EI1020/MT1020. Databases Academic Year 2020-21

Session 5. SQL functions and operators

Document version: 2021-02-18

1 Objectives

The objectives to be covered in this practice are the following:

- To be able to use the functions and operators introduced in this session in the `SELECT`, `WHERE`, and `ORDER BY` clauses in order to design SQL queries.
- To be aware that each DBMS provides many functions and operators for each datatype, whose descriptions are always included in both the reference manual and the interactive help.

2 Some advice for this session

As usual, first create a working directory for this session within your directory `EI1020`. Jump into it and save each SQL query in a different file with the `.sql` extension. Use a naming system that can be employed in all sessions. For example, the file containing the first exercise of today might be named `s05_ej01.sql`.

Before writing each query, you can check in the course website the results your query should obtain, so that you can find out if you have understood the problem statement correctly.

3 In the previous session ...

You practiced some clauses of the SQL `SELECT` sentence. You checked that expressions may also show up in the `SELECT`, `WHERE`, and `ORDER BY` clauses. You also checked that columns and expressions in the `SELECT` clause can be labeled by using the `AS` keyword and that the output of the SQL queries can be ordered by using the `ORDER BY` clause. You also practiced the use of the `DISTINCT` clause.

The format of the `SELECT` sentence is the following:

```
SELECT [DISTINCT] { * | column [, column]}
FROM   table
[WHERE search_condition]
[ORDER BY column [ASC|DESC] [,column [ASC|DESC] ]
[FETCH FIRST n ROWS ONLY ];
```

4 In this session ...

Next, we will introduce some functions and operators that can be employed in the `SELECT`, `WHERE`, and `ORDER BY` clauses to build advanced expressions. **Although many of DBMS follow the SQL standard, each DBMS usually usually provides additional operators and functions** to increase the power of the language. It is strongly recommended that **you consult the manuals** to find out the features of each particular system.

4.1 Datatypes

The datatypes available in PostgreSQL can be consulted in the user manual (see `http://postgresql.org`). This section describes the most-common standard datatypes and their corresponding functions and operators, usually employed in real-world applications as well as in our practice database and exercises.

- `VARCHAR( n )`: String of up to `n` characters.
- `NUMERIC( n, m )`: It stores a number with up to `n` digits. The `n` argument is the total count of significant digits in the whole number (number of digits to both sides of the decimal point). The `m` is the count of decimal digits in the fractional part (to the right of the decimal point). This datatype can store numbers with a very large number of digits and perform accurate calculations. It is especially recommended for storing monetary amounts and other quantities where accuracy is required. However, arithmetic is slower than that of the integer type.
- `INTEGER`: Integer values (4 bytes).
- `DATE`: This datatype allows to store a date.
- `TIMESTAMP`: This datatype allows to store both time and date.
- `BOOLEAN`: Though this datatype has not been used in our practice database, it is very interesting. The two values can be expressed with the `TRUE` and `FALSE` keywords. When printing these values, the `'t'` and `'f'` characters are shown for true and false, respectively. Note that any of the following values is considered true in PostgreSQL: `TRUE`, `'t'`, `'true'`, `'y'`, `'yes'`, and `'1'`. Analogously, any of the following values is considered false in PostgreSQL: `FALSE`, `'f'`, `'false'`, `'n'`, `'no'`, and `'0'`. Anyway, **the use of the `TRUE` and `FALSE` keywords is recommended** since they are part of the SQL standard.

4.1.1 Nulls

Keep in mind that the null is not a value, but it means no value. The null is represented by the constant `NULL`. When printing it, nothing is displayed. In the last sessions, it was explained how to convert null values by using the `COALESCE` function.

