Aula 13 - Outros comandos SQL

UC10 - Criar banco de dados

- Vamos ver alguns comandos muito úteis para fazer consultas em SQL.

Aliases

- Podemos criar apelidos tanto para atributos quanto para tabelas. Um apelido pode ser criado tanto no SELECT quanto no FROM.
- Para criar um apelido, utiliza-se a palavra “AS”. No entanto, a palavra “AS” é opcional. Pode-se simplesmente colocar o apelido depois do nome da tabela ou do atributo.
- O apelido para um atributo é particularmente útil quando usamos uma expressão aritmética ou função no atributo, porque nesse caso a coluna resultante ficará sem um nome. Também podemos usá-lo simplesmente para mudar o nome da coluna que irá aparecer no resultado da consulta. O exemplo abaixo cria um apelido para a coluna “tbClasse.preço_classe x 1,05”. Note que o apelido está entre aspas porque temos uma frase. Se o apelido tivesse uma única palavra não haveria necessidade de aspas.

```
SELECT tbClasse.nome_classe,
       tbClasse.preco_classe * 1.05 AS 'Preço com aumento de 5%'

FROM tbClasse;
```

- Outro uso do apelido é para tabelas. Podemos criar um apelido para uma tabela e utilizarmos esse apelido tanto no SELECT quanto no WHERE.

Exemplo: Encontre o nome dos títulos e o nome de sua respectiva categoria para filmes de 2009.

```
SELECT T.nome_titulo, C.nome_categoria
FROM tbTitulo T
INNER JOIN tbCategoria C ON C.codigo_categoria = T.codigo_categoria
WHERE T.ano = 2009;
```

- Finalmente, um apelido pode ser usado quando precisamos comparar dados de uma mesma tabela entre si. Por exemplo, suponha que queiramos saber todos os filmes que possuem o mesmo nome, mas foram lançados em anos diferentes. Para isso, temos que pegar um registro da tabela tbTitulo e comparar com os registros da mesma tabela, como mostra o exemplo:

```
SELECT T1.nome_titulo, T1.ano
FROM tbTitulo T1
INNER JOIN tbTitulo T2 ON T1.nome_titulo = T2.nome_titulo
WHERE T2.ano <> T1.ano;
```

Comando LIKE

- O comando LIKE é utilizado quando queremos comparar strings em uma consulta. No entanto, diferentemente dos operadores relacionais de igual (=) e diferente (<>) que comparam a string exata, o comando LIKE permite que utilizemos operadores que comparam se parte de uma string está contida em algum registro de uma tabela.
- Os operadores utilizados são:
    - Porcentagem (%): para comparar parte de uma string.
    - Sublinhado (_): para comparar um caractere.

Selecione todos os títulos que comecem com a Letra 'M' e o ano em que foram lançados:

```
SELECT T.nome_titulo, T.ano
FROM tbTitulo T
WHERE T.nome_titulo LIKE 'M%';
```

Selecione todos os títulos que contenham a substring 'ama':

```
SELECT T.nome_titulo
FROM tbTitulo T
WHERE T.nome_titulo LIKE '%ama%';
```

Selecione todos os títulos que tenham apenas 5 caracteres:

```
SELECT T.nome_titulo
FROM tbTitulo T
WHERE T.nome_titulo LIKE '_ _ _ _ _';
```

Selecione todos os títulos que tenham pelo menos 5 caracteres e termine com 'a':

```
SELECT T.nome_titulo
FROM tbTitulo T
WHERE T.nome_titulo LIKE '_ _ _ _%a';
```

- Podemos utilizar o caractere \ se quisermos comparar caracteres especiais. Para pesquisar diferenças em strings pode-se usar o comando NOT LIKE. Por exemplo:

Selecione todos os títulos que não tenham a substring ‘ama%’:

```
SELECT T.nome_titulo
FROM tbTitulo T
WHERE T.nome_titulo NOT LIKE '%ama\%%';
```

