

N_BANC DAD II5 - Texto de apoio

Site: [EAD Mackenzie](#)

Tema: BANCO DE DADOS {TURMAS 03B} 2023/1

Livro: N_BANC DAD II5 - Texto de apoio

Impresso por: FELIPE BALDIM GUERRA .

Data: sexta, 14 abr 2023, 02:30

Índice

1. LINGUAGEM SQL: COMANDOS INSERT, UPDATE, DELETE E SELECT BÁSICO

2. CREATE TABLE

3. DROP TABLE

4. INSERT

5. UPDATE

6. DELETE

7. SELECT

7.1. SELECT com Funções Agregadas

7.2. SELECT com GROUP BY

8. REFERÊNCIAS

1. LINGUAGEM SQL: COMANDOS INSERT, UPDATE, DELETE E SELECT BÁSICO

Introdução

A linguagem **SQL** (*Structured Query Language*) é a linguagem padrão de acesso a bancos de dados relacionais. Alguns exemplos de bancos de dados relacionais são: Oracle, Microsoft SQL Server, MySQL, dentre outros.

Os comandos da linguagem SQL podem ser divididos em três classes:

- Linguagem de Definição de Dados (**DDL**): inclui comandos para definir, alterar e remover tabelas e índices.
- Linguagem de Manipulação de Dados (**DML**): inclui comandos para inserir, remover, atualizar e consultar os dados armazenados nas tabelas.
- Linguagem de Controle de Dados (**DCL**): inclui comandos para trabalhar em ambiente multiusuário, permitindo estabelecer níveis de segurança e manipular transações.

A seguir, você aprenderá os seguintes comandos da linguagem SQL:

- **CREATE TABLE** e **DROP TABLE** (que fazem parte da DDL).

- **INSERT**, **UPDATE**, **DELETE** e **SELECT** (envolvendo uma única tabela). Todos esses comandos fazem parte da DML.

Importante: Você deve utilizar algum banco de dados relacional para aprender e praticar a linguagem SQL. Uma sugestão é utilizar o Oracle na nuvem, sendo necessário apenas acessar o link livesql.oracle.com, criar um usuário e começar a praticar. Assista à videoaula “Como utilizar o Oracle Live SQL”, da professora Elisângela Botelho Gracias, com um exemplo de utilização deste serviço da Oracle.

Alguns dos operadores que você pode utilizar na linguagem SQL são:

- Lógicos: **AND**, **OR** e **NOT**.
- Aritméticos: **+** (adição), **-** (subtração), ***** (multiplicação) e **/** (divisão).
- Relacionais: **<** e **<=** (menor e menor ou igual, respectivamente), **>** e **>=** (maior e maior ou igual, respectivamente), **<>** e **=** (diferente e igual, respectivamente), **LIKE** (especifica um padrão de comparação) e **BETWEEN** (especifica um intervalo de valores).
- Conjunturais: **IN**, **NOT IN**, dentre outros.

2. CREATE TABLE

Este comando **cria uma tabela**, dando-lhe um nome e especificando seus atributos, chave primária, chave estrangeira (se for o caso) e outras restrições. Para cada atributo, é definido um nome, um domínio e, se necessário, uma restrição.

Sintaxe do comando CREATE TABLE:

```
CREATE TABLE nome_tabela
(nome_atributo1      tipo    [NOT NULL] [UNIQUE] [CHECK( )],
 nome_atributo2      tipo    [NOT NULL] [UNIQUE] [CHECK( )],
 ...
 nome_atributoN      tipo    [NOT NULL] [UNIQUE] [CHECK( )],
 [PRIMARY KEY (atributos_compoem_chave_primaria)],
 [FOREIGN KEY (nome_atributo) REFERENCES tabela_origem(atributo_origem)]
);
```

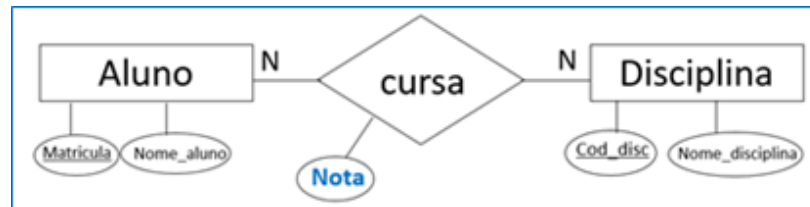
Em que:

- **nome_tabela**: indica o nome da tabela a ser criada;
- **nome_atributo**: indica o nome do campo a ser criado na tabela;
- **tipo**: indica a definição do tipo de atributo (INTEGER, VARCHAR(N), CHAR(N) etc., sendo "N" o número de caracteres);
- **tabela_origem**: indica a tabela em que a chave estrangeira foi originalmente criada;
- **atributo_origem**: indica o nome da chave primária na tabela em que foi criada;
- **NOT NULL**: não permite a inserção de valores nulos;
- **UNIQUE**: não permite que os valores de um atributo se repitam. É possível ter vários atributos UNIQUE;
- **CHECK(condição)**: permite validar os valores inseridos neste atributo, por meio de uma condição;
- **PRIMARY KEY**: é o atributo (ou atributos) que identifica unicamente cada linha da tabela, ou seja, não pode se repetir nem ser nulo;
- **FOREIGN KEY**: é o atributo que faz o relacionamento entre as tabelas e ele está sempre ligado à chave primária de uma outra tabela.

Atenção: tudo que está entre [] é opcional, mas se for utilizar, tire-o. E o ponto e vírgula (;) indica a finalização de um comando da linguagem SQL.

Exemplo que será utilizado para **explicar os comandos CREATE TABLE, INSERT, UPDATE e DELETE:**

Modelo Entidade-Relacionamento (MER)



Modelo Relacional (gerado a partir do MER)

Aluno = {Matricula, Nome_Aluno}

Disciplina = {Cod_Disc, Nome_Disc}

Aluno_Disciplina = {Matricula, Cod_Disc, Nota}

- *Matricula* é **chave estrangeira** que referencia *Matricula* da tabela *Aluno*
- *Cod_Disc* é **chave estrangeira** que referencia *Cod_Disc* da tabela *Disciplina*

Exemplo 1: criação das três tabelas desse banco de dados.

```
CREATE TABLE Aluno
(Matricula INTEGER CHECK(Matricula >= 1),
 Nome_aluno VARCHAR(50) NOT NULL,
 PRIMARY KEY (Matricula)
);
```

```
CREATE TABLE Disciplina
(Cod_disc INTEGER CHECK(Cod_disc >= 1),
 Nome_disc VARCHAR(25) NOT NULL UNIQUE,
 PRIMARY KEY (Cod_disc)
);
```

```
CREATE TABLE Aluno_Disciplina
(Matricula INTEGER,
 Cod_disc INTEGER,
 Nota INTEGER CHECK(Nota BETWEEN 0 AND 10),
 PRIMARY KEY (Matricula, Cod_disc),
 FOREIGN KEY (Matricula) REFERENCES Aluno(Matricula),
 FOREIGN KEY (Cod_disc) REFERENCES Disciplina(Cod_disc)
);
```

O que você deve analisar, cuidadosamente, no script de criação das tabelas Aluno, Disciplina e Aluno_Disciplina:

- O tipo de dado e restrições (se houver) de cada um dos atributos das tabelas.
- Chave primária de cada uma das tabelas.
- Chaves estrangeiras da tabela Aluno_Disciplina.

3. DROP TABLE

Este comando **remove uma tabela** existente no banco de dados.

Sintaxe do comando DROP TABLE:

```
DROP TABLE nome_tabela;
```

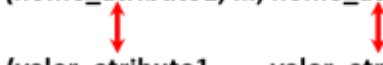
Atenção: nas tabelas que criamos anteriormente, não seria possível eliminar a tabela Aluno ou a tabela Disciplina antes de eliminar a tabela Aluno_Disciplina, pois esta tem duas chaves estrangeiras que referenciam as tabelas Aluno e Disciplina.

4. INSERT

O comando INSERT possibilita a **inclusão de dados** em uma tabela.

Sintaxe do comando INSERT:

```
INSERT  
INTO nome_tabela (nome_atributo1, ..., nome_atributoN)  
VALUES (valor_atributo1, ..., valor_atributoN);
```



Para **inserir dados em uma tabela**, é necessário conhecer o **script de criação da tabela** para ver as restrições existentes, como PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE, CHECK e **não as violar**.

Exemplo 2: inserção de um novo aluno com matrícula 1 e nome 'Joao'.

```
INSERT  
INTO Aluno (Matricula, Nome_aluno)  
VALUES (1, 'Joao');
```

Resultado da inserção (Exemplo 2) e alguns comentários:

Aluno

| Matricula | Nome_aluno |
|-----------|------------|
| 1 | Joao |

- valores numéricos são inseridos como se escrevem, enquanto as **cadeias de caracteres** têm sempre que ser delimitadas por **aspas simples**;
- Os valores serão inseridos nos respectivos atributos, obedecendo a ordem em que foram escritos, ou seja, o primeiro valor corresponde ao primeiro atributo, o segundo valor corresponde ao segundo atributo e assim sucessivamente.

