Banco de Dados

Aulas Práticas - Laboratório 4

Profª Cristina Verçosa Pérez Barrios de Souza
cristina.souza@pucpr.br

Tópicos

- INSERT: NULL e DEFAULT
- Integridade Referencial (IR)
  - ... ON UPDATE CASCADE / RESTRICT / SET NULL
  - ... ON DELETE CASCADE / RESTRICT / SET NULL
- INSERT em VIEW
- PROGRAMAÇÃO SQL
  - VARIÁVEIS
  - IF / CASE
  - FUNÇÕES MATEMÁTICAS
  - STORED PROCEDURES
    - Recursão

Instrução de INSERT: DEFAULT e NULL

- No Insert, é possível trabalhar com valores DEFAULT e NULL
  - Precedência de NULL e DEFAULT para INSERT:
    - Se coluna omitida, ela assumirá o valor DEFAULT, se existir
    - Se não houver DEFAULT (padrão), será tentado o valor NULL

	Status da Coluna			
	Existe **DEFAULT**		Não existe **DEFAULT**	
Entrada	**Null**	**Not Null**	**Null**	**Not Null**
Coluna omitida	Default	Default	Null	Erro
Null inserido explicitamente	Null	Erro	Null	Erro
Default inserido explicitamente	Default	Default	Null	Erro

Prática 4.1: INSERT: DEFAULT e NULL

– Execute no Database **LAB_04**:

Neste exercício, a criação da **Tab_Depto** define seus campos com configuração para **NULL**, **NOT NULL** e **DEFAULT**.

RESPONDER:

1. Em b.1), como é feito o tratamento de campos omitidos?
2. Em b.2), como é feito o tratamento de campos omitidos?
3. Em b.3), como é feito o tratamento do **NULL**?
4. Em b.4), o que acontece neste **INSERT** do **NULL**?
5. Em b.5), o que acontece neste **INSERT** do **NULL**?
6. Em b.5), como ficou povoada a **Tab_Depto**?

(a)

```
CREATE DATABASE Lab_04;
USE Lab_04;

CREATE TABLE Tab_Depto
(
   ID          INT          AUTO_INCREMENT PRIMARY KEY,
   Nome        VARCHAR(60)  NOT NULL DEFAULT ('Vendas'), -- NOT NULL + DEFAULT
   Localizacao VARCHAR(60)  DEFAULT ('Bloco A'),          -- NULL + DEFAULT
   Sala        CHAR(3)      NOT NULL,                     -- NOT NULL
   Fone        VARCHAR(20)                                -- NULL
);
```

(b)

```
INSERT Tab_Depto (Sala, Fone) VALUES ('80', '(41)3021-4040'); -- b.1) tratamento de campos omitidos
INSERT Tab_Depto (Sala) VALUES ('100');                       -- b.2) tratamento de campos omitidos
INSERT Tab_Depto (Localizacao, Sala) VALUES (NULL,'200');     -- b.3) tratamento de NULL
INSERT Tab_Depto (Sala) VALUES (NULL);                        -- b.4) tratamento de NULL
INSERT Tab_Depto (Localizacao, Sala) VALUES (NULL, '300');    -- b.5) tratamento de NULL

SELECT * FROM Tab_Depto;                                       -- b.6) Tab_Depto povoada
```

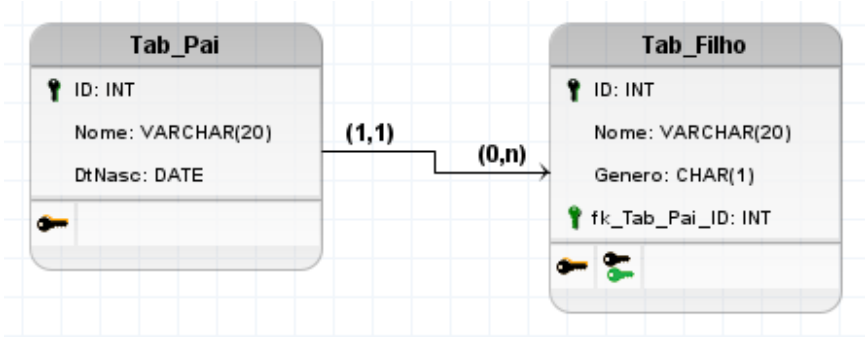
Integridade Referencial (IR)

- **Constrain Foreign Key**
  - Restrição, ou regra que garante a integridade referencial dos dados
  - Se uma regra de integridade referencial é violada, o procedimento normal é o SGBD rejeitar a operação que viola a regra.
  - Porém, é possível definir mais precisamente uma ação que será desencadeada se a regra for violada (em um DELETE ou um UPDATE)

  - Ex:

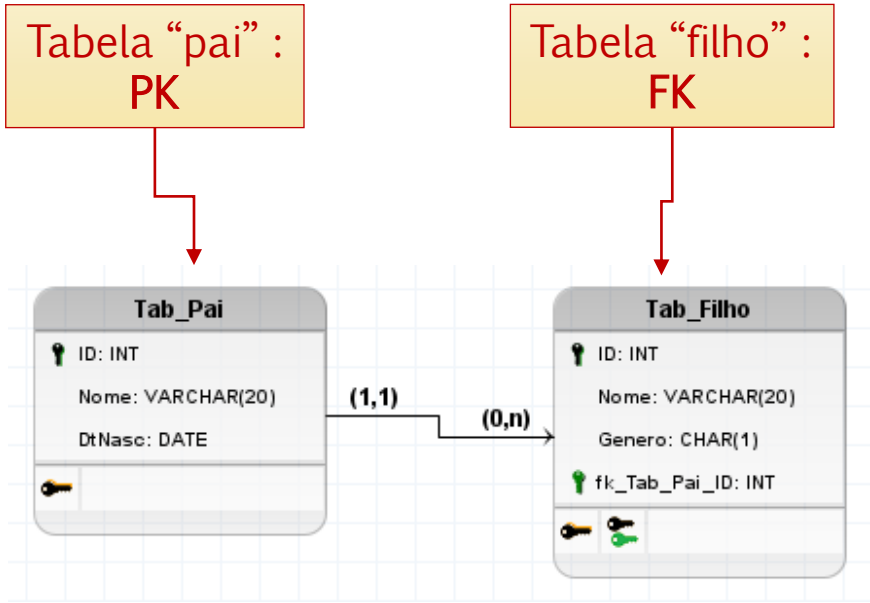
    ALTER TABLE *Tab_Filho* ADD CONSTRAINT *FK_constraint_name*
    FOREIGN KEY (*fk_Tab_Pai_ID*)
    REFERENCES *Tab_Pai (ID)*
    ON DELETE *option*
    ON UPDATE *option;*

