FACET
PÓS-GRADUAÇÃO

Projeto de Banco de Dados e OO .NET

Banco de Dados Relacionais

Prof: Foad Mobini Kesheh
foadmk@gmail.com

Banco de Dados

- São coleções organizadas de dados que se relacionam de forma a criar algum sentido (Informação) representando algum aspecto da realidade.
- Database-management system (DBMS):
  - MySQL, PostgreSQL, MongoDB, MariaDB, Microsoft SQL Server, Oracle, Sybase, SAP HANA, MemSQL, SQLite and IBM DB2

Bancos de Dados Relacionais

- Em definição simplificada, o modelo baseia-se em dois conceitos: conceito de entidade e relação
  - Uma entidade é um elemento caracterizado pelos dados que são recolhidos na sua identificação vulgarmente designado por tabela. Na construção da tabela identificam-se os dados da entidade. A atribuição de valores a uma entidade constrói um registro da tabela.
  - A relação determina o modo como cada registro de cada tabela se associa a registros de outras tabelas.

Exemplo de Dado Relacional

SQL

Structured Query Language, ou **Linguagem de Consulta Estruturada** ou **SQL**, é a linguagem de pesquisa declarativa padrão para banco de dados relacional (base de dados relacional). Muitas das características originais do SQL foram inspiradas na álgebra relacional.

Subconjuntos do SQL:

- **DDL - Linguagem de Definição de Dados**
- **DCL - Linguagem de Controle de Dados**
- **DML - Linguagem de Manipulação de Dados**
- **DQL - Linguagem de Consulta de Dados**
- **DTL - Linguagem de Transação de Dados**

SQL – Cláusulas

- As cláusulas são condições de modificação utilizadas para definir os dados que deseja selecionar ou modificar em uma consulta.
  - FROM – Utilizada para especificar a tabela que se vai selecionar os registros.
  - WHERE – Utilizada para especificar as condições que devem reunir os registros que serão selecionados.
  - GROUP BY – Utilizada para separar os registros selecionados em grupos específicos.
  - HAVING – Utilizada para expressar a condição que deve satisfazer cada grupo.
  - ORDER BY – Utilizada para ordenar os registros selecionados com uma ordem especifica.
  - DISTINCT – Utilizada para selecionar dados sem repetição.
  - UNION - combina os resultados de duas consultas SQL em uma única tabela para todas as linhas correspondentes.

Operadores Lógicos

- AND – E lógico. Avalia as condições e devolve um valor verdadeiro caso ambos sejam corretos.
- OR – OU lógico. Avalia as condições e devolve um valor verdadeiro se algum for correto.
- NOT – Negação lógica. Devolve o valor contrário da expressão.

Operadores relacionais

- >, >=, =, <=, <, <> – Utilizados para comparação
- BETWEEN – Utilizado para especificar valores dentro de um intervalo fechado.
- LIKE – Utilizado na comparação de um modelo e para especificar registros de um banco de dados. "Like" + extensão % significa buscar todos resultados com o mesmo início da extensão.
- IN - Utilizado para verificar se o valor procurado está dentro de uma lista. Ex.: valor IN (1,2,3,4).

Data Definition Language (DDL)

- Usado para criar e modificar a estrutura dos objetos de banco de dados.
- Principais: CREATE, ALTER, DROP

Exemplos DDL

```
CREATE TABLE Empregados (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(30) NOT NULL,
sobrenome VARCHAR(30) NOT NULL,
email VARCHAR(50),
data_registro TIMESTAMP);

ALTER TABLE `empregados`
CHANGE COLUMN `sobrenome` `sobrenome` VARCHAR(60) NOT NULL AFTER `nome`,
ADD COLUMN `status` VARCHAR(10) NOT NULL AFTER `sobrenome`;

RENAME TABLE `empregados` TO `terceirizados`;

TRUNCATE TABLE terceirizados;

DROP TABLE terceirizados;

CREATE USER 'novo_usuario'@'localhost' IDENTIFIED BY 'senha';

CREATE DATABASE teste;
```

Exercício 1

- Criar um banco de dados chamado “posfacet”
- Criar no banco de dados “posfacet” duas tabelas:
  - Tabela 1:
    - Nome: “gostam_futebol”
    - Coluna 1: “id_futebol” do tipo INT primary key com autoincremento setado
    - Coluna 2: “nome” do tipo VARCHAR, tamanho 50
  - Tabela 2:
    - Nome: “gostam_volei”:
    - Coluna 1: “id_volei” do tipo INT primary key com autoincremento setado
    - Coluna 2: “nome” do tipo VARCHAR, tamanho 50

