[MAC0439] Laboratório de Bancos de Dados
Aula 16 (Parte 2)
Linguagem SQL: Visões

Kelly Rosa Braghetto

DCC–IME–USP

19 de outubro de 2016

(Slides baseados no material da Profa. Jennifer Widom, da *Stanford University*)

Tabelas × visões

Tabelas

- **Tabela = relação armazenada**
- São definidas com o comando CREATE TABLE e existem **fisicamente** no BD
  - Espera-se que existam indefinidamente
  - Não mudam a menos que seja aplicado sobre elas algum comando SQL de modificação de dados

Visões

- **Visão = relação virtual**
- Visões são relações que não são armazenadas fisicamente
- São definidas como consultas
- Elas podem ser consultadas como se existissem fisicamente
- Em alguns casos, é até mesmo possível modificar as visões

Por que usar visões?

- Por segurança (para esconder dados de alguns usuários)
- Para tornar algumas consultas mais fáceis ou naturais de serem expressas
- Para modularidade

⇒ Aplicações de bancos de dados reais usam **muitas** (mas muitas mesmo!) visões.

Declaração de visões

- Formato geral:

CREATE VIEW <nome-visão> **AS** <definição-visão>;
ou
CREATE VIEW <nome-visão> (Atrib_1, Atrib_2, ..., Atrib_n)
AS <definição-visão>;

- A definição da visão é uma consulta Q
- Sempre que aplicarmos uma consulta sobre a visão, a SQL se comportará como se Q estivesse sendo executada naquele momento e a consulta será aplicada sobre o resultado de Q

Exemplo (1)

BD de Inscritos para Intercâmbio

```
Estudante(ID, nome, endereco, media)
Universidade(unome, cidade, rank)
Inscricao(ID, unome, data, curso, decisao)
```

– Valores p/ o atributo decisao: 'N' (não decidido), 'A' (aprovado) e 'R' (reprovado)

Visão: Inscritos recentemente para o MIT, mas sem decisão

```
CREATE VIEW MInscRecente AS
SELECT ID, data, curso
FROM Inscricao
WHERE unome = 'MIT' AND data > '2016-09-01'
      AND decisao = 'N'
```

Exemplo (1)

A visão MInscRecente pode ser usada na seguinte consulta:

```
SELECT MInscRecente.ID, nome, curso
FROM MInscRecente, Estudante
WHERE MInscRecente.ID = Estudante.ID
      AND media > 7.5
```

que pode ser "traduzida" como:

```
SELECT MInscRecente.ID, nome, curso
FROM (SELECT ID, data, curso FROM Inscricao
      WHERE unome = 'MIT' AND data > '2016-09-01'
            AND decisao = 'N') MInscRecente,
     Estudante
WHERE MInscRecente.ID = Estudante.ID
      AND media > 7.5
```

Exemplo (1)

```
SELECT MInscRecente.ID, nome, curso
FROM (SELECT ID, data, curso FROM Inscricao
      WHERE unome = 'MIT' AND data > '2016-09-01'
            AND decisao = 'N') MInscRecente,
     Estudante
WHERE MInscRecente.ID = Estudante.ID
      AND media > 7.5
```

Versão "planificada" da consulta anterior:

```
SELECT Inscricao.ID, nome, curso
FROM Inscricao, Estudante
WHERE Inscricao.ID = Estudante.ID
      AND unome = 'MIT' AND data > '2016-09-01'
      AND decisao = 'N'
      AND media > 7.5
```

Exemplo (1)

Podemos ainda criar uma visão sobre outra visão:

```
CREATE VIEW AceitosMIT AS
    SELECT MInscRecente.ID, nome, curso
    FROM MInscRecente, Estudante
    WHERE MInscRecente.ID = Estudante.ID
          AND media > 7.5
```

Exemplo (2)