  - Onde *option* pode ser:
    - RESTRICT | CASCADE | SET NULL



Integridade Referencial (IR)

- ON DELETE *option* / ON UPDATE *option*;
  - *option* = CASCADE:
    - exclui (ou atualiza) a linha da tabela pai e automaticamente também exclui (ou atualiza) as linhas correspondentes na tabela filho.
  - *option* = RESTRICT:
    - rejeita a operação de exclusão ou atualização da tabela pai, quando esta for referenciada como FK na tabela filho.
  - *option* = SET NULL:
    - exclui (ou atualiza) a linha da tabela pai e define como NULL a coluna de chave estrangeira na tabela filho.



Integridade Referencial (IR)

› Vamos criar e povoar as tabelas como a seguir:

Tabela “pai” : PK

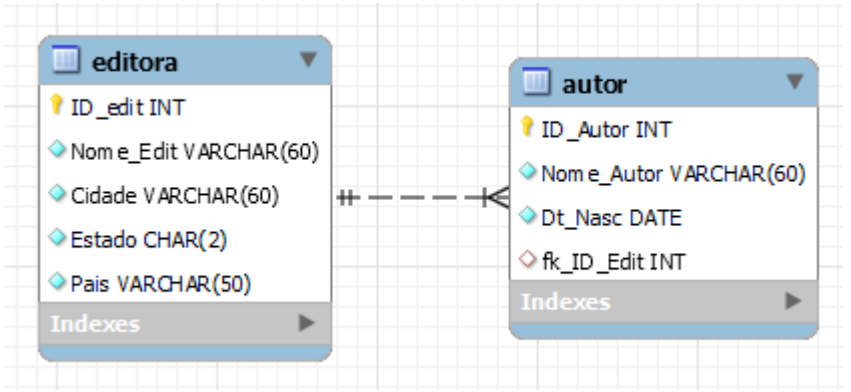


Tabela “filho” : FK

	ID_edit	Nome_Edit	Cidade	Estado	Pais
▶	1	Editora AAA	São Paulo	SP	Brasil
	2	Editora Sul	Porto Alegre	RS	Brasil
	3	LTC	São Paulo	SP	Brasil
	4	CENGAGE	Rio de Janeiro	RJ	Brasil
	5	Três Estrelas	Alagoas	CE	Brasil

	ID_Autor	Nome_Autor	Dt_Nasc	fk_ID_Edit
▶	100	José	1956-09-08	1
	101	Maria	1975-04-18	2
	102	Antônia	1954-12-10	3
	103	Armínio	1976-07-28	5
	104	Luiza	1945-11-09	5

Prática 4.2: Ações para manter a IR

- Execute no Database **LAB_04**:

(a)

```
CREATE TABLE Editora
(
   ID_edit     INT AUTO_INCREMENT PRIMARY KEY, -- Tabela PAI
   Nome_Edit   VARCHAR(60)  NOT NULL,
   Cidade      VARCHAR(60)  NOT NULL,
   Estado      CHAR(2)      NOT NULL,
   Pais        VARCHAR(50)  NOT NULL
);

INSERT Editora (Nome_Edit, Cidade, Estado, Pais) VALUES ('Editora AAA', 'São Paulo', 'SP', 'Brasil');
INSERT Editora (Nome_Edit, Cidade, Estado, Pais) VALUES ('Editora Sul', 'Porto Alegre', 'RS', 'Brasil');
INSERT Editora (Nome_Edit, Cidade, Estado, Pais) VALUES ('LTC', 'São Paulo', 'SP', 'Brasil');
INSERT Editora (Nome_Edit, Cidade, Estado, Pais) VALUES ('CENGAGE', 'Rio de Janeiro', 'RJ', 'Brasil');
INSERT Editora (Nome_Edit, Cidade, Estado, Pais) VALUES ('Três Estrelas', 'Alagoas', 'CE', 'Brasil');
```

Prática 4.2: Ações para manter a IR

```
ON UPDATE CASCADE
ON DELETE CASCADE
```

- Execute no Database **LAB_04** para avaliar o **CASCADE**:

(b)

```
CREATE TABLE Autor
(
   ID_Autor     INT AUTO_INCREMENT PRIMARY KEY, -- Tabela FILHO
   Nome_Autor   VARCHAR(60)  NOT NULL,
   Dt_Nasc      DATE  NOT NULL,
   fk_ID_Edit   INT NULL
);
ALTER TABLE Autor ADD  CONSTRAINT FK_Autor_Editora FOREIGN KEY(fk_ID_edit)
REFERENCES Editora (ID_edit)
ON UPDATE CASCADE
ON DELETE CASCADE ;

ALTER TABLE Autor AUTO_INCREMENT = 100; -- Seed = 100 (início do AUTO_INCREMENT)

INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('José', '1956-09-08', 1);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Maria', '1975-04-18', 2);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Antônia', '1954-12-10', 3);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Armínio', '1976-07-28', 5);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Luiza', '1945-11-09', 5);
```

Prática 4.2: Ações para manter a IR

```
ON UPDATE CASCADE
ON DELETE CASCADE
```

› Execute os códigos abaixo:

(c)

```
SELECT * FROM Editora;
SELECT * FROM Autor;


UPDATE Editora
   SET ID_edit = 50
   WHERE ID_edit = 5;


SELECT * FROM Editora
SELECT * FROM Autor
```

(d)

```
SELECT * FROM Editora;
SELECT * FROM Autor;


DELETE FROM Editora
WHERE ID_edit = 1;


SELECT * FROM Editora
SELECT * FROM Autor
```

RESPONDER:

1. Em **c)**, qual foi o resultado obtido nas tabelas **Editora** e **Autor**, após o **UPDATE** executado? Por que isso ocorreu?
2. Em **d)**, qual foi o resultado obtido nas tabelas **Editora** e **Autor**, após o **DELETE** executado? Por que isso ocorreu?

