Laboratorium Analiza i bazy danych

Łańcuchy znaków i wyrażenia regularne

```
2  ## Wprowadzenie

   W każdym silniku baz danych istnieją mechanizmy do porównywania, dopasowywania i manipulowania ciągam

   Oprócz podstawowej funkcjonalności polegającej na odpowiedzi na pytanie „czy ten ciąg pasuje do tego

   Istnieją trzy osobne podejścia do dopasowywania wzorców zapewniane przez PostgreSQL:
   - LIKE/ILIKE,
   - SIMILAR TO (standard SQL:1999),
   - wyrażenia regularne w stylu POSIX.

   Ten dokument ma na celu przybliżenie podstawowych funkcji działania na ciągach znakowych w PostgreSQL


     File "<ipython-input-2-030f41216694>", line 3
       W każdym silniku baz danych istnieją mechanizmy do porównywania, dopasowywania i manipulowania cia
               ^
   SyntaxError: invalid syntax
```

LIKE/ILIKE

Wyrażenie LIKE zwraca wartość *true*, jeśli ciąg znaków odpowiada dokładnie podanemu wzorcowi. ILIKE natomiast to klauzula umożliwiająca dopasowanie wzorca jednak nie zwraca ona uwagę na wielkość liter. Wyrażeniem przeciwnym jest NOT LIKE lub NOT ILIKE. Składnia tego wyrażenia to:

string (LIKE|ILIKE) wzorzec, string NOT (LIKE|ILIKE) wzorzec

Przy tym typie klauzuli wzorce tworzymy przy użyciu dwóch operatorów:

- _ - zastępuje pojedynczy znak,
- % - zastępuje dowolną długość znaków.

Przykład:

Wyrażenie	Wynik
'abc' LIKE 'abc'	true
'abc' LIKE 'a%'	true
'abc' LIKE '*b*'	true

Wyrażenie	Wynik
'abc' LIKE 'c'	false
'ABC' ILIKE 'abc'	true
'ABC' LIKE 'abc'	false

Klauzule LIKE/ILIKE można zastąpić operatorami:

- ~~ równoważny do LIKE
- ~~* równoważny do ILIKE
- !~~ równoważny do LIKE
- !~~* równoważny do NOT ILIKE

SIMILAR TO

Klauzula SIMILAR TO tak samo jak LIKE/ILIKE zwraca wartość *true* lub *false* w zależności od tego, czy podany wzorzec pasuje do podanego ciągu. Różnica pomiędzy tymi operatorami polega na tym, że SIMILAR TO interpretuje wzorzec za pomocą definicji wyrażenia regularnego w standardzie SQL. Wyrażenia regularne SQL są połączeniem notacji LIKE i zwykłej notacji wyrażeń regularnych (POSIX).

Składnia tego zapytania ma postać:

string SIMILAR TO wzorzec, string NOT SIMILAR TO wzorzec

Oprócz funkcji zapożyczonych z LIKE, SIMILAR TO obsługuje te metaznaki pasujące do wzorca zapożyczone z wyrażeń regularnych POSIX:

- | - oznacza naprzemienność (jedną z dwóch alternatyw).
- * - oznacza powtórzenie poprzedniego elementu zero lub więcej razy.
- + - oznacza powtórzenie poprzedniego elementu jeden lub więcej razy.
- ? - oznacza powtórzenie poprzedniego elementu zero lub jeden raz.
- {m} - oznacza powtórzenie poprzedniego elementu dokładnie m razy.
- {m,} - oznacza powtórzenie poprzedniego elementu m lub więcej razy.
- {m, n} - oznacza powtórzenie poprzedniego elementu co najmniej mi nie więcej niż n razy.

Nawiasów () można używać do grupowania elementów w jeden element logiczny. Wyrażenie w nawiasie [...] określa klasę znaków, podobnie jak w wyrażeniach regularnych POSIX.

Wyrażenie	Wynik
'abc' SIMILAR TO 'abc'	true

Wyrażenie	Wynik
'abc' SIMILAR TO 'a'	false
'abc' SIMILAR TO '%(b|d)%'	true
'abc' SIMILAR TO '(b|c)%'	false

Składnia POSIX

Wyrażenia regularne POSIX zapewniają więcej reguł tworzenia wzorców niż operatory LIKE i SIMILAR TO. Wiele narzędzi uniksowych, takich jak grep, sed lub awk, używa języka dopasowywania wzorców podobnego do tego używanego w PostgreSQL.

