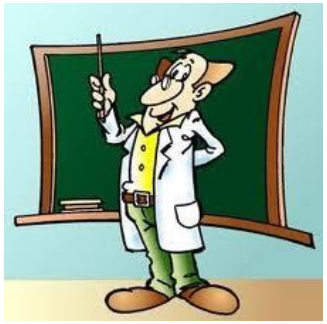
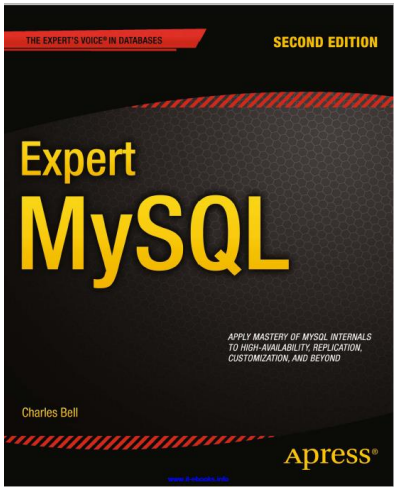
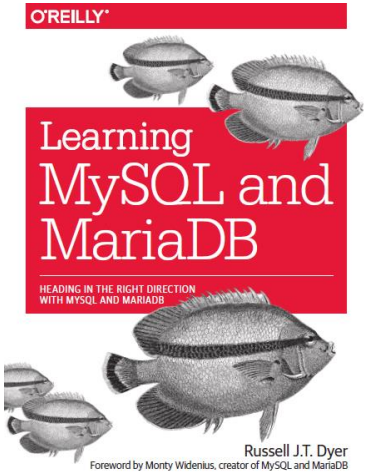
Unidade 21 - MySQL Triggers

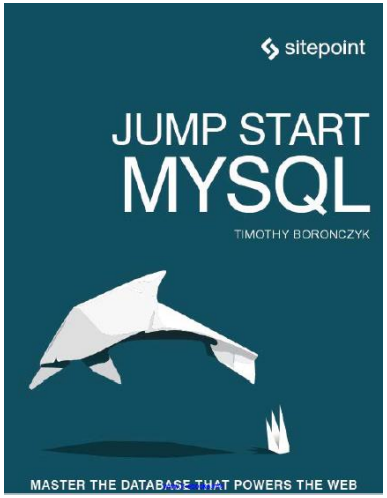


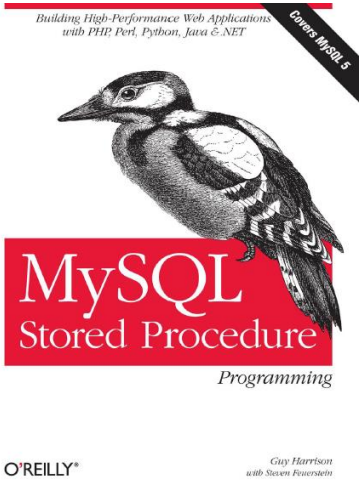
Prof. Aparecido V. de Freitas
Doutor em Engenharia da Computação pela EPUSP
aparecidovfreitas@gmail.com

