SQL – *Structured Query Language*

Disciplina: Banco de Dados I

Prof: Aglaê Pereira Zaupa

Unoeste – Universidade do Oeste Paulista

FIPP – Faculdade do Oeste Paulista

Sumário

- Revisão de consultas básicas (Projeção e Seleção)
- Junção
- Renomeação
- União, Intersecção e Diferença
- Junção externa sem operador específico
- Consultas aninhadas
- Quantificador Existencial e Universal
- Campos vazios
- Junção externa
- Funções
- Cláusulas GROUP BY e HAVING
- Referências

DML – Linguagem de Manipulação de Dados

- DML – Data Manipulation Language
- SQL oferece quatro instruções para manipulação da base de dados, sendo uma delas para consultas e as outras três para atualização do conteúdo das tabelas existentes na base
  - Select – consulta os dados armazenados nas tabelas
  - Insert - insere uma ou mais linhas em uma tabela
  - Update - altera os dados de uma ou mais linhas de uma tabela
  - Delete - exclui uma ou mais linhas de uma tabela

Select – Estrutura básica

- O modelo básico de uma instrução de consulta em SQL é

```
SELECT [DISTINCT|ALL] <lista de colunas>
FROM <lista de tabelas>
[WHERE <critério>]
```

- O modelo básico de execução da instrução SQL é o seguinte:
  - É feito o produto cartesiano das tabelas envolvidas
  - São selecionadas as linhas da tabela que obedecem ao critério
  - É feita a projeção sobre as colunas que vão ao resultado
- Na prática, a SQL pode executar uma expressão de uma forma mais inteligente, evitando o acesso a dados desnecessários para que seja processada mais eficientemente

DML – Revisão de Consultas básicas
Projeção e Seleção

- Obter dados de todos os clientes

```
SELECT cli_nome
FROM clientes
```

ou

```
SELECT *
FROM clientes
```

- Obter os códigos e nomes de clientes do bairro 1 que tem seu nome iniciado por Jo

```
SELECT cli_codigo, cli_nome
FROM clientes
WHERE bai_codigo = 1 AND
      cli_nome like 'Jo%'
```

DML – Revisão de Consultas básicas
Projeção e Seleção

- No primeiro caso, se houver dois clientes com o mesmo nome, haverá duas linhas idênticas no resultado
- O resultado é um multi-conjunto (“bag") e não conjunto de linhas
- SQL não elimina linhas duplicadas por default
  - eliminação dessas linhas é uma operação cara em termos de performance (normalmente, envolve ordenação de linhas (“sort")

Exemplos de projeção - DISTINCT

- Cláusula DISTINCT especifica a eliminação de linhas duplicadas (equivalente à projeção em álgebra relacional)

```
SELECT DISTINCT cli_nome
FROM clientes
```

Cláusula WHERE

- A expressão lógica na cláusula WHERE pode envolver uma série de operadores
  - = <> < <= > >=
  - status BETWEEN 5 AND 12
  - AND OR NOT
  - nome LIKE '%Marques%'
    - testa se Marques faz parte do campo Nome
    - trata-se de reconhecimento de padrões
    - qualquer caractere é _
    - qualquer seqüência é %

Condições na cláusula WHERE

- Expressões podem aparecer no resultado da consulta

```
SELECT piz_numero, pit_tamanho, pit_preco/2.3 as precodolar,
  NULL
FROM pizza_tamanho
```

- Expressões podem envolver constantes ou qualquer coluna das tabelas
- As operações permitidas são
  - adição
  - subtração
  - multiplicação
  - divisão

Junção

- Para cada item de pedido com quantidade maior que 3, obter o código do pedido, o número, o tamanho e o nome da pizza

```
SELECT ped_numero, Pedidos_itens.piz_numero, pit_tamanho, piz_nome
FROM Pedidos_itens, Pizza
WHERE ite_qtde > 3 AND
        Pedidos_itens.piz_numero=Pizza.piz_numero
```

- A consulta envolve duas tabelas: a cláusula WHERE especifica os atributos de junção das duas tabelas
- Corresponde à seqüência Seleção-Junção-Projeção comum na álgebra relacional
- Observar a qualificação da coluna piz_numero (Sempre que um nome de coluna aparecer em duas ou mais tabelas, ele deve ser qualificado. Apenas nomes não ambíguos não necessitam qualificação)