⋆ **Exercise 1**: Complete the following sentence so that it displays the description of the articles whose minimum stocks are 0 or unknown. Use the `COALESCE` function.

```
1  SELECT ..............
2  FROM   articulos
3  WHERE  ..............
```

4.2 Logical operators

These operators are the following: `AND`, `OR`, and `NOT`. Since null values can show up, most DBMS including PostgreSQL employs a three-valued logic. The next table shows this logic, where `T` and `F` stand for true and false, respectively:

a	b	a `AND` b	a `OR` b	`NOT` b
T	T	T	T	F
T	F	F	T	T
T	null	null	T	null
F	F	F	F	
F	null	F	null	
null	null	null	null	

4.3 Comparison operators

`<`	Less than.
`>`	Greater than.
`<=`	Less or equal than.
`>=`	Greater or equal than.
`=`	Equal to.
`<>  !=`	Not equal to.
`a BETWEEN x AND y`	Equivalent to: `x <= a AND a <= y`
`a NOT BETWEEN x AND y`	Equivalent to: `a < x OR a > y`
`a IS NULL`	Return True if `a` is null.
`a IS NOT NULL`	Return True if `a` is not null.
`a IN (v1, v2, ...)`	Equivalent to: `(a = v1) OR (a = v2) OR ...`

4.4 Mathematical operators

`+`	Addition.
`-`	Subtraction.
`*`	Multiplication.
`/`	Division (if both operands are integer, its result is truncated).
`%`	Remainder of the integer division.
`^`	Power. Example: `3^2 = 9`.
`|/`	Square root. Example: `|/ 25 = 5`.
`||/`	Cube root. Example: `||/ 27 = 3`.
`!`	Factorial. Example: `5 !  = 120`.
`!!`	Factorial as prefix operator. Example: `!!  5 = 120`.
`@`	Absolute value.

Operators that perform operations on binary datatypes are not included in the previous list.

4.5 Mathematical functions

`ABS(x)`	Returns the absolute value of `x`.
`SIGN(x)`	Returns the sign of `x` (-1, 0, 1).
`MOD(x,y)`	Returns the remainder of `x` divided by `y`.
`SQRT(x)`	Returns the square root of `x`.
`CBRT(x)`	Returns the cube root of `x`.
`CEIL(x)`	Returns smallest integer greater than or equal to `x`.
`FLOOR(x)`	Returns largest integer less than or equal to `x`.
`ROUND(x)`	Returns `x` rounded to the closest integer.
`ROUND(x,n)`	Returns `x` rounded to `n` decimal places, if `n` is positive. If `n` is negative, returns the closest integer to `x`, which is multiple to 10^n
`TRUNC(x)`	Returns `x` truncated to 0 decimal places, that is, it removes the decimal part of `x`.
`TRUNC(x,n)`	Returns `x` truncated to `n` decimal places, if `n` is positive. If `n` is negative, returns the closest integer below `x` which is multiple to 10^n.

In addition to these, PostgreSQL includes many other functions for processing logarithms, conversions of angles, trigonometry, etc.

4.6 String functions and operators

In SQL, strings are delimited by single quotes: `'string'`. Recall that PostgreSQL allows to use the dollar quoting for complex and long strings.

`string || string` : Concatenate both strings.

`string LIKE expr` : Returns `TRUE` if the `string` matches the pattern `expr`. Different wildcard characters can be used in `expr`: _ for one character and % for zero or more characters.

- Example 1:

```
1  SELECT * FROM clientes WHERE nombre LIKE 'A%';
```

  Shows customers whose name starts with `A`.

- Example 2:

```
1  SELECT * FROM clientes WHERE nombre LIKE 'A_';
```

  Shows customers whose name only contains two characters, and the first one is `A`.

- Example 3:

```
1  SELECT * FROM pueblos WHERE nombre LIKE '_A %';
```

  Shows villages such that the second character of its name is `A` and the third character is blank.

`LENGTH(string)` : Returns the number of characters the string contains.

`CHAR_LENGTH(string)` : It is the standard SQL function equivalent to `LENGTH`.

`POSITION(substring IN string)` : Returns an integer value representing the starting position of `substring` within `string`.

`SUBSTR(string, n [, len])` : Returns the substring of `string` beginning at the position `n`. If `len` is not omitted, it sets the maximum size of the substring; if `len` is omitted, the rest of the substring is returned.

SUBSTRING(string FROM n [FOR len]) : It is the standard SQL function equivalent to SUBSTR.

LOWER(string) : Returns string in lower case.

