Resumo do Capítulo: Recursos Avançados de SQL para Analistas

Agrupando Dados

Quando os dados devem ser divididos em grupos por valores de campo, o comando **GROUP BY** é usado:

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

Depois de saber quais campos você estará agrupando, certifique-se de que todos esses campos estejam listados no bloco SELECT e no bloco GROUP BY. A própria função de agregação não deveria ser incluída no bloco GROUP BY; caso contrário, a consulta não funcionará. SQL GROUP BY opera do mesmo jeito que o método `groupby()` em pandas.

GROUP BY pode ser usado com qualquer função de agregação: COUNT, AVG, SUM, MAX, MIN. Você pode chamar várias funções ao mesmo tempo.

Ordenando Dados

Os resultados da análise geralmente são apresentados em uma determinada ordem. Para ordenar os dados por um campo, use o comando **ORDER BY**.

```
SELECT
  field_1,
  field_2,
  ...,
  field_n,
  AGGREGATE_FUNCTION(field) AS here_you_are
FROM
  table_name
WHERE -- se necessário
  condition
GROUP BY
  field_1,
  field_2,
  ...,
  field_n,
ORDER BY -- se necessário. Liste apenas os campos
--pelos quais os dados de tabela devem ser ordenados
  field_1,
  field_2,
  ...,
  field_n,
  here_you_are;
```

Ao contrário de GROUP BY, com ORDER BY, apenas os campos pelos quais queremos ordenar os dados devem ser listados no bloco de comando.

Poderiam ser usados dois modificadores com o comando ORDER BY para ordenar os dados em colunas:

- **ASC** (o padrão) ordena os dados na ordem crescente.
- **DESC** ordena os dados na ordem decrescente.

Os modificadores do ORDER BY são colocados logo após o campo pelo qual os dados são ordenados:

```
ORDER BY
  field_name DESC
-- ordenando os dados na ordem decrescente

ORDER BY
  field_name ASC;
-- ordenando os dados na ordem crescente
```

O comando **LIMIT** define um limite para o número de linhas no resultado. Ele sempre vem no final de uma instrução, seguido pelo número de linhas em que o limite deveria ser definido (*n*):

```
SELECT
  field_1,
  field_2,
  ...,
  field_n,
  AGGREGATE_FUNCTION(field) AS here_you_are
FROM
  table_name
WHERE -- se necessário
  condition
GROUP BY
  field_1,
  field_2,
  ...,
  field_n,
ORDER BY -- se necessário. Liste apenas os campos
--pelos quais os dados de tabela devem ser ordenados
  field_1,
  field_2,
  ...,
  field_n,
  here_you_are
LIMIT -- se necessário
  n;
-- n: o número máximo de linhas a serem retornadas
```

Processando os Dados dentro de um Agrupamento

A construção **WHERE** é usada para classificar dados por linhas. Seus parâmetros são, na verdade, linhas da tabela. Quando precisamos ordenar dados por resultados de funções de agregação, usamos a construção **HAVING**, que tem muito em comum com **WHERE**:

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

```
FROM
  TABLE
WHERE -- se necessário
  condition
GROUP BY
  field_1,
  field_2,
  ...,
  field_n
HAVING
  AGGREGATE_FUNCTION(field_for_grouping) > n
ORDER BY -- se necessário. Liste apenas os campos
--pelos quais os dados devem ser ordenados
  field_1,
  field_2,
  ...,
  field_n,
  here_you_are
LIMIT -- se necessário
  n;
```

A seleção resultante incluirá apenas as linhas para as quais a função de agregação produz resultados que atendem à condição indicada nos blocos HAVING e WHERE.

HAVING e WHERE têm muito em comum. Então, por que não podemos passar todas as nossas condições para um deles? O fato é que o comando WHERE é compilado antes que as operações de agrupamento e aritmética sejam realizadas. É por isso que é impossível definir os parâmetros de ordenação para os resultados de uma função de agregação com WHERE. Daí a necessidade de HAVING.