Junção

- Obter os nomes dos clientes que pediram pizzas de tamanho ‘P’

```
SELECT cli_nome
FROM clientes, pedidos, pedidos_itens
WHERE pit_tamanho = ‘P’ AND
        pedidos.cli_codigo = clientes.cli_codigo AND
        pedidos.ped_numero = pedidos_itens.ped_numero
```

- Sintaxe básica não inclui uma operação como a junção natural
- O usuário é obrigado a especificar os atributos de junção

Renomeação

- Obter os nomes dos clientes que pediram pizzas de tamanho ‘P’

```
SELECT cli_nome
FROM clientes c, pedidos as p, pedidos_itens pi
WHERE pit_tamanho = ‘P’ AND
        p.cli_codigo = c.cli_codigo AND
        p.ped_numero = pi.ped_numero
```

Renomeação

- Obter todos os pares de nomes de clientes que se encontram no mesmo bairro

```
SELECT c1.cli_nome, c2.cli_nome
FROM clientes c1, clientes c2
WHERE c1.bai_codigo = c2.bai_codigo AND
        c1.cli_codigo < c2.cli_codigo
```

- Neste caso, é necessário renomear a tabela de clientes, já que a consulta faz referência a duas linhas da mesma tabela
- Isso é feito por meio de aliases definidos na cláusula FROM
- Aliases têm função análoga à das variáveis de tupla do cálculo relacional
- A comparação de códigos de clientes serve apenas para eliminar duas linhas do mesmo cliente e evitar que o mesmo par de clientes apareça no resultado também na ordem inversa

Renomeação

- Para cada empregado que possui gerente, obter o nome do empregado seguido do nome de seu gerente

EMP(CodEmp, NomeEmp, CodEmpGer)
CodEmpGer referencia EMP

```
SELECT EMP.NomeEmp, GER.NomeEmp
FROM EMP, EMP as GER
WHERE EMP.CodEmpGer=GER.CodEmp
```

União

- Obter os códigos dos clientes que sejam do bairro 1 ou que tenham efetuado pedidos pelo funcionário ‘Crispim’

```
SELECT cli_codigo
FROM clientes
WHERE bai_codigo = 1
UNION
  SELECT cli_codigo
  FROM pedidos p, funcionarios f
  WHERE p.fun_codigo = f.fun_codigo AND
        fun_nome = ‘Crispim’
```

  - O operador de união é equivalente ao da álgebra relacional
  - A união elimina duplicatas
  - O operador UNION ALL não elimina duplicatas - implementação mais eficiente

Intersecção e diferença

- Em alguns SGBD também são implementadas, além da união (Firebird / Interbase não implementam)
  - intersecção (operador INTERSECT) e
  - diferença (operador EXCEPT ou MINUS)

Exemplos - junção externa sem operador específico

- Para cada empregado, obter o nome do empregado seguido do nome de seu gerente, caso ele o possua

```
SELECT EMP.NomeEmp, GER.NomeEmp
FROM EMP, EMP as GER
WHERE EMP.CodEmpGer=GER.CodEmp
UNION
  SELECT NomeEmp, NULL
  FROM EMP
  WHERE EMP.CodEmpGer IS NULL
```

Exemplos – consultas aninhadas

- Obter os códigos dos clientes que efetuaram pedidos com o funcionário ‘Jorge’

```
SELECT cli_codigo
FROM pedidos, funcionarios
WHERE fun_nome = ‘Jorge’ AND
        pedidos.fun_codigo = funcionarios.fun_codigo
```

- Neste caso, o resultado da consulta envolve apenas colunas da tabela pedidos, mas a cláusula FROM referencia também a tabela funcionarios

Exemplos - consultas aninhadas

- Para este tipo de consulta, a solução mais natural é por meio de consultas aninhadas (ou consultas embutidas – 'sub-select')

```
SELECT cli_codigo
FROM pedidos
WHERE fun_codigo IN
        (SELECT fun_codigo
         FROM funcionarios
         WHERE fun_nome = 'Jorge')
```

Exemplos – consultas aninhadas

- Obter os nomes dos clientes que pediram pizzas de tamanho ‘P’

```
SELECT DISTINCT cli_nome
FROM pedidos p, pedidos_itens pi, clientes c
WHERE pit_tamanho = ‘P' AND
        p.ped_numero = pi.ped_numero AND
        p.cli_codigo = c.cli_codigo
```