Data Control Language (DCL)

- Usado para criar permissões e integridade referencial e também é usado para controlar o acesso a banco de dados
- Principais: GRANT, REVOKE

Exemplos DCL

- **GRANT SELECT, DELETE, UPDATE, INSERT ON *.* TO** 'novo_usuario'@'localhost';
- **REVOKE SELECT, DELETE, INSERT, UPDATE ON *.* FROM** 'novo_usuario'@'localhost';
- **GRANT SELECT, INSERT, DELETE, UPDATE ON** `teste`**.* TO** 'novo_usuario'@'localhost';
- **GRANT ALL ON *.* TO** 'novo_usuario'@'localhost‘;
- FLUSH PRIVILEGES;

Exercício 2

- Criar um usuário “pos_facet” com uma senha
- Dar acesso total a esse usuário apenas ao banco de dados “posfacet”

Data Manipulation Language (DML)

- Usado para armazenar, modificar, apagar, inserir e atualizar dados no banco de dados
- Principais: INSERT, UPDATE, DELETE

INSERT

```
INSERT INTO `gostam_volei` (`id_volei`, `nome`) VALUES (1, 'Lucia');
INSERT INTO `gostam_volei` (`id_volei`, `nome`) VALUES (2, 'Cleber');
INSERT INTO `gostam_volei` (`id_volei`, `nome`) VALUES (3, 'Maria');
INSERT INTO `gostam_volei` (`id_volei`, `nome`) VALUES (4, 'João');

INSERT INTO `gostam_futebol` (`id_futebol`, `nome`) VALUES
(1, 'João'),
(2, 'Maria'),
(3, 'Pedro'),
(4, 'Carlos');

INSERT INTO `gostam_futebol` (`nome`) SELECT nome FROM jogadores_futebol;
```

UPDATE

```
UPDATE gostam_futebol g SET g.nome = "Carlos Agusto"
WHERE g.id_futebol = 4;

DELETE FROM gostam_futebol WHERE nome LIKE "%Agusto";

DELETE FROM gostam_futebol;  //Deleta todos registros da tabela

TRUNCATE TABLE gostam_futebol;  //Reseta a tabela (deleta todos os registros e seta a chave primaria para iniciar em 1
```

Exercício 3

- Inserir os seguintes registros na tabela gostam_futebol (id_futebol, nome):
  - 1, João
  - 2, Maria
  - 3, Pedro
  - 4, Carlos
- Inserir os seguintes registros na tabela gostam_volei (id_volei, nome):
  - 1, Lucia
  - 2, Cleber
  - 3, Maria
  - 4, João
  - 5, Marcos
- Alterar o registro 5 da tabela gostam_volei, Marcos para 5, Marcos Aurélio e verificar se for alterado.
- Excluir o registro 5 da tabela gostam_volei.
- Obs: Usar o comando **SELECT * FROM [nome_tabela]** para testar

Data Query Language (DQL)

- Usado para consultar dados no banco de dados
- Principais: SELECT

SELECT, JOIN, UNION

```
SELECT * from gostam_futebol f ;

SELECT * FROM gostam_futebol v WHERE v.nome IN (SELECT nome FROM gostam_volei v);

SELECT id_futebol AS id, 'Futebol' AS esporte, nome AS nome FROM gostam_futebol
UNION
SELECT id_volei, 'Volei', nome FROM gostam_volei;

SELECT * from gostam_futebol f  JOIN gostam_volei v on f.nome = v.nome;
SELECT * from gostam_futebol f JOIN gostam_volei v USING (nome);
SELECT * from gostam_futebol f NATURAL JOIN gostam_volei v;

SELECT * from gostam_futebol f LEFT JOIN gostam_volei v on f.nome = v.nome;

SELECT * from gostam_futebol f RIGHT JOIN gostam_volei v on f.nome = v.nome;

SELECT * from gostam_futebol f RIGHT JOIN gostam_volei v on f.nome = v.nome
UNION
SELECT * from gostam_futebol f LEFT JOIN gostam_volei v on f.nome = v.nome

SELECT DISTINCT t1.nome FROM
(SELECT f.nome FROM gostam_futebol f
UNION
SELECT v.nome FROM gostam_volei v) AS t1;
```

Exercício 4

- Selecionar todos nomes da tabela gostam_volei que contém a letra “a” ou “A”;
- Selecionar todos os nomes que aparecem nas tabelas gostam_volei e também na tabela gostam_futebol;
- Selecionar todos os nomes da tabela gostam_volei que não aparecem na tabela gostam_futebol;

Funções de Agregação

- AVG – Utilizada para calcular a média dos valores de um campo determinado.
- COUNT – Utilizada para devolver o número de registros da seleção.
- SUM – Utilizada para devolver a soma de todos os valores de um campo determinado.
- MAX – Utilizada para devolver o valor mais alto de um campo especificado.
- MIN – Utilizada para devolver o valor mais baixo de um campo especificado.

