Aula 19

Prof: Henrique Augusto Maltauro

Implementar Banco de Dados Para WEB

SQL

- **DQL: SELECT (DISTINCT)**

O comando **DISTINCT** vai remover registros duplicados da consulta.

Ele deve ser usado logo depois do comando **SELECT**.

SQL

- DQL: SELECT (DISTINCT)

```
SELECT
DISTINCT
    C.ID,
    C.NOME
FROM PRODUTO AS P
LEFT JOIN CATEGORIA AS C ON P.ID_CATEGORIA = C.ID
```

SQL

- **DQL: SELECT (TOP)**

O comando **TOP** vai determinar uma quantidade limite de resultados que a consulta vai retornar.

Ele deve ser informado logo depois do comando **SELECT** e logo depois do comando **TOP** deve ser informado qual é a quantidade limite de resultados com um número inteiro.

SQL

- DQL: SELECT (TOP)

```
SELECT                  TOP                  5
    P.ID,
    P.NOME
FROM PRODUTO AS P
```

SQL

- **DQL: SELECT (ORDER BY)**

O comando **ORDER BY** é utilizado para ordenar os resultados da consulta.

Ele é o último comando a ser informado no script da consulta, e logo após ele é informado as colunas que serão levadas em consideração para a ordenação, todas separadas por vírgula.

SQL

- DQL: SELECT (ORDER BY)

```
SELECT
  P.ID,
  P.NOME,
  C.NOME AS CATEGORIA,
  P.VALOR
FROM PRODUTO AS P
 LEFT JOIN CATEGORIA AS C ON P.ID_CATEGORIA = C.ID
ORDER BY C.ID, P.VALOR
```

SQL

- **DQL: SELECT (ORDER BY)**

Por padrão, os registros são informados em **ordem crescente**.

Mas, depois de cada coluna é possível informar os comandos **ASC** para indicar que a ordenação será de forma **crescente** e **DESC** para indicar que a ordenação será de forma **decrescente**.

SQL

- DQL: SELECT (ORDER BY)

```
SELECT
  P.ID,
  P.NOME,
  C.NOME AS CATEGORIA,
  P.VALOR
FROM PRODUTO AS P
 LEFT JOIN CATEGORIA AS C ON P.ID_CATEGORIA = C.ID
ORDER BY C.ID DESC, P.VALOR ASC
```

SQL

- `WHERE`

Na estrutura do comando `WHERE`, ficou faltando algumas outras validações que são possíveis realizar para filtrar os registros.

SQL

- `WHERE`

Estas validações são:

- ➔ `IS NULL`
- ➔ `IS NOT NULL`
- ➔ `IN`
- ➔ `BETWEEN`
- ➔ `LIKE`

SQL

- `WHERE: IS NULL`

O comando `IS NULL` vai validar se a coluna informada **possui valores nulos**.

Se essa validação retornar um valor verdadeiro, aquele registro será apresentado no resultado da consulta.

SQL

- WHERE: IS NULL

```sql
SELECT
    P.ID,
    P.NOME
FROM          PRODUTO          AS          P
WHERE P.ID_UNIDADE IS NULL
```

SQL

- `WHERE: IS NOT NULL`

O comando `IS NOT NULL` vai validar se a coluna informada **não possui valores nulos**.

Se essa validação retornar um valor verdadeiro, aquele registro será apresentado no resultado da consulta.

SQL

- WHERE: IS NOT NULL

```
SELECT
    P.ID,
    P.NOME
FROM          PRODUTO          AS          P
WHERE P.ID_UNIDADE IS NOT NULL
```

SQL

- **WHERE: IN**

O comando **IN** vai receber uma lista de valores, todos eles **dentro de parênteses e separados por vírgula**, e vai validar se o valor da coluna informada é **igual a algum dos valores da lista**.

Se essa validação retornar um valor verdadeiro, aquele registro será apresentado no resultado da consulta.

SQL

- WHERE: IN

```sql
SELECT
    P.ID,
    P.NOME
FROM PRODUTO AS P
WHERE P.ID_CATEGORIA IN (1, 5, 3, 7)
```

SQL

- `WHERE: IN`

É possível reverter o comando `IN` com um comando `NOT` antes dele.

SQL

- WHERE: IN

```sql
SELECT
 P.ID,
P.NOME
     FROM          PRODUTO          AS          P
WHERE P.ID_CATEGORIA NOT IN (1, 5, 3, 7)
```

SQL

- `WHERE: BETWEEN`

O comando `BETWEEN` vai receber dois valores, separados por um comando `AND`, e vai validar se o valor da coluna informada está **entre alguns dos dois valores**.

Se essa validação retornar um valor verdadeiro, aquele registro será apresentado no resultado da consulta.

SQL

- WHERE: BETWEEN

```sql
SELECT
    P.ID,
    P.NOME
FROM PRODUTO AS P
WHERE P.VALOR BETWEEN 100 AND 1000
```

SQL

- **WHERE: BETWEEN**

É possível reverter o comando **BETWEEN** com um comando **NOT** antes dele.

SQL

- WHERE: BETWEEN

```sql
SELECT
    P.ID,
    P.NOME
FROM PRODUTO AS P
WHERE P.VALOR NOT BETWEEN 100 AND 1000
```

SQL

- `WHERE: LIKE`

O comando `LIKE` vai receber uma **string padronizada** e vai validar se o valor da coluna informada **respeita o padrão da string**.

Se essa validação retornar um valor verdadeiro, aquele registro será apresentado no resultado da consulta.

SQL

- **WHERE: LIKE**

Para entendermos essa **string padronizada**, precisamos entender dois símbolos que irão ser usados para definir esse padrão.

São eles o símbolo de **porcentagem %**, e o símbolo de **underline _**.

SQL

- **WHERE: LIKE**

O símbolo de **porcentagem %** representa **zero, um, ou múltiplos caracteres**.

O símbolo de **underline _** representa **um único caracter**.

Compreendido isso, a **string padronizada** vai seguir uma ideia mais ou menos como na tabela a seguir.

SQL

- WHERE: LIKE

‘c%’	Qualquer valor que comece com a letra c
‘%c’	Qualquer valor que termine com a letra c
‘%c%’	Qualquer valor que tenha a letra c em qualquer lugar
‘_r%’	Qualquer valor que tenha a letra r na segunda posição
‘c_%’	Qualquer valor que comece com a letra c, e que tenha pelo menos 2 caracteres
‘c__%’	Qualquer valor que comece com a letra c, e que tenha pelo menos 3 caracteres
‘c%r’	Qualquer valor que comece com a letra c, e termine com a letra r

SQL

- WHERE: LIKE

```
SELECT
    P.ID,
    P.NOME
FROM          PRODUTO          AS          P
WHERE P.NOME LIKE '%t%'
```

SQL

- `WHERE: LIKE`

É possível reverter o comando `LIKE` com um comando `NOT` antes dele.

SQL

- WHERE: LIKE

```
SELECT
 P.ID,
P.NOME
    FROM        PRODUTO        AS        P
WHERE P.NOME NOT LIKE '%t%'
```

Exercício

Exercício

gg.gg/SenacBD19

github.com/hmaltaurodev/slides