- Também pode ser resolvida com consultas aninhadas:

```
SELECT cli_nome FROM clientes
WHERE cli_codigo IN
    (SELECT cli_codigo FROM pedidos
     WHERE ped_numero IN
            (SELECT ped_numero FROM pedidos_itens
             WHERE pit_tamanho=‘P'))
```

Exemplos – consultas aninhadas

- Obter os números dos pedidos realizados para a pizza de calabresa:

```
SELECT ped_numero
FROM pedidos_itens, pizza
WHERE piz_nome = ‘calabresa' AND
        pedidos_itens.piz_numero = pizza.piz_numero
```

- Solução com consultas aninhadas:

```
SELECT ped_numero
FROM pedidos_itens
WHERE piz_numero IN
      (SELECT piz_numero
       FROM pizza
       WHERE piz_nome = ‘calabresa')
```

Exemplos – consultas aninhadas

- Outra solução com consultas aninhadas (alternativa):

```
SELECT ped_numero
FROM pedidos_itens
WHERE ‘calabresa' IN
(SELECT piz_nome
FROM pizza
WHERE piz_numero=pedidos_itens.piz_numero)
```

Prioridade para o atributo da consulta mais interna

- A última solução pode ser expressa como: “obter os números dos pedidos para os quais ‘calabresa' é o nome de uma pizza pedida"
- **Idealmente, um otimizador de consultas deveria estar em condições de executar qualquer das três alternativas com a mesma performance**
- **O programador não deveria ser obrigado a conhecer a solução ótima**

Quantificador Existencial

- Obter os nomes dos clientes para os quais há ao menos um pedido

```
SELECT cli_nome
FROM clientes
WHERE EXISTS
   (SELECT *
    FROM pedidos
    WHERE pedidos.cli_codigo = clientes.cli_codigo)
```

- A cláusula EXISTS tem função análoga ao quantificador existencial no cálculo relacional

Quantificador Existencial

- Obter os nomes dos clientes para os quais não há pedidos

```
SELECT cli_nome
FROM clientes
WHERE NOT EXISTS
   (SELECT *
    FROM pedidos
    WHERE pedidos.cli_codigo = clientes.cli_codigo)
```

Quantificador Universal

- Obter os códigos dos fornecedores que possuem embarques para todas as peças de cor vermelha
- Obter os nomes das pizzas que são oferecidas em todos os tamanhos no cardápio
- Essa operação seria resolvida usando o operador de divisão de álgebra relacional ou o quantificador universal de cálculo relacional
- Em SQL, não há cláusula análoga ao quantificador universal - é necessário usar a negação do quantificador existencial

Quantificador universal
Resolução com Quantificador existencial

- Obter os nomes das pizzas que são oferecidas em todos os tamanhos no cardápio

```
SELECT piz_nome
FROM pizza
WHERE NOT EXISTS
   (SELECT distinct(pit_tamanho)
    FROM pizza_tamanho tamanhos
    WHERE
       NOT EXISTS
          (SELECT *
           FROM pizza_tamanho pt
           WHERE pt.piz_numero =pizza.piz_numero AND
                    tamanhos.pit_tamanho=pt.pit_tamanho))
```

- Interpretação: Obter os nomes das pizzas, tal que não exista um tamanho de pizza para a qual não exista o tamanho relacionado

Campos vazios

- Obter os nomes dos funcionários para os quais o número do celular foi informado (não está vazio)

```
SELECT fun_nome
FROM funcionarios
WHERE fun_celular IS NOT NULL
```

- Comparação (=, <, >, ...) com vazio resulta em UNKNOWN
- Há uma lógica de três valores (TRUE, FALSE e UNKNOWN)
- Uma expressão lógica pode ser testada por um valor lógico:
  - teste_de_comparação IS [TRUE | FALSE | UNKNOWN]
  - ((vendas - quota) > 1000) IS UNKNOWN

Campos vazios

- Suponha três clientes que moram na 'Av. da Saudade', um que mora na 'Av. Cel. Marcondes' e dois clientes com endereço não informado. Quantos clientes retornam?
  - Consulta 1:

```
SELECT cli_nome
FROM clientes
WHERE cli_endereco = 'Av. da Saudade'
```

  - Consulta2:

```
SELECT cli_nome
FROM clientes
WHERE cli_endereco <> 'Av. da Saudade'
```

