Linguagem de Consulta e Linguagem de Manipulação de Dados

Raimundo C Vasconcelos

Linguagens de Consulta

- Linguagem através da qual os usuários obtêm informações do banco de dados .
- Nivel + alto que as linguagens de programação tradicionais
- Podem ser classificadas em:
  - Procedural: usuário “ensina” o sistema a realizar uma seqüência de operações no BD para obter o resultado desejado. Exemplos: Algebra Relacional.
  - Não-Procedural: o usuário descreve a informação desejadam, sem fornecer um procedimento para obtenção dessa informação. Exemplo: calculo relacional de tupla, calculo relacional de domínio.

Linguagens de Consulta

- Sistemas de Banco de Dados comerciais oferecem linguagens de consultas que incorporam elementos de ambos os enfoques (procedurais e não-procedurais).
  - SQL – enfoque procedural é maior.
  - QBE – enfoque não-procedural é maior.

- Embora SQL – Structured Query Language – seja uma linguagem de consulta ela incorpora operações de manipulação de dados (DML) e operações de definição de dados (SQL-DDL).

SQL – Structured Query Language

- Linguagem de consulta padrão para sistemas de BD relacionais.

- Características do modelo relacional:
  - cada tabela (relação) tem seu nome diferente das demais na mesma base de dados
  - cada coluna tem seu nome diferente das demais na mesma tabela
  - colunas contém os atributos (todos do mesmo domínio)
  - linhas contém informações de um registro (uma tupla) da tabela
  - cada célula pode conter no máximo um item de dado
  - ordem das linhas é irrelevante
  - ordem das colunas é irrelevante
  - nunca temos duas linhas iguais
  - chave primária

SQL – Structured Query Language

- Uma consulta básica em SQL é composta por 3 cláusulas:

  Select A1, A2, ... An

  From r1, r2 . . . . . rm

  Where P

  - cláusula select contém a lista de atributos A1, A2, . . . An que serão exibidos na tabela resultado (pode ser substituída por *)
  - cláusula from contém a lista da relações (tabelas) cujos dados serão combinados para resolução da consulta.
  - cláusula where contém o predicado da consulta (condições) e pode ser omitida.

SQL

- O resultado de uma consulta -> uma tabela (relação).
- SQL permite a ocorrência de tuplas repetidas na tabela resultado. Para evitar a repetição de tuplas usa-se :

  Select distinct A1, A2, . . . An

  From r1, r2, . . . rm

  Where P

- Conectivos lógicos:
  - and --> e; or --> ou; not --> negação.

SQL

- Esquemas das tabelas de um sistema bancário simplificado

  Agências (**cod_ag**, nome, cidade);
  Clientes (**cod_cli**, nome, endereco, cidade);
  Contas (**cod_ag**, **n_conta**, cod_cli, saldo);
  Emprestimos (**cod_ag**, **n_empr**, cod_cli, valor);

- Exemplos:
  - Forneça o código e nome das agências onde há empréstimos com valor maior que 10000.00.

SQL

```
Select distinct Agências.cod_ag, nome
From Agências, Empréstimos
Where valor > 10000.00 and Agências.cod_ag=
  Empréstimos.cod_ag;
```

- Forneça o nome e endereço dos clientes com contas na agência 010 ou 034.

SQL

```
Select distinct Nome, Endereco
From Clientes, Contas
Where Clientes.cod_cli = Contas.cod_cli and (cod_ag =
  010 or cod_ag = 034);
```

SQL - OPERAÇÕES DE CONJUNTO

- **UNION:** faz a união de tuplas de duas tabelas (compatíveis).

- Tabelas compatíveis:
  - Mesmo número de atributos (colunas)
  - Colunas correspondentes pertencem ao mesmo domínio.

- Forneça o nome e a cidade dos clientes da agência 0052.

SQL - OPERAÇÕES DE CONJUNTO

```
Select distinct nome, cidade
From Empréstimos, Clientes
Where cod_ag = 0052 and Clientes.cod_cli = Empréstimos.cod_cli
UNION
Select distinct nome, cidade
From Contas, Clientes
Where cod_ag = 0052 and Clientes.cod_cli = Contas.cod_cli;
```

SQL - OPERAÇÕES DE CONJUNTO

- **CONECTIVO IN**: Se a tupla pertencer à tabela o **in** retorna verdadeiro, senão retorna falso.

