Utilizando Views, Stored Procedures e Triggers

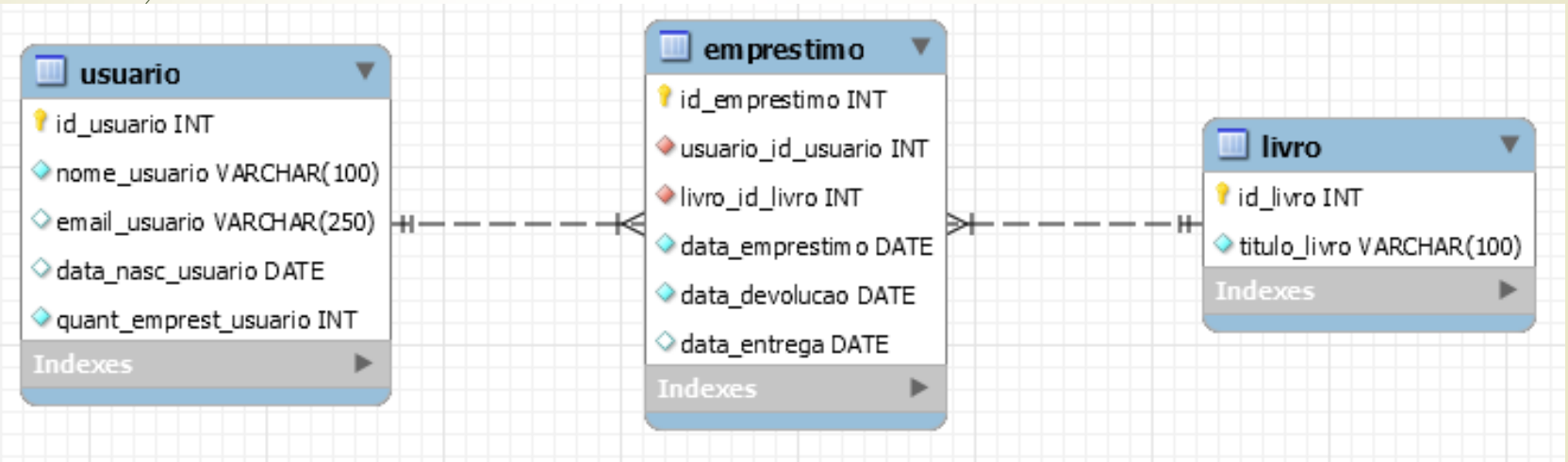
Crédito: Daniel Cosme Mendonça Maia

Prática de Laboratório

- Nesta prática de laboratório, utilizaremos o MySQL 5.5 ou superior;
- Vamos criar um banco de dados de uma "biblioteca" para simular os comandos aprendidos na aula anterior;
- Este modelo é extremamente simplificado e está voltado somente para experimentação didática.

Modelo Relacional

- Crie um banco de dados com o nome “biblio”, a partir do modelo abaixo:



Criando as estruturas

```
CREATE DATABASE biblio;

USE biblio;

CREATE TABLE livro(
    id_livro int not null primary key auto_increment,
    titulo_livro varchar(100) not null);
```

Criando as estruturas

```
CREATE TABLE usuario(
    id_usuario int not null primary key auto_increment,
    nome_usuario varchar(100) not null,
    email_usuario varchar(250),
    data_nasc_usuario date,
    quant_emprest_usuario int not null default 0);
CREATE TABLE emprestimo(
    id_emprestimo int not null primary key auto_increment,
    usuario_id_usuario int not null,
    livro_id_livro int not null,
    data_emprestimo date not null,
    data_devolucao date not null,
    data_entrega date,
    foreign key(usuario_id_usuario) references usuario(id_usuario),
    foreign key(livro_id_livro) references livro(id_livro));
```