Comando ORDER BY

- O comando ORDER BY, como o próprio nome diz, é utilizado para ordenar o resultado de uma consulta. Ele não altera a ordem dos dados na tabela física, somente o resultado da consulta aparece ordenado.
- Por padrão, quando você usa o ORDER BY, a maioria dos SGBD ordenam em ordem crescente (do menor para o maior). No entanto, é possível determinar a ordem utilizando explicitamente:
    - ASC para ordem crescente (do menor para o maior);
    - DESC para ordem decrescente (do maior para o menor);

Selecione o nome dos títulos e o ano em ordem crescente de nome e decrescente de ano.

```
SELECT T.nome_titulo, T.ano
FROM tbTitulo T
ORDER BY T.nome_titulo ASC, T.ano DESC;
```

- No exemplo anterior, o resultado foi ordenado em ordem crescente pelo nome do título e em caso de nomes iguais de títulos ordena-se pelo ano em ordem decrescente.
- É importante destacar que quando você usa o ORDER BY, o SGBD terá um trabalho adicional para fazer a ordenação do resultado, o que pode comprometer o desempenho da consulta. Por isso, só use o comando ORDER BY quando ele for realmente necessário.

Funções Agregadas

- Todo SGDB oferece um conjunto de funções para tratamento de data, hora, caracteres etc. Outro uso dessas funções é o cálculo estatístico. Para isso, existem 5 funções comuns a qualquer SGDB, chamadas funções agregadas, que são bastante utilizadas em consultas SQL.

Função AVG

- Essa função calcula a média aritmética, dado um conjunto de valores numéricos. Por exemplo, podemos calcular a média de preços de filmes das classes.

```
SELECT AVG(tbClasse.preco_classe) AS Media
FROM tbClasse;
```

Função SUM

- Assim como a função AVG, a função SUM também é uma função numérica. Essa função faz o somatório de um conjunto de valores numéricos. Por exemplo, podemos somar todos os anos da tabela tbTitulo, como mostrado abaixo.

```
SELECT SUM(tbTitulo.ano) AS 'Soma dos Anos'
FROM tbTitulo;
```

Função MIN e Função MAX

- As funções MIN e MAX são funções que retornam, respectivamente, o valor mínimo e o valor máximo de um campo. Essas funções funcionam com conjuntos de dados numéricos e não numéricos.

Selecione a data de empréstimo mais antiga.

```
SELECT MIN (tbEmprestimoDevolucao.data_emprestimo) AS 'Data mais Antiga'
FROM tbEmprestimoDevolucao;
```

Selecione a data de cadastro do último cliente.

```
SELECT MAX(tbCliente.data_cadastro) AS 'Data do último cadastro'
FROM tbCliente;
```

Função COUNT

- A função COUNT tem por objetivo contar o número de ocorrências de um atributo na base de dados. Esta função pode ser aplicada a qualquer tipo de atributo (numérico e não numérico). Por exemplo, suponha que queiramos contar o número de filmes que o cliente Pedro emprestou em 2009.

```
SELECT COUNT(ED.data_emprestimo) AS 'Numero de Locações', C.nome_cli
FROM tbEmprestimoDevolucao ED
INNER JOIN tbCliente C ON ED.codigo_cli = C.codigo_cli
WHERE ED.data_emprestimo BETWEEN '2009-01-01' AND '2009-12-31';
```

- Ou ainda podemos querer descobrir quantos títulos existem da categoria “Terror”.

```
SELECT COUNT(T.codigo_titulo) AS 'Quantidade de Títulos'
FROM tbTitulo T
INNER JOIN tbCategoria C ON T.codigo_categoria = C.codigo_categoria
WHERE C.nome_categoria LIKE 'Terror';
```

- Para contar o número total de registros em uma tabela, utilizamos a função COUNT(*). Por exemplo, vamos contar quantas linhas existem na tabela tbFilme.

```
SELECT COUNT(*) AS 'Quantidade de Linhas em tbFilme'
FROM tbFilme;
```

- O comando DISTINCT pode ser utilizado para eliminar repetições em qualquer função. No entanto, a SQL não permite o uso do DISTINCT com COUNT(*), uma vez que o DISTINCT serve para eliminar redundâncias e o COUNT(*) somente conta o número de linhas.
- Todas as funções agregadas, exceto o COUNT(*), ignoram os valores nulos dos seus conjuntos de valores de entrada.

