[MAC0313] Introdução aos Sistemas de Bancos de Dados

Aula 19
Linguagem SQL (Parte 3)

Consultas Envolvendo Operações de Teoria dos Conjuntos, Agrupamento/Agregação e Junções

18 de outubro de 2016

Profª Kelly Rosa Braghetto

(Adaptação dos slides do prof. Jeffrey Ullman, da *Stanford University*)

União, Intersecção e Diferença

- União, intersecção e diferença de relações são expressas nas seguintes formas, todas envolvendo subconsultas:
  - (<subconsulta>) **UNION** (<subconsulta>)
  - (<subconsulta>) **INTERSECT** (<subconsulta>)
  - (<subconsulta>) **EXCEPT** (<subconsulta>)

Exemplo: Intersecção

- Usando

  Apreciador(nome_cliente, nome_refri), Venda(nome_lanch, nome_refri, preco) e Frequentador(nome_cliente, nome_lanch), encontre os clientes e refris tais que:

  1. O cliente aprecia o refri e
  2. O cliente frequenta pelo menos uma lanchonete que vende o refri

“Truque”:
a subconsulta é uma tabela armazenada.

Solução

Refris vendidos nas lanchonetes que o cliente frequenta.

```
(SELECT * FROM Apreciador)
   INTERSECT
(SELECT nome_cliente, nome_refri
 FROM Venda, Frequentador
 WHERE Frequentador.nome_lanch
            = Venda.nome_lanch
);
```

Semântica de multiconjunto

- Embora os comandos SELECT-FROM-WHERE usem a “semântica de multiconjunto”, o padrão para a união, intersecção e diferença é a **semântica de conjunto**.
  - Ou seja, as duplicações de tuplas são eliminadas quando a operação é aplicada.

Motivação: eficiência

- É muito caro eliminar duplicações de uma relação.
- A operação de projeção considera somente uma tupla por vez (não requer a ordenação das tuplas) – por isso a consulta fica mais eficiente quando não é preciso remover duplicações.
- Para intersecções e diferenças, é mais eficiente ordenar as relações antes.
  - Nesse caso, as duplicações já podem ser facilmente eliminadas.

Controlando a eliminação de duplicações

- É possível forçar que o resultado de uma consulta seja um conjunto (ou seja, não tenha repetições) usando a cláusula **DISTINCT**:

  SELECT DISTINCT . . .

- Para forçar que o resultado seja um multiconjunto (ou seja, que as duplicações não sejam eliminadas) use a cláusula **ALL**, como em:

  . . . UNION ALL . . .

Exemplo: DISTINCT

- A partir de

  Venda(nome_lanch, nome_refri, preco), encontre todos os diferentes preços cobrados por refrigerantes:

```
SELECT DISTINCT preco
FROM Venda;
```

- Sem o DISTINCT, cada preço poderia ser listado tantas vezes quanto o número de pares (nome_lanch,nome_refri) associados a esse preço na tabela.

Exemplo: ALL

- Usando as relações Frequentador(nome_cliente, nome_lanch) e Apreciador(nome_cliente, nome_refri):

```
(SELECT nome_cliente FROM Frequentador)
EXCEPT ALL
(SELECT nome_cliente FROM Apreciador);
```

- Lista os clientes que frequentam mais lanchonetes do que o número de refris que eles gostam (e faz isso tantas vezes quanto for a diferença entre essas contagens).

Agregações

- **SUM**, **AVG**, **COUNT**, **MIN** e **MAX** podem ser aplicados a uma coluna na cláusula SELECT para produzir a agregação da referida coluna.
- Além disso, **COUNT(*)** conta o número de tuplas.

