Bancos de Dados

CIC0097

Prof. Pedro Garcia Freitas

https://pedrogarcia.gitlab.io/

pedro.garcia@unb.br

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciências da Computação

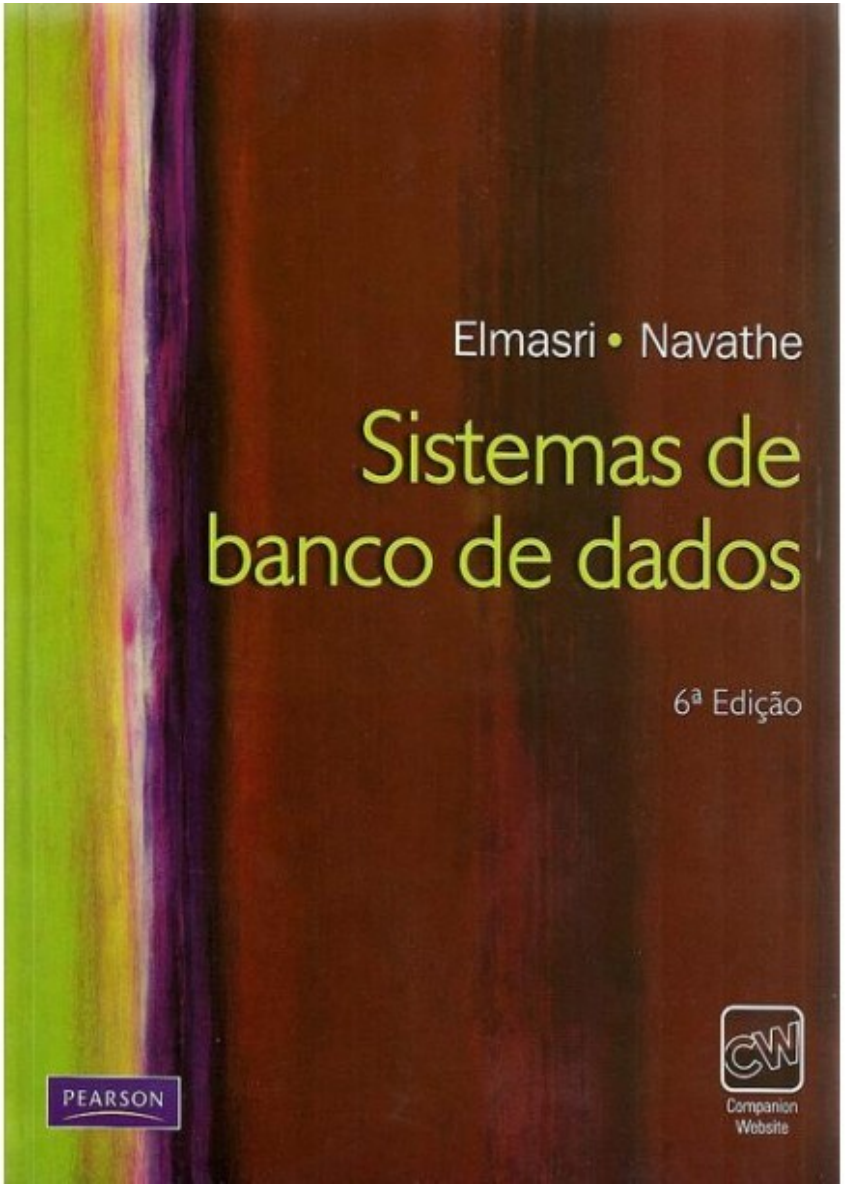
Este conjunto de slides não deve ser utilizado ou republicado sem a expressa permissão do autor.

This set of slides should not be used or republished without the author's express permission.

Módulo 16

Linguagem de Consulta Estruturada Parte 5: Restrições (*constraints*), verificações (*asserts*), regras ativas (*triggers*), paradigma evento-condição-ação

CIC0097/2023.1
T1/T2



Esta aula se baseia no Capítulo 5 (SQL básica) do Elmasri e Navathe (6ª Edição).

1. Objetivos

Esta aula descreve recursos mais avançados da linguagem SQL padrão para bancos de dados relacionais, incluindo os comandos **`CREATE ASSERTION`**, **`CREATE TRIGGER`**, etc.

2. Introdução

Existem várias alternativas para aplicar restrições que não podem ser consideradas apenas pelo modelo de dados:

1. Utilizar funcionalidades da linguagem de programação selecionada para desenvolver a aplicação que irá interagir com o banco de dados, ou utilizar outros frameworks de construção de sistemas.
2. Empregar a linguagem de programação de propósito geral que está disponível no Sistema Gerenciador de Banco de Dados (SGBD).
3. Explorar recursos avançados do SQL.
4. Combinar as capacidades das opções mencionadas acima.

2. Introdução

Recursos avançados de SQL incluem

- asserções
- regras ativas (gatilhos)
- visões

2. Introdução

Recursos avançados de SQL incluem

- asserções
- regras ativas (gatilhos)
- visões

Cada Sistema Gerenciador de Banco de Dados (SGBD) oferece distintas abordagens para aplicar restrições utilizando a linguagem SQL.

