELECT ts_rank(m.fti, ts.q) as tsrank, ts.q, m.subject, m.body
      FROM messages m, ts WHERE m.fti @@ ts.q ORDER BY tsrank DESC LIMIT 4
     ) AS inner_query LIMIT 1;
-[ RECORD 1 ]-----------------------------------------------------------
subject     | Is anyone aware of data loss causing MultiXact bugs in 9.3.2?
tsrank      | 0.9999975
ts_headline | <b>data</b> <b>loss</b> causing <b>MultiXact</b>
              bugs in 9.3.2?: I've had multiple complaints
              of apparent <b>data</b>
```

Pattern Matching: Example Use Cases

- Equal
- Anchored
- Anchored case-insensitive
- Reverse Anchored case-insensitive
- Unanchored case-insensitive
- Fuzzy
- Complex Search with Relevancy Ranking

Equal

- Find all the rows where column matches `'<pattern>'`
- Equal operator with suitable index is best
- Without an index

```
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom = 'Joseph Conway <mail@joeconway.com>';
          QUERY PLAN
-----------------------------------------------------------------------
 Gather [...] Workers Launched: 2 ->  Parallel Seq Scan on messages
     (cost=0.00..486545.28 rows=38 width=8)
     (actual time=151.101..408.027 rows=5 loops=3)
     Filter: (efrom = 'Joseph Conway <mail@joeconway.com>'::text)
     Rows Removed by Filter: 1533543
 Planning Time: 0.081 ms
 Execution Time: 425.212 ms
```

Equal

- With an index

```
CREATE INDEX from_idx ON messages(efrom);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom = 'Joseph Conway <mail@joeconway.com>';
          QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using from_idx on messages
       (cost=0.43..363.15 rows=91 width=8)
       (actual time=0.025..0.035 rows=14 loops=1)
   Index Cond: (efrom = 'Joseph Conway <mail@joeconway.com>'::text)
 Planning Time: 0.085 ms
 Execution Time: 0.051 ms
```

Anchored

- Find all the rows where column matches `'<pattern>%'`
- LIKE operator with suitable index is best
- This index does not do the job

```
CREATE INDEX from_idx ON messages(efrom);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom LIKE 'Joseph Conway%';
          QUERY PLAN
-----------------------------------------------------------------------
 Gather  Workers Launched: 2 ->  Parallel Seq Scan on messages
   (cost=0.00..486545.28 rows=36 width=8)
   (actual time=97.537..440.006 rows=5 loops=3)
         Filter: (efrom ~~ 'Joseph Conway%'::text)
         Rows Removed by Filter: 1533543
 Planning Time: 0.133 ms
 Execution Time: 474.500 ms
```

Anchored

- Note the `text_pattern_ops` - this works

```
CREATE INDEX pattern_idx ON messages(efrom text_pattern_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom LIKE 'Joseph Conway%';
          QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using pattern_idx on messages
   (cost=0.43..8.45 rows=86 width=8)
   (actual time=0.022..0.035 rows=14 loops=1)
   Index Cond: ((efrom ~>=~ 'Joseph Conway'::text)
               AND (efrom ~<~ 'Joseph Conwaz'::text))
   Filter: (efrom ~~ 'Joseph Conway%'::text)
 Planning Time: 0.122 ms
 Execution Time: 0.051 ms
```

Anchored Case-Insensitive

- Find all the rows where column matches `'<pattern>%'`
  $\Rightarrow$ but in Case-Insensitive way
- LIKE operator with suitable expression index is good

```
CREATE INDEX lower_pattern_idx ON messages(lower(efrom) text_pattern_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE lower(efrom) LIKE 'joseph conway%';
          QUERY PLAN
------------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
   ->  Bitmap Index Scan on lower_pattern_idx
         (cost=0.00..117.11 rows=7668 width=0)
         (actual time=0.017..0.017 rows=14 loops=1)
         Index Cond: ((lower(efrom) ~>=~ 'joseph conway'::text)
                     AND (lower(efrom) ~<~ 'joseph conwaz'::text))
 Planning Time: 0.081 ms
 Execution Time: 0.076 ms
```

Anchored Case-Insensitive

- Can also use trigram GIN index with ILIKE

```
CREATE INDEX trgm_gin_idx
ON messages USING gin (efrom gin_trgm_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom ILIKE 'joseph conway%';
          QUERY PLAN
------------------------------------------------------------------------
 Bitmap Heap Scan on messages
   [...]
   ->  Bitmap Index Scan on trgm_gin_idx
         (cost=0.00..193.56 rows=86 width=0)
         (actual time=14.071..14.072 rows=155 loops=1)
         Index Cond: (efrom ~~* 'joseph conway%'::text)
 Planning Time: 0.288 ms
 Execution Time: 14.416 ms
```

Anchored Case-Insensitive

- Or a trigram GiST index with ILIKE

```
CREATE INDEX trgm_gist_idx
ON messages USING gist (efrom gist_trgm_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom ILIKE 'joseph conway%';
          QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using trgm_gist_idx on messages
    (cost=0.41..353.92 rows=86 width=8)
    (actual time=2.606..42.746 rows=14 loops=1)
   Index Cond: (efrom ~~* 'joseph conway%'::text)
   Rows Removed by Index Recheck: 141
 Planning Time: 0.268 ms
 Execution Time: 42.769 ms
```

Reverse Anchored Case-Insensitive

- Find all the rows where column matches `'%<pattern>'`
  $\Rightarrow$ but in Case-Insensitive way
- LIKE operator with suitable expression index is good