Prática 4.2: Ações para manter a IR

```
ON UPDATE RESTRICT
ON DELETE RESTRICT
```

– Execute no Database **LAB_04** para avaliar o **RESTRICT**:

(e)

```
UPDATE Editora
   SET ID_edit = 5
   WHERE ID_edit = 50;

-- FAÇA O DROP, E NOVAMENTE O CREATE E INSERT DE TODOS OS VALORES DA TABELA EDITORA
DROP TABLE Autor;
-- Alterando Tabela Autor
CREATE TABLE Autor -- Alterando Tabela Autor
(
   ID_Autor     INT AUTO_INCREMENT PRIMARY KEY, -- Tabela FILHO
   Nome_Autor   VARCHAR(60)  NOT NULL,
   Dt_Nasc      DATE  NOT NULL,
   fk_ID_Edit   INT NULL
);
ALTER TABLE Autor ADD  CONSTRAINT FK_Autor_Editora FOREIGN KEY(fk_ID_edit)
REFERENCES Editora (ID_edit)
ON UPDATE RESTRICT
ON DELETE RESTRICT ;

ALTER TABLE Autor AUTO_INCREMENT = 100; -- Seed = 100 (início do AUTO_INCREMENT)

INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('José', '1956-09-08', 1);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Maria', '1975-04-18', 2);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Antônia', '1954-12-10', 3);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Armínio', '1976-07-28', 5);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Luiza', '1945-11-09', 5);
```

Prática 4.2: Ações para manter a IR

```
ON UPDATE RESTRICT
ON DELETE RESTRICT
```

› Execute os códigos abaixo:

(f)

```
SELECT * FROM Editora;
SELECT * FROM Autor;


UPDATE Editora
   SET ID_edit = 50
   WHERE ID_edit = 5;


SELECT * FROM Editora
SELECT * FROM Autor
```

(g)

```
SELECT * FROM Editora;
SELECT * FROM Autor;


DELETE FROM Editora
WHERE ID_edit = 1;


SELECT * FROM Editora
SELECT * FROM Autor
```

RESPONDER:

1. Em **f)**, qual foi o resultado obtido nas tabelas **Editora** e **Autor**, após o **UPDATE** executado? Por que isso ocorreu?
2. Em **g)**, qual foi o resultado obtido nas tabelas **Editora** e **Autor**, após o **DELETE** executado? Por que isso ocorreu?

Prática 4.2: Ações para manter a IR

```
ON UPDATE SET NULL
ON DELETE SET NULL
```

- Execute no Database **LAB_04** para avaliar o **SET NULL**:

(h)

```
UPDATE Editora
   SET ID_edit = 5
   WHERE ID_edit = 50;

DROP TABLE Autor;

CREATE TABLE Autor -- Alterando Tabela Autor
(
   ID_Autor     INT AUTO_INCREMENT PRIMARY KEY, -- Tabela FILHO
   Nome_Autor   VARCHAR(60)  NOT NULL,
   Dt_Nasc      DATE  NOT NULL,
   fk_ID_Edit   INT NULL
);
ALTER TABLE Autor ADD  CONSTRAINT FK_Autor_Editora FOREIGN KEY(fk_ID_edit)
REFERENCES Editora (ID_edit)
ON UPDATE SET NULL
ON DELETE SET NULL;

ALTER TABLE Autor AUTO_INCREMENT = 100; -- Seed = 100 (início do AUTO_INCREMENT)

INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('José', '1956-09-08', 1);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Maria', '1975-04-18', 2);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Antônia', '1954-12-10', 3);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Armínio', '1976-07-28', 5);
INSERT Autor (Nome_Autor, Dt_Nasc, fk_ID_Edit) VALUES ('Luiza', '1945-11-09', 5);
```

Prática 4.2: Ações para manter a IR

```
ON UPDATE SET NULL
ON DELETE SET NULL
```

› Execute os códigos abaixo:

(i)

```
SELECT * FROM Editora;
SELECT * FROM Autor;


UPDATE Editora
   SET ID_edit = 50
   WHERE ID_edit = 5;


SELECT * FROM Editora
SELECT * FROM Autor
```

(j)

```
SELECT * FROM Editora;
SELECT * FROM Autor;


DELETE FROM Editora
WHERE ID_edit = 1;


SELECT * FROM Editora
SELECT * FROM Autor
```

RESPONDER:

1. Em **i)**, qual foi o resultado obtido nas tabelas **Editora** e **Autor**, após o **UPDATE** executado? Por que isso ocorreu?
2. Em **j)**, qual foi o resultado obtido nas tabelas **Editora** e **Autor**, após o **DELETE** executado? Por que isso ocorreu?

INSERT em VIEWS

- Modificações feitas por meio de VIEWs acabam modificando os dados em uma tabela "**base**" (e somente na tabela), pois VIEWs não armazenam dados

Prática 4.3: INSERT em VIEWS

- Execute no Database **LAB_04**:

RESPONDA:

1. Em **b)**, após o **1º. INSERT**, o que foi exibido na **VIEW**? Por que esse resultado foi apresentado?
2. Em **b)**, após o **2º. INSERT**, o que foi exibido na **VIEW**? Por que esse resultado foi apresentado?