Inserindo dados

```
INSERT INTO livro (titulo_livro) VALUES
    ("Bagagem"),
    ("O Cortiço"),
    ("Lira dos Vinte Anos"),
    ("Quarup"),
    ("O Tronco"),
    ("A escrava Isaura"),
    ("O Pagador de Promessas"),
    ("O que é isso, Companheiro?"),
    ("Vidas Secas"),
    ("Grande Sertão Veredas");
```

Inserindo dados

```
INSERT INTO usuario
    (nome_usuario, email_usuario, data_nasc_usuario)
    VALUES
    ("João Silva", "joao@email.com", "1992-08-09"),
    ("Maria Mota", "maria@provedor.net", "1984-05-17"),
    ("Eduardo Cançado", "edu@email.com", "1996-02-23"),
    ("Silvia Alencar", "silvia@provedor.net", "1973-09-20"),
    ("Gabriela Medeiros", "gabi@email.com", "1993-01-10"),
    ("Karina Silva", "karin@email.com", "1995-03-25");
```

Inserindo dados

```
INSERT INTO emprestimo (usuario_id_usuario, livro_id_livro,
    data_emprestimo, data_devolucao, data_entrega) VALUES
    (1, 4, "2014-01-15", "2016-02-15", "2016-02-10"),
    (3, 2, "2014-02-22", "2016-03-22", "2016-03-21"),
    (2, 6, "2014-02-22", "2016-03-22", null),
    (2, 8, "2014-03-21", "2016-04-21", null),
    (1, 10, "2014-03-23", "2016-04-23", "2016-03-28”),
    (4, 2, "2014-03-23", "2016-04-23", null),
    (4, 7, "2014-03-23", "2016-04-23", null),
    (5, 3, "2014-03-24", "2016-04-24", null),
    (5, 9, "2014-03-24", "2016-04-24", null),
    (5, 1, "2014-03-24", "2016-04-24", null),
    (6, 3, "2014-03-01", "2016-04-01", "2014-03-28);
```

Criando Views (Visões)

Criando uma visão (contatos_usuarios) que mostre apenas o nome e e-mail dos usuários da biblioteca

```
CREATE VIEW contatos_usuarios AS
SELECT nome_usuario, email_usuario FROM
                                    usuario;
```

Visualizando dados de uma View

Para visualizar os dados de uma View basta utilizar uma consulta SELECT, como o exemplo abaixo:

```
SELECT * FROM contatos_usuarios;
```

Criando Views (Visões)

#Criando uma visão (aniversarios_usuarios) que mostre apenas o nome e data de nascimento dos usuários da biblioteca

```
CREATE VIEW aniversarios_usuarios AS
SELECT nome_usuario, data_nasc_usuario FROM usuario;
```

Criando Views (Visões)

#Criando uma visão (emprestimos_realizados) que mostre o nome do usuario, título do livro, data de emprestimo, devolução e entrega de todos os empréstimos realizados na bilioteca

```
CREATE VIEW emprestimos_realizados AS
SELECT usuario.nome_usuario usuario, livro.titulo_livro livro,
    emprestimo.data_emprestimo emprestimo,
    emprestimo.data_devolucao devolucao,
    emprestimo.data_entrega entrega
FROM usuario, livro, emprestimo
WHERE emprestimo.usuario_id_usuario = usuario.id_usuario
    AND emprestimo.livro_id_livro = livro.id_livro;
```

Criando Views (Visões)

#Criando uma visão (emprestimos_atrasados) que mostre todos os empréstimos atrasados com base na visão empréstimos realizados

```
CREATE VIEW emprestimos_atrasados AS
SELECT * FROM emprestimos_realizados WHERE devolucao < now() AND entrega is null;
```

Criando Views (Visões)

#Criando uma visão (quantidade_emprestimos_usuario) que mostre a quantidade de emprestimos realizados por cada usuario