COUNT

```
SELECT COUNT(TABELA_1.nome) FROM
(SELECT id_futebol AS id, 'Futebol' AS esporte, nome AS nome FROM
gostam_futebol
UNION
SELECT id_volei, 'Volei', nome FROM gostam_volei) AS TABELA_1;

SELECT COUNT(DISTINCT(TABELA_1.nome)) FROM
(SELECT id_futebol AS id, 'Futebol' AS esporte, nome AS nome FROM
gostam_futebol
UNION
SELECT id_volei, 'Volei', nome FROM gostam_volei) AS TABELA_1;
```

SUM, MAX, MIN, AVG

```
SELECT AVG(id_futebol) FROM gostam_futebol;
SELECT SUM(id_futebol) FROM gostam_futebol;
SELECT MAX(id_futebol) FROM gostam_futebol;
SELECT MIN(id_futebol) FROM gostam_futebol;
```

Exercício 5

- Selecionar a quantidade de nomes que contém a letra “a” ou “A” na tabela gostam_volei;

- Selecionar a soma da quantidade de caracteres de todos os nomes da tabela gostam_futebol;

ACID

- **ACID** é um conceito que se refere às quatro propriedades de transação de um sistema de banco de dados: **A**tomicidade, **C**onsistência, **I**solamento e **D**urabilidade.
- **Transação***: Uma transação é uma sequência de operações executadas como uma única unidade lógica de trabalho.*
- ***Atomicidade:*** Em uma transação envolvendo duas ou mais partes de informações discretas, ou a transação será executada totalmente ou não será executada, garantindo assim que as transações sejam atômicas.
- ***Consistência:*** A transação cria um novo estado válido dos dados ou em caso de falha retorna todos os dados ao seu estado antes que a transação foi iniciada.
- ***Isolamento:*** Uma transação em andamento mas ainda não validada deve permanecer isolada de qualquer outra operação, ou seja, garantimos que a transação não será interferida por nenhuma outra transação concorrente.
- ***Durabilidade:*** Dados validados são registados pelo sistema de tal forma que mesmo no caso de uma falha e/ou reinício do sistema, os dados estão disponíveis em seu estado correto.

Data Transactional Language (DTL)

- Usado para gerenciar diferentes operações que ocorrem dentro de um banco de dados (Mudanças realizadas for DML).
- Se tratando do MySQL, apenas o **InnoDB** suportá transações a nível ACID.
- Exemplos: START TRANSATION, COMMIT, ROLLBACK

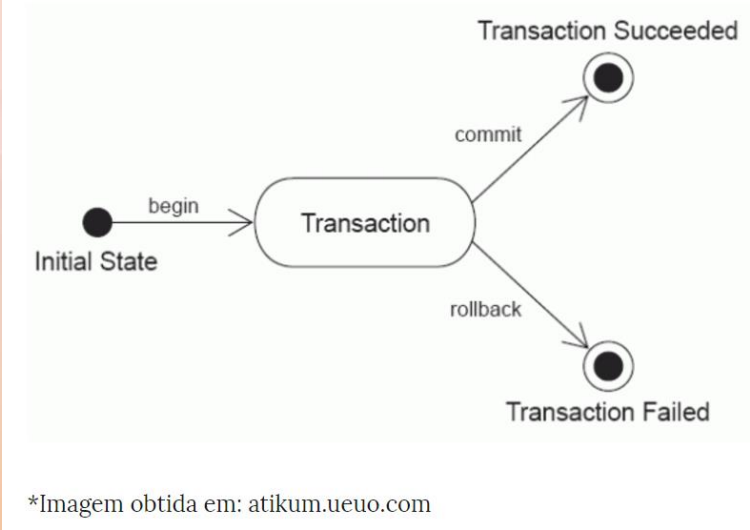


*Imagem obtida em: atikum.ueuo.com

Exemplo

```
START TRANSACTION;
INSERT INTO gostam_futebol VALUES (5, 'Alex');
INSERT INTO gostam_volei VALUES (5, 'Alex');
ROLLBACK; // Não salva nada
COMMIT; // Salva os dados
```

Procedimentos Armazenados

- **Método de programação comum:**
  - Loops
  - Condições
  - Chamadas de Métodos
- **Por que usar procedimentos armazenados:**
  - Em vez de você ter a mesma sequência de processos espalhada pela sua aplicação você pode encapsular tudo num procedimento identificável e acessível no banco de dados;
  - Pode migrar processamentos complexos para o servidor, mantendo o cliente apenas com apresentação de dados.

