SQL search for keywords and statistics

Markus Schatten, PhD

Faculty of Organization and Informatics,
University of Zagreb
Pavlinska 2, 42000 Varaždin
markus.schatten@foi.hr

08.04.2011.

Hochschule
Bonn-Rhein-Sieg

Outline

Tools

- PostgreSQL 8.4

Tools

- PostgreSQL 8.4
- OpenOffice.org Base
  - Connect with `host=localhost dbname=enron port=5432` username `ubuntu`

Why pattern matching in databases?

- “Good” databases often do not need pattern matching

Why pattern matching in databases?

- “Good” databases often do not need pattern matching
- A curious case:
  - field name “`data`”
  - type `varchar( 200 )`
  - structure: first 10 characters internal user ID (numeric), then user’s surname ended by a ’$’ sign, then user’s name again ended by a ’$’ sign, one character indicator of user type ...

Why pattern matching in databases?

- “Good” databases often do not need pattern matching
- A curious case:
  - field name “`data`”
  - type `varchar( 200 )`
  - structure: first 10 characters internal user ID (numeric), then user’s surname ended by a ’$’ sign, then user’s name again ended by a ’$’ sign, one character indicator of user type ...
- Databases with lots of textual information (web applications, document management systems, ...)

Why pattern matching in databases?

- “Good” databases often do not need pattern matching
- A curious case:
  - field name “`data`”
  - type `varchar( 200 )`
  - structure: first 10 characters internal user ID (numeric), then user’s surname ended by a ’$’ sign, then user’s name again ended by a ’$’ sign, one character indicator of user type ...
- Databases with lots of textual information (web applications, document management systems, ...)
- ...

Outline

The `LIKE` expression

- Standard SQL pattern matching expression

The `LIKE` expression

- Standard SQL pattern matching expression
- Uses simple wildcard characters to match strings
  - `%` matches any 0 or more characters
  - `_` matches exactly one character

The `LIKE` expression

- Standard SQL pattern matching expression
- Uses simple wildcard characters to match strings
  - `%` matches any 0 or more characters
  - `_` matches exactly one character
- Usefull for keyword search and (very) simple patterns

```
SELECT 'e-Discovery' LIKE '%e%very%'

SELECT 'e-Discovery' LIKE '%Disco%'

SELECT 'e-Discovery' LIKE '___is______'

SELECT 'e-Discovery' NOT LIKE '_e%'

SELECT 'e-Discovery' NOT LIKE '%asy%'

SELECT 'e-Discovery' LIKE '%over_'
```

```
SELECT *
FROM emails
WHERE subject LIKE '%risk%'

SELECT *
FROM emails
WHERE "From" LIKE '%@enron.com%'
```

The `ILIKE` expression

- PostgreSQL specific pattern matching expression (not standard!)

The `ILIKE` expression

- PostgreSQL specific pattern matching expression (not standard!)
- Same functionality as `LIKE` but case insensitive

```
SELECT *
FROM emails
WHERE subject ILIKE '%risk%'

SELECT *
FROM emails
WHERE recipients NOT ILIKE '%@enron.com%'
```

LIKE and ILIKE

- Shortcuts:
  - ~~ = LIKE
  - !~~ = NOT LIKE
  - ~~* = ILIKE
  - !~~* = NOT ILIKE

`LIKE` and `ILIKE`

- Shortcuts:
  - `~~ = LIKE`
  - `!~~ = NOT LIKE`
  - `~~* = ILIKE`
  - `!~~* = NOT ILIKE`
- Advantage - very fast

`LIKE` and `ILIKE`

- Shortcuts:
  - `~~ = LIKE`
  - `!~~ = NOT LIKE`
  - `~~* = ILIKE`
  - `!~~* = NOT ILIKE`
- Advantage - very fast
- Dissadvantage - very limited

The `SIMILAR TO` expression

- “a curious cross between `LIKE` notation and common regular expression notation”

The `SIMILAR TO` expression

- "a curious cross between `LIKE` notation and common regular expression notation"
- Uses SQL standard definition of regular expressions

The `SIMILAR TO` expression

- “a curious cross between `LIKE` notation and common regular expression notation”
- Uses SQL standard definition of regular expressions
- Like the `LIKE` expression always match the whole string

`SIMILAR TO` metacharacters

- `%` matches any 0 or more characters

`SIMILAR TO` metacharacters

- `%` matches any 0 or more characters
- `_` matches exactly one character

`SIMILAR TO` metacharacters

- `%` matches any 0 or more characters
- `_` matches exactly one character
- `|` denotes alternation (either of two alternatives)

