INSTITUITO FEDERAL DO PIAUÍ – IFPI
CURSO: Tecnologia em Análise e Desenvolvimento de Sistemas
DISCIPLINA: Banco de Dados
PROFESSOR: Dr. Fábio Gomes

Revisão SQL

CRIAÇÃO DE BANCO DE DADOS

```
CREATE SCHEMA | DATABASE nomeBancoDados;
```

CRIAÇÃO DE TABELAS

```
CREATE TABLE nomeTabela (
       nomeCampo1 tipoCampo1 [CONSTRAINT],
       nomeCampoN tipoCampoN,
       CONSTRAINT nomeConstraint1 tipoConstraint1 descricaoConstraint1,
       CONSTRAINT nomeConstraintN tipoConstraintN descricaoConstraintN
);
```

SQL CONSTRAINTS

Constraints são usadas para restringir a inclusão/alteração de dados nas tabelas do banco de dados e podem ser especificadas na criação (com CREATE TABLE) ou na alteração da tabela (com ALTER TABLE). Principais *constraints*:

- NOT NULL: a coluna não aceita valores nulos;
- UNIQUE: a coluna não deve ter repetições (similar à chave primária);
- PRIMARY KEY: chave primária;
- FOREIGN KEY: chave estrangeira;
- CHECK: checar valores para a(s) coluna(s);
- DEFAULT: inserir um valor default para a coluna, quando o valor não é especificado.

```
CREATE TABLE artista (
       codigoArtista int,
       CPF int,
       codigoCD int NOT NULL,
       salario double,
       dataNascimento date DEFAULT '1950-01-01',
       CONSTRAINT uniqueCPF UNIQUE (cpf),
       CONSTRAINT PKcodigoArtista PRIMARY KEY (codigoArtista),
       CONSTRAINT FKcodigoCD FOREIGN KEY (codigoCD) REFERENCES cd (codigoCD),
       CONSTRAINT checkSalario CHECK (salario > 545)
);
```

INSERÇÃO DE DADOS

```
INSERT INTO nomeTabela VALUES (valorCampo1, valorCampo2, ... , valorCampoN);

INSERT INTO nomeTabela (nomeCampo1, nomeCampo2) VALUES (valorCampo1, valorCampo2);

INSERT INTO nomeTabela SELECT * FROM nomeDaTabela2;

INSERT INTO nomeTabela (nomeCampo1, nomeCampo2) SELECT (nomeCampo3, nomeCampo4) FROM nomeDaTabela2;
```

ALTERAÇÃO DE DADOS

```
UPDATE nomeTabela SET
nomeCampo1 = valor1,
nomeCampo2 = valor2,
nomeCampoN = valorN
WHERE condição
```

EXCLUSÃO DE DADOS

```
DELETE FROM nomeTabela
WHERE condição
```

CONSULTA DE DADOS

SINTAXE GERAL

SELECT [DISTINCT | ALL] coluna1, coluna2, ... , colunan AS apelidoColuna
FROM nomeDaTabelan AS apelidoTabela
WHERE condição
GROUP BY coluna1, coluna2, ... , colunan
HAVING condição
ORDER BY coluna1, coluna2, ... colunan [ASC | DESC]
DISTINCT : elimina linhas repetidas no resultado da consulta
GROUP BY coluna1, coluna2, ..., colunaN = dados são sumarizados pelas colunas que são especificadas. As colunas utilizadas na clausula SELECT deverão obrigatoriamente constar na clausula GROUP BY

Condições:
IS NULL
IS NOT NULL
IS LIKE '%string' => cadeia ilimitada
IS LIKE '_string' => 0 ou 1 caractere
BETWEEN 0 AND 0
IN {'A', 'B, ... 'Z'}
NOT IN {'A', 'B, ... 'Z'}

FUNÇÕES AGREGADORAS

COUNT (*) = numero de linhas da tabela
COUNT (coluna) = numero de ocorrências não-nulas de coluna na tabela pesquisada
SUM (coluna) = soma dos valores não-nulos de coluna
AVG (coluna) = média dos valores não-nulos de coluna
MAX (coluna) = maior valor de coluna
MIN (coluna) = menor valor de coluna