```
CREATE INDEX rev_lower_pattern_idx ON messages(lower(reverse(efrom)) text_pattern_ops);
EXPLAIN ANALYZE SELECT ts FROM messages WHERE lower(reverse(efrom))
                LIKE reverse('%joeconway.com>');
             QUERY PLAN
-------------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
  ->  Bitmap Index Scan on rev_lower_pattern_idx
        (cost=0.00..117.11 rows=7668 width=0)
        (actual time=0.325..0.325 rows=3970 loops=1)
        Index Cond: ((lower(reverse(efrom)) ~>=~ '>moc.yawnoceoj'::text)
                     AND (lower(reverse(efrom)) ~<~ '>moc.yawnoceok'::text))
 Planning Time: 0.107 ms
 Execution Time: 6.402 ms
```

Reverse Anchored Case-Insensitive

- Can also use trigram GIN index with ILIKE

```
CREATE INDEX trgm_gin_idx
ON messages USING gin (efrom using gin_trgm_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom ILIKE '%joeconway.com>';
          QUERY PLAN
-------------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
   ->  Bitmap Index Scan on trgm_gin_idx
         (cost=0.00..200.54 rows=4073 width=0)
         (actual time=23.774..23.774 rows=3970 loops=1)
         Index Cond: (efrom ~~* '%joeconway.com>'::text)
 Planning Time: 0.412 ms
 Execution Time: 30.870 ms
```

Reverse Anchored Case-Insensitive

- Or a trigram GiST index with ILIKE

```
CREATE INDEX trgm_gist_idx
ON messages USING gist (efrom gist_trgm_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom ILIKE '%joeconway.com>';
          QUERY PLAN
-----------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
   ->  Bitmap Index Scan on trgm_gist_idx
         (cost=0.00..342.96 rows=4073 width=0)
         (actual time=43.055..43.056 rows=3970 loops=1)
         Index Cond: (efrom ~~* '%joeconway.com>'::text)
 Planning Time: 0.270 ms
 Execution Time: 49.870 ms
```

Unanchored Case-Insensitive

- Find all the rows where column matches `'%<pattern>%'`
  $\Rightarrow$ but in Case-Insensitive way

- This cannot use expression or pattern_ops index ☹

```
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom ILIKE '%Conway%';
          QUERY PLAN
-----------------------------------------------------------------------
 Gather [...] Workers Launched: 2 ->  Parallel Seq Scan on messages
   (cost=0.00..486545.28 rows=3060 width=8)
   (actual time=15.897..881.018 rows=2881 loops=3)
         Filter: (efrom ~~* '%Conway%'::text)
         Rows Removed by Filter: 1524915
 Planning Time: 0.315 ms
 Execution Time: 903.556 ms
```

Unanchored Case-Insensitive

- Use trigram GIN index with ILIKE

```
CREATE INDEX trgm_gin_idx
ON messages USING gin (efrom gin_trgm_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom ILIKE '%Conway%';
          QUERY PLAN
------------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
   ->  Bitmap Index Scan on trgm_gin_idx
         (cost=0.00..100.91 rows=7345 width=0)
         (actual time=1.939..1.939 rows=8643 loops=1)
         Index Cond: (efrom ~~* '%Conway%'::text)
 Planning Time: 0.263 ms
 Execution Time: 16.097 ms
```

Unanchored Case-Insensitive

- Or a trigram GiST index with ILIKE

```
CREATE INDEX trgm_gist_idx
ON messages USING gist (efrom gist_trgm_ops);
EXPLAIN ANALYZE SELECT ts FROM messages
                WHERE efrom ILIKE '%Conway%';
          QUERY PLAN
-------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
   ->  Bitmap Index Scan on trgm_gist_idx
         (cost=0.00..615.50 rows=7345 width=0)
         (actual time=91.825..91.825 rows=8643 loops=1)
         Index Cond: (efrom ~~* '%Conway%'::text)
 Planning Time: 0.264 ms
 Execution Time: 105.172 ms
```

Fuzzy

- Find rows where column matches `'<pattern>'` $\Rightarrow$ but in inexact way
- Use dmetaphone function with an expression index
- Might also use Soundex, Levenshtein, Metaphone, or pg_trgm

```
CREATE INDEX dmet_expr_idx ON messages(dmetaphone(efrom));
EXPLAIN ANALYZE SELECT efrom FROM messages
WHERE dmetaphone(efrom) = dmetaphone('josef konwei');
          QUERY PLAN
-----------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
   ->  Bitmap Index Scan on dmet_expr_idx
         (cost=0.00..64.94 rows=7668 width=0)
         (actual time=0.029..0.029 rows=236 loops=1)
         Index Cond: (dmetaphone(efrom) = 'JSFK'::text)
 Planning Time: 0.094 ms
 Execution Time: 0.261 ms
```

Complex Requirements

- Full Text Search
  - Complex multi-word searching with Relevancy Ranking

```
EXPLAIN ANALYZE SELECT ts FROM messages
WHERE fti @@ 'bug:A & deadlock:D & startup:D';
          QUERY PLAN
------------------------------------------------------------------------
 Bitmap Heap Scan on messages [...]
   ->  Bitmap Index Scan on messages_fti_idx
         (cost=0.00..20.61 rows=4 width=0)
         (actual time=3.838..3.838 rows=271 loops=1)
         Index Cond: (fti @@ '''bug'':A & ''deadlock'':D &
                             ''startup'':D'::tsquery)
 Planning Time: 0.144 ms
 Execution Time: 8.395 ms
```

Questions?

Thank You!
conway@amazon.com
mail@joeconway.com