LAS EXPRESIONES REGULARES

Las expresiones regulares (conocidas cómo *regex* o *regexp*) son patrones para validación de datos. En una expresión regular podemos incluir caracteres que deberán aparecer en el dato a validar, como letras, dígitos o signos de puntuación. Podemos indicar que dichos caracteres estén en un rango determinado (por ejemplo, dígitos del 6 al 9, o letras de la C a la R). También podemos indicar que tal o cual carácter aparezca al principio o al final de un dato a validar, o que se encuentre repetido determinado número de veces. Además, podemos agrupar partes de la expresión regular, de tal forma que se evalúe el dato a validar en un orden específico.

La tabla que parece a continuación es un compendio de los distintos patrones que se pueden incluir en una expresión regular.

PATRÓN	USO
`.`	El punto se emplea como un comodín para buscar un carácter cualquiera, en una posición determinada o no, en el dato a validar. Si queremos usarlo para buscar, específicamente, un punto, deberemos escaparlo con la barra invertida (`\.` en lugar de `.`).
`*`	El asterisco indica que el carácter, o secuencia de caracteres, que le precede puede aparecer cualquier número de veces (incluso ninguna, lo que le hace un operador extremadamente ambiguo). Si lo que queremos es buscar, específicamente, un asterisco, deberemos escaparlo con la barra invertida (`\*` en lugar de `*`).
`|`	Alternativa. Es un operado que nos permite buscar un patrón u otro, de forma que se validará un dato cuando incluya cualquiera de los patrones que se buscan. Por ejemplo, dada una expresión regular como `azul|rojo`, dará por válidas las frases `El coche es azul` o `El camión es rojo`, pero no validará `El coche es verde` o `La moto es roja`.
`?`	El signo de interrogación indica que el carácter, o secuencia de caracteres, que le precede puede aparecer una vez, o ninguna, pero no más de una vez en el dato a validar. Por ejemplo, el patrón `o?` validaría `Hla` y también `Hola`, pero no `Hoola`. Si queremos buscar en el dato a validar específicamente el carácter `?` deberemos escaparlo con la barra invertida (`\?` en lugar de `?`).
`+`	Este operado indica que el carácter, o secuencia de caracteres, que le precede debe aparecer, al menos, una vez. Por ejemplo, el patrón `i+` validaría `Junio` o `Noviembre`, pero no `Enero`. Si queremos

	buscar, específicamente, el carácter `+` deberemos escaparlo con la barra invertida (`\+` en lugar de `+`).
`()`	Los paréntesis se usan para agrupar caracteres, o secuencias de caracteres, o para establecer precedencias. Por ejemplo, el patrón `(f|g)ruta` validaría un dato que contenga `fruta` o `gruta`. Si queremos buscar específicamente un carácter `(` o `)` deberemos escaparlo con la barra invertida (`\(` en lugar de `(`, o `\)` en lugar de `)`). Si queremos buscar una secuencia formada por los dos paréntesis (abierto y cerrado), uno a continuación del otro, deberemos escapar cada uno de ellos (`\(\)` en lugar de `()`).
`\`	La barra invertida es un operador extremadamente útil en la construcción de expresiones regulares. Permite escapar cualquier operador (ya hemos visto algunos casos en esta tabla) de forma que deje de ser un operador como tal, y represente un carácter específico. Además, permite crear determinadas secuencias para buscar. Por ejemplo, si creamos un patrón que incluya `\t` no buscará la letra `t`, sino una tabulación. Uno de los usos, en este sentido, más útiles, es especificar un carácter mediante su código Unicode. Por ejemplo, el patrón `\u03A9` buscará el carácter `Ω`. Es útil cuando hay que buscar caracteres que no están disponibles en el teclado que empleamos. Se puede encontrar una tabla Unicode en este enlace. Si queremos buscar específicamente una barra invertida, deberemos escapar dicho carácter con otra barra invertida (`\\` en lugar de `\`).
`[rango]`	Los corchetes permiten definir un rango de caracteres a buscar. Por ejemplo, el patrón `[a-f]` buscará cualquier letra **minúscula** que esté entre la `a` y la `f`; el patrón `[A-F]` buscará cualquier letra **mayúscula** que esté entre la `A` y la `F`; el patrón `([A-F]|[a-f])` buscará cualquier letra que esté entre la `A` y la `F`, y también entre la `a` y la `f`. Esto puede expresarse de forma simplificada cómo `[A-Fa-f]`. Si queremos buscar específicamente un corchete, deberemos escaparlo (`\[` en lugar de `[`, o `\]` en lugar de `]`).
`{}`	Este operador contiene un valor numérico, o rango de valores numéricos, que indican el número de veces que el carácter o secuencia precedente debe aparecer en un dato, para que sea validado. Por ejemplo, el patrón `a{3}` validará un dato cuando, dentro del mismo,