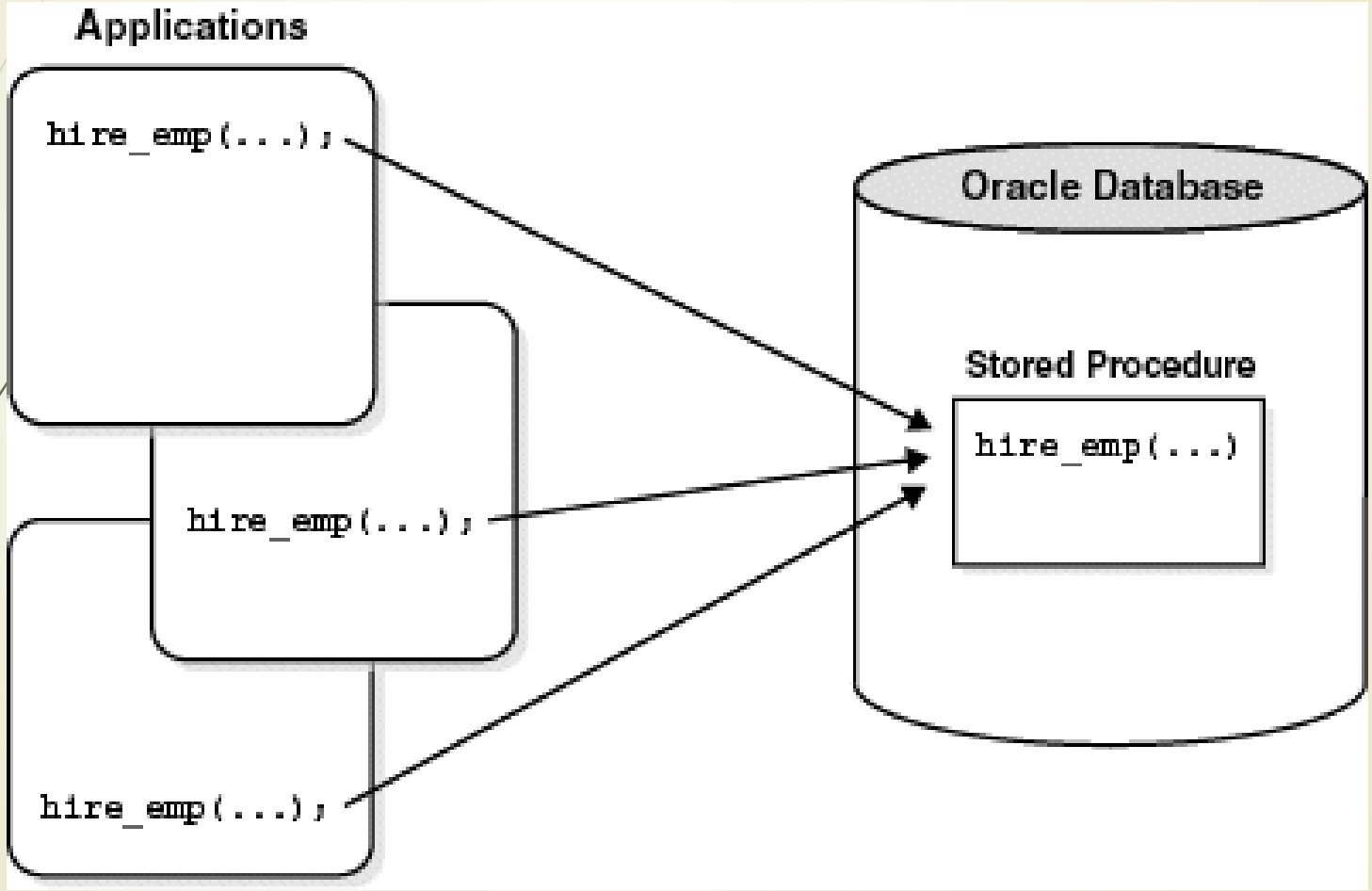
```
CREATE VIEW quantidade_emprestimos_usuario AS
SELECT usuario, count(*) num_livros FROM emprestimos_realizados GROUP BY usuario;
```

Alterando dados na View

#Alterando dados na visão

```
UPDATE emprestimos_realizados
 SET entrega = "2014-09-29"
WHERE usuario = "Karina Silva";
```

Stored Procedures (Procedimentos Armazenados)



Stored Procedures

#Criando um procedimento de cadastro de empréstimos, que receba o código do usuário e do livro como parâmetros de entrada. A data de empréstimo e devolução devem ser definidas automaticamente pelo banco de dados, por exemplo, data de empréstimo = hoje, e a data de devolução = 30 dias após o a data de empréstimo.