Exemplos:

Qual o preço médio dos livros por assunto ?
```
SELECT assunto, AVG(preço)
FROM livro
GROUP BY assunto
```

Quantos livros existem para cada assunto ?
```
SELECT assunto, COUNT(*)
FROM livro
GROUP BY assunto
```

Qual o preço do livro mais caro de cada assunto ?
```
SELECT assunto, MAX(preço)
FROM livro
GROUP BY assunto
```

HAVING = realizar restrições com base nos resultados das funções agregadoras. Servirá de filtro para as linhas constantes do resultado do comando SQL. Será seguida de uma expressão lógica similar a clausula WHERE

Quais os assuntos cujo preço médio dos livros ultrapassa R$ 50,00 ?
```
SELECT assunto, AVG(preco)
FROM livro
GROUP BY assunto
HAVING AVG (preço) > 50
```

Quais os assuntos que possuem pelo menos 2 livros ?
```
SELECT assunto, COUNT(*)
FROM livro
GROUP BY assunto
HAVING COUNT(*) > 1
```

FUNÇÕES COM STRINGS

UPPER (coluna | string) = converte string para maiúsculo
LOWER (coluna | string) = converte string para minúsculo
TRIM (coluna | string) = remove espaços a direita ou a esquerda da string
RTRIM (coluna | string) = remove espaços a direita da string
LTRIM (coluna | string) = remove espaços a esquerda da string
LENGTH (coluna | string)= tamanho de string
SUBSTRING (coluna | string, inicio, quantidade) = extrai “quantidade” caracteres de string a partir de “inicio”

REALIZANDO OPERAÇÕES ARITMÉTICAS

Listar os novos preços dos livros se os valores fossem reajustados em 10%
SELECT titulo, preço * 1.1 AS novoPreco FROM livro

FUNÇÕES COM DATA

DAY (coluna) = dia
MONTH (coluna) = mês
YEAR (coluna) = ano

JUNÇÃO INTERNA

O resultado é composto pelas linhas das tabelas de origem que atenderem a clausula de junção, ou seja, os valores das colunas de junção possuem o mesmo valor em ambas as tabelas

```
SELECT coluna1, coluna2, colunaN
FROM tabela1 INNER JOIN tabela2
ON tabela1.coluna = tabela2.coluna
```

JUNÇÃO EXTERNA A ESQUERDA

O resultado é composto pelas linhas da tabela posicionada a esquerda do termo LEFT OUTER JOIN no comando e que atendem aos critérios definidos na clausula WHERE, independentemente se existem valores correspondentes na coluna de junção da tabela posicionada à direita do comando. Caso não existam valores correspondentes na tabela a direita, as colunas desta tabela terão valor NULL.

```
SELECT coluna1, coluna2, colunaN
FROM tabela1 LEFT OUTER JOIN tabela2
ON tabela1.coluna = tabela2.coluna
```

Listar os assuntos e respectivos títulos

```
SELECT descricao, titulo
FROM assunto LEFT OUTER JOIN livro
ON sigla = assunto
```

JUNÇÃO EXTERNA A DIREITA

O resultado é composto pelas linhas da tabela posicionada a direita do termo RIGHT OUTER JOIN no comando e que atendem aos critérios definidos na clausula WHERE, independentemente se existem valores correspondentes na coluna de junção da tabela posicionada à esquerda do comando. Caso não existam valores correspondentes na tabela a esquerda, as colunas desta tabela terão valor NULL.

```
SELECT coluna1, coluna2, colunaN
FROM tabela1 RIGHT OUTER JOIN tabela2
ON tabela1.coluna = tabela2.coluna
```

Listar os assuntos e respectivos títulos

```
SELECT descricao, titulo
FROM livro RIGHT OUTER JOIN assunto
ON sigla = assunto
```

JUNÇÃO EXTERNA COMPLETA

O resultado é composto pelas linhas das tabelas que atendem aos critérios definidos na clausula WHERE, independentemente se existem valores correspondentes na coluna de junção. Caso não existam valores correspondentes nas tabelas, as colunas desta tabela terão valor NULL.