3. Asserções (*assertions*)

- Em SQL, os usuários podem especificar restrições gerais (que vão além das restrições de chaves) por meio de *asserções declarativas*.
- Comando **`CREATE ASSERTION`** (DDL).
- Cada asserção recebe um nome de restrição e é especificada por uma condição semelhante à cláusula **`WHERE`** de uma consulta SQL.

3. Asserções (*assertions*)

```
CREATE ASSERTION <nome da restrição>
CHECK (<condição>);
```

3. Asserções (*assertions*)

Exemplo **1**: Especificar a restrição de que o salário de um servidor não pode ser maior que o salário do gerente do departamento qual o qual o servidor trabalha.

- **SERVIDOR(cpf,nome,sobrenome,endereco,dt_nasc,salario, sexo, fk_cpf_supervisor,fk_dnumero)**
- **DEPARTAMENTO(numero, nome, fk_cpf_gerente, dtinicio)**

3. Asserções (*assertions*)

```
CREATE ASSERTION restricao_salarial
CHECK (NOT EXISTS (
          SELECT *
           FROM Servidor AS S,
                Servidor AS G,
                Departamento AS D
          WHERE S.fk_dnumero=D.numero
            AND D.fk_cpf_gerente=G.cpf
            AND S.salario>G.salario
));
```

3. Asserções (*assertions*)

```
CREATE ASSERTION restricao_salarial
CHECK (NOT EXISTS (
          SELECT *
           FROM Servidor AS
                Servidor X
));
```

- A restrição de nome “restrição_salarial” pode ser usado mais tarde para se referir à restrição ou para modificá-la ou excluí-la.

3. Asserções (*assertions*)

```
CREATE ASSERTION restricao_salarial
CHECK (NOT EXISTS (
          SELECT *
```

- A restrição de nome “restrição_salarial” pode ser usado mais tarde para se referir à restrição ou para modificá-la ou excluí-la.
- Sempre que alguma tupla no banco de dados fizer que a condição de um comando **ASSERTION** seja avaliada como **FALSE**, a restrição é violada.

3. Asserções (*assertions*)

- A restrição de nome “restrição_salarial” pode ser usado mais tarde para se referir à restrição ou para modificá-la ou excluí-la.
- Sempre que alguma tupla no banco de dados fizer que a condição de um comando **`ASSERTION`** seja avaliada como **`FALSE`**, a restrição é violada.
- A restrição é satisfeita por um estado do banco de dados se **nenhuma combinação de tuplas** nesse estado do banco de dados violar a restrição.

3. Asserções (*assertions*)

- Asserções são úteis para garantir a integridade dos dados e assegurar que os dados no banco de dados atendam a certas condições. Elas podem ser usadas para impor regras de negócio ou garantir a consistência dos dados. No entanto, também podem demandar tempo para serem criadas e mantidas, portanto, nem sempre são utilizadas na prática.
- **DROP ASSERTION <nome da asserção>;**

4. Regras ativas (*triggers*)

- Um trigger no SQL é um processo que funciona como gatilho para executar algumas ações automaticamente sempre que uma mudança ocorre no banco de dados.
- Comando **`CREATE TRIGGER`**.
- Ou seja, é principalmente uma função voltada para a automação: sempre que uma ação ocorre, o trigger executa outra ação como resposta imediata.

4. Regras ativas (*triggers*)

- Como funciona um *trigger*: Funciona como um mecanismo automático que depende de algum gatilho para disparar. Depende, portanto, de manipulações na tabela.
- Nesse sentido, existem dois tipos principais: o **`AFTER`** e o **`INSTEAD OF`**. (Modelo Evento-Condição-Ação)

4. Regras ativas (*triggers*)

- Modelo Evento-Condição-Ação:
  - **Evento:** geralmente operações de alteração aplicadas ao banco de dados (**`insert`**, **`update`**, **`delete`**)
  - **`BEFORE`** – significa que o gatilho deverá ser executado antes do evento ter efeito no banco de dados.
  - **`AFTER`** – significa que o gatilho deverá ser executado depois do evento ter efeito no banco de dados.
  - **`INSTEAD OF`** - executa algo no lugar do evento inicial, quando este é detectado.

4. Regras ativas (*triggers*)

```
CREATE TRIGGER <nome do gatilho>
ON <nome da tabela>
INSTEAD OF/AFTER/BEFORE
   {[INSERT],[UPDATE],[DELETE] }
AS
   {sql_statements}
```

4. Regras ativas (*triggers*)

Exemplo **2**: Sempre que o salário de um servidor for maior que um supervisor direto, o supervisor precisa ser avisado.