(a)

```
CREATE TABLE Tab_Um (
ID_um    INT PRIMARY KEY NOT NULL,
col_1    CHAR(3) NOT NULL
);
CREATE TABLE Tab_Dois (
fk_ID_um INT PRIMARY KEY NOT NULL,
col_2    CHAR(3) NOT NULL,
FOREIGN KEY (FK_ID_um)
REFERENCES Tab_Um (ID_um)
);

CREATE VIEW JuntaUmDois AS (
SELECT ID_um, col_1, fk_ID_um, col_2
FROM Tab_Um JOIN Tab_Dois
ON (Tab_Um.ID_um = Tab_Dois.fk_ID_um)
);
```

(b)

```
-- 1º. INSERT
INSERT Tab_Um (ID_um, col_1) VALUES (5, 'AAA');

SELECT * FROM JuntaUmDois;
SELECT * FROM Tab_Um;
SELECT * FROM Tab_Dois;

-- 2º. INSERT
INSERT Tab_Dois(fk_ID_um, col_2) VALUES (5, 'XXX');

SELECT * FROM JuntaUmDois;
SELECT * FROM Tab_Um;
SELECT * FROM Tab_Dois;
```

Prática 4.3: INSERT em VIEWS

- Execute no Database **LAB_04**:

(c)

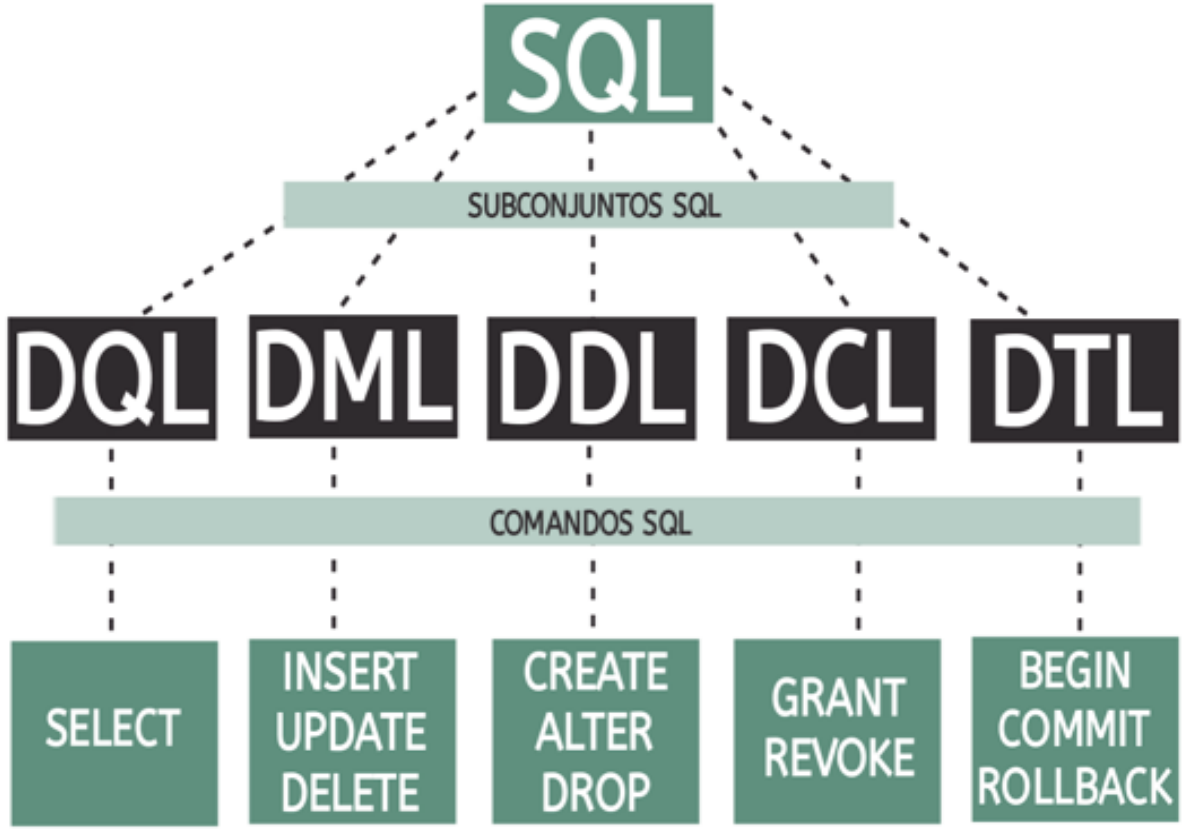
```
-- 1º. INSERT
INSERT JuntaUmDois (ID_Um, col_1)
VALUES (10, 'BBB');

SELECT * FROM JuntaUmDois;
SELECT * FROM Tab_Um;
SELECT * FROM Tab_Dois;

-- 2º. INSERT
INSERT JuntaUmDois(fk_ID_um, col_2)
VALUES (20, 'YYY');

SELECT * FROM JuntaUmDois;
SELECT * FROM Tab_Um;
SELECT * FROM Tab_Dois;
```

RESPONDA:

1. Em **c)**, após o **1º. INSERT**, o que foi exibido na **VIEW**? Por que esse resultado foi apresentado?
2. Em **c)**, o **2º. INSERT na VIEW** funcionou na **Tab_Dois**? Por que?
3. Em **c)**, como devemos alterar o **2º. INSERT**, para que ele funcione na Tab_Dois e também seja exibido na **VIEW**?
4. Após corrigir o **2º. INSERT**, mostre como ficam preenchidas as **Tab_Um**, **Tab_Dois** e **VIEW JuntaUmDois**.