Procedimentos Armazenados - Vantagens

- **Manutenção fácil**

  Tudo está num único lugar (tabelas e queries). A aplicação não precisa ser alterada em caso de mudanças no esquema do banco de dados;
- **Teste Isolado**
  - Pode ser testado independentemente da aplicação
- **Velocidade e Otimização**
  - Os procedimentos armazenados ficam em cache do servidor
  - Podem ser criados planos de execução independentes da aplicação
- **Segurança**
  - Os usuários do banco de dados podem ter acessos limitados;
  - Podem ser criadas interfaces para que os dados das tabelas fiquem protegidos;
  - Aplicar uma interface de segurança nos dados é mais simples do que aplicar na aplicação em si.

Procedimentos Armazenados - Desvantagens

- **Limitações para desenvolvimento de códigos complexos**
- **Portabilidade limitada**
  - Por exemplo se você mudar seu DBMS você provavelmente terá que rescrever os procedimentos.
- **Testes**
  - Alguns erros só aparecem quando o procedimento é executado
- **Localização das Regras de Negócio**
  - Os procedimentos armazenados não são facilmente agrupáveis e encapsuláveis. Isso significa que as regras de negócio ficam espalhadas em diversos procedimentos.
  - A opinião da maioria dos programadores é que as regras de negócio não devem ficar na camada de dados.
- **Complexidade de Manutenção**
  - Manter dezenas de procedimentos que contém códigos simples pode consumir muito tempo. Como resultado não é recomendável que simples SELECTs sejam encapusulados em stored procedures.
- **Custo**
  - Dependendo da estrutura da empresa, muitas vezes é necessário contratar um profissional extra dedicado ao desenvolvimento no banco de dados já que o desenvolvedor não tem permissão para atuar no banco de dados.