Dica: Usar as funções curdate() e adddate() do MySQL

EMPRESTAR

```
delimiter $$
 CREATE PROCEDURE emprestar(IN usuario INT, IN livro INT)
BEGIN
    SET @dt_emprestimo = curdate();
    SET @dt_devolucao  = adddate(@dt_emprestimo, 30);
    INSERT INTO emprestimo (usuario_id_usuario, livro_id_livro,
    data_emprestimo, data_devolucao)
   VALUES (usuario, livro, @dt_emprestimo, @dt_devolucao);
END $$
 delimiter ;
```

Stored Procedure Emprestar

Para utilizar um procedimento armazenado basta utilizar o comando CALL para realizar a "chamada" do procedimento. Exemplo:

CALL emprestar(3,8);

Emprestar

```
delimiter $$

CREATE PROCEDURE Emprestar  (IN usuario INT, IN livro INT, OUT
    devolucao DATE)
BEGIN
    SET @dt_emprestimo = curdate();
    SET @dt_devolucao  = adddate(@dt_emprestimo, 30);
    SELECT @dt_devolucao INTO devolucao;
    INSERT INTO emprestimo (usuario_id_usuario, livro_id_livro,
    data_emprestimo, data_devolucao) VALUE (usuario, livro,
    @dt_emprestimo, @dt_devolucao);
END $$
delimiter ;
```

Stored Procedures Devolver

#Criando um procedimento para devolver um livro, que receba o código do empréstimo como parâmetro de entrada. A data de entrega deve ser preenchida automaticamente pelo banco de dados (hoje).

Dica: Usar a função curdate() do MySQL

Devolver

```
delimiter $$

CREATE PROCEDURE Devolver(IN emprestimo INT)
BEGIN
    SET @dt_entrega = curdate();

    UPDATE emprestimo SET data_entrega = @dt_entrega
    WHERE id_emprestimo = emprestimo;
END $$

delimiter ;
```

Functions Contar Emprestimos

#Criando uma função que retorne a quantidade vezes que determinado livro foi emprestado. É necessário receber o código do livro como parâmetro de entrada.

Dica: Usar a função count() do MySQL

Função Contar_emprestimos

```
delimiter $$
 CREATE FUNCTION  Contar_emprestimos  (livro INT)
    RETURNS INT DETERMINISTIC
BEGIN
    DECLARE num INT;
    SELECT count(*) INTO num FROM emprestimo
    WHERE livro_id_livro = livro;
    RETURN num;
END $$
delimiter ;
```