Exemplo 3: inserção de mais dois alunos.

```
INSERT
INTO Aluno (Matricula, Nome_aluno)
VALUES (2, 'Maria');

INSERT
INTO Aluno (Matricula, Nome_aluno)
VALUES (3, 'Sergio');
```

Resultado da inserção (Exemplo 3) e alguns comentários:

| Aluno | |
|-----------|------------|
| Matricula | Nome_aluno |
| 1 | Joao |
| 2 | Maria |
| 3 | Sergio |

- observe que a tabela Aluno agora tem três linhas e cada aluno possui um valor para a Matrícula que é diferente dos demais alunos.

Exemplo 4: inserção de duas disciplinas e, depois, inserção de um mesmo aluno em duas disciplinas (tabela Aluno_Disciplina).

```
INSERT
INTO Disciplina (Cod_disc, Nome_disc)
VALUES (100, 'Banco de Dados I');

INSERT
INTO Disciplina (Cod_disc, Nome_disc)
VALUES (101, 'Banco de Dados II');
```

```

INSERT
INTO Aluno_Disciplina (Matricula, Cod_disc, Nota)
VALUES (1, 100, 9);

INSERT
INTO Aluno_Disciplina (Matricula, Cod_disc, Nota)
VALUES (1, 101, 8);

```

O que é importante você analisar nessas inserções:

- O atributo Nome_Disc foi definido como UNIQUE, então não é possível ter **duas disciplinas com mesmo nome**.
- A tabela **Aluno_Disciplina** tem **duas chaves estrangeiras**, portanto só é possível inserir alunos que já existam na tabela Aluno e disciplinas que já existam na tabela Disciplina.
- No atributo **Nota**, só poderá ser inserido um valor entre **0 e 10**.

Agora, após as inserções, as tabelas **Aluno**, **Disciplina** e **Aluno_Disciplina** têm os seguintes **dados**:

| Aluno | | Disciplina | |
|-----------|------------|------------|-------------------|
| Matricula | Nome_aluno | Cod_disc | Nome_disc |
| 1 | Joao | 100 | Banco de Dados I |
| 2 | Maria | 101 | Banco de Dados II |
| 3 | Sergio | | |

| Aluno_Disciplina | | |
|------------------|----------|------|
| Matricula | Cod_disc | Nota |
| 1 | 100 | 9 |
| 1 | 101 | 8 |

O INSERT poderá falhar em inúmeras situações, dentre as quais se destacam:

- ao tentar inserir mais de uma vez a **mesma chave primária**;
- ao tentar inserir mais de uma vez o mesmo valor em um atributo **UNIQUE**;
- ao tentar inserir o valor **NULL** em um atributo **NOT NULL**;
- se o tipo do dado enviado na cláusula VALUES não estiver de acordo com o tipo de dado definido para aquele atributo;

- se algum dos atributos obrigatórios for ignorado;
- se o número de atributos for diferente do número de valores;
- se existir algum tipo de restrição no atributo a que os dados não obedecem.

5. UPDATE

Este comando possibilita a **atualização de dados** em uma tabela.

Sintaxe do comando UPDATE:

```
UPDATE nome_tabela  
SET nome_atributo1 = valor1, ... nome_atributoN = valorN  
[WHERE (condições)];
```

Exemplo 5: aumente em um ponto a nota dos alunos que tiraram uma nota maior ou igual a 7 e menor ou igual a 9.

```
UPDATE Aluno_Disciplina  
SET Nota = Nota + 1  
WHERE (Nota >= 7) AND (Nota <= 9);
```

Resultado da inserção (Exemplo 5) e alguns comentários:

| Aluno_Disciplina | | |
|------------------|----------|------|
| Matricula | Cod_disc | Nota |
| 1 | 100 | 10 |
| 1 | 101 | 9 |

- observe que, na cláusula WHERE, foram utilizados vários **operadores** (>=, AND e <=) para criar as condições.

Importante: se você não colocar nenhuma condição para a atualização dos dados em uma tabela, todas as linhas serão atualizadas (desde que não viole as restrições definidas).

6. DELETE

Este comando possibilita a **remoção de dados** em uma tabela.

