SQL Avançado
Atualização e Inserção de Dados

Sumário

- Comando INSERT
  - Inclusão simples, inclusão de dados com SELECT
- Comando UPDATE
  - Alteração simples, alteração usando subquery
- Comando DELETE
  - Eliminação simples, eliminação utilizando subquery
- Comando TRUNCATE TABE
  - Remoção de todos os registros de uma tabela

Comando Insert

- **Propósito**
  - O comando INSERT **inclui um registro no final de uma** tabela de acordo com os valores especificados
- **Sintaxe**

  INSERT INTO <tabela> [<lista de campos>]

  VALUES <lista de valores>
- **Argumentos**
  - <tabela>: especifica a tabela na qual o registro será incluído
  - <lista de campos>: especifica os campos da tabela cujos valores serão inseridos

Comando Insert

- **Argumentos (cont.)**
  - VALUES <lista de valores>: especifica os valores (correspondentes aos respectivos campos) a serem inseridos na tabela

Comando Insert

- Exemplo: Inserir na Tabela Cliente os seguintes valores: codigo = 7, nome = Claudia Melo, tipo = PF, endereco = R. Macapá, 33, Bessa, cidade = 1, cep = 58000-00, fone = 3246-9522

```
Insert  Into Cliente (codigo, nome, tipo, endereco, cidade, cep, fone)
Values (7, 'Claudia Melo', 'PF', R. Macapá, 33, Bessa', '1',
        '58000-00', '3246-9522')
```

Incluindo dados com Select

- **Propósito**
  - Inserir dados em uma tabela tendo como valores o resultado de um SELECT
- **Sintaxe**

  INSERT INTO **<tabela para insercao>** SELECT

  <lista de campos> FROM <tabela para consulta>

  [WHERE, GROUP BY, ORDER BY, etc.]
- **Observações**
  - As colunas resultantes do SELECT têm que casar com os tipos dos dados das colunas da cláusula INTO
  - Não se usa o **VALUES no INSERT** (neste caso)

Incluindo dados com Select

```
CREATE TABLE tabela1(
   coluna1 int IDENTITY,
   coluna2 varchar(40) NULL,
   coluna3 char(1) NULL,
   coluna4 varchar(30) NULL
)
        INSERT INTO tabela1
        SELECT nome, sexo, endereco
        FROM funcionario
        WHERE estcivil <> 'C'
```

Incluindo dados com Select

```
DROP TABLE tabela1

CREATE TABLE tabela1 (
pedido smallint ,
cliente varchar(40) NULL,
vendedor varchar(40) NULL,
CONSTRAINT PK_tabela PRIMARY KEY (pedido),
)

INSERT INTO tabela1
SELECT p.código, c.nome, f.nome
FROM pedido p, cliente c, funcionario f
WHERE c.código = p.cliente and f.codigo = p.vendedor
```

Comando Update

- **Propósito**
  - O comando UPDATE altera os valores armazenados no(s) registro(s) de uma tabela
- **Sintaxe**

```
UPDATE <tabela>
SET <coluna1=valor1 [,coluna2=valor2,...,
      coluna_n=valor_n]>
[WHERE <condição >]
```

- **Argumentos**
  - <tabela>: especifica a tabela na qual o(s) registro(s) será(ão) alterados
  - SET <coluna1=valor1[,coluna2=valor2,...]>:
    - Especifica as colunas a serem alteradas e seus novos valores

Alteração Simples

- **Argumentos** (cont.)
  - WHERE <condição>: especifica as condições que precisam ser satisfeitas pelos registros que terão seus valores atualizados
- Exemplo: Aumente o salário dos funcionários casados em 25%

```
UPDATE funcionario
SET salario = salario*1.25
WHERE estCivil = 'C'
```

Alteração Simples

- Exemplo: Dar um aumento de 15% no salário dos funcionários homens, casados ou divorciados, e que tenham nascido entre 01/01/50 e 31/12/70.

```
UPDATE funcionario
SET salario=salario*1.15
WHERE sexo='M' AND estCivil IN ('C','D')
AND dataNasc BETWEEN '1950-01-01'
AND '1970-12-31'
```

Alteração utilizando subquery

- Exemplo: Fornecer um aumento de 18% no salário dos funcionários que tenham o salário menor que a média de salários da empresa.

```
UPDATE funcionario
SET salario=salario*1.18
WHERE salario < (SELECT AVG(salario) FROM
                 funcionario)
```