PROGRAMAÇÃO SQL

- **DML**: linguagem de manipulação de dados
  - Data Manipulation Language
- **DDL**: linguagem de definição de dados
  - Data Definition Language
- **DQL**: linguagem de consulta de dados
  - Data Query Language
- **DCL**: linguagem de controle de dados
  - Data Control Language
  - Permite controlar o acesso aos dados da nossa base
- **DTL** ou **TCL**: linguagem de transação de dados
  - Data Transaction Language ou Transaction Control Language
  - Permite gerenciar transações feitas no banco



Programação SQL: Fundamentos

- **A SQL é uma linguagem de programação?**
  - **NÃO**
    - Não é alternativa para Java, C, C++,
    - Não é alternativa para ambientes de desenvolvimento (IDE)
    - Não é adequada para construir grandes aplicações
    - Não oferece **interface** com o usuário
    - Não oferece **I/O de arquivo** ou dispositivo
    - Programação simples e limitada
  - **SIM**
    - Linguagem especializada (procedural)
    - Permite **variáveis**, **condicionais**, laços de repetição, etc.
    - Adequada para trabalhar em conjunto com outras linguagens

Programação SQL: Fundamentos

- **Variáveis de Sessão**
  - Têm escopo e visibilidade limitados, existem apenas na sessão, ou conexão com o BD onde foi declarada
  - São definidas pelo programador SQL (usuário)
    - As variáveis definidas pelo usuário são específicas da sessão. Ou seja, não podem ser vistas ou usadas por outros usuários.
    - Todas as variáveis de uma sessão são liberadas automaticamente quando a sessão finaliza (usuário sai da sessão).
  - Sintaxe para declarar uma variável de sessão:

```
SET @cost = 5, @cost1 = 8.00;   -- declara variáveis de sessão
SELECT @cost1;                  -- mostra o valor da variável de sessão
SELECT  @cost;                  -- mostra o valor da variável de sessão
```

No **MySQL**, o usuário **não pode criar variáveis globais personalizadas** (de **sistema** ou de **sessão**).
Apenas pode alterar variáveis de sistema globais ou de sessão existentes

Programação SQL: Fundamentos

- **Variáveis Locais**
  - Têm escopo e visibilidade limitados, existem apenas dentro do bloco BEGIN ... END onde foi declarada

  - Sintaxe para declarar uma variável de sessão:

```
BEGIN -- inicia escopo do bloco
    DECLARE nome VARCHAR(20) DEFAULT 'Fulano';        -- declara variáveis local
    SELECT @cost1;                                    -- mostra o valor da variável local
    SELECT  @cost;                                    -- mostra o valor da variável local
END -- finaliza escopo do bloco
```

Programação SQL: Fundamentos

› Prática 4.4: Variáveis

- No database LAB_04, execute os comandos:

(a)

```
USE LAB_04;
CREATE TABLE Empresa (
     ID      INT PRIMARY KEY AUTO_INCREMENT,
     Nome    VARCHAR(20),
     Atuacao VARCHAR(50),
     Cidade  VARCHAR(20),
     Estado   VARCHAR(2)
);

INSERT Empresa (Nome, Atuacao, Cidade, Estado)  VALUES
('ACME Corp.','Cartoons','São Paulo','SP'),
('Estrela Ltda.', 'Transporte passageiros','Campinas','SP'),
('Aurora','Panificadora','Belo Horizonte','MG'),
('Azul','Aviação','São Paulo','SP'),
('Leão Ltda.','Bebidas','Curitiba','PR'),
('Petit S.A.','Queijos e frios', 'Uberlândia','MG'),
('Barreados Corp.','Alimentos congelados','Morretes','PR');

SELECT * FROM Empresa;
```

(b)

```
CREATE TABLE Estoque (
ID INT PRIMARY KEY AUTO_INCREMENT,
Nome    VARCHAR(20),
Qtde    INT DEFAULT 10,
ValUnit DECIMAL(10,2)
);

INSERT Estoque (Nome, Qtde, ValUnit)
VALUES
('caderno',200,15.00),
('borracha',50,6.50),
('caneta',300, 5.50),
('régua 30cm',80, 10.00),
('lápis',500, 4.00),
('bloco A4',35, 18.45);

SELECT * FROM  Estoque;
```

Programação SQL: Fundamentos

› Prática 4.4: Variáveis

- Na database LAB_04, execute:

RESPONDA:

1. Nas linhas **1.** e **2.**, o que acontece se não inicializarmos as variáveis?

2. É preciso **executar de uma única vez** todos os comandos do script apresentado, para exibir o conteúdo final das variáveis de sessão do exemplo? Por que?

(c)

```
 1.  SET @nome_produto   = 'none'; -- Declara e inicializa variáveis de sessão
 2.  SET @total_produtos = -1;
 3.
 4.  SELECT nome INTO @nome_produto
 5.  FROM   Estoque
 6.  WHERE  ID = 3;
 7.
 8.  SELECT COUNT(*) INTO @total_produtos
 9.  FROM   Estoque;
10.
11.  SELECT @nome_produto AS 'Produto com ID = 3';
12.  SELECT @total_produtos AS 'Total de Produtos Cadastrados';
13.
```

Programação SQL: Fundamentos

RESPONDA:

1. Em que linhas do código estão os **delimitadores do bloco de comandos da procedure**?
2. O que é feito nas linhas **5.** e **6.**?
3. Quais os delimitadores do **laço** de repetição **WHILE**?
4. O que é feito nas linhas **8.**, **.9** e **.15**?

› Prática 4.4: Variáveis

- Na database LAB_04, execute:

(d)

```
1.  -- Altera delimitador de comando para não conflitar com a SP
2.  DELIMITER $$
3.  CREATE PROCEDURE proc_demo1() -- Cria procedure proc_demo1(), sem parâmetros
4.  BEGIN
5.          DECLARE i INT DEFAULT 0;     -- Declara e inicializa variáveis locais
6.          DECLARE output VARCHAR(100) DEFAULT 'Saída = ';
7.          WHILE i < 10 DO
8.             SET output = CONCAT(output, i , ', ');
9.             SET i = i + 1;
10.         END WHILE;
11.         SELECT output;
12. END $$
13. DELIMITER ; -- Retorna ao delimitador padrão da linguagem
14.
15. CALL proc_demo1();
```

Programação SQL: Fundamentos

› Prática 4.5: IF / CASE

– Na database LAB_05, execute:

RESPONDA:

1. Em **a.**, qual a diferença entre os comandos SELECT? Qual o resultado?
2. Em **b.**, apresente e explique o que aparece na coluna **'Nível Estoque'** do **SELECT**?