Preste atenção especial à ordem em que os comandos são introduzidos:

1) GROUP BY

2) HAVING

3) ORDER BY

Esta ordem é **obrigatória**. Caso contrário, o código não funcionará.

Operadores e Funções para Trabalhar com Datas

Temos duas funções principais para trabalhar com os valores de data e hora: **EXTRACT** e **DATE_TRUNC**. Ambas as funções são chamadas no bloco SELECT.

Veja como é a função EXTRACT:

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

EXTRACT, sem surpresa nenhuma, extrai a informação que você precisa da marca temporal. Você pode recuperar:

- `century` — século
- `day` — dia
- `doy` — dia do ano, de 1 a 365/366
- `isodow` — dia da semana sob ISO 8601, o formato internacional de data e hora; Segunda-feira é 1, domingo é 7
- `hour` — hora
- `milliseconds` — milissegundos
- `minute` — minuto
- `second` — segundo
- `month` — mês
- `quarter` — trimestre
- `week` — semana do ano
- `year` — ano

DATE_TRUNC trunca a data quando você precisa apenas de um certo nível de precisão. (Por exemplo, se você precisa saber em que dia foi feito o pedido, mas a hora não importa, você pode usar DATE_TRUNC com o argumento "day".) Diferentemente de EXTRACT, a data truncada resultante é fornecida como uma string. A coluna da qual deveria ser tirada a data completa vem depois de uma vírgula:

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

Você pode usar os seguintes argumentos com a função DATE_TRUNC:

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

Subconsultas

Uma subconsulta, ou **consulta interna**, é uma consulta dentro de uma consulta. Ela recupera informação que será usada posteriormente na **consulta externa**.

As subconsultas podem ser usadas em vários locais dentro de uma consulta. Se uma subconsulta estiver dentro do bloco FROM, SELECT selecionará os dados da tabela que é gerada pela subconsulta. O nome da tabela é indicado na consulta interna e a consulta externa se refere às colunas da tabela. As subconsultas são sempre colocadas entre parênteses:

```
SELECT
SUBQUERY_1.column_name,
SUBQUERY_1.column_name_2
FROM -- para tornar o código legível, coloque as subconsultas em linhas novas
  -- indente as subconsultas
  (SELECT
    column_name,
    column_name_2
```

```
  FROM
    table_name
  WHERE
    column_name = value) AS SUBQUERY_1;
-- lembre-se de nomear sua subconsulta no bloco FROM
```

Você pode precisar de subconsultas em vários locais da sua consulta. Vamos colocar uma no bloco WHERE. A consulta principal comparará os resultados da subconsulta com os valores da tabela no bloco FROM externo. Quando houver uma correspondência, os dados serão selecionados:

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

Agora vamos adicionar a construção IN à nossa amostra e coletar dados de várias colunas:

```
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);
```

Funções de Janela

No SQL, uma janela é uma sequência de linhas nas quais se fazem cálculos. Poderia ser a tabela inteira ou, por exemplo, as seis linhas acima da atual. Trabalhar com essas janelas é diferente de trabalhar com pedidos habituais.

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

A função que precede a palavra-chave OVER será executada nos dados dentro da janela. Se não indicarmos nenhum parâmetro (como aqui), será usado todo o resultado da consulta.

Se quisermos agrupar os dados, usamos PARTITION BY:

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

Uma Análise Mais Detalhada das Funções de Janela

Palavras-chave mais importantes ao usar funções de janela:

ORDER BY — nos permite definir a ordem de ordenação das linhas pelas quais a janela será executada.

ROWS — onde indicamos as molduras das janelas sobre as quais deveria ser calculada uma função de agregação.

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

Indicadores de quadros:

- UNBOUNDED PRECEDING — todas as linhas que estão acima da atual
- N PRECEDING — as *n* linhas acima da atual
- CURRENT ROW — a linha atual
- N FOLLOWING — as *n* linhas abaixo da atual
- UNBOUNDED FOLLOWING — todas as linhas abaixo da atual