Campos vazios

- Suponha três clientes que moram na 'Av. da Saudade', um que mora na 'Av. Cel. Marcondes' e dois clientes com endereço não informado. Quantos clientes retornam?
  - Consulta 3:

```
SELECT cli_nome
FROM clientes
WHERE cli_endereco <> 'Av. da Saudade' OR
        cli_endereco IS NULL
```

Junção externa

- Modelo básico do SQL não oferece a possibilidade de fazer junção externa. Exemplo:
- Obter o código de cada cliente, junto com o código de cada pedido realizado. Caso o cliente não tenha pedidos, obter seu código seguido de NULL

```
SELECT cli_codigo, ped_numero
FROM pedidos
  UNION
     SELECT c.cli_codigo, NULL AS ped_numero
     FROM clientes c
     WHERE cli_codigo NOT IN
                  (SELECT cli_codigo FROM pedidos)
```

Junção externa

- SQL1 não continha junção externa
- Vários produtos implementaram extensões
- Exemplo de SQL Server (pré-v.7) e Sybase:

```
SELECT c.cli_codigo, ped_numero
FROM clientes c, pedidos p
WHERE c.cli_codigo *= p.cli_codigo
```

Junção externa no Oracle

- Em Oracle, deve-se informar qual é a coluna da tabela que deve ter a linha NULL imaginária incluída (sinal de adição)

```
SELECT c.cli_codigo, ped_numero
FROM clientes c, pedidos p
WHERE c.cli_codigo = p.cli_codigo (+)
```

Junção externa no Interbase / FireBird

```
SELECT c.cli_codigo, p.ped_numero
FROM clientes c
        LEFT JOIN pedidos p
        ON c.cli_codigo = p.cli_codigo
```

Junção externa no SQL2

- SQL2 possui um modelo completo com vários tipos de junções, incluindo a junção theta, a equijunção, a junção natural e a junção externa
- A equijunção e a junção natural ainda não foram implementadas pela maioria dos SGBD

```
SELECT c.cli_codigo, ped_numero
FROM clientes c
        LEFT JOIN Pedidos p
        ON c.cli_codigo = p.cli_codigo
```

Junção externa no SQL2 - equijunção

- A sintaxe apresentada na transparência anterior corresponde a uma junção theta de álgebra relacional
- SQL2 provê tanto a equijunção quanto a junção natural

- Equijunção (cláusula USING)

```
SELECT c.cli_codigo, ped_numero
FROM clientes c
        LEFT JOIN pedidos USING (cli_codigo)
```

Junção externa no SQL2 – junção natural

- Junção Natural (cláusula NATURAL):

```
SELECT c.cli_codigo, ped_numero
FROM clientes c
        NATURAL JOIN pedidos
```

SQL2 - Opções de junção

- Opções de junção
  - tabela1 [INNER] JOIN tabela2
  - tabela1 FULL [OUTER] JOIN tabela2
  - tabela1 LEFT [OUTER] JOIN tabela2
  - tabela1 RIGHT [OUTER] JOIN tabela2
- tabela1 e tabela2 podem também ser um outro SELECT
- Produto cartesiano
  - tabela1 CROSS JOIN tabela2

SQL –modelo estendido de consulta

- SQL no modelo básico visto até aqui não possui o mesmo poder de expressão de álgebra e cálculo relacional
- Isso significa que SQL no modelo básico não oferece a possibilidade de executar consultas recursivas ou que envolvam a agregação de dados
- No modelo estendido, SQL possibilita a manipulação de agregados de dados
- SQL padrão (SQL2) não possui facilidades para tratar recursividade, mas SQL3 e diversos produtos oferecem extensões (Ex.: instrução CONNECT de Oracle)

SQL –modelo estendido de consulta

- A sintaxe de uma instrução de consulta no modelo estendido é:

  SELECT [DISTINCT|ALL]<lista de colunas>

  FROM <lista de tabelas>

  [WHERE <critério>]

  [GROUP BY <lista de colunas>

  [HAVING <critério>]]

  [ORDER BY <lista de colunas>]

SQL –modelo estendido de consulta

O modelo mental de execução da instrução SQL é o seguinte:

1. É feito o produto cartesiano das tabelas envolvidas
2. São selecionadas as linhas da tabela que obedecem ao critério da cláusula WHERE
3. São criados grupos de linhas que contenham valores idênticos nas colunas GROUP BY
4. São selecionados os grupos que obedecem ao critério da cláusula HAVING
5. É feita a classificação do resultado pelos valores da cláusula ORDER BY
6. É feita a projeção sobre as colunas que vão ao resultado