UPPER(string) : Returns string in upper case.

INITCAP(string) : Returns string with the first character of each word in upper case and the remaining ones in lower case.

The previous three functions are very important. Since the data is usually stored in the database in any form or combination of uppercase and lowercase letters, they must be used when searching strings.

- Example 1:

```
1  SELECT * FROM provincias WHERE nombre LIKE 'A%';
```

Shows the provinces whose name starts with A (A capital). If a name is written in lower case, it will not be shown.

- Example 2:

```
1  SELECT * FROM provincias WHERE UPPER(nombre) LIKE 'A\%';
```

Shows the provinces whose name begins with A (both lowercase and uppercase).

BTRIM(string) : Returns the string without any leading space characers and trailing space characters. The B letter stands for both (ends).

LTRIM(string) : Returns the string without any leading space characters.

RTRIM(string) : Returns the string without any trailing space characters.

BTRIM(string, list) : Returns the string removing any characters included in list from both ends.

Example: `SELECT BTRIM('-++-+Hello+-face-+queue++-+-', '+-');`

LTRIM(string, list) : Returns the string without any leading characters included in list.

RTRIM(string, list) : Returns the string without any trailing characters included in list.

TRIM(side list FROM string) : It is the SQL standard function equivalent to the three previous ones. If side is BOTH, it is equivalent to BTRIM; If side is LEADING, it is equivalent to LTRIM; If side is TRAILING, it is equivalent to RTRIM.

Example: `SELECT TRIM(BOTH '+-' FROM '-++-+Hello+-face-+queue++-+-');`

CHR(n) : Returns the character whose ASCII code is n.

LPAD(string, n, [, c]) : Returns string padded on the left with the c character to complete the length specified by n. If c is not specified, the string is filled in with spaces. If the length of the string is more than n characters, the result is truncated.

RPAD(string, n, [, c]) : Returns string padded on the right with the c character to complete the length specified by n. If c is not specified, the string is filled in with spaces. If the length of the string is more than n characters, the result is truncated.

⋆ **Exercise 2**: How many rows does the following query return? Do not execute the sentence and explain your answer.

```
SELECT * FROM provincias WHERE UPPER(nombre) LIKE 'a%';
```

Solution:

4.7 Data operators and functions

In PostgreSQL the format of the dates can be chosen by using the `SET DATESTYLE` command. Specifically, to display dates in European style (day before the month before year with the slash character (/) as separator) use the following command:

```
SET DATESTYLE TO EUROPEAN, SQL;
```

Like the rest of datatypes, the `DATE` datatype has its own functions and operators.

In SQL, the + and - operators can be employed to add or subtract days to a date. For example, the `CURRENT_DATE + 7` expression adds 7 days to the current date.

Some of the functions are shown next. The most common functions are the following:

`CURRENT_DATE` : Standard function that returns the current date (the result is of datatype `DATE`).

`CURRENT_TIME` : Standard function that returns the current time (the result is of datatype `TIME`).

`CURRENT_TIMESTAMP` : Standard function that returns the current date and time (the result is of datatype `TIMESTAMP`).

`EXTRACT (field FROM data)` : Standard function that returns the part of `data` indicated by the `field` argument. The `data` argument can be of the following datatypes: `DATE`, `TIME`, and `TIMESTAMP`. The result returned is of datatype `DOUBLE PRECISION`.

The following values can be specified for the `field` argument:

`day`	Day of the month (1:31).
`dow`	Day of the week (0:6, starting on Sunday).
`doy`	Day of the year (1:366).
`week`	Week of the year (1:53).
`month`	Month of the year (1:12).
`quarter`	Quarter (1:4).
`year`	Year.
`hour`	Hour.
`minute`	Minute.
`second`	Second.

Next are some examples of these functions:

```
SELECT CURRENT_TIMESTAMP;
SELECT 365 - EXTRACT(doy FROM CURRENT_DATE) AS dias_faltan;
SELECT EXTRACT(year FROM CURRENT_DATE) AS anyo_actual;
SELECT EXTRACT(week FROM TO_DATE('16/03/2012','dd/mm/yyyy'));
```

SQL includes some functions to convert between different datatypes. All of them have the same structure: They receive a piece of `data` of a datatype and it is converted to another datatype according to `format`.