Sintaxe do comando DELETE:

```
DELETE  
FROM nome_tabela  
[WHERE (condições)];
```

Exemplo 6: elimine todas as linhas da tabela Aluno_Disciplina que tenham o Cod_disc igual a 101.

```
DELETE  
FROM Aluno_Disciplina  
WHERE (Cod_disc = 101);
```

Resultado da inserção (Exemplo 6):

| Aluno_Disciplina | | |
|------------------|----------|------|
| Matricula | Cod_disc | Nota |
| 1 | 100 | 10 |
| 1 | 101 | 9 |

Importante: se você não colocar nenhuma condição para a remoção dos dados em uma tabela, todos os dados desta tabela serão removidos, desde que não viole as restrições definidas. A tabela ainda existe, mas, agora, sem nenhum dado!

Atenção: não deixe de assistir à videoaula “Linguagem SQL: comandos CREATE e DROP TABLE, INSERT, UPDATE e DELETE”, com a professora Elisângela Botelho Gracias, a qual contém uma explicação simples e breve sobre os comandos CREATE e DROP TABLE, INSERT, UPDATE e DELETE.

7. SELECT

Este comando possibilita a consulta de uma ou mais tabelas, de acordo com os critérios estabelecidos e com as necessidades. Nesta aula, você aprenderá a respeito de consultas envolvendo apenas uma tabela.

Sintaxe do comando SELECT:

```
SELECT [DISTINCT] nome_atributo1,... nome_atributoN
FROM nome_tabela1, ... nome_tabelaN
[WHERE (condições)]
[GROUP BY nome_atributo1,... nome_atributoN]
[HAVING (condições)]
[ORDER BY nome_atributo1 {ASC | DESC}, ...
        nome_atributoN {ASC | DESC}];
```

Na sintaxe acima, tudo que está entre [] é opcional e:

- **SELECT**: é o que se deseja no resultado da consulta;
- **DISTINCT**: não permite repetição de valores no resultado;
- **FROM**: é o local de onde buscar os dados necessários;
- **WHERE**: são condições para busca dos resultados;
- **GROUP BY**: formam agrupamento de dados;
- **HAVING**: são as condições para o agrupamento;
- **ORDER BY**: estabelece a ordenação desejada do resultado.

Os **exemplos** utilizados para explicar o **comando SELECT** utilizarão o seguinte banco de dados, apresentado a seguir:

- a Figura 1 apresenta o MER (foi utilizada a ferramenta brModelo);
- na Figura 2, temos o Modelo Relacional (foi utilizada a ferramenta DBDesigner);
- a Figura 3 tem as tabelas populadas deste banco de dados.

Figura 1 – Modelo Entidade-Relacionamento (MER)

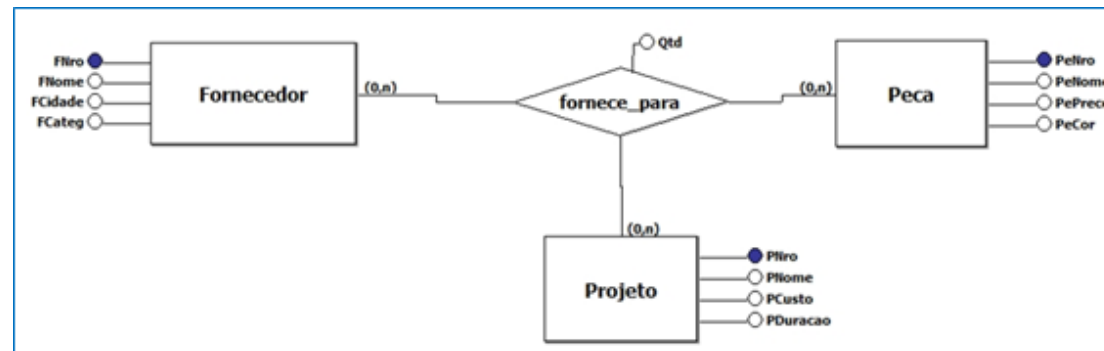


Figura 2 – Modelo Relacional (gerado a partir do MER)

