4) Trabalhando com Conjuntos de Dados

Utilizando a União, Intersecção e Subtração de conjuntos de dados

Combinação de consultas

Pode-se combinar os resultados de duas consultas utilizando as operações de conjunto união, interseção e diferença [1] [2] [3] [4] [5] . A sintaxe é

consulta1 UNION [ALL] consulta2
consulta1 INTERSECT [ALL] consulta2
consulta1 EXCEPT [ALL] consulta2

onde consulta1 e consulta2 são consultas que podem utilizar qualquer uma das funcionalidades mostradas até aqui. As operações de conjuntos também podem ser aninhadas ou encadeadas. Por exemplo:

consulta1 UNION consulta2 UNION consulta3

significa, na verdade,
(consulta1 UNION consulta2) UNION consulta3

Efetivamente, UNION anexa o resultado da consulta2 ao resultado da consulta1 (embora não haja garantia que esta seja a ordem que as linhas realmente retornam). Além disso, são eliminadas do resultado as linhas duplicadas, do mesmo modo que no DISTINCT, a não ser que seja utilizado UNION ALL.

INTERSECT retorna todas as linhas presentes tanto no resultado da consulta1 quanto no resultado da consulta2. As linhas duplicadas são eliminadas, a não ser que seja utilizado INTERSECT ALL. EXCEPT retorna todas as linhas presentes no resultado da consulta1, mas que não estão presentes no resultado da consulta2 (às vezes isto é chamado de diferença entre duas consultas). Novamente, as linhas duplicadas são eliminadas a não ser que seja utilizado EXCEPT ALL.

Para ser possível calcular a união, a interseção, ou a diferença entre duas consultas, as duas consultas devem ser "compatíveis para união", significando que ambas devem retornar o mesmo número de colunas, e que as colunas correspondentes devem possuir tipos de dado compatíveis, conforme descrito na Seção 10.5.

Nota: O exemplo abaixo foi escrito pelo tradutor, não fazendo parte do manual original.

Exemplo. Linhas diferentes em duas tabelas com definições idênticas
Este exemplo mostra a utilização de EXCEPT e UNION para descobrir as linhas diferentes de duas tabelas semelhantes.

```
CREATE TEMPORARY TABLE a (c1 text, c2 text, c3 text);
INSERT INTO a VALUES ('x', 'x', 'x');
INSERT INTO a VALUES ('x', 'x', 'y'); -- nas duas tabelas
INSERT INTO a VALUES ('x', 'y', 'x');

CREATE TEMPORARY TABLE b (c1 text, c2 text, c3 text);
INSERT INTO b VALUES ('x', 'x', 'y'); -- nas duas tabelas
INSERT INTO b VALUES ('x', 'x', 'y'); -- nas duas tabelas
INSERT INTO b VALUES ('x', 'y', 'y');
INSERT INTO b VALUES ('y', 'y', 'y');
INSERT INTO b VALUES ('y', 'y', 'y');

-- No comando abaixo só um par ('x', 'x', 'y') é removido do resultado
-- Este comando executa no DB2 8.1 sem alterações.

(SELECT 'a-b' AS dif, a.* FROM a EXCEPT ALL SELECT 'a-b', b.* FROM b)
UNION ALL
(SELECT 'b-a', b.* FROM b EXCEPT ALL SELECT 'b-a', a.* FROM a);

 dif | c1 | c2 | c3
-----+----+----+----
 a-b | x  | x  | x
 a-b | x  | y  | x
 b-a | x  | x  | y
 b-a | x  | y  | y
 b-a | y  | y  | y
 b-a | y  | y  | y
(6 linhas)

-- No comando abaixo são removidas todas as linhas ('x', 'x', 'y'),
-- e só é mostrada uma linha ('y', 'y', 'y') no resultado
-- Este comando executa no DB2 8.1 sem alterações.
-- Este comando executa no Oracle 10g trocando EXCEPT por MINUS.

(SELECT 'a-b' AS dif, a.* FROM a EXCEPT SELECT 'a-b', b.* FROM b)
UNION
(SELECT 'b-a', b.* FROM b EXCEPT SELECT 'b-a', a.* FROM a);

 dif | c1 | c2 | c3
-----+----+----+----
 a-b | x  | x  | x
 a-b | x  | y  | x
 b-a | x  | y  | y
 b-a | y  | y  | y
(4 linhas)
```