Exemplo: Agregação

- A partir de
  Venda(nome_lanch, nome_refri, preço),
  encontre o preço médio de Fanfa:

```
SELECT AVG(preço)
FROM Venda
WHERE nome_refri = 'Fanfa';
```

Eliminando duplicações em uma agregação

- Pode-se usar o DISTINCT dentro de uma agregação
- Exemplo: encontre o número de preços *diferentes* cobrados pela Fanfa:

```
SELECT COUNT(DISTINCT preço)
FROM Venda
WHERE nome_refri = 'Fanfa';
```

Valores NULL são ignorados na agregação

- Um NULL nunca contribui para uma soma, média ou contagem, e nunca pode ser nem o mínimo, nem o máximo de uma coluna
- Mas se não existir valores não nulos em uma coluna, então o resultado da agregação é NULL
  - Exceção: COUNT de um conjunto vazio é 0

Exemplo: efeito de NULLs

```
SELECT count(*)
FROM Venda WHERE
nome_refri = ’Fanfa’;
```

O número de lanchonetes que vendem Fanfa.

```
SELECT count(preço)
FROM Venda WHERE
nome_refri = ’Fanfa’;
```

O número de lanchonetes que vendem Fanfa a um preço conhecido.

Agrupamento

- Depois de uma expressão SELECT-FROM-WHERE, podemos adicionar GROUP BY e uma lista de atributos.
- A relação resultante do SELECT-FROM-WHERE é agrupada de acordo com os valores de todos os referidos atributos e qualquer agregação é aplicada somente dentro de cada grupo.

Exemplo: agrupamento

- A partir de

  Venda(nome_lanch, nome_refri, preço), encontre o preço médio de cada refri:

```
SELECT nome_refri, AVG(preço)
FROM Venda
GROUP BY nome_refri;
```

nome_refri	AVG(preço)
Fanfa	2.33
...	...

Exemplo: agrupamento

- A partir de Venda(nome_lanch, nome_refri, preço) e Frequentador(nome_cliente, nome_lanch), encontre, para cada cliente, o preço médio da Fanfa nas lanchonetes que ele frequenta :

```
SELECT nome_cliente, AVG(preço)
FROM Frequentador, Venda
WHERE nome_refri = 'Fanfa' AND
Frequentador.nome_lanch =
                    Venda.nome_lanch
GROUP BY nome_cliente;
```

Computa todas as tuplas cliente-lanch-preço para Fanfa.

Depois, as agrupa pelo cliente.

Restrição no SELECT: listas com agregação

- Se um agrupamento é usado, então cada elemento da lista do SELECT precisa ser:
  1. Uma agregação, ou
  2. Um atributo da lista do GROUP BY.

Exemplo de consulta incorreta

- Alguém pode pensar que é possível encontrar a lanchonete que vende Fanfa mais barato usando:

```
SELECT nome_lanch, MIN(preço)
FROM Venda
WHERE nome_refri = 'Fanfa';
```

- Mas essa consulta **NÃO** é permitida em SQL.

Cláusulas HAVING

- HAVING <condição> pode aparecer depois da cláusula GROUP BY
- Se aparecer, a condição é aplicada sobre cada grupo. Grupos que não satisfazem a condição são eliminados da resposta da consulta

Exemplo: HAVING

- A partir de
  Venda(nome_lanch, nome_refri, preço) e
  Refrigerante(nome, fabricante),
  encontre o preço médio dos refris que são servidos em pelo menos 3 lanchonetes ou que são fabricados pela Cola-Coca.

Solução

```
SELECT nome_refri, AVG(preço)
FROM Venda
GROUP BY nome_refri
HAVING COUNT(nome_lanch) >= 3   OR
nome_refri  IN
      (SELECT nome
       FROM Refrigerantes
       WHERE fabricante =  'Cola-Coca');
```

Grupos de refri com pelo menos 3 lanchonetes não nulas e também grupos em que o fabricante é a Cola-Coca.

Refris fabricados pela Cola-Coca.

Requisitos para as condições do HAVING

- Vale qualquer coisa dentro de uma subconsulta
- Fora de subconsultas, o HAVING pode referenciar um elemento somente se ele for:
  1. Um atributo agrupador, ou
  2. Uma agregação

  (essa é a mesma condição usada para cláusulas SELECT com agregação)

Expressões de Junção (JOIN)

- SQL possui várias versões de junções
- Mas é sempre possível obter o mesmo efeito deles por meio de uma consulta do tipo SELECT-FROM-WHERE.
- A expressões JOIN podem ser usadas no lugar de relações em uma cláusula FROM.

