

O'REILLY®

Second  
Edition

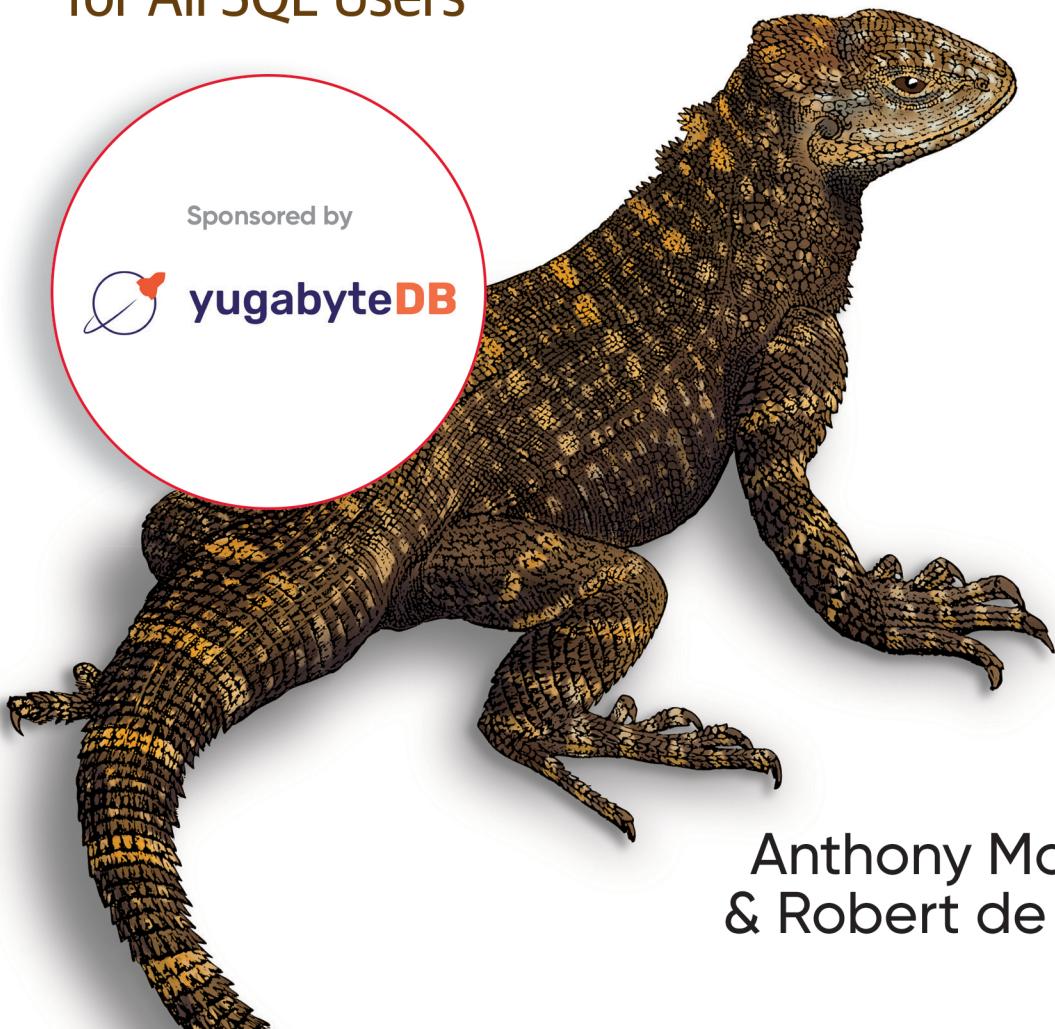
# SQL Cookbook

Query Solutions and Techniques  
for All SQL Users

Sponsored by



yugabyteDB



Anthony Molinaro  
& Robert de Graaf



# SQL Cookbook

You may know SQL basics, but are you taking advantage of its expressive power? This second edition applies a highly practical approach to Structured Query Language (SQL) so you can create and manipulate large stores of data. Based on real-world examples, this updated cookbook provides a framework to help you construct solutions and executable examples in several flavors of SQL, including Oracle, DB2, SQL Server, MySQL, and PostgreSQL.

SQL programmers, analysts, data scientists, database administrators—and even relatively casual SQL users—will find *SQL Cookbook* to be a valuable problem-solving guide for everyday issues. No other resource offers recipes in this unique format to help you tackle nagging day-to-day conundrums with SQL.

## The second edition includes:

- Fully revised recipes that recognize the greater adoption of window functions in SQL implementations
- Additional recipes that reflect the widespread adoption of common table expressions (CTEs) for more readable, easier-to-implement solutions
- New recipes to make SQL more useful for people who aren't database experts, including data scientists
- Expanded solutions for working with numbers and strings
- Up-to-date SQL recipes throughout the book to guide you through the basics

"It's great to see a SQL cookbook updated for modern SQL topics, including window functions, common table expressions, and recursive hierarchical queries."

—Thomas Nield  
Author of *Getting Started with SQL* (O'Reilly)

"Anthony and Robert bring a level of excitement I have yet to see in any book covering SQL. Their delivery of effective and efficient solutions is well paced, and it reinforces and complements earlier lessons learned."

—Scott Haines  
Senior Principal Software Engineer, Twilio

Anthony Molinaro is a data scientist at Johnson & Johnson. His primary areas of research include nonparametric methods, time series analysis, and large-scale database characterization and transformation.

Robert de Graaf is senior data scientist at RightShip.



SECOND EDITION

---

# SQL Cookbook

*Query Solutions and Techniques  
for All SQL Users*

*Anthony Molinaro and Robert de Graaf*

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

## **SQL Cookbook**

by Anthony Molinaro and Robert de Graaf

Copyright © 2021 Robert de Graaf. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Acquisitions Editor:** Jessica Haberman

**Indexer:** WordCo Indexing Services, Inc.

**Development Editor:** Virginia Wilson

**Interior Designer:** David Futato

**Production Editor:** Kate Galloway

**Cover Designer:** Karen Montgomery

**Copyeditor:** Kim Wimpsett

**Illustrator:** O'Reilly Media

**Proofreader:** nSight, Inc.

December 2005: First Edition

December 2020: Second Edition

### **Revision History for the Second Edition**

2020-11-03: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781492077442> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *SQL Cookbook*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors, and do not represent the publisher's views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Yugabyte. See our [statement of editorial independence](#).

978-1-098-10014-8

[LSI]

# Prefácio

SQL é a língua franca do profissional de dados. Ao mesmo tempo, nem sempre recebe a atenção que merece em comparação com a ferramenta quente do dia. Como resultado, é comum encontrar pessoas que usam SQL com frequência, mas raramente ou nunca vão além das consultas mais simples, muitas vezes porque acreditam que isso é tudo que existe.

Este livro mostra o quanto o SQL pode fazer, expandindo as caixas de ferramentas dos usuários. Ao final do livro você terá visto como o SQL pode ser usado para análise estatística; fazer relatórios de maneira semelhante às ferramentas de Business Intelligence; para combinar dados de texto; realizar análises sofisticadas de dados de data; e muito mais.

A primeira edição do *SQL Cookbook* tem sido uma escolha popular como o “segundo livro sobre SQL” – o livro que as pessoas leem depois de aprenderem o básico – desde seu lançamento original. Possui muitos pontos fortes, como sua ampla gama de temas e seu estilo amigável.

No entanto, sabe-se que a computação avança rapidamente, mesmo quando se trata de algo tão maduro como o SQL, que tem raízes que remontam à década de 1970. Embora esta nova edição não cubra novos recursos de linguagem, uma mudança importante é que os recursos que eram novos na época da primeira edição, e encontrados em algumas implementações e não em outras, agora estão estabilizados e padronizados. Como resultado, temos muito mais espaço para desenvolver soluções padrão do que era possível anteriormente.

Existem dois exemplos principais que é importante destacar. Expressões de tabela comuns (CTEs), incluindo CTEs recursivas, estavam disponíveis em algumas implementações no momento em que a primeira edição foi lançada, mas agora estão disponíveis em todas as cinco. Eles foram introduzidos para resolver algumas limitações práticas do SQL, algumas das quais podem ser vistas diretamente nestas receitas. Um novo apêndice sobre CTEs recursivos nesta edição sublinha a sua importância e explica a sua relevância.

As funções de janela também eram novas o suficiente na época do lançamento da primeira edição e não estavam disponíveis em todas as implementações. Eles também eram novos o suficiente para que um apêndice especial fosse escrito para explicá-los, o que permanece. Agora, entretanto, as funções de janela estão em todas as implementações deste livro.

Eles também estão em todas as outras implementações SQL que conhecemos, embora existam tantos bancos de dados por aí, é impossível garantir que não haja nenhum que negligencie funções de janela e/ou CTEs.

Além de padronizar as consultas sempre que possível, trouxemos novo material para os Capítulos 6 e 7. O material em [Capítulo 7](#) desbloqueia novas aplicações de análise de dados em receitas sobre o desvio absoluto mediano e a lei de Benford. Em [Capítulo 6](#), temos uma nova receita para ajudar a combinar os dados pelo som do texto e movemos o material sobre expressões regulares para [Capítulo 6](#) de [Capítulo 14](#).

## Para quem é este livro

Este livro destina-se a qualquer usuário de SQL que queira aprofundar suas dúvidas. Em termos de habilidade, é destinado a alguém que conhece pelo menos um pouco de SQL - você deve ter lido o livro de Alan Beaulieu *Learning SQL*, por exemplo – e idealmente você teve que escrever consultas em dados disponíveis para responder a um problema da vida real.

Além desses parâmetros vagos, este é um livro para todos os usuários de SQL, incluindo engenheiros de dados, cientistas de dados, pessoal de visualização de dados, pessoal de BI, etc. Alguns desses usuários podem nunca ou raramente acessar bancos de dados diretamente, mas usam sua visualização de dados, BI ou ferramenta estatística para consultar e buscar dados. A ênfase está em questões práticas que podem resolver problemas do mundo real. Onde aparece uma pequena quantidade de teoria, ela existe para apoiar diretamente os elementos práticos.

## O que está faltando neste livro

Este é um livro prático, principalmente sobre o uso de SQL para compreender dados. Ele não cobre aspectos teóricos de bancos de dados, design de banco de dados ou a teoria por trás do SQL, exceto quando necessário para explicar receitas ou técnicas específicas.

Também não cobre extensões de bancos de dados para lidar com tipos de dados como XML e JSON. Existem outros recursos disponíveis para esses tópicos especializados.

## Plataforma e Versão

SQL é um alvo móvel. Os fornecedores estão constantemente injetando novos recursos e funcionalidades em seus produtos. Assim, você deve saber antecipadamente quais versões das diversas plataformas foram utilizadas na elaboração deste texto:

- DB2 11.5
- Oracle Database 19c
- PostgreSQL 12

- SQL Server 2017
- MySQL 8.0

## Tabelas usadas neste livro

A maioria dos exemplos neste livro envolve o uso de duas tabelas, EMP e DEPT. A tabela EMP é uma tabela simples de 14 linhas com apenas campos numéricos, de string e de data. A tabela DEPT é uma tabela simples de quatro linhas com apenas campos numéricos e de string. Essas tabelas aparecem em muitos textos antigos de bancos de dados, e o relacionamento muitos para um entre departamentos e funcionários é bem compreendido.

Todas as soluções deste livro, exceto algumas poucas, vão contra essas tabelas. Em nenhum lugar ajustamos os dados de exemplo para configurar uma solução que você provavelmente não teria chance de implementar no mundo real, como fazem alguns livros.

O conteúdo do EMP e do DEPT é mostrado aqui, respectivamente:

```
select * from emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-2005	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-2006	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30
7566	JONES	MANAGER	7839	02-APR-2006	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-2006	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-2006	2850		30
7782	CLARK	MANAGER	7839	09-JUN-2006	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-2007	3000		20
7839	KING	PRESIDENT		17-NOV-2006	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-2006	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-2008	1100		20
7900	JAMES	CLERK	7698	03-DEC-2006	950		30
7902	FORD	ANALYST	7566	03-DEC-2006	3000		20
7934	MILLER	CLERK	7782	23-JAN-2007	1300		10

```
select * from dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Além disso, você encontrará quatro tabelas dinâmicas usadas neste livro: T1, T10, T100 e T500. Como essas tabelas existem apenas para facilitar os pivôs, não lhes demos nomes inteligentes. O número após o “T” em cada uma das tabelas dinâmicas significa o número de linhas em cada tabela, começando em 1. Por exemplo, aqui estão os valores para T1 e T10:

```
select id from t1;
```

ID
1

```
select id from t10;
```

ID
1
2
3
4
5
6
7
8
9
10

As tabelas dinâmicas são um atalho útil quando precisamos criar uma série de linhas para facilitar uma consulta.

Além disso, alguns fornecedores permitem instruções parciais SELECT. Por exemplo, você pode ter SELECT sem uma cláusula FROM. Às vezes neste livro usaremos uma tabela de suporte, T1, com uma única linha, em vez de usar consultas parciais para maior clareza. O uso é semelhante à tabela DUAL da Oracle, mas ao usar a tabela T1, fazemos a mesma coisa de maneira padronizada em todas as implementações que estamos analisando.

Quaisquer outras tabelas são específicas para receitas e capítulos específicos e serão introduzidas no texto quando apropriado.

## Convenções utilizadas neste livro

Usamos uma série de convenções tipográficas e de codificação neste livro. Reserve um tempo para se familiarizar com eles. Isso melhorará sua compreensão do texto. As convenções de codificação em particular são importantes porque não podemos repeti-las para cada receita do livro. Em vez disso, listamos aqui as convenções importantes.

## Convenções Tipográficas

As seguintes convenções tipográficas são usadas neste livro:

### *MAIÚSCULAS*

Usado para indicar palavras-chave SQL no texto.

### *minúscula*

Usado para todas as consultas em exemplos de código. Outras linguagens, como C e Java, usam letras minúsculas para a maioria das palavras-chave, e achamos que são muito mais legíveis do que letras maiúsculas. Assim, todas as consultas estarão em letras minúsculas.

### **Largura constante em negrito**

Indica a entrada do usuário em exemplos que mostram uma interação.



Indica uma dica, sugestão ou nota geral.



Indica um aviso ou cuidado.

## Convenções de codificação

Nossa preferência por maiúsculas e minúsculas em instruções SQL é sempre usar letras minúsculas, tanto para palavras-chave quanto para identificadores especificados pelo usuário. Por exemplo:

```
select empno, ename
      from emp;
```

Sua preferência pode ser outra. Por exemplo, muitos preferem palavras-chave SQL em maiúsculas. Use o estilo de codificação de sua preferência ou o que seu projeto exigir.

Apesar do uso de letras minúsculas em exemplos de código, usamos consistentemente letras maiúsculas para palavras-chave e identificadores SQL no texto. Fazemos isso para que esses itens se destaquem como algo diferente da prosa normal. Por exemplo:

A consulta anterior representa um SELECT na tabela EMP.

Embora este livro cubra bancos de dados de cinco fornecedores diferentes, decidimos usar um formato para todos os resultados:

```
EMPNO ENAME
-----
7369 SMITH
7499 ALLEN
...
```

Muitas soluções utilizam *visualizações embutidas* ou subconsultas na cláusula FROM. O padrão ANSI SQL exige que tais visualizações recebam aliases de tabela. (A Oracle é o único fornecedor que permite que você evite especificar esses aliases.) Assim, nossas soluções usam aliases como X e Y para identificar os conjuntos de resultados das visualizações in-line:

```
select job, sal
from (select job, max(sal) sal
       from emp
      group by job)x;
```

Observe a letra X após o parêntese final de fechamento. Essa letra X passa a ser o nome da “tabela” retornada pela subconsulta na cláusula FROM. Embora os aliases de coluna sejam uma ferramenta valiosa para escrever código auto documentado, os aliases em visualizações in-line (para a maioria das receitas deste livro) são simplesmente formalidades. Eles normalmente recebem nomes triviais, como X, Y, Z, TMP1 e TMP2. Nos casos em que um alias melhor possa fornecer mais compreensão, nós os utilizamos.

Você notará que o SQL na seção “Solução” das receitas normalmente é numerado, por exemplo:

```
1 select ename
2   from emp
3  where deptno = 10
```

O número não faz parte da sintaxe; serve apenas para fazer referência a partes da consulta por número na seção “Discussão”.

## Aprendizagem on-line O'Reilly

**O'REILLY**® Há mais de 40 anos, *O'Reilly Media* forneceu treinamento, conhecimento e insights em tecnologia e negócios para ajudar as empresas a ter sucesso.

Nossa rede exclusiva de especialistas e inovadores compartilha seu conhecimento e experiência por meio de livros, artigos e nossa plataforma de aprendizagem on-line. A plataforma de aprendizagem on-line da O'Reilly oferece acesso sob demanda a cursos de treinamento ao vivo, caminhos de aprendizagem aprofundados, ambientes de codificação interativos e uma vasta coleção de textos e vídeos da O'Reilly e de mais de 200 outros editores. Para mais informações visite <http://oreilly.com>.

## Como entrar em contato conosco

Por favor, envie comentários e perguntas sobre este livro à editora:

O'Reilly Media, Inc.  
Rodovia 1005 Gravenstein Norte  
Sebastopol, CA 95472  
800-998-9938 (nos Estados Unidos ou Canadá)  
707-829-0515 (internacional ou local)  
707-829-0104 (fax)

Temos uma página web para este livro, onde listamos erratas, exemplos e qualquer informação adicional. Você pode acessar esta página em <https://oreil.ly/sql-ckbk-2e>.

E-mail [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com) para comentar ou fazer perguntas técnicas sobre este livro.

Para novidades e informações sobre nossos livros e cursos, acesse <http://oreilly.com>.

Encontre-nos no Facebook: <http://facebook.com/oreilly>

Siga-nos no Twitter: <http://twitter.com/oreillymedia>

Assista-nos no YouTube: <http://www.youtube.com/oreillymedia>

## Agradecimentos da segunda edição

Muitas pessoas excelentes ajudaram nesta segunda edição. Obrigado a Jess Haberman, Virginia Wilson, Kate Galloway e Gary O'Brien da O'Reilly. Obrigado a Nicholas Adams por salvar o dia repetidamente no Atlas. Muito obrigado aos revisores técnicos: Alan Beaulieu, Scott Haines e Thomas Nield.

Por fim, muito obrigado à minha família — Clare, Maya e Leda — por gentilmente suportar a perda de mim para outro livro por um tempo.

—Robert de Graaf

## Agradecimentos da primeira edição

Este livro não existiria sem todo o apoio que recebemos de muitas pessoas. Gostaria de agradecer à minha mãe, Connie, a quem este livro é dedicado. Sem o seu trabalho árduo e sacrifício, eu não estaria onde estou hoje. Obrigado por tudo, mãe. Estou grato e agradecido por tudo que você fez por meu irmão e por mim. Fui abençoada por ter você como minha mãe.

Para meu irmão, Joe: Toda vez que eu voltava de Baltimore para fazer uma pausa na escrita, você estava lá para me lembrar como as coisas são ótimas quando não estamos trabalhando e como devo terminar de escrever para poder voltar ao trabalho. Coisas mais importantes na vida. Você é um bom homem e eu o respeito. Estou extremamente orgulhoso de você e orgulhoso de chamá-lo de meu irmão.

À minha maravilhosa noiva, Georgia: sem o seu apoio eu não teria conseguido ler todas as mais de 600 páginas deste livro. Você esteve aqui compartilhando essa experiência comigo, dia após dia. Eu sei que foi tão difícil para você quanto foi para mim. Passei o dia todo trabalhando e a noite toda escrevendo, mas você foi ótimo em tudo isso. Você foi compreensivo e solidário, e serei eternamente grato. Obrigado. Eu te amo.

Aos meus futuros sogros: À minha sogra e ao meu sogro, Kiki e George, obrigado pelo apoio durante toda esta experiência. Você sempre me fez sentir em casa sempre que eu fazia uma pausa e vinha me visitar, e garantiu que Georgia e eu estivéssemos sempre bem alimentados. Para minhas cunhadas, Anna e Kathy, sempre foi divertido voltar para casa e sair com vocês, proporcionando a Georgia e a mim uma pausa muito necessária do livro e de Baltimore.

Ao meu editor, Jonathan Gennick, sem o qual este livro não existiria: Jonathan, você merece muito crédito por este livro. Você foi além do que um editor normalmente faria e por isso merece muito agradecimento. Desde fornecer receitas a toneladas de reescritas até manter as coisas bem-humoradas apesar dos prazos próximos, eu não teria conseguido sem você. Sou grato por tê-lo como meu editor e grato pela oportunidade que você me deu. Sendo um DBA experiente e autor, foi um prazer trabalhar com alguém do seu nível técnico e especialização. Não consigo imaginar que existam muitos editores por aí que possam, se assim o decidirem, parar de editar e trabalhar praticamente em qualquer lugar como administrador de banco de dados (DBA); Jônatas pode. Ser um DBA certamente lhe dá uma vantagem como editor, pois você geralmente sabe o que quero dizer, mesmo quando estou tendo problemas para expressá-lo. O'Reilly tem sorte de ter você na equipe e eu tenho sorte de ter você como editor.

Gostaria de agradecer a Ales Spetic e Jonathan Gennick por *Livro de receitas do Transact-SQL*. Isaac Newton disse a famosa frase: “Se vi um pouco mais longe, foi por estar sobre os ombros de gigantes”. Na seção de agradecimentos do *Livro de receitas do Transact-SQL*, Ales Spetic escreveu algo que é uma prova dessa famosa citação e acho que deveria estar em todos os livros de SQL. Incluo aqui suas palavras:

Espero que este livro complemente as obras existentes de autores notáveis como Joe Celko, David Rozenshtein, Anatoly Abramovich, Eugine Berger, Iztik Ben-Gan, Richard Snodgrass e outros. Passei muitas noites estudando seu trabalho e aprendi quase tudo que sei em seus livros. Enquanto escrevo estas linhas, estou ciente de que para cada noite que passei descobrindo seus segredos, eles devem ter passado 10 noites colocando seu conhecimento de uma forma consistente e legível. É uma honra poder retribuir algo à comunidade SQL.

Gostaria de agradecer a Sanjay Mishra por seu excelente livro *Mastering Oracle SQL*, e também por me colocar em contato com Jonathan. Se não fosse por Sanjay, talvez eu nunca tivesse conhecido Jonathan e nunca teria escrito este livro. É incrível como um simples e-mail pode mudar sua vida. Gostaria de agradecer especialmente a David Rozenshtein pelo seu livro *Essence of SQL*, que me proporcionou uma compreensão sólida de como pensar e resolver problemas em conjuntos/SQL. Gostaria de agradecer a David Rozenshtein, Anatoly Abramovich e Eugene Birger pelo livro *Optimizing Transact-SQL*, com o qual aprendi muitas das técnicas SQL avançadas que uso hoje.

Gostaria de agradecer a toda equipe da Wireless Generation, uma grande empresa com ótimas pessoas. Um grande obrigado a todas as pessoas que dedicaram seu tempo para revisar, criticar ou oferecer conselhos para me ajudar a concluir este livro: Jesse Davis, Joel Patterson, Philip Zee, Kevin Marshall, Doug Daniels, Otis Gospodnetic, Ken Gunn, John Stewart, Jim Abramson, Adam Mayer, Susan Lau, Alexis Le-Quoc e Paul Feuer. Gostaria de agradecer a Maggie Ho pela revisão cuidadosa do meu trabalho e pelos comentários extremamente úteis sobre a atualização da função da janela. Gostaria de agradecer a Chuck Van Buren e Gillian Gutenberg pelos seus excelentes conselhos sobre corrida. Os treinos matinais me ajudaram a limpar minha mente e relaxar. Acho que não teria conseguido terminar este livro sem sair um pouco. Gostaria de agradecer a Steve Kang e Chad Levinson por aguentarem todas as minhas conversas incessantes sobre diferentes técnicas de SQL nas noites em que tudo o que queriam era ir à Union Square para tomar uma cerveja e um hambúrguer na Heartland Brewery depois de um longo dia de trabalho. trabalhar. Gostaria de agradecer a Aaron Boyd por todo o seu apoio, palavras gentis e, o mais importante, bons conselhos. Aaron é honesto, trabalhador e um cara muito direto; pessoas como ele tornam uma empresa melhor. Gostaria de agradecer a Olivier Pomel por seu apoio e ajuda na redação deste livro, em particular pela solução DB2 para criação de listas delimitadas a partir de linhas. Olivier contribuiu com essa solução mesmo sem ter um sistema DB2 para testá-la! Expliquei a ele como funcionava a cláusula WITH e, minutos depois, ele apresentou a solução que você vê neste livro.

Jonah Harris e David Rozenshtein também forneceram comentários úteis sobre a revisão técnica do manuscrito. E Arun Marathe, Nuno Pinto do Souto e Andrew Odewahn opinaram no esboço e na escolha das receitas enquanto este livro estava em fase de formação. Muito obrigado a todos vocês.

Quero agradecer a John Haydu e à equipe de desenvolvimento de cláusula MODEL da Oracle Corporation por dedicarem seu tempo para revisar o artigo de cláusula MODEL que escrevi para O'Reilly e, em última análise, me dar uma melhor compreensão de como essa cláusula funciona. Gostaria de agradecer a Tom Kyte, da Oracle Corporation, por me permitir adaptar sua função TO\_BASE em uma solução somente SQL. Bruno Denuit da Microsoft respondeu a perguntas que eu tinha sobre a funcionalidade das funções de janela introduzidas no SQL Server 2005. Simon Riggs do PostgreSQL me manteve atualizado sobre os novos recursos SQL no PostgreSQL (muito obrigado: Simon, por saber o que estava por vir fora e quando,

Consegui incorporar alguns novos recursos SQL, como a função GENERATE\_SERIES, que acho que proporcionou soluções mais elegantes em comparação com tabelas dinâmicas).

Por último, mas não menos importante, gostaria de agradecer a Kay Young. Quando você é talentoso e apaixonado pelo que faz, é ótimo poder trabalhar com pessoas que também são talentosas e apaixonadas. Muitas das receitas que você vê neste texto vieram do trabalho com Kay e da criação de soluções SQL para problemas cotidianos na Wireless Generation. Quero agradecer e dizer que aprecio absolutamente toda a ajuda que você me deu durante tudo isso; desde conselhos até correções gramaticais e códigos, você desempenhou um papel fundamental na redação deste livro. Tem sido ótimo trabalhar com você, e a Wireless Generation é uma empresa melhor porque você está lá.

—Anthony Molinaro

# CAPÍTULO 1

## Recuperando Registros

Este capítulo enfoca as instruções básicas SELECT. É importante ter uma compreensão sólida do básico, pois muitos dos tópicos abordados aqui não estão presentes apenas em receitas mais difíceis, mas também são encontrados no SQL cotidiano.

### 1.1 Recuperando todas as linhas e colunas de uma tabela

#### Problema

Você tem uma tabela e deseja ver todos os dados nela.

#### Solução

Use o caractere especial \* e emita um SELECT na tabela:

```
1 select *
2   from emp
```

#### Discussão

O caractere \* tem um significado especial no SQL. Usá-lo retornará todas as colunas da tabela especificada. Como não há nenhuma cláusula WHERE especificada, todas as linhas também serão retornadas. A alternativa seria listar cada coluna individualmente:

```
select empno,ename,job,sal,mgr,hiredate,comm,deptno
      from emp
```

Em consultas ad hoc executadas interativamente, é mais fácil usar SELECT \*. No entanto, ao escrever o código do programa, é melhor especificar cada coluna individualmente. O desempenho será o mesmo, mas sendo explícito você sempre saberá quais colunas está retornando da consulta. Da mesma forma, essas consultas são mais fáceis de entender por

outras pessoas além de você (que podem ou não conhecer todas as colunas nas tabelas da consulta). Problemas com SELECT \* também podem surgir se sua consulta estiver dentro do código e o programa obtiver um conjunto de colunas da consulta diferente do esperado. Pelo menos, se você especificar todas as colunas e uma ou mais estiver ausente, é mais provável que qualquer erro gerado seja rastreável à(s) coluna(s) ausente(s) específica(s).

## 1.2 Recuperando um subconjunto de linhas de uma tabela

### Problema

Você tem uma tabela e deseja ver apenas as linhas que atendem a uma condição específica.

### Solução

Use a cláusula WHERE para especificar quais linhas manter. Por exemplo, para visualizar todos os funcionários atribuídos ao departamento número 10:

```
1 select *
2   from emp
3 where deptno = 10
```

### Discussão

A cláusula WHERE permite recuperar apenas as linhas nas quais você está interessado. Se a expressão na cláusula WHERE for verdadeira para qualquer linha, essa linha será retornada.

A maioria dos fornecedores oferece suporte a operadores comuns, como =, <, >, <=, >=, ! e <>. Além disso, você pode querer linhas que satisfaçam várias condições; isso pode ser feito especificando AND, OR e parênteses, conforme mostrado na próxima receita.

## 1.3 Encontrando Linhas que Satisfazem Múltiplas Condições

### Problema

Você deseja retornar linhas que satisfaçam várias condições.

### Solução

Use a cláusula WHERE juntamente com as cláusulas OR e AND. Por exemplo, se você quiser encontrar todos os funcionários do departamento 10, junto com todos os funcionários que ganham comissão, junto com todos os funcionários do departamento 20 que ganham no máximo \$ 2.000:

```

1 select *
2   from emp
3  where deptno = 10
4    or comm is not null
5    or sal <= 2000 and deptno=20

```

## Discussão

Você pode usar uma combinação de AND, OR e parênteses para retornar linhas que satisfaçam várias condições. No exemplo de solução, a cláusula WHERE localiza linhas de modo que:

- O DEPTNO é 10
- O COMM não é NULL
- O salário é de \$ 2.000 ou menos para qualquer funcionário do DEPTNO 20.

A presença de parênteses faz com que as condições dentro deles sejam avaliadas em conjunto.

Por exemplo, considere como o conjunto de resultados muda se a consulta for escrita entre parênteses, conforme mostrado aqui:

```

select *
  from emp
 where (
      depto = 10
      or comm is not null
      or sal <= 2000
    )
  and deptno=20

EMPNO ENAME   JOB      MGR HIREDATE      SAL       COMM  DEPTNO
----- -----
 7369 SMITH    CLERK   7902 17-DEC-1980    800        20
 7876 ADAMS    CLERK   7788 12-JAN-1983   1100        20

```

## 1.4 Recuperando um subconjunto de colunas de uma tabela

### Problema

Você tem uma tabela e deseja ver os valores de colunas específicas em vez de todas as colunas.

### Solução

Especifique as colunas nas quais você está interessado. Por exemplo, para ver apenas o nome, o número do departamento e o salário dos funcionários:

```

1 select ename,deptno,sal
2   from emp

```

Ao especificar as colunas na cláusula SELECT, você garante que nenhum dado estranho seja retornado. Isso pode ser especialmente importante ao recuperar dados em uma rede, pois evita a perda de tempo inerente à recuperação de dados desnecessários.

## 1.5 Fornecendo nomes significativos para colunas

### Problema

Você gostaria de alterar os nomes das colunas retornadas por sua consulta para que sejam mais legíveis e compreensíveis. Considere esta consulta que retorna os salários e comissões de cada funcionário:

```
1 select sal,comm  
2   from emp
```

O que é SAL? É curto para *oferta*? É o nome de alguém? O que é COM? É comunicação? Você deseja que os resultados tenham rótulos mais significativos.

### Solução

Para alterar os nomes dos resultados da consulta, use a palavra-chave AS no formulário *nome\_original AS novo\_nome*. Alguns bancos de dados não requerem AS, mas todos aceitam:

```
1 select sal as salary, comm as commission  
2   from emp
```

SALARY	COMMISSION
800	
1600	300
1250	500
2975	
1250	1400
2850	
2450	
3000	
5000	
1500	0
1100	
950	
3000	
1300	

### Discussão

Usar a palavra-chave AS para dar novos nomes às colunas retornadas pela sua consulta é conhecido como *aliasing* aquelas colunas. Os novos nomes que você dá são conhecidos como *apelido*.

A criação de bons aliases pode ajudar bastante a tornar uma consulta e seus resultados comprehensíveis para outras pessoas.

## 1.6 Fazendo referência a uma coluna com alias na cláusula WHERE

### Problema

Você usou aliases para fornecer nomes de coluna mais significativos para seu conjunto de resultados e gostaria de excluir algumas das linhas usando a cláusula WHERE. No entanto, sua tentativa de referenciar nomes de alias na cláusula WHERE falha:

```
select sal as salary, comm as commission  
      from emp  
     where salary < 5000
```

### Solução

Ao agrupar sua consulta como uma exibição em linha, você pode referenciar as colunas com alias:

```
1 select *  
2   from (  
3 select sal as salary, comm as commission  
4   from emp  
5      ) x  
6  where salary < 5000
```

### Discussão

Neste exemplo simples, você pode evitar a exibição em linha e referenciar COMM ou SAL diretamente na cláusula WHERE para obter o mesmo resultado. Esta solução apresenta o que você precisaria fazer ao tentar fazer referência a qualquer um dos itens a seguir em uma cláusula WHERE:

- funções agregadas
- Subconsultas escalares
- Funções de janelas
- Apelido

Colocar sua consulta, aquela que fornece aliases, em uma exibição em linha permite que você faça referência às colunas com alias em sua consulta externa. Por que você tem que fazer isso? A cláusula WHERE é avaliada antes do SELECT; portanto, SALARY e COMMISSION ainda não existem quando a cláusula WHERE da consulta “Problema” é avaliada. Esses aliases não são aplicados até que o processamento da cláusula WHERE seja concluído. No entanto, a cláusula FROM é avaliada antes do WHERE. Ao colocar a consulta original em uma cláusula FROM, os resultados dessa consulta são gerados antes da cláusula mais externa

Cláusula WHERE e sua cláusula mais externa WHERE “vê” os nomes alternativos. Essa técnica é particularmente útil quando as colunas em uma tabela não são nomeadas muito bem.



A exibição em linha nesta solução tem alias de X. Nem todos os bancos de dados exigem que uma exibição em linha tenha um alias explicitamente, mas alguns exigem. Todos eles aceitam.

## 1.7 Valores de coluna de concatenação

### Problema

Você deseja retornar valores em várias colunas como uma coluna. Por exemplo, você gostaria de produzir este conjunto de resultados de uma consulta na tabela EMP:

```
CLARK WORKS AS A MANAGER  
KING WORKS AS A PRESIDENT  
MILLER WORKS AS A CLERK
```

No entanto, os dados necessários para gerar esse conjunto de resultados vêm de duas colunas diferentes, as colunas ENAME e JOB na tabela EMP:

```
select ename, job  
  from emp  
 where deptno = 10
```

ENAME	JOB
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

### Solução

Encontre e use a função interna fornecida pelo seu DBMS para concatenar valores de várias colunas.

#### DB2, Oracle, PostgreSQL

Esses bancos de dados usam a barra vertical dupla como o operador de concatenação:

```
1 select ename||' WORKS AS A '||job as msg  
2 from emp  
3 where deptno = 10
```

## MySQL

Este banco de dados suporta uma função chamada CONCAT:

```
1 select concat(ename, ' WORKS AS A ',job) as msg
2   from emp
3  where deptno=10
```

## SQL Server

Use o operador + para concatenação:

```
1 select ename + ' WORKS AS A ' + job as msg
2   from emp
3  where deptno=10
```

## Discussão

Use a função CONCAT para concatenar valores de várias colunas. O || é um atalho para a função CONCAT em DB2, Oracle e PostgreSQL, enquanto + é o atalho para SQL Server.

## 1.8 Usando a lógica condicional em uma instrução SELECT

### Problema

Você deseja executar operações IF-ELSE em valores em sua instrução SELECT. Por exemplo, você gostaria de produzir um conjunto de resultados de modo que, se um funcionário for pago US\$2.000 ou menos, uma mensagem de “UNDERPAID” é retornada; se um funcionário receber \$4.000 ou mais, uma mensagem de “OVERPAID” será retornada; e se eles chegarem a algum ponto intermediário, “OK” será retornado. O conjunto de resultados deve ficar assim:

ENAME	SAL	STATUS
SMITH	800	UNDERPAID
ALLEN	1600	UNDERPAID
WARD	1250	UNDERPAID
JONES	2975	OK
MARTIN	1250	UNDERPAID
BLAKE	2850	OK
CLARK	2450	OK
SCOTT	3000	OK
KING	5000	OVERPAID
TURNER	1500	UNDERPAID
ADAMS	1100	UNDERPAID
JAMES	950	UNDERPAID
FORD	3000	OK
MILLER	1300	UNDERPAID

Use a expressão CASE para executar a lógica condicional diretamente em sua instrução SELECT:

```
1 select ename,sal,
2       case when sal <= 2000 then 'UNDERPAID'
3             when sal >= 4000 then 'OVERPAID'
4             else 'OK'
5       end as status
6   from emp
```

## Discussão

A expressão CASE permite executar lógica de condição em valores retornados por uma consulta. Você pode fornecer um alias para uma expressão CASE para retornar um conjunto de resultados mais legível. Na solução, você verá o alias STATUS dado ao resultado da expressão CASE. A cláusula ELSE é opcional. Omita ELSE e a expressão CASE retornará NULL para qualquer linha que não satisfaça a condição de teste.

## 1.9 Limitando o número de linhas retornadas

### Problema

Você deseja limitar o número de linhas retornadas em sua consulta. Você não está preocupado com a ordem; qualquer *n*as linhas servirão.

### Solução

Use a função interna fornecida pelo seu banco de dados para controlar o número de linhas retornadas.

#### DB2

No DB2, use a cláusula FETCH FIRST:

```
1 select *
2 from emp fetch first 5 rows only
```

#### MySQL e PostgreSQL

Faça a mesma coisa no MySQL e PostgreSQL usando LIMIT:

```
1 select *
2 from emp limit 5
```

## Oracle

No Oracle, coloque uma restrição no número de linhas retornadas restringindo ROWNUM na cláusula WHERE:

```
1 select *
2   from emp
3  where rownum <= 5
```

## SQL Server

Use a palavra-chave TOP para restringir o número de linhas retornadas:

```
1 select top 5 *
2   from emp
```

## Discussão

Muitos fornecedores fornecem cláusulas como FETCH FIRST e LIMIT que permitem especificar o número de linhas a serem retornadas de uma consulta. O Oracle é diferente, pois você deve usar uma função chamada ROWNUM que retorna um número para cada linha retornada (um valor crescente a partir de um).

Aqui está o que acontece quando você usa ROWNUM  $\leq 5$  para retornar as cinco primeiras linhas:

1. O Oracle executa sua consulta.
2. O Oracle busca a primeira linha e a chama de linha número um.
3. Já passamos da linha número cinco? Se não, o Oracle retorna a linha, porque atende aos critérios de ser numerada menor ou igual a cinco. Se sim, o Oracle não retorna a linha.
4. O Oracle busca a próxima linha e avança o número da linha (para dois, depois para três, depois para quatro e assim por diante).
5. Vá para a etapa 3.

Como mostra esse processo, os valores do ROWNUM da Oracle são atribuídos *depois* cada linha é buscada. Este é um ponto importante e fundamental. Muitos desenvolvedores Oracle tentam retornar apenas, digamos, a quinta linha retornada por uma consulta especificando ROWNUM = 5.

Usar uma condição de igualdade em conjunto com ROWNUM é uma má ideia. Aqui está o que acontece quando você tenta retornar, digamos, a quinta linha usando ROWNUM = 5:

1. O Oracle executa sua consulta.
2. O Oracle busca a primeira linha e a chama de linha número um.
3. Já chegamos à linha número cinco? Se não, o Oracle descarta a linha, porque ela não atende aos critérios. Se sim, o Oracle retorna a linha. Mas a resposta nunca será sim!

4. O Oracle busca a próxima linha e a chama de linha número um. Isso ocorre porque a primeira linha a ser retornada da consulta deve ser numerada como uma.
5. Vá para a etapa 3.

Estude esse processo de perto e você verá por que o uso de ROWNUM = 5 para retornar a quinta linha falha. Você não pode ter uma quinta linha se não retornar primeiro as linhas de um a quatro!

Você pode notar que ROWNUM = 1 funciona, de fato, para retornar a primeira linha, o que pode parecer contradizer a explicação até agora. A razão pela qual ROWNUM = 1 funciona para retornar a primeira linha é que, para determinar se há alguma linha na tabela, o Oracle deve tentar buscar pelo menos uma vez. Leia o processo anterior cuidadosamente, substituindo um por cinco, e você entenderá por que não há problema em especificar ROWNUM = 1 como condição (para retornar uma linha).

## 1.10 Retornando n registros aleatórios de uma tabela

### Problema

Você deseja retornar um número específico de registros aleatórios de uma tabela. Você deseja modificar a seguinte instrução de modo que execuções sucessivas produzam um conjunto diferente de cinco linhas:

```
select ename, job  
  from emp
```

### Solução

Pegue qualquer função interna suportada pelo seu DBMS para retornar valores aleatórios. Use essa função em uma cláusula ORDER BY para classificar as linhas aleatoriamente. Em seguida, use a técnica da receita anterior para limitar o número de linhas classificadas aleatoriamente a serem retornadas.

### DB2

Use a função interna RAND em conjunto com ORDER BY e FETCH:

```
1 select ename,job  
2   from emp  
3  order by rand() fetch first 5 rows only
```

### MySQL

Use a função interna RAND em conjunto com LIMIT e ORDER BY:

```
1 select ename,job  
2   from emp  
3  order by rand() limit 5
```

## PostgreSQL

Use a função interna RANDOM em conjunto com LIMIT e ORDER BY:

```
1 select ename,job
2   from emp
3  order by random() limit 5
```

## Oracle

Use a função integrada VALUE, encontrada no pacote integrado DBMS\_RANDOM, em conjunto com ORDER BY e a função integrada ROWNUM:

```
1 select *
2   from (
3 select ename, job
4   from emp
5  order by dbms_random.value()
6      )
7 where rownum <= 5
```

## SQL Server

Use a função interna NEWID em conjunto com TOP e ORDER BY para retornar um conjunto de resultados aleatórios:

```
1 select top 5 ename,job
2   from emp
3  order by newid()
```

## Discussão

A cláusula ORDER BY pode aceitar o valor de retorno de uma função e usá-lo para alterar a ordem do conjunto de resultados. Todas essas soluções restringem o número de linhas a serem retornadas *depois* a função na cláusula ORDER BY é executada. Os usuários não-Oracle podem achar útil examinar a solução Oracle, pois ela mostra (conceitualmente) o que está acontecendo sob as coberturas das outras soluções.

É importante que você não confunda o uso de uma função na cláusula ORDER BY com o uso de uma constante numérica. Ao especificar uma constante numérica na cláusula ORDER BY, você está solicitando que a classificação seja feita de acordo com a coluna nessa posição ordinal na lista SELECT. Quando você especifica uma função na cláusula ORDER BY, a classificação é executada no resultado da função conforme ela é avaliada para cada linha.

## 1.11 Encontrando Valores Nulos

### Problema

Você deseja localizar todas as linhas que são nulas para uma coluna específica.

Para determinar se um valor é nulo, você deve usar IS NULL:

```
1 select *
2   from emp
3 where comm is null
```

## Discussão

NULL nunca é igual/não igual a nada, nem mesmo a si mesmo; portanto, você não pode usar = ou != para testar se uma coluna é NULL. Para determinar se uma linha tem valores NULL, você deve usar IS NULL. Você também pode usar IS NOT NULL para localizar linhas sem nulo em uma determinada coluna.

## 1.12 Transformando Nulos em Valores Reais

### Problema

Você tem linhas que contêm nulos e gostaria de retornar valores não nulos no lugar desses nulos.

### Solução

Use a função COALESCE para substituir valores reais por nulos:

```
1 select coalesce(comm,0)
2   from emp
```

### Discussão

A função COALESCE aceita um ou mais valores como argumentos. A função retorna o primeiro valor não nulo da lista. Na solução, o valor de COMM é retornado sempre que COMM não for nulo. Caso contrário, um zero é retornado.

Ao trabalhar com nulos, é melhor aproveitar a funcionalidade integrada fornecida pelo seu DBMS; em muitos casos, você descobrirá que várias funções funcionam igualmente bem para essa tarefa. COALESCE funciona para todos os DBMSs. Além disso, CASE também pode ser usado para todos os DBMSs:

```
select case
      when comm is not null then comm
      else 0
      end
   from emp
```

Embora você possa usar CASE para converter nulos em valores, você pode ver que é muito mais fácil e sucinto usar COALESCE.

## 1.13 Procurando padrões

### Problema

Você deseja retornar linhas que correspondam a uma substring ou padrão específico.

Considere a seguinte consulta e conjunto de resultados:

```
select ename, job
  from emp
 where deptno in (10,20)
```

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
SCOTT	ANALYST
KING	PRESIDENT
ADAMS	CLERK
FORD	ANALYST
MILLER	CLERK

Dos funcionários dos departamentos 10 e 20, você deseja retornar apenas aqueles que têm um “I” em algum lugar do nome ou um cargo que termina com “ER”:

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

### Solução

Use o operador LIKE em conjunto com o operador curinga SQL (%):

```
1 select ename, job
2 from emp
3 where deptno in (10,20)
4 and (ename like '%I%' or job like '%ER')
```

### Discussão

Quando usado em uma operação de correspondência de padrão LIKE, a porcentagem do operador (%) corresponde a qualquer sequência de caracteres. A maioria das implementações SQL também fornece o operador sublinhado (“\_”) para corresponder a um único caractere. Ao incluir o padrão de pesquisa “I” com % operadores, qualquer string que contenha um “I” (em qualquer posição) será retornada. Se você não incluir o padrão de pesquisa com %, o local onde você colocar o operador afetará os resultados da consulta.

Por exemplo, para localizar cargos que terminam em “ER”, prefixe o operador % para “ER”; se o requisito for pesquisar todos os cargos que começam com “ER”, acrescente o operador % a “ER”.

## **1.14 Resumindo**

Essas receitas podem ser simples, mas também são fundamentais. A recuperação de informações é o núcleo da consulta ao banco de dados, e isso significa que essas receitas estão no cerne de praticamente tudo o que é discutido no restante do livro.

## CAPÍTULO 2

# Classificando Resultados da Consulta

Este capítulo se concentra na personalização da aparência dos resultados da consulta. Ao entender como controlar como seu conjunto de resultados é organizado, você pode fornecer dados mais legíveis e significativos.

## 2.1 Retornando resultados de consulta em uma ordem especificada

### Problema

Você deseja exibir os nomes, cargos e salários dos funcionários do departamento 10 em ordem com base em seus salários (do menor para o maior). Você deseja retornar o seguinte conjunto de resultados:

ENAME	JOB	SAL
MILLER	CLERK	1300
CLARK	MANAGER	2450
KING	PRESIDENT	5000

### Solução

Use a cláusula ORDER BY:

```
1 select ename,job,sal
2   from emp
3  where deptno = 10
4 order by sal asc
```

### Discussão

A cláusula ORDER BY permite que você ordene as linhas do seu conjunto de resultados. A solução classifica as linhas com base no SAL em ordem crescente.

Por padrão, ORDER BY classificará em ordem crescente e, portanto, a cláusula ASC é opcional. Como alternativa, especifique DESC para classificar em ordem decrescente:

```
select ename,job,sal
  from emp
 where deptno = 10
 order by sal desc
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
CLARK	MANAGER	2450
MILLER	CLERK	1300

Você não precisa especificar o nome da coluna na qual classificar. Em vez disso, você pode especificar um número que representa a coluna. O número começa em 1 e corresponde aos itens na lista SELECT da esquerda para a direita. Por exemplo:

```
select ename,job,sal
  from emp
 where deptno = 10
 order by 3 desc
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
CLARK	MANAGER	2450
MILLER	CLERK	1300

O número 3 na cláusula ORDER BY deste exemplo corresponde à terceira coluna da lista SELECT, que é SAL.

## 2.2 Classificando por vários campos

### Problema

Você deseja classificar as linhas primeiro de EMP por DEPTNO ascendente e, em seguida, por salário decrescente. Você deseja retornar o seguinte conjunto de resultados:

EMPNO	DEPTNO	SAL	ENAME	JOB
7839	10	5000	KING	PRESIDENT
7782	10	2450	CLARK	MANAGER
7934	10	1300	MILLER	CLERK
7788	20	3000	SCOTT	ANALYST
7902	20	3000	FORD	ANALYST
7566	20	2975	JONES	MANAGER
7876	20	1100	ADAMS	CLERK
7369	20	800	SMITH	CLERK
7698	30	2850	BLAKE	MANAGER
7499	30	1600	ALLEN	SALESMAN

7844	30	1500	TURNER	SALESMAN
7521	30	1250	WARD	SALESMAN
7654	30	1250	MARTIN	SALESMAN
7900	30	950	JAMES	CLERK

## Solução

Liste as diferentes colunas de classificação na cláusula ORDER BY, separadas por vírgulas:

```
1 select empno,deptno,sal,ename,job
2   from emp
3  order by deptno, sal desc
```

## Discussão

A ordem de precedência em ORDER BY é da esquerda para a direita. Se você estiver ordenando usando a posição numérica de uma coluna na lista SELECT, esse número não deve ser maior que o número de itens na lista SELECT. Você geralmente tem permissão para ordenar por uma coluna que não esteja na lista SELECT, mas para fazer isso você deve nomear explicitamente a coluna. No entanto, se você estiver usando GROUP BY ou DISTINCT em sua consulta, não poderá ordenar por colunas que não estejam na lista SELECT.

## 2.3 Classificando por Substrings

### Problema

Você deseja classificar os resultados de uma consulta por partes específicas de uma string. Por exemplo, você deseja retornar nomes de funcionários e cargos da tabela EMP e classificar pelos dois últimos caracteres no campo JOB. O conjunto de resultados deve se parecer com o seguinte:

ENAME	JOB
KING	PRESIDENT
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK
JONES	MANAGER
CLARK	MANAGER
BLAKE	MANAGER
ALLEN	SALESMAN
MARTIN	SALESMAN
WARD	SALESMAN
TURNER	SALESMAN
SCOTT	ANALYST
FORD	ANALYST

### **DB2, MySQL, Oracle e PostgreSQL**

Use a função SUBSTR na cláusula ORDER BY:

```
select ename,job
  from emp
 order by substr(job,length(job)-1)
```

### **SQL Server**

Use a função SUBSTRING na cláusula ORDER BY:

```
select ename,job
  from emp
 order by substring(job,len(job)-1,2)
```

## **Discussão**

Usando a função substring do seu DBMS, você pode classificar facilmente por qualquer parte de uma string. Para classificar pelos dois últimos caracteres de uma string, encontre o final da string (que é o comprimento da string) e subtraia dois. A posição inicial será o penúltimo caractere da string. Você então pega todos os personagens após essa posição inicial. A SUBSTRING do SQL Server é diferente da função SUBSTR, pois requer um terceiro parâmetro que especifica quantos caracteres devem ser usados. Neste exemplo, qualquer número maior ou igual a dois funcionará.

## **2.4 Classificação de dados alfanuméricos mistos**

### **Problema**

Você misturou dados alfanuméricos e deseja classificar pela parte numérica ou de caracteres dos dados. Considere esta visão, criada a partir da tabela EMP:

```
create view V
as
select ename||' '|deptno as data
  from emp

select * from V

DATA
-----
SMITH 20
ALLEN 30
WARD 30
JONES 20
MARTIN 30
```

BLAKE 30  
CLARK 10  
SCOTT 20  
KING 10  
TURNER 30  
ADAMS 20  
JAMES 30  
FORD 20  
MILLER 10

Você deseja classificar os resultados por DEPTNO ou ENAME. A classificação por DEPTNO produz o seguinte conjunto de resultados:

DATA  
-----  
CLARK 10  
KING 10  
MILLER 10  
SMITH 20  
ADAMS 20  
FORD 20  
SCOTT 20  
JONES 20  
ALLEN 30  
BLAKE 30  
MARTIN 30  
JAMES 30  
TURNER 30  
WARD 30

Classificar por produz para ENAME o seguinte conjunto de resultados:

DATA  
-----  
ADAMS 20  
ALLEN 30  
BLAKE 30  
CLARK 10  
FORD 20  
JAMES 30  
JONES 20  
KING 10  
MARTIN 30  
MILLER 10  
SCOTT 20  
SMITH 20  
TURNER 30  
WARD 30

### Oracle, SQL Server e PostgreSQL

Use as funções REPLACE e TRANSLATE para modificar a string para classificação:

```
/* ORDER BY DEPTNO */

1 select data
2   from V
3  order by replace(data,
4                  replace(
5                      translate(data,'0123456789','#####'), '#', ''), ''))

/* ORDER BY ENAME */

1 select data
2   from V
3  order by replace(
4      translate(data,'0123456789','#####'), '#', '')
```

### DB2

A conversão implícita de tipo é mais rigorosa no DB2 do que no Oracle ou PostgreSQL, portanto, você precisará converter DEPTNO em um CHAR para que a exibição V seja válida. Em vez de recriar a exibição V, esta solução simplesmente usará uma exibição em linha. A solução usa REPLACE e TRANSLATE da mesma forma que a solução Oracle e PostgreSQL, mas a ordem dos argumentos para TRANSLATE é ligeiramente diferente para DB2:

```
/* ORDER BY DEPTNO */

1 select *
2   from (
3 select ename||' '||cast(deptno as char(2)) as data
4   from emp
5      ) v
6  order by replace(data,
7                  replace(
8                      translate(data,'#####', '0123456789'), '#', ''), ''))

/* ORDER BY ENAME */

1 select *
2   from (
3 select ename||' '||cast(deptno as char(2)) as data
4   from emp
5      ) v
6  order by replace(
7      translate(data,'#####', '0123456789'), '#', '')
```

## MySQL

A função TRANSLATE não é atualmente suportada por essas plataformas; assim, uma solução para este problema não será fornecida.

## Discussão

As funções TRANSLATE e REPLACE removem os números ou caracteres de cada linha, permitindo que você os classifique facilmente por um ou outro. Os valores passados para ORDER BY são mostrados nos seguintes resultados da consulta (usando a solução Oracle como exemplo, pois a mesma técnica se aplica a todos os três fornecedores; apenas a ordem dos parâmetros passados para TRANSLATE é o que diferencia o DB2):

```
select data,
       replace(data,
               replace(
                   translate(data,'0123456789','#####'), '#', ''),
               replace(
                   translate(data,'0123456789','#####'), '#', '')) nums,
       replace(
               translate(data,'0123456789','#####'), '#', '') chars
  from V
```

DATA	NUMS	CHARS
SMITH 20	20	SMITH
ALLEN 30	30	ALLEN
WARD 30	30	WARD
JONES 20	20	JONES
MARTIN 30	30	MARTIN
BLAKE 30	30	BLAKE
CLARK 10	10	CLARK
SCOTT 20	20	SCOTT
KING 10	10	KING
TURNER 30	30	TURNER
ADAMS 20	20	ADAMS
JAMES 30	30	JAMES
FORD 20	20	FORD
MILLER 10	10	MILLER

## 2.5 Lidando com Nulos ao Ordenar

### Problema

Você deseja classificar os resultados do EMP por COMM, mas o campo pode ser anulado. Você precisa de uma maneira de especificar se os nulos são classificados por último:

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500

MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

ou se eles classificam primeiro:

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

## Solução

Dependendo de como você deseja que os dados sejam exibidos e como seu RDBMS específico classifica os valores NULL, você pode classificar a coluna anulável em ordem crescente ou decrescente:

```

1 select ename,sal,comm
2   from emp
3  order by 3

1 select ename,sal,comm
2   from emp
3  order by 3 desc

```

Esta solução coloca você em uma posição tal que se a coluna anulável contiver valores não NULL, eles também serão ordenados em ordem crescente ou decrescente, de acordo com o que você pedir; isso pode ou não ser o que você tem em mente. Se, em vez disso, você quiser classificar os valores NULL de maneira diferente dos valores não NULL, por exemplo, você deseja classificar os valores não NULL em ordem crescente ou decrescente e todos os valores NULL por último, você pode usar uma expressão CASE para classificar condicionalmente a coluna.

## DB2, MySQL, PostgreSQL e SQL Server

Use uma expressão CASE para “sinalizar” quando um valor for NULL. A ideia é ter um sinalizador com dois valores: um para representar NULLs e outro para não-NULLs. Assim que tiver isso, basta adicionar esta coluna sinalizadora à cláusula ORDER BY. Você poderá facilmente controlar se os valores NULL são classificados primeiro ou por último sem interferir nos valores não NULL:

```
/* NON-NULL COMM SORTED ASCENDING, ALL NULLS LAST */

1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4       case when comm is null then 0 else 1 end as is_null
5   from emp
6      ) x
7 order by is_null desc,comm
```

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

```
/* NON-NULL COMM SORTED DESCENDING, ALL NULLS LAST */
```

```
1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4       case when comm is null then 0 else 1 end as is_null
5   from emp
6      ) x
7 order by is_null desc,comm desc
```

ENAME	SAL	COMM
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0
SMITH	800	

JONES	2975
JAMES	950
MILLER	1300
FORD	3000
ADAMS	1100
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000

```
/* NON-NULL COMM SORTED ASCENDING, ALL NULLS FIRST */
```

```
1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4       case when comm is null then 0 else 1 end as is_null
5   from emp
6     ) x
7 order by is_null,comm
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400

```
/* NON-NULL COMM SORTED DESCENDING, ALL NULLS FIRST */
```

```
1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4       case when comm is null then 0 else 1 end as is_null
5   from emp
6     ) x
7 order by is_null,comm desc
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	

BLAKE	2850
SCOTT	3000
KING	5000
JAMES	950
MILLER	1300
FORD	3000
ADAMS	1100
MARTIN	1250
WARD	1250
ALLEN	1600
TURNER	1500
	0

## Oracle

Os usuários Oracle podem usar a solução para as outras plataformas. Eles também podem usar a seguinte solução somente para Oracle, aproveitando a extensão NULLS FIRST e NULLS LAST para a cláusula ORDER BY para garantir que NULLs sejam classificados primeiro ou por último, independentemente de como os valores não NULL são classificados:

```
/* NON-NUL COMM SORTED ASCENDING, ALL NULLS LAST */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm nulls last
```

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

```
/* NON-NUL COMM SORTED ASCENDING, ALL NULLS FIRST */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm nulls first
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	

CLARK	2450
BLAKE	2850
SCOTT	3000
KING	5000
JAMES	950
MILLER	1300
FORD	3000
ADAMS	1100
TURNER	1500
ALLEN	1600
WARD	1250
MARTIN	1250
	1400

```
/* NON-NUL COMM SORTED DESCENDING, ALL NULLS FIRST */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm desc nulls first
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

## Discussão

A menos que seu RDBMS forneça uma maneira de classificar facilmente valores NULL primeiro ou último sem modificar valores não NULL na mesma coluna (como o Oracle faz), você precisará de uma coluna auxiliar.



No momento em que este livro foi escrito, os usuários do DB2 podem usar NULLS FIRST e NULLS LAST na subcláusula ORDER BY da cláusula OVER em funções de janela, mas não na cláusula ORDER BY para todo o conjunto de resultados.

O objetivo desta coluna extra (apenas na consulta, não na tabela) é permitir que você identifique valores NULL e os classifique completamente, primeiro ou último. A consulta a seguir retorna o conjunto de resultados para exibição em linha X para a solução não Oracle:

```
select ename,sal,comm,
       case when comm is null then 0 else 1 end as is_null
  from emp
```

ENAME	SAL	COMM	IS_NULL
SMITH	800		0
ALLEN	1600	300	1
WARD	1250	500	1
JONES	2975		0
MARTIN	1250	1400	1
BLAKE	2850		0
CLARK	2450		0
SCOTT	3000		0
KING	5000		0
TURNER	1500	0	1
ADAMS	1100		0
JAMES	950		0
FORD	3000		0
MILLER	1300		0

Usando os valores retornados por IS\_NULL, você pode facilmente classificar NULLS primeiro ou último sem interferir na classificação de COMM.

## 2.6 Classificando em uma chave dependente de dados

### Problema

Você deseja classificar com base em alguma lógica condicional. Por exemplo, se JOB for SALES-MAN, você deseja classificar em COMM; caso contrário, você deseja classificar por SAL. Você deseja retornar o seguinte conjunto de resultados:

ENAME	SAL	JOB	COMM
TURNER	1500	SALESMAN	0
ALLEN	1600	SALESMAN	300
WARD	1250	SALESMAN	500
SMITH	800	CLERK	
JAMES	950	CLERK	
ADAMS	1100	CLERK	
MILLER	1300	CLERK	
MARTIN	1250	SALESMAN	1400
CLARK	2450	MANAGER	
BLAKE	2850	MANAGER	
JONES	2975	MANAGER	

SCOTT	3000	ANALYST
FORD	3000	ANALYST
KING	5000	PRESIDENT

## Solução

Use uma expressão CASE na cláusula ORDER BY:

```
1 select ename,sal,job,comm
2 from emp
3 order by case when job = 'SALESMAN' then comm else sal end
```

## Discussão

Você pode usar a expressão CASE para alterar dinamicamente como os resultados são classificados. Os valores passados para o ORDER BY são os seguintes:

```
select ename,sal,job,comm,
       case when job = 'SALESMAN' then comm else sal end as ordered
     from emp
    order by 5
```

ENAME	SAL	JOB	COMM	ORDERED
TURNER	1500	SALESMAN	0	0
ALLEN	1600	SALESMAN	300	300
WARD1	250	SALESMAN	500	500
SMITH	800	CLERK		800
JAMES	950	CLERK		950
ADAMS	1100	CLERK		1100
MILLER	1300	CLERK		1300
MARTIN	1250	SALESMAN	1400	1400
CLARK2	450	MANAGER		2450
BLAKE2	850	MANAGER		2850
JONES2	975	MANAGER		2975
SCOTT	3000	ANALYST		3000
FORD	3000	ANALYST		3000
KING	5000	PRESIDENT		5000

## 3.1 Resumindo

A classificação dos resultados da consulta é uma das principais habilidades de qualquer usuário de SQL. A cláusula ORDER BY pode ser muito poderosa, mas, como vimos neste capítulo, ainda requer algumas nuances para ser usada com eficiência. É importante dominar seu uso, pois muitas das receitas dos capítulos posteriores dependem dele.

## CAPÍTULO 3

# Trabalhando com Várias Tabelas

Este capítulo apresenta o uso de junções e operações de conjunto para combinar dados de várias tabelas. As junções são a base do SQL. As operações de conjunto também são importantes. Se você quiser dominar as consultas complexas encontradas nos últimos capítulos deste livro, deve começar por aqui, com operações de junção e conjunto.

## 3.1 Empilhando um Rowset em cima de outro

### Problema

Você deseja retornar dados armazenados em mais de uma tabela, empilhando conceitualmente um conjunto de resultados sobre o outro. As tabelas não possuem necessariamente uma chave comum, mas suas colunas possuem os mesmos tipos de dados. Por exemplo, você deseja exibir o nome e o número do departamento dos funcionários do departamento 10 na tabela EMP, juntamente com o nome e o número do departamento de cada departamento na tabela DEPT. Você deseja que o conjunto de resultados se pareça com o seguinte:

ENAME_AND_DNAME	DEPTNO
CLARK	10
KING	10
MILLER	10
ACCOUNTING	10
RESEARCH	20
SALES	30
OPERATIONS	40

### Solução

Use a operação set UNION ALL para combinar linhas de várias tabelas:

```
1 select ename as ename_and_dname, deptno
2   from emp
3 where deptno = 10
4 union all
5 select '-----', null
6   from t1
7 union all
8 select dname, deptno
9   from dept
```

## Discussão

UNION ALL combina linhas de várias fontes de linha em um conjunto de resultados. Como em todas as operações de conjunto, os itens em todas as listas SELECT devem corresponder em número e tipo de dados. Por exemplo, ambas as consultas a seguir falharão:

```
select deptno    |  select deptno, dname
      from dept    |    from dept
      union all    |    union all
      select ename  |  select deptno
      from emp     |    from emp
```

É importante observar que UNION ALL incluirá duplicatas, se existirem. Se você deseja filtrar duplicatas, use o operador UNION. Por exemplo, uma UNION entre EMP.DEPTNO e DEPT.DEPTNO retorna apenas quatro linhas:

```
select deptno
      from emp
      union
      select deptno
      from dept

      DEPTNO
-----
      10
      20
      30
      40
```

Especificar UNION em vez de UNION ALL provavelmente resultará em uma operação de classificação para eliminar duplicatas. Lembre-se disso ao trabalhar com grandes conjuntos de resultados. O uso de UNION é aproximadamente equivalente à consulta a seguir, que aplica DISTINCT à saída de um UNION ALL:

```
select distinct deptno
      from (
      select deptno
      from emp
      union all
      select deptno
      from dept
      )
```

DEPTNO
10
20
30
40

Você não usaria DISTINCT em uma consulta, a menos que fosse necessário, e a mesma regra se aplica a UNION: não use em vez de UNION ALL, a menos que seja necessário. Por exemplo, embora neste livro tenhamos limitado o número de tabelas para fins de ensino, na vida real, se você estiver consultando uma tabela, pode haver uma maneira mais adequada de consultar uma única tabela.

## 3.2 Combinando Linhas Relacionadas

### Problema

Você deseja retornar linhas de várias tabelas unindo-se em uma coluna comum conhecida ou unindo-se em colunas que compartilham valores comuns. Por exemplo, você deseja exibir os nomes de todos os funcionários do departamento 10 junto com a localização do departamento de cada funcionário, mas esses dados são armazenados em duas tabelas separadas. Você deseja que o conjunto de resultados seja o seguinte:

ENAME	LOC
CLARK	NEW YORK
KING	NEW YORK
MILLER	NEW YORK

### Solução

Junte a tabela EMP à tabela DEPT em DEPTNO:

```

1 select e.ename, d.loc
2   from emp e, dept d
3  where e.deptno = d.deptno
4    and e.deptno = 10

```

### Discussão

A solução é um exemplo de *juntar*, ou mais precisamente uma *união equitativa*, que é um tipo de *junção interna*. Uma junção é uma operação que combina linhas de duas tabelas em uma. Uma junção equitativa é aquela em que a condição de junção é baseada em uma condição de igualdade (por exemplo, onde um número de departamento é igual a outro). Uma junção interna é o tipo original de junção; cada linha retornada contém dados de cada tabela.

## Capítulo 3

42

Conceitualmente, o conjunto de resultados de uma junção é produzido criando primeiro um produto cartesiano (todas as combinações possíveis de linhas) a partir das tabelas listadas na cláusula FROM, conforme mostrado aqui:

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
  from emp e, dept d
 where e.deptno = 10
```

ENAME	LOC	EMP_DEPTNO	DEPT_DEPTNO
CLARK	NEW YORK	10	10
KING	NEW YORK	10	10
MILLER	NEW YORK	10	10
CLARK	DALLAS	10	20
KING	DALLAS	10	20
MILLER	DALLAS	10	20
CLARK	CHICAGO	10	30
KING	CHICAGO	10	30
MILLER	CHICAGO	10	30
CLARK	BOSTON	10	40
KING	BOSTON	10	40
MILLER	BOSTON	10	40

Todos os funcionários da tabela EMP (no departamento 10) são retornados junto com todos os departamentos da tabela DEPT. Então, a expressão na cláusula WHERE envolvendo e.deptno e d.deptno (a junção) restringe o conjunto de resultados de forma que as únicas linhas retornadas sejam aquelas em que EMP.DEPTNO e DEPT.DEPTNO são iguais:

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
  from emp e, dept d
 where e.deptno = d.deptno
   and e.deptno = 10
```

ENAME	LOC	EMP_DEPTNO	DEPT_DEPTNO
CLARK	NEW YORK	10	10
KING	NEW YORK	10	10
MILLER	NEW YORK	10	10

Uma solução alternativa faz uso de uma cláusula explícita JOIN (a palavra-chave INNER é opcional):

```
select e.ename, d.loc
  from emp e inner join dept d
    on (e.deptno = d.deptno)
   where e.deptno = 10
```

Use a cláusula JOIN se preferir ter a lógica de junção na cláusula FROM em vez da cláusula WHERE. Ambos os estilos são compatíveis com ANSI e funcionam em todas as versões mais recentes dos RDBMSs deste livro.

## 3.3 Encontrando linhas em comum entre duas tabelas

### Problema

Você deseja localizar linhas comuns entre duas tabelas, mas há várias colunas nas quais você pode unir. Por exemplo, considere a seguinte visão V criada a partir da tabela EMP para fins didáticos:

```
create view V
as
select ename,job,sal
  from emp
 where job = 'CLERK'

select * from V

ENAME      JOB          SAL
-----
SMITH      CLERK        800
ADAMS      CLERK        1100
JAMES      CLERK        950
MILLER     CLERK        1300
```

Somente os funcionários são retornados da exibição V. No entanto, a exibição não mostra todas as colunas possíveis EMP. Você deseja retornar o EMPNO, ENAME, JOB, SAL e DEPTNO de todos os funcionários no EMP que correspondem às linhas da exibição V. Você deseja que o conjunto de resultados seja o seguinte:

EMPNO	ENAME	JOB	SAL	DEPTNO
7369	SMITH	CLERK	800	20
7876	ADAMS	CLERK	1100	20
7900	JAMES	CLERK	950	30
7934	MILLER	CLERK	1300	10

### Solução

Junte as tabelas em todas as colunas necessárias para retornar o resultado correto. Como alternativa, use a operação set INTERSECT para evitar a realização de uma junção e, em vez disso, retornar a interseção (linhas comuns) das duas tabelas.

#### MySQL e SQLServer

Join table EMP para visualizar V usando várias condições de junção:

```
1 select e.empno,e.ename,e.job,e.sal,e.deptno
2   from emp e, V
3  where e.ename = v.ename
4    and e.job    = v.job
5    and e.sal    = v.sal
```

Como alternativa, você pode executar a mesma junção por meio da cláusula JOIN:

```
1 select e.empno,e.ename,e.job,e.sal,e.deptno
2   from emp e join V
3  on (   e.ename    = v.ename
4       and e.job     = v.job
5       and e.sal     = v.sal )
```

### DB2, Oracle e PostgreSQL

A solução MySQL e SQL Server também funciona para DB2, Oracle e PostgreSQL. É a solução que você deve usar se precisar retornar valores da visualização V.

Se você realmente não precisa retornar colunas da visão V, você pode usar a operação set INTERSECT junto com um predicado IN:

```
1 select empno,ename,job,sal,deptno
2   from emp
3  where (ename,job,sal) in (
4    select ename, job, sal de emp
5  intersect
6    select ename,job,sal from V
7 )
```

## Discussão

Ao realizar junções, você deve considerar as colunas adequadas para unir a fim de retornar resultados corretos. Isso é especialmente importante quando as linhas podem ter valores comuns para algumas colunas e valores diferentes para outras.

A operação de configuração INTERSECT retornará linhas comuns a ambas as fontes de linha. Ao usar o INTERSECT, é necessário comparar o mesmo número de itens, com o mesmo tipo de dados, de duas tabelas. Ao trabalhar com operações de conjunto, lembre-se de que, por padrão, linhas duplicadas não serão retornadas.

## 3.4 Recuperando valores de uma tabela que não existem em outra

### Problema

Você deseja encontrar esses valores em uma tabela, chame-a de tabela de origem, que também não existe em alguma tabela de destino.

Por exemplo, você deseja descobrir quais departamentos (se houver) na tabela DEPT não existem na tabela EMP. Nos dados de exemplo, DEPTNO 40 da tabela DEPT não existe na tabela EMP, portanto o conjunto de resultados deve ser o seguinte:

```
DEPTNO
-----
40
```

## Solução

Ter funções que executam diferenças de conjuntos é particularmente útil para esse problema. DB2, PostgreSQL, SQL Server e Oracle suportam operações de diferença de conjunto. Se o seu DBMS não suportar uma função de diferença definida, use uma subconsulta conforme mostrado para MySQL.

### DB2, PostgreSQL e SQL Server

Use a operação de configuração, EXCETO:

```
1 select deptno from dept
2 except
3 select deptno from emp
```

### Oracle

Use a operação definida MENOS:

```
1 select deptno from dept
2 minus
3 select deptno from emp
```

### MySQL

Use uma subconsulta para retornar todos os DEPTNOs da tabela EMP para uma consulta externa que procura na tabela DEPT as linhas que não estão entre as linhas retornadas da subconsulta:

```
1 select deptno
2 from dept
3 where deptno not in (select deptno from emp)
```

## Discussão

### DB2, PostgreSQL e SQL Server

As funções de definição de diferença facilitam esta operação. O operador EXCEPT pega o primeiro conjunto de resultados e remove dele todas as linhas encontradas no segundo conjunto de resultados. A operação é muito parecida com uma subtração.

Existem restrições quanto ao uso de operadores de conjunto, incluindo EXCEPT. Os tipos de dados e o número de valores a serem comparados devem corresponder em ambas as listas SELECT.

Além disso, EXCEPT não retornará duplicatas e, ao contrário de uma subconsulta usando NOT IN, NULLs não apresentam problemas (consulte a discussão sobre MySQL). O operador EXCEPT retornará linhas da consulta superior (a consulta antes do EXCEPT) que não existem na consulta inferior (a consulta após o EXCEPT).

## Oracle

A solução Oracle é idêntica à solução que usa o operador EXCETO; no entanto, o Oracle chama seu operador de diferença de conjunto MINUS em vez de EXCEPT. Caso contrário, a explicação anterior também se aplica ao Oracle.

## MySQL

A subconsulta retornará todos os DEPTNOs da tabela EMP. A consulta externa retorna todos os DEPTNOs da tabela DEPT que “não estão” ou “não estão incluídos” no conjunto de resultados retornado da subconsulta.

A eliminação de duplicados é algo que você deve considerar ao usar as soluções MySQL. As soluções baseadas em EXCEPT e MINUS usadas para as outras plataformas eliminam linhas duplicadas do conjunto de resultados, garantindo que cada DEPTNO seja relatado apenas uma vez. Claro, isso só pode ser o caso de qualquer maneira, já que DEPTNO é um campo-chave em meus dados de exemplo. Se DEPTNO não fosse um campo-chave, você poderia usar DISTINCT da seguinte forma para garantir que cada valor DEPTNO ausente de EMP seja relatado apenas uma vez:

```
select distinct deptno
  from dept
 where deptno not in (select deptno from emp)
```

Esteja atento aos NULLs ao usar NOT IN. Considere a seguinte tabela, NEW\_DEPT:

```
create table new_dept(deptno integer)
insert into new_deptvalues (10)
insert into new_dept values (50)
insert into new_dept values (null)
```

Se você tentar encontrar os DEPTNOs na tabela DEPT que não existem na tabela NEW\_DEPT e usar uma subconsulta com NOT IN, verá que a consulta não retorna nenhuma linha:

```
select *
  from dept
 where deptno not in (select deptno from new_dept)
```

Os DEPTNOs 20, 30 e 40 não estão na tabela NEW\_DEPT, mas não foram retornados pela consulta. Por que? O motivo é o valor NULL presente na tabela NEW\_DEPT. Três linhas são retornadas pela subconsulta, com DEPTNOs de 10, 50 e NULL. IN e NOT IN são essencialmente operações OR e produzirão resultados diferentes por causa de como os valores NULL são tratados por avaliações OR lógicas.

Para entender isso, examine estas tabelas de verdade (seja T=verdadeiro, F=falso, N=nulo):

OR	T	F	N
	T	T	T
	F	T	F
	N	N	N

NOT
T   F
F   T
N   N

AND	T	F	N
	T	T	F
	F	F	F
	N	N	F

Agora considere o seguinte exemplo usando IN e seu equivalente usando OR:

```
select deptno
  from dept
 where deptno in ( 10,50,null )

DEPTNO
-----
      10

select deptno
  from dept
 where (deptno=10 or deptno=50 or deptno=null)

DEPTNO
-----
      10
```

Por que apenas o DEPTNO 10 foi devolvido? Existem quatro DEPTNOs em DEPT, (10, 20, 30, 40), e cada um é avaliado em relação ao predicado (deptno=10 ou deptno=50 ou deptno=nulo). De acordo com as tabelas de verdade anteriores, para cada DEPTNO (10, 20, 30, 40), o predicado produz:

```
DEPTNO=10
(deptno=10 or deptno=50 or deptno=null)
= (10=10 or 10=50 or 10=null)
= (T or F or N)
= (T or N)
= (T)
```

```
DEPTNO=20
(deptno=10 or deptno=50 or deptno=null)
= (20=10 or 20=50 or 20=null)
= (F or F or N)
= (F or N)
= (N)
```

```
DEPTNO=30
(deptno=10 or deptno=50 or deptno=null)
= (30=10 or 30=50 or 30=null)
= (F or F or N)
= (F or N)
= (N)
```

```
DEPTNO=40
(deptno=10 or deptno=50 or deptno=null)
= (40=10 or 40=50 or 40=null)
= (F or F or N)
= (F or N)
= (N)
```

Agora é óbvio porque apenas DEPTNO 10 foi retornado ao usar IN e OR. Em seguida, considere o mesmo exemplo usando NOT IN e NOT OR:

```
select deptno
  from dept
 where deptno not in ( 10,50,null )

( no rows )

select deptno
  from dept
 where not (deptno=10 or deptno=50 or deptno=null)

( no rows )
```

Por que nenhuma linha é retornada? Vamos verificar as tabelas verdade:

```
DEPTNO=10
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (10=10 or 10=50 or 10=null)
= NOT (T or F or N)
= NOT (T or N)
= NOT (T)
= (F)
```

```
DEPTNO=20
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (20=10 or 20=50 or 20=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)
```

```

DEPTNO=30
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (30=10 or 30=50 or 30=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)

DEPTNO=40
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (40=10 or 40=50 or 40=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)

```

Em SQL, “TRUE or NULL” é TRUE, mas “FALSE or NULL” é NULL! Você deve ter isso em mente ao usar predicados IN e ao executar avaliações OR lógicas e valores NULL estão envolvidos.

Para evitar o problema com NOT IN e NULLs, use uma subconsulta correlacionada em conjunto com NOT EXISTS. O termo *subconsulta correlacionada* é usado porque as linhas da consulta externa são referenciadas na subconsulta. O exemplo a seguir é uma solução alternativa que não será afetada por linhas NULL (voltando à consulta original da seção “Problema”):

```

select d.deptno
  from dept d
 where not exists (
   select 1
     from emp e
    where d.deptno = e.deptno
 )

```

```

DEPTNO
-----
40

```

```

select d.deptno
  from dept d
 where not exists (
   select 1
     from new_dept nd
    where d.deptno = nd.deptno
 )

```

```

DEPTNO
-----
30
40
20

```

Conceitualmente, a consulta externa nesta solução considera cada linha da tabela DEPT. Para cada linha DEPT, acontece o seguinte:

1. A subconsulta é executada para verificar se o número do departamento existe na tabela EMP. Observe a condição D.DEPTNO = E.DEPTNO, que reúne os números dos departamentos das duas tabelas.
2. Se a subconsulta retornar resultados, EXISTS (...) será avaliado como true e NOT EXISTS (...) será avaliado como FALSE, e a linha considerada pela consulta externa será descartada.
3. Se a subconsulta não retornar nenhum resultado, NOT EXISTS (...) será avaliado como TRUE, e a linha considerada pela consulta externa será retornada (porque é para um departamento não representado na tabela EMP).

Os itens na lista SELECT da subconsulta não são importantes ao usar uma subconsulta correlacionada com EXISTS/NOT EXISTS, e é por isso que escolhemos selecionar NULL, para forçá-lo a focar na junção na subconsulta em vez dos itens na lista SELECT .

## 3.5 Recuperando linhas de uma tabela que não correspondem a linhas em outra

### Problema

Você deseja localizar linhas que estão em uma tabela que não possuem uma correspondência em outra tabela, para duas tabelas que possuem chaves comuns. Por exemplo, você deseja descobrir quais departamentos não têm funcionários. O conjunto de resultados deve ser o seguinte:

DEPTNO	DNAME	LOC
40	OPERATIONS	BOSTON

Encontrar o departamento em que cada funcionário trabalha requer uma união equivalente em DEPTNO de EMP para DEPT. A coluna DEPTNO representa o valor comum entre as tabelas. Infelizmente, uma junção por equivalência não mostrará qual departamento não possui funcionários. Isso ocorre porque, ao unir EMP e DEPT, você retorna todas as linhas que satisfazem a condição de junção. Em vez disso, você deseja apenas as linhas de DEPT que não atendem à condição de junção.

Este é um problema sutilmente diferente da receita anterior, embora à primeira vista possam parecer iguais. A diferença é que a receita anterior fornece apenas uma lista de números de departamento não representados na tabela EMP. Usando esta receita, no entanto, você pode facilmente retornar outras colunas da tabela DEPT; você pode retornar mais do que apenas números de departamento.

## Solução

Retorne todas as linhas de uma tabela juntamente com as linhas de outra que podem ou não ter uma correspondência na coluna comum. Em seguida, mantenha apenas as linhas sem correspondência.

### DB2, MySQL, PostgreSQL e SQL Server

Use uma junção externa e filtre para NULLs (a palavra-chave OUTER é opcional):

```

1 select d.*
2 from dept d left outer join emp e
3 on (d.deptno = e.deptno)
4 where e.deptno is null

```

## Discussão

Essa solução funciona por junção externa e, em seguida, mantendo apenas as linhas que não têm correspondência. Este tipo de operação é algumas vezes chamado de *anti-junção*. Para ter uma ideia melhor de como um anti-join funciona, primeiro examine o conjunto de resultados sem filtrar por NULLs:

```

select e.ename, e.deptno as emp_deptno, d.*
  from dept d left join emp e
    on (d.deptno = e.deptno)

```

ENAME	EMP_DEPTNO	DEPTNO	DNAME	LOC
SMITH	20	20	RESEARCH	DALLAS
ALLEN	30	30	SALES	CHICAGO
WARD	30	30	SALES	CHICAGO
JONES	20	20	RESEARCH	DALLAS
MARTIN	30	30	SALES	CHICAGO
BLAKE	30	30	SALES	CHICAGO
CLARK	10	10	ACCOUNTING	NEW YORK
SCOTT	20	20	RESEARCH	DALLAS
KING	10	10	ACCOUNTING	NEW YORK
TURNER	30	30	SALES	CHICAGO
ADAMS	20	20	RESEARCH	DALLAS
JAMES	30	30	SALES	CHICAGO
FORD	20	20	RESEARCH	DALLAS
MILLER	10	10	ACCOUNTING	NEW YORK
		40	OPERATIONS	BOSTON

Observe que a última linha tem um valor NULL para EMP.ENAME e EMP\_DEPTNO. Isso ocorre porque nenhum funcionário trabalha no departamento 40. A solução usa a cláusula WHERE para manter apenas as linhas em que EMP\_DEPTNO é NULL (mantendo assim apenas as linhas de DEPT que não têm correspondência em EMP).