Comando GROUP BY

- O comando GROUP BY é usado para formar grupos e categorizar os resultados por meio desses grupos. Este comando sempre estará associado a uma função agregada uma vez que seu objetivo é o de aplicar uma função agregada, a um grupo de registros. Por exemplo, suponha que você queira descobrir quantos títulos existem por categoria. A consulta ficaria como mostrado a seguir.

```
SELECT COUNT(T.codigo_titulo) AS 'Quantidade de Títulos', C.nome_categoria
FROM tbTitulo T
INNER JOIN tbCategoria C ON T.codigo_categoria = C.codigo_categoria
GROUP BY C.nome_categoria;
```

- A consulta anterior agrupa a quantidade de títulos pelo nome da categoria. Assim, registros que possuem os mesmos valores de categoria são contados e colocados em um grupo.
- Se quiséssemos saber a quantidade total de títulos, não seria necessário utilizar a cláusula GROUP BY. A consulta ficaria assim:

```
SELECT COUNT(T.codigo_titulo) AS 'Quantidade de Títulos'
FROM tbTitulo T
INNER JOIN tbCategoria C ON T.codigo_categoria = C.codigo_categoria;
```

- Como dito anteriormente, pode-se utilizar o DISTINCT com uma função agregada. Por exemplo, vamos descobrir a quantidade de títulos que cada cliente já tomou emprestado.

```
SELECT C.nome_cli, COUNT(DISTINCT F.codigo_titulo) AS 'Quantidade de Títulos'
FROM tbCliente C
INNER JOIN tbEmprestimoDevolucao ED ON C.codigo_cli = ED.codigo_cli
INNER JOIN tbFilme F ON F.codigo_filme = ED.codigo_filme
GROUP BY C.nome_cli
ORDER BY COUNT(DISTINCT F.codigo_titulo);
```

- O DISTINCT foi utilizado na consulta para impedir que no caso do cliente que pegou um mesmo título em duas ou três diferentes oportunidades, esse título seja contado mais de uma vez. Isso porque não queremos saber a quantidade de empréstimos de um cliente e sim quantos títulos diferentes ele já emprestou.
- Note que foi feito o INNER JOIN de 3 tabelas embora as informações que eram necessárias para a consulta estivesse nas tabelas tbFilme e tbCliente. A tabela tbEmprestimoDevolucao teve que ser utilizada na consulta uma vez que é ela quem relaciona as duas outras tabelas (através das FK).
- Também foi utilizado um ORDER BY a fim de que o resultado seja ordenado do cliente que emprestou mais títulos para o que emprestou menos títulos.

Comando HAVING

- Vimos que o WHERE é o comando que permite definir condições para uma consulta. No entanto, é muito comum precisarmos aplicar uma condição aos grupos formados pelo GROUP BY em vez de aplicar a condição a todos os registros da tabela. Para aplicar a condição aos grupos, usamos o comando HAVING.
- Sendo assim, o comando HAVING sempre irá aparecer junto a um comando GROUP BY. Por exemplo, suponha que você queria descobrir o nome dos clientes que tiveram mais de 100 empréstimos (agora pode ser o mesmo título várias vezes).

```
SELECT C.nome_cli, COUNT(ED.data_emprestimo) AS 'Quantidade de Emprestimos'
FROM tbCliente C
INNER JOIN tbEmprestimoDevolucao ED ON C.codigo_cli = ED.codigo_cli
GROUP BY C.nome_cli
HAVING COUNT(ED.data_emprestimo) > 100;
```