Notas

[1] Dados dois conjuntos A e B: chama-se diferença entre A e B o conjunto formado pelos elementos de A que não pertencem a B; chama-se interseção de A com B o conjunto formado

pelos elementos comuns ao conjunto A e ao conjunto B; chama-se união de A com B o conjunto formado pelos elementos que pertencem a A ou B. Edwaldo Bianchini e Herval Paccola - Matemática - Operações com conjuntos. (N. do T.)

[2] SQL Server — UNION combina os resultados de duas ou mais consultas em um único conjunto de resultados que inclui todas as linhas que pertencem à união das consultas. A operação UNION é diferente de utilizar junções que combinam colunas de duas tabelas. As regras básicas para combinar conjuntos de resultados de duas consultas utilizando UNION são as seguintes: a) O número e a ordem das colunas devem ser os mesmos em todas as consultas; b) Os tipos de dado devem ser compatíveis. UNION ALL inclui as linhas duplicadas. SQL Server 2005 Books Online — UNION (Transact-SQL) (N. do T.)

[3] SQL Server — EXCEPT e INTERSECT retornam valores distintos comparando os resultados de duas consultas. EXCEPT retorna todos os valores distintos da consulta à esquerda que não se encontram na consulta à direita. INTERSECT retorna todos os valores distintos retornados pelas consultas à esquerda e a direita do operando INTERSECT. SQL Server 2005 Books Online — EXCEPT and INTERSECT (Transact-SQL) (N. do T.)

[4] Oracle — Os operadores de conjunto combinam os resultados de duas consultas componentes em um único resultado. As consultas que contém operadores de conjunto são chamadas de consultas compostas. Os operadores de conjunto disponíveis são: UNION, UNION ALL, INTERSECT e MINUS (equivalente ao EXCEPT). Oracle® Database SQL Reference 10g Release 1 (10.1) Part Number B10759-01. (N. do T.)

[5] DB2 — Os operadores de conjunto UNION, EXCEPT e INTERSECT correspondem aos operadores relacionais união, diferença e interseção. DB2 Version 9 for Linux, UNIX, and Windows (N. do T.)

Detalhes em: http://pgdocptbr.sourceforge.net/pg80/queries-union.html

Artigo do Juliano : http://imasters.uol.com.br/artigo/954:

POSTGRESQL Interagindo com banco de dados

Agora que você já tem o banco de dados PostgreSQL instalado e rodando, e já se identificou com alguma ferramenta para manipulação das bases de dados, vamos começar a interagir com o banco de dados. A intenção não é ensinar SQL, mas sim, mostrar como verificar no PostgreSQL determinadas funcionalidades existentes em outros bancos de dados, bem como algumas de suas particularidades.
Primeiro, criaremos 3 tabelas:

```
CREATE TABLE cliente (
cliente_id SERIAL NOT NULL,
desde      DATE NULL,
nome       VARCHAR(60) NULL,
CONSTRAINT XPKcliente
   PRIMARY KEY (cliente_id)
);
```

```
CREATE TABLE venda (
venda_id   SERIAL NOT NULL,
cliente_id INT4 NOT NULL,
data       DATE NULL,
valor      NUMERIC(15,2) NULL,
produto    VARCHAR(30) NULL,
CONSTRAINT XPKvenda
   PRIMARY KEY (venda_id),
CONSTRAINT cliente_vendas
   FOREIGN KEY (cliente_id)
      REFERENCES cliente
);

CREATE INDEX XIF1venda ON venda
(
cliente_id
);

CREATE TABLE troca (
troca_id   SERIAL NOT NULL,
cliente_id INT4 NOT NULL,
data       DATE NULL,
produto    VARCHAR(30) NULL,
troca      VARCHAR(30) NULL,
CONSTRAINT XPKtroca
   PRIMARY KEY (troca_id),
CONSTRAINT cliente_trocas
   FOREIGN KEY (cliente_id)
      REFERENCES cliente
);

CREATE INDEX XIF1troca ON troca
(
cliente_id
);
```