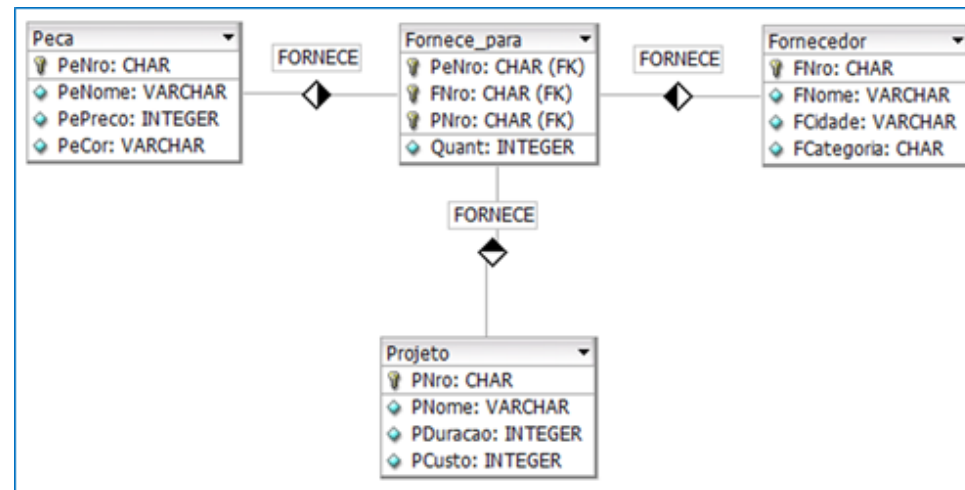


Figura 3 – Tabelas populadas com dados

PEÇA

| PENRO | PENOME | PEPRECO | PECOR |
|-------|----------|---------|----------|
| PE1 | Cinto | 22 | Azul |
| PE2 | Volante | 18 | Vermelho |
| PE3 | Lanterna | 14 | Preto |
| PE4 | Limpador | 9 | Amarelo |
| PE5 | Painel | 43 | Vermelho |

PROJETO

| PNRO | PNOME | PDURACAO | PCUSTO |
|------|---------|----------|--------|
| P1 | Detroit | 5 | 43000 |
| P2 | Pegasus | 3 | 37000 |
| P3 | Alfa | 2 | 26700 |
| P4 | Sea | 3 | 21200 |
| P5 | Paraíso | 1 | 17000 |

FORNECE_PARA

| PENRO | FNRO | PNRO | QUANT |
|-------|------|------|-------|
| PE1 | F5 | P4 | 5 |
| PE2 | F2 | P2 | 1 |
| PE3 | F3 | P4 | 2 |
| PE4 | F4 | P5 | 3 |
| PE5 | F1 | P1 | 1 |
| PE2 | F2 | P3 | 1 |
| PE4 | F3 | P5 | 2 |

FORNECEDOR

| FNRO | FNOME | FCIDADE | FCATEGORIA |
|------|------------|------------|------------|
| F1 | Plastec | Campinas | B |
| F2 | CM | Sao Paulo | D |
| F3 | Kirurgic | Campinas | A |
| F4 | Piloto | Piracicaba | A |
| F5 | Equipament | Sao Carlos | C |

Atenção: o script de criação das tabelas e inserção de dados encontra-se no final desta aula.

Exemplo 7: obtenha, **sem repetição**, o **código** de todas as **peças** que já foram utilizadas em quaisquer projetos, em **ordem crescente** do código da peça.

```
SELECT DISTINCT PeNro
FROM Fornece_para
ORDER BY PeNro ASC;
```

Resultado da consulta (Exemplo 7) e alguns comentários:

| PeNro |
|-------|
| PE1 |
| PE2 |
| PE3 |
| PE4 |
| PE5 |

- **DISTINCT** retorna o resultado da consulta, **eliminando as linhas duplicadas**, ou seja, se mais de uma linha do resultado da consulta contém valores iguais, ele só trará uma linha com estes valores;
- **ORDER BY** traz o resultado da consulta ordenado e, como foi utilizado **ASC**, a ordem é crescente.

Exemplo 8: obtenha o nome e a duração, em **DIAS**, de cada projeto.

```
SELECT PNome, (PDuracao * 30) AS Dias
FROM Projeto;
```

Resultado da consulta (Exemplo 8) e alguns comentários:

| PNome | Dias |
|---------|------|
| Detroit | 150 |
| Pegasus | 90 |
| Alfa | 60 |
| Sea | 90 |
| Paraíso | 30 |

- observe que é possível utilizar os **operadores aritméticos (+, -, *, /)** em uma consulta, formatando o resultado da consulta da maneira que desejar;
- o **AS** permite alterar o nome de um atributo/ expressão no resultado da consulta.

Exemplo 9: obtenha, em **ordem crescente** de preço, o nome das peças de **cor vermelha OU amarela E** com **preço de 9, 18, 22, 40 ou 90**

```
SELECT PeNome
FROM Peca
WHERE ((PeCor = 'Vermelho') OR (PeCor = 'Amarelo'))
AND (PePreco IN (09, 18, 22, 40, 90))
ORDER BY PePreco ASC;
```

Resultado da consulta (Exemplo 9) e alguns comentários:

| PeNome |
|----------|
| Limpador |
| Volante |

- observe que foram utilizados os operadores lógicos **OR** e **AND**;
- foi utilizado, também, o operador **IN** (que é igualdade para um conjunto de valores). O **NOT IN** é a **negação** do IN.