Nota: A função **WEEKDAY(data)** retorna o número do dia da semana para uma data.
0 = segunda-feira, 1 = terça-feira, 2 = quarta-feira,
3 = quinta-feira, 4 = sexta-feira, 5 = sábado, 6 = domingo.

(a)

```
-- IF: testa de uma CONDIÇÃO é TRUE ou FALSE, apenas

-- SINTAXE: IF(condition, value_if_true, value_if_false)

SELECT IF (WEEKDAY(NOW()) IN (5, 6), 'É FIM de semana', 'É DIA de semana') AS 'DIA DE HOJE';

SELECT IF (WEEKDAY('2023-09-24') IN (5, 6), 'É FIM de semana', 'É DIA de semana') AS '24/09/2023';
```

(b)

```
SELECT ID AS 'Código', Nome,
     Qtde AS 'Quantidade',
     IF (Qtde < 100, 'Baixo (menor que 100)', 'Em boa quantidade') AS 'Nível Estoque'
FROM Estoque;
```

Programação SQL: Fundamentos

› Prática 4.5: IF / CASE

– Na database LAB_04, execute:

(c)

```
-- CASE: retorna valor que pode ser atribuído
-- SINTAXE:
-- CASE
--    WHEN condition1 THEN result1
--    WHEN condition2 THEN result2
--    WHEN conditionN THEN resultN
--    ELSE result
-- END;

INSERT INTO Estoque (Nome, ValUnit) VALUES ('cola bastão', 15.00); -- 1º. INSERT

INSERT INTO Estoque (Nome, Qtde, ValUnit) VALUES ('tesoura', NULL, 15.00); -- 2º. INSERT

SELECT  ID AS 'Código', Nome, Qtde AS 'Quantidade',
        CASE
          WHEN Qtde < 100 THEN 'BAIXO'
          WHEN Qtde BETWEEN 100 AND 300 THEN 'OK'
          WHEN Qtde > 300 THEN 'ALTO'
          ELSE 'DESCONHECIDO'
        END AS 'Nível Estoque'
FROM Estoque
ORDER BY Nome;
```

RESPONDER:

1. No **1º. INSERT**, não foi especificado um valor de **Qtde**. Qual o **Nível de Estoque** apresentado no **SELECT**? Por que esse valor foi exibido?
2. No **2º. INSERT**, foi especificado que de **Qtde = NULL**. Qual o **Nível de Estoque** apresentado no **SELECT**? Por que esse valor foi exibido?
3. Quando a opção **ELSE** do **CASE** é a executada?
4. Apresente o resultado do **SELECT**.

Programação SQL: Fundamentos

- **Operadores Aritméticos**

  + - * /

  - int, smallint, tinyint, numeric, decimal, float, real,...

- **Operadores Lógicos**

  AND OR NOT

- **Operadores Relacionais**

  = > < <= >= <>

- **Referência**: https://www.w3schools.com/mysql/mysql_operators.asp

Programação SQL: Fundamentos

› Prática 4.6: Comparação

RESPONDA:
1. Apresente o resultado de cada **SELECT**.
2. Qual consulta de **SELECT** satisfaz o enunciado? Por que?

- **Problema:**
  - **Encontrar as empresas que não estão nem em São Paulo, SP nem em Morretes, PR.**

- Na database LAB_04, execute:

(a)

```
SELECT   nome, Atuacao, Cidade, Estado
FROM     Empresa
WHERE
((Cidade <> 'São Paulo' AND Estado <> 'SP')
OR
(Cidade <> 'Morretes' AND Estado <> 'PR')
);
```

(b)

```
SELECT   nome, Atuacao, Cidade, Estado
FROM     Empresa
WHERE NOT (
((Cidade = 'São Paulo' AND Estado = 'SP')
OR
(Cidade = 'Morretes' AND Estado = 'PR'))
);
```

Programação SQL: Fundamentos

Experimente executar esses exemplos.

- **Funções Matemáticas**
  - Exemplos:
    - **ABS** (numeric_expr) - Valor absoluto da expressão numérica

      Exemplo:
      ```
      SELECT ABS(-136.17);-- Calcula valor absoluto de -136.17
      ```

    - **COS** (float_expr) - Cosseno trigonométrico do ângulo em radianos

      Exemplo:
      ```
      SELECT COS(PI());      -- Calcula cosseno de 180º (pi)
      ```

    - **POWER** (float_expr, float_expr,) - Valor exponencial da expressão numérica

      Exemplo:
      ```
      SELECT POWER(2, 3);  -- Potência: 2^3 = 8
      ```

    - **ROUND** (numeric_expr, length) - Expressão numérica arredondada

      Exemplo:
      ```
      SELECT ROUND(-136.1459,2) -- Arredonda para 2 casas decimais
      ```

    - **SQRT** (flat_expr) - Raíz quadrada da expressão numérica

      Exemplo:
      ```
      SELECT SQRT(144);      -- Calcula raíz quadrada de 144
      ```

Programação SQL: Fundamentos

› Prática 4.7: Funções Matemáticas

- Na database LAB_04, execute:

RESPONDER:

1. Qual é a variável utilizada no exemplo e como ela foi definida?
2. O que faz a função **PI()**?
3. O que faz a função **CONCAT()**?
4. O que faz a função **CONVERT()**?
5. Apresente o resultado do comando **SELECT** do exemplo.