Bibliografia









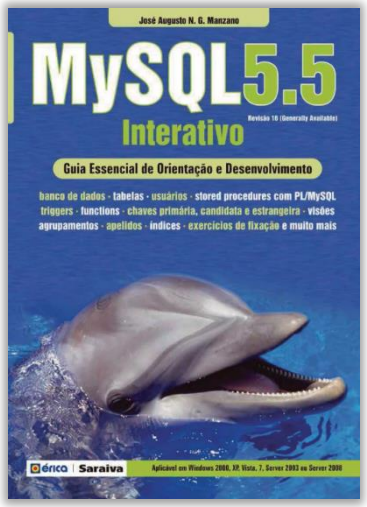
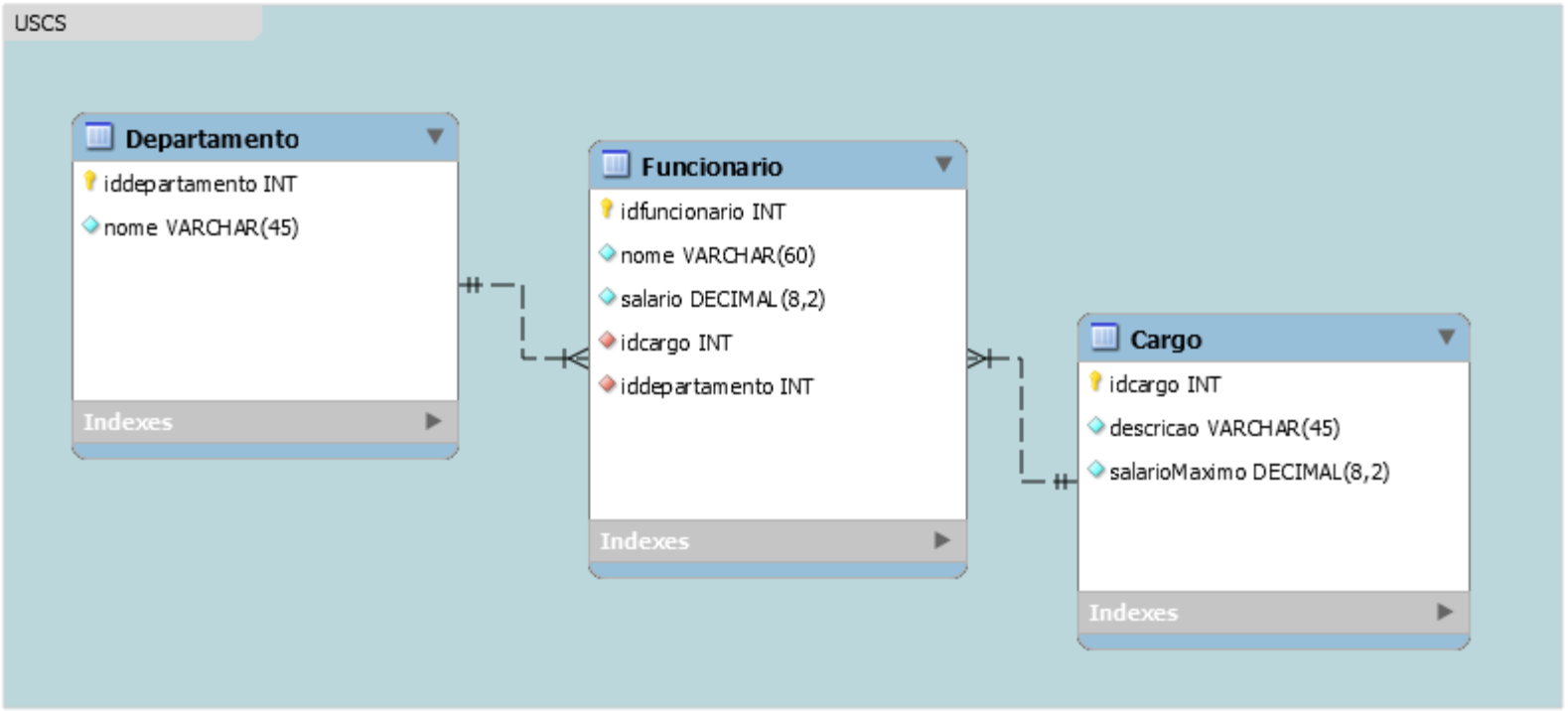


Diagrama Entidade Relacionamento

Este é o diagrama entidade relacionamento utilizado para essa unidade!



