# Resumen del capítulo: Funciones avanzadas de SQL para analistas

## Agrupación de datos

Cuando los datos se van a dividir en grupos por valores de campo, se utiliza el comando **GROUP BY**:

```
 SELECT
  field_1,
  field_2,
  ...,
  field_n,
  AGGREGATE_FUNCTION(field) AS here_you_are
FROM
  table_name
WHERE -- si es necesario
  condition
GROUP BY
  field_1,
  field_2,
  ...,
  field_n
```

Una vez que sabes por qué campos agruparás, asegúrate de que todos esos campos estén enumerados tanto en el bloque SELECT como en el bloque GROUP BY. La función de agregación en sí misma no debe incluirse en el bloque GROUP BY; de lo contrario, la consulta no cumplirá. GROUP BY de SQL funciona de manera muy similar al método `groupby()` en pandas.

GROUP BY se puede usar con cualquier función de agregación: COUNT, AVG, SUM, MAX, MIN. Puedes llamar a varias funciones a la vez.

## Ordenar datos

Los resultados del análisis normalmente se presentan en un cierto orden. Para ordenar los datos por un campo, utiliza el comando **ORDER BY**.

```
SELECT
  field_1,
  field_2,
  ...,
  field_n,
  AGGREGATE_FUNCTION(field) AS here_you_are
FROM
  table_name
WHERE -- si es necesario
  condition
GROUP BY
  field_1,
  field_2,
  ...,
  field_n,
ORDER BY -- si es necesario. Enumera solo los campos
--por los que tenemos que ordenar los datos de la tabla
  field_1,
  field_2,
  ...,
  field_n,
  here_you_are;
```

A diferencia de GROUP BY, con ORDER BY solo los campos por los que queremos ordenar los datos deben aparecer en el bloque de comandos.

Se pueden utilizar dos modificadores con el comando ORDER BY para ordenar los datos en columnas:

- **ASC** (el valor predeterminado) ordena los datos en orden ascendente.
- **DESC** ordena los datos en orden descendente.

Los modificadores ORDER BY se escriben justo después del campo por el que se ordenan los datos:

```
ORDER BY
  field_name DESC
-- ordenar los datos en orden descendente

ORDER BY
  field_name ASC;
-- ordenar los datos en orden ascendente
```

El comando **LIMIT** establece un límite para el número de filas en el resultado. Siempre viene al final de una instrucción, seguido del número de filas en las que se establecerá el límite *(n)*:

```
SELECT
  field_1,
  field_2,
  ...,
  field_n,
  AGGREGATE_FUNCTION(field) AS here_you_are
FROM
  table_name
WHERE -- si es necesario
  condition
GROUP BY
  field_1,
  field_2,
  ...,
  field_n,
ORDER BY -- si es necesario. Enumera solo los campos
--por los que tenemos que ordenar los datos de la tabla
  field_1,
  field_2,
  ...,
  field_n,
  here_you_are
LIMIT -- si es necesario
  n;
-- n: el número máximo de filas a devolver
```

## Procesamiento de datos dentro de una agrupación

La construcción **WHERE** se usa para ordenar datos por filas. Sus parámetros son, de hecho, filas de tabla. Cuando necesitamos ordenar datos por resultados de funciones de agregación, usamos la construcción **HAVING**, que tiene mucho en común con **WHERE**:

```
 SELECT
  field_1,
  field_2,
  ...,
  field_n,
  AGGREGATE_FUNCTION(field) AS here_you_are
```

```
FROM
  TABLE
WHERE -- si es necesario
  condition
GROUP BY
  field_1,
  field_2,
  ...,
  field_n
HAVING
  AGGREGATE_FUNCTION(field_for_grouping) > n
ORDER BY -- si es necesario. Enumera solo los campos
--por los que tenemos que ordenar los datos
  field_1,
  field_2,
  ...,
  field_n,
  here_you_are
LIMIT -- si es necesario
  n;
```

La selección resultante incluirá solo aquellas filas para las que la función de agregación produzca resultados que cumplan la condición indicada en los bloques HAVING y WHERE.

HAVING y WHERE tienen mucho en común. Entonces ¿por qué no podemos pasar todas nuestras condiciones a una de ellas? El caso es que el comando WHERE se compila antes de realizar las operaciones de agrupación y aritméticas. Es por eso que es imposible establecer parámetros de clasificación para los resultados de una función de agregación con WHERE. De allí viene la necesidad de usar HAVING.

Presta atención especial al orden en que aparecen los comandos:

1) GROUP BY

2) HAVING

3) ORDER BY

Este orden es **obligatorio**. De lo contrario, el código no funcionará.

## Operadores y funciones para trabajar con fechas

Tenemos dos funciones principales para trabajar con valores de fecha y hora: **EXTRACT** y **DATE_TRUNC**. Ambas funciones se llaman en el bloque SELECT.