## 3.6 Adicionando junções a uma consulta sem interferir em outras junções

### Problema

Você tem uma consulta que retorna os resultados desejados. Você precisa de informações adicionais, mas ao tentar obtê-las, perde dados do conjunto de resultados original. Por exemplo, você deseja retornar todos os funcionários, a localização do departamento em que trabalham e a data em que receberam um bônus. Para este problema, a tabela EMP\_BONUS contém os seguintes dados:

```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7369	14-MAR-2005	1
7900	14-MAR-2005	2
7788	14-MAR-2005	3

A consulta com a qual você começa se parece com isto:

```
select e.ename, d.loc
  from emp e, dept d
 where e.deptno=d.deptno
```

ENAME	LOC
SMITH	DALLAS
ALLEN	CHICAGO
WARD	CHICAGO
JONES	DALLAS
MARTIN	CHICAGO
BLAKE	CHICAGO
CLARK	NEW YORK
SCOTT	DALLAS
KING	NEW YORK
TURNER	CHICAGO
ADAMS	DALLAS
JAMES	CHICAGO
FORD	DALLAS
MILLER	NEW YORK

Você deseja adicionar a esses resultados a data em que um bônus foi concedido a um funcionário, mas a junção à tabela EMP\_BONUS retorna menos linhas do que você deseja, porque nem todos os funcionários têm bônus:

```
select e.ename, d.loc, eb.received
  from emp e, dept d, emp_bonus eb
 where e.deptno=d.deptno
   and e.empno=eb.empno
```

ENAME	LOC	RECEIVED
SCOTT	DALLAS	14-MAR-2005
SMITH	DALLAS	14-MAR-2005
JAMES	CHICAGO	14-MAR-2005

O conjunto de resultados desejado é o seguinte:

ENAME	LOC	RECEIVED
ALLEN	CHICAGO	
WARD	CHICAGO	
MARTIN	CHICAGO	
JAMES	CHICAGO	14-MAR-2005
TURNER	CHICAGO	
BLAKE	CHICAGO	
SMITH	DALLAS	14-MAR-2005
FORD	DALLAS	
ADAMS	DALLAS	
JONES	DALLAS	
SCOTT	DALLAS	14-MAR-2005
CLARK	NEW YORK	
KING	NEW YORK	
MILLER	NEW YORK	

## Solução

Você pode usar uma junção externa para obter as informações adicionais sem perder os dados da consulta original. Primeiro junte a tabela EMP à tabela DEPT para obter todos os funcionários e a localização do departamento em que trabalham, depois faça a junção externa à tabela EMP\_BONUS para retornar a data do bônus, se houver. A seguir está a sintaxe do servidor DB2, MySQL, PostgreSQL e SQL:

```

1 select e.ename, d.loc, eb.received
2 from emp e join dept d
3 on (e.deptno=d.deptno)
4 left join emp_bonus eb
5 on (e.empno=eb.empno)
6 order by 2

```

Você também pode usar uma subconsulta escalar (uma subconsulta colocada na lista SELECT) para imitar uma junção externa:

```

1 select e.ename, d.loc,
2 (select eb.received from emp_bonus eb
3 where eb.empno=e.empno) as received
4 from emp e, dept d
5 where e.deptno=d.deptno
6 order by 2

```

A solução de subconsulta escalar funcionará em todas as plataformas.

Uma junção externa retornará todas as linhas de uma tabela e as linhas correspondentes de outra. Consulte a receita anterior para obter outro exemplo dessa junção. A razão pela qual uma junção externa funciona para resolver esse problema é que ela não resulta na eliminação de nenhuma linha que, de outra forma, seria retornada. A consulta retornará todas as linhas que retornaria sem a junção externa. E também retorna a data de recebimento, se houver.

O uso de uma subconsulta escalar também é uma técnica conveniente para esse tipo de problema, pois não exige que você modifique junções já corretas em sua consulta principal. Usar uma subconsulta escalar é uma maneira fácil de incluir dados extras em uma consulta sem comprometer o conjunto de resultados atual. Ao trabalhar com subconsultas escalares, você deve garantir que elas retornem um valor escalar (único). Se uma subconsulta na lista SELECT retornar mais de uma linha, você receberá um erro.

## Veja também

Ver [Receita 14.10](#) para obter uma solução alternativa para o problema de não poder retornar várias linhas de uma subconsulta da lista SELECT.

## 3.7 Determinando se duas tabelas têm os mesmos dados

### Problema

Você deseja saber se duas tabelas ou exibições têm os mesmos dados (cardinalidade e valores). Considere a seguinte visão:

```
create view V
as
select * from emp where deptno != 10
union all
select * from emp where ename = 'WARD'

select * from V
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-2005	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-2006	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30
7566	JONES	MANAGER	7839	02-APR-2006	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-2006	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-2006	2850		30
7788	SCOTT	ANALYST	7566	09-DEC-2007	3000		20
7844	TURNER	SALESMAN	7698	08-SEP-2006	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-2008	1100		20
7900	JAMES	CLERK	7698	03-DEC-2006	950		30

7902 FORD	ANALYST	7566	03-DEC-2006	3000		20
7521 WARD	SALESMAN	7698	22-FEB-2006	1250	500	30

Você deseja determinar se esta visão tem exatamente os mesmos dados que a tabela EMP. A linha do funcionário WARD é duplicada para mostrar que a solução revelará não apenas dados diferentes, mas também dados duplicados. Com base nas linhas da tabela EMP, a diferença será as três linhas para os funcionários do departamento 10 e as duas linhas para o funcionário WARD. Você deseja retornar o seguinte conjunto de resultados:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	1
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	2
7782	CLARK	MANAGER	7839	09-JUN-2006	2450		10	1
7839	KING	PRESIDENT		17-NOV-2006	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-2007	1300		10	1

## Solução

Funções que executam SET diferença MINUS ou EXCEPT, dependendo do seu DBMS, tornam o problema de comparação de tabelas relativamente fácil de resolver. Se o seu DBMS não oferece tais funções, você pode usar uma subconsulta correlacionada.

### DB2 e PostgreSQL

Use as operações definidas EXCEPT e UNION ALL para encontrar a diferença entre a visualização V e a tabela EMP combinada com a diferença entre a tabela EMP e a visualização V:

```

1  (
2   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
3       count(*) as cnt
4   from V
5   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
6 except
7   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
8       count(*) as cnt
9   from emp
10  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11  )
12 union all
13 (
14  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15      count(*) as cnt
16  from emp
17  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18 except
19  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20      count(*) as cnt
21  from v
22  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23)

```

Use as operações definidas MINUS e UNION ALL para encontrar a diferença entre a visualização V e a tabela EMP combinada com a diferença entre a tabela EMP e a visualização V:

```
1 (
2   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
3         count(*) as cnt
4   from V
5   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
6   minus
7   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
8         count(*) as cnt
9   from emp
10  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11 )
12 union all
13 (
14   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15         count(*) as cnt
16   from emp
17   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18   minus
19   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20         count(*) as cnt
21   from v
22   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23)
```

## **MySQL e SQLServer**

Use uma subconsulta correlacionada e UNION ALL para localizar as linhas na visualização V e não na tabela EMP combinadas com as linhas na tabela EMP e não na visualização V:

```
1 select *
2 from (
3 select e.empno,e.ename,e.job,e.mgr,e.hiredate,
4 e.sal,e.comm,e.deptno, count(*) as cnt
5 from emp e
6 group by empno,ename,job,mgr,hiredate,
7 sal,comm,deptno
8 ) e
9 where not exists (
10 select null
11 from (
12 select v.empno,v.ename,v.job,v.mgr,v.hiredate,
13 v.sal,v.comm,v.deptno, count(*) as cnt
14 from v
15 group by empno,ename,job,mgr,hiredate,
16 sal,comm,deptno
17 ) v
18 where v.empno      = e.empno
19 and v.ename       = e.ename
```

```

1      and v.job      = e.job
2      and coalesce(v.mgr,0) = coalesce(e.mgr,0)
3      and v.hiredate = e.hiredate
4      and v.sal       = e.sal
5      and v.deptno   = e.deptno
6      and v.cnt      = e.cnt
7      and coalesce(v.comm,0) =
coalesce(e.comm,0) 27 )
28      union all
29      select *
30      from (
31      select v.empno,v.ename,v.job,v.mgr,v.hiredate,
32          v.sal,v.comm,v.deptno, count(*) as cnt
33      from v
34      group by empno,ename,job,mgr,hiredate,
35          sal,comm,deptno
36      ) v
37      where not exists (
38      select null
39      from (
40      select e.empno,e.ename,e.job,e.mgr,e.hiredate,
41          e.sal,e.comm,e.deptno, count(*) as cnt
42      from emp e
43      group by empno,ename,job,mgr,hiredate,
44          sal,comm,deptno
45      ) e
46      where v.empno      = e.empno
47      and v.ename       = e.ename
48      and v.job        = e.job
49      and coalesce(v.mgr,0) = coalesce(e.mgr,0)
50      and v.hiredate   = e.hiredate
51      and v.sal        = e.sal
52      and v.deptno    = e.deptno
53      and v.cnt       = e.cnt
28      and coalesce(v.comm,0) =
coalesce(e.comm,0)
55)

```

## Discussão

Apesar de usar técnicas diferentes, o conceito é o mesmo para todas as soluções:

1. Encontre as linhas na tabela EMP que não existem na visão V.
2. Combine (UNION ALL) essas linhas com as linhas da visão V que não existem na tabela EMP.

Se as tabelas em questão forem iguais, nenhuma linha será retornada. Se as tabelas forem diferentes, as linhas que causam a diferença serão retornadas. Como uma primeira etapa fácil ao comparar tabelas, você pode comparar apenas as cardinalidades em vez de incluí-las na comparação de dados.

A consulta a seguir é um exemplo simples disso e funcionará em todos os DBMSs:

```
select count(*)
  from emp
 union
select count(*)
  from dept
```

COUNT(\*)

-----

4  
14

Como UNION filtrará duplicatas, apenas uma linha será retornada se as cardinalidades das tabelas forem as mesmas. Como duas linhas são retornadas neste exemplo, você sabe que as tabelas não contêm conjuntos de linhas idênticos.

### DB2, Oracle e PostgreSQL

MINUS e EXCEPT funcionam da mesma forma, então usaremos EXCEPT para esta discussão. As consultas antes e depois de UNION ALL são semelhantes. Então, para entender como funciona a solução, basta executar a query antes do UNION ALL sozinho. O seguinte conjunto de resultados é produzido pela execução das linhas 1–11 na seção “Solução”:

```
(  
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,  
        count(*) as cnt  
   from V  
  group by empno,ename,job,mgr,hiredate,sal,comm,deptno  
except  
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,  
        count(*) as cnt  
   from emp  
  group by empno,ename,job,mgr,hiredate,sal,comm,deptno  
)  
  
EMPNO ENAME      JOB          MGR  HIREDATE        SAL    COMM DEPTNO CNT  
----- -----  
7521  WARD       SALESMAN    7698 22-FEB-2006  1250    500     30    2
```

O conjunto de resultados representa uma linha encontrada na exibição V que não está na tabela EMP ou tem uma cardinalidade diferente daquela mesma linha na tabela EMP. Nesse caso, a linha duplicada do funcionário WARD é localizada e retornada. Se você ainda estiver tendo problemas para entender como o conjunto de resultados é produzido, execute cada consulta em cada lado de EXCEPT individualmente. Você notará que a única diferença entre os dois conjuntos de resultados é o CNT do funcionário WARD retornado pela visualização V.

A parte da consulta após UNION ALL faz o oposto da consulta anterior a UNION ALL. A consulta retorna linhas na tabela EMP que não estão na visão V:

```

(
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from emp
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
   minus
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from v
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	1
7782	CLARK	MANAGER	7839	09-JUN-2006	2450		10	1
7839	KING	PRESIDENT		17-NOV-2006	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-2007	1300		10	1

Os resultados são então combinados por UNION ALL para produzir o conjunto de resultados final.

## MySQL e SQLServer

As consultas antes e depois de UNION ALL são semelhantes. Para entender como funciona a solução baseada em subconsultas, basta executar a consulta anterior ao UNION ALL sozinho. A consulta a seguir é das linhas 1 a 27 na solução:

```

select *
  from (
  select e.empno,e.ename,e.job,e.mgr,e.hiredate,
         e.sal,e.comm,e.deptno, count(*) as cnt
    from emp e
   group by empno,ename,job,mgr,hiredate,
            sal,comm,deptno
      ) e
 where not exists (
select null
  from (
  select v.empno,v.ename,v.job,v.mgr,v.hiredate,
         v.sal,v.comm,v.deptno, count(*) as cnt
    from v
   group by empno,ename,job,mgr,hiredate,
            sal,comm,deptno
      ) v
 where v.empno = e.empno
   and v.ename = e.ename
   and v.job = e.job
   and v.mgr = e.mgr
   and v.hiredate = e.hiredate
)

```

```

    and v.sal      = e.sal
    and v.deptno   = e.deptno
    and v.cnt      = e.cnt
    and coalesce(v.comm,0) = coalesce(e.comm,0)
)

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	1
7782	CLARK	MANAGER	7839	09-JUN-2006	2450		10	1
7839	KING	PRESIDENT		17-NOV-2006	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-2007	1300		10	1

Observe que a comparação não é entre a tabela EMP e a visualização V, mas entre a visualização sequencial E e a visualização sequencial V. A cardinalidade de cada linha é localizada e retornada como um atributo para essa linha. Você está comparando cada linha e sua contagem de ocorrências. Se você estiver tendo problemas para entender como a comparação funciona, execute as subconsultas de forma independente. A próxima etapa é localizar todas as linhas (incluindo CNT) na visualização sequencial E que não existem na visualização sequencial V. A comparação usa uma subconsulta correlacionada e NOT EXISTS. As junções determinarão quais linhas são as mesmas e o resultado será todas as linhas da exibição em linha E que não são as linhas retornadas pela junção. A consulta após UNION ALL faz o oposto; ele encontra todas as linhas na visualização inline V que não existem na visualização inline E:

```

select *
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
       v.sal,v.comm,v.deptno, count(*) as cnt
  from v
 group by empno,ename,job,mgr,hiredate,
           sal,comm,deptno
)
v
 where not exists (
select null
  from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
       e.sal,e.comm,e.deptno, count(*) as cnt
  from emp e
 group by empno,ename,job,mgr,hiredate,
           sal,comm,deptno
)
e
 where v.empno = e.empno
   and v.ename = e.ename
   and v.job = e.job
   and v.mgr = e.mgr
   and v.hiredate = e.hiredate
   and v.sal = e.sal
   and v.deptno = e.deptno
   and v.cnt = e.cnt
   and coalesce(v.comm,0) = coalesce(e.comm,0)
)

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
- 7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	2

Os resultados são então combinados por UNION ALL para produzir o conjunto de resultados final.



Ales Spetic e Jonathan Gennick dão uma solução alternativa em seu livro *Livro de receitas do Transact-SQL*(O'Reilly). Veja a seção “Comparando Dois Conjuntos para Igualdade” no Capítulo 2 de seu livro.

## 3.8 Identificando e evitando produtos cartesianos

### Problema

Você deseja retornar o nome de cada funcionário do departamento 10 junto com a localização do departamento. A consulta a seguir está retornando dados incorretos:

```
select e.ename, d.loc
  from emp e, dept d
 where e.deptno = 10
```

ENAME	LOC
CLARK	NEW YORK
CLARK	DALLAS
CLARK	CHICAGO
CLARK	BOSTON
KING	NEW YORK
KING	DALLAS
KING	CHICAGO
KING	BOSTON
MILLER	NEW YORK
MILLER	DALLAS
MILLER	CHICAGO
MILLER	BOSTON

O conjunto de resultados correto é o seguinte:

ENAME	LOC
CLARK	NEW YORK
KING	NEW YORK
MILLER	NEW YORK

Use uma junção entre as tabelas na cláusula FROM para retornar o conjunto de resultados correto:

```
1 select e.ename, d.loc
2 from emp e, dept d
3 where e.deptno = 10
4 and d.deptno = e.deptno
```

## Discussão

Vejamos os dados da tabela DEPT:

```
select * from dept
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Você pode ver que o departamento 10 está em Nova York e, portanto, pode saber que o retorno de funcionários com qualquer local diferente de Nova York está incorreto. O número de linhas retornadas pela consulta incorreta é o produto das cardinalidades das duas tabelas na cláusula FROM. Na consulta original, o filtro no EMP para o departamento 10 resultará em três linhas. Como não há filtro para DEPT, todas as quatro linhas de DEPT são retornadas. Três multiplicado por quatro é doze, então a consulta incorreta retorna doze linhas. Geralmente, para evitar um produto cartesiano, você aplicaria a regra  $n-1$  onde  $n$  representa o número de tabelas na cláusula FROM e  $n-1$  representa o número mínimo de junções necessárias para evitar um produto cartesiano. Dependendo de quais são as chaves e as colunas de junção em suas tabelas, você pode precisar de mais do que  $n-1$  se junta, mas  $n-1$  é um bom lugar para começar ao escrever consultas.



Quando usados corretamente, os produtos cartesianos podem ser úteis. Usos comuns de produtos cartesianos incluem transpor ou girar (e não girar) um conjunto de resultados, gerar uma sequência de valores e imitar um loop (embora os dois últimos também possam ser realizados usando um CTE recursivo).

## 3.9 Executando junções ao usar agregações

### Problema

Você deseja realizar uma agregação, mas sua consulta envolve várias tabelas. Você deseja garantir que as uniões não interrompam a agregação.

Por exemplo, você deseja encontrar a soma dos salários dos funcionários do departamento 10 junto com a soma de seus bônus. Alguns funcionários possuem mais de um bônus, e a junção entre a tabela EMP e a tabela EMP\_BONUS está fazendo com que valores incorretos sejam retornados pela função de agregação SUM. Para este problema, a tabela EMP\_BONUS contém os seguintes dados:

```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7934	17-MAR-2005	1
7934	15-FEB-2005	2
7839	15-FEB-2005	3
7782	15-FEB-2005	1

Agora, considere a seguinte consulta que retorna o salário e o bônus de todos os funcionários do departamento 10. A tabela BONUS.TYPE determina o valor do bônus. Um bônus do tipo 1 é de 10% do salário de um funcionário, o tipo 2 é de 20% e o tipo 3 é de 30%.

```
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3
             end as bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
   and e.deptno = 10
```

EMPNO	ENAME	SAL	DEPTNO	BONUS
7934	MILLER	1300	10	130
7934	MILLER	1300	10	260
7839	KING	5000	10	1500
7782	CLARK	2450	10	245

Até agora tudo bem. No entanto, as coisas dão errado quando você tenta uma junção à tabela EMP\_BONUS para somar os valores de bônus:

```
select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
```

```

        else .3
    end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
) x
group by deptno

```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	10050	2135

Enquanto o TOTAL\_BONUS está correto, o TOTAL\_SAL está incorreto. A soma de todos os salários no departamento 10 é 8750, como mostra a seguinte consulta:

```

select sum(sal) from emp where deptno=10

SUM(SAL)
-----
8750

```

Por que TOTAL\_SAL está incorreto? O motivo são as linhas duplicadas na coluna SAL criadas pela junção. Considere a seguinte consulta, que une as tabelas EMP e EMP\_BONUS:

```

select e.ename,
      e.sal
  from emp e, emp_bonus eb
 where e.empno = eb.empno
   and e.deptno = 10

ENAME          SAL
-----
CLARK          2450
KING           5000
MILLER         1300
MILLER         1300

```

Agora é fácil ver porque o valor de TOTAL\_SAL está incorreto: o salário de MILLER é contado duas vezes. O conjunto de resultados final que você realmente procura é:

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	2135

## Solução

Você deve ter cuidado ao computar agregações entre junções. Normalmente, quando duplicatas são retornadas devido a uma junção, você pode evitar erros de cálculo por funções agregadas de duas maneiras: você pode simplesmente usar a palavra-chave DISTINCT na chamada para a função agregada, de modo que apenas instâncias únicas de cada valor sejam usadas na computação; ou você

pode executar a agregação primeiro (em uma exibição em linha) antes da junção, evitando assim o cálculo incorreto pela função agregada porque a agregação já será computada antes mesmo de você ingressar, evitando assim o problema completamente. As soluções a seguir usam DISTINCT. A seção “Discussão” discutirá a técnica de usar uma visualização em linha para realizar a agregação antes da união.

## MySQL e PostgreSQL

Realize um somatório apenas dos vencimentos DISTINCT:

```

1 select deptno,
2 sum(distinct sal) as total_sal,
3 sum(bonus) as total_bonus
4 from (
5 select e.empno,
6 e.ename,
7 e.sal,
8 e.deptno,
9 e.sal*case when eb.type = 1 then .1
10when eb.type = 2 then .2
11else .3
12end as bonus
13from emp e, emp_bonus eb
14where e.empno = eb.empno
15and e.deptno = 10
16) x
17group by deptno

```

## DB2, Oracle e SQL Server

Essas plataformas suportam a solução anterior, mas também suportam uma solução alternativa usando a função de janela SUM OVER:

```

1 select distinct deptno,total_sal,total_bonus
2 from (
3 select e.empno,
4 e.ename,
5 sum(distinct e.sal) over
6 (partition by e.deptno) as total_sal,
7 e.deptno,
8 sum(e.sal*case when eb.type = 1 then .1
9 when eb.type = 2 then .2
10else .3 end) over
11(partition by deptno) as total_bonus
12from emp e, emp_bonus eb
13where e.empno = eb.empno
14and e.deptno = 10
15) x

```

### MySQL e PostgreSQL

A segunda consulta da seção “Problema” desta receita une a tabela EMP e a tabela EMP\_BONUS e retorna duas linhas para o funcionário MILLER, que é o que causa o erro na soma de EMP.SAL (o salário é somado duas vezes). A solução é simplesmente somar os valores distintos EMP.SAL que são retornados pela consulta. A consulta a seguir é uma solução alternativa — necessária se houver valores duplicados na coluna que você está somando. A soma de todos os salários no departamento 10 é calculada primeiro e essa linha é então unida à tabela EMP, que é então unida à tabela EMP\_BONUS.

A consulta a seguir funciona para todos os DBMSs:

```
select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                      when eb.type = 2 then .2
                      else .3 end) as total_bonus
  from emp e,
       emp_bonus eb,
       (
       select deptno, sum(sal) as total_sal
         from emp
        where deptno = 10
      group by deptno
      ) d
 where e.deptno = d.deptno
   and e.empno = eb.empno
  group by d.deptno,d.total_sal
```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	2135

### DB2, Oracle e SQL Server

Essa solução alternativa aproveita a função de janela SUM OVER. A consulta a seguir é retirada das linhas 3–14 em “Solução” e retorna o seguinte conjunto de resultados:

```
select e.empno,
       e.ename,
       sum(distinct e.sal) over
      (partition by e.deptno) as total_sal,
       e.deptno,
       sum(e.sal*case when eb.type = 1 then .1
                      when eb.type = 2 then .2
                      else .3 end) over
      (partition by deptno) as total_bonus
```

```
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
```

EMPNO	ENAME	TOTAL_SAL	DEPTNO	TOTAL_BONUS
7934	MILLER	8750	10	2135
7934	MILLER	8750	10	2135
7782	CLARK	8750	10	2135
7839	KING	8750	10	2135

A função de janelamento, SUM OVER, é chamada duas vezes, primeiro para calcular a soma dos salários distintos para a partição ou grupo definido. Nesse caso, a partição é DEPTNO 10 e a soma dos salários distintos para DEPTNO 10 é 8750. A próxima chamada para SUM OVER calcula a soma dos bônus para a mesma partição definida. O conjunto de resultados final é produzido tomando os valores distintos para TOTAL\_SAL, DEPTNO e TOTAL\_BONUS.

## 3.10 Executando junções externas ao usar agregações

### Problema

Comece com o mesmo problema da [Receita 3.9](#), mas modifique a tabela EMP\_BONUS de modo que a diferença neste caso seja que nem todos os funcionários do departamento 10 receberam bônus. Considere a tabela EMP\_BONUS e uma consulta para (aparentemente) encontrar a soma de todos os salários do departamento 10 e a soma de todos os bônus de todos os funcionários do departamento 10:

```
select * from emp_bonus

EMPNO RECEIVED          TYPE
----- -----
- 7934 17-MAR-2005      1
7934 15-FEB-2005        2

select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                   when eb.type = 2 then .2
                   else .3 end as bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
```

```

        and e.deptno = 10
    )
group by deptno

DEPTNO  TOTAL_SAL  TOTAL_BONUS
-----  -----
      10       2600        390

```

O resultado para TOTAL\_BONUS está correto, mas o valor retornado para TOTAL\_SAL não representa a soma de todos os salários do departamento 10. A consulta a seguir mostra porque TOTAL\_SAL está incorreto:

```

select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end as bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
   and e.deptno = 10

```

EMPNO	ENAME	SAL	DEPTNO	BONUS
7934	MILLER	1300	10	130
7934	MILLER	1300	10	260

Em vez de somar todos os salários do departamento 10, apenas o salário de MILLER é somado e é somado erroneamente duas vezes. Por fim, você gostaria de retornar o seguinte conjunto de resultados:

```

DEPTNO  TOTAL_SAL  TOTAL_BONUS
-----  -----
      10       8750        390

```

## Solução

A solução é semelhante à de [Receita 3.9](#), mas aqui você junta externamente a EMP\_BONUS para garantir que todos os funcionários do departamento 10 sejam incluídos.

### DB2, MySQL, PostgreSQL e SQL Server

Junte-se externamente a EMP\_BONUS e execute a soma apenas em salários distintos do departamento 10:

```

1 select deptno,
2 sum(distinct sal) as total_sal,
3 sum(bonus) as total_bonus
4 from (
5 select e.empno,
6 e.ename,

```

```

7 e.sal,
8 e.deptno,
9 e.sal*case when eb.type is null then 0
10when eb.type = 1 then .1
11when eb.type = 2 then .2
12else .3 end as bonus
13from emp e left outer join emp_bonus eb
14on (e.empno = eb.empno)
15where e.deptno = 10
16
17 group by deptno

```

Você também pode usar a função de janela SUM OVER:

```

1 select distinct deptno,total_sal,total_bonus
2 from (
3 select e.empno,
4 e.ename,
5 sum(distinct e.sal) over
6 (partition by e.deptno) as total_sal,
7 e.deptno,
8 sum(e.sal*case when eb.type is null then 0
9 when eb.type = 1 then .1
10when eb.type = 2 then .2
11else .3
12end) over
13(partition by deptno) as total_bonus
14from emp e left outer join emp_bonus eb
15on (e.empno = eb.empno)
16where e.deptno = 10
17) x

```

## Discussão

A segunda consulta da seção “Problema” desta receita une a tabela EMP e a tabela EMP\_BONUS e retorna apenas linhas para o funcionário MILLER, que é o que causa o erro na soma de EMP.SAL (os demais funcionários do DEPTNO 10 não possuem bônus, e seus salários não estão incluídos na soma). A solução é unir externamente a tabela EMP à tabela EMP\_BONUS, de modo que até mesmo funcionários sem bônus sejam incluídos no resultado. Se um funcionário não tiver um bônus, NULL será retornado para EMP\_BONUS.TYPE. É importante ter isso em mente, pois a instrução CASE foi modificada e é um pouco diferente de [Receita 3.9](#). Se EMP\_BONUS.TYPE for NULL, a expressão CASE retorna zero, o que não tem efeito na soma.

A consulta a seguir é uma solução alternativa. A soma de todos os salários do departamento 10 é calculado primeiro e, em seguida, associado à tabela EMP, que é então associada à tabela EMP\_BONUS (evitando assim a junção externa). A consulta a seguir funciona para todos os DBMSs:

```

select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                      when eb.type = 2 then .2
                      else .3 end) as total_bonus
  from emp e,
       emp_bonus eb,
       (
select deptno, sum(sal) as total_sal
   from emp
  where deptno = 10
 group by deptno
      ) d
 where e.deptno = d.deptno
   and e.empno = eb.empno
 group by d.deptno,d.total_sal

DEPTNO  TOTAL_SAL  TOTAL_BONUS
-----  -----  -----
 - 10      8750        390

```

## 3.11 Retornando dados ausentes de várias tabelas

### Problema

Você deseja retornar dados ausentes de várias tabelas simultaneamente. Retornar linhas da tabela DEPT que não existem na tabela EMP (quaisquer departamentos que não tenham funcionários) requer uma junção externa. Considere a seguinte consulta, que retorna todos os DEPTNOs e DNAMEs do DEPT junto com os nomes de todos os funcionários de cada departamento (se houver um funcionário em um determinado departamento):

```

select d.deptno,d.dname,e.ename
  from dept d left outer join emp e
    on (d.deptno=e.deptno)

```

DEPTNO	DNAME	ENAME
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	WARD
20	RESEARCH	JONES
30	SALES	MARTIN
30	SALES	BLAKE
10	ACCOUNTING	CLARK
20	RESEARCH	SCOTT
10	ACCOUNTING	KING
30	SALES	TURNER
20	RESEARCH	ADAMS
30	SALES	JAMES

20 RESEARCH	FORD
10 ACCOUNTING	MILLER
40 OPERATIONS	

A última linha, o departamento OPERATIONS, é retornada apesar desse departamento não ter nenhum funcionário, porque a tabela EMP foi unida externamente à tabela DEPT. Agora, suponha que houvesse um funcionário sem departamento. Como você retornaria o conjunto de resultados anterior junto com uma linha para o funcionário sem departamento? Em outras palavras, você deseja associar externamente à tabela EMP e à tabela DEPT e na mesma consulta. Depois de criar o novo funcionário, uma primeira tentativa pode ser assim:

```
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
select 1111,'YODA','JEDI',null,hiredate,sal,comm,null
from emp
where ename = 'KING'
```

```
select d.deptno,d.dname,e.ename
from dept d right outer join emp e
on (d.deptno=e.deptno)
```

DEPTNO	DNAME	ENAME
10	ACCOUNTING	MILLER
10	ACCOUNTING	KING
10	ACCOUNTING	CLARK
20	RESEARCH	FORD
20	RESEARCH	ADAMS
20	RESEARCH	SCOTT
20	RESEARCH	JONES
20	RESEARCH	SMITH
30	SALES	JAMES
30	SALES	TURNER
30	SALES	BLAKE
30	SALES	MARTIN
30	SALES	WARD
30	SALES	ALLEN
		YODA

Essa junção externa consegue retornar o novo funcionário, mas perdeu as OPERATIONS do departamento do conjunto de resultados original. O conjunto de resultados final deve retornar uma linha para YODA, bem como OPERATIONS, como o seguinte:

DEPTNO	DNAME	ENAME
10	ACCOUNTING	CLARK
10	ACCOUNTING	KING
10	ACCOUNTING	MILLER
20	RESEARCH	ADAMS
20	RESEARCH	FORD
20	RESEARCH	JONES
20	RESEARCH	SCOTT
20	RESEARCH	SMITH
		YODA

30 SALES	ALLEN
30 SALES	BLAKE
30 SALES	JAMES
30 SALES	MARTIN
30 SALES	TURNER
30 SALES	WARD
40 OPERATIONS	YODA

## Solução

Use uma junção externa completa para retornar dados ausentes de ambas as tabelas com base em um valor comum.

### DB2, MySQL, PostgreSQL e SQL Server

Use o comando explícito FULL OUTER JOIN para retornar as linhas ausentes de ambas as tabelas junto com as linhas correspondentes:

```
1 select d.deptno,d.dname,e.ename
2 from dept d full outer join emp e
3 on (d.deptno=e.deptno)
```

Como alternativa, como o MySQL ainda não possui um FULL OUTER JOIN, UNION os resultados das duas junções externas diferentes:

```
1 select d.deptno,d.dname,e.ename
2 from dept d right outer join emp e
3 on (d.deptno=e.deptno)
4 union
5 select d.deptno,d.dname,e.ename
6 from dept d left outer join emp e
7 on (d.deptno=e.deptno)
```

## Oráculo

Os usuários do Oracle ainda podem usar qualquer uma das soluções anteriores. Como alternativa, você pode usar a sintaxe de junção externa proprietária da Oracle:

```
1 select d.deptno,d.dname,e.ename
2 from dept d, emp e
3 where d.deptno = e.deptno(+)
4 union
5 select d.deptno,d.dname,e.ename
6 from dept d, emp e
7 where d.deptno(+) = e.deptno
```

## Discussão

A junção externa completa é simplesmente a combinação de junções externas em ambas as tabelas. Para ver como uma junção externa completa funciona “nos bastidores”, basta executar cada junção externa e, em seguida, unir os resultados.

A consulta a seguir retorna linhas da tabela DEPT e todas as linhas correspondentes da tabela EMP (se houver):

```
select d.deptno,d.dname,e.ename
  from dept d left outer join emp e
    on (d.deptno = e.deptno)
```

DEPTNO	DNAME	ENAME
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	WARD
20	RESEARCH	JONES
30	SALES	MARTIN
30	SALES	BLAKE
10	ACCOUNTING	CLARK
20	RESEARCH	SCOTT
10	ACCOUNTING	KING
30	SALES	TURNER
20	RESEARCH	ADAMS
30	SALES	JAMES
20	RESEARCH	FORD
10	ACCOUNTING	MILLER
40	OPERATIONS	

Esta próxima consulta retorna linhas da tabela EMP e quaisquer linhas correspondentes da tabela DEPT (se houver):

```
select d.deptno,d.dname,e.ename
  from dept d right outer join emp e
    on (d.deptno = e.deptno)
```

DEPTNO	DNAME	ENAME
10	ACCOUNTING	MILLER
10	ACCOUNTING	KING
10	ACCOUNTING	CLARK
20	RESEARCH	FORD
20	RESEARCH	ADAMS
20	RESEARCH	SCOTT
20	RESEARCH	JONES
20	RESEARCH	SMITH
30	SALES	JAMES
30	SALES	TURNER
30	SALES	BLAKE
30	SALES	MARTIN
30	SALES	WARD
30	SALES	ALLEN
		YODA

Os resultados dessas duas consultas são unidos para fornecer o conjunto de resultados final.

## 3.12 Usando NULLs em operações e comparações

### Problema

NULL nunca é igual ou não igual a qualquer valor, nem mesmo a si mesmo, mas você deseja avaliar os valores retornados por uma coluna anulável como avaliaria valores reais. Por exemplo, você deseja localizar todos os funcionários no EMP cuja comissão (COMM) é menor que a comissão do funcionário WARD. Funcionários com comissão NULL também devem ser incluídos.

### Solução

Use uma função como COALESCE para transformar o valor NULL em um valor real que pode ser usado na avaliação padrão:

```

1 select ename,comm
2   from emp
3 where coalesce(comm,0) < ( select comm
4                               from emp
5                             where ename = 'WARD' )

```

### Discussão

A função COALESCE retornará o primeiro valor não NULL da lista de valores passados para ela. Quando um valor NULL é encontrado, ele é substituído por zero, que é então comparado com os WARDs da comissão. Isso pode ser visto colocando a função COALESCE na lista SELECT:

```

seleccione ename,comm,coalesce(comm,0)
  de emp
onde coalesce(comm,0) < (seleccione comm
                           de emp
                           onde ename = 'WARD' )

```

ENAME	COMM	COALESCE(COMM,0)
SMITH		0
ALLEN	300	300
JONES		0
BLAKE		0
CLARK		0
SCOTT		0
KING		0
TURNER	0	0
ADAMS		0
JAMES		0
FORD		0
MILLER		0

### 3.13 Resumindo

As junções são um aspecto crucial da consulta de bancos de dados — será a norma que você precise juntar duas ou mais tabelas para encontrar o que está procurando. Dominar as diferentes combinações e categorias de junções abordadas neste capítulo o preparará para o sucesso.

## CAPÍTULO 4

# Inserindo, Atualizando e Excluindo

Os últimos capítulos se concentraram em técnicas básicas de consulta, todas centradas na tarefa de obter dados de um banco de dados. Este capítulo vira a mesa e se concentra nas três áreas de tópicos a seguir:

- Inserindo novos registros em seu banco de dados
- Atualizando registros existentes
- Excluindo registros que você não deseja mais

Para facilitar a sua localização quando você precisar delas, as receitas neste capítulo foram agrupadas por tópico: todas as receitas de inserção vêm primeiro, seguidas pelas receitas de atualização e, finalmente, as receitas para exclusão de dados.

A inserção geralmente é uma tarefa simples. Começa com o simples problema de inserir uma única linha. Muitas vezes, no entanto, é mais eficiente usar uma abordagem baseada em conjunto para criar novas linhas. Para isso, você também encontrará técnicas para inserir várias linhas por vez.

Da mesma forma, atualizar e deletar começam como tarefas simples. Você pode atualizar um registro e excluir um registro. Mas você também pode atualizar conjuntos inteiros de registros de uma só vez e de maneiras muito poderosas. E há muitas maneiras úteis de excluir registros. Por exemplo, você pode excluir linhas em uma tabela dependendo se elas existem em outra tabela.

O SQL ainda tem uma maneira, uma adição relativamente nova ao padrão, permitindo que você insira, atualize e exclua tudo de uma vez. Isso pode não parecer muito útil agora, mas a instrução MERGE representa uma maneira poderosa de sincronizar uma tabela de banco de dados com uma fonte externa de dados (como um feed de arquivo simples de um sistema remoto). Confira [Receita 4.11](#) neste capítulo para detalhes.

## 4.1 Inserindo um Novo Registro

### Problema

Você deseja inserir um novo registro em uma tabela. Por exemplo, você deseja inserir um novo registro na tabela DEPT. O valor para DEPTNO deve ser 50, DNAME deve ser PROGRAMMING e LOC deve ser BALTIMORE.

### Solução

Use a instrução INSERT com a cláusula VALUES para inserir uma linha por vez:

```
insert into dept (deptno,dname,loc)
values (50,'PROGRAMMING','BALTIMORE')
```

Para DB2, SQL Server, PostgreSQL e MySQL, você tem a opção de inserir uma linha por vez ou várias linhas por vez, incluindo várias listas VALUES:

```
/* inserção de várias linhas */
insert into dept (deptno,dname,loc)
values (1,'A','B'),
       (2,'B','C')
```

### Discussão

A instrução INSERT permite criar novas linhas nas tabelas do banco de dados. A sintaxe para inserir uma única linha é consistente em todas as marcas de banco de dados.

Como atalho, você pode omitir a lista de colunas em uma instrução INSERT:

```
insert into dept
values (50,'PROGRAMMING','BALTIMORE')
```

No entanto, se você não listar suas colunas de destino, deverá inserir em *todos* das colunas da tabela e fique atento à ordem dos valores na lista VALUES; você deve fornecer valores na mesma ordem em que o banco de dados exibe as colunas em resposta a uma consulta SELECT\*. De qualquer forma, você deve estar atento às restrições de coluna porque, se não inserir em todas as colunas, criará uma linha em que alguns valores são nulos. Isso pode causar um erro se houver colunas restritas para não aceitar nulos.

## 4.2 Inserindo Valores Padrão

### Problema

Uma tabela pode ser definida para obter valores padrão para colunas específicas. Você deseja inserir uma linha de valores padrão sem precisar especificar esses valores.

Considere a seguinte tabela:

```
create table D (id integer default 0)
```

Você deseja inserir zero sem especificar explicitamente zero na lista de valores de uma instrução INSERT. Você deseja inserir explicitamente o padrão, qualquer que seja o padrão.

## Solução

Todas as marcas suportam o uso da palavra-chave DEFAULT como forma de especificar explicitamente o valor padrão para uma coluna. Algumas marcas fornecem maneiras adicionais de resolver o problema.

O exemplo a seguir ilustra o uso da palavra-chave DEFAULT:

```
insert into D values (default)
```

Você também pode especificar explicitamente o nome da coluna, o que precisará fazer sempre que não estiver inserindo em todas as colunas de uma tabela:

```
insert into D (id) values (default)
```

Banco de dados Oracle8i e versões anteriores não suportam a palavra-chave DEFAULT. Antes do Database Oracle9i , não havia como inserir explicitamente um valor de coluna padrão.

O MySQL permite que você especifique uma lista de valores vazia se todas as colunas tiverem um valor padrão definido:

```
insert into D values ()
```

Nesse caso, todas as colunas serão definidas com seus valores padrão.

PostgreSQL e SQL Server suportam uma cláusula DEFAULT VALUES:

```
insert into D default values
```

A cláusula DEFAULT VALUES faz com que todas as colunas assumam seus valores padrão.

## Discussão

A palavra-chave DEFAULT na lista de valores inserirá o valor que foi especificado como padrão para uma determinada coluna durante a criação da tabela. A palavra-chave está disponível para todos os DBMSs.

Os usuários do MySQL, PostgreSQL e SQL Server têm outra opção disponível se todas as colunas da tabela forem definidas com um valor padrão (como a tabela D neste caso). Você pode usar uma lista vazia VALUES (MySQL) ou especificar a cláusula DEFAULT VALUES (PostgreSQL e SQL Server) para criar uma nova linha com todos os valores padrão; caso contrário, você precisa especificar DEFAULT para cada coluna na tabela.

Para tabelas com uma mistura de colunas padrão e não padrão, inserir valores padrão para uma coluna é tão fácil quanto excluir a coluna da lista de inserção; você não precisa usar a palavra-chave DEFAULT.

Digamos que a tabela D tenha uma coluna adicional que não foi definida com um valor padrão:

```
create table D (id integer default 0, foo varchar(10))
```

Você pode inserir um padrão para ID listando apenas FOO na lista de inserção:

```
insert into D (name) values ('Bar')
```

Essa instrução resultará em uma linha na qual ID é 0 e FOO é BAR. ID assume seu valor padrão porque nenhum outro valor é especificado.

## 4.3 Substituindo um valor padrão por NULL

### Problema

Você está inserindo em uma coluna com um valor padrão e deseja substituir esse valor padrão definindo a coluna como NULL. Considere a seguinte tabela:

```
create table D (id integer default 0, foo VARCHAR(10))
```

Você deseja inserir uma linha com um valor NULL para ID.

### Solução

Você pode especificar explicitamente NULL em sua lista de valores:

```
insert into d (id, foo) values (null, 'Brighten')
```

### Discussão

Nem todo mundo percebe que você pode especificar explicitamente NULL na lista de valores de uma instrução INSERT. Normalmente, quando você não deseja especificar um valor para uma coluna, você deixa essa coluna fora de suas listas de colunas e valores:

```
insert into d (foo) values ('Brighten')
```

Aqui, nenhum valor para ID é especificado. Muitos esperariam que a coluna assumisse o valor nulo, mas, infelizmente, um valor padrão foi especificado no momento da criação da tabela, portanto, o resultado do INSERT anterior é que ID assume o valor zero (o padrão). Ao especificar NULL como o valor para uma coluna, você pode definir a coluna como NULL apesar de qualquer valor padrão (exceto onde uma restrição foi aplicada especificamente para evitar NULLs).

## 4.4 Copiando linhas de uma tabela para outra

### Problema

Você deseja copiar linhas de uma tabela para outra usando uma consulta. A consulta pode ser complexa ou simples, mas no final você deseja que o resultado seja inserido em outra tabela.

Por exemplo, você deseja copiar linhas da tabela DEPT para a tabela DEPT\_EAST. A tabela DEPT\_EAST já foi criada com a mesma estrutura (mesmas colunas e tipos de dados) que DEPT e atualmente está vazia.

## Solução

Use a instrução INSERT seguida de uma consulta para produzir as linhas desejadas:

```
1 insert into dept_east (deptno,dname,loc)
2 select deptno,dname,loc
3 from dept
4 where loc in ( 'NEW YORK','BOSTON')
```

## Discussão

Simplesmente siga a instrução INSERT com uma consulta que retorne as linhas desejadas. Se você deseja copiar todas as linhas da tabela de origem, exclua a cláusula WHERE da consulta. Como uma inserção regular, você não precisa especificar explicitamente em quais colunas está inserindo. Mas se você não especificar suas colunas de destino, deverá inserir dados em *todos* das colunas da tabela, e você deve ficar atento à ordem dos valores na lista SELECT, conforme descrito anteriormente em [Receita 4.1](#).

## 4.5 Copiando uma definição de tabela

### Problema

Você deseja criar uma nova tabela com o mesmo conjunto de colunas de uma tabela existente. Por exemplo, você deseja criar uma cópia da tabela DEPT e chamá-la de DEPT\_2. Você não deseja copiar as linhas, apenas a estrutura de colunas da tabela.

## Solução

### DB2

Use a cláusula LIKE com o comando CREATE TABLE:

```
create table dept_2 like dept
```

### Oracle, MySQL e PostgreSQL

Use o comando CREATE TABLE com uma subconsulta que não retorne nenhuma linha:

```
1 create table dept_2
2 as
3 select *
4 from dept
5 where 1 = 0
```

Use a cláusula INTO com uma subconsulta que não retorne nenhuma linha:

```
1 select *  
2 into dept_2  
3 from dept  
4 where 1 = 0
```

## **Discussão**

### **DB2**

O comando CREATE TABLE...LIKE do DB2 permite que você use facilmente uma tabela como padrão para criar outra. Simplesmente especifique o nome da sua tabela de padrões seguindo a palavra-chave LIKE.

### **Oracle, MySQL e PostgreSQL**

Ao usar Create Table As Select (CTAS), todas as linhas de sua consulta serão usadas para preencher a nova tabela que você está criando, a menos que você especifique uma condição falsa na cláusula WHERE. Na solução fornecida, a expressão “1 = 0” na cláusula WHERE da consulta faz com que nenhuma linha seja retornada. Assim, o resultado da instrução CTAS é uma tabela vazia com base nas colunas da cláusula SELECT da consulta.

### **SQL Server**

Ao usar INTO para copiar uma tabela, todas as linhas de sua consulta serão usadas para preencher a nova tabela que você está criando, a menos que você especifique uma condição falsa na cláusula WHERE de sua consulta. Na solução fornecida, a expressão “1 = 0” no predicado da consulta faz com que nenhuma linha seja retornada. O resultado é uma tabela vazia com base nas colunas da cláusula SELECT da consulta.

## **4.6 Inserindo em várias tabelas ao mesmo tempo**

### **Problema**

Você deseja pegar as linhas retornadas por uma consulta e inserir essas linhas em várias tabelas de destino. Por exemplo, você deseja inserir linhas de DEPT nas tabelas DEPT\_EAST, DEPT\_WEST e DEPT\_MID. Todas as três tabelas têm a mesma estrutura (mesmas colunas e tipos de dados) que DEPT e estão vazias no momento.

### **Solução**

A solução é inserir o resultado de uma consulta nas tabelas de destino. A diferença de [Receita 4.4](#) é que, para esse problema, você tem várias tabelas de destino.

## Oracle

Use a instrução INSERT ALL ou INSERT FIRST. Ambos compartilham a mesma sintaxe, exceto pela escolha entre as palavras-chave ALL e FIRST. A instrução a seguir usa INSERT ALL para fazer com que todas as tabelas de destino possíveis sejam consideradas:

```

1  insert all
2  when loc in ('NEW YORK','BOSTON') then
3  into dept_east (deptno,dname,loc) values (deptno,dname,loc)
4  when loc = 'CHICAGO' then
5  into dept_mid (deptno,dname,loc) values (deptno,dname,loc)
6  else
7  into dept_west (deptno,dname,loc) values (deptno,dname,loc)
8  select deptno,dname,loc
9  from dept

```

## DB2

Insert em uma visão inline que executa um UNION ALL nas tabelas a serem inseridas. Você também deve certificar-se de colocar restrições nas tabelas que garantirão que cada linha vá para a tabela correta:

```

create table dept_east
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc in ('NEW YORK','BOSTON')))

create table dept_mid
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc = 'CHICAGO'))

create table dept_west
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc = 'DALLAS'))

1  insert into (
2  select * from dept_west union all
3  select * from dept_east union all
4  select * from dept_mid
5  ) select * from dept

```

## MySQL, PostgreSQL e SQL Server

No momento em que este livro foi escrito, esses fornecedores não oferecem suporte a inserções de várias tabelas.

### **Oracle**

A inserção multitable do Oracle usa cláusulas WHEN-THEN-ELSE para avaliar as linhas do SELECT aninhado e inseri-las de acordo. No exemplo desta receita, INSERT ALL e INSERT FIRST produziriam o mesmo resultado, mas há uma diferença entre os dois. INSERT FIRST sairá da avaliação WHEN-THEN-ELSE assim que encontrar uma condição avaliada como verdadeira; INSERT ALL avaliará todas as condições mesmo se os testes anteriores forem verdadeiros. Assim, você pode usar INSERT ALL para inserir a mesma linha em mais de uma tabela.

### **DB2**

Minha solução DB2 é meio que um hack. Requer que as tabelas a serem inseridas tenham restrições definidas para garantir que cada linha avaliada da subconsulta vá para a tabela correta. A técnica é inserir em uma visão que é definida como UNION ALL das tabelas. Se as restrições de verificação não forem únicas entre as tabelas no INSERT (ou seja, várias tabelas têm a mesma restrição de verificação), a instrução INSERT não saberá onde colocar as linhas e falhará.

### **MySQL, PostgreSQL e SQL Server**

No momento em que este livro foi escrito, apenas o Oracle e o DB2 fornecem mecanismos para inserir linhas retornadas por uma consulta em uma ou mais das várias tabelas dentro da mesma instrução.

## **4.7 Bloqueando Inserções em Certas Colunas**

### **Problema**

Você deseja impedir que usuários ou um aplicativo de software errante insiram valores em determinadas colunas da tabela. Por exemplo, você deseja permitir que um programa seja inserido no EMP, mas apenas nas colunas EMPNO, ENAME e JOB.

### **Solução**

Crie uma exibição na tabela expondo apenas as colunas que deseja expor. Em seguida, force todas as inserções a passarem por essa exibição.

Por exemplo, para criar uma exibição expondo as três colunas no EMP:

```
create view new_emps as
select empno, ename, job
from emp
```

Conceda acesso a esta exibição para os usuários e programas autorizados a preencher apenas os três campos na exibição. Não conceda a esses usuários acesso de inserção à tabela EMP. Os usuários podem criar novos registros EMP inserindo na exibição NEW\_EMPS, mas não poderão fornecer valores para colunas diferentes dos três especificados na definição da exibição.

## Discussão

Quando você insere em uma exibição simples, como na solução, seu servidor de banco de dados traduzirá essa inserção na tabela subjacente. Por exemplo, a seguinte inserção:

```
insert into new_emps
  (empno ename, job)
  values (1, 'Jonathan', 'Editor')
```

será traduzido nos bastidores para:

```
insert into emp
  (empno ename, job)
  values (1, 'Jonathan', 'Editor')
```

Também é possível, mas talvez menos útil, inserir em uma visão inline (atualmente suportada apenas pelo Oracle):

```
insert into
  (select empno, ename, job
   from emp)
  values (1, 'Jonathan', 'Editor')
```

A inserção de visualização é um tópico complexo. As regras tornam-se complicadas muito rapidamente para todos, exceto para as visões mais simples. Se você planeja fazer uso da capacidade de inserir em exibições, é imperativo que você consulte e compreenda totalmente a documentação do fornecedor sobre o assunto.

## 4.8 Modificando registros em uma tabela

### Problema

Você deseja modificar valores para algumas ou todas as linhas em uma tabela. Por exemplo, você pode querer aumentar os salários de todos no departamento 20 em 10%. O conjunto de resultados a seguir mostra DEPTNO, ENAME e SAL para os funcionários desse departamento:

```
select deptno,ename,sal
  from emp
 where deptno = 20
 order by 1,3
```

DEPTNO	ENAME	SAL
20	SMITH	800

20 ADAMS	1100
20 JONES	2975
20 SCOTT	3000
20 FORD	3000

Você deseja aumentar todos os valores de SAL em 10%.

## Solução

Use a instrução UPDATE para modificar as linhas existentes em uma tabela de banco de dados. Por exemplo:

```
1 update emp
2 set sal = sal*1.10
3 where deptno = 20
```

## Discussão

Use a instrução UPDATE juntamente com uma cláusula WHERE para especificar quais linhas atualizar; se você excluir uma cláusula WHERE, todas as linhas serão atualizadas. A expressão SAL\*1.10 nesta solução retorna o salário acrescido de 10%.

Ao se preparar para uma atualização em massa, você pode querer visualizar os resultados. Você pode fazer isso emitindo uma instrução SELECT que inclua as expressões que você planeja colocar em suas cláusulas SET. O SELECT a seguir mostra o resultado de um aumento salarial de 10%:

```
select deptno,
       ename,
       sal      as orig_sal,
       sal*.10 as amt_to_add,
       sal*1.10 as new_sal
  from emp
 where deptno=20
 order by 1,5
```

DEPTNO	ENAME	ORIG_SAL	AMT_TO_ADD	NEW_SAL
20	SMITH	800	80	880
20	ADAMS	1100	110	1210
20	JONES	2975	298	3273
20	SCOTT	3000	300	3300
20	FORD	3000	300	3300

O aumento salarial é dividido em duas colunas: uma para mostrar o aumento sobre o salário antigo e a outra para mostrar o novo salário.

## 4.9 Atualizando quando existem linhas correspondentes

### Problema

Você deseja atualizar linhas em uma tabela quando existem linhas correspondentes em outra. Por exemplo, se um funcionário aparece na tabela EMP\_BONUS, você deseja aumentar o salário desse funcionário (na tabela EMP) em 20%. O seguinte conjunto de resultados representa os dados atualmente na tabela EMP\_BONUS:

```
select empno, ename
from emp_bonus

EMPNO ENAME
-----
7369 SMITH
7900 JAMES
7934 MILLER
```

### Solução

Use uma subconsulta na cláusula UPDATE WHERE de sua instrução para localizar funcionários na tabela EMP que também estão na tabela EMP\_BONUS. Seu UPDATE então atuará apenas nessas linhas, permitindo que você aumente o salário deles em 20%:

```
1 update emp
2 set sal=sal*1.20
3 where empno in ( select empno from emp_bonus)
```

### Discussão

Os resultados da subconsulta representam as linhas que serão atualizadas na tabela EMP. O predicado IN testa os valores de EMPNO da tabela EMP para ver se eles estão na lista de valores EMPNO retornados pela subconsulta. Quando estiverem, os valores SAL correspondentes são atualizados.

Como alternativa, você pode usar EXISTS em vez de IN:

```
update emp
  set sal = sal*1.20
where exists ( select null
                 from emp_bonus
               where emp.empno=emp_bonus.empno )
```

Você pode se surpreender ao ver NULL na lista SELECT da subconsulta EXISTS. Não tema, pois NULL não tem um efeito adverso na atualização. Indiscutivelmente, aumenta a legibilidade, pois reforça o fato de que, ao contrário da solução usando uma subconsulta com um operador IN, o que conduzirá a atualização (ou seja, quais linhas serão atualizadas) será controlado pela cláusula WHERE da subconsulta, não os valores retornado como resultado da lista SELECT da subconsulta.

## 4.10 Atualizando com valores de outra tabela

### Problema

Você deseja atualizar linhas em uma tabela usando valores de outra. Por exemplo, você tem uma tabela chamada NEW\_SAL, que contém os novos salários de determinados funcionários. O conteúdo da tabela NEW\_SAL é o seguinte:

```
select *
from new_sal
```

DEPTNO	SAL
10	4000

A coluna DEPTNO é a chave primária da tabela NEW\_SAL. Você deseja atualizar os salários e comissões de determinados funcionários na tabela EMP usando a tabela de valores NEW\_SAL se houver uma correspondência entre EMP.DEPTNO e NEW\_SAL.DEPTNO, atualizar EMP.SAL para NEW\_SAL.SAL e atualizar EMP.COMM para 50% de NOVO\_SAL.SAL. As linhas no EMP são as seguintes:

```
select deptno,ename,sal,comm
from emp
order by 1
```

DEPTNO	ENAME	SAL	COMM
10	CLARK	2450	
10	KING	5000	
10	MILLER	1300	
20	SMITH	800	
20	ADAMS	1100	
20	FORD	3000	
20	SCOTT	3000	
20	JONES	2975	
30	ALLEN	1600	300
30	BLAKE	2850	
30	MARTIN	1250	1400
30	JAMES	950	
30	TURNER	1500	0
30	WARD	1250	500

### Solução

Use uma junção entre NEW\_SAL e EMP para localizar e retornar os novos valores COMM para a instrução UPDATE. É bastante comum que atualizações como esta sejam realizadas por meio de subconsulta correlacionada ou, alternativamente, usando um CTE. Outra técnica envolve a criação de uma visão (tradicional ou embutida, dependendo do que seu banco de dados suporta) e então atualizar essa visão.

**DB2**

Use uma subconsulta correlacionada para definir novos valores SAL e COMM no EMP. Use também uma subconsulta correlacionada para identificar quais linhas do EMP devem ser atualizadas:

```

1 update emp e set (e.sal,e.comm) = (select ns.sal, ns.sal/2
2 from new_sal ns
3 where ns.deptno=e.deptno)
4 where exists ( select *
5 from new_sal ns
6 where ns.deptno = e.deptno)
```

**MySQL**

Inclua EMP e NEW\_SAL na cláusula UPDATE da instrução UPDATE e junte-se à cláusula WHERE:

```

1 update emp e, new_sal ns
2 set e.sal=ns.sal,
3 e.comm=ns.sal/2
4 where e.deptno=ns.deptno
```

**Oracle**

O método para a solução DB2 funcionará para Oracle, mas, como alternativa, você pode atualizar uma visualização sequencial:

```

1 update (
2 select e.sal as emp_sal, e.comm as emp_comm,
3        ns.sal as ns_sal, ns.sal/2 as ns_comm
4   from emp e, new_sal ns
5  where e.deptno = ns.deptno
6 ) set emp_sal = ns_sal, emp_comm = ns_comm
```

**PostgreSQL**

O método usado para a solução DB2 funcionará para PostgreSQL, mas você também pode (muito convenientemente) juntar diretamente na instrução UPDATE:

```

1 update emp
2   set sal = ns.sal,
3       comm = ns.sal/2
4   from new_sal ns
5  where ns.deptno = emp.deptno
```

**SQL Server**

O método usado para a solução DB2 funcionará para SQL Server, mas como alternativa você pode (de forma semelhante à solução PostgreSQL) juntar diretamente na instrução UPDATE:

```
1 update e
2   set e.sal = ns.sal,
3       e.comm = ns.sal/2
4   from emp e,
5        new_sal ns
6  where ns.deptno = e.deptno
```

## Discussão

Antes de discutir as diferentes soluções, vale mencionar algo importante sobre as atualizações que usam consultas para fornecer novos valores. Uma cláusula WHERE na subconsulta de uma atualização correlacionada não é igual à cláusula WHERE da tabela que está sendo atualizada. Se você observar a instrução UPDATE na seção “Problema”, a junção em DEPTNO entre EMP e NEW\_SAL é feita e retorna linhas para a cláusula SET da instrução UPDATE. Para funcionários em DEPTNO 10, valores válidos são retornados porque há um DEPTNO correspondente na tabela NEW\_SAL. Mas e os funcionários dos outros departamentos? NEW\_SAL não possui nenhum outro departamento, então o SAL e COMM para funcionários nos DEPTNOs 20 e 30 são definidos como NULL. A menos que você esteja fazendo isso via LIMIT ou TOP ou qualquer outro mecanismo que seu fornecedor forneça para limitar o número de linhas retornadas em um conjunto de resultados, a única maneira de restringir as linhas de uma tabela em SQL é usar uma cláusula WHERE. Para executar corretamente este UPDATE, use uma cláusula WHERE na tabela que está sendo atualizada junto com uma cláusula WHERE na subconsulta correlacionada.

### DB2

Para garantir que você não atualize todas as linhas da tabela EMP, lembre-se de incluir uma subconsulta correlacionada na cláusula WHERE de UPDATE. Realizar a junção (a subconsulta correlacionada) na cláusula SET não é suficiente. Ao usar uma cláusula WHERE em UPDATE, você garante que apenas as linhas em EMP que correspondem em DEPTNO à tabela NEW\_SAL sejam atualizadas. Isso vale para todos os RDBMSs.

### Oracle

Na solução Oracle usando a exibição de junção de atualização, você está usando junções equivalentes para determinar quais linhas serão atualizadas. Você pode confirmar quais linhas estão sendo atualizadas executando a consulta de forma independente. Para poder usar esse tipo de UPDATE com sucesso, você deve primeiro entender o conceito de preservação de chave. A coluna DEPTNO da tabela NEW\_SAL é a chave primária dessa tabela; assim, seus valores são únicos dentro da tabela. Ao unir entre EMP e NEW\_SAL, no entanto, NEW\_SAL.DEPTNO não é exclusivo no conjunto de resultados, conforme mostrado aqui:

```
select e.empno, e.deptno e_dept, ns.sal, ns.deptno ns_deptno
  from emp e, new_sal ns
 where e.deptno = ns.deptno
```

EMPNO	E_DEPT	SAL	NS_DEPTNO	90
7782	10	4000	10	
7839	10	4000	10	
7934	10	4000	10	

Para permitir que o Oracle atualize essa junção, uma das tabelas deve ser preservada por chave, o que significa que, se seus valores não forem exclusivos no conjunto de resultados, devem ser pelo menos exclusivos na tabela de onde vem. Nesse caso, NEW\_SAL possui uma chave primária em DEPTNO, o que a torna única na tabela. Por ser único em sua tabela, ele pode aparecer várias vezes no conjunto de resultados e ainda será considerado chave preservada, permitindo assim que a atualização seja concluída com sucesso.

### PostgreSQL, SQL Server e MySQL

A sintaxe é um pouco diferente entre essas plataformas, mas a técnica é a mesma. Poder juntar diretamente na instrução UPDATE é extremamente conveniente. Como você especifica qual tabela atualizar (a tabela listada após a palavra-chave UPDATE), não há confusão quanto a quais linhas da tabela são modificadas. Além disso, como você está usando junções na atualização (já que há uma cláusula explícita WHERE), você pode evitar algumas das armadilhas ao codificar atualizações de subconsultas correlacionadas; em particular, se você perdeu uma junção aqui, seria óbvio que você teria um problema.

## 4.11 Mesclando Registros

### Problema

Você deseja inserir, atualizar ou excluir registros condicionalmente em uma tabela, dependendo da existência ou não de registros correspondentes. (Se existir um registro, atualize; caso contrário, insira; se depois de atualizar uma linha não atender a uma determinada condição, exclua-a.) Por exemplo, você deseja modificar a tabela EMP\_COMMISIION de forma que:

- Se algum funcionário em EMP\_COMMISIION também existir na tabela EMP, atualize sua comissão (COMM) para 1000.
- Para todos os funcionários que potencialmente terão seu COMM atualizado para 1000, se seu SAL for menor que 2000, exclua-os (eles não devem existir em EMP\_[.keep-together] COMMISIION).
- Caso contrário, insira os valores EMPNO, ENAME e DEPTNO da tabela EMP na tabela EMP\_COMMISIION.

Essencialmente, você deseja executar um UPDATE ou um INSERT dependendo se uma determinada linha de EMP tem uma correspondência em EMP\_COMMISIION. Então você deseja executar um DELETE se o resultado de um UPDATE causar uma comissão muito alta.

As seguintes linhas estão atualmente nas tabelas EMP e EMP\_COMMISSION, respectivamente:

```
select deptno,empno,ename,comm  
  from emp  
 order by 1
```

DEPTNO	EMPNO	ENAME	COMM
10	7782	CLARK	
10	7839	KING	
10	7934	MILLER	
20	7369	SMITH	
20	7876	ADAMS	
20	7902	FORD	
20	7788	SCOTT	
20	7566	JONES	
30	7499	ALLEN	300
30	7698	BLAKE	
30	7654	MARTIN	1400
30	7900	JAMES	
30	7844	TURNER	0
30	7521	WARD	500

```
select deptno,empno,ename,comm  
  from emp_commission  
 order by 1
```

DEPTNO	EMPNO	ENAME	COMM
10	7782	CLARK	
10	7839	KING	
10	7934	MILLER	

## Solução

A instrução projetada para resolver esse problema é a instrução MERGE e pode executar um UPDATE ou um INSERT, conforme necessário. Por exemplo:

```
1 merge into emp_commission ec  
2 using (select * from emp) emp  
3   on (ec.empno=emp.empno)  
4 when matched then  
5       update set ec.comm = 1000  
6       delete where (sal < 2000)  
7 when not matched then  
8       insert (ec.empno,ec.ename,ec.deptno,ec.comm)  
9       values (emp.empno,emp.ename,emp.deptno,emp.comm)
```

Atualmente, o MySQL não possui uma instrução MERGE; caso contrário, essa consulta deve funcionar em qualquer RDBMS deste livro e em vários outros.

## Discussão

A junção na linha 3 da solução determina quais linhas já existem e serão atualizadas. A junção é entre EMP\_COMMISSION (alias de EC) e a subconsulta (alias de EMP). Quando a união é bem-sucedida, as duas linhas são consideradas “correspondentes” e o UPDATE especificado na cláusula WHEN MATCHED é executado. Caso contrário, nenhuma correspondência é encontrada e o INSERT em WHEN NOT MATCHED é executado. Assim, as linhas da tabela EMP que não tiverem linhas correspondentes baseadas em EMPNO na tabela EMP\_COMMISSION serão inseridas em EMP\_COMMISSION. De todos os funcionários da tabela EMP, apenas os da DEPTNO 10 devem ter sua COMM atualizada em EMP\_COMMISSION, enquanto os demais funcionários são inseridos. Além disso, como MILLER está em DEPTNO 10, ele é candidato a ter seu COMM atualizado, mas como seu SAL é menor que 2.000, ele é excluído de EMP\_COMMISSION.

## 4.12 Excluindo todos os registros de uma tabela

### Problema

Você deseja excluir todos os registros de uma tabela.

### Solução

Use o comando DELETE para excluir registros de uma tabela. Por exemplo, para excluir todos os registros do EMP, use o seguinte:

```
delete from emp
```

## Discussão

Ao usar o comando DELETE sem uma cláusula WHERE, você excluirá todas as linhas da tabela especificada. Às vezes, TRUNCATE, que se aplica a tabelas e, portanto, não usa a cláusula WHERE, é preferido por ser mais rápido. Pelo menos no Oracle, no entanto, TRUNCATE não pode ser desfeito. Você deve verificar cuidadosamente a documentação do fornecedor para obter uma visão detalhada das diferenças de desempenho e reversão entre TRUNCATE e DELETE em seu RDBMS específico.

## 4.13 Excluindo registros específicos

### Problema

Você deseja excluir registros que atendem a um critério específico de uma tabela.

Use o comando DELETE com uma cláusula WHERE especificando quais linhas excluir. Por exemplo, para excluir todos os funcionários do departamento 10, use o seguinte:

```
delete from emp where deptno = 10
```

## Discussão

Usando uma cláusula WHERE com o comando DELETE, você pode excluir um subconjunto de linhas em uma tabela em vez de todas as linhas. Não se esqueça de verificar se você está excluindo os dados corretos visualizando o efeito de sua cláusula WHERE usando SELECT — você pode excluir os dados errados mesmo em uma situação simples. Por exemplo, no caso anterior, um erro de digitação pode levar à exclusão dos funcionários do departamento 20 em vez do departamento 10!

## 4.14 Excluindo um único registro

### Problema

Você deseja excluir um único registro de uma tabela.

### Solução

Este é um caso especial de [Receita 4.13](#). A chave é garantir que seu critério de seleção seja restrito o suficiente para especificar apenas o registro que você deseja excluir. Frequentemente, você desejará excluir com base na chave primária. Por exemplo, para excluir o funcionário CLARK (EMPNO 7782):

```
delete from emp where empno = 7782
```

## Discussão

Excluir é sempre identificar as linhas a serem excluídas, e o impacto de um DELETE sempre se resume à sua cláusula WHERE. Omite a cláusula WHERE e o escopo de um DELETE é a tabela inteira. Ao escrever as condições na cláusula WHERE, você pode restringir o escopo a um grupo de registros ou a um único registro. Ao excluir um único registro, você normalmente deve identificar esse registro com base em sua chave primária ou em uma de suas chaves exclusivas.



Se o seu critério de exclusão for baseado em uma chave primária ou exclusiva, você pode ter certeza de excluir apenas um registro. (Isso ocorre porque seu RDBMS não permitirá que duas linhas contenham os mesmos valores de chave primária ou exclusiva.) Caso contrário, você pode querer verificar primeiro, para ter certeza de que não está prestes a excluir inadvertidamente mais registros do que pretende.

## 4.15 Excluindo violações de integridade referencial

### Problema

Você deseja excluir registros de uma tabela quando esses registros se referem a registros inexistentes em alguma outra tabela. Por exemplo, alguns funcionários são atribuídos a departamentos que não existem. Você deseja excluir esses funcionários.

### Solução

Use o predicado NOT EXISTS com uma subconsulta para testar a validade dos números de departamento:

```
delete from emp
where not exists (
    select * from dept
    where dept.deptno = emp.deptno
)
```

Como alternativa, você pode escrever a consulta usando um predicado NOT IN:

```
delete from emp
where deptno not in (select deptno from dept)
```

### Discussão

Excluir é realmente selecionar: o verdadeiro trabalho está em escrever a cláusula WHERE condições para descrever corretamente os registros que você deseja excluir.

A solução NOT EXISTS usa uma subconsulta correlacionada para testar a existência de um registro em DEPT com um DEPTNO correspondente a um determinado registro EMP. Se tal registro existir, o registro EMP será retido. Caso contrário, ele é excluído. Cada registro EMP é verificado dessa maneira.

A solução IN usa uma subconsulta para recuperar uma lista de números de departamentos válidos. Os DEPTNOS de cada registro EMP são verificados nessa lista. Quando um registro EMP é encontrado com um DEPTNO fora da lista, o registro EMP é excluído.

## 4.16 Excluindo registros duplicados

### Problema

Você deseja excluir registros duplicados de uma tabela. Considere a seguinte tabela:

```
create table dupes (id integer, name varchar(10))

insert into dupes values (1, 'NAPOLEON')
insert into dupes values (2, 'DYNAMITE')
```

```
insert into dupes values (3, 'DYNAMITE')
insert into dupes values (4, 'SHE SELLS')
insert into dupes values (5, 'SEA SHELLS')
insert into dupes values (6, 'SEA SHELLS')
insert into dupes values (7, 'SEA SHELLS')

select * from dupes order by 1

ID NAME
-----
1 NAPOLEON
2 DYNAMITE
3 DYNAMITE
4 SHE SELLS
5 SEA SHELLS
6 SEA SHELLS
7 SEA SHELLS
```

Para cada grupo de nomes duplicados, como SEA SHELLS, você deseja reter arbitrariamente um ID e excluir o restante. No caso de SEA SHELLS, você não se importa se apaga as linhas 5 e 6, ou as linhas 5 e 7, ou as linhas 6 e 7, mas no final você quer apenas um registro para SEA SHELLS.

## Solução

Use uma subconsulta com uma função agregada como MIN para escolher arbitrariamente o ID a ser retido (neste caso, apenas o NAME com o menor valor para ID não é excluído):

```
1 delete from dupes
2 where id not in ( select min(id)
3   from dupes
4   group by name)
```

Para usuários do MySQL, você precisará de uma sintaxe ligeiramente diferente porque não pode referenciar a mesma tabela duas vezes em uma exclusão (no momento em que este livro foi escrito):

```
1 delete from dupes
2 where id not in
3 (select min(id)
4   from (select id,name from dupes) tmp
5   group by name)
```

## Discussão

A primeira coisa a fazer ao excluir duplicatas é definir exatamente o que significa duas linhas serem consideradas “duplicatas” uma da outra. Para meu exemplo nesta receita, a definição de “duplicado” é que dois registros contêm o mesmo valor em sua coluna NAME. Tendo essa definição em vigor, você pode olhar para alguma outra coluna para discriminar entre cada conjunto de duplicatas, para identificar os registros a serem retidos.

É melhor se esta coluna (ou colunas) discriminantes for uma chave primária. Usamos a coluna ID, que é uma boa escolha porque não há dois registros com o mesmo ID.

A chave para a solução é agrupar pelos valores que são duplicados (por NOME, neste caso) e, em seguida, usar uma função agregada para selecionar apenas um valor-chave a ser retido. A subconsulta no exemplo “Solução” retornará o menor ID para cada NOME, que representa a linha que você não excluirá:

```
select min(id)
  from dupes
 group by name
```

MIN(ID)
-----
2
1
5
4

O DELETE exclui qualquer ID na tabela que não seja retornado pela subconsulta (neste caso, os IDs 3, 6 e 7). Se você está tendo problemas para ver como isso funciona, execute a subconsulta primeiro e inclua o NOME na lista SELECT:

```
select name, min(id)
  from dupes
 group by name
```

NAME	MIN(ID)
DYNAMITE	2
NAPOLEON	1
SEA SHELLS	5
SHE SELLS	4

As linhas retornadas pela subconsulta representam aquelas a serem retidas. O predicado NOT IN na instrução DELETE faz com que todas as outras linhas sejam excluídas.

## 4.17 Excluindo registros referenciados de outra tabela

### Problema

Você deseja excluir registros de uma tabela quando esses registros são referenciados em alguma outra tabela. Considere a tabela a seguir, denominada DEPT\_ACCIDENTS, que contém uma linha para cada acidente ocorrido em uma empresa de manufatura. Cada linha registra o departamento em que ocorreu o acidente e também o tipo de acidente.

```
create table dept_accidents
( deptno    integer,
  accident_name  varchar(20) )
```

```
insert into dept_accidents values (10,'BROKEN FOOT')
insert into dept_accidents values (10,'FLESH WOUND')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FLOOD')
insert into dept_accidents values (30,'BRUISED GLUTE')

select * from dept_accidents

DEPTNO ACCIDENT_NAME
-----
10 BROKEN FOOT
10 FLESH WOUND
20 FIRE
20 FIRE
20 FLOOD
30 BRUISED GLUTE
```

Você deseja excluir do EMP os registros dos funcionários que trabalham em um departamento com três ou mais acidentes.

## Solução

Use uma subconsulta e a função agregada COUNT para localizar os departamentos com três ou mais acidentes. Em seguida, exclua todos os funcionários que trabalham nesses departamentos:

```
1 delete from emp
2 where deptno in ( select deptno
3 from dept_accidents
4 group by deptno
5 having count(*) >= 3)
```

## Discussão

A subconsulta identificará quais departamentos possuem três ou mais acidentes:

```
select deptno
  from dept_accidents
 group by deptno
having count(*) >= 3

DEPTNO
-----
20
```

O DELETE excluirá todos os funcionários nos departamentos retornados pela subconsulta (neste caso, apenas no departamento 20).

## 4.18 Resumindo

Inserir e atualizar dados pode parecer tomar menos tempo do que consultar dados, e no restante do livro vamos nos concentrar em consultas. No entanto, ser capaz de manter os dados em um banco de dados é claramente fundamental para sua finalidade, e essas receitas são uma parte crucial do conjunto de habilidades necessárias para manter um banco de dados. Alguns desses comandos, especialmente comandos que removem ou excluem dados, podem ter consequências duradouras. Sempre visualize todos os dados que pretende excluir para ter certeza de que está realmente excluindo o que pretende e familiarize-se com o que pode e o que não pode ser desfeito em seu RDBMS específico.

## CAPÍTULO 5

# Consultas de metadados

Este capítulo apresenta receitas que permitem encontrar informações sobre um determinado esquema. Por exemplo, você pode querer saber quais tabelas você criou ou quais chaves estrangeiras não estão indexadas. Todos os RDBMSs neste livro fornecem tabelas e exibições para obter esses dados. As receitas neste capítulo o ajudarão a coletar informações dessas tabelas e exibições.

Embora em alto nível a estratégia de armazenar metadados em tabelas e visualizações dentro do RDBMS seja comum, a implementação final não é padronizada no mesmo grau que a maioria dos recursos da linguagem SQL abordados neste livro. Portanto, em comparação com outros capítulos, neste capítulo é muito mais comum ter uma solução diferente para cada RDBMS.

A seguir, uma seleção das consultas de esquema mais comuns escritas para cada um dos RDBMSs abordados no livro. Há muito mais informações disponíveis do que as receitas deste capítulo podem mostrar. Consulte a documentação do seu RDBMS para obter a lista completa de catálogos ou tabelas/visualizações de dicionário de dados quando precisar ir além do que é apresentado aqui.



Para fins de demonstração, todas as receitas neste capítulo assumem que existe um esquema denominado SMEAGOL.

## 5.1 Listando tabelas em um esquema

### Problema

Você deseja ver uma lista de todas as tabelas que criou em um determinado esquema.

As soluções a seguir pressupõem que você esteja trabalhando com o esquema SMEAGOL. A abordagem básica para uma solução é a mesma para todos os RDBMSs: você consulta uma tabela do sistema (ou exibição) contendo uma linha para cada tabela no banco de dados.

## DB2

Consulta SYSCAT.TABLES:

```
1 select tabname
2   from syscat.tables
3  where tabschema = 'SMEAGOL'
```

## Oracle

Consulta SYS.ALL\_TABLES:

```
select table_name
      from all_tables
     where owner = 'SMEAGOL'
```

## PostgreSQL, MySQL e SQL Server

Consulta INFORMATION\_SCHEMA.TABLES:

```
1 select table_name
2   from information_schema.tables
3  where table_schema = 'SMEAGOL'
```

## Discussão

De uma maneira deliciosamente circular, os bancos de dados expõem informações sobre si mesmos por meio dos próprios mecanismos que você cria para seus próprios aplicativos: tabelas e exibições. A Oracle, por exemplo, mantém um extenso catálogo de visualizações do sistema, como ALL\_TABLES, que você pode consultar para obter informações sobre tabelas, índices, concessões e qualquer outro objeto de banco de dados.



As exibições de catálogo da Oracle são apenas isso, exibições. Eles são baseados em um conjunto subjacente de tabelas que contêm as informações em um formato pouco amigável. As exibições dão uma cara utilizável aos dados do catálogo da Oracle.

As exibições do sistema Oracle e as tabelas do sistema DB2 são específicas de cada fornecedor. PostgreSQL, MySQL e SQL Server, por outro lado, suportam algo chamado esquema *information*, que é um conjunto de visualizações definido pelo padrão ISO SQL. É por isso que a mesma consulta pode funcionar para todos os três bancos de dados.

## 5.2 Listando as colunas de uma tabela

### Problema

Você deseja listar as colunas em uma tabela, juntamente com seus tipos de dados e sua posição na tabela em que estão.

### Solução

As soluções a seguir pressupõem que você deseja listar colunas, seus tipos de dados e sua posição numérica na tabela chamada EMP no esquema SMEAGOL.

#### DB2

Consulta SYSCAT.COLUMNS:

```
1 select colname, typename, colno
2   from syscat.columns
3  where tabname  = 'EMP'
4    and tabschema = 'SMEAGOL'
```

#### Oracle

Consulta ALL\_TAB\_COLUMNS:

```
1 select column_name, data_type, column_id
2   from all_tab_columns
3  where owner      = 'SMEAGOL'
4    and table_name = 'EMP'
```

#### PostgreSQL, MySQL e SQL Server

Consulta INFORMATION\_SCHEMA.COLUMNS:

```
1 select column_name, data_type, ordinal_position
2   from information_schema.columns
3  where table_schema = 'SMEAGOL'
4    and table_name    = 'EMP'
```

### Discussão

Cada fornecedor oferece maneiras de obter informações detalhadas sobre os dados da coluna. Nos exemplos anteriores, apenas o nome da coluna, o tipo de dados e a posição são retornados. Informações úteis adicionais incluem comprimento, capacidade de nulidade e valores padrão.

## 5.3 Listando colunas indexadas para uma tabela

### Problema

Você deseja listar índices, suas colunas e a posição da coluna (se disponível) no índice de uma determinada tabela.

### Solução

As soluções específicas do fornecedor a seguir pressupõem que você está listando índices para a tabela EMP no esquema SMEAGOL.

#### DB2

Consulta SYSCAT.INDEXES:

```
1 select a.tabname, b.indname, b.colname, b.colseq
2   from syscat.indexes a,
3        syscat.indexcoluse b
4  where a.tabname    = 'EMP'
5    and a.tabschema = 'SMEAGOL'
6    and a.indschema = b.indschema
7    and a.indname   = b.indname
```

#### Oracle

Consulta SYS.ALL\_IND\_COLUMNS:

```
select table_name, index_name, column_name, column_position
  from sys.all_ind_columns
 where table_name = 'EMP'
   and table_owner = 'SMEAGOL'
```

#### PostgreSQL

Consulta PG\_CATALOG.PG\_INDEXES e INFORMATION\_SCHEMA.COLUMNS:

```
1 select a.tablename,a.indexname,b.column_name
2   from pg_catalog.pg_indexes a,
3        information_schema.columns b
4  where a.schemaname = 'SMEAGOL'
5    and a.tablename = b.table_name
```

#### MySQL

Use o comando SHOW INDEX:

```
show index from emp
```

## SQL Server

Consulte SYS.TABLES, SYS.INDEXES, SYS.INDEX\_COLUMNS e SYS.COLUMNS:

```

1 select a.name table_name,
2       b.name index_name,
3       d.name column_name,
4       c.index_column_id
5   from sys.tables a,
6        sys.indexes b,
7        sys.index_columns c,
8        sys.columns d
9  where a.object_id = b.object_id
10    and b.object_id = c.object_id
11    and b.index_id = c.index_id
12    and c.object_id = d.object_id
13    and c.column_id = d.column_id
14  and a.name      = 'EMP'

```

## Discussão

Quando se trata de consultas, é importante saber quais colunas são/não são indexadas. Os índices podem fornecer um bom desempenho para consultas em colunas que são usadas com frequência em filtros e que são bastante seletivas. Os índices também são úteis ao unir tabelas. Ao saber quais colunas são indexadas, você já está um passo à frente dos problemas de desempenho, caso ocorram. Além disso, você pode querer encontrar informações sobre os próprios índices: quantos níveis de profundidade eles têm, quantas chaves distintas existem, quantos blocos existem e assim por diante. Essas informações também estão disponíveis nas visualizações/tabelas consultadas nas soluções desta receita.

## 5.4 Listando restrições em uma tabela

### Problema

Você deseja listar as restrições definidas para uma tabela em algum esquema e as colunas nas quais elas são definidas. Por exemplo, você deseja localizar as restrições e as colunas em que estão para a tabela EMP.

### Solução

#### DB2

Consulte SYSCAT.TABCONST e SYSCAT.COLUMNS:

```

1 select a.tabname, a.constrname, b.colname, a.type
2   from syscat.tabconst a,
3        syscat.columns b
4  where a.tabname    = 'EMP'

```

```
5  and a.tabschema = 'SMEAGOL'
6  and a.tabname      = b.tabname
7  and a.tabschema = b.tabschema
```

## Oracle

Consulte SYS.ALL\_CONSTRAINTS e SYS.ALL\_CONS\_COLUMNS:

```
1 select a.table_name,
2       a.constraint_name,
3       b.column_name,
4       a.constraint_type
5  from all_constraints a,
6       all_cons_columns b
7 where a.table_name      = 'EMP'
8   and a.owner            = 'SMEAGOL'
9   and a.table_name      = b.table_name
10  and a.owner           = b.owner
11  and a.constraint_name = b.constraint_name
```

## PostgreSQL, MySQL e SQL Server

Consulta INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS e INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE:

```
1 select a.table_name,
2       a.constraint_name,
3       b.column_name,
4       a.constraint_type
5  from information_schema.table_constraints a,
6       information_schema.key_column_usage b
7 where a.table_name      = 'EMP'
8   and a.table_schema    = 'SMEAGOL'
9   and a.table_name      = b.table_name
10  and a.table_schema    = b.table_schema
11  and a.constraint_name = b.constraint_name
```

## Discussão

As restrições são uma parte tão crítica dos bancos de dados relacionais que nem é preciso dizer por que você precisa saber quais são as restrições em suas tabelas. Listar as restrições nas tabelas é útil por vários motivos: você pode querer encontrar tabelas sem uma chave primária, você pode querer descobrir quais colunas deveriam ser chaves estrangeiras, mas não são (ou seja, tabelas filhas têm dados diferentes das tabelas pais e você quer saber como isso aconteceu), ou você pode querer saber sobre restrições de verificação (as colunas são anuláveis? Elas precisam satisfazer uma condição específica? etc.).

## 5.5 Listando chaves estrangeiras sem índices correspondentes

### Problema

Você deseja listar tabelas que possuem colunas de chave estrangeira que não são indexadas. Por exemplo, você deseja determinar se as chaves estrangeiras na tabela EMP são indexadas.

### Solução

#### DB2

Consulte SYSCAT.TABCONST, SYSCAT.KEYCOLUSE, SYSCAT.INDEXES e SYSCAT.INDEXCOLUSE:

```
1 select fkeys.tabname,
2       fkeys.constname,
3       fkeys.colname,
4       ind_cols.indname
5   from (
6 select a.tabschema, a.tabname, a.constname, b.colname
7   from syscat.tabconst a,
8        syscat.keycoluse b
9 where a.tabname    = 'EMP'
10    and a.tabschema = 'SMEAGOL'
11    and a.type      = 'F'
12    and a.tabname    = b.tabname
13    and a.tabschema = b.tabschema
14      ) fkeys
15      left join
16      (
17 select a.tabschema,
18       a.tabname,
19       a.indname,
20       b.colname
21   from syscat.indexes a,
22        syscat.indexcoluse b
23 where a.indschema = b.indschema
24    and a.indname    = b.indname
25      ) ind_cols
26 on (fkeys.tabschema = ind_cols.tabschema
27      and fkeys.tabname    = ind_cols.tabname
28      and fkeys.colname    = ind_cols.colname )
29 where ind_cols.indname is null
```

#### Oracle

Consulte SYS.ALL\_CONS\_COLUMNS, SYS.ALL\_CONSTRAINTS e SYS.ALL\_IND\_COLUMNS:

```

1 select a.table_name,
2       a.constraint_name,
3       a.column_name,
4       c.index_name
5   from all_cons_columns a,
6        all_constraints b,
7        all_ind_columns c
8 where a.table_name      = 'EMP'
9   and a.owner          = 'SMEAGOL'
10  and b.constraint_type = 'R'
11  and a.owner          = b.owner
12  and a.table_name     = b.table_name
13  and a.constraint_name = b.constraint_name
14  and a.owner          = c.table_owner (+)
15  and a.table_name     = c.table_name (+)
16  and a.column_name    = c.column_name (+)
17  and c.index_name     is null

```

## PostgreSQL

Consulte INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE,  
 INFORMATION\_SCHEMA.REFERENTIAL\_CONSTRAINTS,  
 INFORMATION\_SCHEMA.COLUMNS, e PG\_CATALOG.PG\_INDEXES:

```

1 select fkeys.table_name,
2       fkeys.constraint_name,
3       fkeys.column_name,
4       ind_cols.indexname
5   from (
6 select a.constraint_schema,
7       a.table_name,
8       a.constraint_name,
9       a.column_name
10  from information_schema.key_column_usage a,
11      information_schema.referential_constraints b
12 where a.constraint_name = b.constraint_name
13   and a.constraint_schema = b.constraint_schema
14   and a.constraint_schema = 'SMEAGOL'
15   and a.table_name      = 'EMP'
16   ) fkeys
17   left join
18   (
19 select a.schemaname, a.tablename, a.indexname, b.column_name
20   from pg_catalog.pg_indexes a,
21      information_schema.columns b
22 where a.tablename = b.table_name
23   and a.schemaname = b.table_schema
24   ) ind_cols
25  on ( fkeys.constraint_schema = ind_cols.schemaname
26      and fkeys.table_name     = ind_cols.tablename
27      and fkeys.column_name    = ind_cols.column_name )
28 where ind_cols.indexname is null

```

## MySQL

Você pode usar o comando SHOW INDEX para recuperar informações de índice, como nome do índice, colunas no índice e posição ordinal das colunas no índice. Além disso, você pode consultar INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE para listar as chaves estrangeiras de uma determinada tabela. No MySQL 5, diz-se que as chaves estrangeiras são indexadas automaticamente, mas na verdade podem ser descartadas. Para determinar se o índice de uma coluna de chave estrangeira foi descartado, você pode executar SHOW INDEX para uma tabela específica e comparar a saída com a de INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE.COLUMN\_NAME para a mesma tabela. Se COLUMN\_NAME estiver listado em KEY\_COLUMN\_USAGE, mas não for retornado por SHOW INDEX, você saberá que a coluna não está indexada.

## SQL Server

Consulte SYS.TABLES, SYS.FOREIGN\_KEYS, SYS.COLUMNS, SYS.INDEXES e SYS.INDEX\_COLUMNS:

```

1 select fkeys.table_name,
2       fkeys.constraint_name,
3       fkeys.column_name,
4       ind_cols.index_name
5   from (
6 select a.object_id,
7       d.column_id,
8       a.name table_name,
9       b.name constraint_name,
10      d.name column_name
11  from sys.tables a
12    join
13      sys.foreign_keys b
14  on ( a.name          = 'EMP'
15      and a.object_id = b.parent_object_id
16      )
17    join
18      sys.foreign_key_columns c
19  on ( b.object_id = c.constraint_object_id )
20    join
21      sys.columns d
22  on ( c.constraint_column_id = d.column_id
23      and a.object_id          = d.object_id
24      )
25    ) fkeys
26  left join
27  (
28 select a.name index_name,
29       b.object_id,
30       b.column_id
31  from sys.indexes a,
32       sys.index_columns b

```

```
29 where a.index_id = b.index_id  
30 ) ind_cols  
31 on (fkeys.object_id = ind_cols.object_id  
32 and fkeys.column_id = ind_cols.column_id )  
33 where ind_cols.index_name is null
```

## Discussão

Cada fornecedor usa seu próprio mecanismo de bloqueio ao modificar linhas. Nos casos em que há um relacionamento pai-filho imposto por meio de chave estrangeira, ter índices na(s) coluna(s) filha(s) pode reduzir o bloqueio (consulte a documentação específica do RDBMS para obter detalhes). Em outros casos, é comum que uma tabela filho seja unida a uma tabela pai na coluna de chave estrangeira, portanto, um índice também pode ajudar a melhorar o desempenho nesse cenário.

## 5.6 Usando SQL para gerar SQL

### Problema

Você deseja criar instruções SQL dinâmicas, talvez para automatizar tarefas de manutenção. Você deseja realizar três tarefas em particular: contar o número de linhas em suas tabelas, desabilitar restrições de chave estrangeira definidas em suas tabelas e gerar scripts de inserção a partir dos dados em suas tabelas.

### Solução

O conceito é usar strings para construir instruções SQL, e os valores que precisam ser preenchidos (como o nome do objeto sobre o qual o comando atua) serão fornecidos pelos dados das tabelas que você está selecionando. Lembre-se de que as consultas geram apenas as instruções; você deve executar essas instruções por meio de script, manualmente ou de qualquer maneira que execute suas instruções SQL. Os exemplos a seguir são consultas que funcionariam em um sistema Oracle. Para outros RDBMSs, a técnica é exatamente a mesma, sendo a única diferença coisas como os nomes das tabelas do dicionário de dados e a formatação das datas. A saída mostrada nas consultas a seguir é uma parte das linhas retornadas de uma instância do Oracle em meu laptop. É claro que seus conjuntos de resultados variam:

```
/* gera SQL para contar todas as linhas em todas as suas tabelas */  
  
select 'select count(*) from'||table_name||';' cnts  
from user_tables;  
  
CNTS  
-----  
select count(*) from ANT;  
select count(*) from BONUS;  
select count(*) from DEMO1;  
select count(*) from DEMO2;
```

```

select count(*) from DEPT;
select count(*) from DUMMY;
select count(*) from EMP;
select count(*) from EMP_SALES;
select count(*) from EMP_SCORE;
select count(*) from PROFESSOR;
select count(*) from T;
select count(*) from T1;
select count(*) from T2;
select count(*) from T3;
select count(*) from TEACH;
select count(*) from TEST;
select count(*) from TRX_LOG;
select count(*) from X;

/* desabilita chaves estrangeiras de todas as tabelas */

select 'alter table '||table_name||
       ' disable constraint '||constraint_name||';' cons
  from user_constraints
 where constraint_type = 'R';

CONS
-----
alter table ANT disable constraint ANT_FK;
alter table BONUS disable constraint BONUS_FK;
alter table DEMO1 disable constraint DEMO1_FK;
alter table DEMO2 disable constraint DEMO2_FK;
alter table DEPT disable constraint DEPT_FK;
alter table DUMMY disable constraint DUMMY_FK;
alter table EMP disable constraint EMP_FK;
alter table EMP_SALES disable constraint EMP_SALES_FK;
alter table EMP_SCORE disable constraint EMP_SCORE_FK;
alter table PROFESSOR disable constraint PROFESSOR_FK;

/* gera um script de inserção a partir de algumas colunas na tabela EMP */

select 'insert into emp(empno,ename,hiredate)'||chr(10)||
       'values( ''||empno||','''||ename|'
      ||'''',to_date(''||'||hiredate||''));' inserts
  from emp
 where deptno = 10;

INSERTS
-----
insert into emp(empno,ename,hiredate)
values( 7782,'CLARK',to_date('09-JUN-2006 00:00:00') );

insert into emp(empno,ename,hiredate)
values( 7839,'KING',to_date('17-NOV-2006 00:00:00') );

```

```
insert into emp(empno,ename,hiredate)
values( 7934,'MILLER',to_date('23-JAN-2007 00:00:00') );
```

## Discussão

Usar SQL para gerar SQL é particularmente útil para criar scripts portáteis, como você pode usar ao testar em vários ambientes. Além disso, como pode ser visto nos exemplos anteriores, o uso de SQL para gerar SQL é útil para realizar manutenção em lote e para descobrir facilmente informações sobre vários objetos de uma só vez. Gerar SQL com SQL é uma operação extremamente simples e, quanto mais você experimentar, mais fácil se tornará. Os exemplos fornecidos devem fornecer uma boa base sobre como criar seus próprios scripts SQL “dinâmicos” porque, francamente, não há muito o que fazer. Trabalhe nisso e você conseguirá.

## 5.7 Descrevendo as exibições de dicionário de dados em um banco de dados Oracle

### Problema

Você está usando Oracle. Você não consegue lembrar quais exibições de dicionário de dados estão disponíveis para você, nem consegue lembrar suas definições de coluna. Pior ainda, você não tem acesso conveniente à documentação do fornecedor.

### Solução

Esta é uma receita específica da Oracle. A Oracle não apenas mantém um conjunto robusto de visualizações de dicionário de dados, mas também há visualizações de dicionário de dados para documentar as visualizações de dicionário de dados. É tudo tão maravilhosamente circular.

Consulte a visualização chamada DICTIONARY para listar as visualizações do dicionário de dados e suas finalidades:

```
select table_name, comments
  from dictionary
 order by table_name;
```

TABLE_NAME	COMMENTS
ALL_ALL_TABLES	Description of all object and relational tables accessible to the user
ALL_APPLY	Details about each apply process that dequeues from the queue visible to the current user
...	

Consulte DICT\_COLUMNS para descrever as colunas em uma determinada visualização de dicionário de dados:

```
select column_name, comments
  from dict_columns
 where table_name = 'ALL_TAB_COLUMNS';
```

COLUMN_NAME	COMMENTS
OWNER	Table, view or cluster name
TABLE_NAME	Column name
COLUMN_NAME	Datatype of the column
DATA_TYPE	Datatype modifier of the column
DATA_TYPE_MOD	Owner of the datatype of the column
DATA_TYPE_OWNER	Length of the column in bytes
DATA_LENGTH	Length: decimal digits (NUMBER) or binary digits (FLOAT)
DATA_PRECISION	

## Discussão

Antigamente, quando o conjunto de documentação da Oracle não estava tão disponível gratuitamente na web, era incrivelmente conveniente que a Oracle disponibilizasse as visualizações DICTIONARY e DICT\_COLUMNS. Conhecendo apenas essas duas exibições, você pode começar a aprender sobre todas as outras exibições e, em seguida, mudar para aprender sobre todo o banco de dados.

Ainda hoje, é conveniente saber sobre DICTIONARY e DICT\_COLUMNS. Frequentemente, se você não tiver certeza de qual visualização descreve um determinado tipo de objeto, poderá emitir uma consulta curinga para descobrir. Por exemplo, para entender quais exibições podem descrever tabelas em seu esquema:

```
select table_name, comments
  from dictionary
 where table_name LIKE '%TABLE%'
 order by table_name;
```

Esta consulta retorna todos os nomes de exibição de dicionário de dados que incluem o termo TABLE. Essa abordagem aproveita as convenções de nomenclatura de exibição de dicionário de dados bastante consistentes da Oracle. Todas as exibições que descrevem tabelas provavelmente contêm TABLE em seu nome. (Às vezes, como no caso de ALL\_TAB\_COLUMNS, TABLE é abreviado como TAB.)

## 5.8 Resumindo

Consultas em metadados abrem uma gama de possibilidades para permitir que o SQL faça mais trabalho do que você e aliviam parte da necessidade de *saber* seu banco de dados. Isso é especialmente útil quando você lida com bancos de dados mais complexos com estruturas igualmente complexas.

## CAPÍTULO 6

# Trabalhando com Strings

Este capítulo enfoca a manipulação de strings em SQL. Lembre-se de que o SQL não foi projetado para realizar manipulações complexas de strings, e você pode (e irá) achar que trabalhar com strings em SQL é complicado e frustrante às vezes. Apesar das limitações do SQL, existem algumas funções incorporadas úteis fornecidas pelos diferentes DBMSs e tentamos usá-las de maneiras criativas. Este capítulo em particular é representativo da mensagem que tentamos transmitir na introdução; SQL é o bom, o ruim e o feio. Esperamos que você tire deste capítulo uma melhor apreciação do que pode e não pode ser feito em SQL ao trabalhar com strings. Em muitos casos, você ficará surpreso com a facilidade de analisar e transformar strings, enquanto em outros momentos ficará horrorizado com o tipo de SQL necessário para realizar uma tarefa específica.

Muitas das receitas a seguir usam as funções TRANSLATE e REPLACE que agora estão disponíveis em todos os SGBDs abordados neste livro, com exceção do MySQL, que só tem `replace`. Neste último caso, vale a pena notar desde o início que você pode replicar o efeito de TRANSLATE usando funções aninhadas REPLACE.

A primeira receita deste capítulo é extremamente importante, pois é aproveitada por várias das soluções subsequentes. Em muitos casos, você gostaria de ter a capacidade de percorrer uma string movendo-a um caractere por vez. Infelizmente, o SQL não torna isso fácil. Como há funcionalidade de loop limitada no SQL, você precisa imitar um loop para percorrer uma string. Chamamos essa operação de “percorrer uma String” ou “percorrer através de uma String”, e a primeira receita explica a técnica. Esta é uma operação fundamental na análise de strings ao usar SQL e é referenciada e usada por quase todas as receitas neste capítulo. Sugerimos enfaticamente que você se sinta confortável com o funcionamento da técnica.

## 6.1 Andando uma String

### Problema

Você deseja percorrer uma string para retornar cada caractere como uma linha, mas o SQL não possui uma operação de loop. Por exemplo, você deseja exibir o ENAME “KING” da tabela EMP como quatro linhas, onde cada linha contém apenas caracteres de KING.

### Solução

Use um produto cartesiano para gerar o número de linhas necessárias para retornar cada caractere de uma string em sua própria linha. Em seguida, use a função de análise de cadeia de caracteres interna do seu DBMS para extrair os caracteres nos quais você está interessado (os usuários do SQL Server usarão SUBSTRING em vez de SUBSTR e DATALENGTH em vez de LENGTH):

```
1 select substr(e.ename,iter.pos,1) as C
2   from (select ename from emp where ename = 'KING') e,
3         (select id as pos from t10) iter
4  where iter.pos <= length(e.ename)
```

C  
-  
K  
I  
N  
G

### Discussão

A chave para iterar pelos caracteres de uma string é juntar-se a uma tabela que tenha linhas suficientes para produzir o número necessário de iterações. Este exemplo usa a tabela T10, que contém 10 linhas (tem uma coluna, ID, contendo os valores de 1 a 10). O número máximo de linhas que podem ser retornadas dessa consulta é 10.

O exemplo a seguir mostra o produto cartesiano entre E e ITER (ou seja, entre o nome específico e as 10 linhas de T10) sem analisar ENAME:

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
```

ENAME	POS
KING	1
KING	2
KING	3
KING	4
KING	5

KING	6
KING	7
KING	8
KING	9
KING	10

A cardinalidade da visualização em linha E é 1 e a cardinalidade da visualização em linha ITER é 10. O produto cartesiano é, então, 10 linhas. A geração desse produto é a primeira etapa para imitar um loop no SQL.



É prática comum referir-se à mesa T10 como uma mesa “pivot”.

A solução usa uma cláusula WHERE para interromper o loop após o retorno de quatro linhas. Para restringir o conjunto de resultados ao mesmo número de linhas de caracteres no nome, essa cláusula WHERE especifica ITER.POS <= LENGTH(E. ENAME) como a condição:

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
 where iter.pos <= length(e.ename)
```

ENAME	POS
KING	1
KING	2
KING	3
KING	4

Agora que você tem uma linha para cada caractere em E.ENAME, pode usar ITER.POS como parâmetro para SUBSTR, permitindo navegar pelos caracteres da string. ITER.POS aumenta com cada linha e, portanto, cada linha pode ser feita para retornar um caractere sucessivo de E.ENAME. É assim que o exemplo de solução funciona.

Dependendo do que você está tentando realizar, você pode ou não precisar gerar uma linha para cada caractere em uma string. A consulta a seguir é um exemplo de percorrer E.ENAME e expor diferentes partes (mais de um único caractere) da string:

```
select substr(e.ename,iter.pos) a,
       substr(e.ename,length(e.ename)-iter.pos+1)
         b
  from (select ename from emp where ename = 'KING') e,
       (select id pos from t10) iter
 where iter.pos <= length(e.ename)
```

A	B
KING	G
ING	NG
NG	ING
G	KING

Os cenários mais comuns para as receitas neste capítulo envolvem percorrer toda a string para gerar uma linha para cada caractere na string ou percorrer a string de modo que o número de linhas geradas reflita o número de caracteres ou delimitadores específicos que estão presentes na string. String.

## 6.2 Incorporando aspas em literais de string

### Problema

Você deseja incorporar aspas em literais de string. Você gostaria de produzir resultados como os seguintes com SQL:

```
QMMARKS
-----
g'day mate
beavers' teeth
'
```

### Solução

Os três SELECTs a seguir destacam diferentes maneiras de criar aspas: no meio de uma string e sozinhos:

```
1 select 'g''day mate' qmarks from t1 union all
2 select 'beavers'' teeth'      from t1 union all
3 select ''''                  from t1
```

### Discussão

Ao trabalhar com aspas, geralmente é útil pensar nelas como parênteses. Quando você tem um parêntese de abertura, você deve sempre ter um parêntese de fechamento. O mesmo vale para citações. Lembre-se de que você sempre deve ter um número par de aspas em qualquer string. Para incorporar uma aspa simples em uma string, você precisa usar duas aspas:

```
select 'apples core', 'apple''s core',
       case when '' is null then 0 else 1 end
     from t1

'APPLESCORE 'APPLE''SCOR CASEWHEN''ISNULLTHEN0ELSE1END
-----
apples core apple's core          0
```

O seguinte é a solução despojada de seus elementos básicos. Você tem duas aspas externas definindo uma string literal e, dentro dessa string literal, você tem duas aspas que, juntas, representam apenas uma aspa na string que você realmente obtém:

```
select ""'' as quote from t1
```

```
Q
```

```
''
```

Ao trabalhar com aspas, lembre-se de que uma string literal composta apenas por duas aspas, sem caracteres intermediários, é NULL.

## 6.3 Contando as ocorrências de um caractere em uma String

### Problema

Você deseja contar o número de vezes que um caractere ou substring ocorre em uma determinada string. Considere a seguinte cadeia:

```
10,CLARK,MANAGER
```

Você deseja determinar quantas vírgulas há na string.

### Solução

Subtraia o comprimento da string sem as vírgulas do comprimento original da string para determinar o número de vírgulas na string. Cada DBMS fornece funções para obter o comprimento de uma string e remover caracteres de uma string. Na maioria dos casos, essas funções são LENGTH e REPLACE, respectivamente (os usuários do SQL Server usarão a função interna LEN em vez de LENGTH):

```
1 select (length('10,CLARK,MANAGER')-
2      length(replace('10,CLARK,MANAGER','','')))/length(',') 
3      as cnt
4  from t1
```

### Discussão

Você chega à solução usando subtração simples. A chamada para LENGTH na linha 1 retorna o tamanho original da string, e a primeira chamada para LENGTH na linha 2 retorna o tamanho da string sem as vírgulas, que são removidas por REPLACE.

Subtraindo os dois comprimentos, você obtém a diferença em termos de caracteres, que é o número de vírgulas na string. A última operação divide a diferença pelo comprimento da string de pesquisa. Essa divisão é necessária se a string que você procura tiver um comprimento maior que 1.

No exemplo a seguir, contar a ocorrência de “LL” na string “HELLO HELLO” sem dividir retornará um resultado incorreto:

```
select
  (length('HELLO HELLO')-
  length(replace('HELLO HELLO','LL','')))/length('LL')
  as correct_cnt,
  (length('HELLO HELLO')-
  length(replace('HELLO HELLO','LL',''))) as incorrect_cnt
from t1

CORRECT_CNT  INCORRECT_CNT
-----  -----
          2            4
```

## 6.4 Removendo caracteres indesejados de uma string

### Problema

Você deseja remover caracteres específicos de seus dados. Um cenário em que isso pode ocorrer é ao lidar com dados numéricos mal formatados, especialmente dados de moeda, onde vírgulas foram usadas para separar zeros e marcadores de moeda são misturados na coluna com a quantidade. Outra situação é que você deseja exportar os dados do seu banco de dados como um arquivo CSV, mas existe um campo de texto contendo vírgulas, que serão lidas como separadores quando o arquivo CSV for acessado. Considere este conjunto de resultados:

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

Você deseja remover todos os zeros e vogais conforme mostrado pelos seguintes valores nas colunas STRIPPED1 e STRIPPED2:

ENAME	STRIPPED1	SAL	STRIPPED2
SMITH	SMTH	800	8
ALLEN	LLN	1600	16

WARD	WRD	1250	125
JONES	JNS	2975	2975
MARTIN	MRTN	1250	125
BLAKE	BLK	2850	285
CLARK	CLRK	2450	245
SCOTT	SCTT	3000	3
KING	KNG	5000	5
TURNER	TRNR	1500	15
ADAMS	DMS	1100	11
JAMES	JMS	950	95
FORD	FRD	3000	3
MILLER	MLLR	1300	13

## Solução

Cada DBMS fornece funções para remover caracteres indesejados de uma string. As funções REPLACE e TRANSLATE são mais úteis para este problema.

### DB2, Oracle, PostgreSQL e SQL Server

Use as funções integradas TRANSLATE e REPLACE para remover caracteres e strings indesejados:

```

1 select ename,
2       replace(translate(ename,'aaaaa','AEIOU'),'a','','') as stripped1,
3       sal,
4       replace(cast(sal as char(4)),'0','','') as stripped2
5   from emp

```

Observe que para DB2, a palavra-chave AS é opcional para designar um alias de coluna e pode ser omitida.

### MySQL

O MySQL não oferece uma função TRANSLATE, então várias chamadas para REPLACE são necessárias:

```

1 select ename,
2       replace(
3       replace(
4       replace(
5       replace(
6       replace(ename,'A','','E','','I','','O','','U','','')
7       as stripped1,
8       sal,
9       replace(sal,0,'') stripped2
7   from emp

```

A função interna REPLACE remove todas as ocorrências de zeros. Para remover as vogais, use TRANSLATE para converter todas as vogais em um caractere específico (usamos “a”; você pode usar qualquer caractere); em seguida, use REPLACE para remover todas as ocorrências desse caractere.

## 6.5 Separando dados numéricos e de caracteres

### Problema

Você tem dados numéricos armazenados com dados de caracteres juntos em uma coluna. Isso pode acontecer facilmente se você herdar dados onde unidades de medida ou moeda foram armazenadas com sua quantidade (por exemplo, uma coluna com *100 km*, AUD \$ 200, ou *40 libras*, em vez da coluna que torna as unidades claras ou uma coluna separada mostrando as unidades quando necessário).

Você deseja separar os dados de caracteres dos dados numéricos. Considere o seguinte conjunto de resultados:

DATA

```
-----  
SMITH800  
ALLEN1600  
WARD1250  
JONES2975  
MARTIN1250  
BLAKE2850  
CLARK2450  
SCOTT3000  
KING5000  
TURNER1500  
ADAMS1100  
JAMES950  
FORD3000  
MILLER1300
```

Você gostaria que o resultado fosse:

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000

TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

## Solução

Use as funções integradas TRANSLATE e REPLACE para isolar o caractere dos dados numéricos. Como outras receitas neste capítulo, o truque é usar TRANSLATE para transformar vários caracteres em um único caractere que você pode referenciar. Dessa forma, você não estará mais procurando por vários números ou caracteres; em vez disso, você está procurando apenas um caractere para representar todos os números ou um caractere para representar todos os caracteres.

### DB2

Use as funções TRANSLATE e REPLACE para isolar e separar os dados numéricos dos caracteres:

```

1 select replace(
2     translate(data,'0000000000','0123456789'),'0','') ename,
3     cast(
4         replace(
5             translate(lower(data),repeat('z',26),
6                 'abcdefghijklmnopqrstuvwxyz'),'z','') as integer) sal
7     from (
8 select ename||cast(sal as char(4)) data
9     from emp
10      ) x

```

### Oracle

Use as funções TRANSLATE e REPLACE para isolar e separar os dados numéricos dos caracteres:

```

1 select replace(
2     translate(data,'0123456789','0000000000'),'0') ename,
3     to_number(
4         replace(
5             translate(lower(data),
6                 'abcdefghijklmnopqrstuvwxyz',
7                 rpad('z',26,'z')),'z')) sal
8     from (
9 select ename||sal data
10    from emp
11      )

```

Use as funções TRANSLATE e REPLACE para isolar e separar os dados numéricos dos caracteres:

```

1 select replace(
2     translate(data,'0123456789','0000000000'),'0','') as ename,
3     cast(
4         replace(
5             translate(lower(data),
6                 'abcdefghijklmnopqrstuvwxyz',
7                 rpad('z',26,'z')),'z','') as integer) as sal
8 from (
9 select ename||sal as data
10 from emp
11      ) x

```

## SQL Server

Use as funções TRANSLATE e REPLACE para isolar e separar os dados numéricos dos caracteres:

```

1 select replace(
2     translate(data,'0123456789','0000000000'),'0','') as ename,
3     cast(
4         replace(
5             translate(lower(data),
6                 'abcdefghijklmnopqrstuvwxyz',
7                 replicate('z',26),'z','') as integer) as sal
8 from (
9 select concat(ename,sal) as data
10 from emp
11      ) x

```

## Discussão

A sintaxe é um pouco diferente para cada DBMS, mas a técnica é a mesma. A sintaxe é ligeiramente diferente para cada DBMS, mas a técnica é a mesma; usaremos a solução Oracle para esta discussão. A chave para resolver esse problema é isolar os dados numéricos e de caracteres. Você pode usar TRANSLATE e REPLACE para fazer isso. Para extrair os dados numéricos, primeiro isole todos os dados de caracteres usando TRANSLATE:

```

select data,
       translate(lower(data),
                  'abcdefghijklmnopqrstuvwxyz',
                  rpad('z',26,'z')) sal
  from (select ename||sal data from emp)

```

DATA	SAL
SMITH800	zzzz800
ALLEN1600	zzzz1600

WARD1250	zzzz1250
JONES2975	zzzzz2975
MARTIN1250	zzzzzz1250
BLAKE2850	zzzzz2850
CLARK2450	zzzzz2450
SCOTT3000	zzzzz3000
KING5000	zzzz5000
TURNER1500	zzzzzz1500
ADAMS1100	zzzzz1100
JAMES950	zzzzz950
FORD3000	zzzz3000
MILLER1300	zzzzzz1300

Usando TRANSLATE, você converte todos os caracteres não numéricos em um Z minúsculo. A próxima etapa é remover todas as instâncias de Z minúsculo de cada registro usando REPLACE, deixando apenas caracteres numéricos que podem ser convertidos em um número:

```
select data,
       to_number(
           replace(
               translate(lower(data),
                       'abcdefghijklmnopqrstuvwxyz',
                       rpad('z',26,'z')),'z')) sal
  from (select ename||sal data from emp)
```

DATA	SAL
SMITH800	800
ALLEN1600	1600
WARD1250	1250
JONES2975	2975
MARTIN1250	1250
BLAKE2850	2850
CLARK2450	2450
SCOTT3000	3000
KING5000	5000
TURNER1500	1500
ADAMS1100	1100
JAMES950	950
FORD3000	3000
MILLER1300	1300

Para extrair os caracteres não numéricos, isole os caracteres numéricos usando TRANSLATE:

```
select data,
       translate(data,'0123456789','0000000000') ename
  from (select ename||sal data from emp)
```

DATA	ENAME
SMITH800	SMITH000
ALLEN1600	ALLEN0000

WARD1250	WARD0000
JONES2975	JONES0000
MARTIN1250	MARTIN0000
BLAKE2850	BLAKE0000
CLARK2450	CLARK0000
SCOTT3000	SCOTT0000
KING5000	KING0000
TURNER1500	TURNER0000
ADAMS1100	ADAMS0000
JAMES950	JAMES000
FORD3000	FORD0000
MILLER1300	MILLER0000

Ao usar TRANSLATE, você converte todos os caracteres numéricos em zero. A próxima etapa é remover todas as instâncias de zero de cada registro usando REPLACE, deixando apenas caracteres não numéricos:

```
select data,
       replace(translate(data,'0123456789','0000000000'),'0') ename
  from (select ename||sal data from emp)
```

DATA	ENAME
- SMITH800	SMITH
ALLEN1600	ALLEN
WARD1250	WARD
JONES2975	JONES
MARTIN1250	MARTIN
BLAKE2850	BLAKE
CLARK2450	CLARK
SCOTT3000	SCOTT
KING5000	KING
TURNER1500	TURNER
ADAMS1100	ADAMS
JAMES950	JAMES
FORD3000	FORD
MILLER1300	MILLER

Junte as duas técnicas e você terá sua solução.

## 6.6 Determinando se uma string é alfanumérica

### Problema

Você deseja retornar linhas de uma tabela somente quando uma coluna de interesse não contém caracteres além de números e letras. Considere a seguinte visão V (os usuários do SQL Server usarão o operador + para concatenação em vez de ||):

```

create view V as
select ename as data
  from emp
 where deptno=10
 union all
select ename||', '$'|| cast(sal as char(4)) ||'.00' as data
  from emp
 where deptno=20
 union all
select ename|| cast(deptno as char(4)) as data
  from emp
 where deptno=30

```

A view V representa sua tabela e retorna o seguinte:

DATA
CLARK
KING
MILLER
SMITH, \$800.00
JONES, \$2975.00
SCOTT, \$3000.00
ADAMS, \$1100.00
FORD, \$3000.00
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30

Porém, dos dados da view você deseja retornar apenas os seguintes registros:

DATA
CLARK
KING
MILLER
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30

Resumindo, você deseja omitir as linhas que contêm dados que não sejam letras e dígitos.

## Solução

A princípio pode parecer intuitivo resolver o problema procurando todos os possíveis caracteres não alfanuméricos que podem ser encontrados em uma string, mas, ao contrário, você achará mais fácil fazer exatamente o oposto: encontrar todos os caracteres alfanuméricos .

Ao fazer isso, você pode tratar todos os caracteres alfanuméricos como um, convertendo-os em um único caractere. A razão pela qual você deseja fazer isso é para que os caracteres alfanuméricos possam ser manipulados juntos, como um todo. Depois de gerar uma cópia da string na qual todos os caracteres alfanuméricos são representados por um único caractere de sua escolha, é fácil isolar os caracteres alfanuméricos de quaisquer outros caracteres.

## DB2

Use a função TRANSLATE para converter todos os caracteres alfanuméricos em um único caractere; em seguida, identifique quaisquer linhas que tenham caracteres diferentes do caractere alfanumérico convertido. Para usuários DB2, as chamadas de função CAST na visão V são necessárias; caso contrário, a exibição não poderá ser criada devido a erros de conversão de tipo. Tome muito cuidado ao trabalhar com conversões para CHAR, pois elas são de comprimento fixo (preenchidas):

```
1 select data
2   from V
3  where translate(lower(data),
4                  repeat('a',36),
5                  '0123456789abcdefghijklmnopqrstuvwxyz') =
5                  repeat('a',length(data))
```

## MySQL

A sintaxe para view V é um pouco diferente no MySQL:

```
create view V as
select ename as data
  from emp
 where deptno=10
 union all
select concat(ename, ' ', sal, '.00') as data
  from emp
 where deptno=20
 union all
select concat(ename,deptno) as data
  from emp
 where deptno=30
```

Use uma expressão regular para localizar facilmente as linhas que contêm dados não alfanuméricos:

```
1 select data
2   from V
3  where data regexp '[^0-9a-zA-Z]' = 0
```

## Oracle e PostgreSQL

Use a função TRANSLATE para converter todos os caracteres alfanuméricos em um único caractere; em seguida, identifique quaisquer linhas que tenham caracteres diferentes do caractere alfanumérico convertido.

As chamadas de função CAST na visão V não são necessárias para Oracle e PostgreSQL. Tome muito cuidado ao trabalhar com conversões para CHAR, pois elas são de comprimento fixo (preenchidas).

Se você decidir lançar, lance para VARCHAR ou VARCHAR2:

```

1 select data
2   from V
3 where translate(lower(data),
4                  '0123456789abcdefghijklmnopqrstuvwxyz',
5                  rpad('a',36,'a')) = rpad('a',length(data),'a')

```

## SQL Server

A técnica é a mesma, com exceção de não haver RPAD no SQL Server:

```

1 select data
2   from V
3 where translate(lower(data),
4                  '0123456789abcdefghijklmnopqrstuvwxyz',
5                  replicate('a',36)) = replicate('a',len(data))

```

## Discussão

A chave para essas soluções é poder fazer referência a vários caracteres simultaneamente. Usando a função TRANSLATE, você pode facilmente manipular todos os números ou todos os caracteres sem ter que “iterar” e inspecionar cada caractere um por um.

### DB2, Oracle, PostgreSQL e SQL Server

Apenas 9 das 14 linhas da visualização V são alfanuméricas. Para encontrar as linhas que são apenas alfanuméricas, basta usar a função TRANSLATE. Neste exemplo, TRANSLATE converte os caracteres 0–9 e a–z em “a”. Uma vez que a conversão é feita, a linha convertida é então comparada com uma string de todos os “a” com o mesmo comprimento (como a linha). Se o comprimento for o mesmo, você saberá que todos os caracteres são alfanuméricos e nada mais.

Usando a função TRANSLATE (usando a sintaxe do Oracle):

```

where translate(lower(data),
                '0123456789abcdefghijklmnopqrstuvwxyz',
                rpad('a',36,'a'))

```

você converte todos os números e letras em um caractere distinto (nós escolhemos “a”). Uma vez que os dados são convertidos, todas as strings que são de fato alfanuméricas podem ser identificadas como uma string composta por apenas um único caractere (neste caso, “a”). Isso pode ser visto executando TRANSLATE sozinho:

```

select data, translate(lower(data),
                      '0123456789abcdefghijklmnopqrstuvwxyz',
                      rpad('a',36,'a'))
from V

```

DATA	TRANSLATE(LOWER(DATA))
CLARK	aaaaa
...	
SMITH, \$800.00	aaaaa, \$aaa.aa
...	
ALLEN30	aaaaaaaa
...	

Os valores alfanuméricos são convertidos, mas os comprimentos das strings não foram modificados. Como os comprimentos são os mesmos, as linhas a serem mantidas são aquelas para as quais a chamada a TRANSLATE retorna todos os "a"s. Você mantém essas linhas, rejeitando as outras, comparando o comprimento de cada string original com o comprimento de sua correspondente cadeia de "a"s:

```
select data, translate(lower(data),
                      '0123456789abcdefghijklmnopqrstuvwxyz',
                      rpad('a',36,'a')) translated,
       rpad('a',length(data),'a') fixed
  from V
```

DATA	TRANSLATED	FIXED
CLARK	aaaaa	aaaaa
...		
SMITH, \$800.00	aaaaa, \$aaa.aa	aaaaaaaaaaaaaaaa
...		
ALLEN30	aaaaaaaa	aaaaaaaa
...		

O último passo é manter apenas as strings onde TRANSLATED é igual a FIXED.

## MySQL

A expressão na cláusula WHERE:

```
onde regexp de dados '[^0-9a-zA-Z]' = 0
```

faz com que as linhas que possuem apenas números ou caracteres sejam retornadas. Os intervalos de valores entre colchetes, “0-9a-zA-Z”, representam todos os números e letras possíveis. O caractere ^ é para negação, então a expressão pode ser declarada como “não números ou letras”. Um valor de retorno de 1 é verdadeiro e 0 é falso, portanto, toda a expressão pode ser declarada como “retorna linhas onde qualquer coisa diferente de números e letras é falsa”.

## 6.7 Extrair iniciais de um nome

### Problema

Você deseja converter um nome completo em iniciais. Considere o seguinte nome:

Stewie Griffin

Você gostaria de retornar:

S.G.

## Solução

É importante ter em mente que o SQL não oferece a flexibilidade de linguagens como C ou Python; portanto, criar uma solução genérica para lidar com qualquer formato de nome não é algo particularmente fácil de fazer em SQL. As soluções apresentadas aqui esperam que os nomes sejam o nome e o sobrenome ou o nome, nome do meio/inicial do meio e sobrenome.

### DB2

Use as funções integradas REPLACE, TRANSLATE e REPEAT para extrair as iniciais:

```

1 select replace(
2     replace(
3         translate(replace('Stewie Griffin', '.', ''), 4
4             repeat('#',26),
5                 'abcdefghijklmnopqrstuvwxyz'),
6                 '#','.'), ' ','.' )
7         ||'.
8 from t1

```

### MySQL

Use as funções integradas CONCAT, CONCAT\_WS, SUBSTRING e SUBSTRING\_INDEX para extrair as iniciais:

```

1 select case
2     when cnt = 2 then
3         trim(trailing '.' from
4             concat_ws('.',
5                 substr(substring_index(name, ' ',1),1,1),
6                 substr(name,
7                     length(substring_index(name, ' ',1))+2,1),
8                 substr(substring_index(name, ' ',-1),1,1),
9                 '.'))
10    else
11        trim(trailing '.' from
12            concat_ws('.',
13                substr(substring_index(name, ' ',1),1,1),
14                substr(substring_index(name, ' ',-1),1,1)
15            ))
16    end as initials
17 from (
18 select name,length(name)-length(replace(name, ' ','')) as cnt
19   from (
20 select replace('Stewie Griffin','.','') as name from t1

```

```

17      )y
18      )x

```

## Oracle e PostgreSQL

Use as funções integradas REPLACE, TRANSLATE e RPAD para extrair as iniciais:

```

1 select replace(
2      replace(
3          translate(replace('Stewie Griffin', '.', ''),
4                  'abcdefghijklmnopqrstuvwxyz',
5                  rpad('#',26,'#') ), '#','' ),' ','.' ) || '.'
6 from t1

```

## SQL Server

```

1 select replace(
2      replace(
3          translate(replace('Stewie Griffin', '.', ''),
4                  'abcdefghijklmnopqrstuvwxyz',
5                  replicate('#',26) ), '#','' ),' ','.' ) + '.'
6 from t1

```

## Discussão

Ao isolar as letras maiúsculas, você pode extrair as iniciais de um nome. As seções a seguir descrevem detalhadamente cada solução específica do fornecedor.

### DB2

A função REPLACE removerá todos os pontos no nome (para lidar com as iniciais do meio) e a função TRANSLATE converterá todas as letras não maiúsculas em #.

```

select translate(replace('Stewie Griffin', '.', ''),
                 repeat('#',26),
                 'abcdefghijklmnopqrstuvwxyz')
from t1

TRANSLATE('STE
-----
S##### G#####

```

Neste ponto, as iniciais são os caracteres que não são #. A função REPLACE é então usada para remover todos os caracteres #:

```

select replace(
    translate(replace('Stewie Griffin', '.', ''),
              repeat('#',26),
              'abcdefghijklmnopqrstuvwxyz'), '#','')
from t1

```

```
REP
---
S G
```

A próxima etapa é substituir o espaço em branco por um ponto usando REPLACE novamente:

```
select replace(
    replace(
        translate(replace('Stewie Griffin', '.', ''),
            repeat('#', 26),
            'abcdefghijklmnopqrstuvwxyz'), '#', ''),
        ' ', '.') || '.')

from t1

REPLA
-----
S.G
```

A etapa final é acrescentar um decimal ao final das iniciais.

### Oracle e PostgreSQL

A função REPLACE removerá todos os pontos no nome (para lidar com as iniciais do meio) e a função TRANSLATE converterá todas as letras não maiúsculas em#.

```
select translate(replace('Stewie Griffin', '.', ''),
    'abcdefghijklmnopqrstuvwxyz',
    rpad('#', 26, '#'))
from t1

TRANSLATE('STE
-----
S##### G#####
```

Neste ponto, as iniciais são os caracteres que não são#. A função REPLACE é então usada para remover todos os caracteres#:

```
select replace(
    translate(replace('Stewie Griffin', '.', ''),
        'abcdefghijklmnopqrstuvwxyz',
        rpad('#', 26, '#')), '#', '')

from t1

REP
---
S G
```

A próxima etapa é substituir o espaço em branco por um ponto usando REPLACE novamente:

```
select replace(
    replace(
        translate(replace('Stewie Griffin', '.', ''),
            'abcdefghijklmnopqrstuvwxyz',
            rpad('#', 26, '#')), '#', ''),
        ' ', '.') || '.'

from t1
```

-----  
S.G.

A etapa final é acrescentar um decimal ao final das iniciais.

## MySQL

A exibição em linha Y é usada para remover qualquer ponto do nome. A visualização inline X localiza o número de espaços em branco no nome para que a função SUBSTR possa ser chamada o número correto de vezes para extrair as iniciais. As três chamadas para SUBSTRING\_INDEX analisam a string em nomes individuais com base na localização do espaço em branco. Como há apenas um nome e um sobrenome, o código na parte ELSE da instrução case é executado:

```
select substr(substring_index(name, ' ',1),1,1) as a,
       substr(substring_index(name,' ', -1),1,1) as b
  from (select 'Stewie Griffin' as name from t1) x
```

A B  
--  
S G

Se o nome em questão tiver um nome do meio ou inicial, a inicial seria retornada executando:

```
substr(nome,comprimento(substr_index(nome, ' ',1))+2,1)
```

que localiza o final do primeiro nome e move dois espaços para o início do nome do meio ou inicial, ou seja, a posição inicial de SUBSTR. Como apenas um caractere é mantido, o nome do meio ou inicial é retornado com êxito. As iniciais são então passadas para CONCAT\_WS, que separa as iniciais por um ponto:

```
select concat_ws('.',
                 substr(substr_index(name, ' ',1),1,1),
                 substr(substr_index(name,' ', -1),1,1),
                 '.') a
  from (select 'Stewie Griffin' as name from t1) x
```

A  
----  
S.G..

A última etapa é cortar o ponto estranho das iniciais.

## 6.8 Ordenando por Partes de uma String

### Problema

Você deseja ordenar seu conjunto de resultados com base em uma substring. Considere os seguintes registros:

ENAME
-----
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER

Você deseja que os registros sejam ordenados com base no *last* dois caracteres de cada nome:

ENAME
-----
ALLEN
TURNER
MILLER
JONES
JAMES
MARTIN
BLAKE
ADAMS
KING
WARD
FORD
CLARK
SMITH
SCOTT

### Solução

A chave para esta solução é encontrar e usar a função interna do seu DBMS para extrair a substring na qual você deseja classificar. Isso normalmente é feito com a função SUBSTR.

Use uma combinação das funções internas LENGTH e SUBSTR para ordenar por uma parte específica de uma string:

```
1 select ename  
2   from emp  
3  order by substr(ename,length(ename)-1,)
```

### **SQL Server**

Use as funções SUBSTRING e LEN para ordenar por uma parte específica de uma string:

```
1 select ename  
2   from emp  
3  order by substring(ename,len(ename)-1,2)
```

## **Discussão**

Usando a expressão SUBSTR em sua cláusula ORDER BY, você pode escolher qualquer parte de uma string para usar na ordenação de um conjunto de resultados. Você também não está limitado a SUBSTR. Você pode ordenar linhas pelo resultado de quase qualquer expressão.

## **6.9 Ordenando por um número em uma string**

### **Problema**

Você deseja ordenar seu conjunto de resultados com base em um número dentro de uma string. Considere a seguinte visão:

```
create view V as  
select e.ename ||' '|  
      cast(e.empno as char(4))||' '|  
      d.dname as data  
     from emp e, dept d  
    where e.deptno=d.deptno
```

Esta visualização retorna os seguintes dados:

DATA		
CLARK	7782	ACCOUNTING
KING	7839	ACCOUNTING
MILLER	7934	ACCOUNTING
SMITH	7369	RESEARCH
JONES	7566	RESEARCH
SCOTT	7788	RESEARCH
ADAMS	7876	RESEARCH
FORD	7902	RESEARCH
ALLEN	7499	SALES
WARD	7521	SALES

```
MARTIN 7654 SALES
BLAKE 7698 SALES
TURNER 7844 SALES
JAMES 7900 SALES
```

Você deseja ordenar os resultados com base no número do funcionário, que fica entre o nome do funcionário e o respectivo departamento:

```
DATA
-----
SMITH 7369 RESEARCH
ALLEN 7499 SALES
WARD 7521 SALES
JONES 7566 RESEARCH
MARTIN 7654 SALES
BLAKE 7698 SALES
CLARK 7782 ACCOUNTING
SCOTT 7788 RESEARCH
KING 7839 ACCOUNTING
TURNER 7844 SALES
ADAMS 7876 RESEARCH
JAMES 7900 SALES
FORD 7902 RESEARCH
MILLER 7934 ACCOUNTING
```

## Solução

Cada solução usa funções e sintaxe específicas para seu DBMS, mas o método (fazendo uso das funções incorporadas REPLACE e TRANSLATE) é o mesmo para cada um. A ideia é usar REPLACE e TRANSLATE para remover não-dígitos das strings, deixando apenas os valores numéricos para classificação.

### DB2

Use as funções integradas REPLACE e TRANSLATE para ordenar por caracteres numéricos em uma string:

```
1 select data
2   from V
3  order by
4    cast(
5      replace(
6        translate(data,repeat('#',length(data))),
7        replace(
8          translate(data,'#####','0123456789'),
9          '#','')),'#','') as integer)
```

### Oracle

Use as funções integradas REPLACE e TRANSLATE para ordenar por caracteres numéricos em uma string:

```
1 select data
2   from V
3  order by
4    to_number(
5      replace(
6        translate(data,
7          replace(
8            translate(data,'0123456789','#####'),
9              '#'),rpad('#',20,'#')),'#'))
```

## PostgreSQL

Use as funções integradas REPLACE e TRANSLATE para ordenar por caracteres numéricos em uma string:

```
1 select data
2   from V
3  order by
4    cast(
5      replace(
6        translate(data,
7          replace(
8            translate(data,'0123456789','#####'),
9              '#','')),rpad('#',20,'#')),'#') as integer)
```

## MySQL

No momento em que este livro foi escrito, o MySQL não fornece a função TRANSLATE.

## Discussão

O objetivo da visão V é apenas fornecer linhas nas quais demonstrar a solução desta receita. A exibição simplesmente concatena várias colunas da tabela EMP. A solução mostra como pegar esse texto concatenado como entrada e classificá-lo pelo número do funcionário incorporado.

A cláusula ORDER BY em cada solução pode parecer intimidante, mas funciona muito bem e é direta quando você a examina peça por peça. Para ordenar pelos números na string, é mais fácil remover quaisquer caracteres que não sejam números. Depois que os caracteres não numéricos são removidos, tudo o que resta a fazer é converter a sequência de numerais em um número e, em seguida, classificar como achar melhor. Antes de examinar cada chamada de função, é importante entender a ordem em que cada função é chamada. Começando com a chamada mais interna, TRANSLATE (linha 8 de cada uma das soluções originais), você vê que:

A partir da chamada mais interna, a sequência de passos é TRANSLATE (linha 8); REPLACE (linha 7); TRANSLATE (linha 6); REPLACE (linha 5). A etapa final é usar CAST para retornar o resultado como um número.

O primeiro passo é converter os números em caracteres que não existem no restante da string. Para este exemplo, escolhemos # e usamos TRANSLATE para converter todos os caracteres não numéricos em ocorrências de #. Por exemplo, a consulta a seguir mostra os dados originais à esquerda e os resultados da primeira tradução:

```
select data,
      translate(data,'0123456789','#####') as tmp
   from V
```

DATA	TMP
CLARK 7782 ACCOUNTING	CLARK ##### ACCOUNTING
KING 7839 ACCOUNTING	KING ##### ACCOUNTING
MILLER 7934 ACCOUNTING	MILLER ##### ACCOUNTING
SMITH 7369 RESEARCH	SMITH ##### RESEARCH
JONES 7566 RESEARCH	JONES ##### RESEARCH
SCOTT 7788 RESEARCH	SCOTT ##### RESEARCH
ADAMS 7876 RESEARCH	ADAMS ##### RESEARCH
FORD 7902 RESEARCH	FORD ##### RESEARCH
ALLEN 7499 SALES	ALLEN ##### SALES
WARD 7521 SALES	WARD ##### SALES
MARTIN 7654 SALES	MARTIN ##### SALES
BLAKE 7698 SALES	BLAKE ##### SALES
TURNER 7844 SALES	TURNER ##### SALES
JAMES 7900 SALES	JAMES ##### SALES

TRANSLATE localiza os numerais em cada string e converte cada um no caractere #. As strings modificadas são então retornadas para REPLACE (linha 11), que remove todas as ocorrências de #:

```
select data,
replace(
translate(data,'0123456789','#####'),'#') as tmp
   from V
```

DATA	TMP
CLARK 7782 ACCOUNTING	CLARK ACCOUNTING
KING 7839 ACCOUNTING	KING ACCOUNTING
MILLER 7934 ACCOUNTING	MILLER ACCOUNTING
SMITH 7369 RESEARCH	SMITH RESEARCH
JONES 7566 RESEARCH	JONES RESEARCH
SCOTT 7788 RESEARCH	SCOTT RESEARCH
ADAMS 7876 RESEARCH	ADAMS RESEARCH
FORD 7902 RESEARCH	FORD RESEARCH
ALLEN 7499 SALES	ALLEN SALES
WARD 7521 SALES	WARD SALES
MARTIN 7654 SALES	MARTIN SALES
BLAKE 7698 SALES	BLAKE SALES
TURNER 7844 SALES	TURNER SALES
JAMES 7900 SALES	JAMES SALES

As strings são então retornadas para TRANSLATE mais uma vez, mas desta vez é o segundo TRANSLATE (mais externo) na solução. TRANSLATE procura na string original quaisquer caracteres que correspondam aos caracteres em TMP. Se algum for encontrado, eles também serão convertidos em #s.

Essa conversão permite que todos os caracteres não numéricos sejam tratados como um único caractere (porque todos são transformados no mesmo caractere):

```
select data, translate(data,
                      replace(
                        translate(data,'0123456789','#####'),
                        '#'),
                        rpad('#',length(data),'#')) as tmp
```

	DATA	TMP
CLARK	7782 ACCOUNTING	#####7782#####
KING	7839 ACCOUNTING	#####7839#####
MILLER	7934 ACCOUNTING	#####7934#####
SMITH	7369 RESEARCH	#####7369#####
JONES	7566 RESEARCH	#####7566#####
SCOTT	7788 RESEARCH	#####7788#####
ADAMS	7876 RESEARCH	#####7876#####
FORD	7902 RESEARCH	#####7902#####
ALLEN	7499 SALES	#####7499#####
WARD	7521 SALES	#####7521#####
MARTIN	7654 SALES	#####7654#####
BLAKE	7698 SALES	#####7698#####
TURNER	7844 SALES	#####7844#####
JAMES	7900 SALES	#####7900#####

O próximo passo é remover todos os caracteres # através de uma chamada para REPLACE (linha 8), deixando você apenas com números:

```
select data, replace(
                      translate(data,
                        replace(
                          translate(data,'0123456789','#####'),
                          '#'),
                          rpad('#',length(data),'#')), '#') as tmp
```

	DATA	TMP
CLARK	7782 ACCOUNTING	7782
KING	7839 ACCOUNTING	7839
MILLER	7934 ACCOUNTING	7934
SMITH	7369 RESEARCH	7369
JONES	7566 RESEARCH	7566
SCOTT	7788 RESEARCH	7788
ADAMS	7876 RESEARCH	7876

FORD	7902	RESEARCH	7902
ALLEN	7499	SALES	7499
WARD	7521	SALES	7521
MARTIN	7654	SALES	7654
BLAKE	7698	SALES	7698
TURNER	7844	SALES	7844
JAMES	7900	SALES	7900

Por fim, converte TMP para um número (linha 4) usando a função DBMS apropriada (geralmente CAST) para fazer isso:

```
select data, to_number(
    replace(
        translate(data,
        replace(
            translate(data,'0123456789','#####'),
            '#'),
        rpad('#',length(data),'#')),'#')) as tmp
from V
```

DATA	TMP	
CLARK	7782 ACCOUNTING	7782
KING	7839 ACCOUNTING	7839
MILLER	7934 ACCOUNTING	7934
SMITH	7369 RESEARCH	7369
JONES	7566 RESEARCH	7566
SCOTT	7788 RESEARCH	7788
ADAMS	7876 RESEARCH	7876
FORD	7902 RESEARCH	7902
ALLEN	7499 SALES	7499
WARD	7521 SALES	7521
MARTIN	7654 SALES	7654
BLAKE	7698 SALES	7698
TURNER	7844 SALES	7844
JAMES	7900 SALES	7900

Ao desenvolver consultas como esta, é útil trabalhar com suas expressões na lista SELECT. Dessa forma, você pode visualizar facilmente os resultados intermediários enquanto trabalha em direção a uma solução final. No entanto, como o objetivo desta receita é ordenar os resultados, você deve colocar todas as chamadas de função na cláusula ORDER BY:

```
select data
  from V
 order by
    to_number(
        replace(
            translate( data,
            replace(
                translate(data,'0123456789','#####'),
                '#'),rpad('#',length(data),'#')),'#'))
```

DATA

```
-----  
SMITH    7369 RESEARCH  
ALLEN    7499 SALES  
WARD     7521 SALES  
JONES    7566 RESEARCH  
MARTIN   7654 SALES  
BLAKE    7698 SALES  
CLARK    7782 ACCOUNTING  
SCOTT    7788 RESEARCH  
KING     7839 ACCOUNTING  
TURNER   7844 SALES  
ADAMS    7876 RESEARCH  
JAMES    7900 SALES  
FORD     7902 RESEARCH  
MILLER   7934 ACCOUNTING
```

Como observação final, os dados na exibição são compostos por três campos, sendo apenas um numérico. Lembre-se de que, se houvesse vários campos numéricos, eles seriam todos concatenados em um número antes de as linhas serem classificadas.

## 6.10 Criando uma lista delimitada de linhas da tabela

### Problema

Você deseja retornar as linhas da tabela como valores em uma lista delimitada, talvez delimitada por vírgulas, em vez de colunas verticais como normalmente aparecem. Você deseja converter um conjunto de resultados a partir disso:

```
DEPTNO EMPS  
-----  
10 CLARK  
10 KING  
10 MILLER  
20 SMITH  
20 ADAMS  
20 FORD  
20 SCOTT  
20 JONES  
30 ALLEN  
30 BLAKE  
30 MARTIN  
30 JAMES  
30 TURNER  
30 WARD
```

para isso:

```
DEPTNO EMPS  
-----  
10 CLARK,KING,MILLER
```

```
20 SMITH,JONES,SCOTT,ADAMS,FORD
30 ALLEN,WARD,MARTIN,BLAKE,TURNER,JAMES
```

## Solução

Cada DBMS requer uma abordagem diferente para esse problema. A chave é aproveitar as funções integradas fornecidas pelo seu DBMS. Compreender o que está disponível para você permitirá que você explore a funcionalidade do seu DBMS e encontre soluções criativas para um problema que normalmente não é resolvido no SQL.

A maioria dos DBMSs agora adotou uma função especificamente projetada para concatenar strings, como a função GROUP\_CONCAT do MySQL (uma das primeiras) ou STRING\_ADD (adicionada ao SQL Server recentemente no SQL Server 2017). Essas funções têm sintaxe semelhante e simplificam essa tarefa.

### DB2

Use LIST\_AGG para construir a lista delimitada:

```
1 select deptno,
2       list_agg(ename ',') within GROUP(Order by 0) as emps
3   from emp
4  group by deptno
```

### MySQL

Use a função interna GROUP\_CONCAT para construir a lista delimitada:

```
1 select deptno,
2       group_concat(ename order by empno separator ,',') as emps
3   from emp
4  group by deptno
```

### Oracle

Use a função integrada SYS\_CONNECT\_BY\_PATH para criar a lista delimitada:

```
1 select deptno,
2       ltrim(sys_connect_by_path(ename,',',''),',') emps
3   from (
4 select deptno,
5       ename,
6       row_number() over
7           (partition by deptno order by empno) rn,
8       count(*) over
9           (partition by deptno) cnt
10  from emp
11      )
12 where level = cnt
13 start with rn = 1
12 connect by prior deptno = deptno and prior rn = rn-1
```

```
1 select deptno,
2        string_agg(ename order by empno separator, ',') as emps
3   from emp
4  group by deptno
```

## Discussão

Ser capaz de criar listas delimitadas em SQL é útil porque é um requisito comum. O padrão SQL:2016 adicionou LIST\_AGG para executar esta tarefa, mas apenas o DB2 implementou esta função até agora. Felizmente, outros DBMS têm funções semelhantes, geralmente com sintaxe mais simples.

### MySQL

A função GROUP\_CONCAT no MySQL concatena os valores encontrados na coluna passada para ela, neste caso ENAME. É uma função agregada, por isso a necessidade de GROUP BY na consulta.

### PostgreSQL e SQL Server

A sintaxe da função STRING\_AGG é semelhante o suficiente para GROUP\_CONCAT para que a mesma consulta possa ser usada com o GROUP\_CONCAT simplesmente alterado para STRING\_AGG.

### Oracle

A primeira etapa para entender a consulta do Oracle é dividi-la. Executando a exibição em linha sozinha (linhas 4 a 10), você gera um conjunto de resultados que inclui o seguinte para cada funcionário: seu departamento, seu nome, uma classificação em seu respectivo departamento que é derivada de uma classificação crescente em EMPNO e um contagem de todos os funcionários de seu departamento. Por exemplo:

```
select deptno,
       ename,
       row_number() over
           (partition by deptno order by empno) rn,
       count(*) over (partition by deptno) cnt
  from emp
```

DEPTNO	ENAME	RN	CNT
10	CLARK	1	3
10	KING	2	3
10	MILLER	3	3
20	SMITH	1	5
20	JONES	2	5
20	SCOTT	3	5
20	ADAMS	4	5

20 FORD	5	5
30 ALLEN	1	6
30 WARD	2	6
30 MARTIN	3	6
30 BLAKE	4	6
30 TURNER	5	6
30 JAMES	6	6

A finalidade da classificação (aliased RN na consulta) é permitir que você percorra a árvore. Como a função ROW\_NUMBER gera uma enumeração começando de um sem duplicatas ou lacunas, apenas subtraia um (do valor atual) para referenciar uma linha anterior (ou pai). Por exemplo, o número antes de 3 é 3 menos 1, que é igual a 2. Nesse contexto, 2 é o pai de 3; você pode observar isso na linha 12. Além disso, as linhas:

```
start with rn = 1
connect by prior deptno = deptno
```

identifique a raiz para cada DEPTNO como tendo RN igual a 1 e crie uma nova lista sempre que um novo departamento for encontrado (sempre que uma nova ocorrência de 1 for encontrada para RN).

Neste ponto, é importante parar e observar a parte ORDER BY da função ROW\_NUMBER. Lembre-se de que os nomes são classificados por EMPNO e a lista será criada nessa ordem. O número de funcionários por departamento é calculado (alias CNT) e é usado para garantir que a consulta retorne apenas a lista que contém todos os nomes de funcionários de um departamento. Isso é feito porque SYS\_CONNECT\_BY\_PATH cria a lista iterativamente e você não deseja terminar com listas parciais.

Para consultas hierárquicas, a pseudocoluna LEVEL começa com 1 (para consultas que não usam CONNECT BY, LEVEL é 0, a menos que você esteja na versão 10g e posterior quando LEVEL está disponível apenas ao usar CONNECT BY) e incrementa em um após cada funcionário em um departamento foi avaliado (para cada nível de profundidade na hierarquia). Por isso, você sabe que quando o LEVEL chegar ao CNT, você alcançou o último EMPNO e terá uma lista completa.



A função SYS\_CONNECT\_BY\_PATH prefixa a lista com o delimitador escolhido (neste caso, uma vírgula). Você pode ou não querer esse comportamento. Na solução desta receita, a chamada para a função LTRIM remove a vírgula inicial da lista.

## 6.11 Convertendo dados delimitados em uma lista IN de vários valores

### Problema

Você delimitou dados que deseja passar para o iterador da lista IN de uma cláusula WHERE. Considere a seguinte cadeia:

```
7654,7698,7782,7788
```

Você gostaria de usar a string em uma cláusula WHERE, mas o seguinte SQL falha porque EMPNO é uma coluna numérica:

```
select ename,sal,deptno
  from emp
 where empno in ( '7654,7698,7782,7788' )
```

Este SQL falha porque, enquanto EMPNO é uma coluna numérica, a lista IN é composta por um único valor de string. Você deseja que essa string seja tratada como uma lista delimitada por vírgulas de valores numéricos.

### Solução

Superficialmente, pode parecer que o SQL deve fazer o trabalho de tratar uma string delimitada como uma lista de valores delimitados para você, mas esse não é o caso. Quando uma vírgula entre aspas é encontrada, o SQL não pode saber que sinaliza uma lista de vários valores. SQL deve tratar tudo entre aspas como uma única entidade, como um valor de string. Você deve dividir a string em EMPNOs individuais. A chave para essa solução é percorrer a string, mas não em caracteres individuais. Você deseja percorrer a string em valores válidos EMPNO.

### DB2

Percorrendo a string passada para a lista IN, você pode facilmente convertê-la em linhas. As funções ROW\_NUMBER, LOCATE e SUBSTR são particularmente úteis aqui:

```
1 select empno,ename,sal,deptno
2   from emp
3  where empno in (
4 select cast(substr(c,2,locate(',',$c,2)-2) as integer) empno
5   from (
6 select substr(csv.emps,cast(iter.pos as integer)) as c
7 from (select ''||'7654,7698,7782,7788'||',' emps
8           from t1) csv,
9           (select id as pos
10          from t100 ) iter
11 where iter.pos <= length(csv.emps)
12      ) x
13 where length(c) > 1
14     and substr(c,1,1) = ','
15      )
```

## MySQL

Ao percorrer a string passada para a lista IN, você pode convertê-la facilmente em linhas:

```

1 select empno, ename, sal, deptno
2   from emp
3 where empno in
4 (
5 select substring_index(
6   substring_index(list.vals,',',iter.pos),',',-1) empno
7   from (select id pos from t10) as iter,
8         (select '7654,7698,7782,7788' as vals
9          from t1) list
10 where iter.pos <=
10      (length(list.vals)-length(replace(list.vals,',','')))+1
12 )

```

## Oracle

Percorrendo a string passada para a lista IN, você pode facilmente convertê-la em linhas. As funções ROWNUM, SUBSTR e INSTR são particularmente úteis aqui:

```

1 select empno,ename,sal,deptno
2   from emp
3 where empno in (
4     select to_number(
5       rtrim(
6         substr(emps,
7           instr(emps,',',1,iter.pos)+1,
8           instr(emps,',',1,iter.pos+1)
9             -instr(emps,',',1,iter.pos)),','))
10    emps
10   from (select ','||'7654,7698,7782,7788'||',' emps from t1) csv,
11        (select rownum pos from emp) iter
12   where iter.pos <= ((length(csv.emps)-
11           length(replace(csv.emps,',')))/length(','))-1
14)

```

## PostgreSQL

Percorrendo a string passada para a lista IN, você pode facilmente convertê-la em linhas. A função SPLIT\_PART facilita a análise da string em números individuais:

```

1 select ename,sal,deptno
2   from emp
3 where empno in (
4 select cast(empno as integer) as empno
5   from (
6 select split_part(list.vals,',',iter.pos) as empno
7   from (select id as pos from t10) iter,
8         (select ','||'7654,7698,7782,7788'||',' as vals
9          from t1) list
10 where iter.pos <=
10      length(list.vals)-length(replace(list.vals,',',''))

```

```
11 ) z  
12 where length(empno) > 0  
14 )
```

## SQL Server

Percorrendo a string passada para a lista IN, você pode facilmente convertê-la em linhas. As funções ROW\_NUMBER, CHARINDEX e SUBSTRING são particularmente úteis aqui:

```
1 select empno,ename,sal,deptno  
2   from emp  
3 where empno in (select substring(c,2,charindex(',',c,2)-2) as empno  
4   from (  
5 select substring(csv.emps,iter.pos,len(csv.emps)) as c  
6 from (select ','+'7654,7698,7782,7788'+',' as emps  
7      from t1) csv,  
8      (select id as pos  
9       from t100) iter  
10 where iter.pos <= len(csv.emps)  
11      ) x  
12 where len(c) > 1  
13   and substring(c,1,1) = ','  
14      )
```

## Discussão

O primeiro e mais importante passo nesta solução é andar na corda. Depois de fazer isso, tudo o que resta é analisar a string em valores numéricos individuais usando as funções fornecidas pelo seu DBMS.

### DB2 e SQLServer

A visualização inline X (linhas 6–11) percorre a string. A ideia nesta solução é “caminhar através” da string de forma que cada linha tenha um caractere a menos que a anterior:

```
,7654,7698,7782,7788,  
7654,7698,7782,7788,  
654,7698,7782,7788,  
54,7698,7782,7788,  
4,7698,7782,7788,  
,7698,7782,7788,  
7698,7782,7788,  
698,7782,7788,  
98,7782,7788,  
8,7782,7788,  
,7782,7788,  
7782,7788,  
782,7788,  
82,7788,  
2,7788,
```

```
,7788,
7788,
788,
88,
8,
,
```

Observe que, colocando a string entre vírgulas (o delimitador), não há necessidade de fazer verificações especiais para saber onde está o início ou o fim da string.

O próximo passo é manter apenas os valores que você deseja usar na lista IN. Os valores a serem mantidos são aqueles com vírgulas à esquerda, com exceção da última linha com sua única vírgula. Use SUBSTR ou SUBSTRING para identificar quais linhas possuem uma vírgula à esquerda e, em seguida, mantenha todos os caracteres encontrados antes da próxima vírgula nessa linha. Feito isso, converta a string em um número para que ela possa ser avaliada adequadamente em relação à coluna numérica EMPNO (linhas 4 a 14):

```
EMPNO
-----
7654
7698
7782
7788
```

A etapa final é usar os resultados em uma subconsulta para retornar as linhas desejadas.

## MySQL

A visualização inline (linhas 5–9) percorre a string. A expressão na linha 10 determina quantos valores estão na string encontrando o número de vírgulas (o delimitador) e adicionando uma. A função SUBSTRING\_INDEX (linha 6) retorna todos os caracteres na string antes (à esquerda de ) da  $n^{\text{a}}$  ocorrência de vírgula (o delimitador):

```
+-----+
| empno |
+-----+
| 7654 |
| 7654,7698 |
| 7654,7698,7782 |
| 7654,7698,7782,7788 |
+-----+
```

Essas linhas são então passadas para outra chamada para SUBSTRING\_INDEX (linha 5); desta vez o  $n$  enésima ocorrência do delimitado é  $-1$ , o que faz com que todos os valores à direita da  $n^{\text{a}}$  ocorrência do delimitador a ser mantido:

```
+-----+
| empno |
+-----+
| 7654 |
| 7698 |
| 7782 |
| 7788 |
+-----+
```

A etapa final é inserir os resultados em uma subconsulta.

## Oracle

O primeiro passo é percorrer a string:

```
select emps,pos
  from (select ','||'7654,7698,7782,7788'||',' emps
        from t1) csv,
       (select rownum pos from emp) iter
  where iter.pos <=
((length(csv.emps)-length(replace(csv.emps,',')))/length(','))-1
```

EMPS	POS
,7654,7698,7782,7788,	1
,7654,7698,7782,7788,	2
,7654,7698,7782,7788,	3
,7654,7698,7782,7788,	4

O número de linhas retornadas representa o número de valores em sua lista. Os valores para POS são cruciais para a consulta, pois são necessários para analisar a string em valores individuais. As strings são analisadas usando SUBSTR e INSTR. POS é usado para localizar a  $n$  enésima ocorrência do delimitador em cada string. Ao colocar as strings entre vírgulas, nenhuma verificação especial é necessária para determinar o início ou o fim de uma string. Os valores passados para SUBSTR e INSTR (linhas 7–9) localizam a  $n^{\text{a}}$  e  $n^{\text{a}}+1$  ocorrência do delimitador. Ao subtrair o valor retornado para a vírgula atual (o local na string onde está a vírgula atual) do valor retornado pela próxima vírgula (o local na string onde está a próxima vírgula), você pode extrair cada valor da string:

```
select substr(emps,
            instr(emps,',',1,iter.pos)+1,
            instr(emps,',',1,iter.pos+1)
            instr(emps,',',1,iter.pos)) emps
  from (select ','||'7654,7698,7782,7788'||',' emps
        from t1) csv,
       (select rownum pos from emp) iter
  where iter.pos <=
((length(csv.emps)-length(replace(csv.emps,',')))/length(','))-1
```

```
EMPS
-----
7654,
7698,
7782,
7788,
```

A etapa final é remover a vírgula final de cada valor, convertê-la em um número e conectá-la a uma subconsulta.

### PostgreSQL

A visualização inline Z (linhas 6–9) percorre a string. O número de linhas retornadas é determinado por quantos valores estão na string. Para encontrar o número de valores na string, subtraia o tamanho da string sem o delimitador do tamanho da string com o delimitador (linha 9). A função SPLIT\_PART faz o trabalho de analisar a string. Ele procura o valor que vem antes da ocorrência do delimitador:

```
select list.vals,
       split_part(list.vals,',',iter.pos) as empno,
       iter.pos
  from (select id as pos from t10) iter,
       (select ','||'7654,7698,7782,7788'||',' as vals
        from t1) list
 where iter.pos <=
       length(list.vals)-length(replace(list.vals,',',''))
```

vals	empno	pos
,7654,7698,7782,7788,		1
,7654,7698,7782,7788,	7654	2
,7654,7698,7782,7788,	7698	3
,7654,7698,7782,7788,	7782	4
,7654,7698,7782,7788,	7788	5

A etapa final é converter os valores (EMPNO) em um número e conectar-lo a uma subconsulta.

## 6.12 Alfabetização de uma String

### Problema

Você deseja colocar em ordem alfabética os caracteres individuais dentro das strings em suas tabelas. Considere o seguinte conjunto de resultados:

```
ENAME
-----
ADAMS
ALLEN
BLAKE
CLARK
```

FORD  
JAMES  
JONES  
KING  
MARTIN  
MILLER  
SCOTT  
SMITH  
TURNER  
WARD

Você gostaria que o resultado fosse:

OLD_NAME	NEW_NAME
ADAMS	AADMS
ALLEN	AELLN
BLAKE	ABEKL
CLARK	ACKLR
FORD	DFOR
JAMES	AEJMS
JONES	EJNOS
KING	GIKN
MARTIN	AIMNRT
MILLER	EILLMR
SCOTT	COSTT
SMITH	HIMST
TURNER	ENRRTU
WARD	ADRW

## Solução

Esse problema é um bom exemplo de como o aumento da padronização permite soluções mais semelhantes e, portanto, portáteis.

### DB2

Para colocar em ordem alfabética linhas de strings, é necessário percorrer cada string e depois ordenar seus caracteres:

```
1 select ename,
2 listagg(c,'') WITHIN GROUP( ORDER BY c)
3 from (
4 select a.ename,
5 substr(a.ename,iter.pos,1
6 ) as c
7 from emp a,
8 (select id as pos from t10) iter
9 where iter.pos <= length(a.ename)
10order by 1,2
11) x
12Group By c
```

## MySQL

A chave aqui é a função GROUP\_CONCAT, que permite não apenas concatenar os caracteres que compõem cada nome, mas também ordená-los:

```

1 select ename, group_concat(c order by c separator '')
2   from (
3 select ename, substr(a.ename,iter.pos,1) c
4   from emp a,
5        ( select id pos from t10 ) iter
6  where iter.pos <= length(a.ename)
7  ) x
8 group by ename

```

## Oracle

A função SYS\_CONNECT\_BY\_PATH permite construir iterativamente uma lista:

```

1 select old_name, new_name
2   from (
3 select old_name, replace(sys_connect_by_path(c,' '), ' ') new_name
4   from (
5 select e.ename old_name,
6       row_number() over(partition by e.ename
7                          order by substr(e.ename,iter.pos,1)) rn,
8       substr(e.ename,iter.pos,1) c
9   from emp e,
10      ( select rownum pos from emp ) iter
11  where iter.pos <= length(e.ename)
12  order by 1
13  ) x
14 start with rn = 1
15 connect by prior rn = rn-1 and prior old_name = old_name
16 )
17 where length(old_name) = length(new_name)

```

## PostgreSQL

O PostgreSQL agora adicionou STRING\_AGG para ordenar os caracteres dentro de uma string.

```

select ename, string_agg(c , ''
                         ORDER BY c)
from (
      select a.ename,
             substr(a.ename,iter.pos,1) as c
      from emp a,
           (select id as pos from t10) iter
     where iter.pos <= length(a.ename)
     order by 1,2
      ) x
      Group By c

```

Se você estiver usando o SQL Server 2017 ou superior, a solução PostgreSQL com STRING\_AGG funcionará. Caso contrário, para alfabetizar linhas de strings, é necessário percorrer cada string e depois ordenar seus caracteres:

```
1 select ename,
2       max(case when pos=1 then c else '' end) +
3       max(case when pos=2 then c else '' end) +
4       max(case when pos=3 then c else '' end) +
5       max(case when pos=4 then c else '' end) +
6       max(case when pos=5 then c else '' end) +
7       max(case when pos=6 then c else '' end)
8   from (
9    select e.ename,
10          substring(e.ename,iter.pos,1) as c,
11          row_number() over (
12              partition by e.ename
13                  order by substring(e.ename,iter.pos,1)) as pos
14   from emp e,
15        (select row_number()over(order by ename) as pos
16         from emp) iter
17  where iter.pos <= len(e.ename)
18      ) x
19 group by ename
```

## Discussão

### SQL Server

A exibição inline X retorna cada caractere em cada nome como uma linha. A função SUBSTR ou SUBSTRING extrai cada caractere de cada nome, e a função ROW\_NUMBER classifica cada caractere em ordem alfabética:

ENAME	C	POS
ADAMS	A	1
ADAMS	A	2
ADAMS	D	3
ADAMS	M	4
ADAMS	S	5
...		

Para retornar cada letra de uma string como uma linha, você deve percorrer a string. Isso é feito com o ITER de exibição em linha.

Agora que as letras de cada nome foram colocadas em ordem alfabética, o último passo é juntar essas letras, em uma string, na ordem em que estão classificadas. A posição de cada letra é avaliada pelas instruções CASE (linhas 2–7). Se um caractere for encontrado em uma posição específica, ele será concatenado ao resultado da próxima avaliação (a seguinte instrução CASE).

Como a função de agregação MAX também é usada, apenas um caractere por posição POS é retornado para que apenas uma linha por nome seja retornada. O CASE de avaliação vai até o número seis, que é o número máximo de caracteres em qualquer nome na tabela EMP.

## MySQL

A exibição inline X (linhas 3–6) retorna cada caractere em cada nome como uma linha. A função SUBSTR extraí cada caractere de cada nome:

```
ENAME  C
-----
ADAMS  A
ADAMS  A
ADAMS  D
ADAMS  M
ADAMS  S
...

```

Visualização em linha ITER é usado para percorrer a cadeia. A partir daí, o restante do trabalho é feito pela função GROUP\_CONCAT. Ao especificar uma ordem, a função não apenas concatena cada letra, mas também em ordem alfabética.

## Oracle

O trabalho real é feito pela visualização em linha X (linhas 5 a 11), onde os caracteres de cada nome são extraídos e colocados em ordem alfabética. Isso é feito percorrendo a string e, em seguida, impondo ordem a esses caracteres. O restante da consulta apenas cola os nomes novamente.

A separação de nomes pode ser vista executando apenas a visualização inline X:

```
OLD_NAME      RN  C
-----
ADAMS          1  A
ADAMS          2  A
ADAMS          3  D
ADAMS          4  M
ADAMS          5  S
...

```

O próximo passo é pegar os caracteres alfabéticos e reconstruir cada nome. Isso é feito com a função SYS\_CONNECT\_BY\_PATH anexando cada caractere aos anteriores:

OLD_NAME	NEW_NAME
ADAMS	A
ADAMS	AA
ADAMS	AAD
ADAMS	AADM
ADAMS	AADMS
...	

A etapa final é manter apenas as strings com o mesmo comprimento dos nomes a partir dos quais foram construídas.

### PostgreSQL

Para facilitar a leitura, a visualização V é usada nesta solução para percorrer a string. A função SUBSTR, na definição da view, extrai cada caractere de cada nome para que a view retorne:

ENAME	C
ADAMS	A
ADAMS	A
ADAMS	D
ADAMS	M
ADAMS	S
...	

A exibição também ordena os resultados por ENAME e por cada letra em cada nome. A visualização inline X (linhas 15–18) retorna os nomes e caracteres da visualização V, o número de vezes que cada caractere ocorre em cada nome e sua posição (em ordem alfabética):

ename	c	cnt	pos
ADAMS	A	2	1
ADAMS	A	2	1
ADAMS	D	1	3
ADAMS	M	1	4
ADAMS	S	1	5

As colunas extras CNT e POS, retornadas pela visualização inline X, são cruciais para a solução. POS é usado para classificar cada caractere e CNT é usado para determinar o número de vezes que o caractere existe em cada nome. A etapa final é avaliar a posição de cada personagem e reconstruir o nome. Você notará que cada declaração case é, na verdade, duas declarações case. Isso é para determinar se um caractere ocorre mais de uma vez em um nome; se isso acontecer, em vez de retornar esse caractere, o que é retornado é esse caractere anexado a si mesmo CNT vezes. A função agregada, MAX, é usada para garantir que haja apenas uma linha por nome.

## 6.13 Identificando strings que podem ser tratadas como números

### Problema

Você tem uma coluna definida para armazenar dados de caracteres. Infelizmente, as linhas contêm dados numéricos e de caracteres mistos. Considere a visão V:

```
create view V as
select replace(mixed, ' ', '') as mixed
  from (
select substr(ename,1,2) ||
       cast(deptno as char(4)) ||
       substr(ename,3,2) as mixed
  from emp
 where deptno = 10
 union all
select cast(empno as char(4)) as mixed
  from emp
 where deptno = 20
 union all
select ename as mixed
  from emp
 where deptno = 30
      ) x
select * from v

MIXED
-----
CL10AR
KI10NG
MI10LL
7369
7566
7788
7876
7902
ALLEN
WARD
MARTIN
BLAKE
TURNER
JAMES
```

Você deseja retornar linhas que sejam apenas números ou que contenham pelo menos um número. Se os números estiverem misturados com dados de caracteres, você deseja remover os caracteres e retornar apenas os números. Para os dados de amostra mostrados anteriormente, você deseja o seguinte conjunto de resultados:

```
-----  
10  
10  
10  
7369  
7566  
7788  
7876  
7902
```

## Solução

As funções REPLACE e TRANSLATE são extremamente úteis para manipular strings e caracteres individuais. A chave é converter todos os números em um único caractere, o que torna mais fácil isolar e identificar qualquer número referindo-se a um único caractere.

### DB2

Use as funções TRANSLATE, REPLACE e POSSTR para isolar os caracteres numéricos em cada linha. As chamadas ao CAST são necessárias na visão V; caso contrário, a exibição não será criada devido a erros de conversão de tipo. Você precisará da função REPLACE para remover espaços em branco estranhos devido à conversão para o CHAR de comprimento fixo:

```
1 select mixed.old,  
2      cast(  
3          case  
4              when  
5                  replace(  
6                      translate(mixed,'9999999999','0123456789'), '9', '') = ''  
7                  then  
8                      mixed  
9                  else replace(  
10                     translate(mixed,  
11                         repeat('#',length(mixed)),  
12                         replace(  
13                             translate(mixed,'9999999999','0123456789'), '9', ''),  
14                             '#', ''))  
15                  end as integer ) mixed  
16      from V  
17      where posstr(translate(mixed,'9999999999','0123456789'), '9') > 0
```

### MySQL

A sintaxe do MySQL é um pouco diferente e definirá a visão V como:

```
create view V as  
select concat(  
    substr(ename,1,2),  
    replace(cast(deptno as char(4)), ' ', ''),
```

```

        substr(ename,3,2)
    ) as mixed
  from emp
 where deptno = 10
 union all
select replace(cast(empno as char(4)), ' ', '')
  from emp where deptno = 20
 union all
select ename from emp where deptno = 30

```

Como o MySQL não suporta a função TRANSLATE, você deve percorrer cada linha e avaliá-la caractere por caractere.

```

1 select cast(group_concat(c order by pos separator '') as unsigned)
2      as MIXED1
3  from (
4 select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
5   from V,
6        ( select id pos from t10 ) iter
7  where iter.pos <= length(v.mixed)
8    and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
9    ) y
10 group by mixed
11 order by 1

```

## Oracle

Use as funções TRANSLATE, REPLACE e INSTR para isolar os caracteres numéricos em cada linha. As chamadas para CAST não são necessárias na visão V. Use a função REPLACE para remover espaços em branco estranhos devido à conversão para o CHAR de comprimento fixo. Se você decidir que gostaria de manter as chamadas de conversão de tipo explícito na definição de visualização, sugerimos que você converta para VARCHAR2:

```

1 select to_number (
2       case
3         when
4           replace(transliterate(mixed,'0123456789','9999999999'),'9')
5           is not null
6         then
7           replace(
8             transliterate(mixed,
9               replace(
10                transliterate(mixed,'0123456789','9999999999'),'9'),
11                  rpad('#',length(mixed),'#')),'#')
12       else
13         mixed
14       end
15     ) mixed
16   from V
17  where instr(transliterate(mixed,'0123456789','9999999999'),'9') > 0

```

Use as funções TRANSLATE, REPLACE e STRPOS para isolar os caracteres numéricos em cada linha. As chamadas para CAST não são necessárias na visão V. Use a função REPLACE para remover espaços em branco estranhos devido à conversão para o CHAR de comprimento fixo. Se você decidir que gostaria de manter as chamadas de conversão de tipo explícito na definição de visualização, sugerimos que você converta para VARCHAR:

```
1 select cast(
2      case
3      when
4          replace(translade(mixed,'0123456789','9999999999'),'9','')
5          is not null
6      then
7          replace(
8              translade(mixed,
9                  replace(
10                 translade(mixed,'0123456789','9999999999'),'9',''),
11                 rpad('#',length(mixed),'#')), '#', '')
12      else
13          mixed
14      end as integer ) as mixed
15  from V
16 where strpos(translade(mixed,'0123456789','9999999999'),'9') > 0
```

## SQL Server

A função integrada ISNUMERIC junto com uma pesquisa curinga permite que você identifique facilmente strings que contêm números, mas obter caracteres numéricos de uma string não é particularmente eficiente porque a função TRANSLATE não é suportada.

## Discussão

A função TRANSLATE é útil aqui, pois permite isolar e identificar facilmente números e caracteres. O truque é converter todos os números em um único caractere; dessa forma, em vez de pesquisar números diferentes, você pesquisa apenas um caractere.

### DB2, Oracle e PostgreSQL

A sintaxe difere ligeiramente entre esses DBMSs, mas a técnica é a mesma. Usaremos a solução para PostgreSQL para a discussão.

O trabalho real é feito pelas funções TRANSLATE e REPLACE. Obter o conjunto de resultados final requer várias chamadas de função, cada uma listada aqui em uma consulta:

```
select mixed as orig,
       translade(mixed,'0123456789','9999999999') as mixed1,
       replace(translade(mixed,'0123456789','9999999999'),'9','') as mixed2,
       translade(mixed,
       replace (
```

```

translate(mixed,'0123456789','9999999999'), '9', ''),
    rpad('#',length(mixed), '#')) as mixed3,
replace(
translate(mixed,
replace(
translate(mixed,'0123456789','9999999999'), '9', ''),
    rpad('#',length(mixed), '#')),'#', '') as mixed4
from v
where strpos(translate(mixed,'0123456789','9999999999'), '9') > 0

```

ORIG	MIXED1	MIXED2	MIXED3	MIXED4	MIXED5
CL10AR	CL99AR	CLAR	##10##	10	10
KI10NG	KI99NG	KING	##10##	10	10
MI10LL	MI99LL	MILL	##10##	10	10
7369	9999		7369	7369	7369
7566	9999		7566	7566	7566
7788	9999		7788	7788	7788
7876	9999		7876	7876	7876
7902	9999		7902	7902	7902

Primeiro, observe que todas as linhas sem pelo menos um número são removidas. Como isso é feito ficará claro conforme você examina cada uma das colunas no conjunto de resultados anterior. As linhas que são mantidas são os valores na coluna ORIG e são as linhas que eventualmente comporão o conjunto de resultados. O primeiro passo para extrair os números é usar a função TRANSLATE para converter qualquer número para um 9 (você pode usar qualquer dígito; 9 é arbitrário); isso é representado pelos valores em MIXED1. Agora que todos os números são 9s, eles podem ser tratados como uma única unidade. O próximo passo é remover todos os números usando a função REPLACE. Como todos os dígitos agora são 9, REPLACE simplesmente procura quaisquer 9s e os remove. Isso é representado pelos valores em MIXED2. A próxima etapa, MIXED3, usa valores que são retornados por MIXED2. Esses valores são então comparados com os valores no ORIG. Se quaisquer caracteres de MIXED2 forem encontrados em ORIG, eles serão convertidos para o caractere # por TRANSLATE. O conjunto de resultados do MIXED3 mostra que as letras, não os números, agora foram separados e convertidos em um único caractere. Agora que todos os caracteres não numéricos são representados por #s, eles podem ser tratados como uma única unidade. A próxima etapa, MIXED4, usa REPLACE para localizar e remover quaisquer caracteres # em cada linha; o que resta são apenas números. A etapa final é converter os caracteres numéricos como números. Agora que você passou pelas etapas, pode ver como funciona a cláusula WHERE. Os resultados de MIXED1 são passados para STRPOS, e se um 9 for encontrado (a posição na string onde o primeiro 9 está localizado), o resultado deve ser maior que 0. Para linhas que retornam um valor maior que zero, significa que não há pelo menos um número nessa linha e deve ser mantido.

## MySQL

O primeiro passo é percorrer cada string, avaliar cada caractere e determinar se é um número:

```

select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
from V,
     ( select id pos from t10 ) iter
where iter.pos <= length(v.mixed)
order by 1,2

+-----+-----+-----+
| mixed | pos  | c    |
+-----+-----+-----+
| 7369  | 1    | 7    |
| 7369  | 2    | 3    |
| 7369  | 3    | 6    |
| 7369  | 4    | 9    |
...
| ALLEN | 1    | A    |
| ALLEN | 2    | L    |
| ALLEN | 3    | L    |
| ALLEN | 4    | E    |
| ALLEN | 5    | N    |
...
| CL10AR | 1    | C    |
| CL10AR | 2    | L    |
| CL10AR | 3    | 1    |
| CL10AR | 4    | 0    |
| CL10AR | 5    | A    |
| CL10AR | 6    | R    |
+-----+-----+-----+

```

Agora que cada caractere em cada string pode ser avaliado individualmente, o próximo passo é manter apenas as linhas que possuem um número na coluna C:

```

select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
from V,
     ( select id pos from t10 ) iter
where iter.pos <= length(v.mixed)
and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
order by 1,2

+-----+-----+-----+
| mixed | pos  | c    |
+-----+-----+-----+
| 7369  | 1    | 7    |
| 7369  | 2    | 3    |
| 7369  | 3    | 6    |
| 7369  | 4    | 9    |
...
| CL10AR | 3    | 1    |
| CL10AR | 4    | 0    |
...
+-----+-----+-----+

```

Neste ponto, todas as linhas na coluna C são números. O próximo passo é usar GROUP\_CONCAT para concatenar os números para formar seus respectivos números inteiros em MIXED. O resultado final é então convertido como um número:

```
select cast(group_concat(c order by pos separator '') as unsigned)
as MIXED1
from (
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
from V,
( select id pos from t10 ) iter
where iter.pos <= length(v.mixed)
and ascii(substr(x.mixed,iter.pos,1)) between 48 and 57
) y
group by mixed
order by 1

+-----+
| MIXED1 |
+-----+
|   10  |
|   10  |
|   10  |
| 7369 |
| 7566 |
| 7788 |
| 7876 |
| 7902 |
+-----+
```

Como observação final, lembre-se de que quaisquer dígitos em cada string serão concatenados para formar um valor numérico. Por exemplo, um valor de entrada de, digamos, 99Gennick87 resultará no retorno do valor 9987. Isso é algo para se ter em mente, principalmente ao trabalhar com dados serializados.

## 6.14 Extraindo a enésima substring delimitada

### Problema

Você deseja extrair uma substring especificada e delimitada de uma string. Considere a seguinte visão V, que gera dados de origem para este problema:

```
create view V as
select 'mo,larry,curly' as name
from t1
union all
select 'tina,gina,jaunita,regina,leena' as name
from t1
```

A saída da exibição é a seguinte:

```
select * from v

NAME
-----
mo,larry,curly
tina,gina,jaunita,regina,leena
```

Você gostaria de extrair o segundo nome em cada linha, então o conjunto de resultados final seria o seguinte:

```
SUB
-----
larry
gina
```

## Solução

A chave para resolver esse problema é retornar cada nome como uma linha individual enquanto preserva a ordem em que o nome existe na lista. Exatamente como você faz essas coisas depende de qual DBMS você está usando.

### DB2

Após percorrer os NAMES retornados pela view V, utilize a função ROW\_NUMBER para manter apenas o segundo nome de cada string:

```
1 select substr(c,2,locate(',',$c,2)-2)
2   from (
3 select pos, name, substr(name, pos) c,
4       row_number() over( partition by name
5                           order by length(substr(name,pos)) desc) rn
6   from (
7 select ',' ||csv.name|| ',' as name,
8       cast(iter.pos as integer) as pos
9   from V csv,
10      (select row_number() over() pos from t100 ) iter
11 where iter.pos <= length(csv.name)+2
12     ) x
13 where length(substr(name,pos)) > 1
14   and substr(substr(name,pos),1,1) = ','
15     ) y
16 where rn = 2
```

### MySQL

Após percorrer os NAMES retornados pela view V, utilize a posição das vírgulas para retornar apenas o segundo nome em cada string:

```

1 select name
2   from (
3 select iter.pos,
4       substring_index(
5           substring_index(src.name,',',iter.pos),',',-1) name
6   from V src,
7       (select id pos from t10) iter,
8 where iter.pos <=
9     length(src.name)-length(replace(src.name,',',''))
10    ) x
11 where pos = 2

```

## Oracle

Após percorrer os NAMES retornados pela visualização V, recupere o segundo nome em cada lista usando SUBSTR e INSTR:

```

1 select sub
2   from (
3 select iter.pos,
4       src.name,
5       substr( src.name,
6             instr( src.name,',',1,iter.pos )+1,
7             instr( src.name,',',1,iter.pos+1 ) -
8             instr( src.name,',',1,iter.pos )-1) sub
9   from (select ',||name||,' as name from V) src,
10       (select rownum pos from emp) iter
11  where iter.pos < length(src.name)-length(replace(src.name,','))
12  )
13 where pos = 2

```

## PostgreSQL

Use a função SPLIT\_PART para ajudar a retornar cada nome individual como uma linha:

```

1 select name
2   from (
3 select iter.pos, split_part(src.name,',',iter.pos) as name
4   from (select id as pos from t10) iter,
5       (select cast(name as text) as name from v) src
7 where iter.pos <=
8     length(src.name)-length(replace(src.name,',',''))+1
9    ) x
10 where pos = 2

```

## SQL Server

A função SQL Server STRING\_SPLIT fará todo o trabalho, mas só pode ocupar uma única célula. Portanto, usamos um STRING\_AGG dentro de um CTE para apresentar os dados da maneira que STRING\_SPLIT exige.