`SIMILAR TO` metacharacters

- `%` matches any 0 or more characters
- `_` matches exactly one character
- `|` denotes alternation (either of two alternatives)
- `*` denotes repetition of the previous item zero or more times

SIMILAR TO metacharacters

- % matches any 0 or more characters
- _ matches exactly one character
- | denotes alternation (either of two alternatives)
- * denotes repetition of the previous item zero or more times
- + denotes repetition of the previous item one or more times

`SIMILAR TO` metacharacters

- `%` matches any 0 or more characters
- `_` matches exactly one character
- `|` denotes alternation (either of two alternatives)
- `*` denotes repetition of the previous item zero or more times
- `+` denotes repetition of the previous item one or more times
- `( )` parentheses can be used to group items into a single logical item

`SIMILAR TO` metacharacters

- `%` matches any 0 or more characters
- `_` matches exactly one character
- `|` denotes alternation (either of two alternatives)
- `*` denotes repetition of the previous item zero or more times
- `+` denotes repetition of the previous item one or more times
- `()` parentheses can be used to group items into a single logical item
- `[...]` a bracket expression specifies a character class

```
SELECT 'never' SIMILAR TO 'never'

SELECT 'gonna' NOT SIMILAR TO 'gon'

SELECT 'give' SIMILAR TO '%(i|w)%'

SELECT 'you up' NOT SIMILAR TO '(you_)|(up)'
```

```
SELECT 'email: mschatte@foi.hr'
  SIMILAR TO 'email: *%@foi.hr'

SELECT 'email:   mschatte@foi.hr'
  SIMILAR TO 'email: *%@foi.hr'
```

```
SELECT 'foooooo' SIMILAR TO 'f(o)*'

SELECT 'foooooo' SIMILAR TO 'f(o)+'

SELECT 'f' SIMILAR TO 'f(o)*'

SELECT 'f' NOT SIMILAR TO 'f(o)+'
```

Bracket expressions

`[0-9]` matches any digit (0-9)

Bracket expressions

`[0-9]` matches any digit (0-9)

`[a-z]` matches any lowercase character

Bracket expressions

`[0-9]` matches any digit (0-9)

`[a-z]` matches any lowercase character

`[A-Z]` matches any uppercase character

Bracket expressions

`[0-9]` matches any digit (0-9)

`[a-z]` matches any lowercase character

`[A-Z]` matches any uppercase character

`[.?#]` matches either the '.', the '?' or the '#' character

Bracket expressions

`[0-9]` matches any digit (0-9)

`[a-z]` matches any lowercase character

`[A-Z]` matches any uppercase character

`[.?#]` matches either the '.', the '?' or the '#' character

`[a-z3-6.,]` matches a lowercase character, a digit from 3 to 6, the '.' or the ',' character

Bracket expressions

`[0-9]` matches any digit (0-9)

`[a-z]` matches any lowercase character

`[A-Z]` matches any uppercase character

`[.?#]` matches either the '.', the '?' or the '#' character

`[a-z3-6.,]` matches a lowercase character, a digit from 3 to 6, the '.' or the ',' character

...

```
SELECT '27 October 2001'
  SIMILAR TO
    '[0-9]+ [A-Z][a-z]+ [1-2][0-9][0-9][0-9]'

SELECT *
FROM emails
WHERE subject
  SIMILAR TO
    '% [0-9]+ [A-Z][a-z]+ [1-2][0-9][0-9][0-9]%'
```

Extracting strings with SUBSTRING/3

- The substring function with three parameters, `substring(`*`string` `from` `pattern` `for` `escape-character`*`)`, provides extraction of a substring that matches an SQL regular expression pattern

Extracting strings with SUBSTRING/3

- The substring function with three parameters, `substring(`*`string`* `from` *`pattern`* `for` *`escape-character`*`)`, provides extraction of a substring that matches an SQL regular expression pattern
- *`pattern`* - pattern as with `SIMILAR TO`

```
SELECT substring(
  'An important date was 27th October 2001 since ...'
  from '% #"[0-9]+th [A-Z][a-z]+ [1-2][0-9][0-9][0-9]#"%'
  for '#' )

SELECT substring(
  subject
  from '% #"[0-9]+th [A-Z][a-z]+ [1-2][0-9][0-9][0-9]#"%'
  for '#' ), emails.*
FROM emails
WHERE subject
  SIMILAR TO
    '% [0-9]+th [A-Z][a-z]+ [1-2][0-9][0-9][0-9]%'
```

Outline

Atoms

`(re)` (where `re` is any regular expression) matches a match for `re`, with the match noted for possible reporting

Atoms

`(re)` (where `re` is any regular expression) matches a match for `re`, with the match noted for possible reporting