SELECT coluna1, coluna2, colunaN
FROM tabela1 FULL OUTER JOIN tabela2
ON tabela1.coluna = tabela2.coluna

JUNÇÃO CRUZADA = PRODUTO CARTESIANO

Todas as combinações possíveis entre linhas de duas tabelas
SELECT coluna1, coluna2, colunaN
FROM tabela1 CROSS JOIN tabela2

JUNÇÃO NATURAL (qualquer tipo de junção)

Não há necessidade da clausula ON, as colunas de junção devem ter o mesmo nome e todas participam da condição de junção.
SELECT coluna1, coluna2, colunaN
FROM tabela1 NATURAL tipo JOIN tabela2

JUNÇÃO BASEADA EM NOMES DE COLUNAS

Similar a junção natural, no entanto, apenas as colunas especificadas em USING participam da condição de junção
SELECT coluna1, coluna2, colunaN
FROM tabela1 tipo JOIN tabela2
USING (coluna1, coluna2, ..., colunaN)

FUNÇÃO COALESCE

Substitui os valores nulos por outro valor designado no comando SQL.
SELECT coluna1, COALESCE (coluna2, 'novo valor' | coluna) AS coluna2
FROM tabela1 tipo JOIN tabela2 ON tabela1.coluna = tabela2.coluna

SUBCONSULTAS

Não-correlacionadas: consulta interna é totalmente independente da consulta externa
Clausula IN = testa se os valores de uma coluna constam em uma tabela temporária

SELECT coluna1, coluna2, colunaN
FROM tabela1
WHERE coluna1 [NOT] IN (
SELECT coluna3, coluna4, colunaM
FROM tabela2)

Relacionadas: consulta interna possui dependência direta da consulta externa
Clausula EXISTS = testa se uma condição é verdadeira ou falsa

SELECT coluna1, coluna2, colunaN
FROM tabelaExterna
WHERE coluna1 [NOT] EXISTS (
SELECT coluna3, coluna4, colunaM
FROM tabelaInterna
WHERE tabelaExterna.coluna1 = tabelaInterna.coluna3)

Substituição de colunas por uma consulta
SELECT coluna1, coluna2, colunaN, (SELECT ...) AS apelido
FROM tabela1, tabela2, tabelaN

Tabela aninhada (substituição de uma tabela por uma consulta)
SELECT coluna1, coluna2, colunaN
FROM tabela1, tabela2, tabelaN, (SELECT ...) AS apelido

OPERAÇÕES DE CONJUNTO

(1) os comandos devem retornar a mesma quantidade de colunas;
(2) as colunas correspondentes devem possuir o mesmo tipo de dados
(3) ALL = mostra linhas repetidas

União
SELECT ....
UNION [ALL]
SELECT ...

Interseção
SELECT ....
INTERSECT [ALL]
SELECT ...

Diferença
SELECT ....
EXCEPT [ALL]
SELECT ...

VISÕES E VISÕES TEMPORÁRIAS

Visão é uma tabela virtual cujo conteúdo provem de tabelas reais, cujos dados que a compõem são definidos através de comandos SELECT realizados sobre as tabelas de um banco de dados.

CREATE VIEW nomeDaVisao AS
COMANDO SELECT

Ex.Listar titulo, preço e nome da editora e descrição do assunto
CREATE VIEW livro_editora_assunto AS
SELECT titulo, preço, nome AS Editora, descricao AS assunto
FROM livro INNER JOIN editora ed ON editora = ed.codigo
INNER JOIN ON assunto.sigla = livro.assunto

SELECT titulo, editora, assunto
FROM livro_editora_assunto

DROP VIEW nomeDaVisao

Visão temporária é uma visão para ser utilizada apenas em um comando, sem que sua definição fique armazenada no banco de dados.