`TO_CHAR(data, format)` : Converts the `data` of any datatype to string.

`TO_DATE(text, format)` : Converts the `text` to date.

`TO_NUMBER(text, format)` : Converts the `text` to number.

Below are some of allowed formats:

Conversions	Date / Time
HH	Time of day (1-12).
HH12	Time of day (1-12).
HH24	Time of day (0-23).
MI	Minute (00-59).
SS	Second (00-59).
YYYY	Year.
YYY	Last three digits of the year.
YY	Last two digits of the year.
Y	Last digit of the year.
MONTH	Name of the month.
MON	Abbreviated month name.
DAY	Name of the day.
DY	Shortcut of the name of the day.
DDD	Number of days within the year (001-366).
DD	Number of days within the month (01-31).
D	Number of days within the week (0-6 starting on Sunday).
WW	Week number of the year (1-53).
W	Number of the week in the month (1-5).
Q	Number of the quarter (1-4).

Conversions	Number
9	digit number.
S	negative value with minus sign.
.	decimal point.
,	thousands separator.

When the format includes a string, the use of upper and lower case in the pattern changes the way the output is displayed. For example, `MONTH` shows the name of the month in upper case, `Month` shows the name with the first letter capitalized, and `month` shows the name in lower case. Any character in the format that does not match any pattern is copied to the output. Below are some examples:

```
1  SELECT TO_CHAR(CURRENT_TIMESTAMP, 'HH12 horas MI m. SS seg.');
2  SELECT TO_NUMBER('-12,454.8','S999,999.9');
```

4.8 CASE operator

Every procedural programming language usually has a conditional statement: If a condition is true, an action is performed; otherwise, another action is performed. Though SQL is not a procedural language, it includes conditional control by using the `CASE` operator.

The next example shows the utility of and how to use this operator.

```
1  SELECT codart, precio,
2         CASE WHEN stock > 500 THEN precio * 0.8
3              WHEN stock between 200 and 500 THEN precio * 0.9
```

```
4               ELSE precio
5           END AS precio_con_descuento
6   FROM    articulos;
```

For each article, the previous sentece shows its code, its price, and the discounted price, obtained as a function of its stock. If the stock is over 500 units, the discount is 20 % (the price is multiplied by 0.8). If the stock is between 200 and 500 units, the discount is 10 % (the price is multiplied by 0.9). Otherwise, no discount is applied. The column, where the discounted price is located, is labeled as `precio_con_descuento`. The operator `CASE` is ended by `END`. It can have as many operators `WHEN ...THEN` as required, but only one `ELSE` keyword.

4.9 COALESCE and NULLIF functions

The `COALESCE` function can have several arguments (two or more). It returns the first non-null argument.

```
1   COALESCE(value [, ...])
```

⋆ **Exercise 3**: Write what the following statement does.

```
1   SELECT COALESCE(stock, stock_min, -1) FROM articulos;
```

Solution:

The `NULLIF` function returns null if `value1` and `value2` are equal; otherwise, it returns `value1`. This function is the inverse of `COALESCE`.

```
1   NULLIF( value1, value2 )
```

The following statement does not show the description if it is `'(vacio)'`.

```
1   SELECT NULLIF( descripcion, '(vacio)' ) FROM articulos;
```

The next sentence shows articles whose stocks are at or below its minimum. If it is below, it shows how many units are needed to reach it.

```
1   SELECT codart, descrip, NULLIF( stock_min - stock, 0 ) as falta
2   FROM   articulos
3   WHERE  stock <= stock_min;
```

5 Examples

1. Problem: Show a list of the codes and dates of last-year invoices belonging to customers whose code is between 50 and 80. The result should be in descending order of the date.