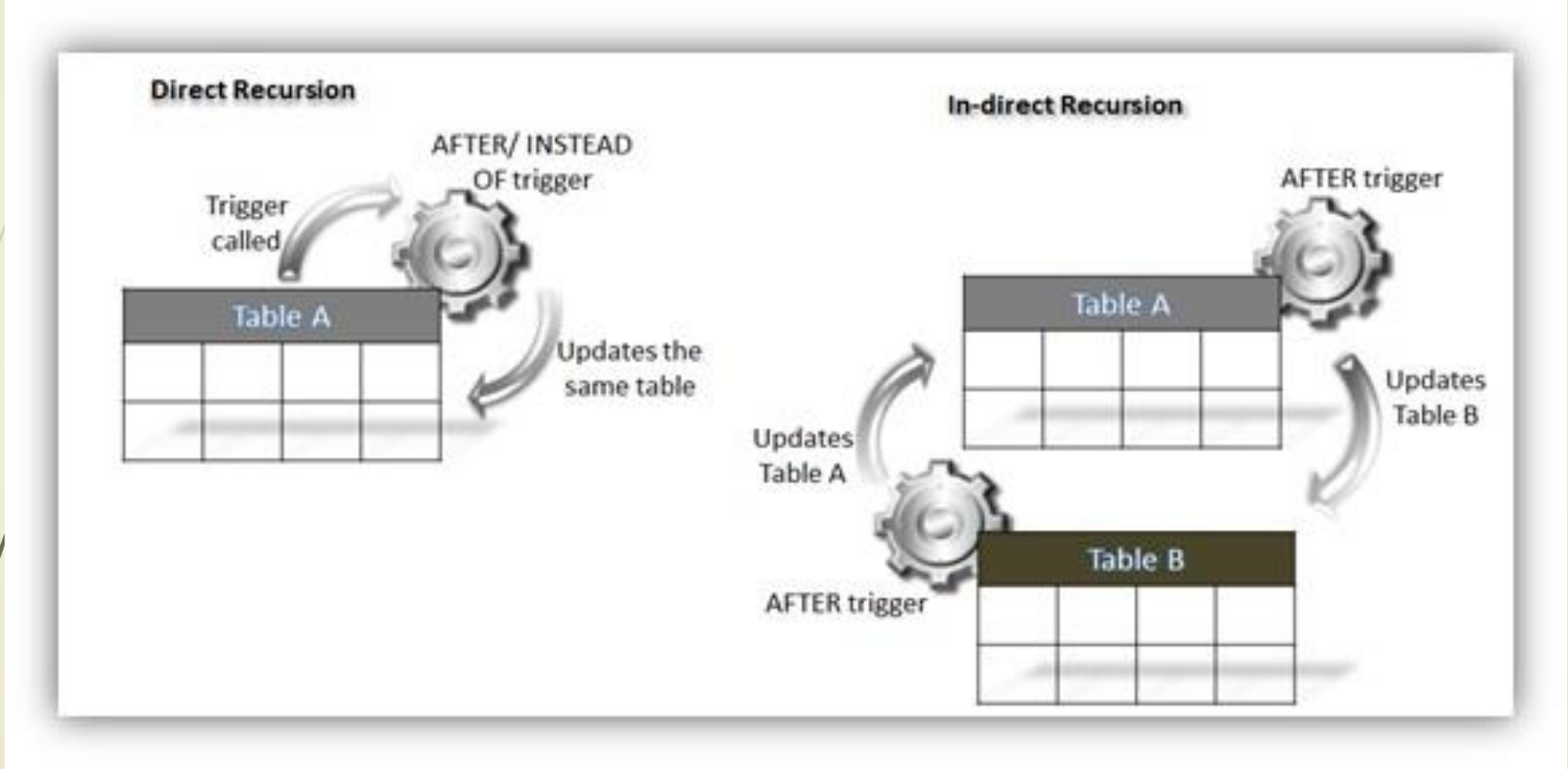
Chamada da função

As funções são "chamadas" através de uma declaração SQL.

Exemplo:

```
SELECT Contar_emprestimos(2);
```

Triggers (Gatilhos)



Triggers (Gatilhos)

#Criando um gatilho que incremente a quantidade de livros emprestados para um determinado usuário (campo da tabela usuário). O gatilho deve ser ativado toda vez que ocorrer uma inserção na tabela de empréstimo.

Triggers (Gatilhos)

```
delimiter $$
CREATE TRIGGER inc_quant_emprestimo AFTER INSERT ON
    emprestimo
FOR EACH ROW
BEGIN
    UPDATE usuario
     SET quant_emprest_usuario = quant_emprest_usuario + 1
    WHERE id_usuario = NEW.usuario_id_usuario;
END $$
delimiter ;
```