BANCO DE DADOS III

Professor Alex Lemos

BD III

- Função
  - Implementação física de Banco de Dados no Sistema Gerenciador de Banco de Dados Relacional
- Atribuições e Responsabilidades
  - Implementar banco de dados.
- Valores e Atitudes
  - Incentivar a criatividade.
  - Desenvolver a criticidade.
  - Fortalecer a persistência e o interesse na resolução de situações-problema.

BD III

- Competências
  - Construir banco de dados relacional utilizando o Sistema Gerenciador de banco de dados.
- Habilidades
  - Utilizar sistema de gerenciamento para banco de dados.
  - Aplicar linguagem SQL na construção de tabelas.

BASES TECNOLÓGICAS

- Estruturas básicas dos blocos SQL:
  - Apresentação dos blocos:
    - blocos anônimos, procedures, functions e triggers.
    - Instruções SQL dentro do bloco: select, insert, update, delete.
    - Estruturas de decisão: apresentação das estruturas de decisão if-then-else e case.
    - Estruturas de repetição: apresentação das estruturas de repetição loop, while e for.
    - Exceções (tratamentos de erros): apresentação das exceções pré-definidas e exceções definidas pelo usuário.

BASES TECNOLÓGICAS

- Procedures: apresentação da sintaxe para criação de procedures e exemplos.
- Functions: apresentação da sintaxe para criação de funções e exemplos.
- Triggers: apresentação da sintaxe para criação de gatilhos e exemplos.
- Views: apresentação da sintaxe para criação de visão controlada e limitada de dados e exemplos.
- Índices.
- Merge Introdução: mesma base e base diferentes.
- Acesso, regras, proprietários e permissões.

BANCO DE DADOS III

Professor Alex Lemos

REVISÃO DOS COMANDOS DE CRUD

- Comando **INSERT**:

```
INSERT INTO gravadora (nome_gravadora, codigo_gravadora, endereco_gravadora, telefone_gravadora)
VALUES ('Outra Gravadora de Teste', 103, 'Rua Boa Morada', '55555555');
```

Onde:

INSERT INTO = Comando para realizar um cadastro de um registro na tabela gravadora.

(nome_gravadora, codigo_gravadora, endereco_gravadora, telefone_gravadora) = Lista de atributos da tabela gravadora.

VALUES = Lista de valores para os atributos listados acima;

REVISÃO DOS COMANDOS DE CRUD

- Comando **SELECT**:

```
1) - SELECT * FROM gravadora;
2) - SELECT codigo_gravadora, nome_gravadora FROM gravadora;
3) - SELECT codigo_gravadora, nome_gravadora FROM gravadora WHERE codigo_gravadora = 1;
```

Onde:

O Comando 1, pesquisa todas as informações da tabela gravadora;

O Comando 2, pesquisa o código e o nome da tabela gravadora;

O Comando 3, pesquisa o código e o nome da tabela gravadora e filtra pelo código da gravadora igual a um;

REVISÃO DOS COMANDOS DE CRUD

- Comando **UPDATE**:

```
1) – UPDATE gravadora SET nome_gravadora = “Novo nome da Gravadora” WHERE codigo_gravadora = 1;
2) - UPDATE gravadora
        SET nome_gravadora = “Novo nome da Gravadora”, telefone_gravadora = “55555555”
      WHERE codigo_gravadora = 1;
3) - UPDATE gravadora
        SET nome_gravadora = “Novo nome da Gravadora”, telefone_gravadora = “55555555”, contato_gravadora = “Alex”
      WHERE codigo_gravadora = 1;
```

Onde:

O Comando 1, atualiza o nome da gravadora para “Novo nome da Gravadora” que possui o código da gravadora igual a um;

O Comando 2, atualiza o nome da gravadora para “Novo nome da Gravadora” e o telefone da gravadora para “55555555” que possui o código da gravadora igual a um;

O Comando 3, atualiza o nome da gravadora para “Novo nome da Gravadora”, telefone da gravadora para “55555555” e o contato da gravadora para “Alex” que possui o código da gravadora igual a um;

REVISÃO DOS COMANDOS DE CRUD

- Comando **DELETE**:

```
1) – DELETE FROM gravadora WHERE codigo_gravadora = 1;

2) – DELETE FROM gravadora WHERE codigo_gravadora = 2;
```

Onde:

O Comando 1, remove as informações da gravadora que possui o código da gravadora igual a um;