- **SERVIDOR(cpf,nome,sobrenome,endereco,dt_nasc ,salario, sexo,fk_cpf_supervisor,fk_dnumero)**

4. Regras ativas (*triggers*)

Exemplo 2: Sempre que o salário de um servidor for maior que um supervisor direto, o supervisor precisa ser avisado.

```
CREATE TRIGGER VIOLACAO_SALARIO
BEFORE INSERT OR UPDATE OF salario, fk_cpf_supervisor
ON Servidor
FOR EACH ROW
WHEN ( NEW.salario > (
    SELECT salario FROM Servidor
    WHERE cpf = NEW.fk_cpf_supervisor )
)
INFORM_SUPERVISOR ( NEW.fk_cpf_supervisor, NEW.cpf);
```

5. Visões (*views*)

- Uma *view* em terminologia SQL é uma única tabela que é derivada de outras tabelas.
- Essas outras tabelas podem ser **tabelas da base** ou outras *views* previamente definidas.

5. Visões (*views*)

- Uma *view* não necessariamente existe em forma física: ela é considerada uma **tabela virtual**, ao contrário das tabelas da base, cujas tuplas sempre estão armazenadas fisicamente no banco de dados.

5. Visões (*views*)

- Isso limita as possíveis operações de atualização (DML) que podem ser aplicadas às *views*, mas não oferece quaisquer limitações sobre a consulta de uma *view* (DQL).
  - Tabela que pode ser consultada de forma livre, mas que está l**imitada no que diz respeito à ações de inserção** de dados.
  - Usadas para **reuso**, **otimização** e **segurança**.

5. Visõ

- Isso li
  atuali
  às *vie*
  limita
  (DQL).
  - Tabela que pode nsultada de forma livre, mas que está l**imitada no q diz respeito à ações de inserção** de dados.
  - Usadas para **otimização** e para **segurança**.

- Pensamos em uma *view* como um modo de especificar uma tabela que precisamos referenciar com frequência.
- Por exemplo, podemos emitir frequentemente consultas que recuperam o nome do servidor e os nomes dos projetos em que o funcionário trabalha. Em vez de ter que especificar a junção das três tabelas toda vez, podemos definir uma *view* que é especificada como o resultado dessas junções (*read only* cache).

5. Visões (*views*)

```
CREATE VIEW <nome da view> AS
   SELECT <colunas selecionadas>
   FROM <nome da tabela>
   WHERE <condição do SELECT>;
```

5. Visões (*views*)

Exemplo **3**: Considerando a tabela **`Produto`** abaixo, crie uma view que apresente os atributos originais de maneira legível e em língua inglesa.

`Produto`

Id	Nome	Fabricante	Quantidade	VlUnitario	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

5. Visões (*views*)

Exemplo **3**: Considerando a tabela **`Produto`** abaixo, crie uma view que apresente os atributos originais de maneira legível e em língua inglesa.

`Produto`

Id	Nome	Fabricante	Quantidade	VlUnitario	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

5. Visões (*views*)

Exemplo **3**: Considerando a tabela **`Produto`** abaixo, crie uma view que apresente os atributos originais de maneira legível e em língua inglesa.

`Produto`

Id	Nome	Fabricante	Quantidade	VlUnitario	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

5. Visões (*views*)

Exemplo 3: Considerando a tabela **`Produto`** abaixo, crie uma view que apresente os atributos originais de maneira legível e em língua inglesa.

```
CREATE VIEW ProdutView AS
    SELECT IdProduto AS code,
        Nome AS product,
        Fabricante AS brand,
        Quantidade AS amount,
        VlUnitario AS unit_value,
        Tipo AS type
    FROM Produtos
```

5. Visões

- Para consultarmos os dados na view usamos o comando SELECT, da mesma forma que se estivéssemos fazendo uma consulta em uma tabela comum.
- **SELECT * FROM ProdutView;**

Exemplo 3: C
crie uma viev
maneira legív

```
CREATE VIEW ProdutView AS
    SELECT IdProduto AS code,
        Nome AS product,
        Fabricante AS brand,
        Quantidade AS amount,
        VlUnitario AS unit_value,
        Tipo AS type
    FROM Produtos
```

5. Visões (*views*)

Exemplo 4: Altere a view `ProdutView` criada anteriormente para que ela exiba apenas os produtos que sejam maiores que R$ 500,00.

5. Visões (*views*)

Exemplo 4: Altere a view `ProdutView` criada anteriormente para que ela exiba apenas os produtos que sejam maiores que R$ 500,00.

```
ALTER VIEW ProdutView AS
    SELECT IdProduto AS Codigo,
        Nome AS Produto,
        Fabricante,
        Quantidade,
        VlUnitario AS [Valor Unitario],
        Tipo
    FROM Produtos WHERE VlUnitario > 499.00
```

5. Visões (*views*)

Exemplo 4: Altere a view `ProdutView` criada anteriormente para que ela exiba apenas os produtos que sejam maiores que R$ 500,00.

`ProdutView`

Código	Produto	Fabricante	Quantidade	Valor Unitário	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

5. Visões (*views*)

Exemplo **5**: Exclua a *view* **`ProdutView`.**

```
DROP VIEW ProdutView;
```

Este material está disponível em: https://github.com/pedrogarciafreitas/slides-bd-2023.1

Dúvidas?