```

1 with agg_tab(name)
2     as
3         (select STRING_AGG(name,',') from V)
4 select value from
5     STRING_SPLIT(
6         (select name from agg_tab),',')

```

## Discussão

### DB2

A sintaxe é ligeiramente diferente entre esses dois DBMSs, mas a técnica é a mesma. Usaremos a solução para DB2 para a discussão. As strings são percorridas e os resultados são representados pela visualização inline X:

```

select ','||csv.name|| ',' as name,
       iter.pos
  from v csv,
       (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2

EMPS                      POS
-----  -----
,tina,gina,jaunita,regina,leena,    1
,tina,gina,jaunita,regina,leena,    2
,tina,gina,jaunita,regina,leena,    3
...

```

O próximo passo é percorrer cada caractere em cada string:

```

select pos, name, substr(name, pos) c,
       row_number() over(partition by name
                          order by length(substr(name, pos)) desc) rn
  from (
select ','||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
       (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
       ) x
 where length(substr(name,pos)) > 1

POS EMPS          C          RN
-----  -----
1 ,mo,larry,curly, ,mo,larry,curly, 1
2 ,mo,larry,curly, mo,larry,curly, 2
3 ,mo,larry,curly, o,larry,curly, 3
4 ,mo,larry,curly, ,larry,curly, 4
...

```

Agora que diferentes partes da string estão disponíveis para você, simplesmente identifique quais linhas manter. As linhas nas quais você está interessado são aquelas que começam com uma vírgula; o resto pode ser descartado:

```
select pos, name, substr(name,pos) c,
       row_number() over(partition by name
                          order by length(substr(name, pos)) desc) rn
  from (
select ','||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
       (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
      ) x
where length(substr(name,pos)) > 1
   and substr(substr(name,pos),1,1) = ','
```

POS	EMPS	C	RN
1	,mo,larry,curly,	,mo,larry,curly,	1
4	,mo,larry,curly,	,larry,curly,	2
10	,mo,larry,curly,	,curly,	3
1	,tina,gina,jaunita,regina,leena,	,tina,gina,jaunita,regina,leena,	1
6	,tina,gina,jaunita,regina,leena,	,gina,jaunita,regina,leena,	2
11	,tina,gina,jaunita,regina,leena,	,jaunita,regina,leena,	3
19	,tina,gina,jaunita,regina,leena,	,regina,leena,	4
26	,tina,gina,jaunita,regina,leena,	,leena,	5

Este é um passo importante, pois define como você obterá *on<sup>a</sup>* substring. Observe que muitas linhas foram eliminadas dessa consulta devido à seguinte condição na cláusula WHERE:

```
substr(substr(name,pos),1,1) = ','
```

Você notará que ,mo,larry,encaracolado, era classificado como 4, mas agora é classificado como 2. Lembre-se, a cláusula WHERE é avaliada antes do SELECT, então as linhas com vírgulas à esquerda são mantidas, então ROW\_NUMBER realiza sua classificação. Neste ponto é fácil ver que, para obter o *n*th substring, você quer linhas onde RN é igual *n*. O último passo é manter apenas as linhas de seu interesse (neste caso onde RN é igual a dois) e usar SUBSTR para extrair o nome dessa linha. O nome a manter é o primeiro nome da linha: Larry de, larry,curly, e gina de ,gina,jaunita,regina,leena,.

## MySQL

A visualização inline X percorre cada string. Você pode determinar quantos valores existem em cada string contando os delimitadores na string:

```
select iter.pos, src.name
  from (select id pos from t10) iter,
       V src
```

```

where iter.pos <=
length(src.name)-length(replace(src.name,',',''))

```

pos	name
1	mo,larry,curly
2	mo,larry,curly
1	tina,gina,jaunita,regina,leena
2	tina,gina,jaunita,regina,leena
3	tina,gina,jaunita,regina,leena
4	tina,gina,jaunita,regina,leena

Nesse caso, há uma linha a menos do que valores em cada string porque isso é tudo o que é necessário. A função SUBSTRING\_INDEX cuida da análise dos valores necessários:

```

select iter.pos,src.name name1,
       substring_index(src.name,',',iter.pos) name2,
       substring_index(
           substring_index(src.name,',',iter.pos),',',-1) name3
  from (select id pos from t10) iter,
       V src
 where iter.pos <=
length(src.name)-length(replace(src.name,',',''))

```

pos	name1	name2	name3
1	mo,larry,curly	mo	mo
2	mo,larry,curly	mo,larry	larry
1	tina,gina,jaunita,regina,leena	tina	tina
2	tina,gina,jaunita,regina,leena	tina,gina	gina
3	tina,gina,jaunita,regina,leena	tina,gina,jaunita	jaunita
4	tina,gina,jaunita,regina,leena	tina,gina,jaunita,regina	regina

Mostramos três campos de nome para que você possa ver como funcionam as chamadas SUBSTRING\_INDEX aninhadas. A chamada interna retorna todos os caracteres à esquerda do  $n^{\text{a}}$  ocorrência de vírgula. A chamada externa retorna tudo à direita da primeira vírgula que encontra (começando no final da string). A etapa final é manter o valor para NAME3 onde POS é igual a  $n$ , neste caso 2.

## SQL Server

STRING\_SPLIT é o burro de carga aqui, mas precisa de seus dados da maneira certa. O CTE serve apenas para transformar as duas linhas da coluna V.names em um único valor, conforme exigido por STRING\_SPLIT ser uma função com valor de tabela.

## Oracle

A visualização inline percorre cada string. O número de vezes que cada string é retornada é determinado por quantos valores estão em cada string. A solução encontra o número de valores em cada string contando o número de delimitadores nela. Como cada string é colocada entre vírgulas, o número de valores em uma string é o número de vírgulas menos um. As strings são, então, UNIONed e unidas a uma tabela com uma cardinalidade que é pelo menos o número de valores na maior string. As funções SUBSTR e INSTR usam o valor de POS para analisar cada string:

```
select iter.pos, src.name,
       substr( src.name,
              instr( src.name, ',',1,iter.pos )+1,
              instr( src.name, ',',1,iter.pos+1 )
              instr( src.name, ',',1,iter.pos )-1) sub
  from (select ','||name||',' as name from v) src,
       (select rownum pos from emp) iter
 where iter.pos < length(src.name)-length(replace(src.name,','))
```

POS	NAME	SUB
1	,mo,larry,curly,	mo
1	, tina,gina,jaunita,regina,leena,	tina
2	,mo,larry,curly,	larry
2	, tina,gina,jaunita,regina,leena,	gina
3	,mo,larry,curly,	curly
3	, tina,gina,jaunita,regina,leena,	jaunita
4	, tina,gina,jaunita,regina,leena,	regina
5	, tina,gina,jaunita,regina,leena,	leena

A primeira chamada para INSTR dentro de SUBSTR determina a posição inicial da substring a ser extraída. A próxima chamada para INSTR dentro de SUBSTR encontra a posição da vírgula  $n$ th (o mesmo que a posição inicial), bem como a posição da vírgula  $n$ th + 1. A subtração dos dois valores retorna o comprimento da substring a ser extraída. Como cada valor é analisado em sua própria linha, basta especificar WHERE POS = $n$  para manter o  $n$ th substring (neste caso, onde POS = 2, então a segunda substring na lista).

## PostgreSQL

A visualização inline X percorre cada string. O número de linhas retornadas é determinado por quantos valores existem em cada string. Para encontrar o número de valores em cada string, encontre o número de delimitadores em cada string e adicione um. A função SPLIT\_PART usa os valores em POS para encontrar o  $n$  enésima ocorrência do delimitador e analise a string em valores:

```
select iter.pos, src.name as name1,
       split_part(src.name,',',iter.pos) as name2
  from (select id as pos from t10) iter,
       (select cast(name as text) as name from v) src
```

```

where iter.pos <=
length(src.name)-length(replace(src.name,',',''))+1

pos |      name1      |   name2
---+-----+-----+
 1 | mo,larry,curly |   mo
 2 | mo,larry,curly | larry
 3 | mo,larry,curly | curly
 1 | tina,gina,jaunita,regina,leena | tina
 2 | tina,gina,jaunita,regina,leena | gina
 3 | tina,gina,jaunita,regina,leena | jaunita
 4 | tina,gina,jaunita,regina,leena | regina
 5 | tina,gina,jaunita,regina,leena | leena

```

Mostramos NAME duas vezes para que você possa ver como SPLIT\_PART analisa cada string usando POS. Depois que cada string é analisada, a etapa final é manter as linhas em que POS é igual a  $n$  a substring em que você está interessado, neste caso, 2.

## 6.15      Analisando um endereço IP

### Problema

Você deseja analisar os campos de um endereço IP em colunas. Considere o seguinte endereço IP:

111.22.3.4

Você gostaria que o resultado de sua consulta fosse:

A	B	C	D
111	22	3	4

### Solução

A solução depende das funções incorporadas fornecidas pelo seu DBMS. Independentemente do seu DBMS, ser capaz de localizar os períodos e os números imediatamente ao seu redor são as chaves para a solução.

### DB2

Use a cláusula recursiva WITH para simular uma iteração por meio do endereço IP enquanto usa SUBSTR para analisá-lo facilmente. Um ponto inicial é adicionado ao endereço IP para que cada conjunto de números tenha um ponto na frente e possa ser tratado da mesma maneira.

```

1 with x (pos,ip) as (
2   values (1,'.92.111.0.222')
3   union all
4   select pos+1,ip from x where pos+1 <= 20
5 )

```

```

6 select max(case when rn=1 then e end) a,
7      max(case when rn=2 then e end) b,
8      max(case when rn=3 then e end) c,
9      max(case when rn=4 then e end) d
10     from (
11 select pos,c,d,
12       case when posstr(d,'.') > 0 then substr(d,1,posstr(d,'.')-1)
13             else d
14           end as e,
15       row_number() over( order by pos desc) rn
16     from (
17 select pos, ip,right(ip,pos) as c, substr(right(ip,pos),2) as d
18   from x
19  where pos <= length(ip)
20  and substr(right(ip,pos),1,1) = '.'
21    ) x
22    ) y

```

## MySQL

A função SUBSTR\_INDEX torna a análise de um endereço IP uma operação fácil:

```

1 select substring_index(substring_index(y.ip,'.',1),'.',-1) a,
2       substring_index(substring_index(y.ip,'.',2),'.',-1) b,
3       substring_index(substring_index(y.ip,'.',3),'.',-1) c,
4       substring_index(substring_index(y.ip,'.',4),'.',-1) d
5   from (select '92.111.0.2' as ip from t1) y

```

## Oracle

Use a função interna SUBSTR e INSTR para analisar e navegar pelo endereço IP:

```

1 select ip,
2       substr(ip, 1, instr(ip,'.')-1 ) a,
3       substr(ip, instr(ip,'.')+1,
4              instr(ip,'.',1,2)-instr(ip,'.')-1 ) b,
5       substr(ip, instr(ip,'.',1,2)+1,
6              instr(ip,'.',1,3)-instr(ip,'.',1,2)-1 ) c,
7       substr(ip, instr(ip,'.',1,3)+1 ) d
8   from (select '92.111.0.2' as ip from t1)

```

## PostgreSQL

Use a função integrada SPLIT\_PART para analisar um endereço IP:

```

1 select split_part(y.ip,'.',1) as a,
2       split_part(y.ip,'.',2) as b,
3       split_part(y.ip,'.',3) as c,
4       split_part(y.ip,'.',4) as d
5   from (select cast('92.111.0.2' as text) as ip from t1) as y

```

Use a cláusula recursiva WITH para simular uma iteração por meio do endereço IP enquanto usa SUBSTR para analisá-lo facilmente. Um ponto inicial é adicionado ao endereço IP para que cada conjunto de números tenha um ponto na frente e possa ser tratado da mesma maneira:

```
1 with x (pos,ip) as (
2     select 1 as pos,'.92.111.0.222' as ip from t1
3     union all
4     select pos+1,ip from x where pos+1 <= 20
5 )
6 select max(case when rn=1 then e end) a,
7       max(case when rn=2 then e end) b,
8       max(case when rn=3 then e end) c,
9       max(case when rn=4 then e end) d
10    from (
11 select pos,c,d,
12       case when charindex('.',d) > 0
13           then substring(d,1,charindex('. ',d)-1)
14           else d
15       end as e,
16       row_number() over(order by pos desc) rn
17    from (
18 select pos, ip,right(ip,pos) as c,
19       substring(right(ip,pos),2,len(ip)) as d
20   from x
21  where pos <= len(ip)
22  and substring(right(ip,pos),1,1) = '.'
23      ) x
24      ) y
```

## Discussão

Usando as funções integradas para seu banco de dados, você pode percorrer facilmente as partes de uma string. A chave é poder localizar cada um dos pontos no endereço. Então você pode analisar os números entre cada um.

Em [Receita 6.17](#) veremos como as expressões regulares podem ser usadas com a maioria dos RDBMSs — analisar um endereço IP também é uma boa área para aplicar essa ideia.

## 6.16 Comparando Strings por Som

### Problema

Entre erros de ortografia e formas legítimas de soletrar palavras de maneira diferente, como ortografia britânica versus americana, muitas vezes duas palavras que você deseja corresponder são representadas por diferentes cadeias de caracteres. Felizmente, o SQL fornece uma maneira de representar o som das palavras, o que permite encontrar strings com o mesmo som, mesmo que os caracteres adjacentes não sejam idênticos.

Por exemplo, você tem uma lista de nomes de autores, incluindo alguns de uma época anterior, quando a ortografia não era tão fixa quanto agora, combinada com alguns erros ortográficos e de digitação extras. A seguinte coluna de nomes é um exemplo:

```
a_name
-----
1 Johnson
2 Jonson
3 Jonsen
4 Jensen
5 Johnsen
6 Shakespeare
7 Shakspear
8 Shaekspir
9 Shakespar
```

Embora isso provavelmente faça parte de uma lista mais longa, você gostaria de identificar quais desses nomes são correspondências fonéticas plausíveis para outros nomes da lista. Embora este seja um exercício em que há mais de uma solução possível, sua solução será mais ou menos assim (o significado da última coluna ficará mais claro no final da receita):

a_name1	a_name2	soundex_name
Jensen	Johnson	J525
Jensen	Jonson	J525
Jensen	Jonsen	J525
Jensen	Johnsen	J525
Johnsen	Johnson	J525
Johnsen	Jonson	J525
Johnsen	Jonsen	J525
Johnsen	Johnsen	J525
...		
Jonson	Jensen	J525
Jonson	Johnsen	J525
Shaekspir	Shakspear	S216
Shakespar	Shakespeare	S221
Shakespeare	Shakespar	S221
Shakspear	Shaekspir	S216

## Solução

Use a função SOUNDEX para converter sequências de caracteres na forma como soam quando falados em inglês. Uma autojunção simples permite comparar valores da mesma coluna.

```
1 select an1.a_name as name1, an2.a_name as name2,
2 SOUNDEX(an1.a_name) as Soundex_Name
3 from author_names an1
4 join author_names an2
5 on (SOUNDEX(an1.a_name)=SOUNDEX(an2.a_name))
6 and an1.a_name not like an2.a_name)
```

O pensamento por trás do SOUNDEX é anterior aos bancos de dados e à computação, já que se originou com o Censo dos EUA como uma tentativa de resolver diferentes grafias de nomes próprios para pessoas e lugares. Existem muitos algoritmos que tentam a mesma tarefa que o SOUNDEX e, é claro, existem versões alternativas para outros idiomas além do inglês. No entanto, cobrimos o SOUNDEX, pois ele vem com a maioria dos RDBMSs.

Soundex mantém a primeira letra do nome e substitui os valores restantes por números que tenham o mesmo valor se forem foneticamente semelhantes. Por exemplo, *m* e *n* são ambos substituídos pelo número 5.

No exemplo anterior, a saída real do Soundex é mostrada na coluna *Soundex\_Name*. Isso é apenas para mostrar o que está acontecendo e não é necessário para a solução; alguns RDBMSs possuem até uma função que esconde o resultado do Soundex, como a função do SQL Server *Difference*, que compara duas strings usando o Soundex e retorna uma escala de similaridade de 0 a 4 (por exemplo, 4 é uma correspondência perfeita entre as saídas do Soundex, representando 4/4 caracteres na versão Soundex se as duas strings corresponderem).

Às vezes, o Soundex será suficiente para suas necessidades; outras vezes não será. No entanto, uma pequena quantidade de pesquisa, possivelmente usando textos como *Correspondência de dados* (Christen, 2012), irá ajudá-lo a encontrar outros algoritmos que são frequentemente (mas nem sempre) simples de implementar como uma função definida pelo usuário ou em outra linguagem de programação para se adequar ao seu gosto e necessidades.

## 6.17 Encontrar texto que não corresponde a um padrão

### Problema

Você tem um campo de texto que contém alguns valores de texto estruturado (por exemplo, números de telefone) e deseja localizar ocorrências em que esses valores estão estruturados incorretamente. Por exemplo, você tem dados como os seguintes:

```
select emp_id, text
from employee_comment
```

EMP_ID	TEXT
-----	
7369	126 Varnum, Edmore MI 48829, 989 313-5351
7499	1105 McConnell Court
	Cedar Lake MI 48812
	Home: 989-387-4321
	Cell: (237) 438-3333

e você deseja listar linhas com números de telefone formatados de forma inválida. Por exemplo, você deseja listar a seguinte linha porque seu número de telefone usa dois caracteres separadores diferentes:

Você deseja considerar válidos apenas os números de telefone que usam o mesmo caractere para ambos os delimitadores.

## Solução

Este problema tem uma solução multipartes:

1. Encontre uma maneira de descrever o universo de números de telefone aparentes que você deseja considerar.
2. Remova da consideração quaisquer números de telefone formatados de forma válida.
3. Veja se você ainda tem algum número de telefone aparente. Se o fizer, você sabe que eles estão formatados de forma inválida.

```
select emp_id, text
from employee_comment
where regexp_like(text, '[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}')
  and regexp_like(
    regexp_replace(text,
      '[0-9]{3}([- ])0-9]{3}\10-9]{4}', '***'),
    '[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}')
```

EMP_ID	TEXT
7369	126 Varnum, Edmore MI 48829, 989 313-5351
7844	989-387.5359
9999	906-387-1698, 313-535.8886

Cada uma dessas linhas contém pelo menos um número de telefone aparente que não está formatado corretamente.

## Discussão

A chave para esta solução está na detecção de um “número de telefone aparente”. Dado que os números de telefone são armazenados em um campo de comentário, qualquer texto no campo pode ser interpretado como um número de telefone inválido. Você precisa de uma maneira de restringir o campo para um conjunto mais razoável de valores a serem considerados. Você não deseja, por exemplo, ver a seguinte linha em sua saída:

EMP_ID	TEXT
7900	Cares for 100-year-old aunt during the day. Schedule only for evening and night shifts.

Claramente não há nenhum número de telefone nesta linha, muito menos um que seja inválido. Todos nós podemos ver isso. A questão é, como você faz o RDBMS “ver” isso? Achamos que você vai gostar da resposta. Por favor, continue lendo.



Esta receita vem (com permissão) de um artigo de Jonathan Gennick chamado “Regular Expression Anti-Patterns”.

A solução usa o Padrão A para definir o conjunto de números de telefone “aparente” a considerar:

**Pattern A:** `[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}`

O padrão A verifica dois grupos de três dígitos seguidos por um grupo de quatro dígitos. Qualquer um de um traço (-), um ponto (.) ou um espaço é aceito como um delimitador entre grupos. Você pode criar um padrão mais complexo. Por exemplo, você pode decidir que também deseja considerar números de telefone de sete dígitos. Mas não se deixe desviar. O ponto agora é que, de alguma forma, você precisa definir o universo de sequências de números de telefone possíveis a serem considerados e, para esse problema, esse universo é definido pelo Padrão A. Você pode definir um Padrão A diferente e a solução geral ainda se aplica.

A solução usa o Padrão A na cláusula WHERE para garantir que apenas as linhas com números de telefone em potencial (conforme definido pelo padrão!) sejam consideradas:

```
select emp_id, text
  from employee_comment
 where regexp_like(text, '[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}')
```

Em seguida, você precisa definir como é um número de telefone “bom”. A solução faz isso usando o Padrão B:

**Padrão B:** `[0-9]{3}([- .])[0-9]{3}\1[0-9]{4}`

Desta vez, o padrão usa \1 para referenciar a primeira subexpressão. Qualquer caractere correspondido por `([- .])` também deve ser correspondido por `\1`. O padrão B descreve bons números de telefone, que devem ser eliminados de consideração (já que não são ruins). A solução elimina os números de telefone bem formatados por meio de uma chamada para REGEXP\_REPLACE:

```
regexp_replace(text,
  '[0-9]{3}([- .])[0-9]{3}\1[0-9]{4}', '***'),
```

Essa chamada para REGEXP\_REPLACE ocorre na cláusula WHERE. Todos os números de telefone bem formatados são substituídos por uma sequência de três asteriscos. Novamente, o Padrão B pode ser qualquer padrão que você desejar. A questão é que o Padrão B descreve o padrão aceitável que você busca.

Tendo substituído números de telefone bem formatados por sequências de três asteriscos (\*), quaisquer números de telefone “aparentes” que permaneçam devem, por definição, estar mal formatados. A solução aplica REGEXP\_LIKE à saída de REGEXP\_LIKE para ver se algum número de telefone mal formatado permanece:

```
and regexp_like(  
    regexp_replace(text,  
        '[0-9]{3}([- .])[0-9]{3}\1[0-9]{4}', '***'),  
        '[0-9]{3}([- .])[0-9]{3}[- .][0-9]{4}')
```



As expressões regulares são um grande tópico por si só, exigindo prática para serem dominadas. Depois de dominá-los, você descobrirá que eles combinam com uma grande variedade de padrões de Strings com facilidade. Recomendamos o estudo de um livro como *Mastering Regular Expressions* por Jeffrey Friedl para colocar suas habilidades de expressão regular no nível necessário.

## 6.18 Resumindo

Combinar strings pode ser uma tarefa dolorosa. O SQL adicionou uma variedade de ferramentas para reduzir a dor, e dominá-las o manterá longe de problemas. Embora muito possa ser feito com as funções de string SQL nativas, usar as funções de expressão regular que estão cada vez mais disponíveis leva isso a outro nível.

## CAPÍTULO 7

# Trabalhando com Números

Este capítulo enfoca operações comuns envolvendo números, incluindo cálculos numéricos. Embora o SQL normalmente não seja considerado a primeira escolha para cálculos complexos, ele é eficiente para tarefas numéricas do dia a dia. Mais importante, como bancos de dados e datawarehouses que suportam SQL provavelmente continuam sendo o local mais comum para encontrar os dados de uma organização, usar o SQL para explorar e avaliar esses dados é essencial para qualquer pessoa que os coloque em funcionamento. As técnicas nesta seção também foram escolhidas para ajudar os cientistas de dados a decidir quais dados são os mais promissores para análise posterior.



Algumas receitas neste capítulo usam funções de agregação e a cláusula GROUP BY. Se você não estiver familiarizado com agrupamento, leia pelo menos a primeira seção principal, chamada “Agrupamento”, em [Apêndice A](#).

## 7.1 Calculando uma média

### Problema

Você deseja calcular o valor médio em uma coluna, seja para todas as linhas em uma tabela ou para algum subconjunto de linhas. Por exemplo, você pode querer encontrar o salário médio de todos os funcionários, bem como o salário médio de cada departamento.

### Solução

Ao calcular a média de todos os salários dos funcionários, basta aplicar a função AVG na coluna que contém esses salários.

Ao excluir uma cláusula WHERE, a média é calculada em relação a todos os valores não NULL:

```
1 select avg(sal) as avg_sal
2   from emp
```

```
AVG_SAL
-----
2073.21429
```

Para calcular o salário médio de cada departamento, use a cláusula GROUP BY para criar um grupo correspondente a cada departamento:

```
1 select deptno, avg(sal) as avg_sal
2   from emp
3  group by deptno
```

```
DEPTNO      AVG_SAL
-----
10          2916.66667
20          2175
30          1566.66667
```

## Discussão

Ao encontrar uma média onde toda a tabela é o grupo ou janela, basta aplicar a função AVG na coluna de seu interesse sem utilizar a cláusula GROUP BY. É importante perceber que a função AVG ignora NULLs. O efeito de valores NULL sendo ignorados pode ser visto aqui:

```
create table t2(sal integer)
insert into t2 values (10)
insert into t2 values (20)
insert into t2 values (null)
select avg(sal)    select distinct 30/2
  from t2           from t2

AVG(SAL)            30/2
-----
15                  15

select avg(coalesce(sal,0))  select distinct 30/3
  from t2                from t2

AVG(COALESCE(SAL,0))    30/3
-----
10                  10
```

A função COALESCE retornará o primeiro valor não NULL encontrado na lista de valores que você passar. Quando os valores NULL SAL são convertidos para zero, a média muda. Ao invocar funções de agregação, sempre pense em como você deseja que os NULLs sejam tratados.

A segunda parte da solução usa GROUP BY (linha 3) para dividir registros de funcionários em grupos com base na afiliação do departamento. GROUP BY automaticamente faz com que funções de agregação, como AVG, sejam executadas e retorno um resultado para cada grupo. Neste exemplo, o AVG executaria uma vez para cada grupo de departamentos de registros de funcionários.

A propósito, não é necessário incluir colunas GROUP BY em sua lista de seleção. Por exemplo:

```
select avg(sal)
  from emp
group by deptno

AVG(SAL)
-----
2916.66667
      2175
1566.66667
```

Você ainda está agrupando por DEPTNO mesmo que não esteja na cláusula SELECT. Incluir a coluna pela qual você está agrupando na cláusula SELECT geralmente melhora a legibilidade, mas não é obrigatório. É obrigatório, no entanto, evitar colocar colunas em sua lista SELECT que não estejam também em sua cláusula GROUP BY.

## Veja também

Ver [Apêndice A](#) para uma atualização sobre a funcionalidade GROUP BY.

## 7.2 Encontrando o valor mínimo/máximo em uma coluna

### Problema

Você deseja encontrar os valores mais altos e mais baixos em uma determinada coluna. Por exemplo, você deseja localizar os salários mais altos e mais baixos de todos os funcionários, bem como os salários mais altos e mais baixos de cada departamento.

### Solução

Ao pesquisar o menor e o maior salário de todos os funcionários, basta utilizar as funções MIN e MAX, respectivamente:

```
1 select min(sal) as min_sal, max(sal) as max_sal
2   from emp

MIN_SAL      MAX_SAL
-----        -----
     800          5000
```

Ao pesquisar o menor e o maior salário de cada departamento, utilize as funções MIN e MAX com a cláusula GROUP BY:

```
1 select deptno, min(sal) as min_sal, max(sal) as max_sal
2   from emp
3 group by deptno
```

DEPTNO	MIN_SAL	MAX_SAL
10	1300	5000
20	800	3000
30	950	2850

## Discussão

Ao procurar os valores mais altos ou mais baixos, e nos casos em que toda a tabela é o grupo ou janela, basta aplicar a função MIN ou MAX à coluna de seu interesse sem usar a cláusula GROUP BY.

Lembre-se de que as funções ignoram MIN e MAX NULLs e que você pode ter grupos NULL, bem como valores NULL para colunas em um grupo. A seguir estão exemplos que levam a uma consulta usando GROUP BY que retorna valores NULL para dois grupos (DEPTNO 10 e 20):

```
select deptno, comm
  from emp
 where deptno in (10,30)
 order by 1
```

DEPTNO	COMM
10	
10	
10	
30	300
30	500
30	
30	0
30	1300
30	

```
select min(comm), max(comm)
  from emp
```

MIN(COMM)	MAX(COMM)
0	1300

```
select deptno, min(comm), max(comm)
  from emp
 group by deptno
```

DEPTNO	MIN(COMM)	MAX(COMM)
10		
20		
30	0	1300

Lembre-se, como [Apêndice A](#) aponta, mesmo que nada além de funções agregadas esteja listado na cláusula SELECT, você ainda pode agrupar por outras colunas na tabela; por exemplo:

```
select min(comm), max(comm)
  from emp
 group by deptno
```

MIN(COMM)	MAX(COMM)
0	1300

Aqui você ainda está agrupando por DEPTNO mesmo que não esteja na cláusula SELECT. Incluir a coluna pela qual você está agrupando na cláusula SELECT geralmente melhora a legibilidade, mas não é obrigatório. É obrigatório, entretanto, que qualquer coluna da lista SELECT de uma consulta GROUP BY também esteja listada na cláusula GROUP BY.

## Veja também

Ver [Apêndice A](#) para uma atualização sobre a funcionalidade GROUP BY.

## 7.3 Somar os valores em uma coluna

### Problema

Você deseja calcular a soma de todos os valores, como todos os salários dos funcionários, em uma coluna.

### Solução

Ao calcular uma soma onde toda a tabela é o grupo ou janela, basta aplicar a função SUM nas colunas de seu interesse sem usar a cláusula GROUP BY:

```
1 select sum(sal)
2   from emp
```

SUM(SAL)
29025

Ao criar vários grupos ou janelas de dados, use a função SUM com a cláusula GROUP BY. O exemplo a seguir soma os salários dos funcionários por departamento:

```
1 select deptno, sum(sal) as total_for_dept
2   from emp
3  group by deptno
```

DEPTNO	TOTAL_FOR_DEPT
10	8750
20	10875
30	9400

## Discussão

Ao buscar a soma de todos os salários de cada departamento, você está criando grupos ou “janelas” de dados. O salário de cada funcionário é somado para produzir um total para seu respectivo departamento. Este é um exemplo de agregação em SQL porque informações detalhadas, como o salário de cada funcionário individual, não são o foco; o foco é o resultado final de cada departamento. É importante observar que a função SUM vai ignorar NULLs, mas você pode ter grupos NULL, que podem ser vistos aqui. O DEPTNO 10 não tem funcionários que recebam comissão; assim, agrupar por DEPTNO 10 ao tentar SUM os valores em COMM resultará em um grupo com um valor NULL retornado por SUM:

```
select deptno, comm
  from emp
 where deptno in (10,30)
 order by 1
```

DEPTNO	COMM
10	
10	
10	
30	300
30	500
30	
30	0
30	1300
30	

```
select sum(comm)
  from emp
      SUM(COMM)
-----
  2100
```

```
1 select deptno, sum(comm)
2   from emp
3  where deptno in (10,30)
4 group by deptno
```

DEPTNO	SUM(COMM)
10	
30	2100

## Veja também

Ver [Apêndice A](#) para uma atualização sobre a funcionalidade GROUP BY.

## 7.4 Contando linhas em uma tabela

### Problema

Você deseja contar o número de linhas em uma tabela ou deseja contar o número de valores em uma coluna. Por exemplo, você deseja encontrar o número total de funcionários, bem como o número de funcionários em cada departamento.

### Solução

Ao contar linhas onde toda a tabela é o grupo ou janela, basta utilizar a função COUNT junto com o caractere \*:

```
1 select count(*)
2   from emp

COUNT(*)
-----
14
```

Ao criar vários grupos ou janelas de dados, use a função COUNT com a cláusula GROUP BY:

```
1 select deptno, count(*)
2   from emp
3 group by deptno

DEPTNO    COUNT(*)
-----
10          3
20          5
30          6
```

Ao contar o número de funcionários de cada departamento, você está criando grupos ou “janelas” de dados. Cada funcionário encontrado incrementa a contagem em um para produzir um total para seu respectivo departamento. Este é um exemplo de agregação em SQL porque informações detalhadas, como salário ou função de cada funcionário, não são o foco; o foco é o resultado final de cada departamento. É importante observar que a função COUNT irá ignorar NULLs quando passar um nome de coluna como argumento, mas incluirá NULLs quando passar o caractere \* ou qualquer constante; considere o seguinte:

```
select deptno, comm
      from emp
```

DEPTNO	COMM
20	
30	300
30	500
20	
30	1300
30	
10	
20	
10	
30	0
20	
30	
20	
10	

```
select count(*), count(deptno), count(comm), count('hello')
      from emp
```

COUNT(*)	COUNT(DEPTNO)	COUNT(COMM)	COUNT('HELLO')
14	14	4	14

```
select deptno, count(*), count(comm), count('hello')
      from emp
     group by deptno
```

DEPTNO	COUNT(*)	COUNT(COMM)	COUNT('HELLO')
10	3	0	3
20	5	0	5
30	6	4	6

Se todas as linhas forem nulas para a coluna passada para COUNT ou se a tabela estiver vazia, COUNT retornará zero. Também deve ser observado que, mesmo que nada além de funções de agregação sejam especificadas na cláusula SELECT, você ainda pode agrupar por outras colunas na tabela, por exemplo:

```
select count(*)
  from emp
 group by deptno
```

```
  COUNT(*)
  -----
  3
  5
  6
```

Observe que você ainda está agrupando por DEPTNO mesmo que não esteja na cláusula SELECT. Incluir a coluna pela qual você está agrupando na cláusula SELECT geralmente melhora a legibilidade, mas não é obrigatório. Se você incluí-lo (na lista SELECT), é obrigatório que ele esteja listado na cláusula GROUP BY.

## Veja também

Ver [Apêndice A](#) para uma atualização sobre a funcionalidade GROUP BY.

## 7.5 Contando valores em uma coluna

### Problema

Você deseja contar o número de valores não NULL em uma coluna. Por exemplo, você gostaria de saber quantos funcionários recebem comissão.

### Solução

Conte o número de valores não NULL na coluna COMM da tabela EMP:

```
select count(comm)
  from emp
```

```
  COUNT(COMM)
  -----
    4
```

### Discussão

Quando você “conta estrela”, como em COUNT(\*), o que você realmente está contando são as linhas (independentemente do valor real, e é por isso que as linhas que contêm valores NULL e não NULL são contadas). Mas quando você conta uma coluna, você está contando o número de valores não NULL nessa coluna. A discussão da receita anterior aborda essa distinção.

Nesta solução, COUNT(COMM) retorna o número de valores não NULL na coluna COMM. Como apenas funcionários comissionados têm comissões, o resultado de COUNT(COMM) é o número desses funcionários.

## 7.6 Gerando um Total Acumulado

### Problema

Você deseja calcular um total acumulado de valores em uma coluna.

### Solução

Como exemplo, as soluções a seguir mostram como calcular um total acumulado de salários para todos os funcionários. Para facilitar a leitura, os resultados são ordenados por SAL sempre que possível, para que você possa visualizar facilmente a progressão do total acumulado.

```
1 select ename, sal,
2       sum(sal) over (order by sal,empno) as running_total
3   from emp
4  order by 2
```

ENAME	SAL	RUNNING_TOTAL
SMITH	800	800
JAMES	950	1750
ADAMS	1100	2850
WARD	1250	4100
MARTIN	1250	5350
MILLER	1300	6650
TURNER	1500	8150
ALLEN	1600	9750
CLARK	2450	12200
BLAKE	2850	15050
JONES	2975	18025
SCOTT	3000	21025
FORD	3000	24025
KING	5000	29025

### Discussão

A função de janela SUM OVER torna a geração de um total corrente uma tarefa simples. A cláusula ORDER BY na solução inclui não apenas a coluna SAL, mas também a coluna EMPNO (que é a chave primária) para evitar valores duplicados no total corrente. A coluna RUNNING\_TOTAL2 no exemplo a seguir ilustra o problema que você pode ter com duplicatas:

```
select empno, sal,
       sum(sal)over(order by sal,empno) as running_total1,
       sum(sal)over(order by sal) as running_total2
  from emp
 order by 2
```

ENAME	SAL	RUNNING_TOTAL1	RUNNING_TOTAL2
SMITH	800	800	800
JAMES	950	1750	1750
ADAMS	1100	2850	2850
WARD	1250	4100	5350
MARTIN	1250	5350	5350
MILLER	1300	6650	6650
TURNER	1500	8150	8150
ALLEN	1600	9750	9750
CLARK	2450	12200	12200
BLAKE	2850	15050	15050
JONES	2975	18025	18025
SCOTT	3000	21025	24025
FORD	3000	24025	24025
KING	5000	29025	29025

Os valores em RUNNING\_TOTAL2 para WARD, MARTIN, SCOTT e FORD estão incorretos. Seus salários ocorrem mais de uma vez e essas duplicatas são somadas e adicionadas ao total corrente. É por isso que EMPNO (que é exclusivo) é necessário para produzir os resultados (corretos) que você vê em RUNNING\_TOTAL1. Considere isto: para ADAMS você vê 2850 para RUNNING\_TOTAL1 e RUNNING\_TOTAL2. Adicione o salário de WARD de 1250 a 2850 e você obtém 4100, mas RUNNING\_TOTAL2 retorna 5350. Por quê? Como WARD e MARTIN têm o mesmo SAL, seus dois salários de 1.250 são somados para produzir 2.500, que é então somado a 2.850 para chegar a 5.350 para WARD e MARTIN. Ao especificar uma combinação de colunas para ordenar que não pode resultar em valores duplicados (por exemplo, qualquer combinação de SAL e EMPNO é única), você garante a progressão correta do total corrente.

## 7.7 Gerando um produto em execução

### Problema

Você deseja calcular um produto em execução em uma coluna numérica. A operação é semelhante a [Receita 7.6](#), mas usando multiplicação em vez de adição.

### Solução

A título de exemplo, todas as soluções calculam os produtos em execução dos salários dos funcionários. Embora um produto contínuo de salários possa não ser tão útil, a técnica pode ser facilmente aplicada a outros domínios mais úteis.

Use a função de janela SUM OVER e aproveite o fato de poder simular a multiplicação adicionando logaritmos:

```
1 select empno,ename,sal,
2       exp(sum(ln(sal))over(order by sal,empno)) as running_prod
3   from emp
4 where deptno = 10
```

EMPNO	ENAME	SAL	RUNNING_PROD
7934	MILLER	1300	1300
7782	CLARK	2450	3185000
7839	KING	5000	15925000000

Não é válido em SQL (ou, formalmente falando, em matemática) computar logaritmos de valores menores ou iguais a zero. Se você tiver esses valores em suas tabelas, precisará evitar passar esses valores inválidos para a função LN do SQL. Precauções contra valores inválidos e NULLs não são fornecidas nesta solução para fins de legibilidade, mas você deve considerar se deve colocar tais precauções no código de produção que você escreve. Se você absolutamente precisar trabalhar com valores negativos e zero, essa solução pode não funcionar para você. Ao mesmo tempo, se você tiver zeros (mas nenhum valor abaixo de zero), uma solução comum é adicionar 1 a todos os valores, observando que o logaritmo de 1 é sempre zero, independentemente da base.

Os usuários do SQL Server usam LOG em vez de LN.

## Discussão

A solução aproveita o fato de que você pode multiplicar dois números por:

1. Calculando seus respectivos logaritmos naturais
2. Somando esses logaritmos
3. Elevando o resultado à potência da constante matemática  $e$  (usando a função EXP)

A única ressalva ao usar essa abordagem é que ela não funciona para somar valores zero ou negativos, porque qualquer valor menor ou igual a zero está fora do intervalo de um logaritmo SQL.

Para obter uma explicação de como funciona a função da janela SUM OVER, consulte [Receita 7.6](#).

## 7.8 Suavizando uma série de valores

### Problema

Você tem uma série de valores que aparecem ao longo do tempo, como números de vendas mensais. Como é comum, os dados mostram muita variação de ponto a ponto, mas você está interessado na tendência geral. Portanto, você deseja implementar uma suavização simples, como média ponderada, para identificar melhor a tendência.

Imagine que você tenha totais de vendas diárias, em dólares, como em uma banca de jornal:

DATE1	SALES
2020-01-01	647
2020-01-02	561
2020-01-03	741
2020-01-04	978
2020-01-05	1062
2020-01-06	1072
...	...

No entanto, você sabe que há volatilidade nos dados de vendas que dificultam o discernimento de uma tendência subjacente. Possivelmente dias diferentes da semana ou do mês são conhecidos por terem vendas especialmente altas ou baixas. Alternativamente, talvez você esteja ciente de que, devido à forma como os dados são coletados, às vezes as vendas de um dia são movidas para o dia seguinte, criando uma baixa seguida de um pico, mas não há uma maneira prática de alocar as vendas para o dia correto. Portanto, você precisa suavizar os dados ao longo de vários dias para obter uma visão adequada do que está acontecendo.

Uma média móvel pode ser calculada somando o valor atual e o anterior  $n-1$  valores e dividindo por  $n$ . Se você também exibir os valores anteriores para referência, espera algo assim:

DATE1	sales	salesLagOne	SalesLagTwo	MovingAverage
2020-01-01	647	NULL	NULL	NULL
2020-01-02	561	647	NULL	NULL
2020-01-03	741	561	647	649.667
2020-01-04	978	741	561	760
2020-01-05	1062	978	741	927
2020-01-06	1072	1062	978	1037.333
2020-01-07	805	1072	1062	979.667
2020-01-08	662	805	1072	846.333
2020-01-09	1083	662	805	850
2020-01-10	970	1083	662	905

A fórmula da média é bem conhecida. Ao aplicar uma ponderação simples à fórmula, podemos torná-la mais relevante para esta tarefa, dando mais peso aos valores mais recentes. Use a função de janela LAG para criar uma média móvel:

```
select date1, sales,lag(sales,1) over(order by date1) as salesLagOne,
       lag(sales,2) over(order by date1) as salesLagTwo,
       (sales
        + (lag(sales,1) over(order by date1))
        + lag(sales,2) over(order by date1))/3 as MovingAverage
  from sales
```

## Discussão

Uma média móvel ponderada é uma das maneiras mais simples de analisar dados de séries temporais (dados que aparecem em intervalos de tempo específicos). Esta é apenas uma maneira de calcular uma média móvel simples — você também pode usar uma partição com média. Embora tenhamos selecionado uma média móvel simples de três pontos, existem diferentes fórmulas com diferentes números de pontos de acordo com as características dos dados que você os aplica [.keep-together]#to — #that's que essa técnica realmente se destaca

Por exemplo, uma média móvel ponderada simples de três pontos que enfatiza o ponto de dados mais recente pode ser implementada com a seguinte variante na solução, onde os coeficientes e o denominador foram atualizados:

```
select date1, sales,lag(sales,1) over(order by date1),
       lag(sales,2) over(order by date1),
       ((3*sales)
        + (2*(lag(sales,1) over(order by date1)))
        + (lag(sales,2) over(order by date1)))/6 as SalesMA
  from sales
```

## 7.9 Calculando um Mode

### Problema

Você quer encontrar o mode (para aqueles que não se lembram, o *mode* em matemática é o elemento que aparece com mais frequência para um determinado conjunto de dados) dos valores de uma coluna. Por exemplo, você deseja encontrar a moda dos salários em DEPTNO 20.

Com base nos seguintes vencimentos:

```
select sal
      from emp
     where deptno = 20
    order by sal
```

```

SAL
-----
800
1100
2975
3000
3000

```

o moda é 3000.

## Solução

### DB2, MySQL, PostgreSQL e SQL Server

Use a função de window DENSE\_RANK para classificar as contagens dos salários para facilitar a extração do moda:

```

1 select sal
2   from (
3 select sal,
4       dense_rank()over( order by cnt desc) as rnk
5   from (
6 select sal, count(*) as cnt
7   from emp
8  where deptno = 20
9 group by sal
10      ) x
11      ) y
12      where rnk = 1

```

### Oracle

Você pode usar a extensão KEEP para a função agregada MAX para encontrar o moda SAL. Uma observação importante é que se houver empates, ou seja, várias linhas que são a moda, a solução usando KEEP manterá apenas uma, que é a de maior salário. Se você quiser ver todas as modas (se houver mais de um), você deve modificar esta solução ou simplesmente usar a solução DB2 apresentada anteriormente. Neste caso, como 3000 é o moda SAL em DEPTNO 20 e também é o SAL mais alto, esta solução é suficiente:

```

1 select max(sal)
2 keep(dense_rank first order by cnt desc) sal
3 from (
4 select sal, count(*) cnt
5 from emp
6 where deptno=20
7 group by sal
8 )

```

### DB2 e SQLServer

A exibição em linha X retorna cada SAL e o número de vezes que ela ocorre. A exibição em linha Y usa a função de window DENSE\_RANK (que permite empates) para classificar os resultados.

Os resultados são classificados com base no número de vezes que cada SAL ocorre, conforme mostrado aqui:

```
1 select sal,
2      dense_rank()over(order by cnt desc) as rnk
3   from (
4 select sal,count(*) as cnt
5   from emp
6  where deptno = 20
7 group by sal
8 ) x
```

SAL	RNK
3000	1
800	2
1100	2
2975	2

A parte mais externa da consulta simplesmente mantém a(s) linha(s) em que RNK é 1.

### Oracle

A exibição em linha retorna cada SAL e o número de vezes que ela ocorre e é mostrada aqui:

```
select sal, count(*) cnt
  from emp
 where deptno=20
group by sal
```

SAL	CNT
800	1
1100	1
2975	1
3000	2

O próximo passo é usar a extensão KEEP da função agregada MAX para encontrar a moda. Se você analisar a cláusula KEEP mostrada aqui, notará três subcláusulas, DENSE\_RANK, FIRST e ORDER BY CNT DESC:

```
keep(dense_rank first order by cnt desc)
```

Isso torna a localização da moda extremamente conveniente. A cláusula KEEP determina qual SAL será retornado por MAX observando o valor de CNT retornado pela visualização inline. Trabalhando da direita para a esquerda, os valores para CNT são ordenados em ordem decrescente; então é mantido o primeiro de todos os valores para CNT retornados na ordem DENSE\_RANK. Observando o conjunto de resultados da visualização em linha, você pode ver que 3000 tem o maior CNT de 2. O MAX(SAL) retornado é o maior SAL que tem o maior CNT, neste caso 3000.

## Veja também

Ver [Capítulo 11](#), particularmente a seção “Finding Knight Values,” para uma discussão mais profunda da extensão KEEP da Oracle de funções agregadas.

## 7.10 Calculando uma mediana

### Problema

Você deseja calcular a mediana (para quem não se lembra, a *mediana* é o valor do membro do meio de um conjunto de elementos ordenados) valor para uma coluna de valores numéricos. Por exemplo, você deseja encontrar a mediana dos salários em DEPTNO 20. Com base nos seguintes salários:

```
select sal
  from emp
 where deptno = 20
 order by sal
```

SAL
800
1100
2975
3000
3000

a mediana é 2975.

### Solução

Além da solução Oracle (que usa funções fornecidas para calcular uma mediana), a introdução de funções de window permite uma solução mais eficiente em comparação com a junção automática tradicional.

#### DB2 e PostgreSQL

Use a função de window PERCENTILE\_CONT para encontrar a mediana:

```

1 select percentile_cont(0.5)
2      within group(order by sal)
3  from emp
4 where deptno=20

```

## SQL Server

Use a função de window PERCENTILE\_CONT para encontrar a mediana:

```

1 select percentile_cont(0.5)
2      within group(order by sal)
3      over()
4  from emp
5 where deptno=20

```

A solução do SQL Server funciona com o mesmo princípio, mas requer uma cláusula OVER.

## MySQL

O MySQL não tem a função PERCENTILE\_CONT, então é necessária uma solução alternativa. Uma maneira é usar a função CUME\_DIST em conjunto com um CTE, recriando efetivamente a função PERCENTILE\_CONT:

```

with rank_tab (sal, rank_sal) as
(
  select sal, cume_dist() over (order by sal)
    from emp
   where deptno=20
),
inter as
(
  select sal, rank_sal from rank_tab
 where rank_sal>=0.5
 union
  select sal, rank_sal from rank_tab
 where rank_sal<=0.5
)
select avg(sal) as MedianSal
      from inter

```

## Oracle

Use as funções MEDIAN ou PERCENTILE\_CONT:

```

1 select median(sal)
2  from emp
3 where deptno=20

1 select percentile_cont(0.5)
1      within group(order by sal)

```

```
2 from emp  
3 where deptno=20
```

## Discussão

### Oracle, PostgreSQL, SQL Server e DB2

Além da função MEDIAN do Oracle, a estrutura de todas as soluções é a mesma. A função PERCENTILE\_CONT permite aplicar diretamente a definição de uma mediana, pois a mediana é por definição o percentil 50. Portanto, aplicar essa função com a sintaxe apropriada e usar 0,5 como argumento encontra a mediana.

Claro, outros percentis também estão disponíveis nesta função. Por exemplo, você pode procurar os percentis 5 e/ou 95 para encontrar outliers (outro método para encontrar outliers é descrito mais adiante neste capítulo, quando discutimos o desvio absoluto mediano).

### MySQL

O MySQL não tem uma função PERCENTILE\_CONT, o que torna as coisas mais complicadas. Para encontrar a mediana, os valores de SAL devem ser ordenados do menor para o maior. A função CUME\_DIST atinge esse objetivo e rotula cada linha com seu percentil. Assim, pode ser usado para obter o mesmo resultado que a função PERCENTILE\_CONT usada na solução para os outros bancos de dados.

A única dificuldade é que a função CUME\_DIST não é permitida em uma cláusula WHERE. Como resultado, você precisa aplicá-lo primeiro em um CTE.

A única armadilha aqui é que, se o número de linhas for par, não haverá uma linha exatamente na mediana. Portanto, a solução é escrita para encontrar a média do maior valor abaixo ou igual à mediana e do menor valor acima ou igual à mediana. Este método funciona para números ímpares e pares de linhas e, se houver um número ímpar de linhas que forneça uma mediana exata, será calculada a média de dois números iguais.

## 7.11 Determinando a porcentagem de um total

### Problema

Você deseja determinar a porcentagem que os valores em uma coluna específica representam em relação a um total. Por exemplo, você deseja determinar qual porcentagem de todos os salários são os salários em DEPTNO 10 (a porcentagem que os salários de DEPTNO 10 contribuem para o total).

Em geral, calcular uma porcentagem em relação a um total em SQL não é diferente de fazê-lo no papel: simplesmente divida e depois multiplique. Neste exemplo, você deseja encontrar a porcentagem do total de salários na tabela EMP que vem do DEPTNO 10. Para fazer isso, basta encontrar os salários do DEPTNO 10 e depois dividir pelo salário total da tabela. Como última etapa, multiplique por 100 para retornar um valor que representa uma porcentagem.

### MySQL e PostgreSQL

Divida a soma dos vencimentos do DEPTNO 10 pela soma de todos os vencimentos:

```
1 select (sum(
2           case when deptno = 10 then sal end)/sum(sal)
3           )*100 as pct
4      from emp
```

### DB2, Oracle e SQL Server

Use uma exibição em linha com a função de window SUM OVER para encontrar a soma de todos os salários junto com a soma de todos os salários em DEPTNO 10. Em seguida, faça a divisão e a multiplicação na consulta externa:

```
1 select distinct (d10/total)*100 as pct
2   from (
3 select deptno,
4       sum(sal)over() total,
5       sum(sal)over(partition by deptno) d10
6   from emp
7       ) x
8  where deptno=10
```

## Discussão

### MySQL e PostgreSQL

A instrução CASE retorna convenientemente apenas os salários de DEPTNO 10. Eles são então somados e divididos pela soma de todos os salários. Como NULLs são ignorados por agregações, uma cláusula ELSE não é necessária na instrução CASE. Para ver exatamente quais valores estão divididos, execute a consulta sem a divisão:

```
select sum(case when deptno = 10 then sal end) as d10,
       sum(sal)
      from emp
```

D10	SUM(SAL)
-----	-----
8750	29025

Dependendo de como você define SAL, pode ser necessário usar CAST explicitamente ao realizar a divisão para garantir o tipo de dados correto. Por exemplo, no DB2, SQL Server e PostgreSQL, se SAL for armazenado como um número inteiro, você pode aplicar CAST para garantir que um valor decimal seja retornado, conforme mostrado aqui:

```
select (cast(
    sum(case when deptno = 10 then sal end)
        as decimal)/sum(sal)
    )*100 as pct
from emp
```

### DB2, Oracle e SQL Server

Em alternativa à solução tradicional, esta solução utiliza funções de window para calcular uma percentagem relativa ao total. Para DB2 e SQL Server, se você armazenou SAL como um número inteiro, precisará usar CAST antes de dividir:

```
select distinct
    cast(d10 as decimal)/total*100 as pct
from (
    select deptno,
        sum(sal)over() total,
        sum(sal)over(partition by deptno) d10
    from emp
    ) x
where deptno=10
```

É importante ter em mente que as funções de window são aplicadas após a cláusula WHERE ser avaliada. Assim, o filtro no DEPTNO não pode ser executado na visualização em linha X. Considere os resultados da visualização inline X sem e com o filtro em DEPTNO. Primeiro sem:

```
select deptno,
    sum(sal)over() total,
    sum(sal)over(partition by deptno) d10
from emp
```

DEPTNO	TOTAL	D10
10	29025	8750
10	29025	8750
10	29025	8750
20	29025	10875
20	29025	10875
20	29025	10875
20	29025	10875
30	29025	9400
30	29025	9400
30	29025	9400
30	29025	9400

30	29025	9400
30	29025	9400

e agora com:

```
select deptno,
       sum(sal)over() total,
       sum(sal)over(partition by deptno) d10
  from emp
 where deptno=10
```

DEPTNO	TOTAL	D10
10	8750	8750
10	8750	8750
10	8750	8750

Como as funções de window são aplicadas após a cláusula WHERE, o valor de TOTAL representa a soma de todos os salários somente em DEPTNO 10. Mas para resolver o problema você quer que o TOTAL represente a soma de todos os salários, ponto final. É por isso que o filtro em DEPTNO deve acontecer fora da visualização inline X.

## 7.12 Agregando Colunas Anuláveis

### Problema

Você deseja executar uma agregação em uma coluna, mas a coluna pode ser anulada. Você deseja que a precisão de sua agregação seja preservada, mas está preocupado porque as funções de agregação ignoram NULLs. Por exemplo, você deseja determinar a comissão média para funcionários em DEPTNO 30, mas há alguns funcionários que não recebem comissão (COMM é NULL para esses funcionários). Como os NULLs são ignorados pelas agregações, a precisão da saída é comprometida. Você gostaria de incluir de alguma forma valores NULL em sua agregação.

### Solução

Use a função COALESCE para converter NULLs em zero para que sejam incluídos na agregação:

```
1 select avg(coalesce(comm,0)) as avg_comm
2   from emp
3  where deptno=30
```

### Discussão

Ao trabalhar com funções de agregação, lembre-se de que NULLs são ignorados. Considere a saída da solução sem usar a função COALESCE:

```
select avg(comm)
  from emp
 where deptno=30

AVG(COMM)
-----
      550
```

Esta consulta mostra uma comissão média de 550 para DEPTNO 30, mas um rápido exame dessas linhas:

```
select ename, comm
  from emp
 where deptno=30
order by comm desc

ENAME          COMM
-----
BLAKE
JAMES
MARTIN        1400
WARD           500
ALLEN          300
TURNER          0
```

mostra que apenas quatro dos seis funcionários podem ganhar uma comissão. A soma de todas as comissões no DEPTNO 30 é 2200, e a média deve ser 2200/6, não 2200/4. Ao excluir a função COALESCE, você responde à pergunta “Qual é a comissão média dos funcionários no DEPTNO 30 *quem pode ganhar uma comissão?*” em vez de “Qual é a comissão média de todos os funcionários no DEPTNO 30?” Ao trabalhar com agregações, lembre-se de tratar NULLs de acordo.

## 7.13 Calculando médias sem valores altos e baixos

### Problema

Você deseja calcular uma média, mas deseja excluir os valores mais altos e mais baixos para (esperançosamente) reduzir o efeito da distorção. Em linguagem estatística, isso é conhecido como *média aparada*. Por exemplo, você deseja calcular o salário médio de todos os funcionários, excluindo os salários mais alto e mais baixo.

### Solução

#### MySQL e PostgreSQL

Use subconsultas para excluir valores altos e baixos:

```
1 select avg(sal)
2   from emp
```

```

3 where sal not in (
4 (select min(sal) from emp),
5 (select max(sal) from emp)
6 )

```

### DB2, Oracle e SQL Server

Use uma exibição em linha com as funções de window MAX OVER e MIN OVER para gerar um conjunto de resultados do qual você pode facilmente eliminar os valores altos e baixos:

```

1 select avg(sal)
2   from (
3 select sal, min(sal)over() min_sal, max(sal)over() max_sal
4   from emp
5      ) x
6 where sal not in (min_sal,max_sal)

```

## Discussão

### MySQL e PostgreSQL

As subconsultas retornam os salários mais altos e mais baixos da tabela. Ao usar NOT IN contra os valores retornados, você exclui os salários mais altos e mais baixos da média. Lembre-se de que, se houver duplicatas (se vários funcionários tiverem os salários mais altos ou mais baixos), todos serão excluídos da média. Se seu objetivo é excluir apenas uma única instância dos valores alto e baixo, simplesmente subtraia-os da SUM e divida:

```

select (sum(sal)-min(sal)-max(sal))/(count(*)-2)
  from emp

```

### DB2, Oracle e SQL Server

A exibição em linha X retorna cada salário junto com os salários mais altos e mais baixos:

```

select sal, min(sal)over() min_sal, max(sal)over() max_sal
  from emp

```

SAL	MIN_SAL	MAX_SAL
800	800	5000
1600	800	5000
1250	800	5000
2975	800	5000
1250	800	5000
2850	800	5000
2450	800	5000
3000	800	5000
5000	800	5000
1500	800	5000
1100	800	5000

950	800	5000
3000	800	5000
1300	800	5000

Você pode acessar os salários altos e baixos em cada linha, portanto, descobrir quais salários são mais altos e/ou mais baixos é trivial. A consulta externa filtra as linhas retornadas da exibição em linha X de moda que qualquer salário que corresponda a MIN\_SAL ou MAX\_SAL seja excluído da média.

### Estatísticas robustas



Na linguagem estatística, uma média calculada com os maiores e menores valores removidos é chamada de *média aparada*. Isso pode ser considerado uma estimativa mais segura da média e é um exemplo de *estatística robusta*, assim chamados porque são menos sensíveis a problemas como viés. Receita 7.16 é outro exemplo de uma ferramenta estatística robusta. Em ambos os casos, essas abordagens são valiosas para quem analisa dados em um RDBMS porque não exigem que o analista faça suposições difíceis de testar com a gama relativamente limitada de ferramentas estatísticas disponíveis em SQL.

## 7.14 Convertendo Strings Alfanuméricas em Números

### Problema

Você tem dados alfanuméricos e gostaria de retornar apenas números. Você deseja retornar o número 123321 da string “paul123f321”.

### Solução

#### DB2

Use as funções TRANSLATE e REPLACE para extrair caracteres numéricos de uma string alfanumérica:

```

1 select cast(
2      replace(
3          translate( 'paul123f321',
4                  repeat('#',26),
5                  'abcdefghijklmnopqrstuvwxyz'), '#', '')
6      as integer ) as num
7  from t1

```

#### Oracle, SQL Server e PostgreSQL

Use as funções TRANSLATE e REPLACE para extrair caracteres numéricos de uma string alfanumérica:

```

1 select cast(
2      replace(
3      translate( 'paul123f321',
4                  'abcdefghijklmnopqrstuvwxyz',
5                  rpad('#',26,'#')),'#','')
6      as integer ) as num
7  from t1

```

## MySQL

No momento em que este livro foi escrito, o MySQL não suporta a função TRANSLATE; assim, uma solução não será fornecida.

## Discussão

A única diferença entre as duas soluções é a sintaxe; O DB2 usa a função REPEAT em vez de RPAD e a lista de parâmetros para TRANSLATE está em uma ordem diferente. A explicação a seguir usa a solução Oracle/PostgreSQL, mas também é relevante para o DB2. Se você executar a consulta de dentro para fora (começando apenas com TRANSLATE), verá que é simples. Primeiro, TRANSLATE converte qualquer caractere não numérico em uma instância de #:

```

select translate( 'paul123f321',
                  'abcdefghijklmnopqrstuvwxyz',
                  rpad('#',26,'#')) as num
from t1

NUM
-----
####123#321

```

Como todos os caracteres não numéricos agora são representados por #, simplesmente use REPLACE para removê-los e, em seguida, use CAST para retornar o resultado como um número. Este exemplo particular é extremamente simples porque os dados são alfanuméricos. Se caracteres adicionais puderem ser armazenados, em vez de procurá-los, é mais fácil abordar esse problema de maneira diferente: em vez de localizar caracteres não numéricos e removê-los, localize todos os caracteres numéricos e remova qualquer coisa que não esteja entre eles. O exemplo a seguir ajudará a esclarecer essa técnica:

```

select replace(
    translate('paul123f321',
              replace(translate( 'paul123f321',
                                '0123456789',
                                rpad('#',10,'#')),'#',''),
              rpad('#',length('paul123f321'),'#')),'#','') as num
from t1

NUM
-----
123321

```

Essa solução parece um pouco mais complicada do que a original, mas não é tão ruim depois que você a analisa. Observe a chamada interna para TRANSLATE:

```
select translate( 'paul123f321',
                  '0123456789',
                  rpad('#',10,'#'))
  from t1

TRANSLATE(
-----
paul###f###
```

Portanto, a abordagem inicial é diferente; em vez de substituir cada caractere não numérico por uma ocorrência de #, substitua cada caractere numérico por uma ocorrência de #. A próxima etapa remove todas as instâncias de #, deixando apenas caracteres não numéricos:

```
select replace(translate( 'paul123f321',
                           '0123456789',
                           rpad('#',10,'#')), '#', '')
  from t1

REPLA
-----
paulf
```

A próxima etapa é chamar TRANSLATE novamente, desta vez para substituir cada um dos caracteres não numéricos (da consulta anterior) por uma instância de # na string original:

```
select translate('paul123f321',
                 replace(translate( 'paul123f321',
                                   '0123456789',
                                   rpad('#',10,'#')), '#', ''),
                 rpad('#',length('paul123f321'), '#'))
  from t1

TRANSLATE(
-----
#####123#321
```

Neste ponto, pare e examine a chamada externa para TRANSLATE. O segundo parâmetro para RPAD (ou o segundo parâmetro para REPEAT para DB2) é o comprimento da string original. Isso é conveniente de usar, pois nenhum caractere pode ocorrer vezes suficientes para ser maior do que a string da qual faz parte. Agora que todos os caracteres não numéricos foram substituídos por ocorrências de #, a última etapa é usar REPLACE para remover todas as ocorrências de #. Agora você fica com um número.

## 7.15 Alterando valores em um total corrente

### Problema

Você deseja modificar os valores em um total acumulado dependendo dos valores em outra coluna. Considere um cenário em que você deseja exibir o histórico de transações de uma conta de cartão de crédito junto com o saldo atual após cada transação. A seguinte visão, V, será usada neste exemplo:

```
create view V (id,amt,trx)
as
select 1, 100, 'PR' from t1 union all
select 2, 100, 'PR' from t1 union all
select 3, 50, 'PY' from t1 union all
select 4, 100, 'PR' from t1 union all
select 5, 200, 'PY' from t1 union all
select 6, 50, 'PY' from t1

select * from V
```

ID	AMT	TR
1	100	PR
2	100	PR
3	50	PY
4	100	PR
5	200	PY
6	50	PY

O ID da coluna identifica exclusivamente cada transação. A coluna AMT representa a quantidade de dinheiro envolvida em cada transação (seja uma compra ou um pagamento). A coluna TRX define o tipo de transação; um pagamento é “PY” e uma compra é “PR”. Se o valor de TRX for PY, você deseja que o valor atual de AMT seja subtraído do total corrente; se o valor de TRX for PR, você deseja que o valor atual de AMT seja adicionado ao total corrente. Por fim, você deseja retornar o seguinte conjunto de resultados:

TRX_TYPE	AMT	BALANCE
PURCHASE	100	100
PURCHASE	100	200
PAYOUT	50	150
PURCHASE	100	250
PAYOUT	200	50
PAYOUT	50	0

### Solução

Use a função de window SUM OVER para criar o total acumulado junto com uma expressão CASE para determinar o tipo de transação:

```

1 select case when trx = 'PY'
2           then 'PAYMENT'
3           else 'PURCHASE'
4       end trx_type,
5       amt,
6       sum(
7           case when trx = 'PY'
8               then -amt else amt
9           end
10      ) over (order by id,amt) as balance
11 from V

```

## Discussão

A expressão CASE determina se o AMT atual é adicionado ou deduzido do total corrente. Se a transação for um pagamento, o AMT é alterado para um valor negativo, reduzindo assim o valor do total corrente. O resultado da expressão CASE é mostrado aqui:

```

select case when trx = 'PY'
            then 'PAYMENT'
            else 'PURCHASE'
        end trx_type,
        case when trx = 'PY'
            then -amt else amt
        end as amt
from V

```

TRX_TYPE	AMT
PURCHASE	100
PURCHASE	100
PAYMENT	-50
PURCHASE	100
PAYMENT	-200
PAYMENT	-50

Depois de avaliar o tipo de transação, os valores para AMT são adicionados ou subtraídos do total corrente. Para obter uma explicação sobre como a função de janela, SUM OVER ou a subconsulta escalar cria o total acumulado, consulte a receita [Receita 7.6](#).

## 7.16 Encontrando outliers usando o desvio absoluto mediano

### Problema

Você deseja identificar valores em seus dados que podem ser suspeitos. Existem vários motivos pelos quais os valores podem ser suspeitos - pode haver um problema de coleta de dados, como um erro com o medidor que registra o valor.

Pode haver um erro de entrada de dados, como um erro de digitação ou similar. Também pode haver circunstâncias incomuns quando os dados foram gerados, o que significa que o ponto de dados está correto, mas eles ainda exigem que você tenha cuidado em qualquer conclusão que tirar dos dados. Portanto, você deseja detectar outliers.

Uma maneira comum de detectar outliers, ensinada em muitos cursos de estatística destinados a não estatísticos, é calcular o desvio padrão dos dados e decidir que os pontos de dados com mais de três desvios padrão (ou alguma outra distância semelhante) são outliers. No entanto, esse método pode identificar erroneamente outliers se os dados não seguirem uma distribuição normal, especialmente se a dispersão dos dados não for simétrica ou não diminuir da mesma forma que uma distribuição normal à medida que você se afasta da média.

## Solução

Primeiro encontre a mediana dos valores usando a receita para encontrar a mediana do início deste capítulo. Você precisará colocar essa consulta em um CTE para disponibilizá-la para consultas posteriores. O desvio é a diferença absoluta entre a mediana e cada valor; o desvio absoluto mediano é a mediana desse valor, então precisamos calcular a mediana novamente.

### SQL Server

O SQL Server possui a função PERCENTILE\_CONT, que simplifica a localização da mediana. Como precisamos encontrar duas medianas diferentes e manipulá-las, precisamos de uma série de CTEs:

```
with median (median)
as
(select distinct percentile_cont(0.5) within group(order by sal)
     over()
  from emp),
Deviation (Deviation)
as
(Select abs(sal-median)
  from emp join median on 1=1),
MAD (MAD) as
(select DISTINCT PERCENTILE_CONT(0.5) within group(order by deviation) over()
  from Deviation )
select abs(sal-MAD)/MAD, sal, ename, job
  from MAD join emp on 1=1
```

### PostgreSQL e DB2

O padrão geral é o mesmo, mas há uma sintaxe diferente para PERCENTILE\_CONT, pois o PostgreSQL e o DB2 tratam PERCENTILE\_CONT como uma função agregada em vez de estritamente uma função de janela:

```

with median (median)
as
(select percentile_cont(0.5) within group(order by sal)
from emp),

devtab (deviation)
as
(select abs(sal-median)
from emp join median),

MedAbsDeviation (MAD) as
(select percentile_cont (0.5) within group(order by deviation)
from devtab)

select abs(sal-MAD)/MAD, sal, ename, job
FROM MedAbsDeviation join emp

```

## Oracle

A receita é simplificada para usuários do Oracle devido à existência de uma função mediana. No entanto, ainda precisamos usar um CTE para lidar com o valor escalar do desvio:

```

with
Deviation (Deviation)
as
(select abs(sal-median(sal))
from emp),

MAD (MAD) as
(select median(Deviation)
from Deviation )

select abs(sal-MAD)/MAD, sal, ename, job
FROM MAD join emp

```

## MySQL

Como vimos na seção anterior sobre a mediana, infelizmente não há função MEDIAN ou PERCENTILE\_CONT no MySQL. Isso significa que cada uma das medianas que precisamos encontrar para calcular o desvio absoluto da mediana são duas subconsultas dentro de uma CTE. Isso torna o MySQL um pouco prolixo:

```

with rank_tab (sal, rank_sal) as (
select sal, cume_dist() over (order by sal)
from emp),
inter as
(
select sal, rank_sal from rank_tab
where rank_sal>=0.5
union
select sal, rank_sal from rank_tab
where rank_sal<=0.5

```

```
)  
,  
  
medianSal (medianSal) as  
  
(  
  select (max(sal)+min(sal))/2  
  from inter),  
  deviationSal (Sal,deviationSal)  
  as (select Sal,abs(sal-medianSal)  
  from emp join medianSal  
  on 1=1  
)  
,  
  
distDevSal (sal,deviationSal,distDeviationSal) as  
  
(  
  select sal,deviationSal,cume_dist() over (order by deviationSal)  
  from deviationSal  
)  
,  
  
DevInter (DevInter, sal) as  
(  
  select min(deviationSal), sal  
  from distDevSal  
  where distDeviationSal >= 0.5  
  
union  
  
  select max(DeviationSal), sal  
  from distDevSal  
  where distDeviationSal <= 0.5  
)  
  
MAD (MedianAbsoluteDeviance) as  
(  
  select abs(emp.sal-(min(devInter)+max(devInter))/2)  
  from emp join DevInter on 1=1  
)  
  
  select emp.sal,MedianAbsoluteDeviance,  
  (emp.sal-  
  deviationSal)/MedianAbsoluteDeviance from  
  (emp join MAD on 1=1)  
  join deviationSal on emp.sal=deviationSal.sal
```

## Discussão

Em cada caso, a receita segue uma estratégia semelhante. Primeiro, precisamos calcular a mediana e, em seguida, precisamos calcular a mediana da diferença entre cada valor e a mediana, que é o desvio absoluto da mediana real.

Por fim, precisamos usar uma consulta para encontrar a razão entre o desvio de cada valor e o desvio mediano. Nesse ponto, podemos usar o resultado de maneira semelhante ao desvio padrão. Por exemplo, se um valor tiver três ou mais desvios da mediana, pode ser considerado um valor atípico, para usar uma interpretação comum.

Conforme mencionado anteriormente, o benefício dessa abordagem sobre o desvio padrão é que a interpretação ainda é válida mesmo que os dados não exibam uma distribuição normal. Por exemplo, pode ser desequilibrado e o desvio absoluto mediano ainda fornecerá uma resposta sólida.

Em nossos dados salariais, há um salário que está a mais de três desvios absolutos da mediana: o do CEO.

Embora existam opiniões divergentes sobre a justiça dos salários do CEO em relação aos da maioria dos outros trabalhadores, dado que o salário atípico pertence ao CEO, ele se encaixa em nossa compreensão dos dados. Em outros contextos, se não houvesse uma explicação clara de por que o valor diferia tanto, isso poderia nos levar a questionar se aquele valor estava correto ou se o valor fazia sentido quando tomado com o restante dos valores (por exemplo, se não é realmente um erro, pode nos fazer pensar que precisamos analisar nossos dados em mais de um subgrupo).

Muitas das estatísticas comuns, como a média e o desvio padrão, assumem que a forma dos dados é uma curva de sino — uma distribuição normal. Isso é verdade para muitos conjuntos de dados e também não é verdade para muitos conjuntos de dados.



Existem vários métodos para testar se um conjunto de dados segue uma distribuição normal, tanto pela visualização dos dados quanto por meio de cálculos. Os pacotes estatísticos geralmente contêm funções para esses testes, mas são inexistentes e difíceis de replicar no SQL. No entanto, muitas vezes existem ferramentas estatísticas alternativas que não assumem que os dados assumem uma forma específica – estatísticas não paramétricas – e são mais seguras de usar.

## 7.17 Encontrando anomalias usando a lei de Benford

### Problema

Embora os outliers, conforme mostrado na receita anterior, sejam uma forma prontamente identificável de dados anômalos, alguns outros dados são menos fáceis de identificar como problemáticos. Uma maneira de detectar situações em que há dados anômalos, mas sem outliers óbvios, é observar a frequência dos dígitos, que geralmente segue a lei de Benford. Embora o uso da lei de Benford seja mais frequentemente associado à detecção de fraude em situações em que humanos adicionaram números falsos a um conjunto de dados, ela pode ser usada de forma mais geral para detectar dados que não seguem os padrões esperados.

Por exemplo, ele pode detectar erros como pontos de dados duplicados, que não necessariamente se destacam como discrepantes.

## Solução

Para usar a lei de Benford, você precisa calcular a distribuição esperada de dígitos e depois a distribuição real para comparar. Embora os usos mais sofisticados olhem para primeiro, segundo e combinações de dígitos, neste exemplo vamos nos ater apenas aos primeiros dígitos.

Você compara a frequência prevista pela lei de Benford com a frequência real de seus dados. Por fim, você deseja quatro colunas - o primeiro dígito, a contagem de quantas vezes cada primeiro dígito aparece, a frequência dos primeiros dígitos prevista pela lei de Benford e a frequência real:

```
with
FirstDigits (FirstDigit)
as
(select left(cast(SAL as CHAR),1) as FirstDigit
 from emp),

TotalCount (Total)
as
(select count(*)
 from emp),

ExpectedBenford (Digit,Expected)
as
(select value,(log10(value + 1) - log10(value)) as expected
 from t10
 where value < 10)

select count(FirstDigit),Digit
,coalesce(count(*)/Total,0) as ActualProportion,Expected
From FirstDigits
Join TotalCount
Right Join ExpectedBenford
on FirstDigits.FirstDigit=ExpectedBenford.Digit
group by Digit
order by Digit;
```

## Discussão

Como precisamos usar duas contagens diferentes — uma das linhas totais e outra do número de linhas contendo cada primeiro dígito diferente — precisamos usar um CTE. A rigor, não precisamos colocar os resultados esperados da lei de Benford em uma consulta separada dentro do CTE, mas fizemos isso neste caso, pois nos permite identificar os dígitos com contagem zero e exibi-los na tabela por meio de a junção certa.

Também é possível produzir a contagem de FirstDigits na consulta principal, mas optamos por não melhorar a legibilidade por não precisar repetir a LEFT(expressão CAST... na cláusula GROUP BY.

A matemática por trás da lei de Benford é simples:

$$\text{Frequência esperada} = \log_{10}\left(\frac{d+1}{d}\right)$$

Podemos usar a tabela dinâmica T10 para gerar os valores apropriados. A partir daí só precisamos calcular as frequências reais para comparação, o que primeiro exige que identifiquemos o primeiro dígito.

A lei de Benford funciona melhor quando há uma coleção relativamente grande de valores para aplicá-la e quando esses valores abrangem mais de uma ordem de grandeza (10, 100, 1.000 etc.). Essas condições não são totalmente atendidas aqui. Ao mesmo tempo, o desvio do esperado ainda deve nos deixar desconfiados de que esses valores são, de certa forma, valores inventados e vale a pena investigar mais a fundo.

## 7.18 Resumindo

Os dados de uma empresa são freqüentemente encontrados em um banco de dados suportado por SQL, então faz sentido usar o SQL para tentar entender esses dados. O SQL não possui toda a gama de ferramentas estatísticas que você esperaria em um pacote desenvolvido especificamente, como SAS, a linguagem de programação estatística R ou as bibliotecas estatísticas do Python. No entanto, ele possui um rico conjunto de ferramentas de cálculo que, como vimos, pode fornecer uma compreensão profunda das propriedades estatísticas de seus dados.

## CAPÍTULO 8

# Data Aritmética

Este capítulo apresenta técnicas para realizar aritmética de data simples. As receitas abrangem tarefas comuns, como adicionar dias a datas, encontrar o número de dias úteis entre datas e encontrar a diferença entre datas em dias.

Ser capaz de manipular datas com sucesso com as funções integradas do seu RDBMS pode melhorar muito sua produtividade. Para todas as receitas deste capítulo, tentamos tirar vantagem das funções integradas de cada RDBMS. Além disso, optamos por usar um formato de data para todas as receitas, DD-MON-YYYY. Obviamente, existem vários outros formatos comumente usados, como DD-MM-YYYY, o formato padrão ISO.

Escolhemos padronizar em DD-MON-YYYY para beneficiar aqueles que trabalham com um RDBMS e desejam aprender outros. Ver um formato padrão ajudará você a se concentrar nas diferentes técnicas e funções fornecidas por cada RDBMS sem ter que se preocupar com os formatos de data padrão.



Este capítulo enfoca a aritmética básica de datas. Você encontrará receitas de tâmaras mais avançadas no capítulo seguinte. As receitas apresentadas neste capítulo usam tipos de dados de data simples. Se você estiver usando tipos de dados de data mais complexos, precisará ajustar as soluções de acordo.

### 8.1 Adição e subtração de dias, meses e anos

#### Problema

Você precisa adicionar ou subtrair um certo número de dias, meses ou anos de uma data. Por exemplo, usando HIREDATE para o funcionário CLARK, você deseja retornar seis datas diferentes: cinco dias antes e depois da contratação de CLARK, cinco meses antes e depois da contratação de CLARK,

e, finalmente, cinco anos antes e depois da contratação de CLARK. CLARK foi contratado em 09-JUN-2006, então você deseja retornar o seguinte conjunto de resultados:

```
HD_MINUS_5D HD_PLUS_5D  HD_MINUS_5M HD_PLUS_5M  HD_MINUS_5Y HD_PLUS_5Y
-----
04-JUN-2006 14-JUN-2006 09-JAN-2006 09-NOV-2006 09-JUN-2001 09-JUN-2001
12-NOV-2006 22-NOV-2006 17-JUN-2006 17-APR-2007 17-NOV-2001 17-NOV-2001
18-JAN-2007 28-JAN-2007 23-AUG-2006 23-JUN-2007 23-JAN-2002 23-JAN-2002
```

## Solução

### DB2

Adição e subtração padrão são permitidas em valores de data, mas qualquer valor que você adicionar ou subtrair de uma data deve ser seguido pela unidade de tempo que representa:

```
1 select hiredate -5 day    as hd_minus_5D,
2      hiredate +5 day    as hd_plus_5D,
3      hiredate -5 month as hd_minus_5M,
4      hiredate +5 month as hd_plus_5M,
5      hiredate -5 year  as hd_minus_5Y,
6      hiredate +5 year  as hd_plus_5Y
7  from emp
8 where deptno = 10
```

### Oracle

Use adição e subtração padrão para dias e use a função ADD\_MONTHS para adicionar e subtrair meses e anos:

```
1 select hiredate-5           as hd_minus_5D,
2      hiredate+5           as hd_plus_5D,
3      add_months(hiredate,-5) as hd_minus_5M,
4      add_months(hiredate,5)  as hd_plus_5M,
5      add_months(hiredate,-5*12) as hd_minus_5Y,
6      add_months(hiredate,5*12) as hd_plus_5Y
7  from emp
8 where deptno = 10
```

### PostgreSQL

Use adição e subtração padrão com a palavra-chave INTERVAL especificando a unidade de tempo para adicionar ou subtrair. Aspas simples são necessárias ao especificar um valor INTERVAL:

```
1 select hiredate - interval '5 day'   as hd_minus_5D,
2      hiredate + interval '5 day'    as hd_plus_5D,
3      hiredate - interval '5 month' as hd_minus_5M,
4      hiredate + interval '5 month' as hd_plus_5M,
5      hiredate - interval '5 year'  as hd_minus_5Y,
6      hiredate + interval '5 year'  as hd_plus_5Y
```

```
7 from emp
8 where deptno=10
```

## MySQL

Use adição e subtração padrão com a palavra-chave INTERVAL especificando a unidade de tempo para adicionar ou subtrair. Ao contrário da solução PostgreSQL, você não coloca aspas simples ao redor do valor INTERVAL:

```
1 select hiredate - interval 5 day    as hd_minus_5D,
2       hiredate + interval 5 day    as hd_plus_5D,
3       hiredate - interval 5 month as hd_minus_5M,
4       hiredate + interval 5 month as hd_plus_5M,
5       hiredate - interval 5 year  as hd_minus_5Y,
6       hiredate + interval 5 year  as hd_plus_5Y
7   from emp
8  where deptno=10
```

Alternativamente, você pode usar a função DATE\_ADD, que é mostrada aqui:

```
1 select date_add(hiredate,interval -5 day)  as hd_minus_5D,
2       date_add(hiredate,interval 5 day)    as hd_plus_5D,
3       date_add(hiredate,interval -5 month) as hd_minus_5M,
4       date_add(hiredate,interval 5 month)  as hd_plus_5M,
5       date_add(hiredate,interval -5 year)  as hd_minus_5Y,
6       date_add(hiredate,interval 5 year)   as hd_plus_5Y
7   from emp
8  where deptno=10
```

## SQL Server

Use a função DATEADD para adicionar ou subtrair diferentes unidades de tempo para/de uma data:

```
1 select dateadd(day,-5,hiredate)  as hd_minus_5D,
2       dateadd(day,5,hiredate)    as hd_plus_5D,
3       dateadd(month,-5,hiredate) as hd_minus_5M,
4       dateadd(month,5,hiredate)  as hd_plus_5M,
5       dateadd(year,-5,hiredate) as hd_minus_5Y,
6       dateadd(year,5,hiredate)   as hd_plus_5Y
7   from emp
8  where deptno = 10
```

## Discussão

A solução Oracle aproveita o fato de que os valores inteiros representam dias ao realizar aritmética de data. No entanto, isso é verdade apenas para aritmética com tipos DATE. O Oracle também possui tipos TIMESTAMP. Para isso, você deve usar a solução INTERVAL mostrada para o PostgreSQL. Cuidado também, ao passar TIMESTAMPs para funções de data de estilo antigo, como ADD\_MONTHS. Ao fazer isso, você pode perder quaisquer segundos fracionários que tais valores TIMESTAMP possam conter.

A palavra-chave INTERVAL e as strings literais que a acompanham representam a sintaxe SQL padrão ISO. O padrão exige que os valores de intervalo sejam colocados entre aspas simples. PostgreSQL (e Database Oracle9i e posterior) está em conformidade com o padrão. O MySQL se desvia um pouco ao omitir o suporte para as aspas.

## 8.2 Determinando o número de dias entre duas datas

### Problema

Você deseja encontrar a diferença entre duas datas e representar o resultado em dias. Por exemplo, você deseja encontrar a diferença em dias entre os HIREDATEs do funcionário ALLEN e do funcionário WARD.

### Solução

#### DB2

Use duas exibições em linha para localizar os HIREDATEs de WARD e ALLEN. Então subtraia um HIREDATE do outro usando a função DIAS:

```
1 select days(ward_hd) - days(alen_hd)
2   from (
3 select hiredate as ward_hd
4   from emp
5  where ename =
6 'WARD' 6 ) x,
7      (
8 select hiredate as alen_hd
9   from emp
10 where ename = 'ALLEN'
11      ) y
```

#### Oracle e PostgreSQL

Use duas exibições em linha para encontrar os HIREDATEs para WARD e ALLEN e, em seguida, subtraia uma data da outra:

```
1 select ward_hd - alen_hd
2   from (
3 select hiredate as ward_hd
4   from emp
5  where ename =
6 'WARD' 6 ) x,
7      (
8 select hiredate as alen_hd
9   from emp
10 where ename = 'ALLEN'
11      ) y
```

## MySQL e SQLServer

Use a função DATEDIFF para encontrar o número de dias entre duas datas. A versão do DATEDIFF do MySQL requer apenas dois parâmetros (as duas datas que você deseja encontrar a diferença em dias entre elas), e a menor das duas datas deve ser passada primeiro para evitar valores negativos (opostos no SQL Server). A versão da função do SQL Server permite que você especifique o que deseja que o valor de retorno represente (neste exemplo, você deseja retornar a diferença em dias). A solução a seguir usa a versão do SQL Server:

```

1 select datediff(day,allen_hd,ward_hd)
2   from (
3 select hiredate as ward_hd
4   from emp
5 where ename = 'WARD'
6 ) x,
7   (
8 select hiredate as allen_hd
9   from emp
10 where ename = 'ALLEN'
11 ) y

```

Os usuários do MySQL podem simplesmente remover o primeiro argumento da função e inverter a ordem em que ALLEN\_HD e WARD\_HD são passados.

## Discussão

Para todas as soluções, as exibições em linha X e Y retornam os HIREDATES para os funcionários WARD e ALLEN, respectivamente. Por exemplo:

```

select ward_hd, allen_hd
  from (
select hiredate as ward_hd
  from emp
 where ename = 'WARD'
) y,
(
select hiredate as allen_hd
  from emp
 where ename = 'ALLEN'
) x

WARD HD      ALLEN HD
-----
22-FEB-2006 20-FEB-2006

```

Você notará que um produto cartesiano é criado, porque não há junção especificada entre X e Y. Nesse caso, a falta de uma junção é inofensiva, pois as cardinalidades para X e Y são ambas 1; assim, o conjunto de resultados acabará por ter uma linha (obviamente, porque  $1 \times 1 = 1$ ).

Para obter a diferença em dias, basta subtrair um dos dois valores retornados do outro usando métodos apropriados para seu banco de dados.

## 8.3 Determinando o número de dias úteis entre duas datas

### Problema

Dadas duas datas, você deseja descobrir quantos dias “úteis” existem entre elas, incluindo as duas datas propriamente ditas. Por exemplo, se 10 de janeiro for uma terça-feira e 11 de janeiro for uma segunda-feira, o número de dias úteis entre essas duas datas será dois, pois ambos os dias são dias úteis típicos. Para esta receita, um “dia útil” é definido como qualquer dia que não seja sábado ou domingo.

### Solução

Os exemplos de solução localizam o número de dias úteis entre os HIREDATEs de BLAKE e JONES. Para determinar o número de dias úteis entre duas datas, você pode usar uma tabela dinâmica para retornar uma linha para cada dia entre as duas datas (incluindo as datas inicial e final). Feito isso, encontrar o número de dias úteis é simplesmente contar as datas retornadas que não são sábado ou domingo.



Se você também deseja excluir feriados, pode criar uma tabela HOLIDAYS. Em seguida, inclua um predicado NOT IN simples para excluir da solução os dias listados em HOLIDAYS.

### DB2

Use a tabela dinâmica T500 para gerar o número necessário de linhas (representando dias) entre as duas datas. Então conte cada dia que não for um fim de semana. Use a função DAYNAME para retornar o nome do dia da semana de cada data. Por exemplo:

```
1 select sum(case when dayname(jones_hd+t500.id day -1 day)
2 in ( 'Saturday','Sunday' )
3 then 0 else 1
4 end) as days
5 from (
6 select max(case when ename = 'BLAKE'
7 then hiredate
8 end) as blake_hd,
9 max(case when ename = 'JONES'
10 then hiredate
11 end) as jones_hd
12 from emp
```

```

13where ename in ( 'BLAKE','JONES' )
14      ) x,
15      t500
16 where t500.id <= blake_hd-jones_hd+1

```

## MySQL

Use a tabela dinâmica T500 para gerar o número necessário de linhas (dias) entre as duas datas. Então conte cada dia que não for um fim de semana. Use a função DATE\_ADD para adicionar dias a cada data. Use a função DATE\_FORMAT para obter o nome do dia da semana de cada data:

```

1 select sum(case when date_format(
2                     date_add(jones_hd,
3                             interval t500.id-1 DAY),'%a')
4                     in ( 'Sat','Sun' )
5                     then 0 else 1
6                 end) as days
7     from (
8 select max(case when ename = 'BLAKE'
9                     then hiredate
10                    end) as blake_hd,
11      max(case when ename = 'JONES'
12                     then hiredate
13                    end) as jones_hd
14    from emp
15   where ename in ( 'BLAKE','JONES'
16      ) 16      ) x,
17      t500
18 where t500.id <= datediff(blake_hd,jones_hd)+1

```

## Oracle

Use a tabela dinâmica T500 para gerar o número necessário de linhas (dias) entre as duas datas e conte cada dia que não seja um fim de semana. Use a função TO\_CHAR para obter o nome do dia da semana de cada data:

```

1 select sum(case when to_char(jones_hd+t500.id-1,'DY')
2 in ( 'SAT','SUN' )
3 then 0 else 1
4 end) as days
5 from (
6 select max(case when ename = 'BLAKE'
7 then hiredate
8 end) as blake_hd,
9 max(case when ename = 'JONES'
10then hiredate
11end) as jones_hd
12from emp
13where ename in ( 'BLAKE','JONES' )
14      ) x,

```

```

15      t500
16 where t500.id <= blake_hd-jones_hd+1

```

### PostgreSQL

Use a tabela dinâmica T500 para gerar o número necessário de linhas (dias) entre as duas datas. Então conte cada dia que não for um fim de semana. Use a função TO\_CHAR para obter o nome do dia da semana de cada data:

```

1 select sum(case when trim(to_char(jones_hd+t500.id-1,'DAY'))
2                      in ( 'SATURDAY','SUNDAY' )
3                      then 0 else 1
4                  end) as days
5   from (
6 select max(case when ename = 'BLAKE'
7                      then hiredate
8                      end) as blake_hd,
9      max(case when ename = 'JONES'
10                     then hiredate
11                     end) as jones_hd
12   from emp
13 where ename in ( 'BLAKE','JONES' )
14 ) x,
15      t500
16 where t500.id <= blake_hd-jones_hd+1

```

### SQL Server

Use a tabela dinâmica T500 para gerar o número necessário de linhas (dias) entre as duas datas e conte cada dia que não seja um fim de semana. Use a função DATENAME para obter o nome do dia da semana de cada data:

```

1 select sum(case when datename(dw,jones_hd+t500.id-1)
2                      in ( 'SATURDAY','SUNDAY' )
3                      then 0 else 1
4                  end) as days
5   from (
6 selectmax(case when ename = 'BLAKE'
7                      then hiredate
8                      end) as blake_hd,
9      max(case when ename = 'JONES'
10                     then hiredate
11                     end) as jones_hd
12   from emp
13 where ename in ( 'BLAKE','JONES'
14 ) x,
15      t500
16 where t500.id <= datediff(day,jones_hd-blake_hd)+1

```

## Discussão

Embora cada RDBMS exija o uso de diferentes funções incorporadas para determinar o nome de um dia, a abordagem geral da solução é a mesma para cada um. A solução pode ser dividida em duas etapas:

1. Retorna os dias entre a data de início e a data de término (inclusive).
2. Conte quantos dias (ou seja, linhas) existem, excluindo fins de semana.

A visualização em linha X executa a primeira etapa. Se você examinar a exibição em linha X, notará o uso da função de agregação MAX, que a receita usa para remover NULLs. Se o uso de MAX não estiver claro, a seguinte saída pode ajudá-lo a entender. A saída mostra os resultados da visualização em linha X sem MAX:

```
select case when ename = 'BLAKE'
            then hiredate
        end as blake_hd,
      case when ename = 'JONES'
            then hiredate
        end as jones_hd
  from emp
 where ename in ( 'BLAKE', 'JONES' )
```

BLAKE_HD	JONES_HD
	-----
	02-APR-2006
01-MAY-2006	

Sem MAX, duas linhas são retornadas. Ao usar MAX você retorna apenas uma linha em vez de duas, e os NULLs são eliminados:

```
select max(case when ename = 'BLAKE'
            then hiredate
        end) as blake_hd,
      max(case when ename = 'JONES'
            then hiredate
        end) as jones_hd
  from emp
 where ename in ( 'BLAKE', 'JONES' )
```

BLAKE_HD	JONES_HD
-----	-----
01-MAY-2006	02-APR-2006

O número de dias (inclusive) entre as duas datas aqui é 30. Agora que as duas datas estão em uma linha, a próxima etapa é gerar uma linha para cada um desses 30 dias. Para retornar os 30 dias (linhas), utilize a tabela T500. Como cada valor de ID na tabela T500 é simplesmente um maior do que o anterior, adicione cada linha retornada por T500 à data anterior (JONES\_HD) para gerar dias consecutivos a partir de

JONES\_HD até e incluindo BLAKE\_HD. O resultado dessa adição é mostrado aqui (usando a sintaxe do Oracle):

```

select x.* , t500.* , jones_hd+t500.id-1
  from (
select max(case when ename = 'BLAKE'
                  then hiredate
              end) as blake_hd,
      max(case when ename = 'JONES'
                  then hiredate
              end) as jones_hd
   from emp
  where ename in ( 'BLAKE','JONES' )
      ) x,
      t500
 where t500.id <= blake_hd-jones_hd+1

BLAKE_HD      JONES_HD          ID JONES_HD+T5
-----  -----  -----
01-MAY-2006 02-APR-2006      1 02-APR-2006
01-MAY-2006 02-APR-2006      2 03-APR-2006
01-MAY-2006 02-APR-2006      3 04-APR-2006
01-MAY-2006 02-APR-2006      4 05-APR-2006
01-MAY-2006 02-APR-2006      5 06-APR-2006
01-MAY-2006 02-APR-2006      6 07-APR-2006
01-MAY-2006 02-APR-2006      7 08-APR-2006
01-MAY-2006 02-APR-2006      8 09-APR-2006
01-MAY-2006 02-APR-2006      9 10-APR-2006
01-MAY-2006 02-APR-2006     10 11-APR-2006
01-MAY-2006 02-APR-2006     11 12-APR-2006
01-MAY-2006 02-APR-2006     12 13-APR-2006
01-MAY-2006 02-APR-2006     13 14-APR-2006
01-MAY-2006 02-APR-2006     14 15-APR-2006
01-MAY-2006 02-APR-2006     15 16-APR-2006
01-MAY-2006 02-APR-2006     16 17-APR-2006
01-MAY-2006 02-APR-2006     17 18-APR-2006
01-MAY-2006 02-APR-2006     18 19-APR-2006
01-MAY-2006 02-APR-2006     19 20-APR-2006
01-MAY-2006 02-APR-2006     20 21-APR-2006
01-MAY-2006 02-APR-2006     21 22-APR-2006
01-MAY-2006 02-APR-2006     22 23-APR-2006
01-MAY-2006 02-APR-2006     23 24-APR-2006
01-MAY-2006 02-APR-2006     24 25-APR-2006
01-MAY-2006 02-APR-2006     25 26-APR-2006
01-MAY-2006 02-APR-2006     26 27-APR-2006
01-MAY-2006 02-APR-2006     27 28-APR-2006
01-MAY-2006 02-APR-2006     28 29-APR-2006
01-MAY-2006 02-APR-2006     29 30-APR-2006
01-MAY-2006 02-APR-2006     30 01-MAY-2006

```

Se você examinar a cláusula WHERE, notará que adicionou 1 à diferença entre BLAKE\_HD e JONES\_HD para gerar as 30 linhas necessárias (caso contrário, você obteria 29 linhas). Você também notará que subtraiu 1 de T500.ID na lista SELECT da consulta externa, pois os valores para ID começam em 1 e adicionar 1 a JONES\_HD faria com que JONES\_HD fosse excluído da contagem final.

Depois de gerar o número de linhas necessárias para o conjunto de resultados, use uma expressão CASE para “sinalizar” se cada um dos dias retornados é dia da semana ou fim de semana (retorne 1 para dia da semana e 0 para fim de semana). A etapa final é usar a função agregada SUM para calcular o número de 1s para obter a resposta final.

## 8.4 Determinando o número de meses ou anos entre duas datas

### Problema

Você deseja encontrar a diferença entre duas datas em termos de meses ou anos. Por exemplo, você deseja encontrar o número de meses entre o primeiro e o último funcionário contratado e também deseja expressar esse valor como um número de anos.

### Solução

Como sempre há 12 meses em um ano, você pode encontrar o número de meses entre 2 datas e depois dividir por 12 para obter o número de anos. Depois de se sentir confortável com a solução, você vai querer arredondar os resultados para cima ou para baixo, dependendo do que deseja para o ano. Por exemplo, o primeiro HIREDATE na tabela EMP é 17-DEC-1980 e o último é 12-JAN-1983. Se você fizer as contas dos anos (1983 menos 1980), obtém 3 anos, mas a diferença em meses é de aproximadamente 25 (pouco mais de 2 anos). Você deve ajustar a solução como achar melhor. As soluções a seguir retornarão 25 meses e aproximadamente 2 anos.

### DB2 e MySQL

Use as funções YEAR e MONTH para retornar o ano de quatro dígitos e o mês de dois dígitos para as datas fornecidas:

```
1 select mnth, mnth/12
2 from (
3 select (year(max_hd) - year(min_hd))*12 +
4 (month(max_hd) - month(min_hd)) as mnth
5 from (
6 select min(hiredate) as min_hd, max(hiredate) as max_hd
7 from emp
8 ) x
9 ) y
```

Use a função MONTHS\_BETWEEN para encontrar a diferença entre duas datas em meses (para obter anos, basta dividir por 12):

```
1 select months_between(max_hd,min_hd),
2       months_between(max_hd,min_hd)/12
3   from (
4 select min(hiredate) min_hd, max(hiredate) max_hd
5   from emp
6      ) x
```

## PostgreSQL

Use a função EXTRACT para retornar o ano de quatro dígitos e o mês de dois dígitos para as datas fornecidas:

```
1 select mnth, mnth/12
2   from (
3 select ( extract(year from max_hd)
4           extract(year from min_hd) ) * 12
5         +
6         ( extract(month from max_hd)
7           extract(month from min_hd) ) as mnth
8   from (
9 select min(hiredate) as min_hd, max(hiredate) as max_hd
10  from emp
11     ) x
12    ) y
```

## SQL Server

Use a função DATEDIFF para encontrar a diferença entre duas datas e use o argumento DATEPART para especificar meses e anos como as unidades de tempo retornadas:

```
1 select datediff(month,min_hd,max_hd),
2       datediff(year,min_hd,max_hd)
3   from (
4 select min(hiredate) min_hd, max(hiredate) max_hd
5   from emp
6      ) x
```

## Discussão

### DB2, MySQL e PostgreSQL

Depois de extrair o ano e o mês para MIN\_HD e MAX\_HD na solução PostgreSQL, o método para localizar os meses e anos entre MIN\_HD e MAX\_HD é o mesmo para todos os três RDBMs. Esta discussão cobrirá todas as três soluções.

A exibição em linha X retorna os HIREDATEs mais antigos e mais recentes na tabela EMP e é mostrado aqui:

```
select min(hiredate) as min_hd,
       max(hiredate) as max_hd
  from emp
```

MIN HD	MAX HD
17-DEC-1980	12-JAN-1983

Para encontrar os meses entre MAX\_HD e MIN\_HD, multiplique a diferença em anos entre MIN\_HD e MAX\_HD por 12 e adicione a diferença em meses entre MAX\_HD e MIN\_HD. Se você estiver tendo problemas para ver como isso funciona, retorne o componente de data para cada data. Os valores numéricos para os anos e meses são mostrados aqui:

```
select year(max_hd) as max_yr, year(min_hd) as min_yr,
       month(max_hd) as max_mon, month(min_hd) as min_mon
  from (
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp
 ) x
```

MAX_YR	MIN_YR	MAX_MON	MIN_MON
1983	1980	1	12

Olhando para esses resultados, encontrar os meses entre MAX\_HD e MIN\_HD é simplesmente  $(1983 - 1980) \times 12 + (1 - 12)$ . Para encontrar o número de anos entre MIN\_HD e MAX\_HD, divida o número de meses por 12. Novamente, dependendo dos resultados que você está procurando, você desejará arredondar os valores.

### Oracle e SQL Server

A exibição em linha X retorna os HIREDATEs mais antigos e mais recentes na tabela EMP e é mostrado aqui:

```
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp
```

MIN HD	MAX HD
- 17-DEC-1980	12-JAN-
1983	

As funções fornecidas pelo Oracle e SQL Server (MONTHS\_BETWEEN e DATEDIFF, respectivamente) retornarão o número de meses entre duas datas fornecidas. Para encontrar o ano, divida o número de meses por 12.

## 8.5 Determinando o número de segundos, minutos ou horas entre duas datas

### Problema

Você deseja retornar a diferença em segundos entre duas datas. Por exemplo, você deseja retornar a diferença entre HIREDATES de ALLEN e WARD em segundos, minutos e horas.

### Solução

Se você puder encontrar o número de dias entre duas datas, poderá encontrar segundos, minutos e horas, pois são as unidades de tempo que compõem um dia.

#### DB2

Use a função DAYS para encontrar a diferença entre ALLEN\_HD e WARD\_HD em dias. Em seguida, multiplique para encontrar cada unidade de tempo:

```
1 select dy*24 hr, dy*24*60 min, dy*24*60*60 sec
2   from (
3 select ( days(max(case when ename = 'WARD'
4                   then hiredate
5                     end)) -
6           days(max(case when ename = 'ALLEN'
7                   then hiredate
8                     end))
9      ) as dy
10  from emp
11    ) x
```

#### MySQL

Use a função DATEDIFF para retornar o número de dias entre ALLEN\_HD e WARD\_HD. Em seguida, multiplique para encontrar cada unidade de tempo:

```
1 select datediff(day,allen_hd,ward_hd)*24 hr,
2       datediff(day,allen_hd,ward_hd)*24*60 min,
3       datediff(day,allen_hd,ward_hd)*24*60*60 sec
4   from (
5 select max(case when ename = 'WARD'
6                   then hiredate
7                     end) as ward_hd,
8       max(case when ename = 'ALLEN'
9                   then hiredate
10                  end) as allen_hd
11  from emp
12    ) x
```

## SQL Server

Use a função DATEDIFF para retornar o número de dias entre ALLEN\_HD e WARD\_HD. Em seguida, use o argumento DATEPART para especificar a unidade de tempo necessária:

```

1 select datediff(day,allen_hd,ward_hd,hour) as hr,
2       datediff(day,allen_hd,ward_hd,minute) as min,
3       datediff(day,allen_hd,ward_hd,second) as sec
4   from (
5 select max(case when ename = 'WARD'
6           then hiredate
7           end) as ward_hd,
8       max(case when ename = 'ALLEN'
9           then hiredate
10          end) as allen_hd
11  from emp
12      ) x

```

## Oracle e PostgreSQL

Use a subtração para retornar o número de dias entre ALLEN\_HD e WARD\_HD. Em seguida, multiplique para encontrar cada unidade de tempo:

```

1 select dy*24 as hr, dy*24*60 as min, dy*24*60*60 as sec
2   from (
3 select (max(case when ename = 'WARD'
4           then hiredate
5           end) -
6       max(case when ename = 'ALLEN'
7           then hiredate
8           end)) as dy
9   from emp
10      ) x

```

## Discussão

A exibição em linha X para todas as soluções retorna os HIREDATES para WARD e ALLEN, conforme mostrado aqui:

```

select max(case when ename = 'WARD'
            then hiredate
            end) as ward_hd,
       max(case when ename = 'ALLEN'
            then hiredate
            end) as allen_hd
  from emp

```

WARD_HD	ALLEN_HD
-----	-----
22-FEV-2006	20-FEV-2006

Multiplique o número de dias entre WARD\_HD e ALLEN\_HD por 24 (horas em um dia), 1440 (minutos em um dia) e 86400 (segundos em um dia).

## 8.6 Contando as ocorrências de dias da semana em um ano

### Problema

Você deseja contar o número de vezes que cada dia da semana ocorre em um ano.

### Solução

Para encontrar o número de ocorrências de cada dia da semana em um ano, você deve:

1. Gere todas as datas possíveis no ano.
2. Formate as datas de forma que resolvam o nome de seus respectivos dias da semana.
3. Conte a ocorrência de cada nome de dia da semana.

### DB2

Use WITH recursivo para evitar a necessidade de SELECT em uma tabela com pelo menos 366 linhas. Use a função DAYNAME para obter o nome do dia da semana para cada data e conte a ocorrência de cada uma:

```
1 with x (start_date,end_date)
2 as (
3 select start_date,
4       start_date + 1 year end_date
5   from (
6 select (current_date
7       dayofyear(current_date) day)
8       +1 day as start_date
9   from t1
10      ) tmp
11 union all
12 select start_date + 1 day, end_date
13   from x
14  where start_date + 1 day < end_date
15 )
16 select dayname(start_date),count(*)
17   from x
18  group by dayname(start_date)
```

### MySQL

Selecione na tabela T500 para gerar linhas suficientes para retornar todos os dias do ano. Use a função DATE\_FORMAT para obter o nome do dia da semana de cada data e, em seguida, conte a ocorrência de cada nome:

```

1 select date_format(
2         date_add(
3             cast(
4                 concat(year(current_date),'-01-01')
5                     as date),
6                     interval t500.id-1 day),
7                     '%W') day,
8         count(*)
9     from t500
10    where t500.id <= datediff(
11                    cast(
12                        concat(year(current_date)+1,'-01-01')
13                            as date),
14                            cast(
15                                concat(year(current_date),'-01-01')
16                                    as date))
17   group by date_format(
18         date_add(
19             cast(
20                 concat(year(current_date), '-01-01')
21                     as date),
22                     interval t500.id-1 day),
23                     '%W')

```

## Oracle

Você pode usar o CONNECT BY recursivo para retornar todos os dias em um ano:

```

1 with x as (
2 select level lvl
3   from dual
4  connect by level <= (
5      add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
6  )
7  )
8 select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
9   from x
10  group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')

```

## PostgreSQL

Use a função interna GENERATE\_SERIES para gerar uma linha para cada dia do ano. Em seguida, use a função TO\_CHAR para obter o nome do dia da semana de cada data. Por fim, conte a ocorrência de cada nome de dia da semana. Por exemplo:

```

1 select to_char(
2 cast(
3 date_trunc('year',current_date)
4 as date) + gs.id-1,'DAY'),
5 count(*)
6 from generate_series(1,366) gs(id)
7 where gs.id <= (cast

```

```
8 ( date_trunc('year',current_date) +
9 interval '12 month' as date) -
10 cast(date_trunc('year',current_date)
11 as date))
12 group by to_char(
13 cast(
14 date_trunc('year',current_date)
15 as date) + gs.id-1,'DAY')
```

## SQL Server

Use o WITH recursivo para evitar a necessidade de SELECT em uma tabela com pelo menos 366 linhas. Use a função DATENAME para obter o nome do dia da semana de cada data e, em seguida, conte a ocorrência de cada nome. Por exemplo:

```
1 with x (start_date,end_date)
2 as (
3 select start_date,
4      dateadd(year,1,start_date) end_date
5   from (
6 select cast(
7      cast(year(getdate()) as varchar) + '-01-01'
8      as datetime) start_date
9   from t1
10      ) tmp
11 union all
12 select dateadd(day,1,start_date), end_date
13   from x
14 where dateadd(day,1,start_date) < end_date
15 )
16 select datename(dw,start_date),count(*)
17   from x
18  group by datename(dw,start_date)
19 OPTION (MAXRECURSION 366)
```

## Discussão

### DB2

A visualização inline TMP, na visualização recursiva WITH X, retorna o primeiro dia do ano atual e é mostrado aqui:

```
select (current_date
       dayofyear(current_date) day)
       +1 day as start_date
  from t1

START_DATE
-----
01-JAN-2005
```

A próxima etapa é adicionar um ano a START\_DATE para que você tenha as datas de início e término. Você precisa conhecer os dois porque deseja gerar todos os dias do ano. START\_DATE e END\_DATE são mostrados aqui:

```
select start_date,
       start_date + 1 year end_date
  from (
select (current_date
         dayofyear(current_date) day)
       +1 day as start_date
  from t1
     ) tmp

START_DATE   END_DATE
-----
01-JAN-2005  01-JAN-2006
```

A próxima etapa é incrementar recursivamente START\_DATE em um dia, parando antes que seja igual a END\_DATE. Uma parte das linhas retornadas pela visão recursiva X é mostrada aqui:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (current_date -
         dayofyear(current_date) day)
       +1 day as start_date
  from t1
     ) tmp
union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select * from x

START_DATE   END_DATE
-----
01-JAN-2005  01-JAN-2006
2- JAN-2005  01-JAN-2006
3- JAN-2005  01-JAN-2006
...
29-JAN-2005  01-JAN-2006
30-JAN-2005  01-JAN-2006
31-JAN-2005  01-JAN-2006
...
1- DEC-2005  01-JAN-2006
2- DEC-2005  01-JAN-2006
3- DEC-2005  01-JAN-2006
...
```

```
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006
```

A etapa final é usar a função DAYNAME nas linhas retornadas pela visão recursiva X e contar quantas vezes cada dia da semana ocorre. O resultado final é mostrado aqui:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (
        current_date -
      dayofyear(current_date) day)
      +1 day as start_date
   from t1
     ) tmp
union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select dayname(start_date),count(*)
  from x
 group by dayname(start_date)

START_DATE COUNT(*)
-----
FRIDAY      52
MONDAY      52
SATURDAY    53
SUNDAY      52
THURSDAY    52
TUESDAY     52
WEDNESDAY   52
```

## MySQL

Esta solução seleciona a tabela T500 para gerar uma linha para cada dia do ano. O comando na linha 4 retorna o primeiro dia do ano atual. Ele faz isso retornando o ano da data retornada pela função CURRENT\_DATE e então acrescentando um mês e um dia (segundo o formato de data padrão do MySQL). O resultado é mostrado aqui:

```
select concat(year(current_date),'-01-01')
  from t1

START_DATE
-----
01-JAN-2005
```

Agora que você tem o primeiro dia do ano atual, use a função DATEADD para somar cada valor de T500.ID para gerar cada dia do ano. Use a função DATE\_FORMAT para retornar o dia da semana para cada data. Para gerar o número necessário de linhas da tabela T500, encontre a diferença em dias entre o primeiro dia do ano atual e o primeiro dia do próximo ano e retorne esse número de linhas (será 365 ou 366). Uma parte dos resultados é mostrada aqui:

```

select date_format(
    date_add(
        cast(
            concat(year(current_date), '-01-01')
                as date),
            interval t500.id-1 day),
        '%W') day
from t500
where t500.id <= datediff(
    cast(
        concat(year(current_date)+1, '-01-01')
            as date),
        cast(
            concat(year(current_date), '-01-01')
                as date))

```

DAY
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005

Agora que você pode retornar todos os dias do ano atual, conte as ocorrências de cada dia da semana retornando pela função DAYNAME. Os resultados finais são mostrados aqui:

```

select date_format(
    date_add(
        cast(
            concat(year(current_date), '-01-01')
                as date),
            interval t500.id-1 day),
        '%W') day,
    count(*)

```

```

from t500
where t500.id <= datediff(
    cast(
        concat(year(current_date)+1,'-01-01')
        as date),
    cast(
        concat(year(current_date),'-01-01')
        as date))
group by date_format(
    date_add(
        cast(
            concat(year(current_date),'-01-01')
            as date),
        interval t500.id-1 day),
    '%W')

DAY      COUNT(*)
-----
FRIDAY      52
MONDAY      52
SATURDAY    53
SUNDAY      52
THURSDAY    52
TUESDAY     52
WEDNESDAY   52

```

## Oracle

As soluções fornecidas selecionam a tabela T500 (uma tabela dinâmica) ou usam CONNECT BY e WITH recursivamente para gerar uma linha para cada dia no ano atual. A chamada para a função TRUNC trunca a data atual para o primeiro dia do ano atual.

Se você estiver usando a solução CONNECT BY/WITH, poderá usar a pseudo-column LEVEL para gerar números sequenciais começando em um. Para gerar o número necessário de linhas necessárias para esta solução, filtre ROWNUM ou LEVEL na diferença de dias entre o primeiro dia do ano atual e o primeiro dia do próximo ano (será 365 ou 366 dias). A próxima etapa é incrementar cada dia adicionando ROWNUM ou LEVEL ao primeiro dia do ano atual. Os resultados parciais são mostrados aqui:

```

/* Oracle 9i e posterior */
with x as (
select level lvl
  from dual
connect by level <= (
    add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
)
)

select trunc(sysdate,'y')+lvl-1      from x

```

Se você estiver usando a solução de tabela dinâmica, poderá usar qualquer tabela ou exibição com pelo menos 366 linhas. E como o Oracle tem ROWNUM, não há necessidade de uma tabela com valores incrementais a partir de um. Considere o exemplo a seguir, que usa a tabela dinâmica T500 para retornar todos os dias do ano atual:

```
/* Oracle 8i and earlier */
select trunc(sysdate,'y')+rownum-1 start_date
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                  - trunc(sysdate,'y'))

START_DATE
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

Independentemente da abordagem adotada, você eventualmente deve usar a função TO\_CHAR para retornar o nome do dia da semana para cada data e, em seguida, contar a ocorrência de cada nome. Os resultados finais são mostrados aqui:

```
/* Oracle 9i and later */
with x as (
  select level lvl
    from dual
   connect by level <=
      add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
)
select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
  from x
 group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')

/* Oracle 8i and earlier */
select to_char(trunc(sysdate,'y')+rownum-1,'DAY') start_date,
       count(*)
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                  - trunc(sysdate,'y'))
 group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')
```

START_DATE	COUNT(*)
FRIDAY	52
MONDAY	52
SATURDAY	53
SUNDAY	52
THURSDAY	52
TUESDAY	52
WEDNESDAY	52

## PostgreSQL

O primeiro passo é usar a função DATE\_TRUNC para retornar o ano da data atual (mostrada aqui, selecionando contra T1 para que apenas uma linha seja retornada):

```
select cast(
    date_trunc('year',current_date)
    as date) as start_date
from t1

START_DATE
-----
01-JAN-2005
```

A próxima etapa é selecionar uma fonte de linha (qualquer expressão de tabela, na verdade) com pelo menos 366 linhas. A solução usa a função GENERATE\_SERIES como fonte de linha. Você pode, obviamente, usar a tabela T500. Em seguida, adicione um dia ao primeiro dia do ano atual até retornar todos os dias do ano (mostrado aqui):

```
select cast( date_trunc('year',current_date)
            as date) + gs.id-1 as start_date
  from generate_series (1,366) gs(id)
 where gs.id <= (cast
                  ( date_trunc('year',current_date) +
                    interval '12 month' as date) -
                  cast(date_trunc('year',current_date)
                        as date))

START_DATE
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
```

```

...
29-DEC-2005
30-DEC-2005
31-DEC-2005

```

A etapa final é usar a função TO\_CHAR para retornar o nome do dia da semana para cada data e, em seguida, contar a ocorrência de cada nome. Os resultados finais são mostrados aqui:

```

select to_char(
    cast(
        date_trunc('year',current_date)
            as date) + gs.id-1,'DAY') as start_dates,
    count(*)
from generate_series(1,366) gs(id)
where gs.id <= (cast
    ( date_trunc('year',current_date) +
        interval '12 month' as date) -
    cast(date_trunc('year',current_date)
        as date))
group by to_char(
    cast(
        date_trunc('year',current_date)
            as date) + gs.id-1,'DAY')

START_DATE      COUNT(*)
-----
FRIDAY          52
MONDAY          52
SATURDAY        53
SUNDAY          52
THURSDAY        52
TUESDAY         52
WEDNESDAY       52

```

## SQL Server

A visualização inline TMP, na visualização recursiva WITH X, retorna o primeiro dia do ano atual e é mostrado aqui:

```

select cast(
    cast(year(getdate()) as varchar) + '-01-01'
        as datetime) start_date
from t1

START_DATE
-----
01-JAN-2005

```

Depois de retornar o primeiro dia do ano atual, adicione um ano a START\_DATE para que você tenha as datas de início e término. Você precisa conhecer os dois porque deseja gerar todos os dias do ano.

START\_DATE e END\_DATE são mostrados aqui:

```
select start_date,
       dateadd(year,1,start_date) end_date
  from (
select cast(
      cast(year(getdate()) as varchar) + '-01-01'
            as datetime) start_date
  from t1
    ) tmp

START_DATE END_DATE
-----
01-JAN-2005 01-JAN-2006
```

Em seguida, incremente recursivamente START\_DATE em um dia e pare antes que seja igual a END\_DATE. Uma parte das linhas retornadas pela visão recursiva X é mostrada abaixo:

```
with x (start_date,end_date)
  as (
  select start_date,
         dateadd(year,1,start_date) end_date
    from (
  select cast(
      cast(year(getdate()) as varchar) + '-01-01'
            as datetime) start_date
    from t1
      ) tmp
  union all
  select dateadd(day,1,start_date), end_date
    from x
   where dateadd(day,1,start_date) < end_date
  )
  select * from x
OPTION (MAXRECURSION 366)

START_DATE END_DATE
-----
01-JAN-2005 01-JAN-2006
2- JAN-2005 01-JAN-2006
3- JAN-2005 01-JAN-2006
...
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
...
1- DEC-2005 01-JAN-2006
2- DEC-2005 01-JAN-2006
3- DEC-2005 01-JAN-2006
...
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
29-DEC-2005 01-JAN-2006
```

A etapa final é usar a função DATENAME nas linhas retornadas pela visão recursiva X e contar quantas vezes cada dia da semana ocorre. O resultado final é mostrado aqui:

```
with x(start_date,end_date)
  as (
    select start_date,
           dateadd(year,1,start_date) end_date
      from (
    select cast(
              cast(year(getdate()) as varchar) + '-01-01'
                as datetime) start_date
        from t1
          ) tmp
  union all
    select dateadd(day,1,start_date), end_date
      from x
     where dateadd(day,1,start_date) < end_date
  )
  select datename(dw,start_date), count(*)
    from x
   group by datename(dw,start_date)
OPTION (MAXRECURSION 366)
```

START_DATE	COUNT(*)
FRIDAY	52
MONDAY	52
SATURDAY	53
SUNDAY	52
THURSDAY	52
TUESDAY	52
WEDNESDAY	52

## 8.7 Determinando a diferença de data entre o registro atual e o próximo registro

### Problema

Você deseja determinar a diferença em dias entre duas datas (especificamente datas armazenadas em duas linhas diferentes). Por exemplo, para cada funcionário no DEPTNO 10, você deseja determinar o número de dias entre o dia em que foi contratado e o dia em que o próximo funcionário (pode ser em outro departamento) foi contratado.

O truque para a solução desse problema é encontrar o HIREDATE mais antigo após a contratação do funcionário atual. Depois disso, basta usar a técnica de [Receita 8.2](#) para encontrar a diferença em dias.

## DB2

Use uma subconsulta escalar para localizar o próximo HIREDATE em relação ao HIREDATE atual. Em seguida, use a função DAYS para encontrar a diferença em dias:

```
1 select x.*,
2       days(x.next_hd) - days(x.hiredate) diff
3   from (
4 select e.deptno, e.ename, e.hiredate,
5       lead(hiredate)over(order by hiredate) next_hd
6   from emp e
7 where e.deptno = 10
8 ) x
```

## MySQL e SQLServer

Use a função lead para acessar a próxima linha. A versão SQL Server de DATEDIFF é usada aqui:

```
1 select x.ename, x.hiredate, x.next_hd,
2       datediff(x.hiredate,x.next_hd,day) as diff
3   from (
4 select deptno, ename, hiredate,
5       lead(hiredate)over(order by hiredate) as next_hd
6   from emp e
7      ) x
8 where e.deptno=10
```

Os usuários do MySQL podem excluir o primeiro argumento (“dia”) e mudar a ordem dos dois argumentos restantes:

```
2       datediff(x.next_hd, x.hiredate) diff
```

## Oracle

Use a função de janela LEAD OVER para acessar o próximo HIREDATE relativo à linha atual, facilitando assim a subtração:

```
1 select ename, hiredate, next_hd,
2       next_hd - hiredate diff
3   from (
4 select deptno, ename, hiredate,
5       lead(hiredate)over(order by hiredate) next_hd
6   from emp
7      )
8 where deptno=10
```

## PostgreSQL

Use uma subconsulta escalar para localizar o próximo HIREDATE em relação ao HIREDATE atual. Em seguida, use subtração simples para encontrar a diferença em dias:

```

1 select x.*,
2       x.next_hd - x.hiredate as diff
3   from (
4 select e.deptno, e.ename, e.hiredate,
5       lead(hiredate)over(order by hiredate) as next_hd
6   from emp e
7  where e.deptno = 10
8 ) x

```

## Discussão

Apesar das diferenças de sintaxe, a abordagem é a mesma para todas essas soluções: use a função de janela LEAD e depois encontre a diferença em dias entre as duas usando a técnica descrita em [Receita 8.2](#).

A capacidade de acessar linhas em torno de sua linha atual sem junções adicionais fornece um código mais legível e eficiente. Ao trabalhar com funções de janela, lembre-se de que elas são avaliadas após a cláusula WHERE, daí a necessidade de uma visualização inline na solução. Se você mover o filtro em DEPTNO para a exibição em linha, os resultados serão alterados (somente os HIREDATEs de DEPTNO 10 serão considerados). Uma observação importante a ser mencionada sobre as funções LEAD e LAG do Oracle é seu comportamento na presença de duplicatas. No prefácio, mencionamos que essas receitas não são codificadas “defensivamente” porque há muitas condições que não se pode prever que podem quebrar o código. Ou, mesmo que se possa prever todos os problemas, às vezes o SQL resultante se torna ilegível. Portanto, na maioria dos casos, o objetivo de uma solução é introduzir uma técnica: uma que você pode usar em seu sistema de produção, mas que deve ser testada e muitas vezes ajustada para funcionar com seus dados específicos. Nesse caso, porém, há uma situação que discutiremos simplesmente porque a solução alternativa pode não ser tão óbvia, principalmente para aqueles que vêm de sistemas não-Oracle. Neste exemplo, não há HIREDATEs duplicados na tabela EMP, mas certamente é possível (e provavelmente provável) que existam valores de data duplicados em suas tabelas. Considere os funcionários no DEPTNO 10 e seus HIREDATEs:

```

select ename, hiredate
  from emp
 where deptno=10
 order by 2

```

ENAME	HIREDATE
CLARK	09-JUN-2006
KING	17-NOV-2006
MILLER	23-JAN-2007

Para este exemplo, vamos inserir quatro duplicatas de modo que haja cinco funcionários (incluindo KING) contratados em 17 de novembro:

```

insert into emp (empno,ename,deptno,hiredate)
values (1,'ant',10,to_date('17-NOV-2006'))

insert into emp (empno,ename,deptno,hiredate)
values (2,'joe',10,to_date('17-NOV-2006'))

insert into emp (empno,ename,deptno,hiredate)
values (3,'jim',10,to_date('17-NOV-2006'))

insert into emp (empno,ename,deptno,hiredate)
values (4,'choi',10,to_date('17-NOV-2006'))

select ename, hiredate
  from emp
 where deptno=10
 order by 2

ENAME    HIREDATE
-----
CLARK    09-JUN-2006
ant      17-NOV-2006
joe      17-NOV-2006
KING     17-NOV-2006
jim      17-NOV-2006
choi     17-NOV-2007
MILLER   23-JAN-2007

```

Agora existem vários funcionários no DEPTNO 10 contratados no mesmo dia. Se você tentar usar a solução proposta (mover o filtro para a exibição em linha para se preocupar apenas com funcionários em DEPTNO 10 e seus HIREDATES) nesse conjunto de resultados, obterá a seguinte saída:

```

select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate)over(order by hiredate) next_hd
  from emp
 where deptno=10
      )

```

ENAME	HIREDATE	NEXT HD	DIFF
CLARK	09-JUN-2006	17-NOV-2006	161
ant	17-NOV-2006	17-NOV-2006	0
joe	17-NOV-2006	17-NOV-2006	0
KING	17-NOV-2006	17-NOV-2006	0
jim	17-NOV-2006	17-NOV-2006	0

choi	17-NOV-2006	23-JAN-2007	67
MILLER	23-JAN-2007	(null)	(null)

Olhando para os valores de DIFF para quatro dos cinco funcionários contratados no mesmo dia, você pode ver que o valor é zero. Isso não está correto. Todos os funcionários contratados no mesmo dia devem ter suas datas avaliadas em relação ao HIREDATE da próxima data em que um funcionário foi contratado (ou seja, todos os funcionários contratados em 17 de novembro devem ser avaliados em relação ao HIREDATE da MILLER). O problema aqui é que a função LEAD ordena as linhas por HIREDATE, mas não ignora as duplicatas. Assim, por exemplo, quando HIREDATE do funcionário ANT é avaliado em relação ao HIREDATE do funcionário JOE, a diferença é zero, portanto, um valor DIFF de zero para ANT. Felizmente, a Oracle forneceu uma solução fácil para situações como esta. Ao invocar a função LEAD, você pode passar um argumento para LEAD para especificar exatamente onde está a linha futura (ou seja, é a próxima linha, 10 linhas depois, etc.). Então, olhando para o funcionário ANT, em vez de olhar uma linha à frente, você precisa olhar cinco linhas à frente (você quer pular todas as outras duplicatas), porque é onde MILLER está. Se você olhar para o funcionário JOE, ele está a quatro fileiras de MILLER, JIM está a três fileiras de MILLER, KING está a duas fileiras de MILLER e o menino bonito CHOI está a uma fileira de MILLER. Para obter a resposta correta, basta passar a distância de cada funcionário para a MILLER como argumento para a LEAD. A solução é mostrada aqui:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
      )
     )
    )
```

ENAME	HIREDATE	NEXT HD	DIFF
CLARK	09-JUN-2006	17-NOV-2006	161
ant	17-NOV-2006	23-JAN-2007	67
joe	17-NOV-2006	23-JAN-2007	67
jim	17-NOV-2006	23-JAN-2007	67
choi	17-NOV-2006	23-JAN-2007	67
KING	17-NOV-2006	23-JAN-2007	67
MILLER	23-JAN-2007	(null)	(null)

Agora os resultados estão corretos. Todos os funcionários contratados no mesmo dia têm seus HIREDATES avaliados em relação ao próximo HIREDATE, não um HIREDATE que corresponda ao seu. Se a solução alternativa não for imediatamente óbvia, simplesmente decomponha a consulta.

Comece com a visualização em linha:

```
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
```

DEPTNO	ENAME	HIREDATE	CNT	RN
10	CLARK	09-JUN-2006	1	1
10	ant	17-NOV-2006	5	1
10	joe	17-NOV-2006	5	2
10	jim	17-NOV-2006	5	3
10	choi	17-NOV-2006	5	4
10	KING	17-NOV-2006	5	5
10	MILLER	23-JAN-2007	1	1

A função de janela COUNT OVER conta o número de vezes que cada HIREDATE ocorre e retorna esse valor para cada linha. Para os HIREDATES duplicados, um valor de 5 é retornado para cada linha com esse HIREDATE. A função de janela ROW\_NUMBER OVER classifica cada funcionário por EMPNO. A classificação é particionada por HIREDATE, portanto, a menos que haja HIREDATES duplicados, cada funcionário terá uma classificação de 1. Neste ponto, todas as duplicatas foram contadas e classificadas, e a classificação pode servir como a distância até o próximo HIREDATE (HIREDATE de MILLER). Você pode ver isso subtraindo RN de CNT e adicionando 1 para cada linha ao chamar LEAD:

```
seleciona o departamento, ename, contratou,
select deptno, ename, hiredate,
       cnt-rn+1 distance_to_miller,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
  )
```

DEPTNO	ENAME	HIREDATE	DISTANCE_TO_MILLER	NEXT HD
10	CLARK	09-JUN-2006	1	17-NOV-2006
10	ant	17-NOV-2006	5	23-JAN-2007
10	joe	17-NOV-2006	4	23-JAN-2007
10	jim	17-NOV-2006	3	23-JAN-2007
10	choi	17-NOV-2006	2	23-JAN-2007
10	KING	17-NOV-2006	1	23-JAN-2007
10	MILLER	23-JAN-2007	1	(null)

Como você pode ver, ao passar a distância adequada para pular à frente, a função LEAD realiza a subtração nas datas corretas.

## 8.8 Resumindo

As datas são um tipo de dados comum, mas têm suas próprias peculiaridades, pois têm mais estrutura do que tipos de dados numéricos simples. Em termos relativos, há menos padronização entre os fornecedores do que em muitas outras áreas, mas cada implementação tem um grupo principal de funções que executam as mesmas tarefas, mesmo quando a sintaxe é ligeiramente diferente. Dominar este grupo principal garantirá seu sucesso com as datas.

## CAPÍTULO 9

# Manipulação de Data

Este capítulo apresenta receitas para pesquisar e modificar datas. Consultas envolvendo datas são muito comuns. Portanto, você precisa saber pensar ao trabalhar com datas e precisa ter um bom entendimento das funções que sua plataforma RDBMS oferece para manipulá-las. As receitas neste capítulo formam uma base importante para o trabalho futuro à medida que você avança para consultas mais complexas envolvendo não apenas datas, mas também horas.

Antes de entrar nas receitas, queremos reforçar o conceito (mencionado no prefácio) de usar essas soluções como diretrizes para resolver seus problemas específicos. Tente pensar no “quadro geral”. Por exemplo, se uma receita resolve um problema para o mês atual, lembre-se de que você pode usar a receita para qualquer mês (com pequenas modificações), não apenas para o mês usado na receita. Novamente, essas receitas são diretrizes, a opção final absoluta. Não há como um livro conter uma resposta para todos os seus problemas, mas se você entender o que é apresentado aqui, modificar essas soluções para atender às suas necessidades é trivial. Considere também versões alternativas dessas soluções. Por exemplo, se a solução usa uma determinada função fornecida pelo seu RDBMS, vale a pena o tempo e o esforço para descobrir se existe uma alternativa — talvez uma que seja mais ou menos eficiente do que a apresentada aqui. Conhecer suas opções fará de você um programador SQL melhor.



As receitas apresentadas neste capítulo usam tipos de dados de data simples. Se você estiver usando tipos de dados de data mais complexos, precisará ajustar as soluções de acordo.

## 9.1 Determinando se um ano é um ano bissexto

### Problema

Você deseja determinar se o ano atual é um ano bissexto.

### Solução

Se você já trabalha com SQL há algum tempo, não há dúvida de que já se deparou com várias técnicas para resolver esse problema. Quase todas as soluções que encontramos funcionam bem, mas a apresentada nesta receita é provavelmente a mais simples. Esta solução simplesmente verifica o último dia de fevereiro; se for dia 29, o ano atual é bissexto.

#### DB2

Use a cláusula recursiva WITH para retornar todos os dias de fevereiro. Use a função agregada MAX para determinar o último dia de fevereiro:

```
1  with x (dy,mth)
2    as (
3 select dy, month(dy)
4   from (
5 select (current_date -
6           dayofyear(current_date) days +1 days)
7           +1 months as dy
8   from t1
9      ) tmp1
10 union all
11 select dy+1 days, mth
12   from x
13 where month(dy+1 day) =
mth 14 )
15 select max(day(dy))
16   from x
```

#### Oracle

Use a função LAST\_DAY para encontrar o último dia de fevereiro:

```
1 select to_char(
2           last_day(add_months(trunc(sysdate,'y'),1))
, 3           'DD')
4   from t1
```

## PostgreSQL

Use a função GENERATE\_SERIES para retornar todos os dias de fevereiro e, em seguida, use a função agregada MAX para encontrar o último dia de fevereiro:

```

1 select max(to_char(tmp2.dy+x.id,'DD')) as dy
2   from (
3 select dy, to_char(dy,'MM') as mth
4   from (
5 select cast(cast(
6           date_trunc('year',current_date) as date)
7           + interval '1 month' as date) as dy
8   from t1
9       ) tmp1
10      ) tmp2, generate_series (0,29) x(id)
11 where to_char(tmp2.dy+x.id,'MM') = tmp2.mth

```

## MySQL

Use a função LAST\_DAY para encontrar o último dia de fevereiro:

```

1 select day(
2       last_day(
3       date_add(
4       date_add(
5       date_add(current_date,
6           interval -dayofyear(current_date) day),
7           interval 1 day),
8           interval 1 month))) dy
9   from t1

```

## SQL Server

Use a cláusula recursiva WITH para retornar todos os dias de fevereiro. Use a função agregada MAX para determinar o último dia de fevereiro:

```

select coalesce
      (day
       (cast(concat
             (year(getdate()),'-02-
              29')
             as date))
      ,28);

```

## Discussão

### DB2

A visualização inline TMP1 na visualização recursiva X retorna o primeiro dia de fevereiro por:

1. Começando com a data atual

2. Usando DAYOFYEAR para determinar o número de dias no ano atual que a data atual representa
3. Subtraindo esse número de dias da data atual para obter 31 de dezembro do ano anterior e adicionando um para obter 1º de janeiro do ano atual
4. Adicionando um mês para chegar a 1º de fevereiro

O resultado de toda essa matemática é mostrado aqui:

```
select (current_date
        dayofyear(current_date) days +1 days) +1 months as dy
      from t1

DY
-----
01-FEB-2005
```

A próxima etapa é retornar o mês da data retornada pela exibição inline TMP1 usando a função MONTH:

```
select dy, month(dy) as mth
      from (
select (current_date
        dayofyear(current_date) days +1 days) +1 months as dy
      from t1
    ) tmp1

DY      MTH
-----
01-FEB-2005    2
```

Os resultados apresentados até agora fornecem o ponto de partida para a operação recursiva gerada a cada dia de fevereiro. Para retornar todos os dias em fevereiro, adicione repetidamente um dia a DY até que você não esteja mais no mês de fevereiro. Uma parte dos resultados da operação WITH é mostrada aqui:

```
with x (dy,mth)
      as (
select dy, month(dy)
      from (
select (current_date -
        dayofyear(current_date) days +1 days) +1 months as dy
      from t1
    ) tmp1
union all
select dy+1 days, mth
      from x
     where month(dy+1 day) = mth
  )
select dy,mth
      from x
```

DY	MTH
-	-
- 01-FEB-2005	2
..	
10-FEB-2005	2
..	
28-FEB-2005	2

O último passo é usar a função MAX da coluna DY para retornar o último dia de fevereiro; se for dia 29, você está em um ano bissexto.

### Oracle

O primeiro passo é encontrar o início do ano usando a função TRUNC:

```
select trunc(sysdate,'y')
      from t1
```

DY
-
-
01-JAN-2005

Como o primeiro dia do ano é 1º de janeiro, o próximo passo é adicionar um mês para chegar a 1º de fevereiro:

```
select add_months(trunc(sysdate,'y'),1) dy
      from t1
```

DY
-
-
01-FEB-2005

O próximo passo é usar a função LAST\_DAY para encontrar o último dia de fevereiro:

```
select last_day(add_months(trunc(sysdate,'y'),1)) dy
      from t1
```

DY
-
-
28-FEB-2005

A etapa final (que é opcional) é usar TO\_CHAR para retornar 28 ou 29.

### PostgreSQL

A primeira etapa é examinar os resultados retornados pela visualização inline TMP1. Use a função DATE\_TRUNC para encontrar o início do ano atual e converter esse resultado em DATE:

```
select cast(date_trunc('year',current_date) as date) as dy
      from t1
```

DY

```
-----  
01-JAN-2005
```

O próximo passo é adicionar um mês ao primeiro dia do ano atual para obter o primeiro dia de fevereiro, lançando o resultado como uma data:

```
select cast(cast(  
    date_trunc('year',current_date) as date)  
        + interval '1 month' as date) as dy  
from t1
```

```
DY  
-----  
01-FEB-2005
```

Em seguida, retorne DY da exibição inline TMP1 junto com o mês numérico de DY. Retorne o mês numérico usando a função TO\_CHAR:

```
select dy, to_char(dy,'MM') as mth  
  from (  
select cast(cast(  
    date_trunc('year',current_date) as date)  
        + interval '1 month' as date) as dy  
  from t1  
 ) tmp1
```

```
DY      MTH  
-----  
01-FEB-2005  2
```

Os resultados mostrados até agora compreendem o conjunto de resultados da visualização inline TMP2. Sua próxima etapa é usar a função extremamente útil GENERATE\_SERIES para retornar 29 linhas (valores de 1 a 29). Cada linha retornada por GENERATE\_SERIES (aliased X) é adicionada a DY a partir da visualização inline TMP2. Os resultados parciais são mostrados aqui:

```
select tmp2.dy+x.id as dy, tmp2.mth  
  from (  
select dy, to_char(dy,'MM') as mth  
  from (  
select cast(cast(  
    date_trunc('year',current_date) as date)  
        + interval '1 month' as date) as dy  
  from t1  
 ) tmp1  
 ) tmp2, generate_series (0,29) x(id)  
where to_char(tmp2.dy+x.id,'MM') = tmp2.mth
```

```
DY      MTH  
-----  
01-FEB-2005  02  
...  
10-FEB-2005  02
```

```
...
28-FEV-2005 02
```

A etapa final é usar a função MAX para retornar no último dia de fevereiro. A função TO\_CHAR é aplicada a esse valor e retornará 28 ou 29.

## MySQL

A primeira etapa é encontrar o primeiro dia do ano atual, subtraindo da data atual o número de dias do ano e adicionando um dia. Faça tudo isso com a função DATE\_ADD:

```
select date_add(
    date_add(current_date,
        interval -dayofyear(current_date) day),
    interval 1 day) dy
from t1

DY
-----
01-JAN-2005
```

Em seguida, adicione um mês novamente usando a função DATE\_ADD:

```
select date_add(
    date_add(
        date_add(current_date,
            interval -dayofyear(current_date) day),
        interval 1 day),
    interval 1 month) dy
from t1

DY
-----
01-FEB-2005
```

Agora que você chegou em fevereiro, use a função LAST\_DAY para encontrar o último dia do mês:

```
select last_day(
    date_add(
        date_add(
            date_add(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day),
            interval 1 month)) dy
from t1

DY
-----
28-FEB-2005
```

A etapa final (que é opcional) é usar a função DAY para retornar 28 ou 29.

## SQL Server

Podemos criar uma nova data na maioria dos RDMSs criando uma string em um formato de data reconhecido e usando CAST para alterar o formato. Podemos, portanto, usar o ano atual recuperando o ano a partir da data atual. No SQL Server, isso é feito aplicando YEAR a GET\_DATE:

```
select YEAR(GETDATE());
```

Isso retornará o ano como um número inteiro. Podemos então criar 29 de fevereiro usando CONCAT e CAST:

```
select cast(concat  
          (year(getdate()), '-02-29'));
```

No entanto, isso não será uma data *real* se o ano atual não for um ano bissexto. Por exemplo, não há data 2019-02-29. Portanto, se tentarmos usar um operador como DAY para encontrar qualquer uma de suas partes, ele retornará NULL. Portanto, use COALESCE e DAY para determinar se há um 29º dia no mês.

## 9.2 Determinando o número de dias em um ano

### Problema

Você deseja contar o número de dias no ano atual.

### Solução

O número de dias no ano atual é a diferença entre o primeiro dia do próximo ano e o primeiro dia do ano atual (em dias). Para cada solução, as etapas são:

1. Encontre o primeiro dia do ano atual.
2. Adicione um ano a essa data (para obter o primeiro dia do próximo ano).
3. Subtraia o ano atual do resultado da Etapa 2.

As soluções diferem apenas nas funções incorporadas que você usa para executar essas etapas.

### DB2

Use a função DAYOFYEAR para ajudar a encontrar o primeiro dia do ano atual e use DAYS para encontrar o número de dias no ano atual:

```

1 select days((curr_year + 1 year)) - days(curr_year)
2   from (
3 select (current_date -
4         dayofyear(current_date) day +
5         1 day) curr_year
6   from t1
7     ) x

```

### Oracle

Use a função TRUNC para encontrar o início do ano atual e use ADD\_MONTHS para encontrar o início do próximo ano:

```

1 select add_months(trunc(sysdate,'y'),12) - trunc(sysdate,'y')
2   from dual

```

### PostgreSQL

Use a função DATE\_TRUNC para encontrar o início do ano atual. Em seguida, use a aritmética de intervalo para determinar o início do próximo ano:

```

1 select cast((curr_year + interval '1 year') as date) - curr_year
2   from (
3 select cast(date_trunc('year',current_date) as date) as curr_year
4   from t1
5     ) x

```

### MySQL

Use ADDDATE para ajudar a encontrar o início do ano atual. Use DATEDIFF e aritmética de intervalo para determinar o número de dias no ano:

```

1 select datediff((curr_year + interval 1 year),curr_year)
2   from (
3 select adddate(current_date,-dayofyear(current_date)+1) curr_year
4   from t1
5     ) x

```

### SQL Server

Use a função DATEADD para encontrar o primeiro dia do ano atual. Use DATEDIFF para retornar o número de dias no ano atual:

```

1 select datediff(d,curr_year,dateadd(yy,1,curr_year))
2   from (
3 select dateadd(d,-datepart(dy,getdate())+1,getdate()) curr_year
4   from t1
5     ) x

```

## DB2

O primeiro passo é encontrar o primeiro dia do ano atual. Use DAYOFYEAR para determinar quantos dias você tem no ano atual. Subtraia esse valor da data atual para obter o último dia do ano passado e adicione 1:

```
select (current_date
        dayofyear(current_date) day +
        1 day) curr_year
  from t1

CURR_YEAR
-----
01-JAN-2005
```

Agora que você tem o primeiro dia do ano atual, basta adicionar um ano a ele; isso lhe dá o primeiro dia do próximo ano. Em seguida, subtraia o início do ano atual do início do ano seguinte.

## Oracle

O primeiro passo é encontrar o primeiro dia do ano atual, o que você pode fazer facilmente invocando a função interna TRUNC e passando Y como o segundo argumento (truncando assim a data para o início do ano):

```
select select trunc(sysdate,'y') curr_year
      from dual

CURR_YEAR
-----
01-JAN-2005
```

Em seguida, adicione um ano para chegar ao primeiro dia do próximo ano. Finalmente, subtraia as duas datas para encontrar o número de dias no ano atual.

## PostgreSQL

Comece encontrando o primeiro dia do ano atual. Para fazer isso, invoque a função DATE\_TRUNC da seguinte forma:

```
select cast(date_trunc('year',current_date) as date) as curr_year
  from t1

CURR_YEAR
-----
01-JAN-2005
```

Você pode facilmente adicionar um ano para calcular o primeiro dia do próximo ano. Então tudo que você precisa fazer é subtrair as duas datas. Certifique-se de subtrair a data anterior da data posterior. O resultado será o número de dias no ano atual.

## MySQL

Seu primeiro passo é encontrar o primeiro dia do ano atual. Use DAYOFYEAR para descobrir quantos dias você tem no ano atual. Subtraia esse valor da data atual e adicione um:

```
select adddate(current_date,-dayofyear(current_date)+1) curr_year  
from t1  
  
CURR_YEAR  
-----  
01-JAN-2005
```

Agora que você tem o primeiro dia do ano atual, sua próxima etapa é adicionar um ano a ele para obter o primeiro dia do próximo ano. Em seguida, subtraia o início do ano atual do início do ano seguinte. O resultado é o número de dias no ano atual.

## SQL Server

Seu primeiro passo é encontrar o primeiro dia do ano atual. Use DATEADD e DATEPART para subtrair da data atual o número de dias no ano em que a data atual é e adicionar 1:

```
select dateadd(d,-datepart(dy,getdate())+1,getdate()) curr_year  
from t1  
  
CURR_YEAR  
-----  
01-JAN-2005
```

Agora que você tem o primeiro dia do ano atual, sua próxima etapa é adicionar um ano a ele para obter o primeiro dia do próximo ano. Em seguida, subtraia o início do ano atual do início do ano seguinte. O resultado é o número de dias no ano atual.

## 9.3 Extraindo unidades de tempo de uma data

### Problema

Você deseja dividir a data atual em seis partes: dia, mês, ano, segundo, minuto e hora. Você deseja que os resultados sejam retornados como números.

O uso da data atual é arbitrário. Sinta-se à vontade para usar esta receita com outras tâmaras. A maioria dos fornecedores já adotou a função padrão ANSI para extrair partes de datas, EXTRACT, embora o SQL Server seja uma exceção. Eles também mantêm seus próprios métodos legados.

## DB2

O DB2 implementa um conjunto de funções integradas que facilitam a extração de partes de uma data. Os nomes das funções HOUR, MINUTE, SECOND, DAY, MONTH, e YEAR correspondem convenientemente às unidades de tempo que você pode retornar: se quiser o dia, use DAY; hora, use HOUR; etc. Por exemplo:

```
1 select    hour( current_timestamp ) hr,
2 minute( current_timestamp ) min,
3 second( current_timestamp ) sec,
4 day( current_timestamp ) dy,
5 month( current_timestamp ) mth,
6 year( current_timestamp ) yr
7 from t1

select
    extract(hour from current_timestamp)
, extract(minute from current_timestamp)
, extract(second from current_timestamp)
, extract(day from current_timestamp)
, extract(month from current_timestamp)
, extract(year from current_timestamp)

      HR   MIN   SEC   DY   MTH   YR
----- 20     28     36    15      6  2005
```

## Oracle

Use as funções TO\_CHAR e TO\_NUMBER para retornar unidades específicas de tempo de uma data:

```
1 select to_number(to_char(sysdate,'hh24')) hour,
2       to_number(to_char(sysdate,'mi')) min,
3       to_number(to_char(sysdate,'ss')) sec,
4       to_number(to_char(sysdate,'dd')) day,
5       to_number(to_char(sysdate,'mm')) mth,
6       to_number(to_char(sysdate,'yyyy')) year
7  from dual

HOUR   MIN   SEC   DAY   MTH   YEAR
----- 20     28     36    15      6  2005
```

## PostgreSQL

Use as funções TO\_CHAR e TO\_NUMBER para retornar unidades específicas de tempo de uma data:

```

1 select to_number(to_char(current_timestamp,'hh24'),'99') as hr,
2       to_number(to_char(current_timestamp,'mi'),'99') as min,
3       to_number(to_char(current_timestamp,'ss'),'99') as sec,
4       to_number(to_char(current_timestamp,'dd'),'99') as day,
5       to_number(to_char(current_timestamp,'mm'),'99') as mth,
6       to_number(to_char(current_timestamp,'yyyy'),'9999') as yr
7  from t1

```

HR	MIN	SEC	DAY	MTH	YR
20	28	36	15	6	2005

## MySQL

Use a função DATE\_FORMAT para retornar unidades de tempo específicas de uma data:

```

1 select date_format(current_timestamp,'%k') hr,
2       date_format(current_timestamp,'%i') min,
3       date_format(current_timestamp,'%s') sec,
4       date_format(current_timestamp,'%d') dy,
5       date_format(current_timestamp,'%m') mon,
6       date_format(current_timestamp,'%Y') yr
7  from t1

```

HR	MIN	SEC	DAY	MTH	YR
- 20	28	36	15	6	2005

## SQL Server

Use a função DATEPART para retornar unidades de tempo específicas de uma data:

```

1 select datepart( hour, getdate()) hr,
2       datepart( minute, getdate()) min,
3       datepart( second, getdate()) sec,
4       datepart( day, getdate()) dy,
5       datepart( month, getdate()) mon,
6       datepart( year, getdate()) yr
7  from t1

```

HR	MIN	SEC	DAY	MTH	YR
20	28	36	15	6	2005

Não há nada sofisticado nessas soluções; apenas aproveite o que você já está pagando. Aproveite o tempo para aprender as funções de data disponíveis para você. Esta receita apenas arranha a superfície das funções apresentadas em cada solução. Você descobrirá que cada uma das funções requer muito mais argumentos e pode retornar mais informações do que esta receita fornece.

## 9.4 Determinando o primeiro e o último dia de um mês

### Problema

Você deseja determinar o primeiro e o último dia do mês atual.

### Solução

As soluções apresentadas aqui são para localizar o primeiro e o último dia do mês atual. Usar o mês atual é arbitrário. Com um pouco de ajuste, você pode fazer as soluções funcionarem para qualquer mês.

#### DB2

Use a função DAY para retornar o número de dias no mês atual que a data atual representa. Subtraia esse valor da data atual e adicione um para obter o primeiro dia do mês. Para obter o último dia do mês, adicione um mês à data atual e, em seguida, subtraia o valor retornado pela função DAY aplicada à data atual:

```
1 select (date(current_date) - day(date(current_date)) day + 1 day) firstday,
2       (date(current_date)+1 month
3        - day(date(current_date)+1 month) day) lastday
4   from t1
```

#### Oracle

Use a função TRUNC para encontrar o primeiro dia do mês e use a função LAST\_DAY para encontrar o último dia do mês:

```
1 select trunc(sysdate,'mm') firstday,
2       last_day(sysdate) lastday
3   from dual
```



Usar TRUNC conforme descrito aqui resultará na perda de qualquer componente de hora do dia, enquanto LAST\_DAY preservará a hora do dia.

## PostgreSQL

Use a função DATE\_TRUNC para truncar a data atual para o primeiro dia do mês atual. Assim que tiver o primeiro dia do mês, adicione um mês e subtraia um dia para encontrar o final do mês atual:

```

1 select firstday,
2       cast(firstday + interval '1 month'
3             - interval '1 day' as date) as lastday
4   from (
5 select cast(date_trunc('month',current_date) as date) as firstday
6   from t1
7      ) x

```

## MySQL

Use as funções DATE\_ADD e DAY para encontrar o número de dias no mês em que a data atual é. Em seguida, subtraia esse valor da data atual e adicione um para encontrar o primeiro dia do mês. Para encontrar o último dia do mês atual, use a função LAST\_DAY:

```

1 select date_add(current_date,
2                   interval -day(current_date)+1 day) firstday,
3         last_day(current_date) lastday
4   from t1

```

## SQL Server

Use as funções DATEADD e DAY para encontrar o número de dias no mês representado pela data atual. Em seguida, subtraia esse valor da data atual e adicione um para encontrar o primeiro dia do mês. Para obter o último dia do mês, adicione um mês à data atual e subtraia desse resultado o valor retornado pela função DAY aplicada à data atual, novamente usando as funções DAY e DATEADD:

```

1 select dateadd(day,-day(getdate())+1,getdate()) firstday,
2       dateadd(day,
3                 -day(dateadd(month,1,getdate())),
4                 dateadd(month,1,getdate())) lastday
5   from t1

```

## Discussão

### DB2

Para encontrar o primeiro dia do mês, basta encontrar o valor numérico do dia atual do mês e subtraí-lo da data atual. Por exemplo, se a data for 14 de março, o valor numérico do dia será 14. Subtrair 14 dias de 14 de março resulta no último dia do mês de fevereiro.

A partir daí, basta adicionar um dia para chegar ao primeiro dia do mês atual. A técnica para obter o último dia do mês é semelhante à do primeiro: subtraia o dia numérico do mês da data atual para obter o último dia do mês anterior. Como queremos o último dia do mês atual (não o último dia do mês anterior), precisamos adicionar um mês à data atual.

### **Oracle**

Para encontrar o primeiro dia do mês atual, use a função TRUNC com “mm” como segundo argumento para “truncar” a data atual até o primeiro dia do mês. Para encontrar o último dia do mês atual, basta usar a função LAST\_DAY.

### **PostgreSQL**

Para encontrar o primeiro dia do mês atual, use a função DATE\_TRUNC com “mês” como segundo argumento para “truncar” a data atual até o primeiro dia do mês. Para encontrar o último dia do mês atual, adicione um mês ao primeiro dia do mês e subtraia um dia.

### **MySQL**

Para encontrar o primeiro dia do mês, use a função DAY. A função DAY retorna o dia do mês para a data passada. Se você subtrair o valor retornado por DAY(CURRENT\_DATE) da data atual, obterá o último dia do mês anterior; adicione um dia para obter o primeiro dia do mês atual. Para encontrar o último dia do mês atual, basta usar a função LAST\_DAY.

### **SQL Server**

Para encontrar o primeiro dia do mês, use a função DAY. A função DAY retorna convenientemente o dia do mês para a data passada. Se você subtrair o valor retornado por DAY(GETDATE()) da data atual, obterá o último dia do mês anterior; adicione um dia para obter o primeiro dia do mês atual. Para encontrar o último dia do mês atual, use a função DATEADD. Adicione um mês à data atual e subtraia dela o valor retornado por DAY(GETDATE()) para obter o último dia do mês atual. Adicione um mês à data atual e subtraia dela o valor retornado por DAY(DATEADD(MONTH,1,GETDATE())) para obter o último dia do mês atual.

## 9.5 Determinando todas as datas para um determinado dia da semana ao longo de um ano

### Problema

Você deseja encontrar todas as datas em um ano que correspondem a um determinado dia da semana. Por exemplo, você pode querer gerar uma lista de sextas-feiras para o ano atual.

### Solução

Independentemente do fornecedor, a chave para a solução é retornar todos os dias do ano atual e manter apenas as datas correspondentes ao dia da semana que lhe interessa. Os exemplos de solução retêm todas as sextas-feiras.

#### DB2

Use a cláusula recursiva WITH para retornar todos os dias do ano atual. Em seguida, use a função DAYNAME para manter apenas as sextas-feiras:

```

1  with x (dy,yr)
2    as (
3 select dy, year(dy) yr
4   from (
5 select (current_date -
6           dayofyear(current_date) days +1 days) as dy
7   from t1
8      ) tmp1
9 union all
10 select dy+1 days, yr
11   from x
12  where year(dy +1 day) = yr
13 )
14 select dy
15   from x
16  where dayname(dy) = 'Friday'
```

#### Oracle

Use a cláusula recursiva CONNECT BY para retornar todos os dias do ano atual. Então use a função TO\_CHAR para manter apenas as sextas-feiras:

```

8 with x
9 as (
10select trunc(sysdate,'y')+level-1 dy
11from t1
12connect by level <=
13add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
14 )
15select *
```

```

16from x
17where to_char( dy, 'dy') = 'fri'
```

## PostgreSQL

Use um CTE recursivo para gerar todos os dias do ano e filtre os dias que não são sextas-feiras. Esta versão utiliza o EXTRACT padrão ANSI, portanto, será executado em uma ampla variedade de RDBMs:

```

1  with recursive cal (dy)
2  as (
3  select current_date
4  -(cast
5    (extract(doy from current_date) as integer)
6    -1)
7  union all
8  select dy+1
9  from cal
10 where extract(year from dy)=extract(year from (dy+1))
11   )
12
13 select dy,extract(dow from dy) from cal
14 where cast(extract(dow from dy) as integer) = 6
```

## MySQL

Use um CTE recursivo para encontrar todos os dias do ano. Em seguida, filtre todos os dias, exceto as sextas-feiras:

```

1      with recursive cal (dy,yr)
2  as
3  (
4  select dy, extract(year from dy) as yr
5  from
6  (select adddate
7        (adddate(current_date, interval - dayofyear(current_date)
8 day), interval 1 day) as dy) as tmp1
9  union all
10  select date_add(dy, interval 1 day), yr
11  from cal
12  where extract(year from date_add(dy, interval 1 day)) = yr
13 )
14  select dy from cal
15  where dayofweek(dy) = 6
```

## SQL Server

Use a cláusula recursiva WITH para retornar todos os dias do ano atual. Em seguida, use a função DAYNAME para manter apenas as sextas-feiras:

```

1  with x (dy,yr)
2  as (
3  select dy, year(dy) yr
```

```

1   from (
2 select getdate()-datepart(dy,getdate())+1 dy
3   from t1
4       ) tmp1
5 union all
6 select dateadd(dd,1,dy), yr
7   from x
8  where year(dateadd(dd,1,dy)) = yr
12 )
13 select x.dy
14   from x
15 where datename(dw,x.dy) = 'Friday'
16 option (maxrecursion 400)

```

## Discussão

### DB2

Para localizar todas as sextas-feiras do ano atual, você deve poder retornar todos os dias do ano atual. O primeiro passo é encontrar o primeiro dia do ano usando a função DAYOFYEAR. Subtraia o valor retornado por DAYOFYEAR(CURRENT\_DATE) da data atual para obter 31 de dezembro do ano anterior e adicione um para obter o primeiro dia do ano atual:

```

select (current_date
        dayofyear(current_date) days +1 days) as dy
      from t1

DY
-----
01-JAN-2005

```

Agora que você tem o primeiro dia do ano, use a cláusula WITH para adicionar repetidamente um dia ao primeiro dia do ano até que você não esteja mais no ano atual. O conjunto de resultados será todos os dias do ano atual (uma parte das linhas retornadas pela visualização recursiva X é mostrada aqui):

```

with x (dy,yr)
  as (
select dy, year(dy) yr
  from (
select (current_date
        dayofyear(current_date) days +1 days) as dy
      from t1
        ) tmp1
union all
select dy+1 days, yr
  from x
 where year(dy +1 day) = yr
)
select dy

```

```
from x
```

```
DY
-----
01-JAN-2020
...
15-FEB-2020
...
22-NOV-2020
...
31-DEC-2020
```

A etapa final é usar a função DAYNAME para manter apenas as linhas que são sextas-feiras.

### Oracle

Para localizar todas as sextas-feiras do ano atual, você deve poder retornar todos os dias do ano atual. Comece usando a função TRUNC para encontrar o primeiro dia do ano:

```
select trunc(sysdate,'y') dy
      from t1

DY
-----
01-JAN-2020
```

Em seguida, use a cláusula CONNECT BY para retornar todos os dias do ano atual (para entender como usar CONNECT BY para gerar linhas, consulte [Receita 10.5](#)).



Como um aparte, esta receita usa a cláusula WITH, mas você também pode usar uma visualização em linha.

Uma parte do conjunto de resultados retornado pela exibição X é mostrada aqui:

```
with x
  as (
select trunc(sysdate,'y')+level-1 dy
  from t1
 connect by level <=
       add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
  )
select *
  from x

DY
-----
01-JAN-2020
...
15-FEB-2020
```

```

...
22-NOV-2020
...
31-DEC-2020

```

A etapa final é usar a função TO\_CHAR para manter apenas as sextas-feiras.

### **PostgreSQL**

Para encontrar as sextas-feiras, primeiro encontre todos os dias. Você precisa encontrar o primeiro dia do ano e, em seguida, usar o CTE recursivo para preencher o restante dos dias. Lembre-se de que o PostgreSQL é um dos pacotes que requer o uso da palavra-chave RECURSIVE para identificar um CTE recursivo.

O passo final é usar a função TO\_CHAR para manter apenas as sextas-feiras.

### **MySQL**

Para localizar todas as sextas-feiras do ano atual, você deve poder retornar todos os dias do ano atual. O primeiro passo é encontrar o primeiro dia do ano. Subtraia o valor retornado por DAYOFYEAR(CURRENT\_DATE) da data atual e adicione um para obter o primeiro dia do ano atual:

```

select adddate(
    adddate(current_date,
        interval -dayofyear(current_date) day),
    interval 1 day ) dy
from t1

DY
-----
01-JAN-2020

```

Depois de obter o primeiro dia do ano, é simples usar um CTE recursivo para adicionar todos os dias do ano:

```

with cal (dy) as
(select current

union all
select dy+1

DY
-----
01-JAN-2020
...
15-FEB-2020
...
22-NOV-2020
...
31-DEC-2020

```

A etapa final é usar a função DAYNAME para manter apenas as sextas-feiras.

### SQL Server

Para localizar todas as sextas-feiras do ano atual, você deve poder retornar todos os dias do ano atual. O primeiro passo é encontrar o primeiro dia do ano usando a função DATEPART. Subtraia o valor retornado por DATEPART(DY,GETDATE()) da data atual e adicione um para obter o primeiro dia do ano atual:

```
select getdate()-datepart(dy,getdate())+1 dy
      from t1

DY
-----
01-JAN-2005
```

Agora que você tem o primeiro dia do ano, use a cláusula WITH e a função DATEADD para adicionar repetidamente um dia ao primeiro dia do ano até que você não esteja mais no ano atual. O conjunto de resultados será todos os dias do ano atual (uma parte das linhas retornadas pela visualização recursiva X é mostrada aqui):

```
with x (dy,yr)
  as (
select dy, year(dy) yr
  from (
select getdate()-datepart(dy,getdate())+1 dy
  from t1
    ) tmp1
union all
select dateadd(dd,1,dy), yr
  from x
 where year(dateadd(dd,1,dy)) = yr
  )
select x.dy
  from x
option (maxrecursion 400)

DY
-----
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

Finalmente, use a função DATENAME para manter apenas as linhas que são sextas-feiras. Para que esta solução funcione, você deve definir MAXRECURSION para pelo menos 366 (o filtro na parte do ano atual, na exibição recursiva X, garante que você nunca gerará mais de 366 linhas).

## 9.6 Determinando a data da primeira e última ocorrências de um dia da semana específico em um mês

### Problema

Você deseja localizar, por exemplo, a primeira e a última segunda-feira do mês atual.

### Solução

A escolha de usar segunda-feira e o mês atual é arbitrária; você pode usar as soluções apresentadas nesta receita para qualquer dia da semana e qualquer mês. Como cada dia da semana tem 7 dias de diferença entre si, assim que você tiver a primeira instância de um dia da semana, poderá adicionar 7 dias para obter o segundo e 14 dias para obter o terceiro. Da mesma forma, se você tiver a última instância de um dia da semana em um mês, poderá subtrair 7 dias para obter o terceiro e subtrair 14 dias para obter o segundo.

#### DB2

Use a cláusula recursiva WITH para gerar cada dia no mês atual e use uma expressão CASE para sinalizar todas as segundas-feiras. A primeira e a última segunda-feira serão a primeira e a última das datas marcadas:

```

1  with x (dy,mth,is_monday)
2    as (
3 select dy,month(dy),
4       case when dayname(dy)='Monday'
5             then 1 else 0
6         end
7   from (
8 select (current_date-day(current_date) day +1 day) dy
9   from t1
10      ) tmp1
11 union all
12 select (dy +1 day), mth,
13       case when dayname(dy +1 day)='Monday'
14             then 1 else 0
15         end
16   from x
17 where month(dy +1 day) = mth
18 )
19 select min(dy) first_monday, max(dy) last_monday
20   from x
21 where is_monday = 1

```

#### Oracle

Use as funções NEXT\_DAY e LAST\_DAY, juntamente com um pouco de aritmética de data inteligente, para encontrar a primeira e a última segunda-feira do mês atual:

```
select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday,
       next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday
  from dual
```

## PostgreSQL

Use a função DATE\_TRUNC para encontrar o primeiro dia do mês. Assim que tiver o primeiro dia do mês, você pode usar aritmética simples envolvendo os valores numéricos dos dias da semana (domingo a sábado é de 1 a 7) para encontrar a primeira e a última segunda-feira do mês atual:

```
1 select first_monday,
2        case to_char(first_monday+28,'mm')
3            when mth then first_monday+28
4            else first_monday+21
5        end as last_monday
6   from (
7 select case sign(cast(to_char(dy,'d') as integer)-2)
8           when 0
9           then dy
10          when -1
11          then dy+abs(cast(to_char(dy,'d') as integer)-2)
12          when 1
13          then (7-(cast(to_char(dy,'d') as integer)-2))+dy
14      end as first_monday,
15      mth
16   from (
17 select cast(date_trunc('month',current_date) as date) as dy,
18           to_char(current_date,'mm') as mth
19  from t1
20      ) x
21      ) y
```

## MySQL

Use a função ADDDATE para encontrar o primeiro dia do mês. Assim que tiver o primeiro dia do mês, você pode usar aritmética simples nos valores numéricos dos dias da semana (domingo a sábado é de 1 a 7) para encontrar a primeira e a última segunda-feira do mês atual:

```
1 select first_monday,
2 case month(adddate(first_monday,28))
3 when mth then adddate(first_monday,28)
4 else adddate(first_monday,21)
5 end last_monday
6 from (
7 select case sign(dayofweek(dy)-2)
8 when 0 then dy
9 when -1 then adddate(dy,abs(dayofweek(dy)-2))
10when 1 then adddate(dy,(7-(dayofweek(dy)-2)))
11end first_monday,
12mth
13from (
```

```
14select adddate(adddate(current_date,-day(current_date)),1) dy,
15month(current_date) mth
16from t1
17) x
18) y
```

## SQL Server

Use a cláusula recursiva WITH para gerar cada dia no mês atual e, em seguida, use uma expressão CASE para sinalizar todas as segundas-feiras. A primeira e a última segunda-feira serão a primeira e a última das datas marcadas:

```
1  with x (dy,mth,is_monday)
2    as (
3 select dy,mth,
4       case when datepart(dw,dy) = 2
5             then 1 else 0
6         end
7   from (
8 select dateadd(day,1,dateadd(day,-day(getdate()),getdate())) dy,
9       month(getdate()) mth
10  from t1
11     ) tmp1
12 union all
13 select dateadd(day,1,dy),
14       mth,
15       case when datepart(dw,dateadd(day,1,dy)) = 2
16             then 1 else 0
17         end
18   from x
19 where month(dateadd(day,1,dy)) = mth
20 )
21 select min(dy) first_monday,
22       max(dy) last_monday
23   from x
24 where is_monday = 1
```

## Discussão

### DB2 e SQLServer

O DB2 e o SQL Server usam funções diferentes para resolver esse problema, mas a técnica é exatamente a mesma. Se você observar ambas as soluções, verá que a única diferença entre as duas é a forma como as datas são adicionadas. Esta discussão cobrirá ambas as soluções, usando o código da solução DB2 para mostrar os resultados das etapas intermediárias.



Se você não tiver acesso à cláusula recursiva WITH na versão do SQL Server ou DB2 que está executando, poderá usar a técnica do PostgreSQL.