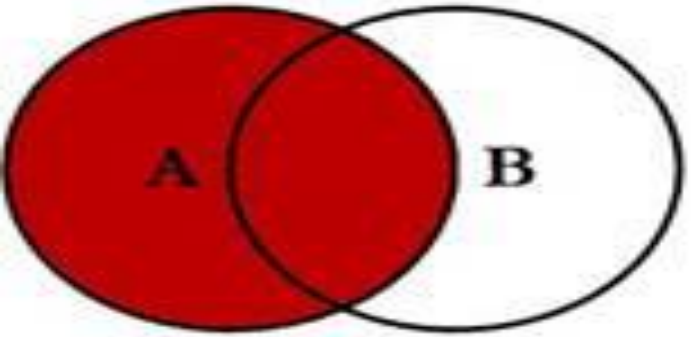
O Comando 2, remove as informações da gravadora que possui o código da gravadora igual a dois;

BANCO DE DADOS III

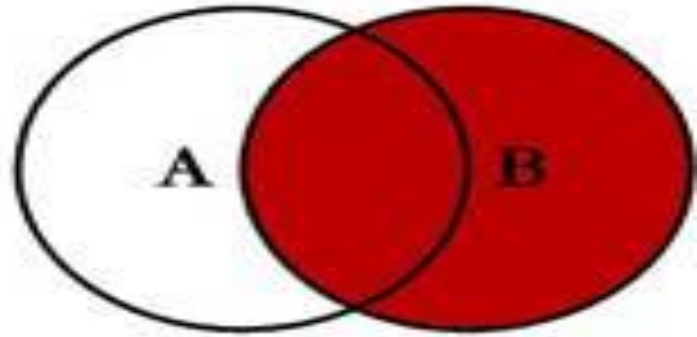
Professor Alex Lemos

- Utilização de operadores.sql

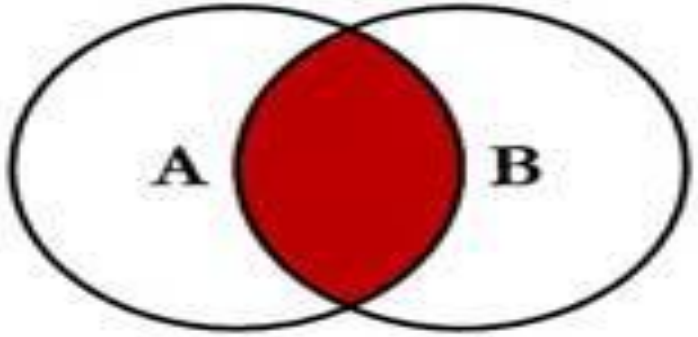
SQL JOINS



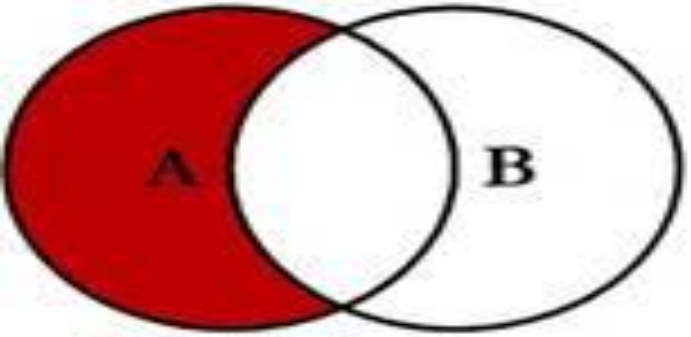

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



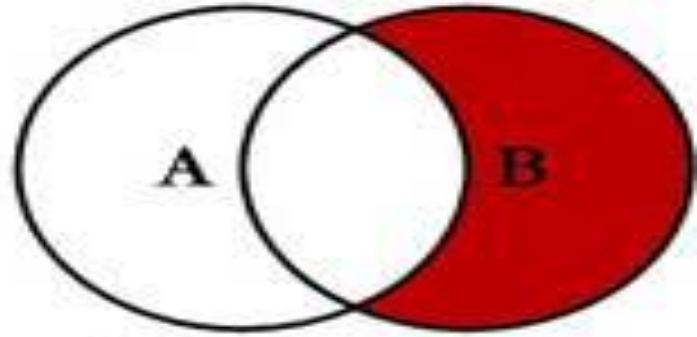

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

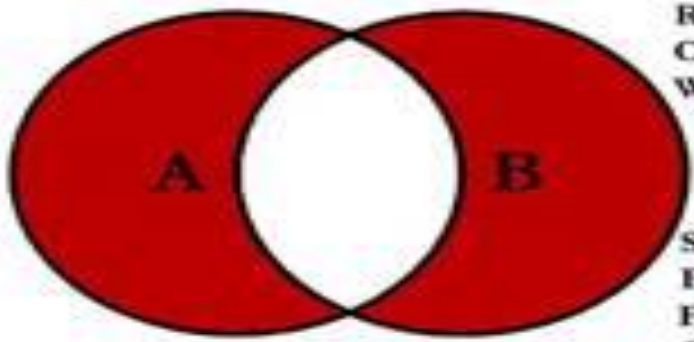



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```


```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

- Funções_Mysql.sql

AGRUPAMENTOS DE DADOS COM A CLÁUSULA GROUP BY

- Agrupamentos de Dados com a Cláusula **GROUP BY**;

✓ SELECT → CAMPOS/COLUNAS
FROM → TABELA
WHERE → CONDIÇÃO
GROUP BY → AGRUPAMENTO

GROUP BY

- Veremos como agrupar dados com **GROUP BY**, tornando mais rápida e fácil a tarefa de totalizar um conjunto de dados.

- Veremos também o uso de outras cláusulas em conjunto com o **GROUP BY**.

GROUP BY

- **GROUP BY –** Com o uso do **GROUP BY**, é possível agruparmos diversos registros com base em uma ou mais colunas de uma tabela.

- Por exemplo, os produtos de uma loja podem ser **agrupados** pelo **preço** (maior valor, menor valor), pelo **tipo** (livros, filmes, séries), etc.

- É isso que muitos **Sistemas/Relatórios**, **Sites de compras** na Internet fazem, por exemplo, em site de compras geralmente possuem diversas opções para o usuário customizar sua escolha na hora de comprar um determinado produto, geralmente separados por categorias.

GROUP BY

- É importante ressaltar que a cláusula GROUP BY é comumente usada em conjunto com as seguintes funções:

  - SUM() – Função que permite a soma dos registros;
  - AVG() – Função que permite o cálculo da média dos valores;
  - COUNT() – Função usada para contar os registros;
  - MAX() – Função que retorna o maior valor de um conjunto de valores;
  - MIN() – Função que retorna o menor valor de um conjunto de valores;

GROUP BY

- A cláusula GROUP BY é responsável por determinar em quais grupos devem ser colocadas as linhas de saída.
- Caso a cláusula SELECT contenha funções agregadas, a cláusula GROUP BY realiza um cálculo a fim de chegar ao valor sumário para cada um dos grupos.

GROUP BY

- Nas situações em que se especifica a cláusula GROUP BY, deve ocorrer uma das seguintes situações:
  - Ou a expressão **GROUP BY** deve ser correspondente à expressão da lista de seleção ou cada uma das colunas
  - Presentes em uma expressão não agregada na lista de seleção deve ser adicionada à lista de **GROUP BY**.

GROUP BY

- Quando usamos o **GROUP BY**, mais não empregamos o uso do **ORDER BY**, o resultado obtido são os grupos em ordem aleatória, lembrando que é essencial o uso do ORDER BY para determinar a ordem de apresentação dos dados.

GROUP BY

- Para compreendermos melhor o uso do **GROUP BY**, considere a tabela Produtos (lembrando que os valores associados aos nomes são fictícios e são apenas usados para este exemplo):

TABELA DE PRODUTOS

Id	Nome	Fabricante	Quantidade	VlUnitario	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

GROUP BY

- Vamos supor que desejemos obter o número de produtos em estoque, agrupados pelo tipo, para que depois seja feita a soma da quantidade existente em cada um dos grupos.

- Para isso usamos a função **SUM()** em conjunto com o **GROUP BY**, como a instrução a seguir nos mostra:

COMANDO DE SELECT