Triggers

```
CREATE DATABASE funcionario;

USE funcionario;

CREATE TABLE  Cargo (
  idcargo INT(11) NOT NULL AUTO_INCREMENT,
  descricao VARCHAR(45) NOT NULL,
  salarioMaximo DECIMAL(8,2) NOT NULL,
  PRIMARY KEY (idcargo));

CREATE TABLE  Departamento (
  iddepartamento INT(11) NOT NULL AUTO_INCREMENT,
  nome VARCHAR(45) NOT NULL,
  PRIMARY KEY (iddepartamento));

CREATE TABLE Funcionario (
  idfuncionario INT(11) NOT NULL AUTO_INCREMENT,
  nome VARCHAR(60) NOT NULL,
  salario DECIMAL(8,2) NOT NULL,
  idcargo INT(11) NOT NULL,
  iddepartamento INT(11) NOT NULL,
  PRIMARY KEY (idfuncionario),
  INDEX fk_Funcionario_Cargo_idx (idcargo ASC),
  INDEX fk_Funcionario_Departamento1_idx (iddepartamento ASC),
  CONSTRAINT fk_Funcionario_Cargo FOREIGN KEY (idcargo)
    REFERENCES Cargo (idcargo),
  CONSTRAINT fk_Funcionario_Departamento1 FOREIGN KEY (iddepartamento)
    REFERENCES Departamento (iddepartamento));
```

FuncionarioCriacaoDB.sql

Triggers

- ✓ **Triggers** são **stored procedures** que são executadas em **resposta** a algum tipo de **evento** que ocorre no banco de dados;
- ✓ O disparo de um trigger pode estar associado à uma resposta de um comando **DDL** (**Insert**, **Update** ou **Delete**);
- ✓ **Triggers** se constituem em poderosos mecanismos de integridade dos dados, assim como são úteis para automatizar certas operações no banco de dados, por exemplo, audit **logging**.

Criação de Triggers

```
CREATE TRIGGER trigger_name
  {BEFORE|AFTER}
  {UPDATE|INSERT|DELETE}
ON table_name
FOR EACH ROW
trigger_statements
```

```
CREATE TRIGGER trigger_name
  {BEFORE|AFTER}
  {UPDATE|INSERT|DELETE}
ON table_name
FOR EACH ROW
trigger_statements
```

Criação de Triggers

- ✓ **trigger_name** , nome associado ao Trigger
- ✓ **BEFORE|AFTER**, especifica se o trigger é executado antes ou após a execução do comando DDL que originou o trigger;
- ✓ **UPDATE|INSERT|DELETE**, define o comando DDL associado ao trigger;
- ✓ **On table_name**, associa o trigger a uma determinada tabela;
- ✓ **FOR EACH ROW**, indica que o trigger será executado uma vez para toda linha da tabela;
- ✓ **trigger_statements**, define o comando ou bloco de comandos que serão executados quando o trigger for invocado.

Triggers - Estados

- ✓ Um registro persistido em um banco de dados pode assumir em um determinado momento dois estados:


```
SELECT d.iddepartamento, d.nome
FROM departamento d;
```

	iddepartamento	nome
▶	1	Suporte Técnico
	2	Infraestrutura
	3	Segurança da Informação
	4	Desenvolvimento

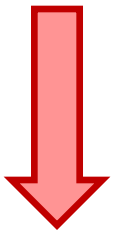
```
UPDATE departamento
SET nome = 'Desenvolvimento Mobile'
WHERE iddepartamento = 4;
```

1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Triggers – Estados

- ✓ Ao executar um evento de **inserção**, **atualização** ou **deleção** em um banco de dados relacional é possível em um intervalo de tempo controlar os **estados** dos registros envolvidos:

```
UPDATE departamento
SET nome = 'Desenvolvimento Mobile'
WHERE iddepartamento = 4;
```



old.nome = ‘Desenvolvimento’

new.nome = ‘Desenvolvimento Mobile’

Triggers – Estados

- ✓ A existência de ambos estados está relacionada à operação em execução, no caso da instrução **SQL Update** é natural a existência de ambos;
- ✓ Agora avalie a condição dos estados das instruções **Insert** e **Delete**:

Quais estados estão disponíveis em cada uma delas?

```
INSERT INTO departamento (nome)
VALUES('Desenvolvimento Web');
```