Produto Cartesiano

- É o tipo de junção mais simples:

  SELECT * FROM **R CROSS JOIN S**;

- As relações envolvidas no produto também podem ser subconsultas parentizadas (isso vale para todos os tipos de JOIN)
- O produto cartesiano sozinho raramente é útil

Junção Natural

- Forma da junção natural: **R NATURAL JOIN S**
- A condição de junção é a igualdade sobre os pares de atributos das duas relações que possuem o mesmo nome

  Venda(nome_lanch, nome_refri, preço)

  Apreciador(Nome_cliente, nome_refri)
- Exemplo:

```
SELECT * FROM Apreciador NATURAL JOIN Venda;
```

- Equivale a:

```
SELECT A.nome_cliente, V.*
FROM Apreciador A, Venda V
WHERE A.nome_refri = Venda.nome_refri
```

Junção Teta

- **R JOIN S ON <condição>**
- Exemplo: usando Cliente(nome, endereço) e Frequentador(nome_cliente, nome_lanch):

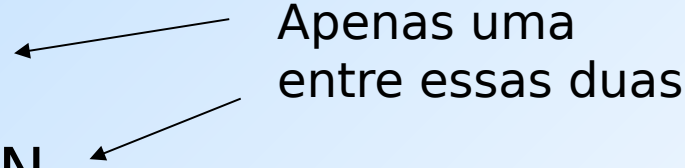
```
SELECT *
FROM Cliente JOIN Frequentador ON
    nome = nome_cliente;
```

nos dá todas quádruplas (c, *e*, *c*, *l*) tais que cliente ***c*** mora no endereço ***e*** e frequenta a lanchonete ***l***.

Junção Externa (Outer Join)

- **R OUTER JOIN S** é o núcleo de uma expressão de junção externa. Ele pode ser modificado por três cláusulas opcionais:

1. **NATURAL** antes de OUTER
2. **ON** <condição> depois de JOIN
3. **LEFT**, **RIGHT**, ou **FULL** antes de OUTER
   - LEFT = inclui apenas as tuplas soltas de R
   - RIGHT = inclui apenas as tuplas soltas de S
   - FULL = inclui as tuplas soltas de ambas



Exemplo:

SELECT * FROM R LEFT OUTER JOIN S ON (B=D AND C=E);

Exemplo: $R \ltimes_{B=D,C=E} S$

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

D	E	F
2	3	10
2	3	11
6	7	12

Relação S

A	B	C	D	E	F
1	2	3	2	3	10
1	2	3	2	3	11
4	5	6	NULL	NULL	NULL
7	8	9	NULL	NULL	NULL

Resultado de $R \ltimes_{B=D,C=E} S$

Exemplo:

SELECT * FROM R RIGHT OUTER JOIN S ON (B=D AND C=E);

Exemplo: $R \rtimes_{B=D,C=E} S$

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

D	E	F
2	3	10
2	3	11
6	7	12

Relação S

A	B	C	D	E	F
1	2	3	2	3	10
1	2	3	2	3	11
NULL	NULL	NULL	6	7	12

Resultado de $R \rtimes_{B=D,C=E} S$

Exemplo:

SELECT * FROM R FULL OUTER JOIN S ON (B=D AND C=E);

Exemplo: $R \bowtie_{B=D,C=E} S$

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

D	E	F
2	3	10
2	3	11
6	7	12

Relação S

A	B	C	D	E	F
1	2	3	2	3	10
1	2	3	2	3	11
4	5	6	NULL	NULL	NULL
7	8	9	NULL	NULL	NULL
NULL	NULL	NULL	6	7	12

Resultado de $R \bowtie_{B=D,C=E} S$

Exemplo:

SELECT * FROM R NATURAL LEFT OUTER JOIN S ;

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

B	C	D
2	3	10
2	3	11
6	7	12

Relação S

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	NULL
7	8	9	NULL

Resultado da junção natural externa à esquerda

Exemplo:

SELECT * FROM R NATURAL RIGHT OUTER JOIN S ;

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

B	C	D
2	3	10
2	3	11
6	7	12

Relação S

A	B	C	D
1	2	3	10
1	2	3	11
NULL	6	7	12

Resultado da junção natural externa à direita

Exemplo:

SELECT * FROM R NATURAL FULL OUTER JOIN S ;

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

B	C	D
2	3	10
2	3	11
6	7	12

Relação S

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	NULL
7	8	9	NULL
NULL	6	7	12

Resultado da junção natural externa completa

Exemplo: junção externa

- Exemplo: usando Cliente(nome, endereço) e Frequentador(nome_cliente, nome_lanch)

```
SELECT *
FROM Cliente LEFT OUTER JOIN Frequentador
                ON nome = nome_cliente;
```

nos dá uma lista de tuplas, onde cada tupla associa os dados de um cliente ao um nome de lanchonete que ele frequenta. Se um cliente não frequenta nenhuma lanchonete, seus dados aparecerão na lista associados ao valor NULL (para o nome de lanchonete).

Exercícios

Aluno(nroAluno, nomeAluno, formação, nivel, idade)

Curso(nome, horario, sala, idProf)

Matriculado(nroAluno, nomeCurso)

Professor(idProf, nomeProf, idDepto)

1) Encontre os nomes de todos os cursos que são ministrados na sala *R128*.
2) Encontre o nome de todos os Juniores (nível = *JR*) que estão matriculados em um curso ministrado por *Ivana Teach*.
3) Encontre os nomes de todos os alunos que estão matriculados em dois cursos que são ministrados no mesmo horário.
4) Encontre os nomes dos alunos matriculados em nenhum curso.
5) Encontre o nome do aluno mais velho que é matriculado em um curso ministrado pelo *Ivana Teach*.
6) Encontre os nomes dos professores que ministram cursos em todas as salas em que algum curso é ministrado.
7) Encontre todos os nomes de pessoas que aparecem no BD.

Exercícios

Aluno(<u>nroAluno</u>, nomeAluno, formação, nivel, idade)

Curso(<u>nome</u>, horario, sala, idProf)

Matriculado(<u>nroAluno, nomeCurso</u>)

Professor(<u>idProf</u>, nomeProf, idDepto)

8) Para cada valor de nível que aparece em Aluno, imprima o nível e idade média dos alunos desse nível.

9) Para cada valor de nível que aparece em Aluno exceto os níveis que possuem menos de 6 alunos, imprima o nível e idade média dos alunos desse nível.

10) Encontre os nomes dos professores para os quais a quantidade de alunos na lista de matriculados de ao menos um dos cursos que eles ministram é maior ou igual a 5.

11) Para cada professor que ministra cursos apenas na sala *R128*, imprima seu nome e o número total de cursos que ele ou ela ministra.

12) Encontre os nomes dos alunos matriculados no número máximo de cursos.

13) Para cada valor de idade que aparece em Aluno, encontre o valor do nível que aparece com mais frequência. Por exemplo, se houver mais alunos no nível *FR* com idade 18 do que os alunos com idade 18 dos níveis *SR*, *JR* ou *SO*, você deve imprimir o par (18,*FR*).

14) Liste todos os pares (*nomeAluno*, *nomeProf*) onde *nomeProf* corresponde ao nome do prof. de um curso no qual o aluno *nomeAluno* está matriculado. Um aluno tem que aparecer na listagem mesmo se não tiver nenhuma matrícula (nesse caso, ele deve aparecer com NULL em *nomeCurso*).

Referências bibliográficas

- *A First Course in Database Systems,*
  Ullman e Widom. 1997.
  Capítulo 5
- *Database Systems – The Complete Book*, Garcia-Molina, Ullman e Widom. 2002.
  Capítulo 6
- *Sistemas de Bancos de Dados* (6ª edição), Elmasri e Navathe. 2010.
  Capítulos 4 e 5