- Primeiramente, deve-se contar o número de empréstimos por cliente, para, em seguida, aplicar a condição do HAVING e retornar apenas aqueles clientes que tiveram mais de 100 locações.
- Se uma cláusula WHERE e HAVING aparecem na mesma consulta, o predicado que aparece em WHERE é aplicado primeiro, depois são criados os grupos para, em seguida, aplicar-se o HAVING. Por exemplo, imagine que agora você queira descobrir o nome dos clientes que tiveram 30 ou mais empréstimos em janeiro de 2010.

```
SELECT C.nome_cli, COUNT(ED.data_emprestimo) AS 'Quantidade de Emprestimos'
FROM tbCliente C
INNER JOIN tbEmprestimoDevolucao ED ON C.codigo_cli = ED.codigo_cli
WHERE ED.data_emprestimo BETWEEN '2010-01-01' AND '2010-01-31'
GROUP BY C.nome_cli
HAVING COUNT(ED.data_emprestimo) >= 30;
```

- Nesse caso, primeiro foram selecionados todos os empréstimos em janeiro de 2010, em seguida foram contados o número de empréstimos somente para as pessoas que emprestaram filmes em janeiro de 2010 e, finalmente, foram selecionados apenas aqueles registros em que o cliente teve 30 ou mais empréstimos.

Valores Nulos

- Como visto anteriormente, o valor nulo indica a ausência de informação. Para testar se um atributo possui valores nulos, ou não, é necessário utilizar o operador IS junto à palavra NULL. Por exemplo, se quisermos saber quais clientes ainda estão em atraso, podemos escrever:

```
SELECT DISTINCT C.nome_cli 'Clientes com Filmes em Atraso'
FROM tbCliente C
INNER JOIN tbEmprestimoDevolucao ED ON C.codigo_cli = ED.codigo_cli
WHERE ED.data_devolucao_prevista < CURDATE()
AND ED.data_devolucao_efetiva IS NULL;
```

- A função CURDATE() que aparece na consulta acima, retorna a data atual do servidor. Essa função está sendo utilizada para verificar se a data atual é maior do que a data cadastrada no atributo referente à devolução prevista do filme. Caso isso ocorra verifica-se se o campo referente à devolução efetiva do filme está preenchido e se não estiver caracteriza que o cliente não entregou o filme e portanto está em atraso.
- Não podemos utilizar os operadores relacionais = e <> para testar a ausência de um valor. Isso ocorre porque NULL não é valor (indica a ausência dele) e só podemos utilizar os operadores relacionais para testar valores.
- O predicado IS NOT NULL pode ser utilizado para verificar se existem valores para um atributo.
- Numa expressão aritmética, (+, –, * e /), se qualquer um dos valores de entrada for nulo, o resultado da expressão também o será.

Tabelas Derivadas

- Você já sabe que no FROM sempre definimos as tabelas que iremos utilizar numa consulta e que o resultado de uma consulta é sempre uma tabela. Sendo assim, quando você escreve uma consulta dentro do FROM e essa consulta é executada, gera-se uma tabela que poderá ser consultada normalmente. A essa consulta que escrevemos dentro do FROM é que damos o nome de tabela derivada.
- O SGBD primeiro executa o que está no FROM, gera a tabela em memória e, em seguida, executa o WHERE da consulta principal. Por exemplo, vamos reescrever usando uma tabela derivada, a consulta apresentada no primeiro box da página 105, que retorna o nome dos clientes que tiveram mais de 100 empréstimos.

```
SELECT tbNovaTabela.nome_cli AS 'NOME DO CLIENTE', tbNovaTabela.
quantidade
  FROM (SELECT C.nome_cli, COUNT(ED.data_emprestimo) AS quantidade
        FROM tbCliente C
        INNER JOIN tbEmprestimoDevolucao ED ON C.codigo_cli = ED.codigo_cli
        GROUP BY C.nome_cli) AS tbNovaTabela
  WHERE tbNovaTabela.quantidade > 100;
```

- Observe que tivemos que criar um apelido (tbNovaTabela) para a tabela gerada no FROM, para que pudéssemos referenciar essa tabela tanto no SELECT mais externo, quanto no WHERE.