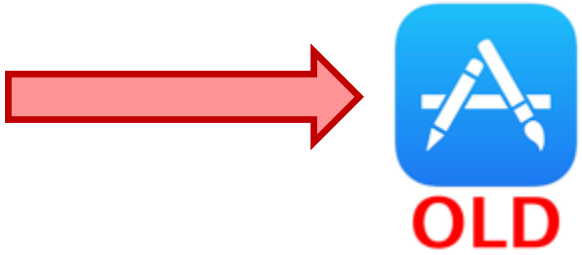
```
DELETE FROM departamento
WHERE iddepartamento = 5;
```

Triggers - Estados

Quais estados estão disponíveis em cada uma delas?

```
INSERT INTO departamento (nome)
VALUES('Desenvolvimento Web');
```


```
DELETE FROM departamento
WHERE iddepartamento = 5;
```



Triggers

- ✓ Assim, ao se executar um evento de inserção, atualização ou deleção em um banco de dados relacional é possível em um determinado intervalo de tempo controlar os estados dos registros envolvidos.

- ✓ Esse intervalo de tempo ou momento possui dois marcadores, sendo eles o **antes** e **depois** da efetivação da transação.

Antes = ***Before***

Depois = ***After***

Triggers x Stored Procedures

- ✓ A principal diferença entre um gatilho e um procedimento armazenado que é **um gatilho é chamado automaticamente quando um evento de modificação de dados** ocorre em uma tabela, enquanto que uma stored procedure deve ser chamada de forma explícita.

Triggers – Vantagens e Desvantagens

Vantagens

- ✓ Proporcionam uma forma alternativa de verificar a integridade dos dados;
- ✓ Tratamento da lógica de negócios na camada de banco de dados;
- ✓ Úteis para criar dados de auditoria em alterações de base de dados.

Desvantagem

- ✓ Podem acarretar sobrecarga do servidor de banco de dados.

Triggers

Triggers estão associados à três instruções **SQL**:

- ✓ **INSERT**
- ✓ **UPDATE**
- ✓ **DELETE**

E para cada uma destas instruções existem dois **momentos** distintos:

- ✓ **BEFORE**
- ✓ **AFTER**

Assim, para cada tabela é possível a definição de no máximo seis **eventos**:

BEFORE INSERT	**BEFORE UPDATE**	**BEFORE DELETE**
AFTER INSERT	**AFTER UPDATE**	**AFTER DELETE**

Triggers

BEFORE INSERT => Ativado **antes** dos dados serem **inseridos** na tabela .

AFTER INSERT => Ativado **após** a **inserção** dos dados na tabela.

BEFORE UPDATE => Ativado **antes** da **atualização** dos dados na tabela.

AFTER UPDATE => Ativado **após** a **atualização** dos dados na tabela.

BEFORE DELETE => Ativado **antes** de os dados serem **removidos**.

AFTER DELETE => Ativado **após** os dados serem **removidos**.

Triggers - Sintaxe

- ✓ O tempo ou momento de execução do gatilho deve possuir exclusivamente um dos dois valores válidos (BEFORE ou AFTER).

- ✓ O evento associado a execução do gatilho deve possuir exclusivamente um dos três valores válidos (INSERT, UPDATE ou DELETE).

Triggers - Exemplo

Para exemplificar a utilização de um trigger, a seguinte tabela deve ser criada com a finalidade de controlar um **log** de auditoria sobre a tabela **departamento**.

```
CREATE TABLE  LogDepartamento (
  idsequencial INT(11) NOT NULL AUTO_INCREMENT,
  iddepartamento INT(11) NOT NULL,
  nomeAntigo VARCHAR(45),
  nomeNovo VARCHAR(45),
  dhEvento DATETIME NOT NULL,
  tipoEvento VARCHAR(20) NOT NULL,
  usuarioResponsavel VARCHAR(60) NOT NULL,
PRIMARY KEY (idsequencial));
```