Exemplo 10: obtenha o nome dos fornecedores residentes em cidades iniciadas com a letra 'S'.

```
SELECT FNome
FROM Fornecedor
WHERE (FCidade LIKE 'S%');
```

Resultado da consulta (Exemplo 10) e alguns comentários:

| FNome |
|------------|
| CM |
| Equipament |

LIKE permite a comparação de partes de uma cadeia de caracteres;

- **%** representa nenhum ou vários caracteres;
- outros exemplos com o **LIKE**:

✓ **'%a'** retornaria todas as cidades que têm como último caractere a letra "a";

✓ **'%os%'** retornaria todas as cidades que têm as letras "os", não importando o que vem antes nem depois.

Exemplo 11: obtenha o **nome dos projetos** com **custo entre 20000 e 30000** (inclusive estes valores).

```
SELECT PNome  
FROM Projeto  
WHERE (PCusto BETWEEN 20000 AND 30000);
```

Resultado da consulta (Exemplo 11):

| PNome |
|-------|
| Alfa |
| Sea |

Exemplo 12: obtenha o **nome dos projetos** que estão sem valor para duração, ou seja, a **duração está nula**.

```
SELECT PNome  
FROM Projeto  
WHERE (PDuracao IS NULL);
```

Comentários sobre a consulta (Exemplo 12):

- ela não retornou nenhum dado, pois todos os projetos têm valor para duração;
- se fosse para obter os projetos que têm algum valor para o atributo duração, deveria utilizar **IS NOT NULL**.

Atenção: não deixe de assistir à videoaula “SELECT Básico”, da professora Elisângela Botelho Gracias, com uma explicação simples e breve sobre como elaborar consultas simples.

7.1. SELECT com Funções Agregadas

Podemos utilizar as seguintes funções agregadas em um SELECT:

- **AVG**: obtém o valor médio de um atributo;
- **COUNT**: obtém o número de linhas analisadas;
- **MAX**: obtém o maior valor de um atributo;
- **MIN**: obtém o menor valor de um atributo;
- **SUM**: obtém a soma dos valores de um atributo.

O **COUNT** pode ser utilizado de três formas:

- **COUNT(*)**: devolve o número de linhas que resulta de um SELECT;
- **COUNT(Atributo)**: devolve o número de linhas em que esse atributo não é NULL;
- **COUNT(DISTINCT Atributo)**: devolve o número de linhas, sem repetição, desse atributo.

Atenção: observe, nos exemplos a seguir, que serão utilizadas as funções agregadas e tudo que vimos anteriormente.

Exemplo 13: obtenha a **média dos custos dos projetos** que têm **duração maior ou igual a três meses**.

```
SELECT AVG(PCusto) AS Media_Custo  
FROM Projeto  
WHERE (PDuracao >= 3);
```

| Media_Custo |
|-------------|
| 33733.333 |

Exemplo 14: obtenha a **quantidade de fornecedores** que pertencem a **cidades iniciadas com a letra 'S'**.

```
SELECT COUNT(FNro) AS Total_Fornec  
FROM Fornecedor  
WHERE (FCidade LIKE 'S%');
```

| Total_Fornec |
|--------------|
| 2 |

Exemplo 15: obtenha o **valor mínimo** e **máximo** de **custo** de um **projeto**.

```
SELECT MIN(PCusto) AS Min_Custo,  
       MAX(PCusto) AS Max_Custo  
FROM Projeto;
```

| Min_Custo | Max_Custo |
|-----------|-----------|
| 17000 | 43000 |

7.2. SELECT com GROUP BY

Quando se deseja **aplicar as funções agregadas a vários grupos** de uma tabela (ou várias), deve-se utilizar o agrupamento – **GROUP BY**.

Neste caso, é necessário **particionar a tabela em grupos que possuem o mesmo valor de atributo**. A cláusula **GROUP BY** especifica o(s) **atributo(s) de agrupamento**. Para **cada grupo**, normalmente, especifica-se a **função agregada** (ou as funções agregadas) desejada.