(a)

```
-- Funções trigonométricas usam ângulos em radianos:

SET     @angle = PI()/4; -- 45º em rad

SELECT CONCAT( 'O SENO do ângulo: ' ,
       CONVERT(ROUND(@angle,3), CHAR) ,
      ' rad = ' ,
       CONVERT(ROUND(SIN(@angle),3), CHAR)) AS 'SENO 45º (ou PI/4 rad)';
```

Programação SQL: Fundamentos

› Prática 4.7: Funções Matemáticas

- Na database LAB_04, execute:

(b)

```
1.  DROP PROCEDURE IF EXISTS proc_demo2;
2.
3.  DELIMITER \\
4.  CREATE PROCEDURE proc_demo2(IN angle FLOAT, OUT output VARCHAR (100))
5.  BEGIN
6.      SET output = '';
7.      SET output = CONCAT (output,
8.        ' [ ANGULO_GRAUS = ', CONVERT(ROUND(@angle * 180 / PI (), 3), CHAR), ']',
9.        ' [ ANGULO_RAD = ', CONVERT(ROUND(@angle, 3 ), CHAR), ']',
10.       ' [ SENO = ', CONVERT(ROUND(SIN(@angle),3 ), CHAR), ']',
11.       ' [ COSSENO = ', CONVERT(ROUND(COS(@angle),3 ), CHAR), ']',
12.       ' [ TANGENTE = ', CONVERT(ROUND(TAN(@angle),3 ), CHAR), ']');
13. END \\
14. DELIMITER ;
15.
```

RESPONDER:

1. Em **a.**, Para que servem os comandos das linhas **3.** e **14.**?
2. Quais são e como são definidos os **parâmetros** da procedure **proc_demo2()**?
3. O que está sendo feito na **atribuição** que inicia na linha **7.**?
4. Em **c.**, apresente e explique o resultado o de cada um dos 3 conjuntos de comandos.

(c)

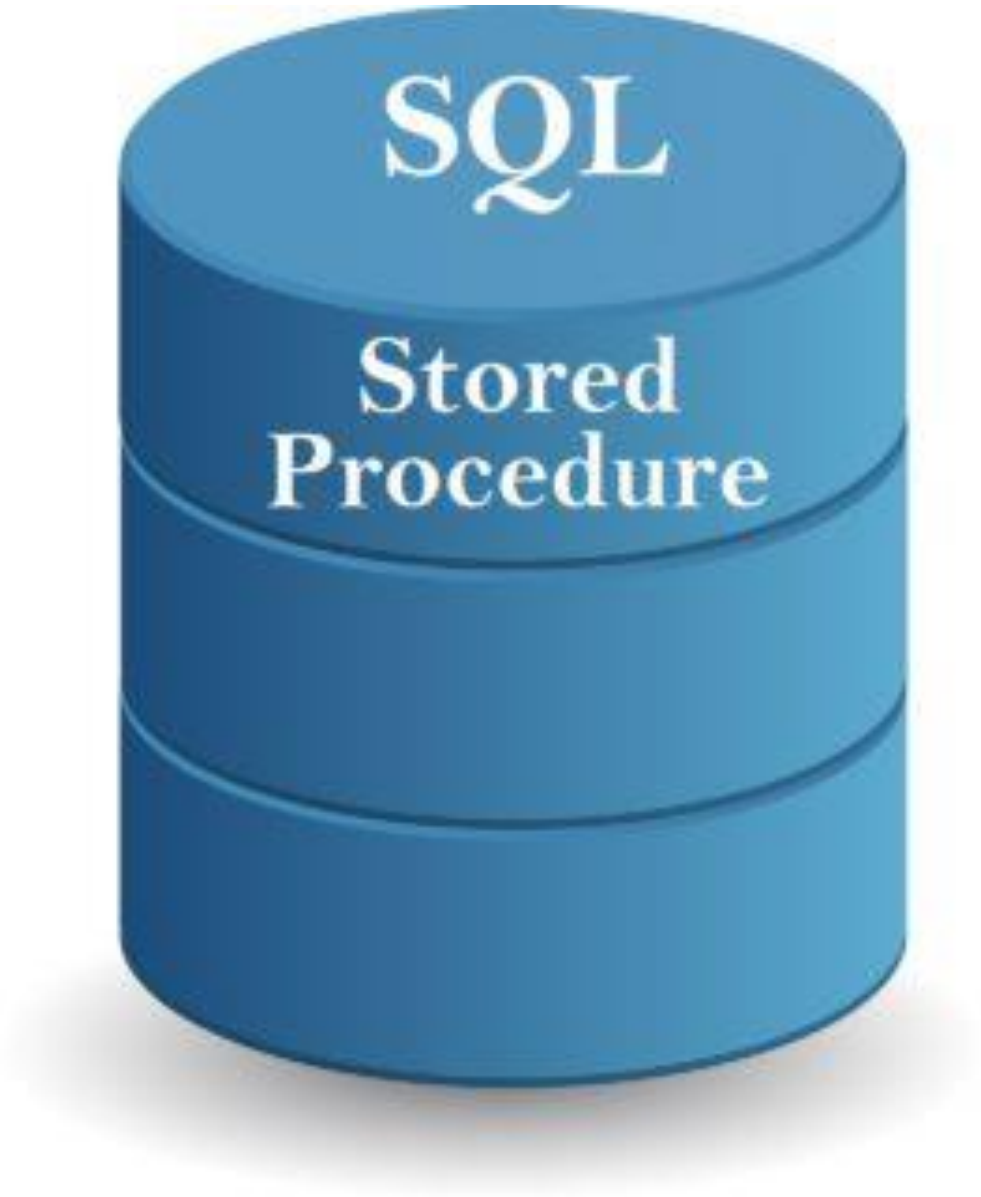
```
SET @angle = PI()/3;
SET @resp = '';
CALL proc_demo2(@angle, @resp);
SELECT @resp AS 'RESPOSTA';

SET @angle = PI()/4;
SET @resp = '';
CALL proc_demo2(@angle, @resp);
SELECT @resp AS 'RESPOSTA';

SET @angle = PI()/6;
SET @resp = '';
CALL proc_demo2(@angle, @resp);
SELECT @resp AS 'RESPOSTA';
```

Programação SQL: Fundamentos

- **Referências rápidas**
  - Comandos básicos: https://www.w3schools.com/MySQL/mysql_sql.asp
  - Manipulação de Database: https://www.w3schools.com/MySQL/mysql_create_db.asp
  - Tipos de Dados: https://www.w3schools.com/MySQL/mysql_datatypes.asp
  - Exemplos gerais (básicos): https://www.w3schools.com/MySQL/mysql_examples.asp

SQL STORED PROCEDURE

Procedimentos Armazenados no SQL



Stored Procedures

- Ou Procedimentos Armazenados
- São sub-rotinas, ou blocos de comandos SQL, que podem ser usados com lógica de programação (com variáveis, condições, laços de repetição, etc).

