Banco de Dados

Laboratório 5

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

Tópicos

- SQL
  - Funções
  - Transações
  - Triggers

SQL FUNCTIONS

Funções no SQL

Função

- **Function**
  - Rotina ou procedimento reutilizável, que contém uma ou mais instruções SQL para realizar operações específicas.
  - Aceita parâmetros de entrada, executa operações e retorna o resultado dessa operação como um valor.

- **Atenção ao declarar Função**
  - Função DETERMINÍSTICA
    - Sempre retorna os mesmos resultados para os mesmos valores de entrada.
  - Função NÃO DETERMINÍSTICA
    - Pode retornar resultados diferentes para os mesmos valores de entrada.
  - Se você não explicitar qual função, DETERMINISTIC ou NOT DETERMINISTIC, o MySQL usará a NOT DETERMINISTIC por padrão

Prática 5.1

- Function
  - Na database LAB_05, execute:

(A)

```
1.  CREATE DATABASE LAB_05;
2.  USE LAB_05;
3.
4.  DELIMITER $$
5.  -- function: não usa IN / OUT nos parâmetros da função
6.  CREATE FUNCTION Diagonal (ladoA FLOAT, ladoB FLOAT)
7.  RETURNS FLOAT
8.  DETERMINISTIC  -- define que a função é determinística
9.  BEGIN
10.     DECLARE DIAG FLOAT DEFAULT -1;
11.     SET DIAG = SQRT(POWER(ladoA, 2) + POWER(ladoB, 2));
12.     RETURN DIAG;
13. END; $$
14.
15. DELIMITER ;
16.
17. SELECT Diagonal(3, 4) AS 'Diagonal do retângulo 3m x 4m';
```

RESPONDA

a) Essa **FUNCTION** tem parâmetros de entrada? Se sim, quais e de quais **TIPOS**?

b) Qual a diferença de **RETURNS** e **RETURN**, nas linhas **.7** e **.12**, respectivamente?

c) Em qual linha do código chamamos a **FUNCION** definida em **(A)**?

d) Explique e apresente a imagem com o resultado do **SELECT**.

Prática 5.2

- Function
  - Na database LAB_05, execute:

(B)

```
1.  DELIMITER $$
2.
3.  CREATE FUNCTION CalcSalario (valor_inicial INT)
4.  RETURNS INT
5.  DETERMINISTIC
6.  BEGIN
7.      DECLARE salario INT DEFAULT 0;
8.      WHILE salario <= 3000 DO
9.          SET salario = salario + valor_inicial;
10.     END WHILE;
11.     RETURN salario;
12. END; $$
13.
14. DELIMITER ;
15.
16. SELECT CalcSalario(500) AS 'Salário Final';
```

RESPONDA

a) Essa **FUNCTION** tem parâmetros de entrada? Se sim, quais e de quais **TIPOS**?

b) Em qual linha do código chamamos a **FUNCION** definida em **(B)**?

c) Apresente e explique o resultado do **SELECT**.

d) Qual é o **Salário Final** para um **valor inicial = 200**?

e) Qual é o **Salário Final** para um **valor inicial = 2000**?

SQL TRANSACTIONS

Transações no SQL

Transações

- DEFINIÇÃO
  - **unidade básica** de trabalho de um SGBD Relacional
  - Conjunto de operações, ou vários comandos SQL

- ATOMICIDADE
  - Ou todo comando da Transação é executado corretamente, ou nenhum comando é realizado
  - Se erro, uma transação retrocede e todos os seus comandos são desfeitos
  - Palavras reservadas no **MySQL**:
    - START TRANSACTION: marca o início de uma transação. As terminam com a instrução COMMIT ou ROLLBACK.
    - COMMIT: marca o término de uma transação **bem-sucedida**; o BD está novamente em um estado consistente e as atualizações feitas se tornam permanentes.
    - ROLLBACK: marca o término de uma transação **mal-sucedida**; deve ser executado se o BD estiver em um estado inconsistente. Logo, quaisquer operações da transação sevem ser desfeitas.

Prática 5.3

- Commit / Rollback
  - Na database LAB_05, execute:

RESPONDER:

a) Em **(A)**, os comandos de **INSERT** da **transação** funcionaram? Por que? Justifique e apresente a imagem com o resultado do **SELECT**.

b) Em **(B)**, os comandos de **INSERT** da **transação** funcionaram? Por que? Justifique e apresente a imagem com o resultado do **SELECT**.