- Normalmente usa subconsultas para montar a tabela com um conjunto de tuplas para fazer os testes

- Forneça o nome e cidade dos clientes que têm empréstimos e contas (as duas coisas ao mesmo tempo) na agência 0052.

SQL - OPERAÇÕES DE CONJUNTO

```
Select distinct nome, cidade
From Empréstimos, Clientes
Where cod_ag = 0052 and
        Clientes.cod_cli = Empréstimos.cod_cli and
        Clientes.cod_cli in (
                Select distinct cod_cli
                From Contas
                Where cod_ag = 0052 )
```

SQL - OPERAÇÕES DE CONJUNTO

- **NOT IN:** testa se uma tupla não pertence a uma tabela

- Forneça o nome dos clientes que possuem empréstimos na ag. 0052, mas que não têm contas nesta agência.

SQL - OPERAÇÕES DE CONJUNTO

```
Select distinct nome, cidade
From Empréstimos, Clientes
Where cod_ag = 0052 and
        Clientes.cod_cli = Empréstimos.cod_cli and
        Clientes.cod_cli not in (
                Select distinct cod_cli
                From Contas
                Where cod_ag = 0052 )
```

- Forneça o nome das agências onde não há nenhum empréstimo com valor maior que 20.000,00.

SQL - OPERAÇÕES DE CONJUNTO

- Forneça o nome das agências onde não há nenhum empréstimo com valor maior que 20.000,00.

```
Select nome
From Agências
Where cod_ag not in (Select cod_ag
                     From Empréstimos
                     Where Valor > 20000.00);
```

SQL - Variáveis de Tuplas

- **VARIÁVEIS DE TUPLAS**: usadas para associar nomes alternativos (mais simples) às relações.
- Razões:
  - Simplificar referencias a nomes de tabelas em consultas
  - Possibilitar a comparação de duas tuplas (registros) da mesma tabela.

- Exemplo: Forneça os nomes dos clientes que tenham contas na mesma agência que o cliente de código 2, mas com saldo maior que o dele (2).

SQL - Variáveis de Tuplas

- Exemplo: Forneça os nomes dos clientes que tenham contas na mesma agência que o cliente de código 2, mas com saldo maior que o dele (2).

```
Select nome
From contas D1, contas D2, Clientes C
Where D1.cod_cli = 2 and D2.cod_cli <> 2 and
        D1.cod_ag = D2.cod_ag and D2.saldo > D1.saldo
 and D2.cod_cli = C.cod_cli;
```

SQL

- **CONECTIVO > ANY** (OU > SOME): usado para verificar se um valor é maior que algum dos valores armazenados em uma tabela.
- Monta-se uma tabela através de uma consulta aninhada para testar os valores.

- Análogos: **< ANY** (ou < some), **≤ ANY** (ou ≤ some), **≥ ANY** (ou ≥ some), **=ANY** (ou = some), **<> ANY** (ou <> some).

- Forneça o nome e endereço dos clientes que tenham saldos em suas contas maior que algum dos saldos do cliente de código 5.

SQL

- Forneça o nome e endereço dos clientes que tenham saldos em suas contas maior que algum dos saldos do cliente de código 5.

```
Select distinct C.Nome, C.Endereco
From Clientes C, Contas D
Where C.cod_cli = D.cod_cli and C.cod_cli <> 5
        and Saldo > any
        (Select Saldo
        From Contas
        Where cod_cli = 5);
```

SQL

- CONECTIVO **> all**: usado para verificar se um valor é maior que todos os valores armazenados em uma tabela. Análogos: **> all, ≤ all, ≥ all**.

- Forneça o código e nome das agências que fizeram empréstimos com valores maiores que todos os empréstimos da agência 0052.