```
SELECT Tipo,
       SUM(Quantidade) AS 'Quantidade em Estoque'
  FROM Produtos
 GROUP BY
       Tipo;
```

RESULTADO

✓Executando a instrução acima temos o seguinte resultado:

Tipo	Quantidade em Estoque
Armário	50.00
Celular	300.00
Console	800.00
Notebook	200.00
Refrigerador	200.00
Smartphone	50.00
Sofá	200.00

GROUP BY

- Podemos também contar o número de produtos em estoque de acordo com os fabricantes disponíveis.

- Assim como no exemplo anterior, mais agora levando em conta os fabricantes, os produtos devem ser agrupados por eles, para que depois sejam contabilizados os produtos em relação a essa divisão. Para isso, devemos usar a seguinte instrução:

COMANDO DE SELECT

```
SELECT Fabricante,
       SUM(Quantidade) AS 'Quantidade em Estoque'
 FROM Produtos
 GROUP BY
       Fabricante
```

RESULTADO

✓Assim temos o seguinte resultado:

Fabricante	Quantidade em Estoque
Apple	50.00
Aracaju	50.00
CCE	200.00
Coréia	200.00
Dell	200.00
Microsoft	350.00
Nintendo	250.00
Samsung	300.00
Sony	200.00

GROUP BY

- Agora vamos somar a quantidade de produtos em estoque de acordo com os tipos e fabricantes disponíveis.

```
SELECT Tipo,
       Fabricante,
       SUM(Quantidade) AS 'Quantidade em
Estoque'
  FROM Produtos
GROUP BY
       Tipo,
       Fabricante
```

RESULTADO

✓Assim teremos este resultado:

Tipo	Fabricante	Quantidade em Estoque
Smartphone	Apple	50.00
Armário	Aracaju	50.00
Refrigerador	CCE	200.00
Sofá	Coréia	200.00
Notebook	Dell	200.00
Console	Microsoft	350.00
Console	Nintendo	250.00
Celular	Samsung	300.00
Console	Sony	200.00

GROUP BY

- Podemos também obter o valor total dos produtos em estoque, agrupados por tipo. Veja como:

```
SELECT Tipo,
       SUM(Quantidade * VlUnitario) AS 'Valor do Estoque'
  FROM Produtos
GROUP BY
       Tipo
```

RESULTADO

✓Temos o seguinte resultado:

Tipo	Valor do Estoque
Armário	6450.0000
Celular	149700.0000
Console	944200.0000
Notebook	379800.0000
Refrigerador	299800.0000
Smartphone	74950.0000
Sofá	99800.0000

BANCO DE DADOS II

Professor Alex Lemos

HAVING

- A cláusula HAVING determina uma condição de busca para um grupo ou um conjunto de registros, definindo critérios para limitar os resultados obtidos a partir do agrupamento de registros.
- É importante lembrar que essa cláusula só pode ser usada em parceria com GROUP BY.

HAVING COM GROUP BY

- A cláusula GROUP BY pode ser empregada, entre outras finalidades, para agrupar os produtos de acordo com cada tipo existente.
- Dentro de cada um dos grupos, a cláusula HAVING pode ser usada para restringir apenas os registros que possuem uma quantidade superior a 200 unidades no estoque, por exemplo.

HAVING COM GROUP BY

- Para fazermos o filtro conforme especificado acima, devemos usar a seguinte instrução:

```
SELECT Tipo
     , SUM(Quantidade) AS 'Quantidade em Estoque'
  FROM Produtos
  GROUP BY Tipo
 HAVING SUM(Quantidade) > 200
```

RESULTADO

- Assim teremos o seguinte resultado:

Tipo	Quantidade em Estoque
Celular	300.00
Console	800.00

- **Perceba que este tipo de filtragem nos retornou apenas 2 registros, que são os tipos que, somados, ultrapassam o valor de 200 unidades em estoque.**

HAVING COM GROUP BY

- Outro exemplo: vamos agrupar os produtos com base nos tipos e fabricantes disponíveis. Logo após, retornaremos apenas os registros cuja quantidade supera novamente as 200 unidades em estoque. Veja a instrução e o resultado a seguir:

HAVING COM GROUP BY

```
SELECT Tipo
     , Fabricante
       , SUM(Quantidade) AS 'Quantidade em Estoque'
  FROM Produtos
 GROUP BY Tipo, Fabricante
HAVING SUM(Quantidade) > 200
```

RESULTADO

- Assim teremos o seguinte resultado:

Tipo	Fabricante	Quantidade em Estoque
Console	Microsoft	350.00
Console	Nintendo	250.00
Celular	Samsung	300.00

HAVING COM GROUP BY

- Vamos agora fazer um exemplo mais complexo:
- Vamos supor que o agrupamento deverá ser feito pelo Nome. Dentro deste agrupamento, desejamos obter apenas aqueles cuja quantidade novamente supera as 200 unidades em estoque e cujo valor estocado seja igual ou superior a 100 mil.

HAVING COM GROUP BY