Alteração utilizando subquery

- Exemplo: Baixar o preço em 12% dos produtos com preço de venda igual ao do produto mais caro vendido na empresa.

```
UPDATE produto SET venda=venda*0.88
WHERE venda = (SELECT MAX(venda) FROM
                      produto)
```

Alteração utilizando subquery

- Exemplo: Alterar de todos os funcionários a função 'Vendedor' para 'Auxiliar de Caixa'.

```
UPDATE funcionario
SET funcao = (SELECT codigo FROM
                    funcao WHERE nome =
                          'Auxiliar de Caixa')
WHERE funcao = (SELECT codigo FROM
                     funcao WHERE nome =
                    'Vendedor')
```

Comando Delete

- **Propósito**
- O comando DELETE **elimina registros de uma tabela** de acordo com a condição especificada.
- **Sintaxe**

  DELETE FROM <tabela>

  [WHERE <condição >]
- **Argumentos**
  - <tabela>: especifica a tabela na qual o(s) registro(s)será(ão) excluídos
  - WHERE <condição>: especifica as condições que precisam ser satisfeitas para a remoção dos registros

Remoção Simples

- Exemplo: Remova todos os registros da tabela 'usuario'.

```
DELETE FROM usuario
```

- Exemplo: Excluir todos os clientes do tipo pessoa física que não possuam telefone

```
DELETE FROM cliente
WHERE tipo='PF' and fone is null
```

Remoção usando subquery

- Exemplo: Excluir todos os funcionários que tenham salário maior que a média de salário da empresa.

```
DELETE FROM funcionario
WHERE salario > (SELECT
      AVG(salario) FROM funcionario)
```

Remoção usando subquery

- Exemplo: Excluir todos os pedidos dos clientes dos estados da Paraíba e Pernambuco.

```
DELETE FROM pedido
WHERE cliente =
(SELECT codigo FROM cliente WHERE
cidade IN (SELECT codigo FROM cidade
WHERE uf IN ('PB','PE')))
```

Comando TRUNCATE TABLE

- **Propósito**
  - Comando usado para excluir todas as linhas de uma tabela
- **Sintaxe**

  TRUNCATE TABLE <tabela>
- **Características**
  - Semelhante ao DELETE FROM sem o WHERE
  - Quase sempre é mais rápido que o DELETE, especialmente em tabelas grandes
  - TRUNCATE TABLE não salva informações no Log de Transações

Regras de Integridade

Introdução

- As regras de integridade fornecem a garantia que mudanças feitas no BD não resultem em perda da consistência dos dados
  - Restrições impostas aos dados

Introdução

- **Em BD I:**

```
CREATE TABLE dvd (
        cod_D int not null identity,
        titulo varchar(40) not null,
        genero varchar(15),
        duracao time,
        situacao varchar(12) default ('Disponível'),
        PRIMARY KEY (cod_D),
        CHECK (situacao in ('Alugada','Disponível'))
)
```

Introdução

- Para assegurar que os dados do BD estão válidos precisa-se formalizar
  - verificações (check)
  - e regras de negócio que os dados devem aderir.
- Esta forma de assegurar consistência é conhecida como restrições de integridade (integrity constraints).

Introdução

- Restrição de Integridade
  - Finalidade é fornecer um meio de assegurar que as mudanças feitas no BD, por usuários autorizados, não levam à perda de consistência dos dados.

Integridade

- Questão: Como garantir Integridade do BD?
  - Solução 1: Inserir as restrições de integridade no código da aplicação
- Problemas:
  - Sobrecarrega o programador, pois ele precisa codificar todas as validações
  - Susceptível a erros → pode-se esquecer de fazer uma validação => BD inconsistente
  - Difícil manutenção => Se RI mudar, então precisa mudar todas as aplicações que validam àquela RI

Integridade

- Solução 2: Inserir as Regras de Integridade no próprio BD, através do subsistema do SGBD.