(A)

```
DROP TABLE IF EXISTS Tab_Teste;

CREATE TABLE Tab_Teste (
col1     INT NOT NULL PRIMARY KEY,
col2     INT NOT NULL);

SELECT * FROM Tab_Teste;
```

(B)

```
START TRANSACTION ;
   INSERT Tab_Teste VALUES (1,111) ;
   INSERT Tab_Teste VALUES (2,222) ;
COMMIT;

SELECT * FROM Tab_Teste;
```

(C)

```
START TRANSACTION ;
   INSERT Tab_Teste VALUES (3,333) ;
   INSERT Tab_Teste VALUES (4,444) ;
ROLLBACK;

SELECT * FROM Tab_Teste;
```

Transações

› Tratamento de Erros

- Nem todos os erros em transações levam ao seu **retrocesso automático**, a fim de garantir a **atomicidade**

- Solução:
  - › Trabalhar com **ações condicionais**, verificando **se houve erro** após a execução de algum comando / operação / instrução
  - › Se for encontrado **erro** e o **erro não for tratado**, o processamento irá para a próxima instrução, podendo deixar o banco em estado inconsistente.
  - › Exemplo:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION -- declara tratamento de erro
  BEGIN
        ROLLBACK;   -- rollback se erro na transação (desfaz comandos que executaram)
        RESIGNAL;   -- retorna a exceção para quem chamou a SP
  END;
```

Prática 5.4

RESPONDER:

1. Em **(A)**, apresente a imagem e explique o resultado do **SELECT**.
2. Em **(B)**, a **SP** tem algum comando que não funciona? Qual e porque está incorreto?
3. Em **(B)**, o **COMMIT** da **SP** executa? Explique.
4. Em **(B)**, apresente a imagem e explique o resultado do **SELECT**.

- Transação SEM tratamento de erro
  - Na database LAB_05, execute:

(A)

```
DROP TABLE IF EXISTS Tab_Teste;

CREATE TABLE Tab_Teste (
col1    INT NOT NULL PRIMARY KEY,
col2    INT NOT NULL);

SELECT * FROM Tab_Teste;
```

(B)

```
DROP PROCEDURE IF EXISTS nãoTratErroTransact;

DELIMITER $$
CREATE PROCEDURE nãoTratErroTransact()
BEGIN
   START TRANSACTION;
        INSERT Tab_Teste VALUES (1,111) ;
        INSERT Tab_Teste VALUES (2,222) ;
        INSERT Tab_Teste VALUES (3,333) ;
        INSERT Tab_Teste VALUES (1,101) ;
   COMMIT; -- esse commando executa?
END $$
DELIMITER ;

CALL nãoTratErroTransact();

SELECT * FROM Tab_Teste;
```

Prática 5.5

- Transação COM tratamento de erro
  - Na database LAB_05, execute:

(A)

```
DROP TABLE IF EXISTS Tab_Teste;

CREATE TABLE Tab_Teste (
col1    INT NOT NULL PRIMARY KEY,
col2    INT NOT NULL);

SELECT * FROM Tab_Teste;
```

RESPONDER:

a) Em **(A)**, apresente a imagem e explique o resultado do **SELECT**.

b) Em **(B)**, a **SP** tem algum comando que não funciona? Qual e porque está incorreto?

c) Em **(B)**, o **COMMIT** da **SP** executa? Explique.

d) Em **(B)**, apresente a imagem e explique o resultado do **SELECT**.

(B)

```
DROP PROCEDURE IF EXISTS tratErroTransact;

DELIMITER $$
CREATE PROCEDURE tratErroTransact()
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
        ROLLBACK;
        RESIGNAL;
   END;
   START TRANSACTION;
        INSERT Tab_Teste VALUES (1,111) ;
        INSERT Tab_Teste VALUES (2,222) ;
        INSERT Tab_Teste VALUES (3,'um') ;
        INSERT Tab_Teste VALUES (3,333) ;
   COMMIT; -- esse commando executa?
END $$
DELIMITER ;

CALL tratErroTransact();

SELECT * FROM Tab_Teste;
```

SQL TRIGGERS

Procedures disparadas automaticamente

Triggers

› Triggers

- Um *gatilho* [trigger] é um tipo de procedimento armazenado, que é executado automaticamente quando ocorre um **evento**, como uma alteração em tabela:
  - O Trigger (procedimento) "dispara" quando ocorre uma operação INSERT, UPDATE ou DELETE numa tabela.

- Geralmente são usados para reforçar restrições *(constrains)* de integridade que não podem ser tratadas pelos recursos mais simples, como CHECKs, valores DEFAULT, CONSTRAINs em geral, a opção NOT NULL etc.

Importante:

Deve-se dar preferência à utilização de DEFAULT e CONSTRAINs em relação a Triggers, sempre que estes fornecerem toda a funcionalidade necessária!

Triggers

- **Exemplos de Sintaxe**:

1. Trigger ANTES de (BEFORE): é disparada antes o comando de INSERT:

```
CREATE TRIGGER TableAInsertTrig
BEFORE INSERT ON TableA FOR EACH ROW
BEGIN ...
END
```

2. Uso de trigger APÓS (AFTER): é disparada após o comando de DELETE:

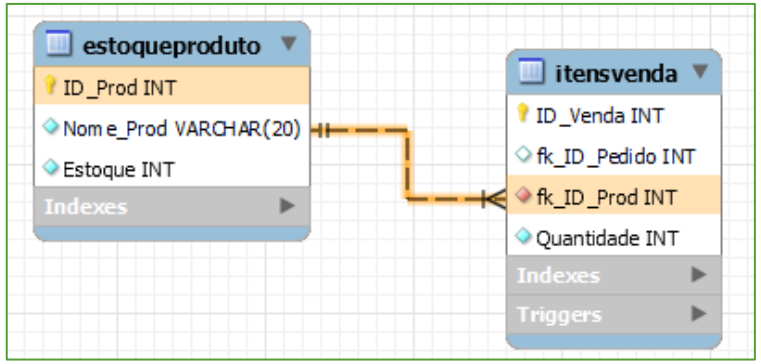
```
CREATE TRIGGER TableBDeleteTrig
AFTER DELETE ON TableB FOR EACH ROW
BEGIN ...
EN
```

Triggers

- **Exemplos de Utilização**:

1. Em um trigger de INSERT:
   - Apenas **NEW.col_name** pode ser usada; não existe **OLD.cold_name**.
2. Em um trigger de DELETE:
   - Apenas **OLD.col_name** pode ser usada; não existe **NEW.col_name**.
3. Em um trigger de UPDATE:
   - Podemos usar ambos os campos **OLD.col_name** e **NEW.col_name**

Prática 5.6



RESPONDA:

a) O que significa o **UNIQUE** da tabela **EstoqueProduto**?

b) Apresente a imagem com o resultado do comando **SELECT * FROM EstoqueProduto;**

c) Escreva um comando de **INSERT** na tabela **EstoqueProduto** que viola a restrição de **UNIQUE**.

- Trigger
  - Na database LAB_05, execute:

(A)

```
DROP TABLE IF EXISTS EstoqueProduto;
-- Cria Tabela EstoqueProduto
CREATE TABLE EstoqueProduto (
ID_Prod     INT PRIMARY KEY,
Nome_Prod   VARCHAR(20) NOT NULL UNIQUE,
Estoque     INT NOT NULL
);
INSERT INTO EstoqueProduto (ID_Prod, Nome_prod, Estoque) VALUES (123, 'Caderno', 100);
INSERT INTO EstoqueProduto (ID_Prod, Nome_prod, Estoque) VALUES (456, 'Bloco A4', 50);
INSERT INTO EstoqueProduto (ID_Prod, Nome_prod, Estoque) VALUES (789, 'Caneta', 200);

SELECT * FROM EstoqueProduto;

DROP TABLE IF EXISTS ItensVenda;
-- Cria Tabela ItensVenda
CREATE TABLE ItensVenda (
ID_Venda              INT AUTO_INCREMENT PRIMARY KEY,
fk_ID_Pedido          INT,    -- Tab Pedido não criada nesta demonstração
fk_ID_Prod            INT NOT NULL REFERENCES EstoqueProduto(ID_Prod), -- FK
Quantidade            INT NOT NULL,UNIQUE (fk_ID_Pedido, fk_ID_Prod)
);

SELECT * FROM ItensVenda;
```

Prática 5.7

- Trigger
  - Na database LAB_05, execute:

RESPONDA:

a) Sobre o **TRIGGER** em **(B)**:
   - Ele atua para qual tabela?
   - Quando ele é disparado?
   - Que atualização ele realiza em que outra tabela?

b) Sobre o **TRIGGER** em **(C)**:
   - Ele atua para qual tabela?
   - Quando ele é disparado?
   - Que atualização ele realiza em que outra tabela?

(B)

```
DROP TRIGGER IF EXISTS Tgr_ItensVenda_Insert;

-- Cria Trigger Tgr_ItensVenda_Insert
DELIMITER $$
CREATE TRIGGER Tgr_ItensVenda_Insert
AFTER INSERT   -- A Trigger dispara após o INSERT
ON ItensVenda
FOR EACH ROW
BEGIN
          UPDATE EstoqueProduto
          SET Estoque = Estoque - NEW.Quantidade
          WHERE ID_Prod = NEW.fk_ID_Prod;
END $$
DELIMITER ;
```

(C)