   After consulting the description of the `FACTURAS` table, you can see that the `fecha` column is of datatype `DATE`. To obtain the invoices of the past year, both the year from the current date (`CURRENT_DATE`) and the year from the invoice date must be extracted.

```
SELECT codfac, fecha
FROM   facturas
WHERE  EXTRACT( year FROM CURRENT_DATE ) =
       EXTRACT( year FROM fecha ) + 1
AND    codcli BETWEEN 50 AND 80
ORDER BY fecha DESC;
```

As previously told, the format of the dates can be changed by executing the `SET DATESTYLE TO EUROPEAN, SQL;` statement.

2. Problem: Show the current date in words.

```
SELECT TO_CHAR(CURRENT_DATE, 'Day, dd of month of yyyy');
```

If you execute this statement, you can see the gap between some words. Notice the following result:

```
Friday    , 10 of march      of 2021
```

The reason is that the `EXTRACT` function applied to the day of the week and to the month always leaves enough space to display the largest word. If the unneeded blanks are not to be obtained, the `RTRIM` function must be used. Next see the new sentence and its result.

```
SELECT RTRIM( TO_CHAR(CURRENT_DATE, 'Day' ) ) ||
       RTRIM( TO_CHAR(CURRENT_DATE, ', dd of month' ) ) ||
       TO_CHAR( CURRENT_DATE, ' of yyyy' );

Friday, 10 of march of 2021
```

6 Session exercises

The exercises in this section must be done individually. You can check if your SQL sentences obtain the expected result by comparing your results with those in this course's website.

1. Show a list of the codes and dates of the invoices done during the month of March of any of last three years (the past year and the previous two). Order the results by descending date.
2. Show a list of the codes and dates of the invoices done in the first 50 days of the past year and belonging to customers whose codes are between 100 and 250. The list should be ordered by the date of invoice.
3. Show a list of codes and dates of the invoices done in the last quarter of the past year and belonging to customers whose codes are between 50 and 100. The list must also show the client code and be ordered by the date of invoice.
4. Show a list of codes and dates of the invoices that belong to customers with codes are 90 or 99, in which no discount has been applied or do not have VAT. Show also the discount and the VAT. Keep in mind that null in the discount and VAT columns can also be interpreted as zero. The list should be ordered by the date of invoice.
5. What names of villages in the Valencian Community end with the syllable 'bo'? The codes of the provinces of the Valencian Community are '03' (Alicante), '12' (Castellón), and '46' (Valencia). The result should be ordered by the town's name and must also show the code of the province.

6. Write in English what this SQL query does.

```
SELECT codart, precio,
       CASE
          WHEN precio > 150 THEN ROUND( precio * 0.90, 2 )
          ELSE  ROUND( precio * 0.85, 2 )
       END AS promocion,
       CASE
          WHEN precio > 150 THEN '10%'
          ELSE '15%'
       END AS dto
FROM   articulos
WHERE  ( precio > 150 AND stock * precio >= 300 )
OR     ( precio <= 150 AND stock * precio >= 150 )
ORDER  BY codart;
```

7. Write in English what this SQL query does.

```
SELECT *
FROM   articulos
WHERE  UPPER( codart ) = LOWER( codart )
ORDER  BY descrip;
```

8. Show a list with the names and the provinces of all the salesmen from the Valencian Community. You can find out the province by using the first two caracters of the postal code.

   If the postal code column did not exist, this information should be extracted from multiple tables.

9. Show the first 10 names of clients with codes lower than 50. The names of the clients in the database employ the following format: "apellidos, nombre". The output name must employ the following format: "nombre apellidos". The result must be sorted by the client names.

10. In which months of the last year did the customers with codes 45, 54, 87 and 102 buy something? Remember what you have to keep in mind:
    Which table contains the data of the query? .....
    Are there any restrictions? .....
    The table has one row for each .....
    and the result must show one row for each .....
    Should you use the `DISTINCT` clause? .....
    Is it necessary to order the result? .....

11. Show the names, addresses, and zip codes of those customers with a zip code beginning with 12. Plus, show a new column labeled `tipo` with the `preferente` word for clients whose village code ends in 9, and the `ordinario` word for clientes whose village code ends in 6.

    Customers have to be displayed sorted by name. The names and the addresses have to be displayed with the first letter of each word capitalised.