Regras de Integridade

- Importante:
  - As regras são compiladas e armazenadas no dicionário de dados

Regras de Integridade

- Vantagens:
  - Validação é tratada pelo SGBD, ao invés de ser feita nas aplicações individuais;
  - Melhor entendimento e manutenção devido à atualização das regras no catálogo
  - Deve existir um meio de atualizar regras com o sistema funcionando
  - Pode-se utilizar linguagem de consulta para as RI's

Modelo Relacional

- Integridade nos BDs Relacionais

1. Integridade de Domínio
   Os valores de uma coluna (atributo) devem estar dentro do conjunto de valores possíveis do domínio
2. Integridade de Vazio
   Define se o valor de uma coluna pode ser nulo ou não.
   obs: nulo não é zero nem branco.

Modelagem Relacional

3. Integridade de Entidade

Os valores de uma chave primária não podem ser nulos

4. Integridade Referencial (chave estrangeira)

Integridade Referencial define que os valores de uma coluna pertencente a uma chave estrangeira (FK) devem existir na chave primária da tabela referenciada.

Restrições de Domínio

- Restrições de domínio são as mais elementares formas de restrição de integridade
- São facilmente verificadas pelo sistema, sempre que um novo item de dados é incorporado ao BD

Restrições de Domínio

- A cláusula **check** permite modos poderosos de restrições de domínio
- Ela permite ao projetista do esquema determinar um predicado que deva ser satisfeito por qualquer valor designado a uma variável cujo tipo seja o domínio

Integridade Referencial

- Garantir que um valor que aparece em uma relação para um dado conjunto de atributos também apareça para um certo conjunto de atributos de outra relação
- Feitos por meio de chaves primárias e estrangeiras

Integridade Referencial

```
create table cliente
(cpf_cliente char(20) not null
 nome_cliente char(20),
 rua_cliente char(30),
 cidade_cliente char(30),
 primary key (cpf_cliente))
```

Integridade Referencial

```
create table conta
(numero_conta char(10) not null
 nome_agencia char(20),
 saldo integer,
 primary key (numero_conta),
 foreign key (nome_agencia) references
                agencia,
 check (saldo >= 0))
```

Integridade Referencial

```
create table depositante
(nome_cliente char(20) not null
 numero_conta char(10),
 primary key
 (nome_cliente, numero_conta),
 foreign key
 (nome_cliente) references cliente,
 foreign key
 (numero_conta) references conta)
```

Visões em SQL

Sumário

- Visões
  - Sintaxe
  - Exemplos
  - Vantagens e Desvantagens

Visão

- Definição
  - Um meio de fornecer ao usuário um modelo de dados personalizado
  - Uma **visão** pode esconder dados que um determinado usuário não precisa ver
- Vantagens
  - Restringir "quais colunas" de uma tabela podem ser acessadas (leitura/modificação)
  - Podem conter valores calculados ou valores de resumo

Visão

- Benefícios diretos
  - Simplificar o uso do sistema
    - É permitido ao usuário restringir a atenção aos dados de interesse
  - Segurança
    - O usuário só acessa os dados que forem pré-determinados

Visão

- Como definimos uma visão?
  - Definição de um **SELECT** que faz uma consulta sobre as tabelas
  - A visão aparece no esquema de dados "como se fosse uma tabela"

Visão

- **Sintaxe de criação**

  CREATE VIEW <nome da visao> **AS**

  (<instrução SELECT> **)**

Visão

- Considere as tabelas abaixo:

Empréstimo

numEmp	*Agencia*	*Valor*
E170	Tambaú	1.500,00
E200	Cristo	2.800,00

Devedor

Nome_Cliente	*Num_empres*
João	E170
Carol	E155

Visão

- Exemplo 1:
  - Um usuário precisa da agência e nome de todos os clientes que têm um empréstimo em cada agência. Este usuário não está autorizado a ver informações específicas aos empréstimos dos clientes.

Visão

```
CREATE VIEW cliente_emprestimo AS
(SELECT e.agencia, d.nome_cliente
 FROM devedor d, emprestimo e
 WHERE e.numEmp = d.num_empres)
```

Na consulta a visão:

```
SELECT *FROM cliente_emprestimo
```

Visão

- Exemplo 2:
  - Crie uma visão que forneça o nome do usuário e a quantidade de mensagens que o mesmo recebeu.

```
CREATE VIEW estatistica_mensagem AS
(SELECT r.destinatario Usuario,
        COUNT(*) QuantRecebidas
 FROM mensagem m, recebedor r
 WHERE m.idmens = r.idmens
 GROUP BY r.destinatario)
```