Wyrażenie regularne to opis ciągu znaków przy użyciu symboli w celu utworzenia wzorca ciągu, która umożliwia dopasowanie wzorców. Mówi się, że łańcuch pasuje do wyrażenia regularnego, jeśli jest on członkiem zbioru regularnego opisanego przez wyrażenie regularne. Podobnie jak w przypadku LIKE, znaki wzorcowe dokładnie pasują do znaków łańcuchowych, chyba że są znakami specjalnymi w języku wyrażeń regularnych - ale wyrażenia regularne używają innych znaków specjalnych niż LIKE. W przeciwieństwie do wzorców LIKE, wyrażenie regularne może pasować w dowolnym miejscu ciągu, chyba że wyrażenie regularne jest wyraźnie zakotwiczone na początku (^) lub na końcu łańcucha ($).

Używanie tego rodzaju dopasowania ciągu znaków odbywa się przez operatory:

- ~ pasuje do wzorca , wielkość liter ma znaczenie
- ~* pasuje do wzorca , wielkość liter nie ma znaczenie
- !~ nie pasuje do wzorca , wielkość liter ma znaczenie
- !~* nie pasuje do wzorca , wielkość liter nie ma znaczenie

Przykład:

Wyrażenie	Wynik
'abc' ~ 'abc'	true
'abc' ~ '^a'	true
'abc' ~ '(b|d)'	true
'abc' ~ '^(b|c)'	false

Język zapytań regularnych w PostgreSQL który będzie omawiany na zajęciach składa się z:

Wzorzec	Dopasowanie

Wzorzec	Dopasowanie
*	sekwencja 0 lub więcej dopasowań atomu
+	sekwencja 1 lub więcej dopasowań atomu
?	sekwencja 0 lub 1 dopasowań atomu
{m.}	sekwencja dokładnie m dopasowań atomu
{m,}	sekwencja m lub więcej dopasowań atomu
{m, n}	sekwencja od m do n (włącznie) dopasowań atomu; m nie może przekraczać n
^	dopasuj od początku łańcucha znaków
$	dopasuj od końca łańcucha znaków

Formularze używające {...} są znane jako granice. Liczby min w granicach są liczbami całkowitymi dziesiętnymi bez znaku z dopuszczalnymi wartościami od 0 do 255 włącznie.

Pełny opis omawianych funkcjonalności dostępny jest w dokumentacji PostgreSQL

Wybrane funkcje działające na znakach

Poza funkcją dopasowania w PostgreSQL istnieje również szereg funkcji predefiniowanych niepozwalających działanie ciągach znaków. W poniższej tabeli przedstawiono wybrane funkcje:

Funkcja	Opis	Przykład	Wynik
ASCII	Zwraca wartość kodu ASCII znaku lub punktu kodu Unicode znaku UTF8	ASCII ('A')	65
CHR	Konwertuj kod ASCII na znak lub punkt kodu Unicode na znak UTF8	CHR (65)	'A'
CONCAT	Połączenie dwóch lub więcej ciągów w jeden	CONCAT('A', 'B', 'C')	'ABC'

Funkcja	Opis	Przykład	Wynik
CONCAT_WS	Połączenie ciągów znaków z separatorem	CONCAT_WS(',', 'A', 'B', 'C')	'A, B, C'
FORMAT	Łącznie ciągów zgodnie z zadanym wzorcem formatowania	FORMAT('Witaj%s', 'PostgreSQL')	'Witaj PostgreSQL'
INITCAP	Konwertuj łańcuch znaków w styl nagłówka	INITCAP('CZEść wAM')	"Cześć Wam"
LEFT	Zwraca pierwszy n znaku z lewej strony ciągu	LEFT('ABC', 1)	'A'
LENGTH	Zwraca liczbę znaków w ciągu	LENGTH('ABC')	3
LOWER	Konwertuj ciąg na małe litery	LOWER ('czEŚĆ wAM')	'cześć wam'
LPAD	Uzupełnieni z lewej strony ciągu do zadanej długości zadanym ciągiem	LPAD('123', 5, '00')	'00123'
LTRIM	Usuwanie najdłuższego ciąg zawierającego określone znaki z lewej strony ciągu wejściowego	LTRIM ('00123')	'123'
MD5	Zwraca skrót MD5 ciągu szesnastkowego	MD5('ABC')	
POSITION	Zwraca lokalizację pod łańcucha w ciągu	POSTION('B' w 'A B C')	3