WITH nomeDaVisaoTemporaria [colunas] AS
COMANDO SQL

PREDICADO CASE

Selecionar resultados diferentes em função de uma ou mais condições.

```
CASE coluna
WHEN valor1 THEN resultado1
WHEN valor2 THEN resultado2
WHEN valor3 THEN resultado3
[ELSE resultado4]
END AS [apelidoDaColuna]
```

```
CASE
WHEN condição1 THEN resultado1
WHEN condição1 THEN resultado2
WHEN condição1 THEN resultado3
[ELSE resultado4]
END AS [apelidoDaColuna]
```

Ex.

```
SELECT titulo, case codigoAssunto
WHEN 1 THEN 'Muito interessante'
WHEN 2 THEN 'Interessante'
WHEN 3 THEN 'Pouco interessante'
ELSE 'Sem interesse'
END AS Importancia
FROM livro
```

```
SELECT titulo, case
WHEN preço > 100 THEN 'Muito caro'
WHEN preço > 50 THEN 'Caro'
ELSE 'Barato'
END AS Preco
FROM livro
```

Comando MERGE

Compara informações de tabelas, inserindo as linhas que não existem e atualizando as outras.

```
MERGE INTO tabelaDestino
USING tabelaOrigem
ON expressãoJunção
WHEN MATCHED THEN operação1
WHEN NOT MATCHED THEN operação2
```

```
MERGE INTO cliente c
USING autor a
ON (a.cpf = c.cpf)
WHEN MATCHED THEN
UPDATE SET c.nome = a.nome, c.endereco = a.endereco, c.dataNascimento = a.dataNascimento
WHEN NOT MATCHED THEN
INSERT (código, nome, CPF, endereço, dataNascimento)
VALUES (a.matricula, a.nome, a.cpf, a.endereco, a.dataNascimento)
```

STORED PROCEDURES (SP) e FUNCTIONS (SF)

Pequenos programas compilados, armazenados e executados diretamente no servidor de banco de dados. São invocados através de comandos SQL.

TRIGGER

Mecanismo que permite a execução automática de comandos e/ou programas a partir de eventos (inserção, alteração ou exclusão) que ocorram no banco de dados.

LINGUAGEM PARA ESCREVER SP, SF E TRIGGER

BLOCOS DE COMANDOS

São pequenos programas compostos por um ou mais comandos SQL. Permitem a especificação de variáveis e cursores próprios aos blocos.

```
BEGIN
        DECLARAÇÃO DE VARIÁVEIS
        DECLARAÇÃO DE COMANDOS
        LISTA DE COMANDOS SQL
END
```

DECLARAÇÃO DE VARIÁVEIS

```
DECLARE nomeVariavel1, nomeVariavel2, ... nomeVariavelN TIPO
```

Em alguns SGBDs, o nome da variável deve iniciar com o caractere “@”.

CURSOR

Mecanismo que permite que as linhas de uma tabela sejam manipuladas uma a uma. Atuam como ponteiros que apontam para as linhas que formam o resultado de uma dada consulta. Pode-se recuperar e manipular os valores de cada linha apontada por um cursor.

DECLARE CURSOR nomeCursor
FOR comandoSELECT
[FOR UPDATE] = indica que o resultado do comando de seleção será atualizável.

OPEN nomeCursor = abrir o cursor
CLOSE nomeCursor = fechar o cursor
FETCH nomeCursor INTO nomeVariavel1, nomeVariavel2, ... nomeVariavelN

O comando FETCH geralmente é usado em conjunto com um comando de repetição (REPEAT, WHILE ou FOR). Ao final, o cursor deve ser fechado.

ATRIBUIÇÃO DE VALORES

SET nomeVariavel = valor;

COMANDO FOR

Permite declarar um cursor, percorrer seu conteúdo e fechá-lo automaticamente.

FOR nomeCursor
comandoSelect
DO comandos SQL
END FOR

COMANDO SELECT INTO

Atribui um valor a uma variável.

SELECT nomeColuna
INTO nomeVariavel
FROM nomeTabela
WHERE condição

COMANDO IF

IF condição THEN
    comandos SQL
ELSEIF condição THEN
    comandos SQL
ELSE
    comandos SQL
END IF

COMANDO WHILE

WHILE condição DO
    comandos SQL
END WHILE

COMANDO REPEAT

REPEAT
    Comandos SQL
UNTIL condição
END REPEAT

COMANDO LOOP

LOOP
    Comandos SQL
END LOOP

Para sair do LOOP, usa-se o comando LEAVE.