Em seguida, iremos popular as tabelas com alguns dados:

```
INSERT INTO cliente (desde,nome)
      VALUES ('2002-01-12','Paulo Santos Macedo');
INSERT INTO cliente (desde,nome)
      VALUES ('2001-07-21','Márcia Barbosa');
INSERT INTO cliente (desde,nome)
      VALUES ('2000-02-27','Anderson Marques');
INSERT INTO cliente (desde,nome)
      VALUES ('2003-01-12','Daniela Freitas');
```

```
INSERT INTO cliente (desde,nome)
      VALUES ('2003-01-15','Ana Júlia Cabral');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (1,'2002-12-23',16,'Relógio');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (3,'2002-12-23',110,'Mala Viagem');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (1,'2002-12-21',10,'Saca-rolha');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (4,'2002-12-20',32,'Fichário');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (2,'2002-12-23',28,'Despertador');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (3,'2002-12-23',43,'Mochila');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (2,'2002-12-21',22,'Rádio');
INSERT INTO venda (cliente_id, data, valor, produto)
      VALUES (4,'2002-12-20',12,'Lapiseira');
INSERT INTO troca (cliente_id, data, produto, troca)
      VALUES (1,'2003-02-12','Relógio','Relógio');
INSERT INTO troca (cliente_id, data, produto, troca)
            VALUES (3,'2003-02-13','Mala Viagem', 'Maleta Executivo');
INSERT INTO troca (cliente_id, data, produto, troca)
      VALUES (1,'2003-02-08','Saca-rolha','Garrafa Térmica');
INSERT INTO troca (cliente_id, data, produto, troca)
      VALUES (4,'2003-02-09','Fichário','Fichário');
```

Agora sim, vamos começar.

COMBINANDO CONSULTAS

Um problema encontrado quando escrevemos consultas em SQL é que, em determinados casos, estas consultas devem ser combinadas para obter o resultado desejado, pois, através de uma consulta única e direta, talvez não seja possível obtê-los. Combinar consultas significa que mais de uma instrução SELECT estará sendo usada na consulta. O resultado desta combinação se dará através das seguintes palavras-chave:

UNION	utilizada para adicionar (unir) os resultados das instruções SELECT apresentadas na consulta
INTERSECT	retorna somente os dados comuns resultantes das instruções SELECT apresentadas na consulta
EXCEPT	mostra todos os dados que não estão incluídos na segunda instrução SELECT apresentada na consulta

Vamos aos exemplos:

Pelos dados iniciais de exemplo, vimos que existem 4 clientes que adquiriram produtos para o natal do ano passado e, alguns deles, tiveram que fazer a troca de alguns produtos por um motivo qualquer.

Queremos saber então, quais clientes NÃO precisaram fazer nenhuma troca:

```
SELECT cliente.nome FROM venda JOIN cliente ON cliente.cliente_id = venda.cliente_id
EXCEPT
SELECT cliente.nome FROM troca JOIN cliente ON cliente.cliente_id = troca.cliente_id;
troca.cliente_id;
nome
-------------------------
Márcia Barbosa
(1 row)
```

...e quais PRECISARAM fazer alguma troca:

```
SELECT cliente.nome FROM venda JOIN cliente ON cliente.cliente_id = venda.cliente_id
INTERSECT
SELECT cliente.nome FROM troca JOIN cliente ON cliente.cliente_id = troca.cliente_id;
nome
-------------------------
Anderson Marques
Daniela Freitas
Paulo Santos Macedo
(3 rows)
```

Agora, queremos saber quais produtos foram movimentados, ou seja, tanto faz se foram vendidos ou trocados:

```
SELECT venda.produto FROM venda
UNION
SELECT troca.troca FROM troca;
produto
-------------------------
Despertador
Fichário
Garrafa Térmica
Lapiseira
Mala Viagem
Maleta Executivo
Mochila
Rádio
Relógio
Saca-rolha
```

(10 rows)

Observações:
A arquitetura da base de dados influi diretamente sobre como serão criadas as consultas, ou seja, quais tabelas contém certos dados e, qual o relacionamento entre eles;