Esqueleto do Procedimento Armazenado

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```

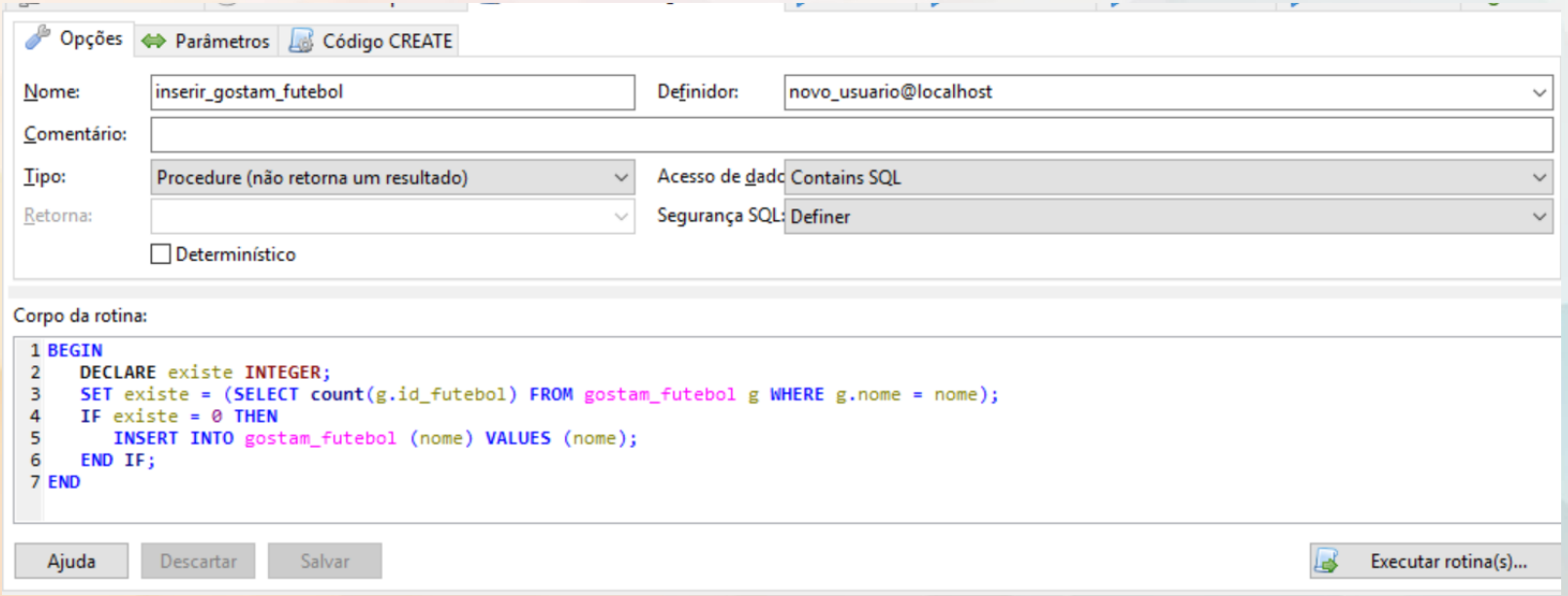
Características

- DETERMINISTIC – Retorna sempre o mesmo valor para determinada entrada
- NOT DETERMINISTIC – Valores diferentes para mesma entrada
- SQL SECURITY DEFINER – Usa os privilégios do usuário que definiu o procedimento
- SQL SECURITY INVOKER – Usa os privilégios do usuário que chamou o procedimento
- *CONTAINS SQL – Não lê e nem escreve em nenhuma tabela*
- *NO SQL – Não contém SQL*
- *READS SQL DATA – Faz apenas SELECT*
- *MODIFIES SQL DATA – Faz INSERT, DELETE ou UPDATE*

Exemplo SQL

```
CREATE DEFINER=`novo_usuario`@`localhost`
PROCEDURE `inserir_gostam_futebol`(IN `nome` VARCHAR(50))
LANGUAGE SQL
NOT DETERMINISTIC
CONTAINS SQL
SQL SECURITY DEFINER
COMMENT ''
BEGIN
    DECLARE existe INTEGER;
    SET existe = (SELECT count(g.id_futebol) FROM gostam_futebol g WHERE g.nome = nome);
    IF existe = 0 THEN
        INSERT INTO gostam_futebol (nome) VALUES (nome);
    END IF;
END
```

No HeidiSQL

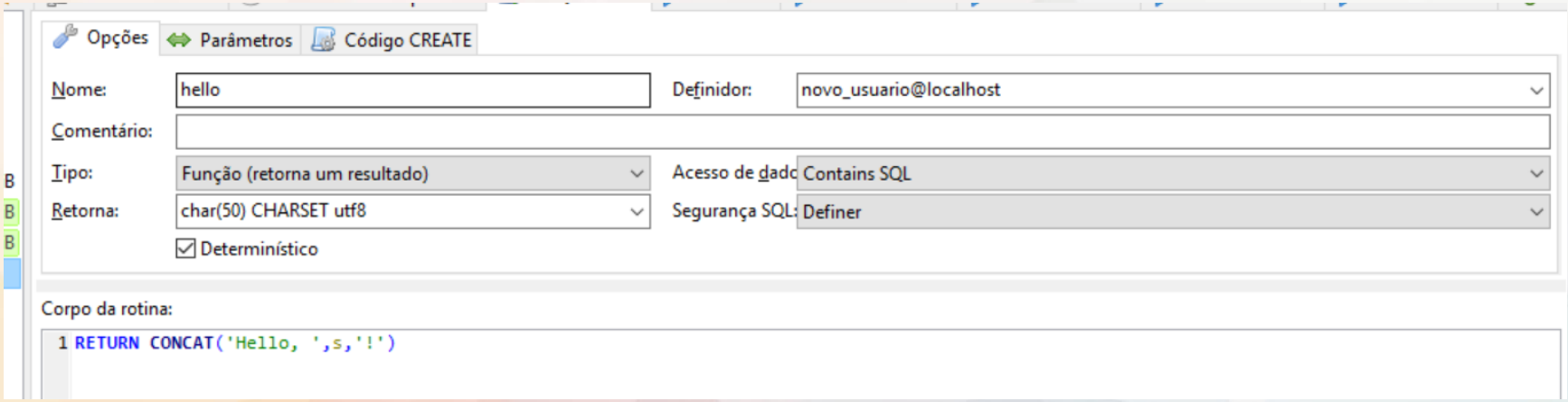


CALL `inserir_gostam_futebol`('Chaves');

Exemplo de Função

```
CREATE DEFINER=`novo_usuario`@`localhost` FUNCTION `hello`(
        `s` CHAR(20)
)
RETURNS char(50) CHARSET utf8
LANGUAGE SQL
DETERMINISTIC
CONTAINS SQL
SQL SECURITY DEFINER
COMMENT ''
RETURN CONCAT('Hello, ',s,'!')
```

No HeidiSQL



SELECT `hello`('World')