A primeira etapa para localizar a primeira e a última segunda-feira do mês atual é retornar o primeiro dia do mês. Visualização sequencial TMP1 na visualização recursiva X localiza o primeiro dia do mês atual localizando primeiro a data atual, especificamente, o dia do mês para a data atual. O dia do mês para a data atual representa quantos dias você tem no mês (por exemplo, 10 de abril é o dia 10 de abril). Se você subtrair o valor deste dia do mês da data atual, terminará no último dia do mês anterior (por exemplo, subtrair 10 de 10 de abril o colocará no último dia de março). Após essa subtração, basta somar um dia para chegar ao primeiro dia do mês atual:

```
select (current_date-day(current_date) day +1 day) dy
      from t1

DY
-----
01-JUN-2005
```

Em seguida, encontre o mês da data atual usando a função MONTH e uma expressão simples CASE para determinar se o primeiro dia do mês é uma segunda-feira:

```
select dy, month(dy) mth,
       case when dayname(dy)='Monday'
             then 1 else 0
        end is_monday
     from (
select (current_date-day(current_date) day +1 day) dy
      from t1
     ) tmp1

DY      MTH  IS_MONDAY
-----
01-JUN-2005  6          0
```

Em seguida, use os recursos recursivos da cláusula WITH para adicionar repetidamente um dia ao primeiro dia do mês até que você não esteja mais no mês atual. Ao longo do caminho, você usará uma expressão CASE para determinar quais dias do mês são segundas-feiras (segundas-feiras serão sinalizadas com 1). Uma parte da saída da visualização recursiva X é mostrada aqui:

```
with x (dy,mth,is_monday)
  as (
select dy,month(dy) mth,
       case when dayname(dy)='Monday'
             then 1 else 0
```

```

        end is_monday
      from (
select (current_date-day(current_date) day +1 day) dy
      from t1
      ) tmp1
    union all
select (dy +1 day), mth,
       case when dayname(dy +1 day)='Monday'
             then 1 else 0
       end
      from x
     where month(dy +1 day) = mth
)
select *
  from x

```

DY	MTH	IS_MONDAY
01-JUN-2005	6	0
2- JUN-2005	6	0
3- JUN-2005	6	0
4- JUN-2005	6	0
5- JUN-2005	6	0
6- JUN-2005	6	1
7- JUN-2005	6	0
8- JUN-2005	6	0
...		

Somente as segundas-feiras terão o valor 1 para IS\_MONDAY, portanto, a etapa final é usar as funções de agregação MIN e MAX nas linhas em que IS\_MONDAY é 1 para localizar a primeira e a última segunda-feira do mês.

### Oracle

A função NEXT\_DAY torna este problema fácil de resolver. Para encontrar a primeira segunda-feira do mês atual, primeiro retorne o último dia do mês anterior por meio de alguma aritmética de data envolvendo a função TRUNC:

```

select trunc(sysdate,'mm')-1 dy
  from dual

DY
-----
31-MAY-2005

```

Em seguida, use a função NEXT\_DAY para encontrar a primeira segunda-feira após o último dia do mês anterior (ou seja, a primeira segunda-feira do mês atual):

```

select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday
  from dual

```

FIRST\_MONDAY

-----

06-JUN-2005

Para encontrar a última segunda-feira do mês atual, comece retornando o primeiro dia do mês atual usando a função TRUNC:

```
select trunc(sysdate,'mm') dy  
      from dual
```

DY

-----

01-JUN-2005

O próximo passo é encontrar a última semana (os últimos sete dias) do mês. Use a função LAST\_DAY para encontrar o último dia do mês e subtraia sete dias:

```
select last_day(trunc(sysdate,'mm'))-7 dy  
      from dual
```

DY

-----

23-JUN-2005

Se não for imediatamente óbvio, volte sete dias a partir do último dia do mês para garantir que terá pelo menos um dia da semana restante no mês. O próximo passo é usar a função NEXT\_DAY para encontrar a próxima (e última) segunda-feira do mês:

```
select next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday  
      from dual
```

LAST\_MONDAY

-----

27-JUN-2005

## PostgreSQL e MySQL

PostgreSQL e MySQL também compartilham a mesma abordagem de solução. A diferença está nas funções que você invoca. Apesar de seus comprimentos, as respectivas consultas são extremamente simples; pouca sobrecarga está envolvida na localização da primeira e da última segunda-feira do mês atual.

O primeiro passo é encontrar o primeiro dia do mês atual. O próximo passo é encontrar a primeira segunda-feira do mês. Como não há função para encontrar a próxima data para um determinado dia da semana, você precisa usar um pouco de aritmética. A expressão CASE começando na linha 7 (de qualquer solução) avalia a diferença entre o valor numérico para o dia da semana do primeiro dia do mês e o valor numérico correspondente na segunda-feira. Dado que a função TO\_CHAR (PostgreSQL), quando chamada com o formato *Doud*, e a função DAYOFWEEK (MySQL) retornará um valor numérico de 1 a 7 representando os dias domingo a sábado, segunda sempre é representado por 2.

primeiro teste avaliado por CASE é o SIGN do valor numérico do primeiro dia do mês (seja ele qual for) menos o valor numérico da segunda-feira (2). Se o resultado for zero, o primeiro dia do mês cairá em uma segunda-feira, ou seja, a primeira segunda-feira do mês. Se o resultado for -1, então o primeiro dia do mês cai em um domingo e, para encontrar a primeira segunda-feira do mês, basta somar a diferença de dias entre 2 e 1 (valores numéricos de segunda-feira e domingo, respectivamente) para o primeiro dia do mês.



Se você está tendo problemas para entender como isso funciona, esqueça os nomes dos dias da semana e apenas faça as contas. Por exemplo, digamos que você esteja começando em uma terça-feira e esteja procurando a próxima sexta-feira. Ao usar TO\_CHAR com o formato *d*, ou DAYOFWEEK, sexta-feira é 6 e terça-feira é 3. Para chegar a 6 de 3, simplesmente pegue a diferença ( $6-3 = 3$ ) e adicione-a ao menor valor ( $(6-3) + 3 = 6$ ). Portanto, independentemente das datas reais, se o valor numérico do dia em que você está começando for menor que o valor numérico do dia que você está procurando, adicionando a diferença entre as duas datas à data em que você está começando, você obterá para a data que você está procurando.

Se o resultado de SIGN for 1, então o primeiro dia do mês cai entre terça e sábado (inclusive). Quando o primeiro dia do mês tiver um valor numérico maior que 2 (segunda-feira), subtraia de 7 a diferença entre o valor numérico do primeiro dia do mês e o valor numérico da segunda-feira (2) e adicione esse valor a o primeiro dia do mês. Terá chegado ao dia da semana que procura, neste caso segunda-feira.



Novamente, se você está tendo problemas para entender como isso funciona, esqueça os nomes dos dias da semana e apenas faça as contas. Por exemplo, suponha que você queira encontrar a próxima terça-feira e comece na sexta-feira. Terça-feira (3) é menor que sexta-feira (6). Para chegar a 3 de 6, subtraia a diferença entre os dois valores de 7 ( $7-(|3-6|) = 4$ ) e adicione o resultado (4) ao dia inicial, sexta-feira. (As barras verticais em  $|3-6|$  geram o valor absoluto dessa diferença.) Aqui, você não está adicionando 4 a 6 (o que lhe dará 10); você está adicionando quatro dias à sexta-feira, o que lhe dará a próxima terça-feira.

A ideia por trás da expressão CASE é criar uma espécie de função “dia seguinte” para PostgreSQL e MySQL. Se você não começar com o primeiro dia do mês, o valor para DY será o valor retornado por CURRENT\_DATE, e o resultado da expressão CASE retornará a data da próxima segunda-feira a partir da data atual (a menos que CURRENT\_DATE seja um segunda-feira, então essa data será retornada).

Agora que você tem a primeira segunda-feira do mês, adicione 21 ou 28 dias para encontrar a última segunda-feira do mês. A expressão CASE nas linhas 2–5 determina se deve adicionar 21 ou 28 dias, verificando se 28 dias levam você ao próximo mês. A expressão CASE faz isso através do seguinte processo:

1. Acrescenta 28 ao valor de FIRST\_MONDAY.
2. Usando TO\_CHAR (PostgreSQL) ou MONTH, a expressão CASE extrai o nome do mês atual do resultado de FIRST\_MONDAY + 28.
3. O resultado da etapa dois é comparado ao valor MTH da exibição em linha. O valor MTH é o nome do mês atual derivado de CURRENT\_DATE. Se os valores de 2 meses forem correspondentes, o mês é grande o suficiente para você precisar adicionar 28 dias, e a expressão CASE retornará FIRST\_MONDAY + 28. Se os valores de dois meses não corresponderem, você não terá espaço para adicionar 28 dias, e a expressão CASE retorna FIRST\_MONDAY + 21 dias. É conveniente que nossos meses sejam tais que 28 e 21 sejam os únicos dois valores possíveis com os quais você precisa se preocupar em somar.



Você pode estender a solução adicionando 7 e 14 dias para encontrar a segunda e a terceira segunda-feiras do mês, respectivamente.

## 9.7 Criando um calendário

### Problema

Você deseja criar um calendário para o mês atual. O calendário deve ser formatado como um calendário que você pode ter em sua mesa: sete colunas e (geralmente) cinco linhas abaixo.

### Solução

Cada solução parecerá um pouco diferente, mas todas resolvem o problema da mesma maneira: retorne todos os dias para o mês atual e gire no dia da semana para cada semana do mês para criar um calendário.

Existem diferentes formatos disponíveis para calendários. Por exemplo, o comando Unix CAL formata os dias de domingo a sábado. Os exemplos nesta receita são baseados em semanas ISO, portanto, o formato de segunda a sexta-feira é o mais conveniente de gerar.

Depois de se familiarizar com as soluções, você verá que reformatar como quiser é simplesmente uma questão de modificar os valores atribuídos pela semana ISO antes da rotação.



À medida que você começar a usar diferentes tipos de formatação com SQL para criar uma saída legível, notará que suas consultas estão ficando mais longas. Não deixe que essas longas consultas o intimidem; as consultas apresentadas para esta receita são extremamente simples, uma vez divididas e executadas peça por peça.

## DB2

Use a cláusula recursiva WITH para retornar todos os dias no mês atual. Em seguida, gire no dia da semana usando CASE e MAX:

```

1  with x(dy,dm,mth,dw,wk)
2  as (
3 select (current_date -day(current_date) day +1 day) dy,
4       day((current_date -day(current_date) day +1 day)) dm,
5       month(current_date) mth,
6       dayofweek(current_date -day(current_date) day +1 day) dw,
7       week_iso(current_date -day(current_date) day +1 day) wk
8   from t1
9  union all
10 select dy+1 day, day(dy+1 day), mth,
11        dayofweek(dy+1 day), week_iso(dy+1 day)
12   from x
13  where month(dy+1 day) = mth
14 )
15 select max(case dw when 2 then dm end) as Mo,
16        max(case dw when 3 then dm end) as Tu,
17        max(case dw when 4 then dm end) as We,
18        max(case dw when 5 then dm end) as Th,
19        max(case dw when 6 then dm end) as Fr,
20        max(case dw when 7 then dm end) as Sa,
21        max(case dw when 1 then dm end) as Su
22   from x
23  group by wk
24  order by wk

```

## Oracle

Use a cláusula recursiva CONNECT BY para retornar todos os dias do mês atual. Em seguida, gire no dia da semana usando CASE e MAX:

```

1  with x
2  as (
3 select *
4  from (
5 select to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
6       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
7       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,

```

```
1      to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
2      to_char(sysdate,'mm') mth
3  from dual
4 connect by level <= 31
12 )
13 where curr_mth = mth
14 )
15 select max(case dw when 2 then dm end) Mo,
16       max(case dw when 3 then dm end) Tu,
17       max(case dw when 4 then dm end) We,
18       max(case dw when 5 then dm end) Th,
19       max(case dw when 6 then dm end) Fr,
20       max(case dw when 7 then dm end) Sa,
21       max(case dw when 1 then dm end) Su
22  from x
23 group by wk
24 order by wk
```

## PostgreSQL

Use a função GENERATE\_SERIES para retornar todos os dias do mês atual. Em seguida, gire no dia da semana usando MAX e CASE:

```
1 select max(case dw when 2 then dm end) as Mo,
2 max(case dw when 3 then dm end) as Tu,
3 max(case dw when 4 then dm end) as We,
4 max(case dw when 5 then dm end) as Th,
5 max(case dw when 6 then dm end) as Fr,
6 max(case dw when 7 then dm end) as Sa,
7 max(case dw when 1 then dm end) as Su
8 from (
9 select *
10from (
11select cast(date_trunc('month',current_date) as date)+x.id,
12to_char(
13cast(
14date_trunc('month',current_date)
15as date)+x.id,'iw') as wk,
16to_char(
17cast(
18date_trunc('month',current_date)
19as date)+x.id,'dd') as dm,
20cast(
21to_char(
22cast(
23date_trunc('month',current_date)
24as date)+x.id,'d') as integer) as dw,
25to_char(
26cast(
27date_trunc('month',current_date)
28as date)+x.id,'mm') as curr_mth,
29to_char(current_date,'mm') as mth
```

```

30from generate_series (0,31) x(id)
31) x
32where mth = curr_mth
33) y
34group by wk
35order by wk

```

## MySQL

Use um CTE recursivo para retornar todos os dias no mês atual. Em seguida, gire no dia da semana usando MAX e CASE:

```

with recursive x(dy,dm,mth,dw,wk)
      as (
    select dy,
           day(dy) dm,
           datepart(m,dy) mth,
           datepart(dw,dy) dw,
           case when datepart(dw,dy) = 1
                 then datepart(ww,dy)-1
                 else datepart(ww,dy)
           end wk
      from (
    select date_add(day,-day(getdate())+1,getdate()) dy
      from t1
      ) x
  union all
    select dateadd(d,1,dy), day(date_add(d,1,dy)), mth,
           datepart(dw,dateadd(d,1,dy)),
           case when datepart(dw,date_add(d,1,dy)) = 1
                 then datepart(wk,date_add(d,1,dy))-1
                 else datepart(wk,date_add(d,1,dy))
           end
      from x
     where datepart(m,date_add(d,1,dy)) = mth
  )
  select max(case dw when 2 then dm end) as Mo,
         max(case dw when 3 then dm end) as Tu,
         max(case dw when 4 then dm end) as We,
         max(case dw when 5 then dm end) as Th,
         max(case dw when 6 then dm end) as Fr,
         max(case dw when 7 then dm end) as Sa,
         max(case dw when 1 then dm end) as Su
    from x
   group by wk
  order by wk;

```

Use a cláusula recursiva WITH para retornar todos os dias no mês atual. Em seguida, gire no dia da semana usando CASE e MAX:

```
1  with x(dy,dm,mth,dw,wk)
2    as (
3 select dy,
4       day(dy) dm,
5       datepart(m,dy) mth,
6       datepart(dw,dy) dw,
7       case when datepart(dw,dy) = 1
8             then datepart(ww,dy)-1
9             else datepart(ww,dy)
10            end wk
11   from (
12 select dateadd(day,-day(getdate())+1,getdate()) dy
13   from t1
14   ) x
15 union all
16 select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
17        datepart(dw,dateadd(d,1,dy)),
18        case when datepart(dw,dateadd(d,1,dy)) = 1
19              then datepart(wk,dateadd(d,1,dy)) -1
20              else datepart(wk,dateadd(d,1,dy))
21            end
22   from x
23 where datepart(m,dateadd(d,1,dy)) = mth
24 )
25 select max(case dw when 2 then dm end) as Mo,
26        max(case dw when 3 then dm end) as Tu,
27        max(case dw when 4 then dm end) as We,
28        max(case dw when 5 then dm end) as Th,
29        max(case dw when 6 then dm end) as Fr,
30        max(case dw when 7 then dm end) as Sa,
31        max(case dw when 1 then dm end) as Su
32   from x
33 group by wk
34 order by wk
```

## Discussão

### DB2

A primeira etapa é retornar todos os dias do mês para o qual você deseja criar um calendário. Faça isso usando a cláusula recursiva WITH. Junto com cada dia do mês (DM), você precisará retornar diferentes partes de cada data: o dia da semana (DW), o mês atual com o qual você está trabalhando (MTH) e a semana ISO para cada dia de o mês (WK).

Os resultados da visão recursiva X antes da ocorrência da recursão (a parte superior de UNION ALL) são mostrados aqui:

```
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1
```

DY	DM	MTH	DW	WK
01-JUN-2005	01	06	4	22

A próxima etapa é aumentar repetidamente o valor de DM (mover pelos dias do mês) até que você não esteja mais no mês atual. Conforme você percorre cada dia do mês, também retornará o dia da semana correspondente a cada dia e em qual semana ISO o dia atual do mês se enquadra. Os resultados parciais são mostrados aqui:

```
with x(dy, dm, mth, dw, wk)
  as (
    select (current_date -day(current_date) day +1 day) dy,
           day((current_date -day(current_date) day +1 day)) dm,
           month(current_date) mth,
           dayofweek(current_date -day(current_date) day +1 day) dw,
           week_iso(current_date -day(current_date) day +1 day) wk
      from t1
   union all
    select dy+1 day, day(dy+1 day), mth,
           dayofweek(dy+1 day), week_iso(dy+1 day)
      from x
     where month(dy+1 day) = mth
  )
 select *
  from x
```

DY	DM	MTH	DW	WK
01-JUN-2005	01	06	4	22
02-JUN-2005	02	06	5	22
...				
21-JUN-2005	21	06	3	25
22-JUN-2005	22	06	4	25
...				
30-JUN-2005	30	06	5	26

O que você está retornando neste ponto é: cada dia do mês atual, o dia numérico de dois dígitos do mês, o mês numérico de dois dígitos, o dia da semana de um dígito (1–7 para Dom–Sáb), e a semana ISO de dois dígitos em que cada dia se enquadra.

Com todas essas informações disponíveis, você pode usar uma expressão CASE para determinar em qual dia da semana cada valor de DM (cada dia do mês) se enquadra. Uma parte dos resultados é mostrada aqui:

```
with x(dy, dm, mth, dw, wk)
  as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1
union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
  from x
 where month(dy+1 day) = mth
)
select wk,
       case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from x
```

WK	MO	TU	WE	TH	FR	SA	SU
22					01		
22					02		
22					03		
22					04		
22					05		
23	06						
23	07						
23	08						
23	09						
23		10					
23		11					
23		12					

Como você pode ver na saída parcial, todos os dias de cada semana são retornados como uma linha. O que você deseja fazer agora é agrupar os dias por semana e, em seguida, recolher todos os dias de cada semana em uma única linha. Use a função de agregação MAX e agrupe por WK (a semana ISO) para retornar todos os dias de uma semana como uma linha. Para formatar corretamente o calendário e garantir que os dias estejam na ordem certa, ordene os resultados por WK. A saída final é mostrada aqui:

```
with x(dy, dm, mth, dw, wk)
  as (
select (current_date -day(current_date) day +1 day) dy,
```

```

day((current_date -day(current_date) day +1 day)) dm,
month(current_date) mth,
dayofweek(current_date -day(current_date) day +1 day) dw,
week_iso(current_date -day(current_date) day +1 day) wk
from t1
union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
  from x
 where month(dy+1 day) = mth
)
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from x
 group by wk
order by wk

```

MO	TU	WE	TH	FR	SA	SU
01	02	03	04	05		
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

## Oracle

Comece usando a cláusula recursiva CONNECT BY para gerar uma linha para cada dia do mês para o qual deseja gerar um calendário. Se você não estiver executando pelo menos o banco de dados Oracle9i, você não pode usar CONNECT BY desta forma. Em vez disso, você pode usar uma tabela dinâmica, como T500 na solução MySQL.

Junto com cada dia do mês, você precisará retornar informações diferentes para cada dia: o dia do mês (DM), o dia da semana (DW), o mês atual com o qual você está trabalhando (MTH), e a semana ISO para cada dia do mês (WK). Os resultados da visão COM X para o primeiro dia do mês atual são mostrados aqui:

```

select trunc(sysdate,'mm') dy,
       to_char(trunc(sysdate,'mm'),'dd') dm,
       to_char(sysdate,'mm') mth,
       to_number(to_char(trunc(sysdate,'mm'),'d')) dw,
       to_char(trunc(sysdate,'mm'),'iw') wk
  from dual

```

DY	DM	MT	DW	WK
01-JUN-2020	01	06	4	22

A próxima etapa é aumentar repetidamente o valor de DM (mover pelos dias do mês) até que você não esteja mais no mês atual. À medida que avança em cada dia do mês, você também retornará o dia da semana para cada dia e a semana ISO na qual o dia atual cai. Os resultados parciais são mostrados aqui (a data completa para cada dia é adicionada para facilitar a leitura):

```
with x
  as (
select *
  from (
select trunc(sysdate, 'mm')+level-1 dy,
      to_char(trunc(sysdate, 'mm')+level-1, 'iw') wk,
      to_char(trunc(sysdate, 'mm')+level-1, 'dd') dm,
      to_number(to_char(trunc(sysdate, 'mm')+level-1, 'd')) dw,
      to_char(trunc(sysdate, 'mm')+level-1, 'mm') curr_mth,
      to_char(sysdate, 'mm') mth
  from dual
connect by level <= 31
)
where curr_mth = mth
)
select *
  from x
```

DY	WK	DM	DW	CU	MT
01-JUN-2020	22	01	4	06	06
02-JUN-2020	22	02	5	06	06
...					
21-JUN-2020	25	21	3	06	06
22-JUN-2020	25	22	4	06	06
...					
30-JUN-2020	26	30	5	06	06

O que você está retornando neste ponto é uma linha para cada dia do mês atual. Nessa linha, você tem: o dia numérico de dois dígitos do mês, o mês numérico de dois dígitos, o dia da semana de um dígito (1–7 para Dom–Sáb) e o número da semana ISO de dois dígitos. Com todas essas informações disponíveis, você pode usar uma expressão CASE para determinar em qual dia da semana cada valor de DM (cada dia do mês) se enquadra. Uma parte dos resultados é mostrada aqui:

```
with x
  as (
select *
  from (
select trunc(sysdate, 'mm')+level-1 dy,
      to_char(trunc(sysdate, 'mm')+level-1, 'iw') wk,
```

```

        to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
        to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
        to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
        to_char(sysdate,'mm') mth
    from dual
connect by level <= 31
)
where curr_mth = mth
)
select wk,
       case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
from x

```

	WK	MO	TU	WE	TH	FR	SA	SU
22		01						
22			02					
22				03				
22					04			
22						05		
23	06							
23		07						
23			08					
23				09				
23					10			
23						11		
23							12	

Como você pode ver na saída parcial, todos os dias de cada semana são retornados como uma linha, mas o número do dia está em uma das sete colunas correspondentes ao dia da semana. Sua tarefa agora é consolidar os dias em uma linha para cada semana. Use a função de agregação MAX e agrupe por WK (a semana ISO) para retornar todos os dias de uma semana como uma linha. Para garantir que os dias estejam na ordem correta, ordene os resultados por WK. A saída final é mostrada aqui:

```

with x
as (
select *
  from (
select to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
connect by level <= 31
)
order by wk
)
group by wk
select wk,
       max(dm) as Mo,
       max(dw) as Tu,
       max(dm) as We,
       max(dw) as Th,
       max(dm) as Fr,
       max(dw) as Sa,
       max(dm) as Su
from x

```

```

        )
      where curr_mth = mth
    )
  select max(case dw when 2 then dm end) Mo,
         max(case dw when 3 then dm end) Tu,
         max(case dw when 4 then dm end) We,
         max(case dw when 5 then dm end) Th,
         max(case dw when 6 then dm end) Fr,
         max(case dw when 7 then dm end) Sa,
         max(case dw when 1 then dm end) Su
  from x
 group by wk
 order by wk

```

MO	TU	WE	TH	FR	SA	SU
--	--	--	--	--	--	--
01	02	03	04	05		
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

## MySQL, PostgreSQL e SQL Server

Essas soluções são as mesmas, exceto pelas diferenças nas funções específicas usadas para chamar datas. Usamos arbitrariamente a solução SQL Serve para a explicação. Comece retornando uma linha para cada dia do mês. Você pode fazer isso usando a cláusula recursiva WITH. Para cada linha retornada, você precisará dos seguintes itens: o dia do mês (DM), o dia da semana (DW), o mês atual com o qual está trabalhando (MTH) e a semana ISO para cada dia do mês (WK). Os resultados da visão recursiva X antes da ocorrência da recursão (a parte superior de UNION ALL) são mostrados aqui:

```

select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
             then datepart(ww,dy)-1
             else datepart(ww,dy)
       end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
  from t1
 ) x

```

DY	DM	MTH	DW	WK
01-JUN-2005	1	6		4 23

Seu próximo passo é aumentar repetidamente o valor de DM (mover pelos dias do mês) até que você não esteja mais no mês atual. Conforme você percorre cada dia do mês, também retornará o dia da semana e o número ISO da semana. Os resultados parciais são mostrados aqui:

```

with x(dy, dm, mth, dw, wk)
  as (
select dy,
      day(dy) dm,
      datepart(m,dy) mth,
      datepart(dw,dy) dw,
      case when datepart(dw,dy) = 1
            then datepart(ww,dy)-1
            else datepart(ww,dy)
      end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
  from t1
    ) x
union all
select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
       datepart(dw,dateadd(d,1,dy)),
       case when datepart(dw,dateadd(d,1,dy)) = 1
             then datepart(wk,dateadd(d,1,dy))-1
             else datepart(wk,dateadd(d,1,dy))
       end
  from x
  where datepart(m,dateadd(d,1,dy)) = mth
)
select *
  from x

DY      DM MTH      DW WK
----- - - - - -
01-JUN-2005 01 06      4 23
02-JUN-2005 02 06      5 23
...
21-JUN-2005 21 06      3 26
22-JUN-2005 22 06      4 26
...
30-JUN-2005 30 06      5 27

```

Para cada dia do mês atual, você agora tem: o dia numérico de dois dígitos do mês, o mês numérico de dois dígitos, o dia da semana de um dígito (1–7 para Dom–Sáb) e os dois números da semana ISO de dígitos.

Agora, use uma expressão CASE para determinar em qual dia da semana cada valor de DM (cada dia do mês) se enquadra. Uma parte dos resultados é mostrada aqui:

```

with x(dy, dm, mth, dw, wk)
  as (
select dy,

```

```

    day(dy) dm,
    datepart(m,dy) mth,
    datepart(dw,dy) dw,
    case when datepart(dw,dy) = 1
        then datepart(ww,dy)-1
        else datepart(ww,dy)
    end wk
from (
select dateadd(day,-day(getdate())+1,getdate()) dy
from t1
) x
union all
select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
datepart(dw,dateadd(d,1,dy)),
case when datepart(dw,dateadd(d,1,dy)) = 1
    then datepart(wk,dateadd(d,1,dy))-1
    else datepart(wk,dateadd(d,1,dy))
end
from x
where datepart(m,dateadd(d,1,dy)) = mth
)
select case dw when 2 then dm end as Mo,
case dw when 3 then dm end as Tu,
case dw when 4 then dm end as We,
case dw when 5 then dm end as Th,
case dw when 6 then dm end as Fr,
case dw when 7 then dm end as Sa,
case dw when 1 then dm end as Su
from x

```

WK	MO	TU	WE	TH	FR	SA	SU
22	01						
22		02					
22			03				
22				04			
22					05		
23	06						
23		07					
23			08				
23				09			
23					10		
23						11	
23							12

Todos os dias de cada semana são retornados como uma linha separada. Em cada linha, a coluna que contém o número do dia corresponde ao dia da semana. Agora você precisa consolidar os dias de cada semana em uma linha. Faça isso agrupando as linhas por WK (a semana ISO) e aplicando a função MAX às diferentes colunas. Os resultados estarão em formato de calendário, conforme mostrado aqui:

```

withx(dy,dm,mth,dw,wk)
    as (
select dy,
      day(dy) dm,
      datepart(m,dy) mth,
      datepart(dw,dy) dw,
      case when datepart(dw,dy) = 1
            then datepart(ww,dy)-1
            else datepart(ww,dy)
      end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
  from t1
    ) x
union all
select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
      datepart(dw,dateadd(d,1,dy)),
      case when datepart(dw,dateadd(d,1,dy)) = 1
            then datepart(wk,dateadd(d,1,dy))-1
            else datepart(wk,dateadd(d,1,dy))
      end
  from x
 where datepart(m,dateadd(d,1,dy)) = mth
)
select max(case dw when 2 then dm end) as Mo,
      max(case dw when 3 then dm end) as Tu,
      max(case dw when 4 then dm end) as We,
      max(case dw when 5 then dm end) as Th,
      max(case dw when 6 then dm end) as Fr,
      max(case dw when 7 then dm end) as Sa,
      max(case dw when 1 then dm end) as Su
  from x
group by wk
order by wk

MO TU WE TH FR SA SU
--- - - - - -
- 01 02 03 04
05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

```

## 9.8 Listando as datas de início e término do trimestre para o ano

### Problema

Você deseja retornar as datas de início e término de cada um dos quatro trimestres de um determinado ano.

Há quatro trimestres em um ano, então você sabe que precisará gerar quatro linhas. Depois de gerar o número desejado de linhas, basta usar as funções de data fornecidas pelo seu RDBMS para retornar ao trimestre em que as datas de início e fim se enquadram. Seu objetivo é produzir o seguinte conjunto de resultados (mais uma vez, a escolha de usar o ano atual é arbitrária):

QTR	Q_START	Q_END
1	01-JAN-2020	31-MAR-2020
2	01-APR-2020	30-JUN-2020
3	01-JUL-2020	30-SEP-2020
4	01-OCT-2020	31-DEC-2020

## DB2

Use a tabela EMP e a função de janela ROW\_NUMBER OVER para gerar quatro linhas. Como alternativa, você pode usar a cláusula WITH para gerar linhas (como muitas das receitas fazem) ou pode consultar qualquer tabela com pelo menos quatro linhas. A solução a seguir usa a abordagem ROW\_NUMBER OVER:

```
1 select quarter(dy-1 day) QTR,
2      dy-3 month Q_start,
3      dy-1 day Q_end
4  from (
5 select (current_date -
6          (dayofyear(current_date)-1) day
7          + (rn*3) month) dy
8  from (
9 select row_number()over() rn
10 from emp
11 fetch first 4 rows only
12      ) x
13      ) y
```

## Oracle

Use a função ADD\_MONTHS para encontrar as datas inicial e final de cada trimestre. Use ROWNUM para representar o trimestre ao qual pertencem as datas inicial e final. A solução a seguir usa a tabela EMP para gerar quatro linhas:

```
1 select rownum qtr,
2 add_months(trunc(sysdate,'y'),(rownum-1)*3) q_start,
3 add_months(trunc(sysdate,'y'),rownum*3)-1 q_end
4 from emp
5 where rownum <= 4
```

## PostgreSQL

Encontre o primeiro dia do ano com base na data atual e use um CTE recursivo para preencher a primeira data dos três trimestres restantes antes de encontrar o último dia de cada trimestre:

```
with recursive x (dy,cnt)
  as (
select
    current_date -cast(extract(day from current_date)as integer) +1 dy
    ,id
  from t1
  union all
select cast(dy + interval '3 months' as date) , cnt+1
    from x
   where cnt+1 <= 4
)
select  cast(dy - interval '3 months' as date) as Q_start
      ,dy-1 as Q_end
      from x
```

## MySQL

Encontre o primeiro dia do ano a partir do dia atual e use um CTE para criar quatro linhas, uma para cada trimestre. Use ADDDATE para encontrar o último dia de cada trimestre (três meses após o último dia anterior ou o primeiro dia do trimestre menos um):

```
1      with recursive x (dy,cnt)
2      as (
3          select
4              adddate(current_date,(-dayofyear(current_date))+1) dy
5              ,id
6              from t1
7              union all
8                  select adddate(dy, interval 3 month ), cnt+1
9                  from x
10                 where cnt+1 <= 4
11
12
13      select quarter(adddate(dy,-1)) QTR
14      , date_add(dy, interval -3 month) Q_start
15      , adddate(dy,-1) Q_end
16      from x
17      order by 1;
```

## SQL Server

Use a cláusula recursiva WITH para gerar quatro linhas. Use a função DATEADD para encontrar as datas inicial e final. Use a função DATEPART para determinar a qual trimestre as datas de início e fim pertencem:

```

1  with x (dy,cnt)
2    as (
3 select dateadd(d,-(datepart(dy,getdate())-
1),getdate()), 4   1
5   from t1
6  union all
7 select dateadd(m,3,dy), cnt+1
8   from x
9  where cnt+1 <=
4 10 )
11 select datepart(q,dateadd(d,-1,dy)) QTR,
12        dateadd(m,-3,dy) Q_start,
13        dateadd(d,-1,dy) Q_end
14   from x
15 order by 1

```

## Discussão

### DB2

A primeira etapa é gerar quatro linhas (com valores de um a quatro) para cada trimestre do ano. A visualização embutida X usa a função de janela ROW\_NUMBER OVER e a cláusula FETCH FIRST para retornar apenas quatro linhas do EMP. Os resultados são mostrados aqui:

```

select row_number()over() rn
  from emp
fetch first 4 rows only

```

```

RN
--
1
2
3
4

```

O próximo passo é encontrar o primeiro dia do ano e, em seguida, adicionar  $n$  meses para isso, onde  $n$  é três vezes RN (você está adicionando 3, 6, 9 e 12 meses ao primeiro dia do ano). Os resultados são mostrados aqui:

```

select (current_date
        (dayofyear(current_date)-1) day
        + (rn*3) month) dy
  from (
select row_number()over() rn
  from emp
fetch first 4 rows only
      ) x

```

```

DY
-----
- 01-APR-
2005
01-JUL-2005

```

01-OCT-2005  
01-JAN-2005

Neste ponto, os valores para DY são um dia após a data final de cada trimestre. A próxima etapa é obter as datas de início e término de cada trimestre. Subtraia um dia de DY para obter o final de cada trimestre e subtraia três meses de DY para obter o início de cada trimestre. Use a função QUARTER em DY-1 (a data final de cada trimestre) para determinar a qual trimestre as datas de início e fim pertencem.

### Oracle

A combinação de ROWNUM, TRUNC e ADD\_MONTHS facilita essa solução. Para encontrar o início de cada trimestre, basta adicionar  $n$  meses até o primeiro dia do ano, onde  $n$  é  $(\text{ROWNUM}-1)*3$  (dando a você 0, 3, 6, 9). Para encontrar o final de cada trimestre, adicione  $n$  meses até o primeiro dia do ano, onde  $n$  é  $\text{ROWNUM}*3$  e subtraia um dia. Além disso, ao trabalhar com trimestres, você também pode achar útil usar TO\_CHAR e/ou TRUNC com a opção de formatação Q.

### PostgreSQL, MySQL e SQL Server

Como algumas das receitas anteriores, esta receita usa a mesma estrutura em três implementações RDMS, mas uma sintaxe diferente para as operações de data. O primeiro passo é encontrar o primeiro dia do ano e, em seguida, adicionar recursivamente  $n$  meses, onde  $n$  é três vezes a iteração atual (existem quatro iterações, portanto, você está adicionando  $3*1$  meses,  $3*2$  meses, etc.), usando a função DATEADD ou seu equivalente. Os resultados são mostrados aqui:

```
with x (dy,cnt)
  as (
select dateadd(d,-(datepart(dy,getdate())-1),getdate()),
       1
  from t1
 union all
select dateadd(m,3,dy), cnt+1
  from x
 where cnt+1 <= 4
)
select dy
  from x

DY
-----
01-APR-2020
01-JUL-2020
01-OCT-2020
01-JAN-2020
```

Os valores para DY são um dia após o final de cada trimestre. Para obter o final de cada trimestre, basta subtrair um dia de DY usando a função DATEADD.

Para encontrar o início de cada trimestre, use a função DATEADD para subtrair três meses de DY. Use a função DATEPART na data final de cada trimestre para determinar a qual trimestre as datas inicial e final pertencem ou seu equivalente. Se você estiver usando o PostgreSQL, observe que precisa de CAST para garantir que os tipos de dados se alinhem depois de adicionar os três meses à data de início, ou os tipos de dados serão diferentes e UNION ALL no CTE recursivo falhará.

## 9.9 Determinando as datas de início e fim do trimestre para um determinado trimestre

### Problema

Ao receber um ano e um trimestre no formato YYYYQ (ano de quatro dígitos, trimestre de um dígito), você deseja retornar as datas de início e término do trimestre.

### Solução

A chave para esta solução é encontrar o trimestre usando a função de módulo no valor YYYYQ. (Como alternativa ao módulo, como o formato do ano é de quatro dígitos, você pode simplesmente substringir o último dígito para obter o trimestre.) Assim que tiver o trimestre, basta multiplicar por três para obter o mês final do trimestre. Nas soluções a seguir, a exibição em linha X retornará todas as combinações de quatro anos e trimestres. O conjunto de resultados para a visualização em linha X é o seguinte:

```
select 20051 as yrq from t1 union all
select 20052 as yrq from t1 union all
select 20053 as yrq from t1 union all
select 20054 as yrq from t1
    YRQ
-----
20051
20052
20053
20054
```

### DB2

Use a função SUBSTR para retornar o ano da visualização inline X. Use a função MOD para determinar qual trimestre você está procurando:

```
1 select (q_end-2 month) q_start,
2 (q_end+1 month)-1 day q_end
3 from (
4 select date(substr(cast(yrq as char(4)),1,4) || '-' ||
5 rtrim(cast(mod(yrq,10)*3 as char(2))) || '-1') q_end
6 from (
7 select 20051 yrq from t1 union all
```

```

8 select 20052 yrq from t1 union all
9 select 20053 yrq from t1 union all
10select 20054 yrq from t1
11) x
12) y

```

## Oracle

Use a função SUBSTR para retornar o ano da visualização inline X. Use a função MOD para determinar qual trimestre você está procurando:

```

1 select add_months(q_end,-2) q_start,
2       last_day(q_end) q_end
3   from (
4 select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
5   from (
6 select 20051 yrq from dual union all
7 select 20052 yrq from dual union all
8 select 20053 yrq from dual union all
9 select 20054 yrq from dual
10      ) x
11      ) y

```

## PostgreSQL

Use a função SUBSTR para retornar o ano da visualização inline X. Use a função MOD para determinar qual trimestre você está procurando:

```

1 select date(q_end-(2*interval '1 month')) as q_start,
2       date(q_end+interval '1 month'-interval '1 day') as q_end
3   from (
4 select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') as q_end
5   from (
6 select 20051 as yrq from t1 union all
7 select 20052 as yrq from t1 union all
8 select 20053 as yrq from t1 union all
9 select 20054 as yrq from t1
10      ) x
11      ) y

```

## MySQL

Use a função SUBSTR para retornar o ano da visualização inline X. Use a função MOD para determinar qual trimestre você está procurando:

```

1 select date_add(
2 adddate(q_end,-day(q_end)+1),
3 interval -2 month) q_start,
4 q_end
5 from (
6 select last_day(
7 str_to_date(

```

```

1      concat(
9      substr(yrq,1,4),mod(yrq,10)*3),'%Y%m')) q_end
10     from (
11 select 20051 as yrq from t1 union all
12 select 20052 as yrq from t1 union all
13 select 20053 as yrq from t1 union all
14 select 20054 as yrq from t1
15      ) x
16      ) y

```

## SQL Server

Use a função SUBSTRING para retornar o ano da visualização inline X. Use a função de módulo (%) para determinar qual trimestre você está procurando:

```

1 select dateadd(m,-2,q_end) q_start,
2       dateadd(d,-1,dateadd(m,1,q_end)) q_end
3   from (
4 select cast(substring(cast(yrq as varchar),1,4)+'-' +
5           cast(yrq%10*3 as varchar)+'-1' as datetime) q_end
6   from (
7 select 20051 as yrq from t1 union all
8 select 20052 as yrq from t1 union all
9 select 20052 as yrq from t1 union all
10 select 20054 as yrq from t1
11      ) x
12      ) y

```

## Discussão

### DB2

O primeiro passo é encontrar o ano e o trimestre com os quais você está trabalhando. Substring fora o ano da exibição em linha X (X.YRQ) usando a função SUBSTR. Para obter o trimestre, use o módulo 10 em YRQ. Assim que tiver o trimestre, multiplique por três para obter o mês final do trimestre. Os resultados são mostrados aqui:

```

select substr(cast(yrq as char(4)),1,4) yr,
       mod(yrq,10)*3 mth
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
      ) x

YR      MTH
----- -----
2005      3
2005      6

```

2005	9
2005	12

Neste ponto, você tem o ano e o mês final de cada trimestre. Use esses valores para construir uma data, especificamente, o primeiro dia do último mês de cada trimestre. Use o operador de concatenação || para colar o ano e o mês e, em seguida, use a função DATE para converter em uma data:

```
select date(substr(cast(yrq as char(4)),1,4) ||'-'||
            rtrim(cast(mod(yrq,10)*3 as char(2))) ||'-1') q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x

Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Os valores para Q-END são o primeiro dia do último mês de cada trimestre. Para chegar ao último dia do mês, adicione um mês a Q-END e depois subtraia um dia. Para encontrar a data de início de cada trimestre, subtraia dois meses de Q-END.

## Oracle

O primeiro passo é encontrar o ano e o trimestre com os quais você está trabalhando. Substring fora o ano da exibição em linha X (X.YRQ) usando a função SUBSTR. Para obter o trimestre, use o módulo 10 em YRQ. Assim que tiver o trimestre, multiplique por três para obter o mês final do trimestre. Os resultados são mostrados aqui:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x

YR      MTH
----- 
2005      3
2005      6
2005      9
2005     12
```

Neste ponto, você tem o ano e o mês final de cada trimestre. Use esses valores para construir uma data, especificamente, o primeiro dia do último mês de cada trimestre. Use o operador de concatenação || para colar o ano e o mês e, em seguida, use a função TO\_DATE para converter em uma data:

```
select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x

Q_END
-----
- 01-MAR-
2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Os valores para Q-END são o primeiro dia do último mês de cada trimestre. Para chegar ao último dia do mês, use a função LAST\_DAY em Q-END. Para encontrar a data de início de cada trimestre, subtraia dois meses de Q-END usando a função ADD\_MONTHS.

### PostgreSQL

O primeiro passo é encontrar o ano e o trimestre com os quais você está trabalhando. Substring fora o ano da exibição em linha X (X.YRQ) usando a função SUBSTR. Para obter o trimestre, use o módulo 10 em YRQ. Assim que tiver o trimestre, multiplique por 3 para obter o mês final do trimestre. Os resultados são mostrados aqui:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x

YR      MTH
-----
2005      3
2005      6
2005      9
2005     12
```

Neste ponto, você tem o ano e o mês final de cada trimestre. Use esses valores para construir uma data, especificamente, o primeiro dia do último mês de cada trimestre. Use o operador de concatenação `||` para colar o ano e o mês e, em seguida, use a função `TO_DATE` para converter em uma data:

```
select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
) x

Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Os valores para `Q-END` são o primeiro dia do último mês de cada trimestre. Para chegar ao último dia do mês, adicione um mês a `Q-END` e subtraia um dia. Para encontrar a data de início de cada trimestre, subtraia dois meses de `Q-END`. Lance o resultado final como datas.

## MySQL

O primeiro passo é encontrar o ano e o trimestre com os quais você está trabalhando. Substring fora o ano da exibição em linha X (`X.YRQ`) usando a função `SUBSTR`. Para obter o trimestre, use o módulo 10 em `YRQ`. Assim que tiver o trimestre, multiplique por três para obter o mês final do trimestre. Os resultados são mostrados aqui:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
) x

YR      MTH
----- 
2005      3
2005      6
2005      9
2005     12
```

Neste ponto, você tem o ano e o mês final de cada trimestre. Use esses valores para construir uma data, especificamente, o último dia de cada trimestre.

Use a função CONCAT para unir o ano e o mês e, em seguida, use a função STR\_TO\_DATE para converter em uma data. Use a função LAST\_DAY para encontrar o último dia de cada trimestre:

```
select last_day(
    str_to_date(
        concat(
            substr(yrq,1,4),mod(yrq,10)*3),'%Y%m')) q_end
from (
    select 20051 as yrq from t1 union all
    select 20052 as yrq from t1 union all
    select 20053 as yrq from t1 union all
    select 20054 as yrq from t1
) x

Q_END
-----
31-MAR-2005
30-JUN-2005
30-SEP-2005
31-DEC-2005
```

Como você já tem o final de cada trimestre, só falta encontrar a data de início de cada trimestre. Use a função DAY para retornar o dia do mês em que cai o final de cada trimestre e subtraia isso de Q-END usando a função ADDDATE para obter o final do mês anterior; adicione um dia para trazê-lo para o primeiro dia do último mês de cada trimestre. A última etapa é usar a função DATE\_ADD para subtrair dois meses do primeiro dia do último mês de cada trimestre para chegar à data de início de cada trimestre.

## SQL Server

O primeiro passo é encontrar o ano e o trimestre com os quais você está trabalhando. Substring fora o ano da exibição em linha X (X.YRQ) usando a função SUBSTRING. Para obter o trimestre, use o módulo 10 em YRQ. Assim que tiver o trimestre, multiplique por três para obter o mês final do trimestre. Os resultados são mostrados aqui:

```
select substring(yrq,1,4) yr, yr%10*3 mth
from (
    select 20051 yrq from t1 union all
    select 20052 yrq from t1 union all
    select 20053 yrq from t1 union all
    select 20054 yrq from t1
) x

YR      MTH
-----
2005      3
2005      6
2005      9
2005     12
```

Neste ponto, você tem o ano e o mês final de cada trimestre. Use esses valores para construir uma data, especificamente, o primeiro dia do último mês de cada trimestre. Use o operador de concatenação + para unir o ano e o mês e, em seguida, usar a função CAST para converter em uma data:

```
select cast(substring(cast(yrq as varchar),1,4) + '-' +
           cast(yrq%10*3 as varchar) + '-1' as datetime) q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
) x

Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Os valores para Q-END são o primeiro dia do último mês de cada trimestre. Para chegar ao último dia do mês, adicione um mês a Q-END e subtraia um dia usando a função DATEADD. Para encontrar a data de início de cada trimestre, subtraia dois meses de Q-END usando a função DATEADD.

## 9.10 Preenchendo datas ausentes

### Problema

Você precisa gerar uma linha para cada data (ou cada mês, semana ou ano) dentro de um determinado intervalo. Esses conjuntos de linhas geralmente são usados para gerar relatórios de resumo. Por exemplo, você deseja contar o número de funcionários contratados todos os meses de cada ano em que algum funcionário foi contratado. Examinando as datas de todos os funcionários contratados, houve contratações de 2000 a 2003:

```
select distinct
       extract(year from hiredate) as year
  from emp

YEAR
-----
2000
2001
2002
2003
```

Você deseja determinar o número de funcionários contratados a cada mês de 2000 a 2003. Uma parte do conjunto de resultados desejado é mostrada aqui:

MTH	NUM_HIRED
- 01-JAN-2001	0
01-FEB-2001	2
01-MAR-2001	0
01-APR-2001	1
01-MAY-2001	1
01-JUN-2001	1
01-JUL-2001	0
01-AUG-2001	0
01-SEP-2001	2
01-OCT-2001	0
01-NOV-2001	1
1- DEC-2001	2

## Solução

O truque aqui é que você deseja retornar uma linha para cada mês, mesmo que nenhum funcionário tenha sido contratado (ou seja, a contagem seria zero). Como não há um funcionário contratado todos os meses entre 2000 e 2003, você mesmo deve gerar esses meses e, em seguida, fazer uma junção externa à tabela EMP em HIREDATE (truncando o HIREDATE real para seu mês para que possa corresponder aos meses gerados quando possível).

## DB2

Use a cláusula recursiva WITH para gerar todos os meses (o primeiro dia de cada mês de 1º de janeiro de 2000 a 1º de dezembro de 2003). Depois de ter todos os meses para o intervalo de datas necessário, junte-se à tabela EMP e use a função agregada COUNT para contar o número de contratações para cada mês:

```

1  with x (start_date,end_date)
2    as (
3 select (min(hiredate)
4        dayofyear(min(hiredate)) day +1 day) start_date,
5      (max(hiredate)
6        dayofyear(max(hiredate)) day +1 day) +1 year end_date
7  from emp
8 union all
9 select start_date +1 month, end_date
10   from x
11  where (start_date +1 month) < end_date
12 )
13 select x.start_date mth, count(e.hiredate) num_hired
14   from x left join emp e
15     on (x.start_date = (e.hiredate-(day(hiredate)-1) day))
16 group by x.start_date
17 order by 1

```

## Oracle

Use a cláusula CONNECT BY para gerar cada mês entre 2000 e 2003. Em seguida, junte-se à tabela EMP e use a função de agregação COUNT para contar o número de funcionários contratados em cada mês:

```

1  with x
2    as (
3 select add_months(start_date,level-1) start_date
4   from (
5 select min(trunc(hiredate,'y')) start_date,
6       add_months(max(trunc(hiredate,'y')),12) end_date
7   from emp
8     )
9 connect by level <= months_between(end_date,start_date)
10 )
11 select x.start_date MTH, count(e.hiredate) num_hired
12   from x left join emp e
13     on (x.start_date = trunc(e.hiredate,'mm'))
14 group by x.start_date
15 order by 1

```

## PostgreSQL

Use CTE para preencher os meses desde a primeira contratação e depois LEFT OUTER JOIN na tabela EMP usando o mês e o ano de cada mês gerado para habilitar o COUNT do número de contratados em cada período:

```

with recursive x (start_date, end_date)
as
(
  select
    cast(min(hiredate) - (cast(extract(day from min(hiredate))
      as integer) - 1) as date)
    , max(hiredate)
  from emp
  union all
    select cast(start_date + interval '1 month' as date)
      , end_date
    from x
    where start_date < end_date
)
select x.start_date, count(hiredate)
from x left join emp
  on (extract(month from start_date) =
        extract(month from emp.hiredate)
      and extract(year from start_date)
        = extract(year from emp.hiredate))
group by x.start_date
order by 1

```

Use um CTE recursivo para gerar cada mês entre as datas inicial e final e, em seguida, verifique as contratações usando uma junção externa para a tabela EMP:

```
with recursive x (start_date,end_date)
as
(
select
    adddate(min(hiredate),
    -dayofyear(min(hiredate))+1) start_date
    ,adddate(max(hiredate),
    -dayofyear(max(hiredate))+1) end_date
    from emp
union all
    select date_add(start_date,interval 1 month)
    , end_date
    from x
    where date_add(start_date, interval 1 month) < end_date
)
select x.start_date mth, count(e.hiredate) num_hired
    from x left join emp e
    on (extract(year_month from start_date)
        =
        extract(year_month from e.hiredate))
    group by x.start_date
    order by 1;
```

## SQL Server

Use a cláusula recursiva WITH para gerar todos os meses (o primeiro dia de cada mês de 1º de janeiro de 2000 a 1º de dezembro de 2003). Depois de ter todos os meses para o intervalo de datas necessário, junte-se à tabela EMP e use a função agregada COUNT para contar o número de contratações para cada mês:

```
1  with x (start_date,end_date)
2    as (
3 select (min(hiredate) -
4         datepart(dy,min(hiredate))+1) start_date,
5         dateadd(yy,1,
6         (max(hiredate) -
7             datepart(dy,max(hiredate))+1)) end_date
8   from emp
9 union all
10 select dateadd(mm,1,start_date), end_date
11   from x
12  where dateadd(mm,1,start_date) <
end_date 13 )
14 select x.start_date mth, count(e.hiredate) num_hired
15   from x left join emp e
16     on (x.start_date =
```

```

15 dateadd(dd,-day(e.hiredate)+1,e.hiredate))
16 group by x.start_date
17 order by 1

```

## Discussão

### DB2

A primeira etapa é gerar todos os meses (na verdade, o primeiro dia de cada mês) de 2000 a 2003. Comece usando a função DAYOFYEAR nas MIN e MAX HIREDATES para encontrar os meses limite:

```

select (min(hiredate)
        dayofyear(min(hiredate)) day +1 day) start_date,
       (max(hiredate)
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
  from emp

START_DATE   END_DATE
-----
01-JAN-2000  01-JAN-2004

```

A próxima etapa é adicionar meses repetidamente a START\_DATE para retornar todos os meses necessários para o conjunto de resultados final. O valor para END\_DATE é um dia a mais do que deveria. Tudo bem. À medida que adiciona meses recursivamente a START\_DATE, pode parar antes de atingir END\_DATE. Uma parte dos meses criados é mostrada aqui:

```

with x (start_date,end_date)
  as (
select (min(hiredate)
        dayofyear(min(hiredate)) day +1 day) start_date,
       (max(hiredate)
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
  from emp
 union all
select start_date +1 month, end_date
  from x
 where (start_date +1 month) < end_date
)
select *
  from x

START_DATE   END_DATE
-----
01-JAN-2000  01-JAN-2004
01-FEB-2000  01-JAN-2004
01-MAR-2000  01-JAN-2004
...
01-OCT-2003  01-JAN-2004
01-NOV-2003  01-JAN-2004
01-DEC-2003  01-JAN-2004

```

Neste ponto, você tem todos os meses de que precisa e pode simplesmente se associar externamente a EMP.HIREDATE. Como o dia de cada START\_DATE é o primeiro dia do mês, trunque EMP.HIREDATE para o primeiro dia do mês. Finalmente, use a função agregada COUNT em EMP.HIREDATE.

### Oracle

A primeira etapa é gerar o primeiro dia de cada mês de 2000 a 2003. Comece usando TRUNC e ADD\_MONTHS junto com os valores MIN e MAX HIREDATE para encontrar os meses limite:

```
select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
  from emp

START_DATE   END_DATE
-----
01-JAN-2000  01-JAN-2004
```

Em seguida, adicione meses repetidamente a START\_DATE para retornar todos os meses necessários para o conjunto de resultados final. O valor para END\_DATE é um dia a mais do que deveria, o que é aceitável. À medida que adiciona meses recursivamente a START\_DATE, pode parar antes de atingir END\_DATE. Uma parte dos meses criados é mostrada aqui:

```
with x as (
  select add_months(start_date,level-1) start_date
    from (
  select min(trunc(hiredate,'y')) start_date,
         add_months(max(trunc(hiredate,'y')),12) end_date
    from emp
      )
  connect by level <= months_between(end_date,start_date)
)
select *
  from x

START_DATE
-----
- 01-JAN-
2000
01-FEB-2000
01-MAR-2000
...
01-OCT-2003
01-NOV-2003
01-DEC-2003
```

Neste ponto, você tem todos os meses de que precisa e pode simplesmente se associar externamente a EMP.HIREDATE. Como o dia de cada START\_DATE é o primeiro dia do mês, trunque EMP.HIREDATE para o primeiro dia do mês em que está. A etapa final é usar a função agregada COUNT em EMP.HIREDATE.

## PostgreSQL

Esta solução usa um CTE para gerar os meses que você precisa e é semelhante às soluções subsequentes para MySQL e SQL Server. A primeira etapa é criar as datas limite usando funções agregadas. Você pode simplesmente encontrar as datas de contratação mais cedo e mais tarde usando as funções MIN() e MAX(), mas a saída faz mais sentido se você encontrar o primeiro dia do mês contendo a data de contratação mais antiga.

## MySQL

Primeiro, encontre as datas limite usando as funções agregadas MIN e MAX juntamente com as funções DAYOFYEAR e ADDDATE. O conjunto de resultados mostrado aqui é da visualização inline X:

```
with recursive x (start_date,end_date)
      as (
    select
        adddate(min(hiredate),
        -dayofyear(min(hiredate))+1)  start_date
      ,adddate(max(hiredate),
        -dayofyear(max(hiredate))+1)  end_date
    from emp
  union all
  select date_add(start_date,interval 1 month)
  , end_date
  from x
  where date_add(start_date, interval 1 month) < end_date
  )
select * from x

select adddate(min(hiredate),-dayofyear(min(hiredate))+1)
      min_hd, adddate(max(hiredate),
        -dayofyear(max(hiredate))+1) max_hd
    from emp

MIN_HD      MAX_HD
----- -----
- 01-JAN-2000 01-JAN-
2003
```

Em seguida, incremente MAX\_HD para o último mês do ano pelo CTE:

MTH
-----
01-JAN-2000
01-FEB-2000
01-MAR-2000
...
01-OCT-2003
01-NOV-2003
01-DEC-2003

Agora que você tem todos os meses necessários para o conjunto de resultados final, faça a junção externa para EMP.HIREDATE (certifique-se de truncar EMP.HIREDATE para o primeiro dia do mês)

e use a função agregada COUNT em EMP.HIREDATE para contar o número de contratações em cada mês.

## SQL Server

Comece gerando todos os meses (na verdade, o primeiro dia de cada mês) de 2000 a 2003. Em seguida, encontre os meses limite aplicando a função DAYOFYEAR aos MIN e MAX HIREDATES:

```
select (min(hiredate) -
        datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
        (max(hiredate) -
        datepart(dy,max(hiredate))+1)) end_date
  from emp

START_DATE END_DATE
-----
01-JAN-2000 01-JAN-2004
```

A próxima etapa é adicionar meses repetidamente a START\_DATE para retornar todos os meses necessários para o conjunto de resultados final. O valor para END\_DATE é um dia a mais do que deveria, o que é bom, pois você pode parar de adicionar recursivamente meses a START\_DATE antes de atingir END\_DATE. Uma parte dos meses criados é mostrada aqui:

```
with x (start_date,end_date)
  as (
select (min(hiredate) -
        datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
        (max(hiredate) -
        datepart(dy,max(hiredate))+1)) end_date
  from emp
 union all
select dateadd(mm,1,start_date), end_date
  from x
 where dateadd(mm,1,start_date) < end_date
)
select *
  from x

START_DATE END_DATE
-----
01-JAN-2000 01-JAN-2004
01-FEB-2000 01-JAN-2004
01-MAR-2000 01-JAN-2004
...
01-OCT-2003 01-JAN-2004
01-NOV-2003 01-JAN-2004
01-DEC-2003 01-JAN-2004
```

Neste ponto, você tem todos os meses de que precisa. Simplesmente junção externa para EMP.HIREDATE. Como o dia de cada START\_DATE é o primeiro dia do mês, trunque EMP.HIREDATE para o primeiro dia do mês. A etapa final é usar a função agregada COUNT em EMP.HIREDATE.

## 9.11 Pesquisando em unidades específicas de tempo

### Problema

Você deseja pesquisar datas que correspondam a um determinado mês, dia da semana ou alguma outra unidade de tempo. Por exemplo, você deseja localizar todos os funcionários contratados em fevereiro ou dezembro, bem como os funcionários contratados em uma terça-feira.

### Solução

Use as funções fornecidas pelo seu RDBMS para encontrar nomes de mês e dia da semana para datas. Esta receita em particular pode ser útil em vários lugares. Considere, se você deseja pesquisar HIREDATES, mas deseja ignorar o ano extraíndo o mês (ou qualquer outra parte do HIREDATE em que esteja interessado), você pode fazê-lo. As soluções de exemplo para esse problema pesquisam por nome de mês e dia da semana. Ao estudar as funções de formatação de data fornecidas pelo seu RDBMS, você pode facilmente modificar essas soluções para pesquisar por ano, trimestre, combinação de ano e trimestre, combinação de mês e ano, etc.

#### DB2 e MySQL

Use as funções MONTHNAME e DAYNAME para encontrar o nome do mês e dia da semana em que um funcionário foi contratado, respectivamente:

```
1 select ename
2   from emp
3 where monthname(hiredate) in ('February','December')
4   or dayname(hiredate) = 'Tuesday'
```

#### Oracle e PostgreSQL

Use a função TO\_CHAR para encontrar os nomes do mês e dia da semana em que um funcionário foi contratado. Use a função RTRIM para remover espaços em branco à direita:

```
1 select ename
2   from emp
3 where rtrim(to_char(hiredate,'month')) in ('february','december')
4   or rtrim(to_char(hiredate,'day')) = 'tuesday '
```

Use a função DATENAME para encontrar os nomes do mês e dia da semana em que um funcionário foi contratado:

```
1 select ename
2   from emp
3 where datename(m,hiredate) in ('February', 'December')
4   or datename(dw,hiredate) = 'Tuesday'
```

## Discussão

A chave para cada solução é simplesmente saber quais funções usar e como usá-las. Para verificar quais são os valores de retorno, coloque as funções na cláusula SELECT e examine a saída. Aqui está o conjunto de resultados para funcionários em DEPTNO 10 (usando a sintaxe do SQL Server):

```
select ename,datename(m,hiredate) mth,datename(dw,hiredate) dw
  from emp
 where deptno = 10
```

ENAME	MTH	DW
CLARK	June	Tuesday
KING	November	Tuesday
MILLER	January	Saturday

Depois de saber o que a(s) função(ões) retorna(m), é fácil encontrar linhas usando as funções mostradas em cada uma das soluções.

## 9.12 Comparando registros usando partes específicas de uma data

### Problema

Você deseja descobrir quais funcionários foram contratados no mesmo mês e dia da semana. Por exemplo, se um funcionário foi contratado na segunda-feira, 10 de março de 2008, e outro funcionário foi contratado na segunda-feira, 2 de março de 2001, você deseja que os dois apareçam como uma correspondência, já que o dia da semana e o mês correspondem. Na tabela EMP, apenas três funcionários atendem a esse requisito. Você deseja retornar o seguinte conjunto de resultados:

```
MSG
-----
JAMES was hired on the same month and weekday as FORD
SCOTT was hired on the same month and weekday as JAMES
SCOTT was hired on the same month and weekday as FORD
```

## Solução

Como você deseja comparar o HIREDATE de um funcionário com o HIREDATE dos outros funcionários, será necessário unir automaticamente a tabela EMP. Isso torna cada combinação possível de HIREDATES disponível para você comparar. Em seguida, basta extrair o dia da semana e o mês de cada HIREDATE e comparar.

### DB2

Após a junção automática da tabela EMP, use a função DAYOFWEEK para retornar o dia numérico da semana. Use a função MONTHNAME para retornar o nome do mês:

```

1 select a.ename ||
2      ' was hired on the same month and weekday as ' ||
3      b.ename msg
4  from emp a, emp b
5 where (dayofweek(a.hiredate),monthname(a.hiredate)) =
6      (dayofweek(b.hiredate),monthname(b.hiredate))
7  and a.empno < b.empno
8 order by a.ename

```

### Oracle e PostgreSQL

Após a junção automática da tabela EMP, use a função TO\_CHAR para formatar o HIREDATE em dia da semana e mês para comparação:

```

1 select a.ename ||
2      ' was hired on the same month and weekday as ' ||
3      b.ename as msg
4  from emp a, emp b
5 where to_char(a.hiredate,'DMON') =
6      to_char(b.hiredate,'DMON')
7  and a.empno < b.empno
8 order by a.ename

```

### MySQL

Após a junção automática da tabela EMP, use a função DATE\_FORMAT para formatar o HIREDATE em dia da semana e mês para comparação:

```

1 select concat(a.ename,
2 ' was hired on the same month and weekday as ',
3 b.ename) msg
4 from emp a, emp b
5 where date_format(a.hiredate,'%w%M') =
6 date_format(b.hiredate,'%w%M')
7 and a.empno < b.empno
8 order by a.ename

```

Após a junção automática da tabela EMP, use a função DATENAME para formatar o HIREDATE em dia da semana e mês para comparação:

```
1 select a.ename +
2       ' was hired on the same month and weekday as ' +
3       b.ename msg
4  from emp a, emp b
5 where datename(dw,a.hiredate) = datename(dw,b.hiredate)
6   and datename(m,a.hiredate) = datename(m,b.hiredate)
7   and a.empno < b.empno
8 order by a.ename
```

## Discussão

A única diferença entre as soluções é a função de data usada para formatar HIREDATE. Usaremos a solução Oracle/PostgreSQL nesta discussão (porque é a mais curta para digitar), mas a explicação vale para as outras soluções também.

A primeira etapa é ingressar no EMP automaticamente para que cada funcionário tenha acesso aos HIREDATES dos outros funcionários. Considere os resultados da consulta mostrados aqui (filtrados para SCOTT):

```
select a.ename as scott, a.hiredate as scott_hd,
       b.ename as other_emps, b.hiredate as other_hds
  from emp a, emp b
 where a.ename = 'SCOTT'
   and a.empno != b.empno
```

SCOTT	SCOTT_HD	OTHER_EMPS	OTHER_HDS
SCOTT	09-DEC-2002	SMITH	17-DEC-2000
SCOTT	09-DEC-2002	ALLEN	20-FEB-2001
09-DEC-2002	WARD	22-FEB-2001	SCOTT
09-DEC-2002	JONES	02-APR-2001	SCOTT
09-DEC-2002	MARTIN	28-SEP-2001	SCOTT
09-DEC-2002	BLAKE	01-MAY-2001	SCOTT
CLARK	09-JUN-2001	SCOTT	09-DEC-2002
17-NOV-2001	SCOTT	09-DEC-2002	KING
17-NOV-2001	SCOTT	09-DEC-2002	TURNER
08-SEP-2001	SCOTT	09-DEC-2002	ADAMS
12-JAN-2003	SCOTT	09-DEC-2002	JAMES
03-DEC-2001	SCOTT	09-DEC-2002	FORD
23-JAN-2002	SCOTT	09-DEC-2002	MILLER

Ao unir automaticamente a tabela EMP, você pode comparar o HIREDATE de SCOTT com o HIREDATE de todos os outros funcionários. O filtro em EMPNO é para que HIREDATE de SCOTT não seja retornado como um dos OTHER\_HDS.

A próxima etapa é usar a(s) função(ões) de formatação de data fornecida(s) pelo seu RDBMS para comparar o dia da semana e o mês dos HIREDATES e manter apenas aqueles que correspondem:

```
select a.ename as emp1, a.hiredate as emp1_hd,
       b.ename as emp2, b.hiredate as emp2_hd
  from emp a, emp b
 where to_char(a.hiredate,'DMON') =
       to_char(b.hiredate,'DMON')
   and a.empno != b.empno
 order by 1
```

EMP1	EMP1_HD	EMP2	EMP2_HD
FORD	03-DEC-2001	SCOTT	09-DEC-2002
FORD	03-DEC-2001	JAMES	03-DEC-2001
JAMES	03-DEC-2001	SCOTT	09-DEC-2002
JAMES	03-DEC-2001	FORD	03-DEC-2001
SCOTT	09-DEC-2002	JAMES	03-DEC-2001
SCOTT	09-DEC-2002	FORD	03-DEC-2001

Neste ponto, os HIREDATEs são correspondidos corretamente, mas há seis linhas no conjunto de resultados em vez das três na seção “Problema” desta receita. O motivo das linhas extras é o filtro em EMPNO. Ao usar “diferente de”, você não filtra os recíprocos. Por exemplo, a primeira linha corresponde a FORD e SCOTT e a última linha corresponde a SCOTT e FORD. As seis linhas no conjunto de resultados são tecnicamente precisas, mas redundantes. Para remover a redundância, use “less than” (os HIREDATEs são removidos para aproximar as consultas intermediárias do conjunto de resultados final):

```
select a.ename as emp1, b.ename as emp2
  from emp a, emp b
 where to_char(a.hiredate,'DMON') =
       to_char(b.hiredate,'DMON')
   and a.empno < b.empno
 order by 1
```

EMP1	EMP2
JAMES	FORD
SCOTT	JAMES
SCOTT	FORD

A etapa final é simplesmente concatenar o conjunto de resultados para formar a mensagem.

## 9.13 Identificação de intervalos de datas sobrepostos

### Problema

Você deseja localizar todas as instâncias de um funcionário iniciando um novo projeto antes de encerrar um projeto existente. Considere a tabela EMP\_PROJECT:

```
select *
  from emp_project
```

EMPNO	ENAME	PROJ_ID	PROJ_START	PROJ_END
7782	CLARK	1	16-JUN-2005	18-JUN-2005
7782	CLARK	4	19-JUN-2005	24-JUN-2005
7782	CLARK	7	22-JUN-2005	25-JUN-2005
7782	CLARK	10	25-JUN-2005	28-JUN-2005
7782	CLARK	13	28-JUN-2005	02-JUL-2005
7839	KING	2	17-JUN-2005	21-JUN-2005
7839	KING	8	23-JUN-2005	25-JUN-2005
7839	KING	14	29-JUN-2005	30-JUN-2005
7839	KING	11	26-JUN-2005	27-JUN-2005
7839	KING	5	20-JUN-2005	24-JUN-2005
7934	MILLER	3	18-JUN-2005	22-JUN-2005
7934	MILLER	12	27-JUN-2005	28-JUN-2005
7934	MILLER	15	30-JUN-2005	03-JUL-2005
7934	MILLER	9	24-JUN-2005	27-JUN-2005
7934	MILLER	6	21-JUN-2005	23-JUN-2005

Observando os resultados do funcionário KING, você vê que KING começou PROJ\_ID 8 antes de terminar PROJ\_ID 5 e começou PROJ\_ID 5 antes de terminar PROJ\_ID 2. Você deseja retornar o seguinte conjunto de resultados:

EMPNO	ENAME	MSG
7782	CLARK	project 7 overlaps project 4
7782	CLARK	project 10 overlaps project 7
7782	CLARK	project 13 overlaps project 10
7839	KING	project 8 overlaps project 5
7839	KING	project 5 overlaps project 2
7934	MILLER	project 12 overlaps project 9
7934	MILLER	project 6 overlaps project 3

## Solução

A chave aqui é encontrar linhas onde PROJ\_START (a data em que o novo projeto começa) ocorre na data PROJ\_START de outro projeto ou depois dela e antes ou na data PROJ\_END desse outro projeto. Para começar, você precisa ser capaz de comparar cada projeto entre si (para o mesmo funcionário). Ao associar automaticamente EMP\_PROJECT no funcionário, você gera todas as combinações possíveis de dois projetos para cada funcionário. Para encontrar as sobreposições, basta localizar as linhas em que PROJ\_START para qualquer PROJ\_ID está entre PROJ\_START e PROJ\_END para outro PROJ\_ID do mesmo funcionário.

### DB2, PostgreSQL e Oracle

Ingresse automaticamente no EMP\_PROJECT. Em seguida, use o operador de concatenação || para construir a mensagem que explica quais projetos se sobrepõem:

```

1 select a.empno,a.ename,
2       'project'||b.proj_id||
3       ' overlaps project'||a.proj_id as msg
4   from emp_project a,
5        emp_project b
6  where a.empno = b.empno
7    and b.proj_start >= a.proj_start
8    and b.proj_start <= a.proj_end
9    and a.proj_id != b.proj_id

```

## MySQL

Ingresse automaticamente no EMP\_PROJECT. Em seguida, use a função CONCAT para construir a mensagem que explica quais projetos se sobrepõem:

```

1 select a.empno,a.ename,
2       concat('project ',b.proj_id,
3              ' overlaps project ',a.proj_id) as msg
4   from emp_project a,
5        emp_project b
6  where a.empno = b.empno
7    and b.proj_start >= a.proj_start
8    and b.proj_start <= a.proj_end
9    and a.proj_id != b.proj_id

```

## SQL Server

Ingresse automaticamente no EMP\_PROJECT. Em seguida, use o operador de concatenação + para construir a mensagem que explica quais projetos se sobrepõem:

```

1 select a.empno,a.ename,
2       'project '+b.proj_id+
3       ' overlaps project '+a.proj_id as msg
4   from emp_project a,
5        emp_project b
6  where a.empno = b.empno
7    and b.proj_start >= a.proj_start
8    and b.proj_start <= a.proj_end
9    and a.proj_id != b.proj_id

```

## Discussão

A única diferença entre as soluções está na concatenação da string, portanto, uma discussão usando a sintaxe do DB2 cobrirá todas as três soluções. O primeiro passo é um self-join de EMP\_PROJECT para que as datas PROJ\_START possam ser comparadas entre os diferentes projetos. A saída da autojunção para o funcionário KING é mostrada aqui. Você pode observar como cada projeto pode “ver” os outros projetos:

```

select a.ename,
       a.proj_id as a_id,
       a.proj_start as
       a_start, a.proj_end as
       a_end, b.proj_id as
       b_id, b.proj_start as
       b_start
  from emp_project a,
       emp_project b
 where a.ename = 'KING'
   and a.empno = b.empno
   and a.proj_id != b.proj_id
order by 2

```

ENAME	A_ID	A_START	A_END	B_ID	B_START
KING	2	17-JUN-2005	21-JUN-2005	8	23-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	14	29-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	11	26-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	5	20-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	2	17-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	8	23-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	11	26-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	14	29-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	2	17-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	14	29-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	5	20-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	11	26-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	2	17-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	8	23-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	14	29-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	5	20-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	2	17-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	8	23-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	5	20-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	11	26-JUN-2005

Como você pode ver no conjunto de resultados, a junção automática facilita a localização de datas sobrepostas: basta retornar cada linha em que B\_START ocorre entre A\_START e A\_END. Se você observar a cláusula WHERE nas linhas 7 e 8 da solução:

```

and b.proj_start >= a.proj_start
and b.proj_start <= a.proj_end

```

está fazendo exatamente isso. Depois de ter as linhas necessárias, construir as mensagens é apenas uma questão de concatenar os valores de retorno.

Os usuários do Oracle podem usar a função de janela LEAD OVER para evitar o self-join, se o número máximo de projetos por funcionário for fixo. Isso pode ser útil se a autojunção for cara para seus resultados específicos (se a autojunção exigir mais recursos do que os tipos necessários para LEAD OVER). Por exemplo, considere a alternativa para o funcionário KING usando LEAD OVER:

```

select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
           when lead(proj_start,1)over(order by proj_start)
               between proj_start and proj_end
           then lead(proj_id)over(order by proj_start)
           when lead(proj_start,2)over(order by proj_start)
               between proj_start and proj_end
           then lead(proj_id)over(order by proj_start)
           when lead(proj_start,3)over(order by proj_start)
               between proj_start and proj_end
           then lead(proj_id)over(order by proj_start)
           when lead(proj_start,4)over(order by proj_start)
               between proj_start and proj_end
           then lead(proj_id)over(order by proj_start)
       end is_overlap
  from emp_project
 where ename = 'KING'

EMPNO ENAME PROJ_ID PROJ_START PROJ_END IS_OVERLAP
----- ----- ----- ----- -----
7839 KING      2 17-JUN-2005 21-JUN-2005      5
7839 KING      5 20-JUN-2005 24-JUN-2005      8
7839 KING      8 23-JUN-2005 25-JUN-2005
7839 KING     11 26-JUN-2005 27-JUN-2005
7839 KING     14 29-JUN-2005 30-JUN-2005

```

Como o número de projetos é fixado em cinco para o funcionário KING, você pode usar LEAD OVER para examinar as datas de todos os projetos sem autojunção. A partir daqui, é fácil produzir o conjunto de resultados final. Simplesmente mantenha as linhas onde IS\_OVERLAP não é NULL:

```

select empno,ename,
       'project'||is_overlap||
       ' overlaps project'||proj_id msg
  from (
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
           when lead(proj_start,1)over(order by proj_start)
               between proj_start and proj_end
           then lead(proj_id)over(order by proj_start)
           when lead(proj_start,2)over(order by proj_start)
               between proj_start and proj_end
           then lead(proj_id)over(order by proj_start)
           when lead(proj_start,3)over(order by proj_start)

```

```

        between proj_start and proj_end
    then lead(proj_id)over(order by proj_start)
    when lead(proj_start,4)over(order by proj_start)
        between proj_start and proj_end
    then lead(proj_id)over(order by proj_start)
    end is_overlap
  from emp_project
  where ename = 'KING'
  )
where is_overlap is not null

```

EMPNO ENAME MSG

```

-----
```

EMPNO	ENAME	MSG
7839	KING	project 5 overlaps project 2
7839	KING	project 8 overlaps project 5

Para permitir que a solução funcione para todos os funcionários (não apenas para a KING), particione por ENAME na função LEAD OVER:

```

select empno,ename,
      'project'||is_overlap||
      ' overlaps project'||proj_id msg
  from (
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
         when lead(proj_start,1)over(partition by ename
                                       order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(partition by ename
                               order by proj_start)
         when lead(proj_start,2)over(partition by ename
                                       order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(partition by ename
                               order by proj_start)
         when lead(proj_start,3)over(partition by ename
                                       order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(partition by ename
                               order by proj_start)
         when lead(proj_start,4)over(partition by ename
                                       order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(partition by ename
                               order by proj_start)
         end is_overlap
  from emp_project
  )
where is_overlap is not null

```

EMPNO	ENAME	MSG
7782	CLARK	project 7 overlaps project 4
7782	CLARK	project 10 overlaps project 7
7782	CLARK	project 13 overlaps project 10
7839	KING	project 5 overlaps project 2
7839	KING	project 8 overlaps project 5
7934	MILLER	project 6 overlaps project 3
7934	MILLER	project 12 overlaps project 9

## 9.14 Resumindo

As manipulações de datas são um problema comum para qualquer pessoa que consulte um banco de dados — uma série de eventos armazenados com suas datas inspira os usuários corporativos a fazerem perguntas criativas baseadas em datas. Ao mesmo tempo, as datas são uma das áreas menos padronizadas de SQLs entre fornecedores. Esperamos que você tire deste capítulo uma ideia de como, mesmo quando a sintaxe é diferente, ainda existe uma lógica comum que pode ser aplicada a consultas que usam datas.

## CAPÍTULO 10

# Trabalhando com intervalos

Este capítulo é sobre consultas “cotidianas” que envolvem intervalos. Os intervalos são comuns na vida cotidiana. Por exemplo, os projetos nos quais trabalhamos variam em períodos de tempo consecutivos. No SQL, muitas vezes é necessário pesquisar intervalos, gerar intervalos ou manipular dados baseados em intervalos. As consultas sobre as quais você lerá aqui são um pouco mais complicadas do que as encontradas nos capítulos anteriores, mas são igualmente comuns e começarão a lhe dar uma noção do que o SQL pode realmente fazer por você quando aprender a tirar o máximo proveito disso.

## 10.1 Localizando um intervalo de valores consecutivos

### Problema

Você deseja determinar quais linhas representam um intervalo de projetos consecutivos. Considere o seguinte conjunto de resultados da exibição V, que contém dados sobre um projeto e suas datas de início e término:

```
select *  
from V  
  
PROJ_ID PROJ_START PROJ_END  
-----  
1 01-JAN-2020 02-JAN-2020  
2 02-JAN-2020 03-JAN-2020  
3 03-JAN-2020 04-JAN-2020  
4 04-JAN-2020 05-JAN-2020  
5 06-JAN-2020 07-JAN-2020  
6 16-JAN-2020 17-JAN-2020  
7 17-JAN-2020 18-JAN-2020  
8 18-JAN-2020 19-JAN-2020  
9 19-JAN-2020 20-JAN-2020  
10 21-JAN-2020 22-JAN-2020
```

```
11 26-JAN-2020 27-JAN-2020
12 27-JAN-2020 28-JAN-2020
13 28-JAN-2020 29-JAN-2020
14 29-JAN-2020 30-JAN-2020
```

Excluindo a primeira linha, o PROJ\_START de cada linha deve ser igual ao PROJ\_END da linha anterior ("antes" é definido como PROJ\_ID-1 para a linha atual). Examinando as cinco primeiras linhas da exibição V, os PROJ\_IDS 1 a 3 fazem parte do mesmo "grupo", pois cada PROJ\_END é igual ao PROJ\_START da linha seguinte. Como você deseja encontrar o intervalo de datas para projetos consecutivos, gostaria de retornar todas as linhas em que o PROJ\_END atual é igual ao PROJ\_START da próxima linha. Se as cinco primeiras linhas incluírem todo o conjunto de resultados, você gostaria de retornar apenas as três primeiras linhas. O conjunto de resultados final (usando todas as 14 linhas da exibição V) deve ser:

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2020	02-JAN-2020
2	02-JAN-2020	03-JAN-2020
3	03-JAN-2020	04-JAN-2020
6	16-JAN-2020	17-JAN-2020
7	17-JAN-2020	18-JAN-2020
8	18-JAN-2020	19-JAN-2020
11	26-JAN-2020	27-JAN-2020
12	27-JAN-2020	28-JAN-2020
13	28-JAN-2020	29-JAN-2020

As linhas com PROJ\_IDS 4, 5, 9, 10 e 14 são excluídas deste conjunto de resultados porque o PROJ\_END de cada uma dessas linhas não corresponde ao PROJ\_START da linha seguinte.

## Solução

Esta solução aproveita ao máximo a função de janela LEAD OVER para olhar para o BEGIN\_DATE da "próxima" linha, evitando assim a necessidade de autojunção, que era necessária antes que as funções de janela fossem amplamente introduzidas:

```
1 select proj_id, proj_start, proj_end
2 from (
3 select proj_id, proj_start, proj_end,
4 lead(proj_start)over(order by proj_id) next_proj_start
5 from V
6 ) alias
7 where next_proj_start = proj_end
```

## Discussão

### DB2, MySQL, PostgreSQL, SQL Server e Oracle

Embora seja possível desenvolver uma solução usando um self-join, a função de janela LEAD OVER é perfeita para esse tipo de problema, e mais intuitiva. A função LEAD OVER permite que você examine outras linhas sem realizar uma autojunção (embora a função deva impor ordem no conjunto de resultados para fazer isso). Considere os resultados da visualização em linha (linhas 3 a 5) para os IDs 1 e 4:

```
select *
  from (
select proj_id, proj_start, proj_end,
       lead(proj_start)over(order by proj_id) next_proj_start
  from v
   )
 where proj_id in ( 1, 4 )
```

PROJ_ID	PROJ_START	PROJ_END	NEXT_PROJ_START
1	01-JAN-2020	02-JAN-2020	02-JAN-2020
4	04-JAN-2020	05-JAN-2020	06-JAN-2020

Examinando esse trecho de código e seu conjunto de resultados, é particularmente fácil ver por que PROJ\_ID 4 foi excluído do conjunto de resultados final da solução completa. Ele foi excluído porque sua data PROJ\_END de 05-JAN-2020 não corresponde à data de início do "próximo" projeto de 06-JAN-2020.

A função LEAD OVER é extremamente útil quando se trata de problemas como este, principalmente ao examinar resultados parciais. Ao trabalhar com funções de janela, lembre-se de que elas são avaliadas após as cláusulas FROM e WHERE, portanto, a função LEAD OVER na consulta anterior deve ser incorporada em uma exibição em linha. Caso contrário, a função LEAD OVER é aplicada ao conjunto de resultados após a cláusula WHERE ter filtrado todas as linhas, exceto PROJ\_ID's 1 e 4.

Agora, dependendo de como você visualiza os dados, você pode querer incluir PROJ\_ID 4 no conjunto de resultados final. Considere as cinco primeiras linhas da visualização V:

```
select *
  from V
 where proj_id <= 5
```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2020	02-JAN-2020
2	02-JAN-2020	03-JAN-2020
3	03-JAN-2020	04-JAN-2020
4	04-JAN-2020	05-JAN-2020
5	06-JAN-2020	07-JAN-2020

Se o seu requisito é tal que PROJ\_ID 4 é de fato contíguo (porque PROJ\_START para PROJ\_ID 4 corresponde a PROJ\_END para PROJ\_ID 3) e apenas PROJ\_ID 5 deve ser descartado, a solução proposta para esta receita está incorreta (!) muito menos, incompleto:

```
select proj_id, proj_start, proj_end
  from (
select proj_id, proj_start, proj_end,
       lead(proj_start)over(order by proj_id) next_start
  from V
 where proj_id <= 5
   )
where proj_end = next_start

PROJ_ID PROJ_START PROJ_END
-----
1 01-JAN-2020 02-JAN-2020
2 02-JAN-2020 03-JAN-2020
3 03-JAN-2020 04-JAN-2020
```

Se você acredita que PROJ\_ID 4 deve ser incluído, basta adicionar LAG OVER à consulta e usar um filtro adicional na cláusula WHERE:

```
select proj_id, proj_start, proj_end
  from (
select proj_id, proj_start, proj_end,
       lead(proj_start)over(order by proj_id) next_start,
       lag(proj_end)over(order by proj_id) last_end
  from V
 where proj_id <= 5
   )
where proj_end = next_start
  or proj_start = last_end

PROJ_ID PROJ_START PROJ_END
-----
1 01-JAN-2020 02-JAN-2020
2 02-JAN-2020 03-JAN-2020
3 03-JAN-2020 04-JAN-2020
4 04-JAN-2020 05-JAN-2020
```

Agora o PROJ\_ID 4 está incluído no conjunto de resultados final e apenas o maligno PROJ\_ID 5 foi excluído. Considere seus requisitos exatos ao aplicar essas receitas ao seu código.

## 10.2 Encontrar diferenças entre linhas no mesmo grupo ou partição

### Problema

Você deseja retornar o DEPTNO, ENAME e SAL de cada funcionário junto com a diferença de SAL entre funcionários do mesmo departamento (ou seja, com o mesmo valor para DEPTNO). A diferença deve ser entre cada funcionário atual e o funcionário contratado imediatamente depois (você deseja ver se há uma correlação entre antiguidade e salário em uma base “por departamento”). Para cada funcionário contratado por último em seu departamento, retorne “N/A” para a diferença. O conjunto de resultados deve ficar assim:

DEPTNO	ENAME	SAL	HIREDATE	DIFF
- 10	CLARK	2450	09-JUN-2006	-2550
10	KING	5000	17-NOV-2006	3700
10	MILLER	1300	23-JAN-2007	N/A
20	SMITH	800	17-DEC-2005	-2175
20	JONES	2975	02-APR-2006	-25
20	FORD	3000	03-DEC-2006	0
20	SCOTT	3000	09-DEC-2007	1900
20	ADAMS	1100	12-JAN-2008	N/A
30	ALLEN	1600	20-FEB-2006	350
30	WARD	1250	22-FEB-2006	-1600
30	BLAKE	2850	01-MAY-2006	1350
30	TURNER	1500	08-SEP-2006	250
30	MARTIN	1250	28-SEP-2006	300
30	JAMES	950	03-DEC-2006	N/A

### Solução

Este é outro exemplo de onde as funções de janela LEAD OVER e LAG OVER são úteis. Você pode acessar facilmente as linhas seguintes e anteriores sem junções adicionais. Métodos alternativos, como subconsultas ou autojunções, são possíveis, mas complicados:

```

1 with next_sal_tab (deptno,ename,sal,hiredate,next_sal)
2 as
3 (select deptno, ename, sal, hiredate,
4      lead(sal)over(partition by deptno
5                      order by hiredate) as next_sal
6   from emp )
7
8   select deptno, ename, sal, hiredate
9 ,    coalesce(cast(sal-next_sal as char), 'N/A') as diff
8   from next_sal_tab

```

Nesse caso, para variar, usamos um CTE em vez de uma subconsulta — ambos funcionarão na maioria dos RDBMSs atualmente, com a preferência geralmente relacionada à legibilidade.

## Discussão

O primeiro passo é usar a função de janela LEAD OVER para encontrar o “próximo” salário de cada funcionário dentro de seu departamento. Os funcionários contratados por último em cada departamento terão um valor NULL para NEXT\_SAL:

```
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) as next_sal
  from emp
```

DEPTNO	ENAME	SAL	HIREDATE	NEXT_SAL
10	CLARK	2450	09-JUN-2006	5000
10	KING	5000	17-NOV-2006	1300
10	MILLER	1300	23-JAN-2007	
20	SMITH	800	17-DEC-2005	2975
20	JONES	2975	02-APR-2006	3000
20	FORD	3000	03-DEC-2006	3000
20	SCOTT	3000	09-DEC-2007	1100
20	ADAMS	1100	12-JAN-2008	
30	ALLEN	1600	20-FEB-2006	1250
30	WARD	1250	22-FEB-2006	2850
30	BLAKE	2850	01-MAY-2006	1500
30	TURNER	1500	08-SEP-2006	1250
30	MARTIN	1250	28-SEP-2006	950
30	JAMES	950	03-DEC-2006	

O próximo passo é tirar a diferença entre o salário de cada funcionário e o salário do funcionário contratado imediatamente depois dele no mesmo departamento:

```
select deptno,ename,sal,hiredate, sal-next_sal diff
      from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
      from emp
     )
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-2006	-2550
10	KING	5000	17-NOV-2006	3700
10	MILLER	1300	23-JAN-2007	
20	SMITH	800	17-DEC-2005	-2175
20	JONES	2975	02-APR-2006	-25
20	FORD	3000	03-DEC-2006	0
20	SCOTT	3000	09-DEC-2007	1900
20	ADAMS	1100	12-JAN-2008	
30	ALLEN	1600	20-FEB-2006	350

30 WARD	1250 22-FEB-2006	- 1600
30 BLAKE	2850 01-MAY-2006	1350
30 TURNER	1500 08-SEP-2006	250
30 MARTIN	1250 28-SEP-2006	300
30 JAMES	950 03-DEC-2006	

O próximo passo é utilizar a função COALESCE para inserir “N/A” quando não houver próximo salário. Para poder retornar “N/A” você deve converter o valor de DIFF para uma string:

```
select deptno,ename,sal,hiredate,
       nvl(to_char(sal-next_sal),'N/A') diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp
  )
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-2006	-2550
10	KING	5000	17-NOV-2006	3700
10	MILLER	1300	23-JAN-2007	N/A
20	SMITH	800	17-DEC-2005	-2175
20	JONES	2975	02-APR-2006	-25
20	FORD	3000	03-DEC-2006	0
20	SCOTT	3000	09-DEC-2007	1900
20	ADAMS	1100	12-JAN-2008	N/A
30	ALLEN	1600	20-FEB-2006	350
30	WARD	1250	22-FEB-2006	-1600
30	BLAKE	2850	01-MAY-2006	1350
30	TURNER	1500	08-SEP-2006	250
30	MARTIN	1250	28-SEP-2006	300
30	JAMES	950	03-DEC-2006	N/A

Embora a maioria das soluções fornecidas neste livro não lide com cenários “e se” (para facilitar a leitura e a sanidade do autor), o cenário envolvendo duplicatas ao usar a função LEAD OVER dessa maneira deve ser discutido. Nos dados de amostra simples na tabela EMP, nenhum funcionário tem HIREDATES duplicados, mas essa é uma situação improvável. Normalmente, não discutiríamos uma situação “e se”, como duplicatas (já que não há nenhuma na tabela EMP), mas a solução alternativa envolvendo o LEAD pode não ser imediatamente óbvia. Considere a seguinte consulta, que retorna a diferença de SAL entre os funcionários do DEPTNO 10 (a diferença é realizada na ordem em que foram admitidos):

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                     order by hiredate) next_sal
  from emp
```

```
where deptno=10 and empno > 10
      )
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-2006	-2550
10	KING	5000	17-NOV-2006	3700
10	MILLER	1300	23-JAN-2007	N/A

Esta solução está correta considerando os dados da tabela EMP, mas se houvesse linhas duplicadas, a solução falharia. Considere o exemplo a seguir, que mostra mais quatro funcionários contratados no mesmo dia da KING:

```
insert into emp (empno,ename,deptno,sal,hiredate)
values (1,'ant',10,1000,to_date('17-NOV-2006'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (2,'joe',10,1500,to_date('17-NOV-2006'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (3,'jim',10,1600,to_date('17-NOV-2006'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (4,'jon',10,1700,to_date('17-NOV-2006'))

select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                     order by hiredate) next_sal
  from emp
 where deptno=10
      )
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-2006	1450
10	ant	1000	17-NOV-2006	-500
10	joe	1500	17-NOV-2006	-3500
10	KING	5000	17-NOV-2006	3400
10	jim	1600	17-NOV-2006	-100
10	jon	1700	17-NOV-2006	400
10	MILLER	1300	23-JAN-2007	N/A

Você notará que, com exceção do funcionário JON, todos os funcionários contratados na mesma data (17 de novembro) avaliam seu salário em relação a outro funcionário contratado na mesma data! Isso está incorreto. Todos os funcionários contratados em 17 de novembro devem ter a diferença de salário computada contra o salário da MILLER, e não de outro funcionário contratado em 17 de novembro. Tomemos, por exemplo, o funcionário ANT. O valor de DIFF para ANT é -500 porque o SAL de ANT é comparado com o SAL de JOE e é 500 a menos que o de JOE

SAL, daí o valor de -500. O valor correto de DIFF para o funcionário ANT deve ser -300 porque ANT ganha 300 a menos que MILLER, que é o próximo funcionário contratado por HIREDATE. A razão pela qual a solução parece não funcionar é devido ao comportamento padrão da função LEAD OVER do Oracle. Por padrão, LEAD OVER olha para frente apenas uma linha. Portanto, para o funcionário ANT, o próximo SAL baseado em HIREDATE é o SAL de JOE, porque LEAD OVER simplesmente olha uma linha à frente e não pula duplicatas. Felizmente, a Oracle se planejou para tal situação e permite que você passe um parâmetro adicional para LEAD OVER para determinar o quanto à frente ele deve olhar. No exemplo anterior, a solução é simplesmente uma questão de contagem: encontre a distância de cada funcionário contratado de 17 de novembro a 23 de janeiro (DATA DE ALUGUER de MILLER). O seguinte mostra como fazer isso:

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal,cnt-rn+1)over(partition by deptno
                               order by hiredate) next_sal
  from (
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate order by sal) rn
  from emp
 where deptno=10
      )
      )
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-2006	1450
10	ant	1000	17-NOV-2006	-300
10	joe	1500	17-NOV-2006	200
10	jim	1600	17-NOV-2006	300
10	jon	1700	17-NOV-2006	400
10	KING	5000	17-NOV-2006	3700
10	MILLER	1300	23-JAN-2007	N/A

Agora a solução está correta. Como você pode ver, todos os funcionários contratados em 17 de novembro agora têm seus salários comparados com o salário da MILLER. Verificando os resultados, o funcionário ANT agora tem um valor de -300 para DIFF, que era o que esperávamos. Se não for imediatamente óbvio, a expressão passa para LEAD OVER; CNT-RN+1 é simplesmente a distância de cada funcionário contratado em 17 de novembro até a MILLER. Considere a seguinte exibição em linha, que mostra os valores para CNT e RN:

```
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate order by sal) rn
  from emp
 where deptno=10
```

DEPTNO	ENAME	SAL	HIREDATE	CNT	RN
10	CLARK	2450	09-JUN-2006	1	1
10	ant	1000	17-NOV-2006	5	1
10	joe	1500	17-NOV-2006	5	2
10	jim	1600	17-NOV-2006	5	3
10	jon	1700	17-NOV-2006	5	4
10	KING	5000	17-NOV-2006	5	5
10	MILLER	1300	23-JAN-2007	1	1

O valor para CNT representa, para cada funcionário com HIREDATE duplicado, quantas duplicatas existem no total para o HIREDATE. O valor para RN representa uma classificação para os funcionários em DEPTNO 10. A classificação é particionada por DEPTNO e HIREDATE, portanto, apenas os funcionários com um HIREDATE que outro funcionário possui terão um valor maior que um. A classificação é classificada por SAL (isso é arbitrário; SAL é conveniente, mas poderíamos facilmente ter escolhido EMPNO). Agora que você sabe quantas duplicatas totais existem e tem uma classificação de cada duplicata, a distância até MILLER é simplesmente o número total de duplicatas menos a classificação atual mais um (CNT-RN+1). Os resultados do cálculo da distância e seu efeito no LEAD OVER são mostrados aqui:

```

select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                      order by hiredate) incorrect,
       cnt-rn+1 distance,
       lead(sal,cnt-rn+1)over(partition by deptno
                      order by hiredate) correct
  from (
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate
                         order by sal) rn
  from emp
 where deptno=10
  )

```

DEPTNO	ENAME	SAL	HIREDATE	INCORRECT	DISTANCE	CORRECT
10	CLARK	2450	09-JUN-2006	1000	1	1000
10	ant	1000	17-NOV-2006	1500	5	1300
10	joe	1500	17-NOV-2006	1600	4	1300
10	jim	1600	17-NOV-2006	1700	3	1300
10	jon	1700	17-NOV-2006	5000	2	1300
10	KING	5000	17-NOV-2006	1300	1	1300
10	MILLER	1300	23-JAN-2007		1	

Agora você pode ver claramente o efeito que você tem quando passa a distância correta para LEAD OVER. As linhas para INCORRECT representam os valores retornados por LEAD OVER usando uma distância padrão de um. As linhas para CORRECT representam os valores retornados por LEAD OVER usando a distância adequada para cada funcionário com uma duplicata HIREDATE para MILLER.