STORED PROCEDURES

```
CREATE PROCEDURE nomeProcedure (
        modoParametro1 nomeParametro1 tipoParametro1,
        modoParametro2 nomeParametro2 tipoParametro2,
        modoParametroN nomeParametroN tipoParametroN)
        LANGUAGE linguagem
AS
BEGIN
        comandos SQL
END
```

Modos de Parâmetro:
IN: valor da variável externa é repassado à variável interna referente ao parâmetro (por valor).
OUT: valor da variável interna é repassado à variável externa correspondente.
INOUT valor da variável externa é repassado à variável interna e alterações ocorridas nas variável interna são efetivadas nas variáveis externas (por referência).

```
CREATE PROCEDURE stPrecoMedio (
        IN numeroLivros INTEGER;
        OUT precoMedio REAL)
LANGUAGE SQL
AS
BEGIN
        DECLARE valorTotal REAL;
        SET precoMedio = 0;
        SET valorTotal = 0;
        FOR meuCursor AS
                SELECT preço FROM livro
        DO
                SET valorTotal = valorTotal + preço;
        END FOR;
        SET precoMedio = valorTotal / numeroLivros;
END

BEGIN
        DECLARE numero INTEGER;
        DECLARE media REAL;
....
        stPrecoMedio(numero, media);
END
```

Apagando uma SP : DROP PROCEDURE nomeProcedimento
Alterando uma SP : ALTER PROCEDURE nomeProcedimento

STORED FUNCTIONS

Similar a STORED PROCEDURES, com o retorno de um valor.

```
CREATE FUNCTION nomeFuncao (
        modoParametro1 nomeParametro1 tipoParametro1,
        modoParametro2 nomeParametro2 tipoParametro2,
        modoParametroN nomeParametroN tipoParametroN)
        RETURNS tipoRetorno
AS
BEGIN
        comandos SQL
END

CREATE FUNCTION precoMedio (
        IN numeroLivros INTEGER)
        RETURNS REAL;
AS
```

```
BEGIN
        ...
        RETURN 1.5;
END
```

Apagando uma SF : DROP FUNCTION nomeFuncao
Alterando uma SP : ALTER FUNCTION nomeFuncao

TRIGGERS

```
CREATE TRIGGER nomeTrigger
        FOR nomeTabela | nomeVisao
        BEFORE | AFTER
        INSERT | UPDATE | DELETE
        POSITION numero
FOR EACH ROW | STATEMENT
WHEN condição
Comandos SQL
```

No MYSQL

```
CREATE TRIGGER trigger_name
[BEFORE | AFTER]  [INSERT | UPDATE | DELETE]
ON table_name
FOR EACH ROW
BEGIN
        trigger_body
END
```

SEGURANÇA DE BANCO DE DADOS

CRIAÇÃO DE USUÁRIOS

CREATE USER nomeUsuario IDENTIFIED BY senha

Ex.: CREATE USER php IDENTIFIED BY ‘senhaphp’

DIREITOS DE OBJETO

Sobre as tabelas de um banco de dados:
SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALL PRIVILEGES

ATRIBUINDO DIREITOS DE OBJETO

```
GRANT direito
ON tabela(coluna)
TO usuário | PUBLIC
[WITH GRANT OPTION]
```

PUBLIC: todos os usuários do banco de dados
WITH GRANT OPTION: transfere ao usuário que recebe o direito a possibilidade de conceder os mesmos direitos que recebeu a outros usuários.

Ex.:
GRANT SELECT ON livro TO php
GRANT SELECT, UPDATE ON autor(nome) TO php
GRANT SELECT ON livro TO PUBLIC

REVOGANDO DIREITOS DE OBJETO

```
REVOKE direito
ON tabela
FROM usuário
```

Ex.:
REVOKE SELECT
ON livro
FROM php

DIREITOS DE SISTEMA

Sobre um banco de dados.
CONNECT, ALTER, CREATE, DROP, EXECUTE

Manual completo de SQL disponível em
http://www.w3schools.com/sql/