```
SELECT Nome
     , SUM(Quantidade) AS 'Quantidade em Estoque'
       , SUM(Quantidade * VlUnitario) AS 'Valor em Estoque'
  FROM Produtos
 GROUP BY Nome
HAVING SUM(Quantidade) > 200
AND SUM (Quantidade * VlUnitario) >= 10000.00
```

RESULTADO

- Assim teremos o seguinte resultado:

Nome	Quantidade em Estoque	Valor em Estoque
GT-I6220 Quad Band	300.00	149700.0000
Wii 120GB	250.00	249750.0000
Xbox 360 120GB	350.00	454650.0000

BANCO DE DADOS III

Professor Alex Lemos

SUBQUERY OU SUBCONSULTA

- Subquery forma alternativa de expressar consultas envolvendo relacionamentos entre tabelas.
- Subquery é um comando SELECT que foi "embutido" em outro comando SELECT, UPDATE, DELETE ou dentro de outra subquery.
- A finalidade da subquery é retornar um conjunto de linhas para a query principal
- Operadores utilizados para tratamento de subconsultas:
  - IN, NOT IN, EXISTS, NOT EXISTS

TIPOS DE SUBCONSULTA

- ***Subconsulta por valor***
- Neste tipo de Subconsulta os dados (ResultSet) retorno um conjunto de valores para a query principal utilizado em conjunto com os operadores IN, NOT IN.

```
select *
  from venda_item
 where cd_produto in (select cd_produto
                        from produto
                       where nome_produto=“caneta”)
```

TIPOS DE SUBCONSULTA

- ***Subconsulta como operadores EXISTS, NOT EXISTS.***
- ***Utilizando uma subquery em conjunto com a cláusula EXISTS().***

```
select *
  from venda_item vi
 where exists (select cd_produto
                 from produto p
                where p.cd_produto = vi.cd_produto
                  and p.nome_produto = 'caneta')
```

TIPOS DE SUBCONSULTA

- ***Subconsulta utilizada na cláusula do FROM***, onde a Subconsulta (ResultSet) será trabalhado como uma tabela física e seu campos podendo ser utilizados na select principal.

```
select v.id_compra, v.data_compra, vi.total_total_compra
  from (select id_venda_item, sum(valor)
          from item
        group by id) vi
        inner join venda v on (vi.id_venda_item = v.id_compra)
where v.data_compra between '01-01-2018' and current_date()
order by 2
```

TIPOS DE SUBCONSULTA

- ***Subquery como atributo***
- Quando precisamos utilizar as funções de agregações que resultam em dados de outra tabela, para cada registro da consulta SELECT