Así es como se ve la función EXTRACT:

```
SELECT
  EXTRACT(date_fragment FROM column_name) AS new_column_with_date
FROM
  Table_with_all_dates;
```

EXTRACT, como es lógico, extrae la información que necesitas de la marca temporal. Puedes recuperar:

- `century` — siglo
- `day` — día
- `doy` — día del año, del 1 al 365/366
- `isodow` — (día de la semana según la ISO 8601, el formato internacional de fecha y hora); el lunes es 1, el domingo es 7
- `hour` — hora
- `milliseconds` — milisegundos
- `minute` — minuto
- `second` — segundo
- `month` — mes
- `quarter` — trimestre
- `week` — semana del año
- `year` — año

**DATE_TRUNC** trunca la fecha cuando solo necesitas un cierto nivel de precisión. (Por ejemplo, si necesitas saber qué día se realizó un pedido, pero la hora no importa, puedes usar DATE_TRUNC con el argumento "day"). A diferencia de EXTRACT, la fecha truncada resultante se proporciona como una cadena. La columna de la que se toma la fecha completa viene después de una coma:

```
SELECT
  DATE_TRUNC('date_fragment_to_be_truncated_to', column_name) AS new_column_with_date
FROM
  Table_with_all_dates;
```

Puedes usar los siguientes argumentos con la función DATE_TRUNC:

`'microseconds'`
`'milliseconds'`
`'second'`
`'minute'`
`'hour'`
`'day'`
`'week'`
`'month'`
`'quarter'`
`'year'`
`'decade'`
`'century'`

## Subconsultas

Una **subconsulta**, o **consulta interna**, es una consulta dentro de una consulta. Recupera información que luego se utilizará en **la consulta externa**.

Se puede utilizar las subconsultas en varias ubicaciones dentro de una consulta. Si una subconsulta está dentro del bloque FROM, SELECT seleccionará datos de la tabla que genera la subconsulta. El nombre de la tabla se indica dentro de la consulta interna y la consulta externa se refiere a las columnas de la tabla. Las subconsultas siempre se escriben entre paréntesis:

```
SELECT
SUBQUERY_1.column_name,
SUBQUERY_1.column_name_2
FROM -- para que el código sea legible, coloca subconsultas en nuevas líneas
  -- sangra las subconsultas
  (SELECT
    column_name,
    column_name_2
```

```
  FROM
    table_name
  WHERE
    column_name = value) AS SUBQUERY_1;
-- recuerda nombrar tu subconsulta en el bloque FROM
```

Es posible que necesites subconsultas en varios lugares dentro de tu consulta. Vamos a añadir una en el bloque WHERE. La consulta principal comparará los resultados de la subconsulta con los valores de la tabla en el bloque externo FROM. Cuando haya una coincidencia, se seleccionarán los datos:

```
SELECT
  column_name,
  column_name_1
FROM
  table_name
WHERE
  column_name =
    (SELECT
      column_1
    FROM
      table_name_2
    WHERE
      column_1  = value);
```

Ahora agreguemos la construcción IN a nuestra muestra y recopilemos datos de varias columnas:

```
SELECT
  column_name,
  column_name_1
FROM
  table_name
WHERE
  column_name IN
      (SELECT
        column_1
      FROM
        table_name_2
      WHERE
        column_1 = value_1 OR column_1 = value_2);
```

## Funciones de ventana

En SQL, una ventana es una secuencia de filas con las que se realizan los cálculos. Puede ser la tabla completa o, por ejemplo, las seis filas por encima de la actual. Trabajar con estas ventanas es diferente a trabajar con solicitudes normales.

```
SELECT
  author_id,
  name,
  price/SUM(price) AS ratio OVER ()
FROM
  books_price;
```

La función que precede a la palabra clave OVER se ejecutará con los datos dentro de la ventana. Si no indicamos ningún parámetro (como aquí), se utilizará todo el resultado de la consulta.

Si queremos agrupar los datos, usamos PARTITION BY:

```
SELECT
  author_id,
  name,
  price/SUM(price) AS ratio OVER (PARTITION BY
                                    author_id)
FROM
  books_price;
```

## Una mirada más detallada a las funciones de la ventana

Palabras clave más importantes al usar funciones de ventana:

ORDER BY: nos permite definir el orden de clasificación de las filas a través de las cuales se ejecutará la ventana

ROWS: donde indicamos los marcos de ventana sobre los cuales se calculará una función de agregación

```
SELECT
  author_id,
  name,
  SUM(price) OVER (ORDER BY
                       author_id
                   ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM
  books_price;
```

Marcos indicadores:

- UNBOUNDED PRECEDING: todas las filas que están por encima de la actual
- N PRECEDING: las *n* filas por encima de la actual
- CURRENT ROW: la fila actual
- N FOLLOWING: las *n* filas debajo de la actual
- UNBOUNDED FOLLOWING: todas las filas debajo de la actual