Funkcja	Opis	Przykład	Wynik
REGEXP_MATCHES	Dopasuj wyrażenie regularne POSIX do łańcucha i zwraca pasujące podciągi	REGEXP_MATCHES ('ABC', '^(A)(..)$', 'g');	{ABC}
REGEXP_REPLACE	Zamienia podciągi pasujące do wyrażenia regularnego POSIX na nowy podciąg	REGEXP_REPLACE ('John Doe','(.) (.)',’\2, \1’);	'Doe, John'
REPEAT	Powtarza ciąg określoną liczbę razy	REPEAT('*', 5)	'*****'
REPLACE	Zamienia wszystkie wystąpienia w ciągu pod łańcucha z podciągu na zadany REPLACE('ABC', 'B', 'A')	'AAC'	
REVERSE	Odwrócenie ciągu	REVERSE ('ABC') 'CBA'	
RIGHT	Zwraca ostatnie n znaków w ciągu. Kiedy n jest ujemne, zwracaj wszystkie oprócz pierwszego	RIGHT('ABC', 2)	'BC'
RPAD	Uzupełnieni z prawej strony ciągu do zadanej długości zadanym ciągiem	RPAD('ABC', 6, 'xo')	'ABCxox'
RTRIM	Usuwa najdłuższy ciąg zawierający określone znaki z prawej strony ciągu wejściowego	RTRIM 'abcxxzx', 'xyz')	'abc'
SPLIT_PART	Dzieli ciąg na określonym ograniczniku i zwraca n-ty pod łańcuch	SPLIT_PART('2017-12-31’, ’-', 2)	’12’

Funkcja	Opis	Przykład	Wynik
SUBSTRING	Wyodrębnia podciąg z ciągu	SUBSTRING('ABC', 1,1)	'A'
TRIM	Usuwa najdłuższy ciąg zawierający określone znaki z lewej, prawej lub obu ciągów wejściowych	TRIM('ABC')	'ABC'
UPPER	Konwertuje ciąg na wielkie litery	UPPER('CZEść wAM')	'CZEŚĆ WAM'

Zadania:

1. Znajdź wszystkie nazwy krajów rozpoczynających się na P.
2. Znajdź wszystkie nazwy krajów rozpoczynających się P i kończących na s.
3. Znajdź wszystkie tytuły filmów, w których znajdują się cyfry.
4. Znajdź wszystkich pracowników z podwójnym imieniem lub nazwiskiem.
5. Znajdź wszystkie nazwiska aktorów rozpoczynających się od P lub C i mających 5 znaków.
6. Znajdź wszystkie tytuły filmów, w których występują słowa Trip lub Alone.
7. Przeanalizuj zapytania:
   - select first_name from actor where first_name ~ '^ Al[a: z, 1: 9] *'
   - select first_name from actor where first_name ~ * '^ al[a: z, 1: 9] *'

Zadanie implementacyjne

Zaimplementuj wszystkie funkcje w pliku main.py zgodnie z opisem a następnie przetestuj je w notatniku.

1
```
import main

import psycopg2 as pg
import pandas.io.sql as psql
import pandas as pd

connection = pg.connect(host='pgsql-196447.vipserv.org', port=5432, dbname='wbauer_adb', user='wbauer
#main.film_title_case_insensitive(['BeD', 'BLOOD', 'SonS', 'son', 'cry'])
main.film_title_case_insensitive(['Giant', 'Harry', 'Birdcage', 'Iron'])
```

```
C:\Users\Sebastian\Desktop\AiBD\ABiD_gr1_SebastianPilch\lab3_regexp-SebastianPilch\main.py:167: UserWa
  df = pd.read_sql(txt, con=connection)
```

1

	title

	title
0	Birdcage Casper
1	Chocolat Harry
2	Giant Troopers
3	Harry Idaho
4	Hoosiers Birdcage
5	Iron Moon
6	Jaws Harry
7	Kentuckian Giant
8	Louisiana Harry
9	Princess Giant
10	Whisperer Giant

1.Znajdź wszystkie nazwy krajów rozpoczynających się na P.