```
select gravadora.codigo_gravadora
      ,gravadora.nome_gravadora
      ,(select count(*)
          from cd
         where cd.codigo_gravadora = gravadora.codigo_gravadora)
  from gravadora;
```

BANCO DE DADOS III

Professor Alex Lemos

PROCEDURE

- Procedure são rotinas(procedimentos) armazenadas no banco de dados que executam um conjunto definido de passos sequenciais. O conceito é análogo a criação de um método em uma linguagem de programação, onde você pode chamar este método(function) diversas vezes sem necessidade de reescrever código.

SINTAXE DE CRIAÇÃO DE PROCEDURE NO MYSQL

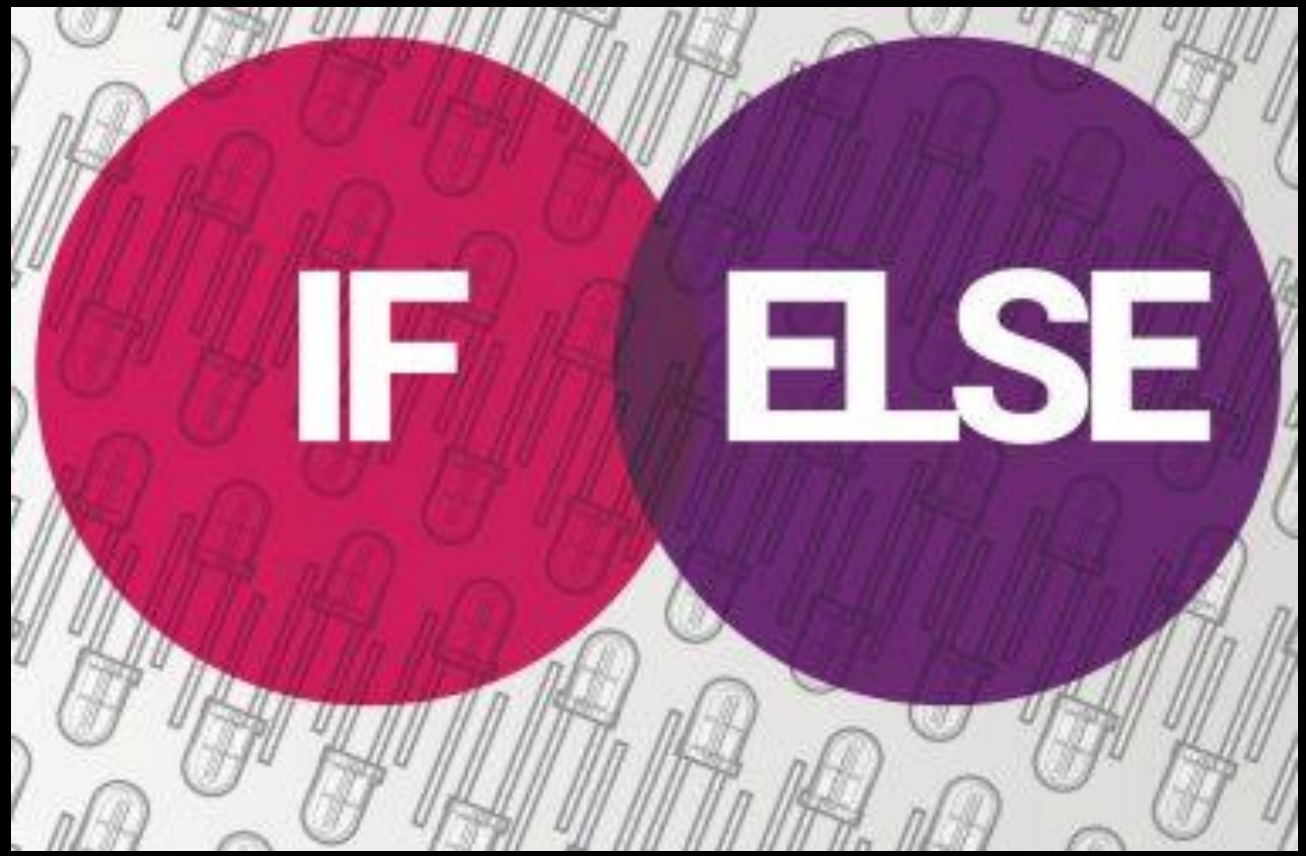
```
#Criando Procedure
  DELIMITER $
  CREATE PROCEDURE sp_teste()
  BEGIN
        SELECT "Olá Mundo!";
  END
  $

  #Chamando a Procedure
  CALL sp_teste();

  #Resultado
  Olá Mundo
```

BANCO DE DADOS III

Professor Alex Lemos

ESTRUTURA DE DECISÃO

- Simples;
- Composta;
- Encadeada;



ESTRUTURA DE DESIÇÃO

SIMPLES