```
DROP TRIGGER IF EXISTS Tgr_ItensVenda_Delete;

-- Cria Trigger Tgr_ItensVenda_Insert
DELIMITER $$
CREATE TRIGGER Tgr_ItensVenda_Delete
AFTER DELETE   -- A Trigger dispara após o DELETE
ON ItensVenda
FOR EACH ROW
BEGIN
          UPDATE EstoqueProduto
          SET Estoque = Estoque + OLD.Quantidade
          WHERE ID_Prod = OLD.fk_ID_Prod;
END $$
DELIMITER ;
```

Prática 5.8

- Trigger
  - Na database LAB_05, execute:

RESPONDA:

a) Sobre o **comandos** em **(D)**:
   - Qual o **TRIGGER** é disparado pelos **INSERTs**?
   - Após os **INSERTs**, o que foi alterado, em que tabelas? Por que?
   - Apresente a imagem com os resultados dos **SELECTs**.

b) Sobre o **comandos** em **(E)**:
   - Qual o **TRIGGER** é disparado pelos **INSERTs**?
   - Após os **INSERTs**, o que foi alterado, em que tabelas? Por que?
   - Apresente a imagem com os resultados dos **SELECTs**.

(D)

```
INSERT INTO ItensVenda (fk_ID_Pedido, fk_ID_Prod, Quantidade) VALUES (1, 123, 30);
INSERT INTO ItensVenda (fk_ID_Pedido, fk_ID_Prod, Quantidade) VALUES (1, 456, 10);
INSERT INTO ItensVenda (fk_ID_Pedido, fk_ID_Prod, Quantidade) VALUES (1, 789, 25);

SELECT * FROM ItensVenda;
SELECT * FROM EstoqueProduto;
```

(E)

```
DELETE FROM ItensVenda WHERE fk_ID_Pedido = 1 AND fk_ID_Prod = 123;
DELETE FROM ItensVenda WHERE fk_ID_Pedido = 1 AND fk_ID_Prod = 789;

SELECT * FROM ItensVenda;
SELECT * FROM EstoqueProduto;
```

RESPONDA:
a) Apresente a imagem com o resultado do **SELECT** em **Editora**;

Prática 5.9

› Trigger para Log
  – Na database LAB_05, gere novamente as tabelas **Editora** e **Autor**:

(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');

SELECT * FROM Editora;
```

Prática 5.10

RESPONDA:
a) Apresente a imagem com o resultado do **SELECT** em **Autor**
b) É possível criar a tabela **Autor** antes da tabela **Editora**? Por que?

- Trigger para Log
  - Na database LAB_05, gere novamente as tabelas **Editora** e **Autor**:

(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);

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);

SELECT * FROM Autor;
```

Prática 5.11

RESPONDA:

a) Para que servem os campos **ID_log**, **Operation** e **ChangeDate** e **UserName**?
b) Qual a diferença entre os campos **OldID_autor** e **NewID_autor**?
c) Qual a diferença entre os campos **OldAutor** e **NewAutor**?
d) Apresente a imagem com o resultado do **SELECT** em **AutorLog**. Quando esta tabela será povoada?

- Trigger para Log
  - Na database LAB_05, gere novamente as tabelas **Editora** e **Autor**:

(C)

```
DROP TABLE IF EXISTS  AutorLog;

-- Tabela de LOG (rastreamento) referente à Tabela Autor
CREATE TABLE AutorLog (
ID_log                INT AUTO_INCREMENT PRIMARY KEY,
Operation             CHAR(6) NOT NULL,     -- Operação realizada
ChangeDate            DATETIME NOT NULL,    -- Data da realização da operação
UserName              VARCHAR(20) NOT NULL, -- Usuário de BD que realizou a operção
OldID_Autor           INT NULL,             -- Valor antigo para ID_Autor
NewID_autor           INT NULL,             -- Valor novo   para ID_Autor
OldAutor              VARCHAR(50) NULL,     -- Valor antigo para Nome de Autor
NewAutor              VARCHAR(50) NULL,     -- Valor novo   para Nome de Autor
OldDtNasc             DATE NULL,            -- Valor antigo para Data de Nascimento do Autor
NewDtNasc             DATE NULL,            -- Valor novo   para Data de Nascimento do Autor
OldID_Edit            INT NULL,             -- Valor antigo para ID do Editor do Autor
NewID_Edit            INT NULL              -- Valor novo   para ID do Editor do Autor
);