Obviamente, existem muitas formas de obter o mesmo resultado, as maneiras apresentadas aqui, são algumas delas;

Podemos perceber que para a execução correta das combinações, usamos somente colunas semelhantes de cada SELECT;

Artigo continuando o anterior, do Juliano Ignácio no iMasters - http://imasters.uol.com.br/artigo/966/postgresql/union_ou_union_all/

UNION ou UNION ALL

Na coluna anterior vimos o uso de UNION, no entanto, nem sempre desejamos o comportamento padrão apresentado. Quando o UNION é executado, os dados referentes às SELECTs envolvidas são ordenados, eliminando a duplicação de registros. Porém, se você deseja unir as consultas de forma a aparecer TODOS os registros, use UNION ALL.

Tomando como exemplo as tabelas criadas na coluna anterior

SELECT venda.produto FROM venda UNION SELECT troca.troca FROM troca;	SELECT venda.produto FROM venda UNION ALL SELECT troca.troca FROM troca;
produto ---------- Despertador Fichário Garrafa Térmica Lapiseira Mala Viagem Maleta Executivo Mochila Relógio Rádio Saca-rolha (10 rows)	produto ---------- Relógio Mala Viagem Saca-rolha Fichário Despertador Mochila Rádio Lapiseira Relógio Maleta Executivo Garrafa Térmica Fichário (12 rows)

SUPRIMINDO AS DUPLICIDADES

Para que valores duplicados não sejam mostrados no resultado de uma consulta (sem usar UNION,

como vimos anteriormente), usamos DISTINCT. Veja que, o valor duplicado se refere ao registro como um todo, e não somente a uma coluna. Para que possamos entender, vamos inserir algumas linhas em nossa tabela de vendas criada anteriormente

```
INSERT INTO venda (cliente_id, data, valor, produto)
VALUES (3,'2003-02-12',16,'Relógio');
INSERT INTO venda (cliente_id, data, valor, produto)
VALUES (1,'2003-02-15',110,'Mala Viagem');
INSERT INTO venda (cliente_id, data, valor, produto)
VALUES (3,'2003-02-16',10,'Saca-rolha');
INSERT INTO venda (cliente_id, data, valor, produto)
VALUES (1,'2003-02-20',32,'Fichário');
```

Após inserir as vendas acima, gostaria de saber quais são os produtos que estamos vendendo desde o início dos lançamentos, como iremos observar, a primeira consulta irá mostrar produtos repetidos, o que não é necessário e, às vezes, atrapalha. Na segunda consulta, a cláusula DISTINCT esconde a duplicidade dos registros e coloca-os na ordem da primeira coluna (neste caso só temos uma mesmo).

SELECT produto FROM venda;	SELECT DISTINCT produto FROM venda;
produto ---------- Relógio Mala Viagem Saca-rolha Fichário Despertador Mochila Rádio Lapiseira Relógio Mala Viagem Saca-rolha Fichário (12 rows)	produto ---------- Despertador Fichário Lapiseira Mala Viagem Mochila Relógio Rádio Saca-rolha (8 rows)

VARIÁVEIS MÁGICAS

O PostgreSQL possui 4 variáveis 'mágicas' que guardam informações sobre o usuário corrente e data e hora atuais, facilitando, talvez, a implementação de rotinas de auditoria.

CURRENT_DATE CURRENT_TIME	SELECT CURRENT_DATE;

CURRENT_TIMESTAMP CURRENT_USER	date ---------- 2003-02-24 (1 row)
Para saber a quanto tempo foram vendidos os produtos, podemos executar a seguinte consulta:	SELECT produto, (CURRENT_DATE - data) AS dias FROM venda ORDER BY produto; produto | dias ----------+------ Despertador | 63 Fichário | 4 Fichário | 66 Lapiseira | 66 Mala Viagem | 63 Mala Viagem | 9 Mochila | 63 Relógio | 63 Relógio | 12 Rádio | 65 Saca-rolha | 8 Saca-rolha | 65 (12 rows)

Veja o script aula4_TeoriaDosConjutos.sql com mais exercícios.

Tutorial online: http://www.w3schools.com/sql/sql_union.asp