	aparezca la secuencia `aaa`. El patrón `[1-5]{2,4}` validará un dato si encuentra una secuencia de entre dos y cuatro dígitos, siempre que estos estén comprendidos entre el 1 y el 5. Es decir, validará `123`, pero no `567` (ya que de los dígitos comprendidos entre el 1 y el 5 solo aparece uno, y buscamos entre dos y cuatro). Si queremos encontrar específicamente los caracteres `{` o `}` deberemos escaparlos (`\{` en lugar de `{`, o `\}` en lugar de `}`).
`$`	Este operador establece que el carácter, o rango de caracteres, que le precede es el último de la línea. Por ejemplo, el patrón `A$` validará las cadenas que terminen con A mayúscula. Es especialmente útil cuando se están validando datos multilínea, por ejemplo, ya que el patrón `((\r|\n)\.)$` buscará los finales de cada línea. Si lo que queremos es buscar este carácter específicamente, deberemos escaparlo con una barra invertida (`\$` en lugar de `$`).
`^`	Al contrario que el anterior, este operador busca coincidencia con el principio de la línea. En este caso, de modo excepcional, el carácter o secuencia que se busca se establece a la derecha del operador, en lugar de precediéndole, cómo ocurre con otros operadores. Por ejemplo, el patrón `^[a-z]` validará un dato que empiece por una letra minúscula. En combinación con los cochetes, su uso cambia. En este caso se buscan NO coincidencias con el carácter o rango indicado. Por ejemplo, el patrón `[^\d]` buscará coincidencias con caracteres que no sean dígitos, pero no necesariamente al principio del dato a validar. Si debiéramos buscar esta coincidencia en la posición inicial, el patrón sería `^[^\d]`. Si lo que queremos es buscar este carácter específicamente, deberemos escaparlo con una barra invertida (`\^` en lugar de `^`).
`!`	Este operador actúa como negación del carácter, secuencia o rango que le sucede. Por ejemplo, el patrón `!\d` buscará caracteres que no sean dígitos. Si lo que queremos es buscar este carácter específicamente, deberemos escaparlo con una barra invertida (`\!` en lugar de `!`).

EXPRESIONES REGULARES EN SQL

Como es sabido, en una sentencia SELECT se puede utilizar la instrucción LIKE para buscar patrones y no coincidencias especificas en un filtro WHERE. Sin embargo, LIKE resulta limitado al tratar de buscar patrones que dependan de la aparición de determinados caracteres o en un orden específico. En estos casos, es posible utilizar

directamente expresiones regulares en un filtro WHERE indicando REGEXP como operador, como puedes ver a continuación:

Encuentra todos los registros que en el campo last_name tengan un dato que comience con 'st':

```
SELECT last_name FROM person WHERE name REGEXP '^st';
```

Encuentra todos los registros que en el campo name tengan un dato que termine con 'on':

```
SELECT name FROM person WHERE name REGEXP 'on$';
```

Aunque los dos ejemplos anteriores podrían perfectamente solucionarse solo utilizando LIKE lo importante es entender la posibilidad de utilizar expresiones regulares. Un ejemplo más complejo:

Encuentra todos los registros que en el campo last_name tengan un dato que comience con una vocal y al final tenga “on”

```
SELECT last_name FROM person WHERE name REGEXP '^[aeiou]|on$';
```

Referencias:
https://eldesvandejose.com/2017/08/18/expresiones-regulares-general/

http://www.w3big.com/es/mysql/mysql-regexp.html