Neste ponto, resta apenas encontrar a diferença entre CORRECT e SAL para cada linha, que já foi mostrada.

## 10.3 Localizando o início e o fim de um intervalo de valores consecutivos

### Problema

Esta receita é uma extensão da receita anterior e usa a mesma visualização V da receita anterior. Agora que você localizou os intervalos de valores consecutivos, deseja encontrar apenas seus pontos inicial e final. Ao contrário da receita anterior, se uma linha não fizer parte de um conjunto de valores consecutivos, você ainda deseja retorná-la. Por que? Porque essa linha representa o início e o fim de seu intervalo. Usando os dados da visualização V:

```
select *
from V
```

	PROJ_ID	PROJ_START	PROJ_END
1	1	01-JAN-2020	02-JAN-2020
2	2	02-JAN-2020	03-JAN-2020
3	3	03-JAN-2020	04-JAN-2020
4	4	04-JAN-2020	05-JAN-2020
5	5	06-JAN-2020	07-JAN-2020
6	6	16-JAN-2020	17-JAN-2020
7	7	17-JAN-2020	18-JAN-2020
8	8	18-JAN-2020	19-JAN-2020
9	9	19-JAN-2020	20-JAN-2020
10	10	21-JAN-2020	22-JAN-2020
11	11	26-JAN-2020	27-JAN-2020
12	12	27-JAN-2020	28-JAN-2020
13	13	28-JAN-2020	29-JAN-2020
14	14	29-JAN-2020	30-JAN-2020

você deseja que o conjunto de resultados final seja o seguinte:

	PROJ_GRP	PROJ_START	PROJ_END
1	1	01-JAN-2020	05-JAN-2020
2	2	06-JAN-2020	07-JAN-2020
3	3	16-JAN-2020	20-JAN-2020
4	4	21-JAN-2020	22-JAN-2020
5	5	26-JAN-2020	30-JAN-2020

Este problema é um pouco mais complicado do que seu antecessor. Primeiro, você deve identificar quais são os intervalos. Um intervalo de linhas é definido pelos valores de PROJ\_START e PROJ\_END. Para que uma linha seja considerada “consecutiva” ou parte de um grupo, seu valor PROJ\_START deve ser igual ao valor PROJ\_END da linha anterior. No caso em que o valor PROJ\_START de uma linha não é igual ao valor PROJ\_END da linha anterior e seu valor PROJ\_END não é igual ao valor PROJ\_START da próxima linha, esta é uma instância de um único grupo de linhas. Depois de identificar os intervalos, você precisa ser capaz de agrupar as linhas nesses intervalos (em grupos) e retornar apenas seus pontos inicial e final.

Examine a primeira linha do conjunto de resultados desejado. O PROJ\_START é o PROJ\_START para PROJ\_ID 1 da visualização V e o PROJ\_END é o PROJ\_END para PROJ\_ID 4 da visualização V. Apesar de PROJ\_ID 4 não ter um valor consecutivo a seguir, é o último de um intervalo de valores consecutivos, e assim está incluído no primeiro grupo.

A abordagem mais direta para esse problema é usar a função de janela LAG OVER. Use LAG OVER para determinar se o PROJ\_END de cada linha anterior é igual ao PROJ\_START da linha atual para ajudar a colocar as linhas em grupos. Depois de agrupados, use as funções de agregação MIN e MAX para encontrar seus pontos inicial e final:

```
1 select proj_grp, min(proj_start), max(proj_end)
2   from (
3 select proj_id,proj_start,proj_end,
4       sum(flag)over(order by proj_id) proj_grp
5   from (
6 select proj_id,proj_start,proj_end,
7       case when
8           lag(proj_end)over(order by proj_id) = proj_start
9           then 0 else 1
10      end flag
11  from V
12      ) alias1
13      ) alias2
14 group by proj_grp
```

## Discussão

A função window LAG OVER é extremamente útil nesta situação. Você pode examinar o valor PROJ\_END de cada linha anterior sem uma autojunção, sem uma subconsulta escalar e sem uma visualização. Os resultados da função LAG OVER sem a expressão CASE são os seguintes:

```
select proj_id,proj_start,proj_end,
       lag(proj_end)over(order by proj_id) prior_proj_end
  from V
```

PROJ_ID	PROJ_START	PROJ_END	PRIOR_PROJ_END
1	01-JAN-2020	02-JAN-2020	
2	02-JAN-2020	03-JAN-2020	02-JAN-2020
3	03-JAN-2020	04-JAN-2020	03-JAN-2020
4	04-JAN-2020	05-JAN-2020	04-JAN-2020
5	06-JAN-2020	07-JAN-2020	05-JAN-2020
6	16-JAN-2020	17-JAN-2020	07-JAN-2020
7	17-JAN-2020	18-JAN-2020	17-JAN-2020
8	18-JAN-2020	19-JAN-2020	18-JAN-2020
9	19-JAN-2020	20-JAN-2020	19-JAN-2020
10	21-JAN-2020	22-JAN-2020	20-JAN-2020
11	26-JAN-2020	27-JAN-2020	22-JAN-2020
12	27-JAN-2020	28-JAN-2020	27-JAN-2020
13	28-JAN-2020	29-JAN-2020	28-JAN-2020
14	29-JAN-2020	30-JAN-2020	29-JAN-2020

A expressão CASE na solução completa simplesmente compara o valor retornado por LAG OVER com o valor PROJ\_START da linha atual; se forem iguais, retorne 0, caso contrário, retorne 1. A próxima etapa é criar um total acumulado nos zeros e uns retornados pela expressão CASE para colocar cada linha em um grupo. Os resultados do total acumulado são mostrados aqui:

```
select proj_id,proj_start,proj_end,
       sum(flag)over(order by proj_id) proj_grp
  from (
select proj_id,proj_start,proj_end,
       case when
              lag(proj_end)over(order by proj_id) = proj_start
              then 0 else 1
         end flag
  from V
      )
```

PROJ_ID	PROJ_START	PROJ_END	PROJ_GRP
1	01-JAN-2020	02-JAN-2020	1
2	02-JAN-2020	03-JAN-2020	1
3	03-JAN-2020	04-JAN-2020	1
4	04-JAN-2020	05-JAN-2020	1
5	06-JAN-2020	07-JAN-2020	2
6	16-JAN-2020	17-JAN-2020	3
7	17-JAN-2020	18-JAN-2020	3
8	18-JAN-2020	19-JAN-2020	3
9	19-JAN-2020	20-JAN-2020	3
10	21-JAN-2020	22-JAN-2020	4

11	26-JAN-2020	27-JAN-2020	5
12	27-JAN-2020	28-JAN-2020	5
13	28-JAN-2020	29-JAN-2020	5
14	29-JAN-2020	30-JAN-2020	5

Agora que cada linha foi colocada em um grupo, simplesmente use as funções agregadas MIN e MAX em PROJ\_START e PROJ\_END, respectivamente, e agrupe pelos valores criados na coluna PROJ\_GRP running total.

## 10.4 Preenchendo valores ausentes em um intervalo de valores

### Problema

Você deseja retornar o número de funcionários contratados a cada ano durante toda a década de 2005, mas há alguns anos em que nenhum funcionário foi contratado. Você gostaria de retornar o seguinte conjunto de resultados:

YR	CNT
2005	1
2006	10
2007	2
2008	1
2009	0
2010	0
2011	0
2012	0
2013	0
2014	0

### Solução

O truque dessa solução é retornar zeros por anos sem funcionários contratados. Se nenhum funcionário foi contratado em um determinado ano, não haverá linhas para esse ano na tabela EMP. Se o ano não existe na tabela, como você pode retornar uma contagem, qualquer contagem, mesmo zero? A solução exige que você faça uma junção externa. Você deve fornecer um conjunto de resultados que retorne todos os anos que deseja ver e, em seguida, realizar uma contagem na tabela EMP para ver se houve funcionários contratados em cada um desses anos.

### DB2

Use a tabela EMP como uma tabela dinâmica (porque ela tem 14 linhas) e a função interna YEAR para gerar uma linha para cada ano na década de 2005. Junte-se externamente à tabela EMP e conte quantos funcionários foram contratados a cada ano:

```

1 select x.yr, coalesce(y.cnt,0) cnt
2 from (
3 select year(min(hiredate)over()) -
4 mod(year(min(hiredate)over()),10) +

```

```

1      row_number()over()-1 yr
2  from emp fetch first 10 rows
only 7      )
8  left join
9      (
10 select year(hiredate) yr1, count(*) cnt
11  from emp
12 group by year(hiredate)
13      ) y
14  on ( x.yr = y.yr1 )

```

## Oracle

A solução Oracle segue a mesma estrutura da solução DB2, com apenas as diferenças na sintaxe que o Oracle trata fazendo com que seja necessária uma solução distinta:

```

1 select x.yr, coalesce(cnt,0) cnt
2   from (
3 select extract(year from min(hiredate)over()) -
4       mod(extract(year from min(hiredate)over()),10) +
5           rownum-1 yr
6   from emp
7 where rownum <= 10
8 ) x
9  left join
10  (
11 select to_number(to_char(hiredate,'YYYY')) yr, count(*) cnt
12  from emp
13 group by to_number(to_char(hiredate,'YYYY'))
14      ) y
15  on ( x.yr = y.yr )

```

## PostgreSQL e MySQL

Use a tabela T10 como uma tabela dinâmica (porque ela tem 10 linhas) e a função integrada EXTRACT para gerar uma linha para cada ano na década de 2005. Junte-se externamente à tabela EMP e conte quantos funcionários foram contratados a cada ano:

```

1 select y.yr, coalesce(x.cnt,0) as cnt
2   from (
3 select min_year-mod(cast(min_year as int),10)+rn as yr
4   from (
5 select (select min(extract(year from hiredate))
6         from emp) as min_year,
7             id-1 as rn
8   from t10
9 ) a
10 ) y
11  left join
12  (
13 select extract(year from hiredate) as yr, count(*) as cnt
14  from emp

```

```

14 group by extract(year from hiredate)
15 ) x
16 on ( y.yr = x.yr )

```

## SQL Server

Use a tabela EMP como uma tabela dinâmica (porque ela tem 14 linhas) e a função interna YEAR para gerar uma linha para cada ano na década de 2005. Junte-se externamente à tabela EMP e conte quantos funcionários foram contratados a cada ano:

```

1 select x.yr, coalesce(y.cnt,0) cnt
2   from (
3 select top (10)
4       (year(min(hiredate)over()) -
5        year(min(hiredate)over())%10) +
6        row_number()over(order by hiredate)-1 yr
7   from emp
8     ) x
9 left join
10  (
11 select year(hiredate) yr, count(*) cnt
12   from emp
13  group by year(hiredate)
14    ) y
15  on ( x.yr = y.yr )

```

## Discussão

Apesar da diferença de sintaxe, a abordagem é a mesma para todas as soluções. A exibição em linha X retorna a cada ano na década de 80, encontrando primeiro o ano do HIREDATE mais antigo. O próximo passo é adicionar RN-1 à diferença entre o primeiro ano e o primeiro ano do módulo dez. Para ver como isso funciona, simplesmente execute a visualização inline X e retorne cada um dos valores envolvidos separadamente. Aqui está listado o conjunto de resultados para visualização inline X usando a função de janela MIN OVER (DB2, Oracle, SQL Server) e uma subconsulta escalar (MySQL, PostgreSQL):

```

select year(min(hiredate)over()) -
       mod(year(min(hiredate)over()),10) +
       row_number()over()-1 yr,
       year(min(hiredate)over()) min_year,
       mod(year(min(hiredate)over()),10) mod_yr,
       row_number()over()-1 rn
  from emp fetch first 10 rows only

```

YR	MIN_YEAR	MOD_YR	RN
2005	2005	0	0
2006	2005	0	1
2007	2005	0	2
2008	2005	0	3
1984	2005	0	4

2010	2005	0	5
2011	2005	0	6
2012	2005	0	7
2013	2005	0	8
2014	2005	0	9

```

select min_year-mod(min_year,10)+rn as yr,
       min_year,
       mod(min_year,10) as mod_yr
       rn
  from (
select (select min(extract(year from hiredate))
           from emp) as min_year,
           id-1 as rn
  from t10
 ) x

```

YR	MIN_YEAR	MOD_YR	RN
2005	2005	0	0
2006	2005	0	1
2007	2005	0	2
2008	2005	0	3
2009	2005	0	4
2010	2005	0	5
2011	2005	0	6
2012	2005	0	7
2013	2005	0	8
2014	2005	0	9

A exibição embutida Y retorna o ano para cada HIREDATE e o número de funcionários contratados durante esse ano:

```

select year(hiredate) yr, count(*) cnt
  from emp
 group by year(hiredate)

```

YR	CNT
2005	1
2006	10
2007	2
2008	1

Por fim, junte externamente a exibição em linha Y à exibição em linha X para que todos os anos sejam retornados, mesmo que não haja funcionários contratados.

# 10.5 Gerando Valores Numéricos Consecutivos

## Problema

Você gostaria de ter um “gerador de fonte de linha” disponível para você em suas consultas. Os geradores de origem de linha são úteis para consultas que exigem dinamização. Por exemplo, você deseja retornar um conjunto de resultados como o seguinte, até qualquer número de linhas que você especificar:

```
ID  
---  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
...  
..
```

Se seu RDBMS fornecer funções integradas para retornar linhas dinamicamente, você não precisará criar uma tabela dinâmica antecipadamente com um número fixo de linhas. É por isso que um gerador dinâmico de linhas pode ser tão útil. Caso contrário, você deve usar uma tabela dinâmica tradicional com um número fixo de linhas (que nem sempre é suficiente) para gerar linhas quando necessário.

## Solução

Esta solução mostra como retornar 10 linhas de números crescentes começando em 1. Você pode facilmente adaptar a solução para retornar qualquer número de linhas.

A capacidade de retornar valores crescentes abre as portas para muitas outras soluções. Por exemplo, você pode gerar números para adicionar a datas para gerar sequências de dias. Você também pode usar esses números para analisar strings.

### DB2 e SQLServer

Use a cláusula recursiva WITH para gerar uma sequência de linhas com valores crescentes. O uso de um CTE recursivo funcionará de fato com a maioria dos RDBMSs hoje:

```
1 with x (id)  
2 as (  
3 select 1  
4 union all  
5 select id+1  
6 from x  
7 where id+1 <= 10
```

```

8 )
9 select * from x

```

## Oracle

No banco de dados Oracle você pode gerar linhas usando a cláusula MODEL:

```

1 select array id
2   from dual
3 model
4   dimension by (0 idx)
5   measures(1 array)
6   rules iterate (10) (
7     array[iteration_number] = iteration_number+1
8   )

```

## PostgreSQL

Use a útil função GENERATE\_SERIES, que é projetada para o propósito expresso de gerar linhas:

```

1 select id
2   from generate_series (1, 10) x(id)

```

## Discussão

### DB2 e SQLServer

A cláusula recursiva WITH incrementa ID (que começa em um) até que a cláusula WHERE seja satisfeita. Para começar, você deve gerar uma linha com o valor 1. Você pode fazer isso selecionando 1 em uma tabela de uma linha ou, no caso do DB2, usando a cláusula VALUES para criar um conjunto de resultados de uma linha.

## Oracle

Na cláusula MODEL solução, há um comando explícito ITERATE que permite gerar várias linhas. Sem a cláusula ITERATE, apenas uma linha será retornada, pois DUAL possui apenas uma linha. Por exemplo:

```

select array id
      from dual
model
  dimension by (0 idx)
  measures(1 array)
  rules ()

```

ID
--
1

A cláusula MODEL não apenas permite acesso de array às linhas, mas permite que você facilmente “crie” ou retorne linhas que não estão na tabela que você está selecionando. Nesta solução, IDX é o índice do array (localização de um valor específico no array) e ARRAY (aliased ID) é o “array” de linhas. O padrão da primeira linha é 1 e pode ser referenciado com ARRAY[0]. A Oracle fornece a função ITERATION\_NUMBER para que você possa rastrear o número de vezes que iterou. A solução itera 10 vezes, fazendo com que ITERATION\_NUMBER vá de 0 a 9. Adicionar um a cada um desses valores produz os resultados de 1 a 10.

Pode ser mais fácil visualizar o que está acontecendo com a cláusula do modelo se você executar a seguinte consulta:

```
select 'array['||idx||'] ='||array as output
  from dual
model
  dimension by (0 idx)
  measures(1 array)
  rules iterate (10) (
    array[iteration_number] = iteration_number+1
  )

OUTPUT
-----
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
array[5] = 6
array[6] = 7
array[7] = 8
array[8] = 9
array[9] = 10
```

## PostgreSQL

Todo o trabalho é feito pela função GENERATE\_SERIES. A função aceita três parâmetros, todos valores numéricos. O primeiro parâmetro é o valor inicial, o segundo parâmetro é o valor final e o terceiro parâmetro é um valor de “etapa” opcional (quanto cada valor é incrementado). Se você não passar um terceiro parâmetro, o incremento é padronizado para um.

A função GENERATE\_SERIES é flexível o suficiente para que você não precise codificar parâmetros. Por exemplo, se você quiser retornar 5 linhas começando no valor 10 e terminando no valor 30, incrementando em 5 de forma que o conjunto de resultados seja o seguinte:

```
ID
---
10
15
20
25
30
```

você pode ser criativo e fazer algo assim:

```
select id
  from generate_series(
    (select min(deptno) from emp),
    (select max(deptno) from emp),
    5
  ) x(id)
```

Observe aqui que os valores reais passados para GENERATE\_SERIES não são conhecidos quando a consulta é escrita. Em vez disso, eles são gerados por subconsultas quando a consulta principal é executada.

## 10.6 Resumindo

As consultas que levam em conta intervalos são uma das solicitações mais comuns dos usuários corporativos — são uma consequência natural da maneira como as empresas operam. Pelo menos algumas vezes, no entanto, um certo grau de destreza é necessário para aplicar o alcance corretamente, e as receitas neste capítulo devem demonstrar como aplicar essa destreza.

# CAPÍTULO 11

## Pesquisa Avançada

Num sentido muito real, todo este livro até agora foi sobre pesquisa. Você já viu todos os tipos de consultas que usam junções e cláusulas WHERE e técnicas de agrupamento para pesquisar e retornar os resultados necessários. Alguns tipos de operações de busca, entretanto, se diferenciam de outras porque representam uma maneira diferente de pensar sobre a busca. Talvez você esteja exibindo um conjunto de resultados uma página por vez. Metade desse problema é identificar (procurar) todo o conjunto de registros que você deseja exibir. A outra metade desse problema é procurar repetidamente a próxima página a ser exibida enquanto o usuário percorre os registros em uma tela. Seu primeiro pensamento pode não ser pensar na paginação como um problema de pesquisa, mas *pode* ser pensado dessa forma e possa ser resolvido dessa forma; esse é o tipo de solução de pesquisa de que trata este capítulo.

### 11.1 Paginando através de um conjunto de resultados

#### Problema

Você deseja paginar ou “rolar” um conjunto de resultados. Por exemplo, você deseja retornar os primeiros cinco salários da tabela EMP, depois os próximos cinco e assim por diante. Seu objetivo é permitir que um usuário visualize cinco registros por vez, rolando para frente a cada clique de um botão *Próximo*.

#### Solução

Como não existe o conceito de primeiro, último ou próximo no SQL, você deve impor ordem nas linhas com as quais está trabalhando. Somente impondo a ordem você poderá retornar intervalos de registros com precisão.

Use a função de window ROW\_NUMBER OVER para impor ordem e especifique a janela de registros que você deseja retornar em sua cláusula WHERE. Por exemplo, use isto para retornar as linhas 1 a 5:

```
select sal
  from (
select row_number() over (order by sal) as rn,
      sal
  from emp
    ) x
where rn between 1 and 5
```

SAL
-----
800
950
1100
1250
1250

Em seguida, use isso para retornar as linhas 6 a 10:

```
select sal
  from (
select row_number() over (order by sal) as rn,
      sal
  from emp
    ) x
where rn between 6 and 10
```

SAL
-----
1300
1500
1600
2450
2850

Você pode retornar qualquer intervalo de linhas desejado simplesmente alterando a cláusula WHERE da sua consulta.

## Discussão

A função de window ROW\_NUMBER OVER na visualização inline X atribuirá um número exclusivo a cada salário (em ordem crescente, começando em 1). Listado aqui está o conjunto de resultados para a visualização in-line X:

```
select row_number() over (order by sal) as rn,
      sal
  from emp
```

RN	SAL
1	800
2	950
3	1100
4	1250
5	1250
6	1300
7	1500
8	1600
9	2450
10	2850
11	2975
12	3000
13	3000
14	5000

Depois que um número for atribuído a um salário, basta escolher a faixa que deseja retornar especificando valores para RN.

Para usuários Oracle, uma alternativa: você pode usar ROWNUM em vez de ROW NUMBER OVER para gerar números de sequência para as linhas:

```
select sal
  from (
select sal, rownum rn
  from (
select sal
  from emp
 order by sal
  )
  )
 where rn between 6 and 10
```

SAL
1300
1500
1600
2450
2850

Usar ROWNUM força você a escrever um nível extra de subconsulta. A subconsulta mais interna classifica as linhas por salário. A próxima subconsulta mais externa aplica números de linha a essas linhas e, finalmente, o SELECT mais externo retorna os dados que você procura.

## 11.2 Ignorando n linhas de uma tabela

### Problema

Você deseja que uma consulta retorne todos os outros funcionários da tabela EMP; você quer o primeiro funcionário, o terceiro funcionário e assim por diante. Por exemplo, do seguinte conjunto de resultados:

```
ENAME
-----
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD
```

você deseja retornar o seguinte:

```
ENAME
-----
ADAMS
BLAKE
FORD
JONES
MARTIN
SCOTT
TURNER
```

### Solução

Para pular o segundo ou quarto ou  $n^{\text{a}}$  linha de um conjunto de resultados, você deve impor ordem no conjunto de resultados; caso contrário, não existe conceito de primeiro ou próximo, segundo ou quarto.

Use a função de window ROW\_NUMBER OVER para atribuir um número a cada linha, que você pode usar em conjunto com a função módulo para pular linhas indesejadas. A função do módulo é MOD para DB2, MySQL, PostgreSQL e Oracle. No SQL Server, use o operador percentual (%). O exemplo a seguir usa MOD para pular linhas pares:

```
1 select ename
2 from (
3 select row_number() over (order by ename) rn,
4 ename
```

```

5   from emp
6   ) x
7   where mod(rn,2) = 1

```

## Discussão

A chamada para a função de window ROW\_NUMBER OVER na visualização inline X atribuirá uma classificação a cada linha (sem empates, mesmo com nomes duplicados). Os resultados são mostrados aqui:

```

select row_number() over (order by ename) rn, ename
  from emp

RN ENAME
-----
1 ADAMS
2 ALLEN
3 BLAKE
4 CLARK
5 FORD
6 JAMES
7 JONES
8 KING
9 MARTIN
10 MILLER
11 SCOTT
12 SMITH
13 TURNER
14 WARD

```

A última etapa é simplesmente usar o módulo para pular todas as outras linhas.

## 11.3 Incorporando lógica OR ao usar junções externas

### Problema

Você deseja retornar o nome e as informações do departamento de todos os funcionários dos departamentos 10 e 20, juntamente com as informações dos departamentos 30 e 40 (mas nenhuma informação do funcionário). Sua primeira tentativa é assim:

```

select e.ename, d.deptno, d.dname, d.loc
  from dept d, emp e
 where d.deptno = e.deptno
   and (e.deptno = 10 or e.deptno = 20)
 order by 2

```

ENAME	DEPTNO	DNAME	LOC
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS

ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS

Como a junção nesta consulta é uma junção interna, o conjunto de resultados não inclui informações de departamento para DEPTNOs 30 e 40.

Você tenta associar EMP externamente ao DEPT com a seguinte consulta, mas ainda não obtém os resultados corretos:

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d left join emp e
    on (d.deptno = e.deptno)
   where e.deptno = 10
     or e.deptno = 20
  order by 2
```

ENAME	DEPTNO	DNAME	LOC
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS

Em última análise, você gostaria que o conjunto de resultados fosse o seguinte:

ENAME	DEPTNO	DNAME	LOC
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

## Solução

Mova a condição OR para a cláusula JOIN:

```
1 select e.ename, d.deptno, d.dname, d.loc
2 from dept d left join emp e
3 on (d.deptno = e.deptno
4 and (e.deptno=10 or e.deptno=20))
5 order by 2
```

Como alternativa, você pode filtrar EMP.DEPTNO primeiro em uma visualização inline e depois na junção externa:

```

1 select e.ename, d.deptno, d.dname, d.loc
2   from dept d
3 left join
4       (select ename, deptno
5        from emp
6       where deptno in ( 10, 20 )
7      ) e on ( e.deptno = d.deptno )
8 order by 2

```

## Discussão

### DB2, MySQL, PostgreSQL e SQL Server

Duas soluções são fornecidas para esses produtos. A primeira move a condição OR para a cláusula JOIN, tornando-a parte da condição de junção. Ao fazer isso, você pode filtrar as linhas retornadas do EMP sem perder os DEPTNOs 30 e 40 do DEPT.

A segunda solução move a filtragem para uma visualização embutida. A visualização embutida E filtra EMP.DEPTNO e retorna linhas EMP de interesse. Estes são então unidos externamente ao DEPT. Como DEPT é a tabela âncora na junção externa, todos os departamentos, inclusive 30 e 40, são retornados.

## 11.4 Determinando quais linhas são recíprocas

### Problema

Você tem uma tabela contendo os resultados de dois testes e deseja determinar quais pares de pontuações são recíprocos. Considere o seguinte conjunto de resultados da visão V:

```

select *
from V

TEST1      TEST2
-----
 20          20
 50          25
 20          20
 60          30
 70          90
 80         130
 90          70
100          50
110          55
120          60
130          80
140          70

```

Examinando esses resultados, você vê que uma pontuação de teste para TEST1 de 70 e TEST2 de 90 é recíproca (existe uma pontuação de 90 para TEST1 e uma pontuação de 70 para TEST2). Da mesma forma, as pontuações de 80 para TEST1 e 130 para TEST2 são recíprocas de 130 para TEST1 e 80 para TEST2. Além disso, as pontuações de 20 para TEST1 e 20 para TEST2 são recíprocas de 20 para TEST2 e 20 para TEST1. Você deseja identificar apenas um conjunto de recíprocos. Você deseja que seu conjunto de resultados seja este:

TEST1	TEST2
20	20
70	90
80	130

isso não:

TEST1	TEST2
20	20
20	20
70	90
80	130
90	70
130	80

## Solução

Use uma autojunção para identificar linhas onde TEST1 é igual a TEST2 e vice-versa:

```
select distinct v1.*  
  from V v1, V v2  
 where v1.test1 = v2.test2  
   and v1.test2 = v2.test1  
   and v1.test1 <= v1.test2
```

## Discussão

A auto-junção resulta em um produto cartesiano no qual cada pontuação TEST1 pode ser comparada com cada pontuação TEST2 e vice-versa. A consulta a seguir identificará os recíprocos:

```
select v1.*  
  from V v1, V v2  
 where v1.test1 = v2.test2  
   and v1.test2 = v2.test1
```

TEST1	TEST2
20	20
20	20
20	20
20	20

90	70
130	80
70	90
80	130

O uso de DISTINCT garante que as linhas duplicadas sejam removidas do conjunto de resultados final. O filtro final na cláusula WHERE (e V1.TEST1 <= V1.TEST2) garantirá que apenas um par de recíprocos (onde TEST1 é o valor menor ou igual) seja retornado.

## 11.5 Selecionando os n principais registros

### Problema

Você deseja limitar um conjunto de resultados a um número específico de registros com base em algum tipo de classificação. Por exemplo, você deseja retornar os nomes e salários dos funcionários com os cinco maiores salários.

### Solução

A solução para este problema depende do uso de uma função de window. Qual função de window você usará depende de como você deseja lidar com os vínculos. A solução a seguir usa DENSE\_RANK para que cada empate salarial conte como apenas um no total:

```

1 select ename,sal
2   from (
3 select ename, sal,
4       dense_rank() over (order by sal desc) dr
5   from emp
6      ) x
7 where dr <= 5

```

O número total de linhas retornadas pode exceder cinco, mas haverá apenas cinco salários distintos. Use ROW\_NUMBER OVER se desejar retornar cinco linhas independentemente dos empates (já que nenhum empate é permitido com esta função).

### Discussão

A função de window DENSE\_RANK OVER na visualização embutida X faz todo o trabalho. O exemplo a seguir mostra a tabela inteira após aplicar essa função:

```

select ename, sal,
       dense_rank() over (order by sal desc) dr
  from emp

```

ENAME	SAL	DR
KING	5000	1

SCOTT	3000	2
FORD	3000	2
JONES	2975	3
BLAKE	2850	4
CLARK	2450	5
ALLEN	1600	6
TURNER	1500	7
MILLER	1300	8
WARD	1250	9
MARTIN	1250	9
ADAMS	1100	10
JAMES	950	11
SMITH	800	12

Agora é apenas uma questão de retornar linhas onde DR é menor ou igual a cinco.

## 11.6 Encontrando registros com os valores mais altos e mais baixos

### Problema

Você deseja encontrar valores “extremos” em sua tabela. Por exemplo, você deseja encontrar os funcionários com os salários mais altos e mais baixos na tabela EMP.

### Solução

#### DB2, Oracle e SQL Server

Use as funções de window MIN OVER e MAX OVER para encontrar os salários mais baixos e mais altos, respectivamente:

```
1 select ename
2   from (
3 select ename, sal,
4       min(sal)over() min_sal,
5       max(sal)over() max_sal
6   from emp
7      ) x
8 where sal in (min_sal,max_sal)
```

### Discussão

#### DB2, Oracle e SQL Server

As funções de window MIN OVER e MAX OVER permitem que cada linha tenha acesso aos salários mais baixos e mais altos. O conjunto de resultados da visualização embutida X é o seguinte:

```
select ename, sal,
       min(sal)over() min_sal,
       max(sal)over() max_sal
  from emp
```

ENAME	SAL	MIN_SAL	MAX_SAL
SMITH	800	800	5000
ALLEN	1600	800	5000
WARD	1250	800	5000
JONES	2975	800	5000
MARTIN	1250	800	5000
BLAKE	2850	800	5000
CLARK	2450	800	5000
SCOTT	3000	800	5000
KING	5000	800	5000
TURNER	1500	800	5000
ADAMS	1100	800	5000
JAMES	950	800	5000
FORD	3000	800	5000
MILLER	1300	800	5000

Dado este conjunto de resultados, tudo o que resta é retornar linhas onde SAL é igual a MIN\_SAL ou MAX\_SAL.

## 11.7 Investigando Linhas Futuras

### Problema

Você deseja encontrar funcionários que ganhem menos do que o funcionário contratado imediatamente depois deles. Com base no seguinte conjunto de resultados:

ENAME	SAL	HIREDATE
SMITH	800	17-DEC-80
ALLEN	1600	20-FEB-81
WARD	1250	22-FEB-81
JONES	2975	02-APR-81
BLAKE	2850	01-MAY-81
CLARK	2450	09-JUN-81
TURNER	1500	08-SEP-81
MARTIN	1250	28-SEP-81
KING	5000	17-NOV-81
JAMES	950	03-DEC-81
FORD	3000	03-DEC-81
MILLER	1300	23-JAN-82
SCOTT	3000	09-DEC-82
ADAMS	1100	12-JAN-83

SMITH, WARD, MARTIN, JAMES e MILLER ganham menos do que a pessoa contratada imediatamente após sua contratação, portanto, esses são os funcionários que você deseja encontrar com uma consulta.

O primeiro passo é definir o que significa “futuro”. Você deve impor ordem em seu conjunto de resultados para poder definir uma linha como tendo um valor “posterior” que outro.

Você pode utilizar a função de window LEAD OVER para acessar o salário do próximo funcionário contratado. É então simples verificar se esse salário é maior:

```
1 select ename, sal, hiredate
2   from (
3 select ename, sal, hiredate,
4       lead(sal)over(order by hiredate) next_sal
5   from emp
6      ) alias
7 where sal < next_sal
```

## Discussão

A função de window LEAD OVER é perfeita para um problema como este. Ele não apenas torna a consulta mais legível do que a solução para os outros produtos, mas LEAD OVER também leva a uma solução mais flexível porque um argumento pode ser passado para ele que determinará quantas linhas à frente ele deve procurar (por padrão, uma). Ser capaz de avançar mais de uma linha é importante no caso de duplicatas na coluna pela qual você está ordenando.

O exemplo a seguir mostra como é fácil usar o LEAD OVER para ver o salário do “próximo” funcionário contratado:

```
select ename, sal, hiredate,
       lead(sal)over(order by hiredate) next_sal
  from emp
```

ENAME	SAL	HIREDATE	NEXT_SAL
SMITH	800	17-DEC-80	1600
ALLEN	1600	20-FEB-81	1250
WARD	1250	22-FEB-81	2975
JONES	2975	02-APR-81	2850
BLAKE	2850	01-MAY-81	2450
CLARK	2450	09-JUN-81	1500
TURNER	1500	08-SEP-81	1250
MARTIN	1250	28-SEP-81	5000
KING	5000	17-NOV-81	950
JAMES	950	03-DEC-81	3000
FORD	3000	03-DEC-81	1300
MILLER	1300	23-JAN-82	3000
SCOTT	3000	09-DEC-82	1100
ADAMS	1100	12-JAN-83	

A etapa final é retornar apenas as linhas onde SAL é menor que NEXT\_SAL. Devido ao intervalo padrão de uma linha do LEAD OVER, se houvesse duplicatas na tabela EMP – em particular, vários funcionários contratados na mesma data – seu SAL seria comparado. Isso pode ou não ter sido o que você pretendia. Se o seu objetivo é comparar o SAL de cada funcionário com o SAL do próximo funcionário contratado, excluindo os demais funcionários contratados no mesmo dia, você pode utilizar a seguinte solução como alternativa:

```
select ename, sal, hiredate
  from (
    select ename, sal, hiredate,
           lead(sal,cnt-rn+1)over(order by hiredate) next_sal
      from (
        select ename,sal,hiredate,
               count(*)over(partition by hiredate) cnt,
               row_number()over(partition by hiredate order by empno) rn
          from emp
        )
      )
     where sal < next_sal
```

A ideia por trás desta solução é encontrar a distância da linha atual até a linha com a qual ela deve ser comparada. Por exemplo, se houver cinco duplicatas, a primeira das cinco precisa pular cinco linhas para chegar à linha LEAD OVER correta. O valor para CNT representa, para cada funcionário com HIREDATE duplicado, quantas duplicatas existem no total para o HIREDATE. O valor para RN representa uma classificação para os funcionários no DEPTNO 10. A classificação é particionada por HIREDATE, portanto, apenas os funcionários com HIREDATE que outro funcionário possui terão um valor maior que um. A classificação é ordenada por EMPNO (isto é arbitrário). Agora que você sabe quantos duplicados existem e tem uma classificação de cada duplicado, a distância até o próximo HIREDATE é simplesmente o número total de duplicados menos a classificação atual mais um (CNT-RN+1).

## Veja também

Para exemplos adicionais de uso do LEAD OVER na presença de duplicatas (e uma discussão mais aprofundada desta técnica), consulte [Receita 8.7](#) e [Receita 10.2](#).

## 11.8 Mudando valores de linha

### Problema

Você deseja retornar o nome e o salário de cada funcionário junto com os próximos salários mais altos e mais baixos. Se não houver salários maiores ou menores, você deseja que os resultados sejam agrupados (o primeiro SAL mostra o último SAL e vice-versa). Você deseja retornar o seguinte conjunto de resultados:

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	5000
JAMES	950	1100	800
ADAMS	1100	1250	950
WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000	800	3000

## Solução

As funções de window LEAD OVER e LAG OVER tornam este problema fácil de resolver e as consultas resultantes muito legíveis. Use as funções de window LAG OVER e LEAD OVER para acessar as linhas anteriores e seguintes relativas à linha atual:

```

1 select ename,sal,
2      coalesce(lead(sal)over(order by sal),min(sal)over()) forward,
3      coalesce(lag(sal)over(order by sal),max(sal)over()) rewind
4 from emp

```

## Discussão

As funções de window LAG OVER e LEAD OVER retornarão (por padrão e a menos que especificado de outra forma) valores da linha antes e depois da linha atual, respectivamente. Você define o que “antes” ou “depois” significa na parte ORDER BY da cláusula OVER. Se você examinar a solução, o primeiro passo é retornar as linhas seguintes e anteriores relativas à linha atual, ordenadas por SAL:

```

select ename,sal,
       lead(sal)over(order by sal) forward,
       lag(sal)over(order by sal) rewind
  from emp

```

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	
JAMES	950	1100	800
ADAMS	1100	1250	950
WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300

ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000		3000

Observe que REWIND é NULL para o funcionário SMITH e FORWARD é NULL para o funcionário KING; isso ocorre porque esses dois funcionários têm os salários mais baixos e mais altos, respectivamente. O requisito na seção “Problema” caso existam valores NULL em FORWARD ou REWIND é “envolver” os resultados, o que significa que para o SAL mais alto, FORWARD deve ser o valor do SAL mais baixo na tabela, e para o SAL mais baixo, REWIND deve ser o valor do maior SAL da tabela. As funções de janela MIN OVER e MAX OVER sem partição ou janela especificada (ou seja, parênteses vazios após a cláusula OVER) retornarão os salários mais baixos e mais altos da tabela, respectivamente. Os resultados são mostrados aqui:

```
select ename,sal,
       coalesce(lead(sal)over(order by sal),min(sal)over()) forward,
       coalesce(lag(sal)over(order by sal),max(sal)over()) rewind
  from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	5000
JAMES	950	1100	800
ADAMS	1100	1250	950
WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000	800	3000

Outro recurso útil do LAG OVER e LEAD OVER é a capacidade de definir o quanto você gostaria de avançar ou retroceder. No exemplo desta receita, você avança ou retrocede apenas uma linha. Se quiser mover três linhas para frente e cinco linhas para trás, isso é simples. Basta especificar os valores 3 e 5, conforme mostrado aqui:

```
select ename,sal,
       lead(sal,3)over(order by sal) forward,
       lag(sal,5)over(order by sal) rewind
  from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	1250	
JAMES	950	1250	
ADAMS	1100	1300	
WARD	1250	1500	
MARTIN	1250	1600	
MILLER	1300	2450	800
TURNER	1500	2850	950
ALLEN	1600	2975	1100
CLARK	2450	3000	1250
BLAKE	2850	3000	1250
JONES	2975	5000	1300
SCOTT	3000		1500
FORD	3000		1600
KING	5000		2450

## 11.9 Resultados de classificação

### Problema

Você deseja classificar os salários na tabela EMP, permitindo ao mesmo tempo empates. Você deseja retornar o seguinte conjunto de resultados:

RNK	SAL
1	800
2	950
3	1100
4	1250
4	1250
5	1300
6	1500
7	1600
8	2450
9	2850
10	2975
11	3000
11	3000
12	5000

### Solução

As funções da janela tornam as consultas de classificação extremamente simples. Três funções de janela são particularmente úteis para classificação: DENSE\_RANK OVER, ROW\_NUMBER OVER e RANK OVER.

Como você deseja permitir empates, use a função de janela DENSE\_RANK OVER:

```
1 select dense_rank() over(order by sal) rnk, sal
2 from emp
```

## Discussão

A função de janela DENSE\_RANK OVER faz todo o trabalho braçal aqui. Entre parênteses após a palavra-chave OVER você coloca uma cláusula ORDER BY para especificar a ordem em que as linhas são classificadas. A solução usa ORDER BY SAL, portanto as linhas do EMP são classificadas em ordem crescente de salário.

## 11.10 Suprimindo duplicatas

### Problema

Você deseja localizar os diferentes tipos de trabalho na tabela EMP, mas não deseja ver duplicatas. O conjunto de resultados deve ser o seguinte:

```
JOB
-----
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

### Solução

Todos os RDBMSs suportam a palavra-chave DISTINCT e é sem dúvida o mecanismo mais fácil para suprimir duplicatas do conjunto de resultados. No entanto, esta receita também cobrirá dois métodos adicionais para suprimir duplicatas.

O método tradicional de usar DISTINCT e às vezes GROUP BY certamente funciona. A solução a seguir é uma alternativa que utiliza a função de janela ROW\_NUMBER OVER:

```
1 select job
2   from (
3 select job,
4       row_number()over(partition by job order by job) rn
5   from emp
6      ) x
7 where rn = 1
```

### Alternativas tradicionais

Use a palavra-chave DISTINCT para suprimir duplicatas do conjunto de resultados:

```
select distinct job
  from emp
```

Além disso, também é possível usar GROUP BY para suprimir duplicatas:

```
select job  
      from emp  
     group by job
```

## Discussão

### DB2, Oracle e SQL Server

Esta solução depende de algumas ideias inovadoras sobre funções de janelas particionadas. Usando PARTITION BY na cláusula OVER de ROW\_NUMBER, você pode redefinir o valor retornado por ROW\_NUMBER para 1 sempre que um novo trabalho for encontrado. Os seguintes resultados são da visualização in-line X:

```
select job,  
       row_number()over(partition by job order by job) rn  
  from emp
```

JOB	RN
ANALYST	1
ANALYST	2
CLERK	1
CLERK	2
CLERK	3
CLERK	4
MANAGER	1
MANAGER	2
MANAGER	3
PRESIDENT	1
SALESMAN	1
SALESMAN	2
SALESMAN	3
SALESMAN	4

Cada linha recebe um número sequencial crescente, e esse número é redefinido para um sempre que a tarefa é alterada. Para filtrar as duplicatas, basta manter as linhas onde RN é 1.

Uma cláusula ORDER BY é obrigatória ao usar ROW\_NUMBER OVER (exceto no DB2), mas não afeta o resultado. Qual trabalho é retornado é irrelevante, desde que você devolva um de cada trabalho.

### Alternativas tradicionais

A primeira solução mostra como usar a palavra-chave DISTINCT para suprimir duplicatas de um conjunto de resultados. Tenha em mente que DISTINCT é aplicado a toda a lista SELECT;

colunas adicionais podem e irão alterar o conjunto de resultados. Considere a diferença entre essas duas consultas:

```
select distinct job
  from emp
```

JOB
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN

```
select distinct job, deptno
  from emp
```

JOB	DEPTNO
ANALYST	20
CLERK	10
CLERK	20
CLERK	30
MANAGER	10
MANAGER	20
MANAGER	30
PRESIDENT	10
SALESMAN	30

Ao adicionar DEPTNO à lista SELECT, o que você retorna é cada par DISTINCT de valores JOB/DEPTNO da tabela EMP.

A segunda solução usa GROUP BY para suprimir duplicatas. Embora usar GROUP BY dessa forma não seja incomum, lembre-se de que GROUP BY e DISTINCT são duas cláusulas muito diferentes que não são intercambiáveis. Incluí GROUP BY nesta solução para completar, pois sem dúvida você o encontrará em algum momento.

## 11.11 Encontrando Valores de Cavaleiro

### Problema

Você deseja retornar um conjunto de resultados que contenha o nome de cada funcionário, o departamento em que trabalha, seu salário, a data em que foi contratado e o salário do último funcionário contratado, em cada departamento. Você deseja retornar o seguinte conjunto de resultados:

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-2007	1300
10	KING	5000	17-NOV-2006	1300
10	CLARK	2450	09-JUN-2006	1300
20	ADAMS	1100	12-JAN-2007	1100
20	SCOTT	3000	09-DEC-2007	1100
20	FORD	3000	03-DEC-2006	1100
20	JONES	2975	02-APR-2006	1100
20	SMITH	800	17-DEC-2005	1100
30	JAMES	950	03-DEC-2006	950
30	MARTIN	1250	28-SEP-2006	950
30	TURNER	1500	08-SEP-2006	950
30	BLAKE	2850	01-MAY-2006	950
30	WARD	1250	22-FEB-2006	950
30	ALLEN	1600	20-FEB-2006	950

Os valores em LATEST\_SAL são os “valores do cavalo” porque o caminho para encontrá-los é análogo ao caminho do cavalo no jogo de xadrez. Você determina o resultado da mesma forma que um cavaleiro determina um novo local: saltando para uma linha e depois virando e saltando para uma coluna diferente (veja [Figura 11-1](#)). Para localizar os valores corretos para LATEST\_SAL, você deve primeiro localizar (saltar para) a linha com o HIREDATE mais recente em cada DEPTNO e, em seguida, selecionar (saltar para) a coluna SAL dessa linha.

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
30	JAMES	950	03-DEC-2006	950
30	MARTIN	1250	28-SEP-2006	950
30	TURNER	1500	08-SEP-2006	950
30	BLAKE	2850	01-MAY-2006	950
30	WARD	1250	22-FEB-2006	950
30	ALLEN	1600	20-FEB-2006	950

*Figura 11-1. O valor do cavaleiro vem de “para cima”*



O termo *valor do cavaleiro* foi cunhado por uma colega de trabalho inteligente de Anthony, Kay Young. Depois de fazer com que ele revisasse as receitas para ver se estavam corretas, Anthony admitiu a Kay que estava perplexo e não conseguiu encontrar um bom título. Como você precisa inicialmente avaliar uma linha e depois “pular” e pegar um valor de outra, Kay criou o termo *valor do cavaleiro*.

## Solução

### DB2 e SQLServer

Utilize a expressão CASE em uma subconsulta para retornar o SAL do último funcionário contratado em cada DEPTNO; para todos os outros salários, retorne 0. Use a função de janela MAX OVER na consulta externa para retornar o SAL diferente de zero para o departamento de cada funcionário:

```

1 select deptno,
2 ename,
3 sal,
4 hiredate,
5 max(latest_sal)over(partition by deptno) latest_sal
6 from (
7 select deptno,
8 ename,
9 sal,
```

```

10 hiredate,
11 case
12 when hiredate = max(hiredate)over(partition by deptno)
13 then sal else 0
14 end latest_sal
15 from emp
16 ) x
17 order by 1, 4 desc

```

## Oracle

Use a função de janela MAX OVER para retornar o SAL mais alto para cada DEPTNO. Use as funções DENSE\_RANK e LAST, enquanto ordena por HIREDATE na cláusula KEEP para retornar o SAL mais alto para o último HIREDATE em um determinado DEPTNO:

```

1 select deptno,
2       ename,
3       sal,
4       hiredate,
5       max(sal)
6           keep(dense_rank last order by hiredate)
7           over(partition by deptno) latest_sal
8   from emp
9  order by 1, 4 desc

```

## Discussão

### DB2 e SQLServer

O primeiro passo é utilizar a função de janela MAX OVER em uma expressão CASE para encontrar o último funcionário contratado, ou mais recentemente, em cada DEPTNO. Se HIREDATE de um funcionário corresponder ao valor retornado por MAX OVER, use a expressão CASE para retornar o SAL desse funcionário; caso contrário, retorne zero. Os resultados disso são mostrados aqui:

```

select deptno,
       ename,
       sal,
       hiredate,
       case
           when hiredate = max(hiredate)over(partition by deptno)
               then sal else 0
           end latest_sal
  from emp

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	CLARK	2450	09-JUN-2006	0
10	KING	5000	17-NOV-2006	0
10	MILLER	1300	23-JAN-2007	1300

20 SMITH	800	17-DEC-2005	0
20 ADAMS	1100	12-JAN-2007	1100
20 FORD	3000	03-DEC-2006	0
20 SCOTT	3000	09-DEC-2007	0
20 JONES	2975	02-APR-2006	0
30 ALLEN	1600	20-FEB-2006	0
30 BLAKE	2850	01-MAY-2006	0
30 MARTIN	1250	28-SEP-2006	0
30 JAMES	950	03-DEC-2006	950
30 TURNER	1500	08-SEP-2006	0
30 WARD	1250	22-FEB-2006	0

Como o valor de LATEST\_SAL será zero ou o SAL do(s) funcionário(s) contratado(s) mais recentemente, você pode agrupar a consulta anterior em uma visualização in-line e usar MAX OVER novamente, mas desta vez para retornar o maior LATEST\_SAL diferente de zero para cada DEPTNO :

```

select deptno,
       ename,
       sal,
       hiredate,
       max(latest_sal)over(partition by deptno) latest_sal
  from (
select deptno,
       ename,
       sal,
       hiredate,
       case
           when hiredate = max(hiredate)over(partition by deptno)
           then sal else 0
        end latest_sal
   from emp
      ) x
 order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-2007	1300
10	KING	5000	17-NOV-2006	1300
10	CLARK	2450	09-JUN-2006	1300
20	ADAMS	1100	12-JAN-2007	1100
20	SCOTT	3000	09-DEC-2007	1100
20	FORD	3000	03-DEC-2006	1100
20	JONES	2975	02-APR-2006	1100
20	SMITH	800	17-DEC-2005	1100
30	JAMES	950	03-DEC-2006	950
30	MARTIN	1250	28-SEP-2006	950
30	TURNER	1500	08-SEP-2006	950
30	BLAKE	2850	01-MAY-2006	950
30	WARD	1250	22-FEB-2006	950
30	ALLEN	1600	20-FEB-2006	950

## Oracle

A chave para a solução Oracle é aproveitar as vantagens da cláusula KEEP. A cláusula KEEP permite classificar as linhas retornadas por um grupo/partição e trabalhar com a primeira ou a última linha do grupo. Considere como seria a solução sem KEEP:

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal) over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-2007	5000
10	KING	5000	17-NOV-2006	5000
10	CLARK	2450	09-JUN-2006	5000
20	ADAMS	1100	12-JAN-2007	3000
20	SCOTT	3000	09-DEC-2007	3000
20	FORD	3000	03-DEC-2006	3000
20	JONES	2975	02-APR-2006	3000
20	SMITH	800	17-DEC-2005	3000
30	JAMES	950	03-DEC-2006	2850
30	MARTIN	1250	28-SEP-2006	2850
30	TURNER	1500	08-SEP-2006	2850
30	BLAKE	2850	01-MAY-2006	2850
30	WARD	1250	22-FEB-2006	2850
30	ALLEN	1600	20-FEB-2006	2850

Ao invés de devolver o SAL do último funcionário contratado, o MAX OVER sem KEEP simplesmente retorna o maior salário de cada DEPTNO. KEEP, nesta receita, permite ordenar os salários por HIREDATE em cada DEPTNO especificando ORDER BY HIREDATE. Então, a função DENSE\_RANK atribui uma classificação a cada HIREDATE em ordem crescente. Finalmente, a função LAST determina em qual linha aplicar a função agregada: a “última” linha com base na classificação de DENSE\_RANK. Neste caso, a função agregada MAX é aplicada à coluna SAL para a linha com o “último” HIREDATE. Em essência, manter o SAL do HIREDATE classificado em último lugar em cada DEPTNO.

Você está classificando as linhas em cada DEPTNO com base em uma coluna (HIREDATE), mas aplicando a agregação (MAX) em outra coluna (SAL). Essa capacidade de classificar em uma dimensão e agregar em outra é conveniente, pois permite evitar junções extras e visualizações inline, como são usadas em outras soluções. Finalmente, adicionando a cláusula OVER após a cláusula KEEP, você pode retornar o SAL “mantido” por KEEP para cada linha da partição.

Alternativamente, você pode solicitar por HIREDATE em ordem decrescente e “guardar” o primeiro SAL. Compare as duas consultas a seguir, que retornam o mesmo conjunto de resultados:

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal)
           keep(dense_rank last order by hiredate)
           over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-2007	1300
10	KING	5000	17-NOV-2006	1300
10	CLARK	2450	09-JUN-2006	1300
20	ADAMS	1100	12-JAN-2007	1100
20	SCOTT	3000	09-DEC-2007	1100
20	FORD	3000	03-DEC-2006	1100
20	JONES	2975	02-APR-2006	1100
20	SMITH	800	17-DEC-2005	1100
30	JAMES	950	03-DEC-2006	950
30	MARTIN	1250	28-SEP-2006	950
30	TURNER	1500	08-SEP-2006	950
30	BLAKE	2850	01-MAY-2006	950
30	WARD	1250	22-FEB-2006	950
30	ALLEN	1600	20-FEB-2006	950

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal)
           keep(dense_rank first order by hiredate desc)
           over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-2007	1300
10	KING	5000	17-NOV-2006	1300
10	CLARK	2450	09-JUN-2006	1300
20	ADAMS	1100	12-JAN-2007	1100
20	SCOTT	3000	09-DEC-2007	1100
20	FORD	3000	03-DEC-2006	1100
20	JONES	2975	02-APR-2006	1100

20 SMITH	800	17-DEC-2005	1100
30 JAMES	950	03-DEC-2006	950
30 MARTIN	1250	28-SEP-2006	950
30 TURNER	1500	08-SEP-2006	950
30 BLAKE	2850	01-MAY-2006	950
30 WARD	1250	22-FEB-2006	950
30 ALLEN	1600	20-FEB-2006	950

## 11.12 Gerando Previsões Simples

### Problema

Com base nos dados atuais, você deseja retornar linhas e colunas adicionais que representam ações futuras. Por exemplo, considere o seguinte conjunto de resultados:

ID	ORDER_DATE	PROCESS_DATE
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

Você deseja retornar três linhas por linha retornada em seu conjunto de resultados (cada linha mais duas linhas adicionais para cada pedido). Juntamente com as linhas extras, você gostaria de retornar duas colunas adicionais fornecendo datas para o processamento esperado do pedido.

No conjunto de resultados anterior, você pode ver que um pedido leva dois dias para ser processado. Para efeitos deste exemplo, digamos que a próxima etapa após o processamento seja a verificação e a última etapa seja o envio. A verificação ocorre um dia após o processamento e o envio ocorre um dia após a verificação. Você deseja retornar um conjunto de resultados expressando todo o procedimento. Em última análise, você deseja transformar o conjunto de resultados anterior no seguinte conjunto de resultados:

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

### Solução

A chave é usar um produto cartesiano para gerar duas linhas adicionais para cada pedido e então simplesmente usar as expressões CASE para criar os valores de coluna necessários.

Use a cláusula recursiva COM para gerar as linhas necessárias para seu produto cartesiano. As soluções DB2 e SQL Server são idênticas, exceto pela função usada para recuperar a data atual. O DB2 usa CURRENT\_DATE e o SQL Server usa GETDATE. O MySQL usa CURDATE e requer a inserção da palavra-chave RECURSIVE após WITH para indicar que este é um CTE recursivo. A solução SQL Server é mostrada aqui:

```
1 with nrows(n) as (
2 select 1 from t1 union all
3 select n+1 from nrows where n+1 <= 3
4 )
5 select id,
6       order_date,
7       process_date,
8       case when nrows.n >= 2
9           then process_date+1
10          else null
11      end as verified,
12      case when nrows.n = 3
13          then process_date+2
14          else null
15      end as shipped
16     from (
17 select nrows.n id,
18       getdate() + nrows.n   as order_date,
19       getdate() + nrows.n+2 as process_date
20     from nrows
21   ) orders, nrows
22 order by 1
```

## Oracle

Utilize a cláusula hierárquica CONNECT BY para gerar as três linhas necessárias para o produto cartesiano. Use a cláusula WITH para permitir reutilizar os resultados retornados por CONNECT BY sem precisar chamá-lo novamente:

```
1 with nrows as (
2 select level n
3   from dual
4 connect by level <= 3
5 )
6 select id,
7       order_date,
8       process_date,
9       case when nrows.n >= 2
10          then process_date+1
11          else null
12      end as verified,
13      case when nrows.n = 3
```

```

7  then process_date+2
8  else null
9  end as shipped
10 from (
11select nrow$.$ id,
12 sysdate+nrow$.$ as order_date,
13 sysdate+nrow$.$+2 as process_date
14 from nrow$.$
15 ) orders, nrow$.$

```

## PostgreSQL

Você pode criar um produto cartesiano de muitas maneiras diferentes; esta solução usa a função PostgreSQL GENERATE\_SERIES:

```

1 select id,
2       order_date,
3       process_date,
4       case when gs.n >= 2
5             then process_date+1
6             else null
7       end as verified,
8       case when gs.n = 3
9             then process_date+2
10            else null
11       end as shipped
12  from (
13 select gs.id,
14       current_date+gs.id as order_date,
15       current_date+gs.id+2 as process_date
16  from generate_series(1,3) gs (id)
17  ) orders,
18       generate_series(1,3)gs(n)

```

## MySQL

MySQL não suporta uma função para geração automática de linhas.

## Discussão

### DB2, MySQL e SQL Server

O conjunto de resultados apresentado na seção “Problema” é retornado via visualização inline ORDERS e é mostrado aqui:

```

with nrow$($ as (
select 1 from t1 union all
select n+1 from nrow$ where n+1 <= 3
)
select nrow$.$ id, getdate() + nrow$.$ as order_date,
      getdate() + nrow$.$ + 2 as process_date
     from nrow$.$

```

ID	ORDER_DATE	PROCESS_DATE
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

Esta consulta simplesmente usa a cláusula WITH para formar três linhas que representam os pedidos que você deve processar. NROWS retorna os valores 1, 2 e 3, e esses números são adicionados a GETDATE (CURRENT\_DATE para DB2, CURDATE() para MySQL) para representar as datas dos pedidos. Como a seção “Problema” afirma que o tempo de processamento leva dois dias, a consulta também adiciona dois dias a ORDER\_DATE (adiciona o valor retornado por NROWS a GETDATE e depois adiciona mais dois dias).

Agora que você definiu seu resultado base, a próxima etapa é criar um produto cartesiano porque o requisito é retornar três linhas para cada pedido. Use NROWS para criar um produto cartesiano para retornar três linhas para cada pedido:

```
with nrows(n) as (
  select 1 from t1 union all
  select n+1 from nrows where n+1 <= 3
)
select nrows.n,
       orders.*
  from (
  select nrows.n id,
         getdate()+nrows.n    as order_date,
         getdate()+nrows.n+2 as process_date
    from nrows
   ) orders, nrows
order by 2,1
```

N	ID	ORDER_DATE	PROCESS_DATE
1	1	25-SEP-2005	27-SEP-2005
2	1	25-SEP-2005	27-SEP-2005
3	1	25-SEP-2005	27-SEP-2005
1	2	26-SEP-2005	28-SEP-2005
2	2	26-SEP-2005	28-SEP-2005
3	2	26-SEP-2005	28-SEP-2005
1	3	27-SEP-2005	29-SEP-2005
2	3	27-SEP-2005	29-SEP-2005
3	3	27-SEP-2005	29-SEP-2005

Agora que você tem três linhas para cada pedido, basta usar uma expressão CASE para criar os valores da coluna de adição para representar o status de verificação e envio.

A primeira linha de cada pedido deve ter um valor NULL para VERIFIED e SHIPPED. A segunda linha de cada pedido deve ter um valor NULL para SHIPPED. A terceira linha de cada pedido deve ter valores não NULL para cada coluna. O conjunto de resultados final é mostrado aqui:

```

with nrows(n) as (
  select 1 from t1 union all
  select n+1 from nrows where n+1 <= 3
)
select id,
       order_date,
       process_date,
       case when nrows.n >= 2
            then process_date+1
            else null

       end as verified,
       case when nrows.n = 3
            then process_date+2
            else null
       end as shipped
  from (
select nrows.n id,
       getdate() + nrows.n    as order_date,
       getdate() + nrows.n + 2 as process_date
  from nrows
 ) orders, nrows
 order by 1

```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

O conjunto de resultados finais expressa o processo completo do pedido, desde o dia em que o pedido foi recebido até o dia em que deverá ser enviado.

## Oracle

O conjunto de resultados apresentado na seção do problema é retornado por meio da visualização in-line ORDERS e é mostrado aqui:

```

with nrows as (
  select level n
  from dual

```

```

connect by level <= 3
)
select nrow$ .n id,
       sysdate+nrow$ .n order_date,
       sysdate+nrow$ .n+2 process_date
  from nrow$

```

ID	ORDER_DATE	PROCESS_DATE
-	-	-
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

Esta consulta simplesmente usa CONNECT BY para formar três linhas que representam os pedidos que você deve processar. Use a cláusula WITH para se referir às linhas retornadas por CONNECT BY como NROWS.N. CONNECT BY retorna os valores 1, 2 e 3, e esses números são adicionados a SYSDATE para representar as datas dos pedidos. Como a seção “Problema” afirma que o tempo de processamento leva dois dias, a consulta também adiciona dois dias a ORDER\_DATE (adiciona o valor retornado por GENERATE\_SERIES a SYS-DATE e depois adiciona mais dois dias).

Agora que você definiu seu resultado base, a próxima etapa é criar um produto cartesiano porque o requisito é retornar três linhas para cada pedido. Use NROWS para criar um produto cartesiano para retornar três linhas para cada pedido:

```

with nrow$ as (
  select level n
    from dual
   connect by level <= 3
)
select nrow$ .n,
       orders.* 
  from (
select nrow$ .n id,
       sysdate+nrow$ .n order_date,
       sysdate+nrow$ .n+2 process_date
  from nrow$
 ) orders, nrow$
```

N	ID	ORDER_DATE	PROCESS_DATE
-	-	-	-
1	1	25-SEP-2005	27-SEP-2005
2	1	25-SEP-2005	27-SEP-2005
3	1	25-SEP-2005	27-SEP-2005
1	2	26-SEP-2005	28-SEP-2005
2	2	26-SEP-2005	28-SEP-2005
3	2	26-SEP-2005	28-SEP-2005
1	3	27-SEP-2005	29-SEP-2005
2	3	27-SEP-2005	29-SEP-2005
3	3	27-SEP-2005	29-SEP-2005

Agora que você tem três linhas para cada pedido, basta usar uma expressão CASE para criar os valores da coluna de adição para representar o status de verificação e envio.

A primeira linha de cada pedido deve ter um valor NULL para VERIFIED e SHIPPED. A segunda linha de cada pedido deve ter um valor NULL para SHIPPED. A terceira linha de cada pedido deve ter valores não NULL para cada coluna. O conjunto de resultados final é mostrado aqui:

```
with nrows as (
  select level n
    from dual
   connect by level <= 3
)
select id,
       order_date,
       process_date,
       case when nrows.n >= 2
             then process_date+1
             else null
         end as verified,
       case when nrows.n = 3
             then process_date+2
             else null
         end as shipped
  from (
  select nrows.n id,
         sysdate+nrows.n order_date,
         sysdate+nrows.n+2 process_date
    from nrows
   ) orders, nrows
```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

O conjunto de resultados finais expressa o processo completo do pedido, desde o dia em que o pedido foi recebido até o dia em que deverá ser enviado.

## PostgreSQL

O conjunto de resultados apresentado na seção do problema é retornado por meio da visualização in-line ORDERS e é mostrado aqui:

```
select gs.id,
       current_date+gs.id as order_date,
       current_date+gs.id+2 as process_date
  from generate_series(1,3) gs (id)
```

ID	ORDER_DATE	PROCESS_DATE
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

Esta consulta simplesmente usa a função GENERATE\_SERIES para formar três linhas que representam os pedidos que você deve processar. GENERATE\_SERIES retorna os valores 1, 2 e 3, e esses números são adicionados a CURRENT\_DATE para representar as datas dos pedidos. Como a seção “Problema” afirma que o tempo de processamento leva dois dias, a consulta também adiciona dois dias a ORDER\_DATE (adiciona o valor retornado por GENERATE\_SERIES a CURRENT\_DATE e depois adiciona mais dois dias). Agora que você definiu seu resultado base, a próxima etapa é criar um produto cartesiano porque o requisito é retornar três linhas para cada pedido. Use a função GENERATE\_SERIES para criar um produto cartesiano para retornar três linhas para cada pedido:

```
select gs.n,
       orders.*
  from (
select gs.id,
       current_date+gs.id as order_date,
       current_date+gs.id+2 as process_date
  from generate_series(1,3) gs (id)
 ) orders,
      generate_series(1,3)gs(n)
```

N	ID	ORDER_DATE	PROCESS_DATE
1	1	25-SEP-2005	27-SEP-2005
2	1	25-SEP-2005	27-SEP-2005
3	1	25-SEP-2005	27-SEP-2005
1	2	26-SEP-2005	28-SEP-2005
2	2	26-SEP-2005	28-SEP-2005
3	2	26-SEP-2005	28-SEP-2005
1	3	27-SEP-2005	29-SEP-2005
2	3	27-SEP-2005	29-SEP-2005
3	3	27-SEP-2005	29-SEP-2005

Agora que você tem três linhas para cada pedido, basta usar uma expressão CASE para criar os valores da coluna de adição para representar o status de verificação e envio.

A primeira linha de cada pedido deve ter um valor NULL para VERIFIED e SHIPPED. A segunda linha de cada pedido deve ter um valor NULL para SHIPPED. A terceira linha de cada pedido deve ter valores não NULL para cada coluna. O conjunto de resultados final é mostrado aqui:

```

select id,
       order_date,
       process_date,
       case when gs.n >= 2
            then process_date+1
            else null
       end as verified,
       case when gs.n = 3
            then process_date+2
            else null
       end as shipped
  from (
select gs.id,
       current_date+gs.id as order_date,
       current_date+gs.id+2 as process_date
  from generate_series(1,3) gs(id)
 ) orders,
      generate_series(1,3)gs(n)

```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

O conjunto de resultados finais expressa o processo completo do pedido, desde o dia em que o pedido foi recebido até o dia em que deverá ser enviado.

## 11.13 Resumindo

As receitas deste capítulo representam problemas práticos que não podem ser resolvidos com uma única função. Esses são alguns dos tipos de problemas que os usuários corporativos frequentemente recorrem a você para resolvê-los.

# CAPÍTULO 12

## Relatórios e Remodelação

Este capítulo apresenta consultas que podem ser úteis para a criação de relatórios. Normalmente envolvem considerações de formatação específicas de relatórios, juntamente com diferentes níveis de agregação. Outro foco deste capítulo é transpor ou dinamizar conjuntos de resultados: remodelar os dados transformando linhas em colunas.

Em geral, essas receitas têm em comum o fato de permitirem apresentar dados em formatos ou formas diferentes da forma como estão armazenados. À medida que seu nível de conforto com a rotação aumenta, você sem dúvida encontrará usos para ela além dos apresentados neste capítulo.

### 12.1 Dinamizando um conjunto de resultados em uma linha

#### Problema

Você deseja obter valores de grupos de linhas e transformar esses valores em colunas em uma única linha por grupo. Por exemplo, você tem um conjunto de resultados exibindo o número de funcionários em cada departamento:

DEPTNO	CNT
10	3
20	5
30	6

Você gostaria de reformatar a saída para que o conjunto de resultados tenha a seguinte aparência:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

Este é um exemplo clássico de dados apresentados em uma forma diferente da forma como são armazenados.

## Solução

Transponha o conjunto de resultados usando CASE e a função agregada SUM:

```
1 select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
2      sum(case when deptno=20 then 1 else 0 end) as deptno_20,
3      sum(case when deptno=30 then 1 else 0 end) as deptno_30
4  from emp
```

## Discussão

Este exemplo é uma excelente introdução ao pivotamento. O conceito é simples: para cada linha retornada pela consulta não dinâmica, use uma expressão CASE para separar as linhas em colunas. Então, como este problema específico é contar o número de funcionários por departamento, utilize a função agregada SUM para contar a ocorrência de cada DEPTNO. Se você estiver tendo problemas para entender como isso funciona exatamente, execute a consulta com a função agregada SUM e inclua DEPTNO para facilitar a leitura:

```
select deptno,
       case when deptno=10 then 1 else 0 end as deptno_10,
       case when deptno=20 then 1 else 0 end as deptno_20,
       case when deptno=30 then 1 else 0 end as deptno_30
  from emp
 order by 1
```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	1	0	0
10	1	0	0
10	1	0	0
20	0	1	0
20	0	1	0
20	0	1	0
20	0	1	0
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1

Você pode pensar em cada expressão CASE como um sinalizador para determinar a qual DEPTNO uma linha pertence. Neste ponto, a transformação “linhas em colunas” já está feita; o próximo passo é simplesmente somar os valores retornados por DEPTNO\_10, DEPTNO\_20 e DEPTNO\_30 e depois agrupar por DEPTNO. A seguir estão os resultados:

```

select deptno,
       sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
 group by deptno

DEPTNO    DEPTNO_10    DEPTNO_20    DEPTNO_30
-----  -----  -----  -----
  10          3          0          0
  20          0          5          0
  30          0          0          6

```

Se você inspecionar esse conjunto de resultados, verá que logicamente a saída faz sentido; por exemplo, DEPTNO 10 possui três funcionários em DEPTNO\_10 e zero nos demais departamentos. Como o objetivo é retornar uma linha, o último passo é remover as cláusulas DEPTNO e GROUP BY e simplesmente somar as expressões CASE:

```

select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp

DEPTNO_10    DEPTNO_20    DEPTNO_30
-----  -----  -----
            3            5            6

```

A seguir está outra abordagem que às vezes você pode ver aplicada a esse mesmo tipo de problema:

```

select max(case when deptno=10 then empcount else null end) as deptno_10
      , max(case when deptno=20 then empcount else null end) as deptno_20,
      , max(case when deptno=30 then empcount else null end) as deptno_30
   from (
select deptno, count(*) as empcount
  from emp
 group by deptno
 ) x

```

Essa abordagem usa uma visualização em linha para gerar a contagem de funcionários por departamento. expressões CASE na consulta principal traduzem linhas em colunas, levando você aos seguintes resultados:

```

DEPTNO_10    DEPTNO_20    DEPTNO_30
-----  -----  -----
      3        NULL        NULL
    NULL         5        NULL
    NULL        NULL         6

```

Então a função MAX recolhe as colunas em uma linha:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

## 12.2 Dinamizando um conjunto de resultados em várias linhas

### Problema

Você deseja transformar linhas em colunas criando uma coluna correspondente a cada um dos valores em uma única coluna. No entanto, ao contrário da receita anterior, você precisa de várias linhas de saída. Como na receita anterior, dinamizar em várias linhas é um método fundamental de remodelar dados.

Por exemplo, você deseja retornar cada funcionário e seu cargo (JOB) e atualmente usa uma consulta que retorna o seguinte conjunto de resultados:

JOB	ENAME
ANALYST	SCOTT
ANALYST	FORD
CLERK	SMITH
CLERK	ADAMS
CLERK	MILLER
CLERK	JAMES
MANAGER	JONES
MANAGER	CLARK
MANAGER	BLAKE
PRESIDENT	KING
SALESMAN	ALLEN
SALESMAN	MARTIN
SALESMAN	TURNER
SALESMAN	WARD

Você gostaria de formatar o conjunto de resultados de forma que cada trabalho tenha sua própria coluna:

CLERKS	ANALYSTS	MGRS	PREZ	SALES
MILLER	FORD	CLARK	KING	TURNER
JAMES	SCOTT	BLAKE		MARTIN
ADAMS		JONES		WARD
SMITH				ALLEN

### Solução

Ao contrário da primeira receita deste capítulo, o conjunto de resultados desta receita consiste em mais de uma linha. Usar a técnica da receita anterior não funcionará para esta receita, pois seria retornado o MAX(ENAME) para cada JOB, o que resultaria em um ENAME para cada JOB (ou seja, uma linha será retornada como na primeira receita).

Para resolver este problema, você deve tornar cada combinação JOB/ENAME única. Então, ao aplicar uma função agregada para remover NULLs, você não perde nenhum ENAME.

Use a função de classificação ROW\_NUMBER OVER para tornar cada combinação JOB/ENAME única. Dinamize o conjunto de resultados usando uma expressão CASE e a função agregada MAX enquanto agrupa no valor retornado pela função de janela:

```

1 select max(case when job='CLERK'
2                 then ename else null end) as clerks,
3       max(case when job='ANALYST'
4                 then ename else null end) as analysts,
5       max(case when job='MANAGER'
6                 then ename else null end) as mgrs,
7       max(case when job='PRESIDENT'
8                 then ename else null end) as prez,
9       max(case when job='SALESMAN'
10                then ename else null end) as sales
11   from (
12 select job,
13       ename,
14       row_number()over(partition by job order by ename) rn
15   from emp
16      ) x
17 group by rn

```

## Discussão

O primeiro passo é usar a função de janela ROW\_NUMBER OVER para ajudar a tornar cada combinação JOB/ENAME única:

```

select job,
       ename,
       row_number()over(partition by job order by ename) rn
  from emp

```

JOB	ENAME	RN
ANALYST	FORD	1
ANALYST	SCOTT	2
CLERK	ADAMS	1
CLERK	JAMES	2
CLERK	MILLER	3
CLERK	SMITH	4
MANAGER	BLAKE	1
MANAGER	CLARK	2
MANAGER	JONES	3
PRESIDENT	KING	1
SALESMAN	ALLEN	1
SALESMAN	MARTIN	2
SALESMAN	TURNER	3
SALESMAN	WARD	4

Atribuir a cada ENAME um “número de linha” exclusivo dentro de um determinado cargo evita quaisquer problemas que poderiam resultar de dois funcionários com o mesmo nome e cargo. O objetivo aqui é poder agrupar por número de linha (em RN) sem eliminar nenhum funcionário do conjunto de resultados devido ao uso de MAX. Esta etapa é a etapa mais importante na resolução do problema. Sem esta primeira etapa, a agregação na consulta externa removerá as linhas necessárias. Considere como seria o conjunto de resultados sem usar ROW\_NUMBER OVER, usando a mesma técnica mostrada na primeira receita:

```
select max(case when job='CLERK'
              then ename else null end) as clerks,
       max(case when job='ANALYST'
              then ename else null end) as analysts,
       max(case when job='MANAGER'
              then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
              then ename else null end) as prez,
       max(case when job='SALESMAN'
              then ename else null end) as sales
  from emp
```

CLERKS	ANALYSTS	MGRS	PREZ	SALES
-----	-----	-----	-----	-----
SMITH	SCOTT	JONES	KING	WARD

Infelizmente, apenas uma linha é retornada para cada JOB: o funcionário com MAX ENAME. Quando chegar a hora de dinamizar o conjunto de resultados, usar MIN ou MAX deve servir como um meio de remover NULLs do conjunto de resultados, e não restringir os ENAMES retornados. Como isso funciona ficará mais claro à medida que você continuar com a explicação.