SQL - Funções

- Funções para computar valores
  - média: AVG
  - mínimo: MIN
  - máximo: MAX
  - total: SUM
  - contagem: COUNT
- As funções podem ser aplicadas sobre
  - Toda a tabela (consulta sem GROUP BY)
  - Grupos de linhas (definidos pela cláusula GROUP BY)

SQL - Funções
Funções sobre toda a tabela

- Obter a quantidade de clientes da base de dados

```
SELECT COUNT(*)
FROM clientes
```

- Obter o número de endereços em que há clientes

```
SELECT COUNT(DISTINCT cli_endereco)
FROM clientes
```

- Obter o número de clientes com endereço informado

```
SELECT COUNT(cli_endereco)
FROM clientes
```

- Obter o tempo máximo de espera

```
SELECT MAX(bai_temp_espera)
FROM bairros
```

Cláusula GROUP BY

- Obter a quantidade de pedidos de cada cliente

```
SELECT cli_codigo,COUNT(*)
FROM pedidos
GROUP BY cli_codigo
```

- Obter a quantidade de pedidos de cada funcionário

```
SELECT fun_codigo,COUNT(*)
FROM pedidos
GROUP BY fun_codigo
```

Cláusula GROUP BY

- Em SELECT que usa GROUP BY, o resultado da consulta pode incluir somente
  - uma constante
  - uma função de agregação calculada sobre o grupo
  - uma coluna que faz parte da cláusula GROUP BY
  - uma expressão envolvendo os acima
- Exemplo de expressão incorreta

```
SELECT c.cli_codigo, c.cli_nome, count(*)
FROM clientes c, pedidos p
WHERE c.cli_codigo = p.cli_codigo
GROUP BY c.cli_codigo
```

- Solução: incluir cli_nome na cláusula GROUP BY

Cláusula GROUP BY

- Obter o número de itens pedidos com quantidade de pizza maior que 3 de cada cliente

```
SELECT p.cli_codigo,COUNT(*)
FROM pedidos_itens pi, pedidos p
WHERE ite_qtde > 3 AND
            p.ped_numero = pi.ped_numero
GROUP BY p.cli_codigo
```

Cláusula GROUP BY

- Obter a quantidade total pedida de pizzas de calabresa para cada cliente

```
SELECT cli_codigo, SUM(ite_qtde)
FROM pedidos p, pedidos_itens pi
WHERE p.ped_numero = pi.ped_numero AND
            pi.piz_numero IN
            (SELECT piz_numero
             FROM pizza
             WHERE piz_nome = 'calabresa')
GROUP BY cli_codigo
```

Cláusula GROUP BY

- Obter a quantidade total pedida de cada pizza. Exibir o resultado por ordem descendente de quantidade

```
SELECT p.piz_numero, piz_nome, SUM(ite_qtde)
FROM pedidos_itens pi, pizza p
WHERE p.piz_numero = pi.piz_numero
GROUP BY p.piz_numero, piz_nome
ORDER BY 3 DESC, Piz_nome ASC
```

- No SQL padrão, a referência à terceira coluna do resultado, que tem dados computados, deve ser feita pelo número da coluna, na forma ORDER BY 3 DESC

Cláusula HAVING

- Cláusula HAVING seleciona grupos

```
SELECT piz_numero, SUM(ite_qtde)
FROM pedidos_itens
GROUP BY piz_numero
HAVING COUNT(*) > 3 AND SUM(ite_qtde) < 200
```

Cláusula HAVING

- Obter os códigos dos clientes que tenham pedidos de mais de 3 pizzas de mussarela, junto com a quantidade de vezes que pediu a pizza de mussarela

```
SELECT p.cli_codigo, COUNT(*)
FROM pedidos_itens pi, pedidos p
WHERE p.ped_numero = pi.ped_numero AND
            piz_numero IN
               (SELECT piz_numero
                FROM pizza
                WHERE piz_nome = 'mussarela')
GROUP BY cli_codigo
HAVING SUM(ite_qtde) > 3
```

Referências

- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados.** Makron Books, 3ª Ed., 1999
- ELMASRI, Ramez; NAVATHE, S. B. **Sistemas de Banco de Dados.** Addison Wesley, 4ª Ed., 2005.