`(?:re)` as above, but the match is not noted for reporting (a ”non-capturing” set of parentheses)

Atoms

- `(re)` (where `re` is any regular expression) matches a match for `re`, with the match noted for possible reporting
- `(?:re)` as above, but the match is not noted for reporting (a ”non-capturing” set of parentheses)
- `.` matches any single character (similar to `_` in `LIKE`)

Atoms

`(re)`	(where `re` is any regular expression) matches a match for `re`, with the match noted for possible reporting
`(?:re)`	as above, but the match is not noted for reporting (a ”non-capturing” set of parentheses)
`.`	matches any single character (similar to `_` in `LIKE`)
`[chars]`	a bracket expression, matching any one of the chars

Atoms - cont.

`\k` (where `k` is a non-alphanumeric character) matches that character taken as an ordinary character, e.g., `\\` matches a backslash character

Atoms - cont.

- `\k` (where `k` is a non-alphanumeric character) matches that character taken as an ordinary character, e.g., `\\` matches a backslash character
- `\c` where c is alphanumeric (possibly followed by other characters) is an escape

Atoms - cont.

- `\k` (where `k` is a non-alphanumeric character) matches that character taken as an ordinary character, e.g., `\\` matches a backslash character
- `\c` where c is alphanumeric (possibly followed by other characters) is an escape
- `{` when followed by a character other than a digit, matches the left-brace character `{`; when followed by a digit, it is the beginning of a bound (see below)

Atoms - cont.

- `\k` (where `k` is a non-alphanumeric character) matches that character taken as an ordinary character, e.g., `\\` matches a backslash character
- `\c` where c is alphanumeric (possibly followed by other characters) is an escape
- `{` when followed by a character other than a digit, matches the left-brace character `{`; when followed by a digit, it is the beginning of a bound (see below)
- `x` where x is a single character with no other significance, matches that character

Quantifiers

- `*` a sequence of 0 or more matches of the atom

Quantifiers

- `*` a sequence of 0 or more matches of the atom
- `+` a sequence of 1 or more matches of the atom

Quantifiers

- `*` a sequence of 0 or more matches of the atom
- `+` a sequence of 1 or more matches of the atom
- `?` a sequence of 0 or 1 matches of the atom

Quantifiers

- `*` a sequence of 0 or more matches of the atom
- `+` a sequence of 1 or more matches of the atom
- `?` a sequence of 0 or 1 matches of the atom
- `{m}` a sequence of exactly `m` matches of the atom

Quantifiers

`*`	a sequence of 0 or more matches of the atom
`+`	a sequence of 1 or more matches of the atom
`?`	a sequence of 0 or 1 matches of the atom
`{m}`	a sequence of exactly `m` matches of the atom
`{m,}`	a sequence of `m` or more matches of the atom

Quantifiers

`*`	a sequence of 0 or more matches of the atom
`+`	a sequence of 1 or more matches of the atom
`?`	a sequence of 0 or 1 matches of the atom
`{m}`	a sequence of exactly `m` matches of the atom
`{m,}`	a sequence of `m` or more matches of the atom
`{m,n}`	a sequence of `m` through `n` (inclusive) matches of the atom; `m` cannot exceed `n`

Constraints

- ^ matches at the beginning of the string
- $ matches at the end of the string

POSIX regular expressions in PostgreSQL

- ~ matches regular expression, case sensitive

POSIX regular expressions in PostgreSQL

- $\sim$ matches regular expression, case sensitive
- $\sim*$ matches regular expression, case insensitive

POSIX regular expressions in PostgreSQL

- $\sim$ matches regular expression, case sensitive
- $\sim*$ matches regular expression, case insensitive
- $!\sim$ does not match regular expression, case sensitive

POSIX regular expressions in PostgreSQL

$\sim$	matches regular expression, case sensitive
$\sim*$	matches regular expression, case insensitive
$!\sim$	does not match regular expression, case sensitive
$!\sim*$	does not match regular expression, case insensitive

```
SELECT
  'Never gonna let you down' ~* '^never'

SELECT
  'Never gonna run around and desert you' !~ '^gonna'

SELECT
  'Never gonna make you cry' ~ 'cry$'

SELECT
  'Never gonna say goodbye' !~* 'SAY$'
```

```
SELECT '27-9-2001' ~ '[0-9]{1,2}\-[0-9]{1,2}\-[1-2][0-9]{3}'

SELECT *
FROM emails
WHERE
  subject ~ '[0-9]{1,2}\-[0-9]{1,2}\-[1-2][0-9]{3}'
```