8
```
pd.read_sql("select c.country from country c where c.country ~~ 'P%'" , con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_26312/4011842811.py:1: UserWarning: pandas only support
  pd.read_sql("select c.country from country c where c.country ~~ 'P%'" , con=connection)
```

8

	country
0	Pakistan
1	Paraguay
2	Peru
3	Philippines
4	Poland
5	Puerto Rico

2.Znajdź wszystkie nazwy krajów rozpoczynających się P i kończących na s.

9
```
pd.read_sql("select c.country from country c where c.country ~~ 'P%s'" , con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_26312/3394920994.py:1: UserWarning: pandas only support
  pd.read_sql("select c.country from country c where c.country ~~ 'P%s'" , con=connection)
```

9

	country
0	Philippines

3.Znajdź wszystkie tytuły filmów, w których znajdują się cyfry.

Cyfry słownie:

```
3 main.film_title_case_insensitive(['one','two','three','four','five','six','seven','eight','nine','zer
```

```
C:\Users\Sebastian\Desktop\AiBD\ABiD_gr1_SebastianPilch\lab3_regexp-SebastianPilch\main.py:167: UserWa
  df = pd.read_sql(txt, con=connection)
```

3

	title
0	Seven Swarm

Cyfry :

```
5 pd.read_sql("select f.title from film f where f.title ~~ '[0-9]'" , con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_14244/3890216700.py:1: UserWarning: pandas only support
  pd.read_sql("select f.title from film f where f.title ~~ '[0-9]'" , con=connection)
```

5

	title

4.Znajdź wszystkich pracowników z podwójnym imieniem lub nazwiskiem.

```
9 pd.read_sql("select s.first_name, s.last_name from staff s where (s.first_name ~~ ' ') or (s.last_nam
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_14244/1386102096.py:1: UserWarning: pandas only support
  pd.read_sql("select s.first_name, s.last_name from staff s where (s.first_name ~~ ' ') or (s.last_na
```

9

	first_name	last_name

5.Znajdź wszystkie nazwiska aktorów rozpoczynających się od P lub C i mających 5 znaków.

```
2 pd.read_sql("select ac.first_name, ac.last_name from actor ac where  ac.last_name ~ '(P|C).{4}$' " ,
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_984/103795398.py:1: UserWarning: pandas only supports S
  pd.read_sql("select ac.first_name, ac.last_name from actor ac where  ac.last_name ~ '(P|C).{4}$' " ,
```

2

	first_name	last_name
0	Ed	Chase
1	Burt	Posey
2	Kenneth	Pesci
3	Sidney	Crowe
4	Jon	Chase
5	Russell	Close

6.Znajdź wszystkie tytuły filmów, w których występują słowa Trip lub Alone.

14
```
main.film_title_case_insensitive(['Trip','Alone'])
```

```
C:\Users\Sebastian\Desktop\AiBD\ABiD_gr1_SebastianPilch\lab3_regexp-SebastianPilch\main.py:167: UserWa
  df = pd.read_sql(txt, con=connection)
```

14

	title
0	Alone Trip
1	Superfly Trip
2	Trip Newton
3	Varsity Trip

7.Przeanalizuj zapytania:

Zapytanie 1:

select first_name from actor where first_name ~ '^ Al[a: z, 1: 9] *'

Odpowiedź:

Zapytanie zwraca imię rozpoczynające się (operator ^) od liter 'Al' oraz zawierające dowolną liczbę znaków (operator *) pojawiających się po Al gdzie pierwszym z nich będzie litera a, kolejne są w zakresie liter od a-z oraz cyfr od 1-9 . Zapyanie jest wpisane w formie like (operator ~) więc istotna jest wielkość liter

Zapytanie 2:

select first_name from actor where first_name ~ * '^ al[a: z, 1: 9] *'

Odpowiedź:

Zapytanie zwraca imię rozpoczynające się (operator ^) od liter 'al' oraz zawierające dowolną liczbę znaków (operator *) pojawiających się po al gdzie pierwszym z nich będzie litera a, kolejne są w zakresie liter od a-z oraz cyfr od 1-9 . Zapyanie jest wpisane w formie ilike (operator ~ *) więc nieistotna jest wielkość liter