SQL

- Forneça o código e nome das agências que fizeram empréstimos com valores maiores que todos os empréstimos da agência 0052.

```
Select distinct A.cod_ag, nome
From Agências A, Empréstimos E
Where A.cog_ag = E.cod_ag and
            valor > all (Select Valor
                         From Empréstimos
                         Where cod_ag = 0052);
```

SQL - Ordenação

- **Para obter resultados em SQL usamos a cláusula Order by A1, A2 . . . . . An**
  - As tuplas são ordenadas primeiro pelo atributo A1, onde há valores iguais, a ordenação é feita por A2 e assim sucessivamente.
  - O padrão de ordenação é ordem crescente (Asc), para obter ordem decrescente usaremos um desc após o nome do atributo.
  - A ordenação de um grande número de tuplas é uma operação de alto custo, portanto deve ser feito quando estritamente necessária.

- Forneça o nome dos clientes que têm contas com saldos maiores que 1000 em ordem alfabética.

SQL

- Forneça o nome dos clientes que têm contas com saldos maiores que 1000 em ordem alfabética.

```
Select distinct Nome, saldo
From Contas D, Clientes C
Where Saldo > 1000 and C.cod_cli=D.cod_cli
Order by Nome;
```

- Forneça o valor dos empréstimos e o nome e o endereço dos clientes que os fizeram, em ordem decrescente de valor e crescente de nome.

SQL

- Forneça o valor dos empréstimos e o nome e o endereço dos clientes que os fizeram, em ordem decrescente de valor e crescente de nome.

```
Select E.Valor, C.Nome, C.Endereço
From Empréstimos E, Clientes C
Where C.cod_cli = E.cod_cli
Order by E.Valor desc, C.Nome asc;
```

SQL – Funções de Grupo

- Funções que são aplicadas em um conjunto de tuplas de uma tabela. Opções:
  - **Média -->** AVG (Ai)
  - **Mínimo -->** MIN (Ai)
  - **Máximo -->** MAX (Ai)
  - **Soma (total) -->** SUM (Ai)
  - **Quantidade (Contar) -->** COUNT (A1) ou COUNT (*)

SQL – Funções de Grupo

- As funções de grupo podem ser aplicadas a todas as tuplas relacionadas pela consulta ou em grupos de tuplas selecionadas.

- Para aplicar a função de grupo em grupos de tuplas é necessário o agrupamento das tuplas usando a cláusula Group by A1, A2 . . . .

- Forneça o nome dos clientes de Jundiaí e o saldo médio de suas contas.

SQL – Funções de Grupo

- Forneça o nome dos clientes de Jundiaí e o saldo médio de suas contas.

```
Select C.Nome, avg(Saldo)
From Contas D, Clientes C
Where C.cod_cli = D.cod_cli and  Cidade = 'Jundiaí'
Group by C.Nome;
```

- Forneça o total de empréstimos da agência 0052.

SQL – Funções de Grupo

- Forneça o total de empréstimos da agência 0052.

```
Select Sum (Valor)
From Empréstimos
Where cod_ag = 0052;
```

- As condições da cláusula where são testadas antes de fazer os agrupamentos das tuplas. Assim sendo, condições envolvendo as funções de grupo não podem ser testadas na cláusula where.
- Para essas condições usaremos a cláusula **having P**.
- Encontre o nome das agências e a quantidade de empréstimos feitas em cada agência (mostrar só o nome das agências onde há mais do que 10 empréstimos).

SQL – Funções de Grupo

- Encontre o nome das agências e a quantidade de empréstimos feitas em cada agência (mostrar só o nome das agências onde há mais do que 10 empréstimos).

```
Select nome, count (n_empr)
From Agências A, Empréstimos E
WhereA.cod._ag = E.cod_ag
Group by nome
Having count (n_empr) > 10;
```

- Forneça o nome e endereço do cliente com o maior total de saldos em contas da agência 010.

SQL – Funções de Grupo

- Forneça o nome e endereço do cliente com o maior total de saldos em contas da agência 010.

```
Select C.nome, C.endereço
From Clientes C, Contas D
Where C.cod_cli = D.cod_cli and cod_ag = 010
Group by C.nome, C.endereço
Having Sum (Saldo) ≥ all
    (Select Sum (Saldo)
    From Contas
    Where cod._ag = 010
    Group by cod_cli);
```

SQL

- CONECTIVO **EXISTS** (E **NOT EXISTS**) : Usado para testar se existe (ou não) pelo menos uma tupla em uma tabela.
- Exists --> retorna verdadeiro se a tabela não está vazia
- Not Exists --> retorna verdadeiro se a tabela está vazia.
- Encontre o código dos clientes que tenham contas na agência 0052 e que não estão cadastrados na tabela de clientes.