```
IF (condição) THEN
        comando 1;
        comando 2;
        ...
        comando N;
END IF;
```

COMPOSTA

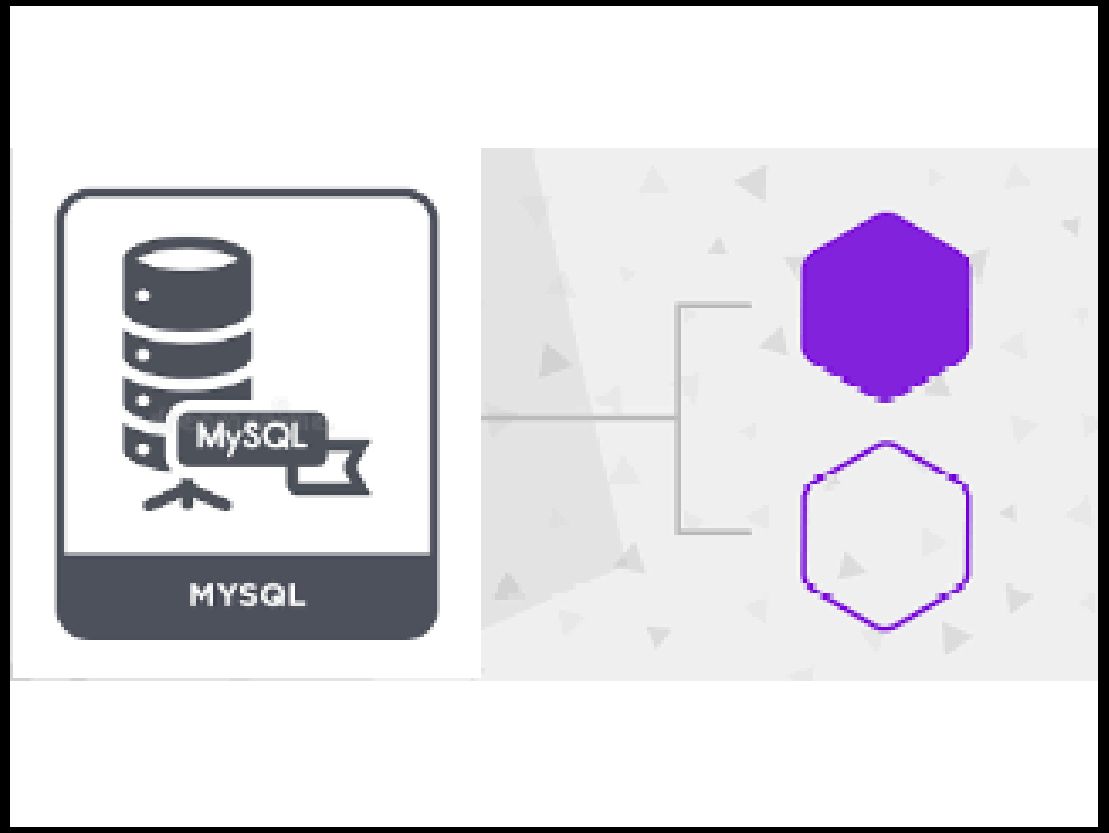
```
IF (condição) THEN
        comando 1;
        comando 2;
        ...
        comando N;
ELSE
        Comando 3;
        Comando 4;
        ...
        Comando N;
END IF;
```

ENCADEADA

```
IF (condição 1) THEN
   IF (condição 2) THEN
      comando 1;
      ...
      comando N;
   END IF;
ELSE
   IF (condição 3) THEN
      Comando 3;
      ...
      Comando N;
    ELSE
       Comando X;
    END IF;
END IF;
```

ESTRUTURA ESCOLHA

- CASE
- WHEN
- THEN



CASE

```
CASE WHEN 10 THEN RETURN 'dez';
        WHEN 20 THEN RETURN 'vinte';
        ELSE RETURN 'outro valor';
END CASE;

CASE valor
        WHEN 10 THEN RETURN 'dez';
        WHEN 20 THEN RETURN 'vinte';
        ELSE RETURN 'outro valor';
 END CASE;
```

BANCO DE DADOS III

Professor Alex Lemos

ESTRUTURA DE REPETIÇÃO

- WHILE
- REPEAT
- LOOP

WHILE

- A lista de instruções dentro do WHILE é repetida desde que a condição seja verdadeira. Uma declaração WHILE pode ser rotulada.

WHILE

```
WHILE contador <= 10 DO

   SELECT contador;

   SET contador = contador + 1;

END WHILE;
```

REPEAT

- A lista de instruções dentro do REPEAT, é repetida enquanto a condição seja verdadeira. Assim, um REPEAT sempre entra no loop pelo menos uma vez.
- Uma declaração REPEAT pode ser rotulada.

REPEAT

```
REPEAT

  SET contador = contador + 1;

  SELECT contador;

  UNTIL contador >= 10

END REPEAT;
```

LOOP

- Implementa uma construção de loop simples, permitindo a execução repetida da lista de comandos. As instruções dentro do loop são repetidas até que o loop seja finalizado. Geralmente, isso é feito com a declaração

  LEAVE

LOOP

```
loop_teste: LOOP

   SET contador = contador + 1;

   IF contador > 10 THEN
      LEAVE loop_teste;
   END IF;

   SELECT contador;

END LOOP loop_teste;
```

BANCO DE DADOS III

Professor Alex Lemos