ABD ELAZIZ AHMED ABDELMONAM

ID/4211241

G/B2

1)

SQL QUERY

```
1 SELECT*
2 FROM customers
3 WHERE creditlimit >=100000 and customerNumber <200 or country ="USA";
```

▶ Execute | Clear | Beautify | Minify

RESULT

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1
112	Signal Gift Stores	King	Jean	7025551838	8489 Strong St
114	Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	636 St Kilda R
119	La Rochelle Gifts	Labrune	Janine	40.67.8555	67, rue des Ci

2)

SQL QUERY

```
SELECT creditlimit, creditlimit + 2000 as new_credit
FROM customers
order by new_credit DESC
```

Execute | Clear | Beautify | Minify

RESULT

creditlimit	new_credit
227600.00	229600.00
210500.00	212500.00
141300.00	143300.00

3)

SQL QUERY

```
SELECT *
FROM customers
LIMIT 3 ;
```

Execute | Clear | Beautify | Minify

RESULT

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1
103	Atelier graphique	Schmitt	Carine	40.32.2555	54, rue Royale
112	Signal Gift Stores	King	Jean	7025551838	8489 Strong St.
114	Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	636 St Kilda Road

REGEXP

1-

SQL QUERY

```
SELECT contactFirstName
FROM customers
where contactFirstName REGEXP '^je' or '^le'
```

Execute | Clear | Beautify | Minify

RESULT

contactFirstName
Jean
Jeff
Jerry

2-

SQL QUERY

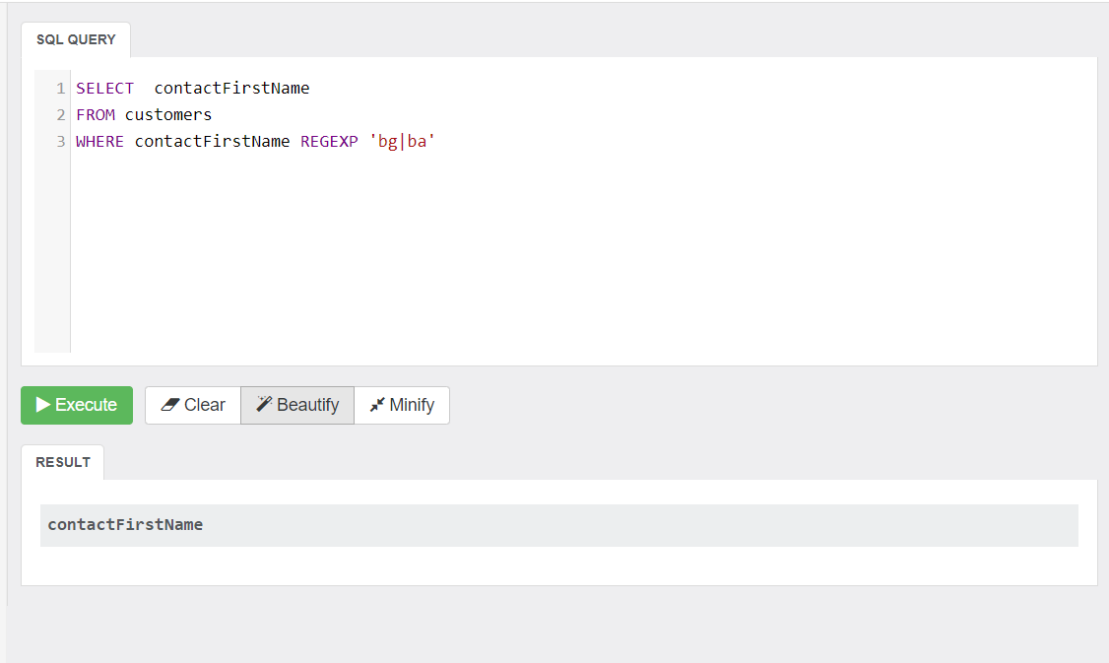
```
SELECT contactFirstName
FROM customers
WHERE contactFirstName REGEXP 'el$' or 'el$'
```

Execute | Clear | Beautify | Minify

RESULT

contactFirstName
Veysel
Michael
Rachel

3-



(REGEXP) abbreviation for "Regular expressions"

It is a method for describing and recognizing texts by describing their components in the form of symbols, and describing the relationships of those symbols in sequence and repetition, in a systematic manner that an algorithm can interpret and apply to a given text to extract the part to which the regular expression applies.

Also, regular expressions are used in computing for word processing, programming language interpreter generators, and for checking software input, and there are applications for them in most programming languages.

Formal definition

Regular expressions consist of constants, which denote sets of strings, and operator symbols, which denote operations over these sets. The following definition is standard, and found as such in most textbooks on formal language theory.[20][21] Given a finite alphabet Σ, the following constants are defined as regular expressions:

- (*empty set*) ∅ denoting the set ∅.
- (empty string) ε denoting the set containing only the "empty" string, which has no characters at all.
- (literal character) `a` in Σ denoting the set containing only the character *a*.

Given regular expressions R and S, the following operations over them are defined to produce regular expressions:

- (concatenation) `(RS)` denotes the set of strings that can be obtained by concatenating a string accepted by R and a string accepted by S (in that order). For example, let R denote {"ab", "c"} and S denote {"d", "ef"}. Then, `(RS)` denotes {"abd", "abef", "cd", "cef"}.
- (alternation) `(R|S)` denotes the set union of sets described by R and S. For example, if R describes {"ab", "c"} and S describes {"ab", "d", "ef"}, expression `(R|S)` describes {"ab", "c", "d", "ef"}.
- (Kleene star) `(R*)` denotes the smallest superset of the set described by *R* that contains ε and is closed under string concatenation. This is the set of all strings that can be made by concatenating any finite number (including zero) of strings from the set described by R. For example, if R denotes {"0", "1"}, `(R*)` denotes the set of all finite binary strings (including the empty string). If R denotes {"ab", "c"}, `(R*)` denotes {ε, "ab", "c", "abab", "abc", "cab", "cc", "ababab", "abcab", ... }.

To avoid parentheses it is assumed that the Kleene star has the highest priority, then concatenation and then alternation. If there is no ambiguity then parentheses may be omitted. For example, `(ab)c` can be written as `abc`, and `a|(b(c*))` can be written as `a|bc*`. Many textbooks use the symbols ∪, +, or ∨ for alternation instead of the vertical bar.

Examples:

- `a|b*` denotes {ε, "a", "b", "bb", "bbb", ...}
- `(a|b)*` denotes the set of all strings with no symbols other than "a" and "b", including the empty string: {ε, "a", "b", "aa", "ab", "ba", "bb", "aaa", ...}
- `ab*(c|ε)` denotes the set of strings starting with "a", then zero or more "b"s and finally optionally a "c": {"a", "ac", "ab", "abc", "abb", "abbc", ...}
- `(0|(1(01*0)*1))*` denotes the set of binary numbers that are multiples of 3: { ε, "0", "00", "11", "000", "011", "110", "0000", "0011", "0110", "1001", "1100", "1111", "00000", ... }