Triggers - Exemplo

Ao inserir um novo **Departamento**, o trigger abaixo é responsável por inserir um registro na tabela **"LogDepartamento",** registrando o código do departamento, o nome antigo, nome novo, data e hora do evento, tipo do evento e o usuário responsável pela ação.

```
DELIMITER $$
CREATE TRIGGER tg_AfterDepartamentoUpdate AFTER UPDATE
ON Departamento
FOR EACH ROW
BEGIN
  INSERT INTO LogDepartamento
  (iddepartamento, nomeAntigo, nomeNovo, dhEvento,
   tipoEvento, usuarioResponsavel)
  VALUES
  (new.iddepartamento, old.nome, new.nome, Now(),
   'Atualização', User());
END$$
DELIMITER ;
```

Triggers - Exemplo

```
UPDATE departamento
SET nome = 'Desenvolvimento Mobile Android'
WHERE iddepartamento = 4;
```

idsequencial	iddepartamento	nomeAntigo	nomeNovo	dhEvento	tipoEvento	usuarioResponsavel
1	4	Desenvolvimento Mobile	Desenvolvimento Mobile Android	2017-09-08 11:55:27	Atualização	root@localhost

```
UPDATE departamento
SET nome = 'Desenvolvimento Mobile IOS'
WHERE iddepartamento = 4;
```

idsequencial	iddepartamento	nomeAntigo	nomeNovo	dhEvento	tipoEvento	usuarioResponsavel
1	4	Desenvolvimento Mobile	Desenvolvimento Mobile Android	2017-09-08 11:55:27	Atualização	root@localhost
2	4	Desenvolvimento Mobile Android	Desenvolvimento Mobile IOS	2017-09-08 11:56:18	Atualização	root@localhost

```
UPDATE departamento
SET nome = 'Desenvolvimento Mobile Windows Phone'
WHERE iddepartamento = 4;
```

idsequencial	iddepartamento	nomeAntigo	nomeNovo	dhEvento	tipoEvento	usuarioResponsavel
1	4	Desenvolvimento Mobile	Desenvolvimento Mobile Android	2017-09-08 11:55:27	Atualização	root@localhost
2	4	Desenvolvimento Mobile Android	Desenvolvimento Mobile IOS	2017-09-08 11:56:18	Atualização	root@localhost
3	4	Desenvolvimento Mobile IOS	Desenvolvimento Mobile Windows Phone	2017-09-08 11:56:54	Atualização	root@localhost

Exclusão de Trigger

- ✓ O procedimento para **exclusão** de um **trigger** é similar ao procedimento de uma **Stored Procedure** ou **Function** .

```
DROP TRIGGER tg_AfterDepartamentoUpdate;
```

Trigger – Exercício 1

- ✓ Recrie o trigger **tg_AfterDepartamentoUpdate** da mesma forma como foi demostrado. O evento deve estar associado a instrução **UPDATE** e o momento de execução é **após** a atualização do registro.

- ✓ Entretanto o log da alteração só deve ocorrer se o novo nome do departamento for diferente do antigo.

Trigger – Exercício 1 - Resolução

Recrie a trigger tg_AfterDepartamentoUpdate da mesma forma como foi demostrado. O evento deve estar associado a instrução UPDATE e o momento de execução é após a atualização do registro.

Entretanto o log da alteração só deve ocorrer se o novo nome do departamento for diferente do antigo.

```
DELIMITER $$
CREATE TRIGGER tg_AfterDepartamentoUpdate AFTER UPDATE
ON Departamento
FOR EACH ROW
BEGIN
  IF (old.nome <> new.nome) THEN
    INSERT INTO LogDepartamento
    (iddepartamento, nomeAntigo, nomeNovo, dhEvento,
     tipoEvento, usuarioResponsavel)
    VALUES
    (new.iddepartamento, old.nome, new.nome, Now(),
     'Atualização', User());
  END IF;
END$$
DELIMITER ;
```

Trigger – Exercício 2

- ✓ Desenvolva mais 2 triggers para a tabela "**departamento**" correlacionados com os eventos de **inserção** e **deleção**.
- ✓ Ambas devem criar um **log** do registro na tabela "**LogDepartamento**".
- ✓ Para confirmar a conclusão do exercício realize a inclusão de dois departamentos e a exclusão de um deles.
- ✓ Confirme a criação dos logs realizando uma consulta na tabela "**LogDepartamento**".