- Permitem escrever "regras de negócio", ou funcionalidades, que executam no SGBD, reduzindo a quantidade a troca de dados (via rede) entre cliente e servidor

- Quando utilizar ***stored procedures***?
  - Quando precisamos utilizar a mesma funcionalidade em diferentes aplicações, desenvolvidas em diferentes linguagens de programação.
  - Quando precisamos centralizar uma funcionalidade compartilhada por diferentes aplicações.

- Sintaxe:
  - CREATE PROCEDURE *proc_name* -- cria a stored procedure
  - CALL *proc_name* -- chama a stored procedure

Prática:
Identifique o armazenamento das procedures no SGBD!

Stored Procedures Recursivas

- Recursão ou Recursividade
  - Em programação, é o mecanismo que permite que uma rotina (procedimento, função ou método na POO) possa ser definida com uma chamada a si mesma.
  - Exemplo:

- **Cálculo do fatorial de n:**

```
int fatorial(int n) {
   if(n <= 1)
      return 1;
   else
      return n * fatorial(n-1);
}
```

fatorial (6)
6 * fatorial (5)
6 * 5 * fatorial (4)
6 * 5 * 4 * fatorial (3)
6 * 5 * 4 * 3 * fatorial (2)
6 * 5 * 4 * 3 * 2 * fatorial (1)
6 * 5 * 4 * 3 * 2 * 1 * 1
6 * 5 * 4 * 3 * 2 * 1
6 * 5 * 4 * 3 * 2
6 * 5 * 4 * 6
6 * 5 * 24
6 * 120
720

Stored Procedures Recursivas

- MySQL: @@max_sp_recursion_depth
  - **Variável Global**
    - No MySQL, o usuário não pode criar variáveis globais personalizadas (de sistema ou de sessão).
    - Apenas pode alterar variáveis de sistema globais ou de sessão existentes
  - O número de vezes que qualquer procedimento armazenado pode ser chamado recursivamente.
  - O valor padrão = 0, que desativa completamente a recursão em procedimentos armazenados.
  - O valor máximo é 255.

```
SET @@max_sp_recursion_depth = 50;  -- Altera a variável global
```

Stored Procedures

Exemplo: chamadas recursivas de Fatorial

		IN			OUT
Chamada	**Iteração**	**param**	**param_menos**	**total = param * tmp_total**	**total**
Fat(4)	1ª chamada	4	3	4 * Fat(3) =	4 * 6 = 24	24
Fat(3)	2ª chamada	3	2	3 * Fat(2) =	3 * 2 = 6	6
Fat(2)	3ª chamada	2	1	2 * Fat(1) =	2 * 1 = 2	2
Fat(1)	4ª chamada	1	--	--	--	1

› Prática 4.8: Stored Procedures Recursivas

- Na database LAB_04, execute:

(a)

```
1.  DROP PROCEDURE IF EXISTS fatorial;
2.
3.  DELIMITER //
4.  CREATE PROCEDURE fatorial(IN param INT, OUT total INT)
5.  BEGIN
6.     DECLARE param_menos INT DEFAULT NULL ;
7.     DECLARE tmp_total   INT DEFAULT -1;
8.     SET @@max_sp_recursion_depth = 50;
9.     IF (param IS NULL) OR (param < 0) OR (param > 12)
10.       THEN SET total = -1;
11.    ELSEIF (param = 0) OR (param = 1)
12.       THEN SET total = 1;
13.    ELSE
14.             SET param_menos = param - 1;
15.       CALL fatorial(param_menos, tmp_total);
16.       IF (tmp_total = -1)
17.                THEN SET total = -1;
18.          ELSE SET total = tmp_total * param;
19.          END IF;
20.    END IF;
21. END //
22. DELIMITER ;
```

RESPONDER:

1. O que é feito no comando da linha **8.**?
2. Se o parâmetro de entrada **param** não tiver valor (= **NULL**) qual será o valor do parâmetro de saída total?
3. Qual o **valor máximo** que podemos utilizar para calcular o fatorial, no exemplo passado?

Stored Procedures

› Prática 4.8: Stored Procedures Recursivas

- Na database LAB_04, execute cada um dos conjuntos de comandos separadamente:

(b)

```
-- conjunto commandos 1:
SET @resp = -1;
CALL fatorial (0, @resp);
SELECT @resp;

-- conjunto commandos 2:
CALL fatorial (13, @resp);
SELECT @resp;

-- conjunto commandos 3:
CALL fatorial (4, @resp);
SELECT @resp;

-- conjunto commandos 4:
CALL fatorial (-4, @resp);
SELECT @resp;

-- conjunto commandos 5:
CALL fatorial (6, @resp);
SELECT @resp;
```

RESPONDER:

Para **cada conjunto de comandos** do exemplo, indique:

1. Mostre o **resultado da execução** do conjunto de comandos, indicando qual o **valor numérico** que está sendo calculado o **fatorial**.
2. **Justifique**, de acordo com a **stored procedure**, o resultado apresentado.

Stored Procedure x View

- As VIEWS (ou visualizações)
  - devem ser usadas para armazenar consultas JOIN (SELECT com JOINs) que são usadas com muita frequência, com campos específicos (colunas) para construir tabelas dinâmicas (ou virtuais) de um conjunto exato de dados que queremos ver.
- As STORED PROCEDURES (ou procedimentos armazenados)
  - Contêm uma lógica mais complexa, como várias instruções, como INSERT, DELETE e UPDATE, e
  - Servem para automatizar grandes fluxos de trabalho SQL.

Referência Bibliográfica

- Sistema de Banco de Dados
  - Abraham Silberschatz, Henry F. Korth, S. Sudaarshan

- Referência do MySQL
  - Chapter 13 SQL Statements: https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html
  - W3Schools: https://www.w3schools.com/mysql/mysql_drop_db.asp
  - MySQL 8.0 Reference Manual: https://dev.mysql.com/doc/refman/8.0/en/