SQL

- Encontre o código dos clientes que tenham contas na agência 0052 e que não estão cadastrados na tabela de clientes.

```
Select distinct D.cod_cli
From Contas D
Where cod_ag = 0052 and not exists (Select *
                                    From Clientes C
                                    Where D.cod_Cli = C.cod_Cli);
```

- Forneça o nome dos clientes que possuem empréstimos em todas as agências de Campinas.

SQL

- Forneça o nome dos clientes que possuem empréstimos em todas as agências de Campinas.

```
Select  C.nome
From Clientes C
Where not exists (Select cod_ag
                  From Agências
                  Where Cidade = 'Campinas' and Cod_Ag not in
                      (Select cod_ag
                      From Empréstimos E
                      Where C.cod_cli = E.cod_cli ) ) ;
```

SQL - Junção natural e junção exterior

- Junção natural e junção exterior (disponíveis no SQL Server)
  - Para os clientes de Jundiaí dê o seu nome e, se for o caso, número da conta e saldo

```
Select  nome, n_conta, saldo
From Clientes C, Contas D
Where C. cod_cli = D.cod_cli  and cidade = 'Jundiaí'
                    ou
Select  nome, n_conta, saldo
From Clientes INNER JOIN  Contas
on Clientes.cod_cli = Contas.cod_cli
Where  cidade = 'Jundiaí'
```

  - irá exibir apenas os clientes que possuem contas, quando queremos que apareçam todos os clientes, junto com suas contas, se houver

SQL

- Junção natural e junção exterior
  - A junção exterior esquerda entre Clientes e Contas (nessa ordem) resolve esse problema

```
Select  nome, n_conta, saldo
From Clientes C LEFT JOIN Contas D
on C.cod_cli = D.cod_cli
Where  cidade = 'Jundiaí'
```

  - Para os clientes que não possuem contas, será retornado nulo para número da conta e saldo

  - RIGHT JOIN
  - FULL JOIN

SQL

- Exemplo: Mostre o número do empréstimo, valor, código e nome da agencia que fez o empréstimo. Mesmo que a agência que não tenha empréstimo mostre seu codigo e nome

```
Select  n_empr, valor, A.cod_ag, nome
From Emprestimos E RIGHT JOIN Agencias A
                on E.cod_ag = A.cod_ag
```

SQL

- **Conectivo LIKE**: usado fazer comparações com partes de nomes.
  - Caracteres coringa : ? Substitui uma letra
    % Substitui um conjunto de letras
  - Exemplo 1: Forneça o número das contas, saldo e nome dos clientes, cujos nomes comecem com a letra A.
  - Exemplo 2: Repita a consulta anterior para as pessoas que tenham no nome a palavra José.

SQL

- Exemplo 1: Forneça o número das contas, saldo e nome dos clientes, cujos nomes comecem com a letra A.

```
Select n_conta, saldo, nome
From Clientes C , Contas D
Where c.cod_cli=D.cod_cli and C.nome like ‘A%’
```

- Exemplo 2: Repita a consulta anterior para as pessoas que tenham no nome a palavra José.

```
Select n_conta, saldo, nome
From Clientes C , Contas D
Where c.cod_cli=D.cod_cli and C.nome like ‘%José%’
```

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- Linguagens de consulta comerciais, na prática são linguagens de manipulação de dados, isto é, oferecem mecanismos para modificar as informações armazenadas no B.D. incluindo comandos para: inserir e remover tuplas e atualizar valores de campos de uma tabela.

- Para que essas operações sejam realizadas com sucesso elas não podem violar nenhuma restrição de integridade e o usuário que as solicitar deve ter as devidas permissões.

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- **INSERÇÃO:**
  - Há duas maneiras de inserir tuplas em tabelas usando SQL:
  - 1) Especificando valor a valor à tupla a ser inserida.