Auxilliary functions

`substring(string from pattern)` where `pattern` is a regular expression extracts the matching substring

Auxilliary functions

`substring(string from pattern)` where `pattern` is a regular expression extracts the matching substring

`regexp_replace(source, pattern, replacement)` replaces the matching substring with `replacement`

Auxilliary functions

`substring(string from pattern)` where `pattern` is a regular expression extracts the matching substring

`regexp_replace(source, pattern, replacement)` replaces the matching substring with `replacement`

`regexp_matches(string, pattern)` returns all matching substrings

Auxilliary functions

`substring(string from pattern)` where `pattern` is a regular expression extracts the matching substring

`regexp_replace(source, pattern, replacement)` replaces the matching substring with `replacement`

`regexp_matches(string, pattern)` returns all matching substrings

`regexp_split_to_table(string, pattern)` uses regular expression match as delimeter and returns splitted table

Auxilliary functions

`substring(string from pattern)` where `pattern` is a regular expression extracts the matching substring

`regexp_replace(source, pattern, replacement)` replaces the matching substring with `replacement`

`regexp_matches(string, pattern)` returns all matching substrings

`regexp_split_to_table(string, pattern)` uses regular expression match as delimeter and returns splitted table

`regexp_split_to_array(string, pattern)` same as above, but returns array

```
SELECT substring( 'Never gonna tell a lie' from 'gonna(.*)lie'
    )

SELECT regexp_replace( '... and hurt you', 'hurt', 'rickroll' )

SELECT regexp_matches( '27 Sep 2001', '([0-9]{1,2})
    ([A-Z][a-z]{2}) ([1-2][0-9]{3})' )

SELECT regexp_split_to_table( 'joza@foi.hr; ivek@foi.hr;
    bara@foi.hr', '\; ' )
```

Outline

Aggregate functions

`avg` finds the average of a numerical attribute

Aggregate functions

`avg` finds the average of a numerical attribute

`sum` finds the sum of a numerical attribute

Aggregate functions

`avg` finds the average of a numerical attribute

`sum` finds the sum of a numerical attribute

`min` find the smalest value of an attribute

Aggregate functions

`avg` finds the average of a numerical attribute

`sum` finds the sum of a numerical attribute

`min` find the smalest value of an attribute

`max` find the highest value of an attribute

Aggregate functions

`avg`	finds the average of a numerical attribute
`sum`	finds the sum of a numerical attribute
`min`	find the smalest value of an attribute
`max`	find the highest value of an attribute
`count`	find the number of values of some attribute

Aggregate functions

`avg`	finds the average of a numerical attribute
`sum`	finds the sum of a numerical attribute
`min`	find the smalest value of an attribute
`max`	find the highest value of an attribute
`count`	find the number of values of some attribute
`variance`	find the variance of some numerical attribute

Aggregate functions

`avg`	finds the average of a numerical attribute
`sum`	finds the sum of a numerical attribute
`min`	find the smalest value of an attribute
`max`	find the highest value of an attribute
`count`	find the number of values of some attribute
`variance`	find the variance of some numerical attribute
`stddev`	find the standard deviation of some numerical attribute

```
SELECT count(*)
FROM emails

SELECT min(date), max(date)
FROM emails
```

GROUP BY and HAVING

Aggregate functions are usually used with a GROUP BY clause to group aggregate values by some given attribute.
HAVING is used for constraints on aggragate expressions.

```
SELECT date, count(*)
FROM emails
GROUP BY date
ORDER BY 2 DESC

SELECT custodianname, count(*)
FROM emails
GROUP BY 1
HAVING count(*) > 100
ORDER BY 2
```

Outline

Project definition

Keyword/pattern definition

Define a set of keywords and data possibly found in email headers (subject, from, to, cc, etc.) which for you as an investigator would be messages of importance. What kind of data is important to an investigative process?

- e.g. keywords: risk, plan, transaction, signature etc.
- e.g. data: email addresses, domains, dates, account numbers, etc.

Keyword extraction

Create queries that will find all the emails sent **to** Enron email addresses (@enron.com) with the keywords and data defined in the previous task.

Project definition - cont.

Mentioned dates

Create queries that will find all emails that mention a date. Extract these dates.

- Think of all possible ways to write a date (e.g. September 27th 2001; 2001-9-27; 27 Sep 2001 etc.)

Communication structure

Try to create a query that will connect emails to their replies (e.g. subject 'Re: Hello' is a reply to subject 'Hello')

- Familiarize your self with the `TEXTCAT/2` function which concatenates strings (to construct a pattern)
- Familiarize your self with the `LIMIT` and `OFFSET` modifiers in `SELECT` (to keep computation reasonable)