Exemplo prático: se você precisa fazer uma consulta para obter a média de idade dos alunos de cada curso da universidade, deve utilizar o **GROUP BY**, sendo que o **grupo é o atributo curso**, e a informação de **cada curso é a média de idade** dos alunos. Neste caso, para **cada valor do atributo curso**, é criado **um grupo**, e sobre **cada grupo é calculada a média de idade**.

Exemplo 16: obtenha o **número de cada peça** e a **quantidade total de cada peça** utilizada em todos os projetos, em ordem crescente do número da peça.

Antes de mostrar a consulta em SQL, você deverá entender os grupos que serão criados, conforme exemplificado a seguir.

| FORNECE_PARA | | | | |
|--------------|-------|------|------|-------|
| | PENRO | FNRO | PNRO | QUANT |
| 1º GRUPO | PE1 | F5 | P4 | 5 |
| 2º GRUPO | PE2 | F2 | P2 | 1 |
| 3º GRUPO | PE3 | F3 | P4 | 2 |
| 4º GRUPO | PE4 | F4 | P5 | 3 |
| 5º GRUPO | PE5 | F1 | P1 | 1 |
| 2º GRUPO | PE2 | F2 | P3 | 1 |
| 4º GRUPO | PE4 | F3 | P5 | 2 |


```
SELECT PeNro, SUM(Quant) AS Soma
FROM Fornece_para
GROUP BY PeNro
ORDER BY PeNro ASC;
```

Resultado da consulta (Exemplo 16) e alguns comentários:

| PeNro | Soma |
|-------|------|
| PE2 | 2 |
| PE3 | 2 |
| PE5 | 1 |

- observe que o atributo utilizado para **agrupar** foi **PeNro**. Logo, para cada valor diferente de PeNro, foi criado um grupo;
- para cada grupo foi calculada a quantidade total – **SUM** – utilizada de cada peça.

Exemplo 17: obtenha o **número de cada peça** e a **quantidade total de cada peça** utilizada em todos os projetos, desde que esse **total seja menor que três**. Retorne, primeiro, em ordem decrescente este total e, depois, em ordem crescente do número da peça.

| PeNro | Soma |
|-------|------|
| PE2 | 2 |
| PE3 | 2 |
| PE5 | 1 |

- observe que esta consulta é muito semelhante à do exemplo 16, tendo somente uma condição do agrupamento (SUM(Quant) > 3);
- a cláusula **HAVING** é a condição de um agrupamento, portanto, ela só existe se a consulta tiver a cláusula **GROUP BY**;
- lembre-se de que a cláusula **WHERE** de uma consulta é **condição de cada linha da tabela**, e **não do agrupamento**, portanto, em uma consulta com agrupamento, é possível ter condições de cada linha da tabela – **WHERE** – e condições de um agrupamento – **HAVING**.

Exemplo 18: obtenha somente o **nome das cidades** que **têm apenas um único fornecedor**, em ordem crescente do nome da cidade.

| FCidade |
|------------|
| Piracicaba |
| São Carlos |
| São Paulo |

- observe que foi feito o **agrupamento** utilizando o atributo **FCidade**, logo, temos **quatro grupos**, conforme ilustrado a seguir.

| | FNRO | FNOME | FCIDADE | FCATEGORIA |
|----------|------|------------|------------|------------|
| 1º GRUPO | F1 | Plastec | Campinas | B |
| 2º GRUPO | F2 | CM | Sao Paulo | D |
| 1º GRUPO | F3 | Kirurgic | Campinas | A |
| 3º GRUPO | F4 | Piloto | Piracicaba | A |
| 4º GRUPO | F5 | Equipament | Sao Carlos | C |

- como foi solicitado as cidades com apenas um único fornecedor, utilizou-se a cláusula **HAVING**;
- a quantidade de fornecedores, por cidade, não aparece no resultado da consulta, pois foi **solicitado somente o nome das cidades**.

Exemplo 19: obtenha a **quantidade de fornecedores de cada peça em cada projeto**, em ordem decrescente desta quantidade e, depois, em ordem crescente do projeto.

```
SELECT PeNro, PNro, COUNT(FNro) AS Total
FROM Fornece_para
GROUP BY PeNro, PNro
ORDER BY COUNT(FNro) DESC, PNro ASC;
```

Resultado da consulta (Exemplo 19) e um comentário:

| PeNro | PNro | Total |
|-------|------|-------|
| PE4 | P5 | 2 |
| PE5 | P1 | 1 |
| PE2 | P2 | 1 |
| PE2 | P3 | 1 |
| PE1 | P4 | 1 |
| PE3 | P4 | 1 |

- observe que foi feito o **agrupamento** utilizando dois atributos – **PeNro** e **PNro**.