    Insert Into r
    values (V1, V2, V3, ... Vn)

    - Os valores V1, V2, V3, ... Vn devem pertencer ao domínio dos atributos correspondentes em “r”.
    - Campos para o quais não temos valores a inserir devem ser preenchidos com “null”, neste caso, a inserção não será permitida se o campo tiver sido definido como obrigatório, isto é, “not-null”.

    - Cadastre o cliente Rodrigo de Jundiaí com código 10 e que abriu uma conta com o n.º 1520-1 na agência 010, com saldo inicial de 50,00.

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- Cadastre o cliente Rodrigo de Jundiaí com código 10 e que abriu uma conta com o n.º 1520-1 na agência 010, com saldo inicial de 50,00.

```
Insert into Clientes
Values (10, 'Rodrigo', null, 'Jundiaí');
```

- OU

```
Insert into Clientes (cod_cli, nome, cidade)
Values (10, 'Rodrigo', 'Jundiaí');

Insert into Contas
Values (010, '1520-1', 10, 50.00);
```

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- 2) Faz-se uma consulta cujo resultado é um conjunto de tuplas que será inserido na tabela “r”.

  Insert into r
  Select A1, A2, ... An
  From r1, r2, ... rm
  Where p

  - Os valores dos atributos A1, A2, ... An, serão cadastrados nos atributos correspondentes de r (devem pertencer aos domínios dos mesmos).

  - Abra uma conta para cada cliente que tem um empréstimo na agência 0052. Essa nova conta terá o mesmo número que o empréstimo e saldo inicial de 20,00.

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- Abra uma conta para cada cliente que tem um empréstimo na agência 0052. Essa nova conta terá o mesmo número que o empréstimo e saldo inicial de 20,00.

```
Insert into Contas
Select 0052, n_empr, cod_cli, 20.00
From Empréstimos
Where Cod_Ag = 0052;
```

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- **EXCLUSÃO:**

  delete r **ou** **delete from r**

  where p **where p**

- Todas as tuplas da relação (tabela) “r” que satisfazem o predicado “p” são excluídas integralmente dessa tabela.
- Cada comando delete opera em uma única tabela => é necessário usar um comando para cada tabela.
- A cláusula **where** pode ser omitida => todas as tuplas da relação “r” serão excluídas (a tabela “r” ficará vazia).
- A cláusula **where** pode ser tão complexa quanto nos comandos de seleção (select – from – where).

- Exclua todas as contas de empréstimos.

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- Exclua todas as contas de empréstimos.

```
Delete Empréstimos;
```

- **A ordem das exclusões é importante**.

- Exclua as agências de Jundiaí bem como as contas dessas agências.

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- Exclua as agências de Jundiaí bem como as contas dessas agências.

```
Delete from Contas
Where cod_ag in  (Select cod_ag
                  From Agências
                  Where cidade = 'Jundiaí');

Delete from Agências
Where Cidade = 'Jundiaí';
```

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- **ATUALIZAÇÃO:**

  Update r

  Set Ai = Vi

  Where p

- As tuplas da tabela “r” que satisfazem o predicado “p” tem o atributo Ai atualizado.
- A ordem de atualização é **<u>importante !!!!!</u>**

- Atualize o saldo das contas da agência 010 com 3% de juros.

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- Atualize o saldo das contas da agência 010 com 3% de juros.

```
Update Contas
Set Saldo = Saldo*1,03
Where Cod_Ag = 010;
```

- Aplique as seguintes taxas de juros nas contas da agência 0052:
  - 3% para contas com até 1.000,00
  - 5% para as demais

SQL - MODIFICANDO INFORMAÇÕES DO B.D.

- Aplique as seguintes taxas de juros nas contas da agência 0052:
  - 3% para contas com até 1.000,00
  - 5% para as demais

```
Update Contas
Set saldo = saldo * 1.05
Where cod_ag = 0052 and saldo > 1000.00;

Update Contas
Set saldo = saldo * 1.03
Where cod_ag = 0052 and saldo <= 1000.00;
```

Referência Bibliográficas

- Silberchatz, A. ; Korth, H. F. ; Sudarshan, S.
  Sistema de Banco de dados - 5a. edição
  Editora Campus

- DATE, C. J.
  Introdução a sistemas de bancos de dados - 8 edição
  Editora Campus.