Visão: Estudantes que se inscreveram para o intercâmbio em mais de uma universidade

```
CREATE VIEW MultiInscricao AS
  SELECT DISTINCT Estudante.*
  FROM Estudante, Inscricao A1, Inscricao A2
  WHERE Estudante.ID = A1.ID
        AND Estudante.ID = A2.ID
        AND A1.unome <> A2.unome
```

Exemplo (3)

É possível fazer todas as junções em uma visão ("relação universal")

```
CREATE VIEW BDTodo AS
  SELECT Estudante.ID, nome, endereco, media,
         Universidade.unome, cidade, rank, data,
         curso, decisao
  FROM Estudante, Universidade, Inscricao
  WHERE Estudante.ID = Inscricao.ID
        AND Inscricao.unome = Universidade.unome
```

Agora podemos fazer uma consulta como ...

```
SELECT * FROM BDTodo
WHERE media < 7.5 AND rank > 3 AND curso = 'psicologia'
```

Exemplo (3)

A consulta ...

```
SELECT * FROM BDTodo
WHERE media < 7.5 AND rank > 3  AND curso = 'psicologia'
```

... equivale a:

```
SELECT Estudante.ID, nome, endereco, media,
       Universidade.unome, cidade, rank, data,
       curso, decisao
FROM Estudante, Universidade, Inscricao
WHERE Estudante.ID = Inscricao.ID
      AND Inscricao.unome = Universidade.unome
      AND media < 7.5 AND rank > 3 AND curso = 'psicologia'
```

Exemplo (4)

Visão com renomeamento de atributos

```
CREATE VIEW InscPorUniversidade(univer, num_inscritos) AS
     SELECT unome, COUNT(*)
     FROM  Inscricao
     GROUP BY unome
```

Modificações em visões

Fazem sentido?

- Não! – Afinal, visões não são armazenadas.
- Sim! – Afinal, visões constituem a "visão" completa que alguns usuários têm do banco de dados.

Conclusão: algumas modificações **não ambíguas** de visões são permitidas.

Visões atualizáveis

- Em visões suficientemente simples, é possível traduzir uma modificação da visão em uma modificação equivalente em uma tabela de base
- SQL2 possui uma definição formal sobre quando modificações em uma visão são permitidas $\Rightarrow$ as regras são bastante complexas!
- **Regra simplificada:** só podem ser modificadas visões que são definidas por meio da projeção (usando SELECT, não SELECT DISTINCT) de alguns atributos de tuplas selecionadas de **uma única relação** R (uma tabela ou uma visão atualizável)

Dois pontos importantes:

- A cláusula **WHERE** usada para a seleção das tuplas que aparecerão na visão não pode envolver R em uma subconsulta
- Os atributos de R não projetados na visão devem poder ser NULL ou devem possuir um valor default definido

Exemplo (1)

A remoção ...

```
DELETE FROM MInscRecente WHERE data <= '2016-09-15'
```

... é transformada em:

```
DELETE FROM Inscricao WHERE data <= '2016-09-15'
                        AND unome = 'MIT'
                        AND data > '2016-09-01'
                        AND decisao = 'N'
```

Exemplo (2)

Vamos definir uma nova visão:

```
CREATE VIEW BCC_SI AS
    SELECT ID, unome, curso
    FROM Inscricao
    WHERE curso = 'BCC' OR curso = 'SI'
```

A seguinte inserção na visão ...

```
INSERT INTO BCC_SI
       VALUES (123, 'Stanford', 'BCC')
```

... é traduzida como:

```
INSERT INTO Inscricao (ID, unome, data, curso, decisao)
       VALUES (123, 'Stanford', NULL, 'BCC', NULL)
```

Exemplo (2)

Ainda sobre a visão ...

```
CREATE VIEW BCC_SI AS
    SELECT ID, unome, curso
    FROM Inscricao
    WHERE curso = 'BCC' OR curso = 'SI'
```

... a seguinte inserção ...

```
INSERT INTO BCC_SI
       VALUES (123, 'Stanford', 'matemática')
```