Exemplo prático: podemos agrupar os alunos de uma universidade por curso e turma para saber a média de idade dos alunos de cada turma de cada curso da universidade. Neste caso, serão utilizados os atributos curso e turma no **GROUP BY**.

Atenção: não deixe de assistir à videoaula “SELECT com agrupamento de dados”, da professora Elisângela Botelho Gracias, com uma explicação simples e breve sobre como criar consultas utilizando o GROUP BY.

Importante: o script de criação do banco de dados utilizado para o comando SELECT está disponível, a seguir.

-- Script de criação do banco de dados utilizado no SELECT

```
CREATE TABLE Peca (
```

```
PeNro CHAR(4),
```

```
PeNome VARCHAR(30) NOT NULL,
```

```
PePreco INTEGER NOT NULL,
```

```
PeCor VARCHAR(20) NOT NULL,  
  
PRIMARY KEY(PeNro));
```

```
CREATE TABLE Fornecedor (  
  
FNro CHAR(4),  
  
FNome VARCHAR(30) NOT NULL,  
  
FCidade VARCHAR(30) NOT NULL,  
  
FCategoria CHAR(1) NOT NULL,  
  
PRIMARY KEY(FNro));
```

```
CREATE TABLE Projeto (  
  
PNro CHAR(4),  
  
PNome VARCHAR(30) NOT NULL,  
  
PDuracao INTEGER NOT NULL,  
  
PCusto INTEGER NOT NULL,  
  
PRIMARY KEY(PNro));
```

```
CREATE TABLE Fornece_para (  
  
PeNro CHAR(4),  
  
FNro CHAR(4),
```

PNro CHAR(4),
Quant INTEGER,
PRIMARY KEY(PeNro,FNro,PNro),
FOREIGN KEY(PeNro) REFERENCES Peca(PeNro),
FOREIGN KEY(FNro) REFERENCES Fornecedor(FNro),
FOREIGN KEY(PNro) REFERENCES Projeto(PNro));

INSERT INTO Peca VALUES ('PE1', 'Cinto', 22, 'Azul');

INSERT INTO Peca VALUES ('PE2', 'Volante', 18, 'Vermelho');

INSERT INTO Peca VALUES ('PE3', 'Lanterna', 14, 'Preto');

INSERT INTO Peca VALUES ('PE4', 'Limpador', 9, 'Amarelo');

INSERT INTO Peca VALUES ('PE5', 'Painel', 43, 'Vermelho');

INSERT INTO Fornecedor VALUES ('F1', 'Plastec', 'Campinas', 'B');

INSERT INTO Fornecedor VALUES ('F2', 'CM', 'Sao Paulo', 'D');

INSERT INTO Fornecedor VALUES ('F3', 'Kirurgic', 'Campinas', 'A');

INSERT INTO Fornecedor VALUES ('F4', 'Piloto', 'Piracicaba', 'A');

INSERT INTO Fornecedor VALUES ('F5', 'Equipament', 'Sao Carlos', 'C');

```
INSERT INTO Projeto VALUES ('P1', 'Detroit', 5, 43000);
```

```
INSERT INTO Projeto VALUES ('P2', 'Pegasus', 3, 37000);
```

```
INSERT INTO Projeto VALUES ('P3', 'Alfa', 2, 26700);
```

```
INSERT INTO Projeto VALUES ('P4', 'Sea', 3, 21200);
```

```
INSERT INTO Projeto VALUES ('P5', 'Paraiso', 1, 17000);
```

```
INSERT INTO Fornece_para VALUES ('PE1', 'F5', 'P4', 5);
```

```
INSERT INTO Fornece_para VALUES ('PE2', 'F2', 'P2', 1);
```

```
INSERT INTO Fornece_para VALUES ('PE3', 'F3', 'P4', 2);
```

```
INSERT INTO Fornece_para VALUES ('PE4', 'F4', 'P5', 3);
```

```
INSERT INTO Fornece_para VALUES ('PE5', 'F1', 'P1', 1);
```

```
INSERT INTO Fornece_para VALUES ('PE2', 'F2', 'P3', 1);
```

```
INSERT INTO Fornece_para VALUES ('PE4', 'F3', 'P5', 2);
```

8. REFERÊNCIAS

- ELMASRI, R.; NAVATHE, S. *Sistemas de banco de dados*. 7. ed. São Paulo: Pearson, 2018.
- RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de gerenciamento de banco de dados*. 3. ed. Porto Alegre: AMGH, 2011.