Trigger – Exercício 2 - Resolução

Desenvolva mais 2 triggers para a tabela “departamento” correlacionadas com os eventos de inserção e deleção.

Ambas devem criar um log do registro na tabela “LogDepartamento”.

Para confirmar a conclusão do exercício realize a inclusão de dois departamentos e a exclusão de um deles.

Confirme a criação dos logs realizando uma consulta na tabela “LogDepartamento”.

```
DELIMITER $$
CREATE TRIGGER tg_AfterDepartamentoInsert AFTER INSERT
ON Departamento
FOR EACH ROW
BEGIN
  INSERT INTO LogDepartamento
  (iddepartamento, nomeAntigo, nomeNovo, dhEvento,
   tipoEvento, usuarioResponsavel)
  VALUES
  (new.iddepartamento, null, new.nome, Now(), 'Inserção', User());
END
DELIMITER ;
```

Trigger – Exercício 2 - Resolução

Desenvolva mais 2 triggers para a tabela “departamento” correlacionadas com os eventos de inserção e deleção.

Ambas devem criar um log do registro na tabela “LogDepartamento”.

Para confirmar a conclusão do exercício realize a inclusão de dois departamentos e a exclusão de um deles.

Confirme a criação dos logs realizando uma consulta na tabela “LogDepartamento”.

```
DELIMITER $$
CREATE TRIGGER tg_AfterDepartamentoDelete AFTER DELETE
ON Departamento
FOR EACH ROW
BEGIN
  INSERT INTO LogDepartamento
  (iddepartamento, nomeAntigo, nomeNovo, dhEvento,
   tipoEvento, usuarioResponsavel)
  VALUES
  (old.iddepartamento, old.nome, Null, Now(), 'Exclusão', User());
END$$
DELIMITER ;
```

Trigger

***** ATENÇÃO *****

Antes de prosseguir realize a exclusão das três triggers criadas até o momento:

```
DROP TRIGGER tg_AfterDepartamentoInsert;
DROP TRIGGER tg_AfterDepartamentoUpdate;
DROP TRIGGER tg_AfterDepartamentoDelete;
```

Trigger – Exercício 3

- ✓ Desenvolva uma Stored Procedure chamada **sp_RestauraDepartamento**, essa Stored Procedure recebe como parâmetro de entrada um código que representa o código de um departamento.

- ✓ Verifique se existe um departamento com esse código cadastrado na tabela "**Departamento**", caso não exista, verifique se existe um departamento com esse código na tabela "**LogDepartamento**" se existir insira novamente esse departamento na tabela **departamento** com o mesmo código e nome que ele possuía.

Trigger – Exercício 3 - Resolução

Desenvolva uma Stored Procedure chamada **sp_RestauraDepartamento**, essa Stored Procedure recebe como parâmetro de entrada um código que representa o código de um departamento.

Verifique se existe um departamento com esse código cadastro na tabela "**Departamento**", caso não exista, verifique se existe um departamento com esse código na tabela "**LogDepartamento**" se existir insira novamente esse departamento na tabela departamento com o mesmo código e nome que ele possuía.

```
DELIMITER $$
CREATE PROCEDURE sp_RestauraDepartamento(IN varIdDepart INT)
BEGIN
  INSERT INTO departamento (iddepartamento, nome)
  (SELECT ld.iddepartamento, ld.nomeAntigo
   FROM logdepartamento ld
   WHERE ld.iddepartamento = varIdDepart AND
       NOT Exists (SELECT d.iddepartamento
                   FROM departamento d
                   WHERE d.iddepartamento = ld.iddepartamento)
   ORDER BY dhEvento DESC
   LIMIT 1
  );
END$$
DELIMITER ;
```

Trigger – Exercício 4

- ✓ Utilizando **triggers**, desenvolva um sistema de **log** para a tabela de **cargos** englobando as três operações a seguir:
  - *Insert;*
  - *Update;*
  - *Delete;*
- ✓ Para concluir esse exercício será necessária a criação de uma nova tabela e três novos **triggers**.