... é traduzida como:

```
INSERT INTO Inscricao (ID, unome, data, curso, decisao)
       VALUES (123, 'Stanford', NULL, 'matemática', NULL)
```

$\Rightarrow$ **ANOMALIA: a tupla inserida no BD não fará parte da visão!**

Exemplo (3)

Retomando a visão MInscRecente ...

```
CREATE VIEW MInscRecente AS
  SELECT ID, data, curso
  FROM Inscricao
  WHERE unome = 'MIT' AND data > '2016-09-01' AND decisao = 'N'
```

... a seguinte inserção ...

```
INSERT INTO MInscRecente
       VALUES (123, '2016-09-12', 'economia')
```

... é traduzida como:

```
INSERT INTO Inscricao (ID, unome, data, curso, decisao)
       VALUES (123, NULL, '2016-09-12', 'economia', NULL)
```

- O SGBD não necessariamente deduz que unome deve ser 'MIT' e decisao deve ser 'N'
- Se ele não deduzir, a tupla não aparecerá na visão!

Remoção da visão

- Uma modificação extrema de uma visão é removê-la inteiramente
- Essa modificação pode ser feita mesmo quando a visão não é atualizável
- Exemplo: **DROP VIEW** BCC_SI
- O comando **DROP VIEW** remove a definição da visão (de modo que não é mais possível fazer consultas ou aplicar modificações sobre a visão)
- Mas a remoção da visão **não** afeta as tuplas das relações subjacentes (no caso do exemplo, as tabelas Estudante e Inscricao não seriam afetadas)
- Já o seguinte comando (por exemplo)

  DROP TABLE Estudantes

  não só acabaria com Estudantes (tuplas + esquema), como também tornaria a visão BCC_SI inutilizável

Visão materializada

- O resultado de uma consulta definida para uma **visão materializada** é colocado em um "cache", como uma tabela concreta, que pode ser atualizada a partir de suas relações de base de tempos em tempos
- O acesso a uma visão materializada é muito mais eficiente que o acesso a uma visão convencional, ao custo de se ter dados que podem (!) estar desatualizados
- Este tipo de visão é especialmente útil em aplicações de *data warehouse* (que envolvem grandes volumes de dados)
- Problema:
  - Todos os SGBDs implementam visões virtuais
  - Mas implementações de visões materializadas são menos frequentes

Exemplo

⇒ Para criar uma visão materializada, usa-se o comando
CREATE MATERIALIZED VIEW

Exemplo de visão materializada: Estudantes que se inscreveram para o intercâmbio em mais de uma universidade

```
CREATE MATERIALIZED VIEW MultiInscricao AS
  SELECT DISTINCT Estudante.*
  FROM Estudante, Inscricao A1, Inscricao A2
  WHERE Estudante.ID = A1.ID
        AND Estudante.ID = A2.ID
        AND A1.unome <> A2.unome
```

Visões Materializadas no PostgreSQL

No PostgreSQL, o comando `CREATE MATERIALIZED VIEW` funciona da seguinte maneira:

- A consulta de definição da visão materializada é executada e o seu resultado é usado para "popular" a visão no momento em que o comando `CREATE MATERIALIZED VIEW` é executado
  - Análogo ao que é feito na criação de tabelas por meio do comando `CREATE TABLE AS [consulta SQL]`
- Depois, a visão pode ser atualizada por meio da execução do comando `REFRESH MATERIALIZED VIEW [nome da visão]`
  - Já as tabelas criadas por meio de `CREATE TABLE AS [consulta SQL]` só podem ser atualizadas por meio dos comandos `INSERT`, `UPDATE` e `DELETE`

Referências Bibliográficas

- *Database Systems – The Complete Book*, Garcia-Molina, Ullman e Widom. 2002.
  Capítulo 6
- *Sistemas de Bancos de Dados* (6ª edição), Elmasri e Navathe. Pearson, 2010.
  Capítulo 5