SELECT * FROM AutorLog;
```

Prática 5.12

RESPONDA:

a) Sobre o **TRIGGER** em **(D)**:
- Ele atua para qual tabela?
- Quando ele é disparado?
- Que atualização ele realiza em que outra tabela?

b) Sobre o **TRIGGER** em **(E)**:
- Ele atua para qual tabela?
- Quando ele é disparado?
- Que atualização ele realiza em que outra tabela?

- Trigger
  - Na database LAB_05, execute:

(D)

```
DROP TRIGGER IF EXISTS AutorLogInsert;
-- Cria Trigger Tgr_ItensVenda_Insert
DELIMITER $$
CREATE TRIGGER AutorLogInsert
AFTER INSERT    -- A Trigger dispara após o INSERT
ON Autor
FOR EACH ROW
BEGIN
  INSERT    INTO  -- Insere registro na tabela AutorLog
  AutorLog (Operation, ChangeDate, UserName, NewID_Autor, NewAutor, NewDtNasc, NewID_Edit)
  SELECT    'Insert', NOW(), CURRENT_USER(), NEW.ID_autor, NEW.nome_autor, NEW.dt_nasc, NEW.fk_ID_edit;
END $$
DELIMITER ;
```

(E)

```
DROP TRIGGER IF EXISTS AutorLogDelete;
-- Cria Trigger Tgr_ItensVenda_Delete
DELIMITER $$
CREATE TRIGGER AutorLogDelete
AFTER DELETE -- A Trigger dispara após o DELETE
ON Autor
FOR EACH ROW
BEGIN
  INSERT    INTO  -- Insere registro na tabela AutorLog
  AutorLog (Operation, ChangeDate, UserName, OldID_Autor, OldAutor, OldDtNasc, OldID_Edit)
  SELECT    'Delete', NOW(), CURRENT_USER(), OLD.ID_autor, OLD.nome_autor, OLD.dt_nasc, OLD.fk_ID_edit;
END $$
DELIMITER ;
```

Prática 5.13

- Trigger
  - Na database LAB_05, execute:

RESPONDA:

a) Sobre o **TRIGGER** em **(F)**:
- Ele atua para qual tabela?
- Quando ele é disparado?
- Que atualização ele realiza em que outra tabela?

(F)

```
DROP TRIGGER IF EXISTS AutorLogUpdate;

-- Cria Trigger Tgr_ItensVenda_Update
DELIMITER $$
CREATE TRIGGER AutorLogUpdate
AFTER UPDATE   -- A Trigger dispara após o UPDATE
ON Autor
FOR EACH ROW
BEGIN
  INSERT    INTO  -- Insere registro na tabela AutorLog
  AutorLog (Operation, ChangeDate, UserName, OldID_Autor, NewID_Autor, OldAutor, NewAutor,OldDtNasc,NewDtNasc,   OldID_Edit, NewID_Edit)
  SELECT  'Update', NOW(), CURRENT_USER(), OLD.ID_autor, NEW.ID_autor, OLD.nome_autor, NEW.nome_autor, OLD.dt_nasc, NEW.dt_nasc, OLD.fk_ID_edit, NEW.fk_ID_edit;
END $$
DELIMITER ;
```

Prática 5.14

- Trigger
  - Na database LAB_05, execute:

(G)

```
-- Comandos 1) Teste de UPDATE ----
UPDATE Autor SET nome_autor = 'José da Silva'
WHERE ID_autor = 100;

SELECT * FROM Autor;
SELECT * FROM AutorLog;

-- Comandos 2) Teste de INSERT ----
INSERT Autor (nome_autor, dt_nasc, fk_ID_Edit)
VALUES
('Karolina', '1976-06-18', 3),
('Cláudio',  '1982-10-28', 4),
('Ricardo',  '1990-02-13', 3);

SELECT * FROM Autor;
SELECT * FROM AutorLog;

-- Comandos 3) Teste de DELETE ---
DELETE FROM Autor
WHERE ID_autor = 102 OR ID_autor = 103;

SELECT * FROM Autor;
SELECT * FROM AutorLog;
```

RESPONDA:

1. Sobre o **Comandos 1** em **(G)**:
   - Qual foi o **Trigger** disparado?
   - O que foi **preenchido** em que **tabela**?
   - Apresente o resultado dos **SELECTs**?
2. Sobre o **Comandos 2** em **(G)**:
   - Qual foi o **Trigger** disparado?
   - O que foi **preenchido** em que **tabela**?
   - Apresente o resultado dos **SELECTs**?
3. Sobre o **Comandos 3** em **(G)**:
   - Qual foi o **Trigger** disparado?
   - O que foi **preenchido** em que **tabela**?
   - Apresente o resultado dos **SELECTs**?

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/