A próxima etapa usa uma expressão CASE para organizar os ENAMES em suas colunas apropriadas (JOB):

```
select rn,
       case when job='CLERK'
              then ename else null end as clerks,
       case when job='ANALYST'
              then ename else null end as analysts,
       case when job='MANAGER'
              then ename else null end as mgrs,
       case when job='PRESIDENT'
              then ename else null end as prez,
       case when job='SALESMAN'
              then ename else null end as sales
  from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
  from emp
 ) x
```

RN	CLERKS	ANALYSTS	MGRS	PREZ	SALES
1		FORD			
2		SCOTT			
1	ADAMS				
2	JAMES				
3	MILLER				
4	SMITH				
1			BLAKE		
2			CLARK		
3			JONES		
1				KING	
1					ALLEN
2					MARTIN
3					TURNER
4					WARD

Neste ponto, as linhas são transpostas para colunas, e a última etapa é remover os NULLs para tornar o conjunto de resultados mais legível. Para remover os NULLs, use a função agregada MAX e agrupe por RN. (Você também pode usar a função MIN. A escolha de usar MAX é arbitrária, pois você agritará apenas um valor por grupo.) Existe apenas um valor para cada combinação RN/JOB/ENAME. O agrupamento por RN em conjunto com as expressões CASE incorporadas nas chamadas para MAX garante que cada chamada para MAX resulte na escolha de apenas um nome de um grupo de valores NULL:

```

select max(case when job='CLERK'
               then ename else null end) as clerks,
       max(case when job='ANALYST'
               then ename else null end) as analysts,
       max(case when job='MANAGER'
               then ename else null end) as mtrs,
       max(case when job='PRESIDENT'
               then ename else null end) as prez,
       max(case when job='SALESMAN'
               then ename else null end) as sales
  from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
  from emp
   ) x
 group by rn

CLERKS ANALYSTS MGRS PREZ SALES
----- ----- ----- ----- -----
MILLER FORD      CLARK  KING   TURNER
JAMES  SCOTT     BLAKE  MARTIN
ADAMS
SMITH

```

A técnica de usar ROW\_NUMBER OVER para criar combinações exclusivas de linhas é extremamente útil para formatar resultados de consultas. Considere a seguinte consulta que cria um relatório esparsão mostrando funcionários por DEPTNO e JOB:

```

select deptno dno, job,
       max(case when deptno=10
                 then ename else null end) as d10,
       max(case when deptno=20
                 then ename else null end) as d20,
       max(case when deptno=30
                 then ename else null end) as d30,
       max(case when job='CLERK'
                 then ename else null end) as clerks,
       max(case when job='ANALYST'
                 then ename else null end) as analists,
       max(case when job='MANAGER'
                 then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                 then ename else null end) as prez,
       max(case when job='SALESMAN'
                 then ename else null end) as sales
  from (
Select deptno,
       job,
       ename,
       row_number()over(partition by job order by ename) rn_job,
       row_number()over(partition by job order by ename) rn_deptno
  from emp
 ) x
 group by deptno, job, rn_deptno, rn_job
 order by 1

```

DNO	JOB	D10	D20	D30	CLERKS	ANALISTS	MGRS	PREZ	SALES
10	CLERK	MILLER			MILLER				
10	MANAGER	CLARK					CLARK		
10	PRESIDENT	KING						KING	
20	ANALYST		FORD			FORD			
20	ANALYST		SCOTT			SCOTT			
20	CLERK		ADAMS		ADAMS				
20	CLERK		SMITH		SMITH				
20	MANAGER		JONES				JONES		
30	CLERK			JAMES	JAMES				
30	MANAGER			BLAKE		BLAKE			
30	SALESMAN			ALLEN			ALLEN		
30	SALESMAN			MARTIN			MARTIN		
30	SALESMAN			TURNER			TURNER		
30	SALESMAN			WARD			WARD		

Simplesmente modificando o que você agrupa (daí os itens não agregados na lista SELECT anterior), você pode produzir relatórios com formatos diferentes.

Vale a pena mudar as coisas para entender como esses formatos mudam com base no que você inclui em sua cláusula GROUP BY.

## 12.3 Dinâmica reversa de um conjunto de resultados

### Problema

Você deseja transformar colunas em linhas. Considere o seguinte conjunto de resultados:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

Você gostaria de converter isso para o seguinte:

DEPTNO	COUNTS_BY_DEPT
10	3
20	5
30	6

Alguns leitores devem ter notado que a primeira listagem é o resultado da primeira receita deste capítulo. Para disponibilizar esta saída para esta receita, podemos armazená-la em uma visualização com a seguinte consulta:

```
create view emp_cnts as
(
  select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
         sum(case when deptno=20 then 1 else 0 end) as deptno_20,
         sum(case when deptno=30 then 1 else 0 end) as deptno_30
   from emp
)
```

Na solução e discussão a seguir, as consultas farão referência à visualização EMP\_CNTS criada pela consulta anterior.

### Solução

Examinando o conjunto de resultados desejado, é fácil ver que você pode executar um simples COUNT e GROUP BY na tabela EMP para produzir o resultado desejado. O objetivo aqui, porém, é imaginar que os dados não são armazenados como linhas; talvez os dados sejam desnormalizados e os valores agregados sejam armazenados como múltiplas colunas.

Para converter colunas em linhas, use um produto cartesiano. Você precisará saber antecipadamente quantas colunas deseja converter em linhas porque a expressão de tabela usada para criar o produto cartesiano deve ter uma cardinalidade de pelo menos o número de colunas que deseja transportar.

Em vez de criar uma tabela de dados desnormalizada, a solução para esta receita usará a solução da primeira receita deste capítulo para criar um conjunto de resultados “amplo”. A solução completa é a seguinte:

```

1 select dept.deptno,
2       case dept.deptno
3           when 10 then emp_cnts.deptno_10
4           when 20 then emp_cnts.deptno_20
5           when 30 then emp_cnts.deptno_30
6       end as counts_by_dept
7   from emp_cnts cross join
8       (select deptno from dept where deptno <= 30) dept

```

## Discussão

A visualização EMP\_CNTS representa a visualização desnormalizada, ou conjunto de resultados “amplo” que você deseja converter em linhas, e é mostrada aqui:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

Como existem três colunas, você criará três linhas. Comece criando um produto cartesiano entre a visualização embutida EMP\_CNTS e alguma expressão de tabela que tenha pelo menos três linhas. O código a seguir usa a tabela DEPT para criar o produto cartesiano; DEPT tem quatro linhas:

```

select dept.deptno,
       emp_cnts.deptno_10,
       emp_cnts.deptno_20,
       emp_cnts.deptno_30
  from (
Select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
    ) emp_cnts,
       (select deptno from dept where deptno <= 30) dept

```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	3	5	6
20	3	5	6
30	3	5	6

O produto cartesiano permite retornar uma linha para cada coluna na visualização embutida EMP\_CNTS. Como o conjunto de resultados final deverá conter apenas o DEPTNO e o número de funcionários desse DEPTNO, utilize uma expressão CASE para transformar as três colunas em uma:

```

select dept.deptno,
       case dept.deptno
         when 10 then emp_cnts.deptno_10
         when 20 then emp_cnts.deptno_20
         when 30 then emp_cnts.deptno_30
       end as counts_by_dept
  from (
    emp_cnts
  cross join (select deptno from dept where deptno <= 30) dept
DEPTNO COUNTS_BY_DEPT
-----
10          3
20          5
30          6

```

## 12.4 Dinâmica reversa de um conjunto de resultados em uma coluna

### Problema

Você deseja retornar todas as colunas de uma consulta como apenas uma coluna. Por exemplo, você deseja retornar ENAME, JOB e SAL de todos os funcionários em DEPTNO 10 e deseja retornar todos os três valores em uma coluna. Você deseja retornar três linhas para cada funcionário e uma linha de espaço em branco entre funcionários. Você deseja retornar o seguinte conjunto de resultados:

```

EMPS
-----
CLARK
MANAGER
2450

KING
PRESIDENT
5000

MILLER
CLERK
1300

```

### Solução

A chave é usar um CTE recursivo combinado com um produto cartesiano para retornar quatro linhas para cada funcionário. [Capítulo 10](#) cobre o CTE recursivo que precisamos e é explorado mais detalhadamente em [Apêndice B](#). Usar a junção cartesiana permite retornar um valor de coluna por linha e ter uma linha extra para espaçamento entre funcionários.

Use a função de janela ROW\_NUMBER OVER para classificar cada linha com base em EMPNO (1–4). Em seguida, use uma expressão CASE para transformar três colunas em uma (a palavra-chave RECURSIVE é necessária após o primeiro WITH no PostgreSQL e MySQL):

```
1  with four_rows (id)
2    as
3  (
4    select 1
5      union all
6    select id+1
7      from four_rows
8      where id < 4
9  )
10 ,
11  x_tab (ename,job,sal,rn )
12  as
13  (
14    select e.ename,e.job,e.sal,
15      row_number()over(partition by e.empno
16      order by e.empno)
17      from emp e
18      join four_rows on 1=1
19  )
20
21  select
22    case rn
23      when 1 then ename
24      when 2 then job
25      when 3 then cast(sal as char(4))
26  end emps
27  from x_tab
```

## Discussão

O primeiro passo é usar a função de janela ROW\_NUMBER OVER para criar uma classificação para cada funcionário no DEPTNO 10:

```
select e.ename,e.job,e.sal,
       row_number()over(partition by e.empno
                         order by e.empno) rn
  from emp e
 where e.deptno=10
```

ENAME	JOB	SAL	RN
CLARK	MANAGER	2450	1
KING	PRESIDENT	5000	1
MILLER	CLERK	1300	1

Neste ponto, a classificação não significa muito. Você está particionando por EMPNO, então a classificação é 1 para todas as três linhas em DEPTNO 10. Depois de adicionar o produto cartesiano, a classificação começará a tomar forma, conforme mostrado nos resultados a seguir:

```
with four_rows (id)
as
(select 1
union all
select id+1
from four_rows
where id < 4
)
select e.ename,e.job,e.sal,
row_number()over(partition by e.empno
order by e.empno)
from emp e
join four_rows on 1=1
```

ENAME	JOB	SAL	RN
CLARK	MANAGER	2450	1
CLARK	MANAGER	2450	2
CLARK	MANAGER	2450	3
CLARK	MANAGER	2450	4
KING	PRESIDENT	5000	1
KING	PRESIDENT	5000	2
KING	PRESIDENT	5000	3
KING	PRESIDENT	5000	4
MILLER	CLERK	1300	1
MILLER	CLERK	1300	2
MILLER	CLERK	1300	3
MILLER	CLERK	1300	4

Você deve parar neste ponto e entender dois pontos principais:

- RN deixa de ser 1 para cada funcionário; agora é uma sequência repetida de valores de 1 a 4, a razão é que as funções de janela são aplicadas após as cláusulas FROM e WHERE serem avaliadas. Portanto, o particionamento por faz com que EMPNO o RN seja redefinido para 1 quando um novo funcionário é encontrado.
- Usamos um CTE recursivo para garantir que para cada funcionário haja quatro linhas. Não precisamos da palavra-chave RECURSIVE no SQL Server ou DB2, mas precisamos para Oracle, MySQL e PostgreSQL.

O trabalho duro agora está feito, e tudo o que resta é usar uma expressão CASE para colocar ENAME, JOB e SAL em uma coluna para cada funcionário (você precisa usar CAST para converter SAL em uma string para manter CASE feliz):

```
with four_rows (id)
as
(select 1
```

```
union all
select id+1
from four_rows
where id < 4
)
,
x_tab (ename,job,sal,rn )
as
(select e.ename,e.job,e.sal,
row_number()over(partition by e.empno
order by e.empno)
from emp e
join four_rows on 1=1)

select case rn
when 1 then ename
when 2 then job
when 3 then cast(sal as char(4))
end emps
from x_tab

EMPS
-----
CLARK
MANAGER
2450

KING
PRESIDENT
5000

MILLER
CLERK
1300
```

## 12.5 Suprimindo valores repetidos de um conjunto de resultados

### Problema

Você está gerando um relatório e quando duas linhas têm o mesmo valor em uma coluna, você deseja exibir esse valor apenas uma vez. Por exemplo, você deseja retornar DEPTNO e ENAME da tabela EMP, deseja agrupar todas as linhas para cada DEPTNO e deseja exibir cada DEPTNO apenas uma vez. Você deseja retornar o seguinte conjunto de resultados:

```
DEPTNO ENAME
-----
10 CLARK
      KING
      MILLER
```

```

20 SMITH
ADAMS
FORD
SCOTT
JONES
30 ALLEN
BLAKE
MARTIN
JAMES
TURNER
WARD

```

## Solução

Este é um problema simples de formatação que é facilmente resolvido pela função de window LAG OVER:

```

1 select
2      case when
3          lag(deptno)over(order by deptno) = deptno then null
4          else deptno end DEPTNO
5      , ename
6  from emp

```

Os usuários Oracle também podem usar DECODE como alternativa ao CASE:

```

1 select to_number(
2      decode(lag(deptno)over(order by deptno),
3              deptno,null,deptno)
4      ) deptno, ename
5  from emp

```

## Discussão

O primeiro passo é usar a função de window LAG OVER para retornar o DEPTNO anterior para cada linha:

```

select lag(deptno)over(order by deptno) lag_deptno,
       deptno,
       ename
  from emp

```

LAG_DEPTNO	DEPTNO	ENAME
10	10	CLARK
10	10	KING
10	10	MILLER
10	20	SMITH
20	20	ADAMS
20	20	FORD
20	20	SCOTT
20	20	JONES

```
20      30 ALLEN
30      30 BLAKE
30      30 MARTIN
30      30 JAMES
30      30 TURNER
30      30 WARD
```

Se você inspecionar o conjunto de resultados anterior, poderá ver facilmente onde DEPTNO corresponde a LAG\_DEPTNO. Para essas linhas, você deseja definir DEPTNO como NULL. Faça isso usando DECODE (TO\_NUMBER está incluído para converter DEPTNO como um número):

```
select to_number(
    CASE WHEN (lag(deptno)over(order by deptno)
= deptno THEN null else deptno END deptno ,
    deptno,null,deptno)
) deptno, ename
from emp

DEPTNO ENAME
-----
10 CLARK
KING
MILLER
20 SMITH
ADAMS
FORD
SCOTT
JONES
30 ALLEN
BLAKE
MARTIN
JAMES
TURNER
WARD
```

## 12.6 Dinamizando um conjunto de resultados para facilitar cálculos entre linhas

### Problema

Você deseja fazer cálculos envolvendo dados de diversas linhas. Para facilitar seu trabalho, você deseja dinamizar essas linhas em colunas de forma que todos os valores necessários fiquem em uma única linha.

Nos dados de exemplo deste livro, DEPTNO 20 é o departamento com maior salário combinado, o que você pode confirmar executando a seguinte consulta:

```
select deptno, sum(sal) as sal
  from emp
 group by deptno
```

DEPTNO	SAL
10	8750
20	10875
30	9400

Você deseja calcular a diferença entre os salários do DEPTNO 20 e DEPTNO 10 e entre DEPTNO 20 e DEPTNO 30.

O resultado final ficará assim:

d20_10_diff	d20_30_diff
2125	1475

## Solução

Transponha os totais usando a função agregada SUM e uma expressão CASE. Em seguida, codifique suas expressões na lista de seleção:

```

1 select d20_sal - d10_sal as d20_10_diff,
2       d20_sal - d30_sal as d20_30_diff
3   from (
4 select sum(case when deptno=10 then sal end) as d10_sal,
5       sum(case when deptno=20 then sal end) as d20_sal,
6       sum(case when deptno=30 then sal end) as d30_sal
7   from emp
8 ) totals_by_dept

```

Também é possível escrever esta consulta usando um CTE, que algumas pessoas podem achar mais legível:

```

with totals_by_dept (d10_sal, d20_sal, d30_sal)
as
(select
     sum(case when deptno=10 then sal end) as d10_sal,
     sum(case when deptno=20 then sal end) as d20_sal,
     sum(case when deptno=30 then sal end) as d30_sal

  from emp)

select  d20_sal - d10_sal as d20_10_diff,
        d20_sal - d30_sal as d20_30_diff
  from totals_by_dept

```

## Discussão

O primeiro passo é dinamizar os salários de cada DEPTNO de linhas para colunas usando uma expressão CASE:

```
select case when deptno=10 then sal end as d10_sal,
       case when deptno=20 then sal end as d20_sal,
       case when deptno=30 then sal end as d30_sal
  from emp
```

D10_SAL	D20_SAL	D30_SAL
800		
	1600	
	1250	
2975		
	1250	
	2850	
2450		
	3000	
5000		1500
	1100	
		950
	3000	
1300		

O próximo passo é somar todos os salários de cada DEPTNO aplicando a função agregada SUM a cada expressão CASE:

```
select sum(case when deptno=10 then sal end) as d10_sal,
       sum(case when deptno=20 then sal end) as d20_sal,
       sum(case when deptno=30 then sal end) as d30_sal
  from emp
```

D10_SAL	D20_SAL	D30_SAL
8750	10875	9400

A etapa final é simplesmente agrupar o SQL anterior em uma visualização embutida e realizar as subtrações.

## 12.7 Criação de grupos de dados de tamanho fixo

### Problema

Você deseja organizar os dados em intervalos de tamanhos uniformes, com um número predeterminado de elementos em cada intervalo. O número total de buckets pode ser desconhecido, mas você deseja garantir que cada bucket tenha cinco elementos. Por exemplo, você deseja organizar os funcionários da tabela EMP em grupos de cinco com base no valor de EMPNO, conforme mostrado nos resultados a seguir:

GRP	EMPNO	ENAME
1	7369	SMITH
1	7499	ALLEN

```

1      7521 WARD
1      7566 JONES
1      7654 MARTIN
2      7698 BLAKE
2      7782 CLARK
2      7788 SCOTT
2      7839 KING
2      7844 TURNER
3      7876 ADAMS
3      7900 JAMES
3      7902 FORD
3      7934 MILLER

```

## Solução

A solução para este problema é bastante simplificada por funções para classificar linhas. Depois que as linhas são classificadas, criar grupos de cinco é simplesmente uma questão de dividir e depois calcular o teto matemático do quociente.

Use a função de window ROW\_NUMBER OVER para classificar cada funcionário por EMPNO. Em seguida, divida por cinco para criar os grupos (os usuários do SQL Server usarão CEILING, não CEIL):

```

1 select ceil(row_number()over(order by empno)/5.0) grp,
2       empno,
3       ename
4  from emp

```

## Discussão

A função de window ROW\_NUMBER OVER atribui uma classificação ou “número de linha” a cada linha classificada por EMPNO:

```

select row_number()over(order by empno) rn,
       empno,
       ename
  from emp

```

RN	EMPNO	ENAME
1	7369	SMITH
2	7499	ALLEN
3	7521	WARD
4	7566	JONES
5	7654	MARTIN
6	7698	BLAKE
7	7782	CLARK
8	7788	SCOTT
9	7839	KING
10	7844	TURNER
11	7876	ADAMS

```
12      7900 JAMES
13      7902 FORD
14      7934 MILLER
```

O próximo passo é aplicar a função CEIL (ou CEILING) após dividir ROW\_NUMBER OVER por cinco. A divisão por cinco organiza logicamente as linhas em grupos de cinco (ou seja, cinco valores menores ou iguais a 1, cinco valores maiores que 1, mas menores ou iguais a 2); o grupo restante (composto pelas últimas 4 linhas desde 14, o número de linhas na tabela EMP, não é múltiplo de 5) tem um valor maior que 2, mas menor ou igual a 3.

A função CEIL retornará o menor número inteiro maior que o valor passado a ela; isso criará grupos de números inteiros. Os resultados da divisão e aplicação do CEIL são apresentados aqui. Você pode seguir a ordem de operação da esquerda para a direita, de RN para DIVISÃO para GRP:

```
select row_number()over(order by empno) rn,
       row_number()over(order by empno)/5.0 division,
       ceil(row_number()over(order by empno)/5.0) grp,
       empno,
       ename
  from emp
```

RN	DIVISION	GRP	EMPNO	ENAME
1	.2	1	7369	SMITH
2	.4	1	7499	ALLEN
3	.6	1	7521	WARD
4	.8	1	7566	JONES
5	1	1	7654	MARTIN
6	1.2	2	7698	BLAKE
7	1.4	2	7782	CLARK
8	1.6	2	7788	SCOTT
9	1.8	2	7839	KING
10	2	2	7844	TURNER
11	2.2	3	7876	ADAMS
12	2.4	3	7900	JAMES
13	2.6	3	7902	FORD
14	2.8	3	7934	MILLER

## 12.8 Criando um número predefinido de buckets

### Problema

Você deseja organizar seus dados em um número fixo de intervalos. Por exemplo, você deseja organizar os funcionários da tabela EMP em quatro grupos. O conjunto de resultados deve ser semelhante ao seguinte:

GRP	EMPNO	ENAME
1	7369	SMITH
1	7499	ALLEN
1	7521	WARD
1	7566	JONES
2	7654	MARTIN
2	7698	BLAKE
2	7782	CLARK
2	7788	SCOTT
3	7839	KING
3	7844	TURNER
3	7876	ADAMS
4	7900	JAMES
4	7902	FORD
4	7934	MILLER

Esta é uma forma comum de organizar dados categóricos, pois dividir um conjunto em vários conjuntos menores de tamanhos iguais é um primeiro passo importante para muitos tipos de análise. Por exemplo, tomar as médias salariais destes grupos ou qualquer outro valor pode revelar uma tendência que é ocultada pela variabilidade quando se olham os casos individualmente.

Este problema é o oposto da receita anterior, onde você tinha um número desconhecido de balde, mas um número predeterminado de elementos em cada balde. Nesta receita, o objetivo é tal que você pode não saber necessariamente quantos elementos há em cada balde, mas você está definindo um número fixo (conhecido) de balde a serem criados.

## Solução

A solução para este problema é simples agora que a função NTILE está amplamente disponível. NTILE organiza um conjunto ordenado no número de buckets que você especifica, com quaisquer retardatários distribuídos nos buckets disponíveis, começando no primeiro bucket. O conjunto de resultados desejados para esta receita reflete isto: os baldes 1 e 2 têm quatro linhas, enquanto os baldes 3 e 4 têm três linhas.

Use a função de window NTILE para criar quatro buckets:

```
1 select ntile(4)over(order by empno) grp,
2      empno,
3      ename
4  from emp
```

## Discussão

Todo o trabalho é feito pela função NTILE. A cláusula ORDER BY coloca as linhas na ordem desejada e a própria função atribui um número de grupo a cada linha, por exemplo, para que o primeiro trimestre (neste caso) seja colocado no grupo um, o segundo no grupo dois, etc.

## 12.9 Criação de histogramas horizontais

### Problema

Você deseja usar SQL para gerar histogramas que se estendem horizontalmente. Por exemplo, você deseja exibir o número de funcionários em cada departamento como um histograma horizontal com cada funcionário representado por uma instância de \*. Você deseja retornar o seguinte conjunto de resultados:

DEPTNO	CNT
10	***
20	*****
30	*****

### Solução

A chave para esta solução é usar a função agregada COUNT e usar GROUP BY DEPTNO para determinar o número de funcionários em cada DEPTNO. O valor retornado por COUNT é então passado para uma função de string que gera uma série de caracteres \*.

#### DB2

Use a função REPEAT para gerar o histograma:

```
1 select deptno,
2       repeat('*',count(*)) cnt
3   from emp
4  group by deptno
```

#### Oracle, PostgreSQL e MySQL

Use a função LPAD para gerar as strings de caracteres necessárias \*:

```
1 select deptno,
2       lpad('*',count(*),'*') as cnt
3   from emp
4  group by deptno
```

#### SQL Server

Gere o histograma usando a função REPLICATE:

```
1 select deptno,
2       replicate('*',count(*)) cnt
3   from emp
4  group by deptno
```

## Discussão

A técnica é a mesma para todos os fornecedores. A única diferença está na função string usada para retornar um \* para cada funcionário. A solução Oracle será utilizada para esta discussão, mas a explicação é relevante para todas as soluções.

O primeiro passo é contar o número de funcionários em cada departamento:

```
select deptno,
       count(*)
  from emp
 group by deptno
```

DEPTNO	COUNT(*)
10	3
20	5
30	6

A próxima etapa é usar o valor retornado por COUNT para controlar o número de caracteres \* a serem retornados para cada departamento. Simplesmente passe COUNT(\*) como argumento para a função de string LPAD para retornar o número desejado de \*:

```
select deptno,
       lpad('*',count(*),'*') as cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	***
20	*****
30	*****

Para usuários do PostgreSQL, pode ser necessário usar CAST para garantir que COUNT(\*) retorne um número inteiro conforme mostrado aqui:

```
select deptno,
       lpad('*',count(*)::integer,'*') as cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	***
20	*****
30	*****

Este CAST é necessário porque o PostgreSQL exige que o argumento numérico do LPAD seja um número inteiro.

## 12.10 Criação de histogramas verticais

### Problema

Você deseja gerar um histograma que cresça de baixo para cima. Por exemplo, você deseja exibir o número de funcionários em cada departamento como um histograma vertical com cada funcionário representado por uma instância de \*. Você deseja retornar o seguinte conjunto de resultados:

```
D10 D20 D30
-----
*   *
*   *
*   *
*   *   *
*   *   *
*   *   *
```

### Solução

A técnica usada para resolver esse problema baseia-se em uma técnica usada anteriormente neste capítulo: use a função ROW\_NUMBER OVER para identificar exclusivamente cada instância de \* para cada DEPTNO. Use a função agregada MAX para dinamizar o conjunto de resultados e agrupar pelos valores retornados por ROW\_NUMBER OVER (os usuários do SQL Server não devem usar DESC na cláusula ORDER BY):

```
1 select max(deptno_10) d10,
2       max(deptno_20) d20,
3       max(deptno_30) d30
4   from (
5 select row_number()over(partition by deptno order by empno) rn,
6       case when deptno=10 then '*' else null end deptno_10,
7       case when deptno=20 then '*' else null end deptno_20,
8       case when deptno=30 then '*' else null end deptno_30
9   from emp
10  ) x
11 group by rn
12 order by 1 desc, 2 desc, 3 desc
```

### Discussão

A primeira etapa é usar a função de window ROW\_NUMBER para identificar exclusivamente cada instância de \* em cada departamento. Use uma expressão CASE para retornar um \* para cada funcionário em cada departamento:

```

select row_number()over(partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
  from emp

RN DEPTNO_10  DEPTNO_20  DEPTNO_30
-----
1 *
2 *
3 *
1   *
2   *
3   *
4   *
5   *
1           *
2           *
3           *
4           *
5           *
6           *

```

A próxima e última etapa é usar a função agregada MAX em cada expressão CASE, agrupando por RN para remover os NULLs do conjunto de resultados. Ordene os resultados ASC ou DESC dependendo de como seu RDBMS classifica NULLs:

```

select max(deptno_10) d10,
       max(deptno_20) d20,
       max(deptno_30) d30
  from (
select row_number()over(partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
  from emp
      ) x
 group by rn
order by 1 desc, 2 desc, 3 desc

D10 D20 D30
-----
*   *
*   *
*   *
*   *   *
*   *   *
*   *   *

```

## 12.11 Retornando colunas não GROUP BY

### Problema

Você está executando uma consulta GROUP BY e deseja retornar colunas em sua lista de seleção que também não estão listadas em sua cláusula GROUP BY. Normalmente, isso não é possível, pois essas colunas desagrupadas não representariam um único valor por linha.

Digamos que você queira encontrar os funcionários que ganham os salários mais altos e mais baixos em cada departamento, bem como os funcionários que ganham os salários mais altos e mais baixos em cada cargo. Você deseja ver o nome de cada funcionário, o departamento em que trabalha, seu cargo e seu salário. Você deseja retornar o seguinte conjunto de resultados:

DEPTNO	ENAME	JOB	SAL	DEPT_STATUS	JOB_STATUS
10	MILLER	CLERK	1300	LOW SAL IN DEPT	TOP SAL IN JOB
10	CLARK	MANAGER	2450		LOW SAL IN JOB
10	KING	PRESIDENT	5000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SCOTT	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	FORD	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SMITH	CLERK	800	LOW SAL IN DEPT	LOW SAL IN JOB
20	JONES	MANAGER	2975		TOP SAL IN JOB
30	JAMES	CLERK	950	LOW SAL IN DEPT	
30	MARTIN	SALESMAN	1250		LOW SAL IN JOB
30	WARD	SALESMAN	1250		LOW SAL IN JOB
30	ALLEN	SALESMAN	1600		TOP SAL IN JOB
30	BLAKE	MANAGER	2850	TOP SAL IN DEPT	

Infelizmente, incluir todas essas colunas na cláusula SELECT arruinará o agrupamento. Considere o seguinte exemplo: o funcionário KING ganha o salário mais alto. Você deseja verificar isso com a seguinte consulta:

```
select ename,max(sal)
  from empgroup by ename
```

Em vez de ver KING e o salário de KING, a consulta anterior retornará todas as 14 linhas da tabela EMP. O motivo é por causa do agrupamento: o MAX(SAL) é aplicado a cada ENAME. Então, parece que a consulta anterior pode ser declarada como “encontre o funcionário com o salário mais alto”, mas na verdade o que está fazendo é “encontre o salário mais alto para cada ENAME na tabela EMP”. Esta receita explica uma técnica para incluir ENAME sem a necessidade de GROUP BY essa coluna.

### Solução

Use uma visualização inline para encontrar os salários altos e baixos por DEPTNO e JOB. Então fique apenas com os funcionários que recebem esses salários.

Use as funções de window MAX OVER e MIN OVER para encontrar os salários mais altos e mais baixos por DEPTNO e JOB. Em seguida, mantenha as linhas onde os salários são aqueles que são maiores ou menores por DEPTNO ou JOB:

```

1 select deptno,ename,job,sal,
2      case when sal = max_by_dept
3          then 'TOP SAL IN DEPT'
4          when sal = min_by_dept
5          then 'LOW SAL IN DEPT'
6      end dept_status,
7      case when sal = max_by_job
8          then 'TOP SAL IN JOB'
9          when sal = min_by_job
10         then 'LOW SAL IN JOB'
11     end job_status
12   from (
13 select deptno,ename,job,sal,
14      max(sal)over(partition by deptno) max_by_dept,
15      max(sal)over(partition by job)    max_by_job,
16      min(sal)over(partition by deptno) min_by_dept,
17      min(sal)over(partition by job)    min_by_job
18   from emp
19      ) emp_sals
20 where sal in (max_by_dept,max_by_job,
21                 min_by_dept,min_by_job)

```

## Discussão

O primeiro passo é utilizar as funções de window MAX OVER e MIN OVER para encontrar os maiores e menores salários por DEPTNO e JOB:

```

select deptno,ename,job,sal,
       max(sal)over(partition by deptno) maxDEPT,
       max(sal)over(partition by job)    maxJOB,
       min(sal)over(partition by deptno) minDEPT,
       min(sal)over(partition by job)    minJOB
  from emp

```

DEPTNO	ENAME	JOB	SAL	MAXDEPT	MAXJOB	MINDEPT	MINJOB
10	MILLER	CLERK	1300	5000	1300	1300	800
10	CLARK	MANAGER	2450	5000	2975	1300	2450
10	KING	PRESIDENT	5000	5000	5000	1300	5000
20	SCOTT	ANALYST	3000	3000	3000	800	3000
20	FORD	ANALYST	3000	3000	3000	800	3000
20	SMITH	CLERK	800	3000	1300	800	800
20	JONES	MANAGER	2975	3000	2975	800	2450
20	ADAMS	CLERK	1100	3000	1300	800	800
30	JAMES	CLERK	950	2850	1300	950	800
30	MARTIN	SALESMAN	1250	2850	1600	950	1250
30	TURNER	SALESMAN	1500	2850	1600	950	1250
30	WARD	SALESMAN	1250	2850	1600	950	1250

30 ALLEN	SALESMAN	1600	2850	1600	950	1250
30 BLAKE	MANAGER	2850	2850	2975	950	2450

Neste ponto, cada salário pode ser comparado com os salários mais altos e mais baixos por DEPTNO e JOB. Observe que o agrupamento (a inclusão de múltiplas colunas na cláusula SELECT) não afeta os valores retornados por MIN OVER e MAX OVER. Esta é a beleza das funções de window: a agregação é calculada sobre um “grupo” ou partição definida e retorna múltiplas linhas para cada grupo. A última etapa é simplesmente agrupar as funções da window em uma visualização embutida e manter apenas as linhas que correspondem aos valores retornados pelas funções da window. Use uma expressão simples CASE para exibir o “status” de cada funcionário no conjunto de resultados final:

```

select deptno,ename,job,sal,
       case when sal = max_by_dept
             then 'TOP SAL IN DEPT'
             when sal = min_by_dept
             then 'LOW SAL IN DEPT'
       end dept_status,
       case when sal = max_by_job
             then 'TOP SAL IN JOB'
             when sal = min_by_job
             then 'LOW SAL IN JOB'
       end job_status
  from (
select deptno,ename,job,sal,
       max(sal)over(partition by deptno) max_by_dept,
       max(sal)over(partition by job) max_by_job,
       min(sal)over(partition by deptno) min_by_dept,
       min(sal)over(partition by job) min_by_job
  from emp
   ) x
 where sal in (max_by_dept,max_by_job,
               min_by_dept,min_by_job)

```

DEPTNO	ENAME	JOB	SAL	DEPT_STATUS	JOB_STATUS
10	MILLER	CLERK	1300	LOW SAL IN DEPT	TOP SAL IN JOB
10	CLARK	MANAGER	2450		LOW SAL IN JOB
10	KING	PRESIDENT	5000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SCOTT	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	FORD	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SMITH	CLERK	800	LOW SAL IN DEPT	LOW SAL IN JOB
20	JONES	MANAGER	2975		TOP SAL IN JOB
30	JAMES	CLERK	950	LOW SAL IN DEPT	
30	MARTIN	SALESMAN	1250		LOW SAL IN JOB
30	WARD	SALESMAN	1250		LOW SAL IN JOB
30	ALLEN	SALESMAN	1600		TOP SAL IN JOB
30	BLAKE	MANAGER	2850	TOP SAL IN DEPT	

## 12.12 Cálculo de subtotais simples

### Problema

Para efeitos desta receita, um *subtotal simples* é definido como um conjunto de resultados que contém valores da agregação de uma coluna junto com um valor total geral da tabela. Um exemplo seria um conjunto de resultados que soma os salários da tabela EMP por JOB e que também inclui a soma de todos os salários da tabela EMP. Os salários somados por JOB são os subtotais, e a soma de todos os salários na tabela EMP é o total geral. Esse conjunto de resultados deve ter a seguinte aparência:

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

### Solução

A extensão ROLLUP para a cláusula GROUP BY resolve este problema perfeitamente. Se ROLLUP não estiver disponível para seu RDBMS, você poderá resolver o problema, embora com mais dificuldade, usando uma subconsulta escalar ou uma consulta UNION.

### DB2 e Oracle

Use a função agregada SUM para somar os salários e use a extensão ROLLUP de GROUP BY para organizar os resultados em subtotais (por JOB) e um total geral (para toda a tabela):

```

1 select case grouping(job)
2           when 0 then job
3           else 'TOTAL'
4       end job,
5       sum(sal) sal
6   from emp
7 group by rollup(job)

```

### SQL Server e MySQL

Use a função agregada SUM para somar os salários e use WITH ROLLUP para organizar os resultados em subtotais (por JOB) e um total geral (para toda a tabela). Em seguida, use COALESCE para fornecer o rótulo TOTAL para a linha do total geral (que de outra forma terá um NULL na coluna JOB):

```
1 select coalesce(job,'TOTAL') job,
2       sum(sal) sal
3   from emp
4 group by job with rollup
```

Com o SQL Server, você também tem a opção de usar a função GROUPING mostrada na receita Oracle/DB2 em vez de COALESCE para determinar o nível de agregação.

## PostgreSQL

Semelhante às soluções SQL Server e MySQL, você usa a extensão ROLLUP para GROUP BY com sintaxe ligeiramente diferente:

```
select coalesce(job,'TOTAL') job,
       sum(sal) sal
     from emp
    group by rollup(job)
```

## Discussão

### DB2 e Oracle

O primeiro passo é utilizar a função agregada SUM, agrupando por JOB para somar os salários por JOB:

```
select job, sum(sal) sal
      from emp
     group by job
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

O próximo passo é usar a extensão ROLLUP para GROUP BY para produzir um total geral para todos os salários junto com os subtotais para cada JOB:

```
select job, sum(sal) sal
      from emp
     group by rollup(job)
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
	29025

A última etapa é usar a função GROUPING na coluna JOB para exibir um rótulo para o total geral. Se o valor de JOB for NULL, a função GROUPING retornará 1, o que significa que o valor de SAL é o total geral criado por ROLLUP. Se o valor de JOB não for NULL, a função GROUPING retornará 0, o que significa que o valor de SAL é o resultado de GROUP BY, não de ROLLUP. Envolva a chamada para GROUPING(JOB) em uma expressão CASE que retorne o nome do trabalho ou o rótulo TOTAL, conforme apropriado:

```
select case grouping(job)
        when 0 then job
        else 'TOTAL'
      end job,
        sum(sal) sal
  from emp
 group by rollup(job)
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

## SQL Server e MySQL

O primeiro passo é utilizar a função agregada SUM, agrupando os resultados por JOB para gerar somas salariais por JOB:

```
select job, sum(sal) sal
  from emp
 group by job
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

A próxima etapa é usar a extensão ROLLUP do GROUP BY para produzir um total geral para todos os salários junto com os subtotais para cada JOB:

```
select job, sum(sal) sal
  from emp
 group by job with rollup
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
	29025

A última etapa é usar a função COALESCE na coluna JOB. Se o valor de JOB for NULL, o valor de SAL será o total geral criado por ROLLUP. Se o valor de JOB não for NULL, o valor de SAL será o resultado do GROUP BY “normal”, não do ROLLUP:

```
select coalesce(job,'TOTAL') job,
       sum(sal) sal
  from emp
 group by job with rollup
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

### PostgreSQL

A solução é igual em seu modo de operação à solução anterior para MySQL e SQL Server. A única diferença é a sintaxe da cláusula ROLLUP: escreva ROLLUP(JOB) após GROUP BY.

## 12.13 Cálculo de subtotais para todas as combinações de expressões possíveis

### Problema

Você deseja encontrar a soma de todos os salários por DEPTNO e por JOB, para cada combinação JOB/ DEPTNO. Você também deseja um total geral para todos os salários na tabela EMP. Você deseja retornar o seguinte conjunto de resultados:

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900

```

30 CLERK      TOTAL BY DEPT AND JOB      950
30 SALESMAN   TOTAL BY DEPT AND JOB      5600
30 MANAGER    TOTAL BY DEPT AND JOB      2850
20 MANAGER    TOTAL BY DEPT AND JOB      2975
20 ANALYST   TOTAL BY DEPT AND JOB      6000
      CLERK    TOTAL BY JOB             4150
      ANALYST   TOTAL BY JOB             6000
      MANAGER    TOTAL BY JOB             8275
      PRESIDENT  TOTAL BY JOB             5000
      SALESMAN   TOTAL BY JOB             5600
10          TOTAL BY DEPT              8750
30          TOTAL BY DEPT              9400
20          TOTAL BY DEPT              10875

GRAND TOTAL FOR TABLE   29025

```

## Solução

Extensões adicionadas ao GROUP BY nos últimos anos tornam esse problema bastante fácil de resolver. Se sua plataforma não fornecer tais extensões para calcular vários níveis de subtotais, você deverá calculá-los manualmente (por meio de autojunções ou subconsultas escalares).

### DB2

Para DB2, você precisará usar CAST para retornar de GROUPING como o tipo de dados CHAR(1):

```

1 select deptno,
2       job,
3       case cast(grouping(deptno) as char(1)) ||
4           cast(grouping(job) as char(1))
5           when '00' then 'TOTAL BY DEPT AND JOB'
6           when '10' then 'TOTAL BY JOB'
7           when '01' then 'TOTAL BY DEPT'
8           when '11' then 'TOTAL FOR TABLE'
9       end category,
10      sum(sal)
11  from emp
12 group by cube(deptno,job)
13 order by grouping(job),grouping(deptno)

```

### Oracle

Use a extensão CUBE para a cláusula GROUP BY com o operador de concatenação ||:

```

1 select deptno,
2       job,
3       case grouping(deptno)||grouping(job)
4           when '00' then 'TOTAL BY DEPT AND JOB'
5           when '10' then 'TOTAL BY JOB'
1 when '01' then 'TOTAL BY DEPT '

```

```
6      when '11' then 'GRAND TOTALFOR TABLE'
7  end category,
8  sum(sal) sal
9  from emp
10 group by cube(deptno,job)
11 order by grouping(job),grouping(deptno)
```

## SQL Server

Use a extensão CUBE para a cláusula GROUP BY. Para SQL Server, você precisará CAST os resultados de GROUPING para CHAR(1) e usará o operador + para concatenação (em oposição ao operador Oracle ||):

```
1 select deptno,
2       job,
3       case cast(grouping(deptno)as char(1))+
4             cast(grouping(job)as char(1))
5             when '00' then 'TOTAL BY DEPT AND JOB'
6             when '10' then 'TOTAL BY JOB'
7             when '01' then 'TOTAL BY DEPT'
8             when '11' then 'GRAND TOTAL FOR TABLE'
9       end category,
10      sum(sal) sal
11  from emp
12 group by deptno,job with cube
13 order by grouping(job),grouping(deptno)
```

## PostgreSQL

PostgreSQL é semelhante ao anterior, mas com sintaxe ligeiramente diferente para o operador CUBE e a concatenação:

```
select deptno,job
,case concat(
cast (grouping(deptno) as char(1)),cast (grouping(job) as char(1))
)
when '00' then 'TOTAL BY DEPT AND JOB'
      when '10' then 'TOTAL BY JOB'
      when '01' then 'TOTAL BY DEPT'
      when '11' then 'GRAND TOTAL FOR TABLE'
end category
,sum(sal) as sal
from emp
group by cube(deptno,job)
```

## MySQL

Embora parte da funcionalidade esteja disponível, ela não está completa, pois o MySQL não possui a função CUBE. Portanto, use vários UNION ALLs, criando somas diferentes para cada um:

```

1 select deptno, job,
2       'TOTAL BY DEPT AND JOB' as category,
3       sum(sal) as sal
4   from emp
5 group by deptno, job
6 union all
7 select null, job, 'TOTAL BY JOB', sum(sal)
8   from emp
9 group by job
10 union all
11 select deptno, null, 'TOTAL BY DEPT', sum(sal)
12   from emp
13 group by deptno
14 union all
15 select null,null,'GRAND TOTAL FOR TABLE', sum(sal)
16   from emp

```

## Discussão

### Oracle, DB2 e SQL Server

As soluções para todos os três são essencialmente as mesmas. O primeiro passo é usar a função agregada SUM e agrupar por DEPTNO e JOB para encontrar os salários totais para cada combinação JOB e DEPTNO:

```

select deptno, job, sum(sal) sal
  from emp
 group by deptno, job

```

DEPTNO	JOB	SAL
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

A próxima etapa é criar subtotais por JOB e DEPTNO junto com o total geral de toda a tabela. Utilize a extensão CUBE para a cláusula GROUP BY para realizar agregações em SAL por DEPTNO, JOB e para toda a tabela:

```

select deptno,
       job,
       sum(sal) sal
  from emp
 group by cube(deptno,job)

```

DEPTNO	JOB	SAL
		29025
	CLERK	4150
	ANALYST	6000
	MANAGER	8275
	SALESMAN	5600
	PRESIDENT	5000
10		8750
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20		10875
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30		9400
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

A seguir, use a função GROUPING em conjunto com CASE para formatar os resultados em uma saída mais significativa. O valor de GROUPING (JOB) será 1 ou 0 dependendo se os valores de SAL são devidos ao GROUP BY ou ao CUBE. Se os resultados forem devidos ao CUBO, o valor será 1; caso contrário, será 0. O mesmo vale para GROUPING (DEPTNO). Olhando para o primeiro passo da solução, você verá que o agrupamento é feito por DEPTNO e JOB. Assim, os valores esperados das chamadas para GROUPING quando uma linha representa uma combinação de DEPTNO e JOB são 0. A consulta a seguir confirma isso:

```
select deptno,
       job,
       grouping(deptno) is_deptno_subtotal,
       grouping(job) is_job_subtotal,
       sum(sal) sal
  from emp
 group by cube(deptno,job)
 order by 3,4
```

DEPTNO	JOB	IS_DEPTNO_SUBTOTAL	IS_JOB_SUBTOTAL	SAL
10	CLERK	0	0	1300
10	MANAGER	0	0	2450
10	PRESIDENT	0	0	5000
20	CLERK	0	0	1900
30	CLERK	0	0	950
30	SALESMAN	0	0	5600
30	MANAGER	0	0	2850
20	MANAGER	0	0	2975
20	ANALYST	0	0	6000
10		0	1	8750

20	0	1	10875
30	0	1	9400
CLERK	1	0	4150
ANALYST	1	0	6000
MANAGER	1	0	8275
PRESIDENT	1	0	5000
SALESMAN	1	0	5600
	1	1	29025

A etapa final é usar uma expressão CASE para determinar a qual categoria cada linha pertence com base nos valores retornados por GROUPING (JOB) e GROUPING (DEPTNO) concatenados:

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by cube(deptno,job)
 order by grouping(job),grouping(deptno)
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
30	MANAGER	TOTAL BY DEPT AND JOB	2850
20	MANAGER	TOTAL BY DEPT AND JOB	2975
20	ANALYST	TOTAL BY DEPT AND JOB	6000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
30		TOTAL BY DEPT	9400
20		TOTAL BY DEPT	10875
		GRAND TOTAL FOR TABLE	29025

Esta solução Oracle converte implicitamente os resultados das funções GROUPING em um tipo de caractere em preparação para concatenar os dois valores. Os usuários do DB2 e do SQL Server precisarão converter explicitamente os resultados das funções GROUPING para CHAR (1), conforme mostrado na solução.

Além disso, os usuários do SQL Server devem usar o operador +, e não o operador ||, para concatenar os resultados das duas chamadas GROUPING em uma string.

Para usuários Oracle e DB2, existe uma extensão adicional para GROUP BY chamada GROUPING SETS; esta extensão é extremamente útil. Por exemplo, você pode usar GROUPING SETS para imitar a saída criada por CUBE conforme mostrado aqui (usuários de DB2 e SQL Server precisarão usar CAST para garantir que os valores retornados pela função GROUPING estejam no formato correto da mesma maneira que em a solução CUBO):

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by grouping sets ((deptno),(job),(deptno,job),())
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400
		GRAND TOTAL FOR TABLE	29025

O que é ótimo em GROUPING SETS é que ele permite definir os grupos. A cláusula GROUPING SETS na consulta anterior faz com que grupos sejam criados por DEPTNO, por JOB e pela combinação de DEPTNO e JOB e, finalmente, os parênteses vazios solicitam um total geral. GROUPING SETS oferece enorme flexibilidade para a criação de relatórios com diferentes níveis de agregação; por exemplo, se você quiser modificar o exemplo anterior para excluir GRAND TOTAL, simplesmente modifique a cláusula GROUPING SETS excluindo os parênteses vazios:

```

/* sem total geral */

select deptno,
       job,
       case grouping(deptno)||grouping(job)
           when '00' then 'TOTAL BY DEPT AND JOB'
           when '10' then 'TOTAL BY JOB'
           when '01' then 'TOTAL BY DEPT'
           when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by grouping sets ((deptno),(job),(deptno,job))

```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400

Você também pode eliminar um subtotal, como o de DEPTNO, simplesmente omitindo (DEPTNO) da cláusula GROUPING SETS:

```

/* nosubtotais por DEPTNO */

select deptno,
       job,
       case grouping(deptno)||grouping(job)
           when '00' then 'TOTAL BY DEPT AND JOB'
           when '10' then 'TOTAL BY JOB'
           when '01' then 'TOTAL BY DEPT'
           when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by grouping sets ((job),(deptno,job),())
 order by 3

```

DEPTNO	JOB	CATEGORY	SAL
<hr/>			
		GRAND TOTAL FOR TABLE	29025
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
	CLERK	TOTAL BY JOB	4150
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
	MANAGER	TOTAL BY JOB	8275
	ANALYST	TOTAL BY JOB	6000

Como você pode ver, GROUPING SETS torna muito fácil brincar com totais e subtotais para observar seus dados de diferentes ângulos.

## MySQL

O primeiro passo é usar a função agregada SUM e agrupar por DEPTNO e JOB:

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
```

DEPTNO	JOB	CATEGORY	SAL
<hr/>			
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600

A próxima etapa é usar UNION ALL para anexar somas TOTAL BY JOB:

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
union all
```

```
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600

O próximo passo é UNION ALL a soma de todos os salários pelo DEPTNO:

```
select deptno, job,
      'TOTAL BY DEPT AND JOB' as category,
      sum(sal) as sal
    from emp
   group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
 union all
select deptno, null, 'TOTAL BY DEPT', sum(sal)
  from emp
 group by deptno
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600

```

10      TOTAL BY DEPT          8750
20      TOTAL BY DEPT          10875
30      TOTAL BY DEPT          9400

```

A etapa final é usar UNION ALL para anexar a soma de todos os salários:

```

13      select deptno, job,
14          'TOTAL BY DEPT AND JOB' as category,
15          sum(sal) as sal
16      from emp
17      group by deptno, job
18      union all
19      select null, job, 'TOTAL BY JOB',
20          sum(sal) from emp
21      group by job
22      union all
23      select deptno, null, 'TOTAL BY DEPT',
24          sum(sal) from emp
25      group by
26          deptno union
27      all
28      select null,null, 'GRAND TOTAL FOR TABLE',
29          sum(sal) from emp

```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400
		GRAND TOTAL FOR TABLE	29025

## 22.12 Identificando linhas que não são subtotais

### Problema

Você usou a extensão CUBE da cláusula GROUP BY para criar um relatório e precisa diferenciar as linhas que seriam geradas por um relatório normal.

cláusula GROUP BY e as linhas que foram geradas como resultado do uso de CUBE ou ROLLUP.

A seguir está o conjunto de resultados de uma consulta usando a extensão CUBE para GROUP BY para criar um detalhamento dos salários na tabela EMP:

DEPTNO	JOB	SAL
		29025
	CLERK	4150
	ANALYST	6000
	MANAGER	8275
	SALESMAN	5600
	PRESIDENT	5000
10		8750
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20		10875
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30		9400
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

Este relatório inclui a soma de todos os salários por DEPTNO e JOB (para cada JOB por DEPTNO), a soma de todos os salários por DEPTNO, a soma de todos os salários por JOB e, finalmente, um total geral (a soma de todos os salários na tabela EMP). Você deseja identificar claramente os diferentes níveis de agregação. Você deseja identificar a qual categoria um valor agregado pertence (ou seja, um determinado valor na coluna SAL representa um total por DEPTNO? Por JOB? O total geral?). Você gostaria de retornar o seguinte conjunto de resultados:

DEPTNO	JOB	SAL	DEPTNO_SUBTOTALS	JOB_SUBTOTALS
		29025	1	1
	CLERK	4150	1	0
	ANALYST	6000	1	0
	MANAGER	8275	1	0
	SALESMAN	5600	1	0
	PRESIDENT	5000	1	0
10		8750	0	1
10	CLERK	1300	0	0
10	MANAGER	2450	0	0
10	PRESIDENT	5000	0	0
20		10875	0	1
20	CLERK	1900	0	0
20	ANALYST	6000	0	0
20	MANAGER	2975	0	0

30	9400	0	1
30 CLERK	950	0	0
30 MANAGER	2850	0	0
30 SALESMAN	5600	0	0

## Solução

Use a função GROUPING para identificar quais valores existem devido à criação de subtotais ou valores por CUBE ou ROLLUP *super agregados*. A seguir está um exemplo para PostgreSQL, DB2 e Oracle:

```

1 select deptno, jo) sal,
2      grouping(deptno) deptno_subtotals,
3      grouping(job) job_subtotals
4  from emp
5 group by cube(deptno,job)

```

A única diferença entre a solução SQL Server e aquela para DB2 e Oracle está em como as cláusulas CUBE/ROLLUP são escritas:

```

1 select deptno, job, sum(sal) sal,
2      grouping(deptno) deptno_subtotals,
3      grouping(job) job_subtotals
4  from emp
5 group by deptno,job with cube

```

Esta receita pretende destacar o uso de CUBE e GROUPING ao trabalhar com subtotais. No momento em que este livro foi escrito, o MySQL não suportava CUBE ou GROUPING.

## Discussão

Se DEPTNO\_SUBTOTALS for 0 e JOB\_SUBTOTALS for 1 (nesse caso JOB é NULL), o valor de SAL representa um subtotal de salários por DEPTNO criado por CUBE. Se JOB\_SUBTOTALS for 0 e DEPTNO\_SUBTOTALS for 1 (nesse caso DEPTNO é NULL), o valor de SAL representa um subtotal de salários por JOB criado pelo CUBE. Linhas com 0 para DEPTNO\_SUBTOTALS e JOB\_SUBTOTALS representam linhas criadas por agregação regular (a soma de SAL para cada combinação DEPTNO/JOB).

## 12.15 Usando expressões case para sinalizar linhas

### Problema

Você deseja mapear os valores em uma coluna, talvez a coluna EMP JOB da tabela, em uma série de sinalizadores “Booleanos”. Por exemplo, você deseja retornar o seguinte conjunto de resultados:

ENAME	IS_CLERK	IS_SALES	IS_MGR	IS_ANALYST	IS_PREZ
KING	0	0	0	0	1
SCOTT	0	0	0	1	0
FORD	0	0	0	1	0
JONES	0	0	1	0	0
BLAKE	0	0	1	0	0
CLARK	0	0	1	0	0
ALLEN	0	1	0	0	0
WARD	0	1	0	0	0
MARTIN	0	1	0	0	0
TURNER	0	1	0	0	0
SMITH	1	0	0	0	0
MILLER	1	0	0	0	0
ADAMS	1	0	0	0	0
JAMES	1	0	0	0	0

Esse conjunto de resultados pode ser útil para depuração e para fornecer uma visualização dos dados diferente daquela que você veria em um conjunto de resultados mais típico.

## Solução

Use uma expressão CASE para avaliar o JOB de cada funcionário e retorne 1 ou 0 para indicar seu JOB. Você precisará escrever uma expressão CASE e, assim, criar uma coluna para cada trabalho possível:

```

1 select ename,
2      case when job = 'CLERK'
3            then 1 else 0
4      end as is_clerk,
5      case when job = 'SALESMAN'
6            then 1 else 0
7      end as is_sales,
8      case when job = 'MANAGER'
9            then 1 else 0
10     end as is_mgr,
11     case when job = 'ANALYST'
12           then 1 else 0
13     end as is_analyst,
14     case when job = 'PRESIDENT'
15           then 1 else 0
16     end as is_pres
17   from emp
18  order by 2,3,4,5,6

```

## Discussão

O código da solução é praticamente autoexplicativo. Se você estiver tendo problemas para entender, basta adicionar JOB à cláusula SELECT:

```
select ename,
       job,
       case when job = 'CLERK'
             then 1 else 0
         end as is_clerk,
       case when job = 'SALESMAN'
             then 1 else 0
         end as is_sales,
       case when job = 'MANAGER'
             then 1 else 0
         end as is_mgr,
       casewhen job = 'ANALYST'
             then 1 else 0
         end as is_analyst,
       casewhen job = 'PRESIDENT'
             then 1 else 0
         end as is_pres
  from emp
 order by 2
```

ENAME	JOB	IS_CLERK	IS_SALES	IS_MGR	IS_ANALYST	IS_PREZ
SCOTT	ANALYST	0	0	0	1	0
FORD	ANALYST	0	0	0	1	0
SMITH	CLERK	1	0	0	0	0
ADAMS	CLERK	1	0	0	0	0
MILLER	CLERK	1	0	0	0	0
JAMES	CLERK	1	0	0	0	0
JONES	MANAGER	0	0	1	0	0
CLARK	MANAGER	0	0	1	0	0
BLAKE	MANAGER	0	0	1	0	0
KING	PRESIDENT	0	0	0	0	1
ALLEN	SALESMAN	0	1	0	0	0
MARTIN	SALESMAN	0	1	0	0	0
TURNER	SALESMAN	0	1	0	0	0
WARD	SALESMAN	0	1	0	0	0

## 12.16 Criando uma matriz esparsa

### Problema

Você deseja criar uma matriz esparsa, como a seguinte transpondo as colunas DEPTNO e JOB da tabela EMP:

D10	D20	D30	CLERKS	MGRS	PREZ	ANALYS	SALES
		SMITH		SMITH			
			ALLEN			ALLEN	
			WARD			WARD	
		JONES			JONES		
			MARTIN			MARTIN	
			BLAKE		BLAKE		
CLARK					CLARK		
		SCOTT				SCOTT	
KING					KING		
			TURNER			TURNER	
		ADAMS		ADAMS			
			JAMES	JAMES			
		FORD				FORD	
MILLER					MILLER		

## Solução

Use expressões CASE para criar uma transformação esparsa de linha para coluna:

```

1 select case deptno when 10 then ename end as d10,
2      case deptno when 20 then ename end as d20,
3      case deptno when 30 then ename end as d30,
4      case job when 'CLERK' then ename end as clerks,
5      case job when 'MANAGER' then ename end as mgrs,
6      case job when 'PRESIDENT' then ename end as prez,
7      case job when 'ANALYST' then ename end as analys,
8      case job when 'SALESMAN' then ename end as sales
9  from emp

```

## Discussão

Para transformar as linhas DEPTNO e JOB em colunas, basta utilizar uma expressão CASE para avaliar os possíveis valores retornados por essas linhas. Isso é tudo que há para fazer. Além disso, se você quiser “densificar” o relatório e se livrar de algumas dessas linhas NULL, você precisará encontrar algo para agrupar. Por exemplo, use a função de janela ROW\_NUMBER OVER para atribuir uma classificação para cada funcionário por DEPTNO e, em seguida, use a função agregada MAX para eliminar alguns dos NULLs:

```

select max(case deptno when 10 then ename end) d10,
       max(case deptno when 20 then ename end) d20,
       max(case deptno when 30 then ename end) d30,
       max(case job when 'CLERK' then ename end) clerks,
       max(case job when 'MANAGER' then ename end) mgrs,
       max(case job when 'PRESIDENT' then ename end) prez,
       max(case job when 'ANALYST' then ename end) analys,
       max(case job when 'SALESMAN' then ename end) sales
  from (
select deptno, job, ename,
       row_number()over(partition by deptno order by empno) rn

```

```
from emp
  ) x
group by rn
```

D10	D20	D30	CLERKS	MGRS	PREZ	ANALYS	SALES
CLARK	SMITH	ALLEN	SMITH	CLARK			ALLEN
KING	JONES	WARD		JONES	KING		WARD
MILLER	SCOTT	MARTIN	MILLER		SCOTT		MARTIN
	ADAMS	BLAKE	ADAMS	BLAKE			
	FORD	TURNER			FORD		TURNER
		JAMES	JAMES				

## 12.17 Agrupando linhas por unidades de tempo

### Problema

Você deseja resumir os dados por algum intervalo de tempo. Por exemplo, você tem um log de transações e deseja resumir as transações em intervalos de cinco segundos. As linhas na tabela TRX\_LOG são mostradas aqui:

```
select trx_id,
       trx_date,
       trx_cnt
  from trx_log
   TRX_ID TRX_DATE           TRX_CNT
-----
  1 28-JUL-2020 19:03:07      44
  2 28-JUL-2020 19:03:08      18
  3 28-JUL-2020 19:03:09      23
  4 28-JUL-2020 19:03:10      29
  5 28-JUL-2020 19:03:11      27
  6 28-JUL-2020 19:03:12      45
  7 28-JUL-2020 19:03:13      45
  8 28-JUL-2020 19:03:14      32
  9 28-JUL-2020 19:03:15      41
 10 28-JUL-2020 19:03:16      15
 11 28-JUL-2020 19:03:17      24
 12 28-JUL-2020 19:03:18      47
 13 28-JUL-2020 19:03:19      37
 14 28-JUL-2020 19:03:20      48
 15 28-JUL-2020 19:03:21      46
 16 28-JUL-2020 19:03:22      44
 17 28-JUL-2020 19:03:23      36
 18 28-JUL-2020 19:03:24      41
 19 28-JUL-2020 19:03:25      33
 20 28-JUL-2020 19:03:26      19
```

Você deseja retornar o seguinte conjunto de resultados:

GRP	TRX_START	TRX_END	TOTAL
1	28-JUL-2020 19:03:07	28-JUL-2020 19:03:11	141
2	28-JUL-2020 19:03:12	28-JUL-2020 19:03:16	178
3	28-JUL-2020 19:03:17	28-JUL-2020 19:03:21	202
4	28-JUL-2020 19:03:22	28-JUL-2020 19:03:26	173

## Solução

Agrupe as entradas em cinco grupos de linhas. Existem diversas maneiras de realizar esse agrupamento lógico; esta receita faz isso dividindo os valores TRX\_ID por cinco, usando uma técnica mostrada anteriormente em [Receita 12.7](#).

Depois de criar os “grupos”, use as funções agregadas MIN, MAX e SUM para encontrar a hora de início, a hora de término e o número total de transações para cada “grupo” (os usuários do SQL Server devem usar CEILING em vez de CEIL) :

```

1 select ceil(trx_id/5.0) as grp,
2      min(trx_date)    as trx_start,
3      max(trx_date)    as trx_end,
4      sum(trx_cnt)     as total
5  from trx_log
6 group by ceil(trx_id/5.0)

```

## Discussão

O primeiro passo, e a chave para toda a solução, é agrupar logicamente as linhas. Ao dividir por cinco e considerar o menor número inteiro maior que o quociente, você pode criar grupos lógicos. Por exemplo:

```

select trx_id,
       trx_date,
       trx_cnt,
       trx_id/5.0 as val,
       ceil(trx_id/5.0) as grp
  from trx_log

```

TRX_ID	TRX_DATE	TRX_CNT	VAL	GRP
1	28-JUL-2020 19:03:07	44	.20	1
2	28-JUL-2020 19:03:08	18	.40	1
3	28-JUL-2020 19:03:09	23	.60	1
4	28-JUL-2020 19:03:10	29	.80	1
5	28-JUL-2020 19:03:11	27	1.00	1
6	28-JUL-2020 19:03:12	45	1.20	2
7	28-JUL-2020 19:03:13	45	1.40	2
8	28-JUL-2020 19:03:14	32	1.60	2
9	28-JUL-2020 19:03:15	41	1.80	2
10	28-JUL-2020 19:03:16	15	2.00	2
11	28-JUL-2020 19:03:17	24	2.20	3
12	28-JUL-2020 19:03:18	47	2.40	3
13	28-JUL-2020 19:03:19	37	2.60	3

14	28-JUL-2020	19:03:20	48	2.80	3
15	28-JUL-2020	19:03:21	46	3.00	3
16	28-JUL-2020	19:03:22	44	3.20	4
17	28-JUL-2020	19:03:23	36	3.40	4
18	28-JUL-2020	19:03:24	41	3.60	4
19	28-JUL-2020	19:03:25	33	3.80	4
20	28-JUL-2020	19:03:26	19	4.00	4

A última etapa é aplicar as funções agregadas apropriadas para encontrar o número total de transações por cinco segundos, juntamente com os horários de início e término de cada transação:

```
select ceil(trx_id/5.0) as grp,
       min(trx_date) as trx_start,
       max(trx_date) as trx_end,
       sum(trx_cnt) as total
  from trx_log
 group by ceil(trx_id/5.0)
```

GRP	TRX_START	TRX_END	TOTAL
1	28-JUL-2020 19:03:07	28-JUL-2020 19:03:11	141
2	28-JUL-2020 19:03:12	28-JUL-2020 19:03:16	178
3	28-JUL-2020 19:03:17	28-JUL-2020 19:03:21	202
4	28-JUL-2020 19:03:22	28-JUL-2020 19:03:26	173

Se seus dados forem um pouco diferentes (talvez você não tenha um ID para cada linha), você sempre pode “agrupar” dividindo os segundos de cada linha TRX\_DATE por cinco para criar um agrupamento semelhante. Em seguida, você pode incluir a hora para cada TRX\_DATE e agrupar pela hora real e “agrupamento” lógico, GRP. A seguir está um exemplo desta técnica (usando as funções TO\_CHAR e TO\_NUMBER da Oracle, você usaria as funções de formatação de data e caracteres apropriadas para sua plataforma):

```
select trx_date, trx_cnt,
       to_number(to_char(trx_date, 'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60, 'miss'))/5.0) grp
  from trx_log
```

TRX_DATE	20	TRX_CNT	HR	GRP
28-JUL-2020 19:03:07		44	19	62
28-JUL-2020 19:03:08		18	19	62
28-JUL-2020 19:03:09		23	19	62
28-JUL-2020 19:03:10		29	19	62
28-JUL-2020 19:03:11		27	19	62
28-JUL-2020 19:03:12		45	19	63
28-JUL-2020 19:03:13		45	19	63
28-JUL-2020 19:03:14		32	19	63
28-JUL-2020 19:03:15		41	19	63
28-JUL-2020 19:03:16		15	19	63
28-JUL-2020 19:03:17		24	19	64
28-JUL-2020 19:03:18		47	19	64
28-JUL-2020 19:03:19		37	19	64

28-JUL-2020 19:03:20	48	19	64
28-JUL-2020 19:03:21	46	19	64
28-JUL-2020 19:03:22	44	19	65
28-JUL-2020 19:03:23	36	19	65
28-JUL-2020 19:03:24	41	19	65
28-JUL-2020 19:03:25	33	19	65
28-JUL-2020 19:03:26	19	19	65

Independentemente dos valores reais de GRP, a chave aqui é que você está agrupando a cada cinco segundos. A partir daí você pode aplicar as funções agregadas da mesma forma que na solução original:

```
select hr,grp,sum(trx_cnt) total
  from (
select trx_date,trx_cnt,
       to_number(to_char(trx_date,'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60,'miss'))/5.0) grp
  from trx_log
      ) x
 group by hr,grp
HR          GRP        TOTAL
----- -----
19           62         141
19           63         178
19           64         202
19           65         173
```

Incluir a hora no agrupamento será útil se o seu log de transações abrange horas. No DB2 e no Oracle, você também pode usar a função de janela SUM OVER para produzir o mesmo resultado. A consulta a seguir retorna todas as linhas de TRX\_LOG junto com um total acumulado para TRX\_CNT por “grupo” lógico e o TOTAL para TRX\_CNT para cada linha no “grupo”:

```
select trx_id, trx_date, trx_cnt,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)
                         order by trx_date
                         range between unbounded preceding
                           and current row) runing_total,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)) total,
       case when mod(trx_id,5.0) = 0 then 'X' end grp_end
  from trx_log
```

TRX_ID	TRX_DATE	TRX_CNT	RUNING_TOTAL	TOTAL	GRP_END
1	28-JUL-2020 19:03:07	44	44	141	
2	28-JUL-2020 19:03:08	18	62	141	
3	28-JUL-2020 19:03:09	23	85	141	
4	28-JUL-2020 19:03:10	29	114	141	
5	28-JUL-2020 19:03:11	27	141	141	X
6	28-JUL-2020 19:03:12	45	45	178	
7	28-JUL-2020 19:03:13	45	90	178	
8	28-JUL-2020 19:03:14	32	122	178	

Capítulo 12	421			
9 28-JUL-2020 19:03:15	41	163	178	
10 28-JUL-2020 19:03:16	15	178	178	X
11 28-JUL-2020 19:03:17	24	24	202	
12 28-JUL-2020 19:03:18	47	71	202	
13 28-JUL-2020 19:03:19	37	108	202	
14 28-JUL-2020 19:03:20	48	156	202	
15 28-JUL-2020 19:03:21	46	202	202	X
16 28-JUL-2020 19:03:22	44	44	173	
17 28-JUL-2020 19:03:23	36	80	173	
18 28-JUL-2020 19:03:24	41	121	173	
19 28-JUL-2020 19:03:25	33	154	173	
20 28-JUL-2020 19:03:26	19	173	173	X

## 12.18 Executando agregações em diferentes grupos/partições simultaneamente

### Problema

Você deseja agregar diferentes dimensões ao mesmo tempo. Por exemplo, você deseja retornar um conjunto de resultados que liste o nome de cada funcionário, seu departamento, o número de funcionários em seu departamento (eles incluídos), o número de funcionários que têm o mesmo cargo (eles próprios incluídos nesta contagem também), e o número total de funcionários na tabela EMP. O conjunto de resultados deve ser semelhante ao seguinte:

ENAME	DEPTNO	DEPTNO_CNT	JOB	JOB_CNT	TOTAL
MILLER	10	3	CLERK	4	14
CLARK	10	3	MANAGER	3	14
KING	10	3	PRESIDENT	1	14
SCOTT	20	5	ANALYST	2	14
FORD	20	5	ANALYST	2	14
SMITH	20	5	CLERK	4	14
JONES	20	5	MANAGER	3	14
ADAMS	20	5	CLERK	4	14
JAMES	30	6	CLERK	4	14
MARTIN	30	6	SALESMAN	4	14
TURNER	30	6	SALESMAN	4	14
WARD	30	6	SALESMAN	4	14
ALLEN	30	6	SALESMAN	4	14
BLAKE	30	6	MANAGER	3	14

### Solução

Use a função de janela COUNT OVER ao especificar diferentes partições ou grupos de dados nos quais realizar a agregação:

```
select ename,
       deptno,
       count(*)over(partition by deptno) deptno_cnt,
```

```

job,
count(*)over(partition by job) job_cnt,
count(*)over() total
from emp

```

## Discussão

Este exemplo realmente mostra o poder e a conveniência das funções da janela. Simplesmente especificando diferentes partições ou grupos de dados para agregar, você pode criar relatórios imensamente detalhados sem ter que se juntar repetidamente e sem ter que escrever subconsultas complicadas e talvez com baixo desempenho em sua lista SELECT. Todo o trabalho é feito pela função de janela COUNT OVER. Para entender o resultado, concentre-se na cláusula OVER por um momento para cada operação COUNT:

```

count(*)over(partition by deptno)

count(*)over(partition by job)

count(*)over()

```

Lembre-se das partes principais da cláusula OVER: a subcláusula PARTITION BY, dividindo a consulta em partições; e a subcláusula ORDER BY, definindo a ordem lógica. Veja o primeiro COUNT, que particiona por DEPTNO. As linhas da tabela EMP serão agrupadas por DEPTNO e a operação COUNT será realizada em todas as linhas de cada grupo. Como não há cláusula de quadro ou janela especificada (sem ORDER BY), todas as linhas do grupo são contadas. A cláusula PARTITION BY encontra todos os valores exclusivos DEPTNO e então a função COUNT conta o número de linhas com cada valor. No exemplo específico de COUNT(\*)OVER(PARTITION BY DEPTNO), a cláusula PARTITION BY identifica as partições ou grupos com valores 10, 20 e 30.

O mesmo processamento é aplicado ao segundo COUNT, que particiona por JOB. A última contagem não é particionada por nada e simplesmente possui parênteses vazios. Um parênteses vazio implica “a tabela inteira”. Portanto, enquanto os dois COUNTs anteriores agregam valores com base nos grupos ou partições definidos, o COUNT final conta todas as linhas na tabela EMP.



Lembre-se de que as funções de janela são aplicadas após a cláusula WHERE. Se você filtrasse o conjunto de resultados de alguma forma, por exemplo, excluindo todos os funcionários em DEPTNO 10, o valor de TOTAL não seria 14 — seria 11. Para filtrar os resultados após a avaliação das funções da janela, você deve fazer seu janelar a consulta em uma visualização embutida e, em seguida, filtrar os resultados dessa visualização.

## 12.19 Executando agregações em uma faixa móvel de valores

### Problema

Você deseja calcular uma agregação móvel, como uma soma móvel sobre os salários na tabela EMP. Você deseja calcular uma soma a cada 90 dias, começando pela HIREDATE do primeiro funcionário. Você deseja ver como os gastos flutuaram a cada período de 90 dias entre o primeiro e o último funcionário contratado. Você deseja retornar o seguinte conjunto de resultados:

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-200	800	800
20-FEB-2011	1600	2400
22-FEB-2011	1250	3650
02-APR-2011	2975	5825
01-MAY-2011	2850	8675
09-JUN-2011	2450	8275
08-SEP-2011	1500	1500
28-SEP-2011	1250	2750
17-NOV-2011	5000	7750
03-DEC-2011	950	11700
03-DEC-2011	3000	11700
23-JAN-2012	1300	10250
09-DEC-2012	3000	3000
12-JAN-2013	1100	4100

### Solução

Ser capaz de especificar uma janela móvel na cláusula de enquadramento ou janelamento das funções de janela torna esse problema fácil de resolver, se o seu RDBMS suportar tais funções. O segredo é fazer o pedido por HIREDATE em sua função de janela e, em seguida, especificar uma janela de 90 dias a partir do primeiro funcionário contratado. O valor será calculado utilizando os salários dos funcionários admitidos até 90 dias antes da CONTRATAÇÃO do funcionário atual (o funcionário atual está incluído no valor). Se você não tiver funções de janela disponíveis, poderá usar subconsultas escalares, mas a solução será mais complexa.

#### DB2 e Oracle

Para DB2 e Oracle, use a função de janela SUM OVER e ordene por HIREDATE. Especifique um intervalo de 90 dias na janela ou cláusula de “enquadramento” para permitir que o valor seja calculado para o salário de cada funcionário e para incluir os salários de todos os funcionários contratados até 90 dias antes.

Como o DB2 não permite especificar HIREDATE na cláusula ORDER BY de uma função de janela (linha 3 no código a seguir), você pode ordenar por DAYS (HIREDATE) em vez disso:

```

1 select hiredate,
2      sal,
3      sum(sal)over(order by days(hiredate)
4                      range between 90 preceding
5                      and current row) spending_pattern
6  from emp e

```

A solução Oracle é mais direta que a do DB2, porque o Oracle permite que funções de janela sejam ordenadas por tipos de data e hora:

```

1 select hiredate,
2      sal,
3      sum(sal)over(order by hiredate
4                      range between 90 preceding
5                      and current row) spending_pattern
6  from emp e

```

## MySQL

Use a função window com sintaxe ligeiramente alterada:

```

1 select hiredate,
2      sal,
3      sum(sal)over(order by hiredate
4                      range interval 90 day preceding ) spending_pattern
5  from emp e

```

## PostgreSQL e SQL Server

Use uma subconsulta escalar para somar os salários de todos os funcionários contratados até 90 dias antes do dia em que cada funcionário foi contratado:

```

1 select e.hiredate,
2      e.sal,
3      (select sum(sal) from emp d
4       where d.hiredate between e.hiredate-90
5                           and e.hiredate) as spending_pattern
6  from emp e
7 order by 1

```

## Discussão

### DB2, MySQL e Oracle

DB2, MySQL e Oracle compartilham a mesma solução lógica. As únicas pequenas diferenças entre as soluções estão em como você especifica HIREDATE na cláusula ORDER BY da função de janela e na sintaxe de especificação do intervalo de tempo no MySQL.

No momento em que este livro foi escrito, o DB2 não permitia um valor DATE em tal cláusula ORDER BY se você estivesse usando um valor numérico para definir o intervalo da janela. (Por exemplo, RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW permite você ordenar por uma data, mas RANGE BETWEEN 90 PRECEDING AND CURRENT ROW não.)

Para entender o que a consulta da solução está fazendo, basta entender o que a cláusula window está fazendo. A janela que você está definindo ordena os salários de todos os funcionários por HIREDATE. Então a função calcula uma soma. A soma não é computada para todos os salários. Em vez disso, o processamento é o seguinte:

1. É avaliado o salário do primeiro funcionário contratado. Como nenhum funcionário foi contratado antes do primeiro funcionário, a soma neste momento é simplesmente o salário do primeiro funcionário.
2. O salário do próximo funcionário (por HIREDATE) é avaliado. O salário desse funcionário está incluído no valor da mudança junto com os demais funcionários contratados até 90 dias antes.

A HIREDATE do primeiro funcionário é 17 de dezembro de 2010, e a HIREDATE do próximo funcionário contratado é 20 de fevereiro de 2011. O segundo funcionário foi contratado menos de 90 dias após o primeiro funcionário e, portanto, o valor móvel do segundo funcionário é 2400 (1600 + 800). Se você estiver tendo problemas para entender de onde vêm os valores em SPENDING\_PATTERN, examine a seguinte consulta e o conjunto de resultados:

```
select distinct
    dense_rank()over(order by e.hiredate) window,
    e.hiredate current_hiredate,
    d.hiredate hiredate_within_90_days,
    d.sal sals_used_for_sum
from emp e,
     emp d
where d.hiredate between e.hiredate-90 and e.hiredate
      
```

	WINDOW CURRENT_HIREDATE	HIREDATE_WITHIN_90_DAYS	SALS_USED_FOR_SUM
1	17-DEC-2010	17-DEC-2010	800
2	20-FEB-2011	17-DEC-2010	800
2	20-FEB-2011	20-FEB-2011	1600
3	22-FEB-2011	17-DEC-2010	800
3	22-FEB-2011	20-FEB-2011	1600
3	22-FEB-2011	22-FEB-2011	1250
4	02-APR-2011	20-FEB-2011	1600
4	02-APR-2011	22-FEB-2011	1250
4	02-APR-2011	02-APR-2011	2975
5	01-MAY-2011	20-FEB-2011	1600
5	01-MAY-2011	22-FEB-2011	1250
5	01-MAY-2011	02-APR-2011	2975
5	01-MAY-2011	01-MAY-2011	2850

6 09-JUN-2011	02-APR-2011	2975
6 09-JUN-2011	01-MAY-2011	2850
6 09-JUN-2011	09-JUN-2011	2450
7 08-SEP-2011	08-SEP-2011	1500
8 28-SEP-2011	08-SEP-2011	1500
8 28-SEP-2011	28-SEP-2011	1250
9 17-NOV-2011	08-SEP-2011	1500
9 17-NOV-2011	28-SEP-2011	1250
9 17-NOV-2011	17-NOV-2011	5000
10 03-DEC-2011	08-SEP-2011	1500
10 03-DEC-2011	28-SEP-2011	1250
10 03-DEC-2011	17-NOV-2011	5000
10 03-DEC-2011	03-DEC-2011	950
10 03-DEC-2011	03-DEC-2011	3000
11 23-JAN-2012	17-NOV-2011	5000
11 23-JAN-2012	03-DEC-2011	950
11 23-JAN-2012	03-DEC-2011	3000
11 23-JAN-2012	23-JAN-2012	1300
12 09-DEC-2012	09-DEC-2012	3000
13 12-JAN-2013	09-DEC-2012	3000
13 12-JAN-2013	12-JAN-2013	1100

Se você observar a coluna WINDOW, apenas as linhas com o mesmo valor WINDOW serão consideradas para cada soma. Tomemos, por exemplo, a WINDOW 3. Os salários utilizados para a soma dessa janela são 800, 1600 e 1250, que totalizam 3650. Se você olhar o resultado final definido na seção “Problema”, verá o SPENDING\_PATTERN para 22 de fevereiro de 2011 (WINDOW 3) é 3650. Como prova, para verificar se o auto agrupamento anterior inclui os salários corretos para as janelas definidas, basta somar os valores em SALS\_USED\_FOR\_SUM e agrupar por CURRENT\_DATE. O resultado deve ser igual ao conjunto de resultados mostrado na seção “Problema” (com a linha duplicada de 3 de dezembro de 2011, filtrada):

```

select current_hiredate,
       sum(sals_used_for_sum) spending_pattern
  from (
select distinct
        dense_rank()over(order by e.hiredate) window,
        e.hiredate current_hiredate,
        d.hiredate hiredate_within_90_days,
        d.sal sals_used_for_sum
   from emp e,
        emp d
  where d.hiredate between e.hiredate-90 and e.hiredate
        ) x
 group by current_hiredate

```

CURRENT_HIREDATE	SPENDING_PATTERN
17-DEC-2010	800
20-FEB-2011	2400
22-FEB-2011	3650

02-APR-2011	5825
01-MAY-2011	8675
09-JUN-2011	8275
08-SEP-2011	1500
28-SEP-2011	2750
17-NOV-2011	7750
03-DEC-2011	11700
23-JAN-2012	10250
09-DEC-2012	3000
12-JAN-2013	4100

## PostgreSQL e SQL Server

A chave para esta solução é usar uma subconsulta escalar (uma auto-junção também funcionará) enquanto usa a função agregada SUM para calcular uma soma para cada 90 dias com base em HIREDATE. Se você estiver tendo problemas para ver como isso funciona, simplesmente converta a solução em uma autojunção e examine quais linhas estão incluídas nos cálculos. Considere o seguinte conjunto de resultados, que retorna o mesmo conjunto de resultados da solução:

```
select e.hiredate,
       e.sal,
       sum(d.sal) as spending_pattern
  from emp e, emp d
 where d.hiredate
       between e.hiredate-90 and e.hiredate
 group by e.hiredate,e.sal
 order by 1\
```

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-2010	800	800
20-FEB-2011	1600	2400
22-FEB-2011	1250	3650
02-APR-2011	2975	5825
01-MAY-2011	2850	8675
09-JUN-2011	2450	8275
08-SEP-2011	1500	1500
28-SEP-2011	1250	2750
17-NOV-2011	5000	7750
03-DEC-2011	950	11700
03-DEC-2011	3000	11700
23-JAN-2012	1300	10250
09-DEC-2012	3000	3000
12-JAN-2013	1100	4100

Se ainda não estiver claro, basta remover a agregação e começar com o produto cartesiano. O primeiro passo é gerar um produto cartesiano utilizando a tabela EMP para que cada HIREDATE possa ser comparado com todos os outros HIREDATES. (Apenas um trecho do conjunto de resultados é mostrado aqui porque existem 196 linhas ( $14 \times 14$ ) retornadas por um cartesiano de EMP):

```
select
    e.hiredate,
    e.sal,
    d.sal,
    d.hiredate
from emp e, emp d
```

HIREDATE	SAL	SAL HIREDATE
17-DEC-2010	800	800 17-DEC-2010
17-DEC-2010	800	1600 20-FEB-2011
17-DEC-2010	800	1250 22-FEB-2011
17-DEC-2010	800	2975 02-APR-2011
17-DEC-2010	800	1250 28-SEP-2011
17-DEC-2010	800	2850 01-MAY-2011
17-DEC-2010	800	2450 09-JUN-2011
17-DEC-2010	800	3000 09-DEC-2012
17-DEC-2010	800	5000 17-NOV-2011
17-DEC-2010	800	1500 08-SEP-2011
17-DEC-2010	800	1100 12-JAN-2013
17-DEC-2010	800	950 03-DEC-2011
17-DEC-2010	800	3000 03-DEC-2011
17-DEC-2010	800	1300 23-JAN-2012
20-FEB-2011	1600	800 17-DEC-2010
20-FEB-2011	1600	1600 20-FEB-2011
20-FEB-2011	1600	1250 22-FEB-2011
20-FEB-2011	1600	2975 02-APR-2011
20-FEB-2011	1600	1250 28-SEP-2011
20-FEB-2011	1600	2850 01-MAY-2011
20-FEB-2011	1600	2450 09-JUN-2011
20-FEB-2011	1600	3000 09-DEC-2012
20-FEB-2011	1600	5000 17-NOV-2011
20-FEB-2011	1600	1500 08-SEP-2011
20-FEB-2011	1600	1100 12-JAN-2013
20-FEB-2011	1600	950 03-DEC-2011
20-FEB-2011	1600	3000 03-DEC-2011
20-FEB-2011	1600	1300 23-JAN-2012

Se você examinar o conjunto de resultados anterior, notará que não há HIREDATE 90 dias antes ou igual a 17 de dezembro, exceto 17 de dezembro. Portanto, a soma dessa linha deve ser apenas 800. Se você examinar o próximo HIREDATE, 20 de fevereiro, você notará que há um HIREDATE que está dentro da janela de 90 dias (dentro dos 90 dias anteriores), e é 17 de dezembro. Se você somar o SAL de 17 de dezembro com o SAL de 20 de fevereiro (porque nós estamos procurando HIREDATEs iguais a cada HIREDATE ou dentro de 90 dias antes), você obtém 2.400, que é o resultado final para esse HIREDATE.

Agora que você sabe como funciona, use um filtro na cláusula WHERE para retornar para cada HIREDATE e HIREDATE que seja igual a ele ou não seja maior que 90 dias antes:

```

d.sal sal_to_sum,
d.hiredate within_90_days
from emp e, emp d
where d.hiredate
    between e.hiredate-90 and e.hiredate
order by 1
HIREDATE      SAL SAL_TO_SUM WITHIN_90_DAYS
-----
17-DEC-2010    800    800    17-DEC-2010
20-FEB-2011   1600    800    17-DEC-2010
20-FEB-2011   1600   1600    20-FEB-2011
22-FEB-2011   1250    800    17-DEC-2010
22-FEB-2011   1250   1600    20-FEB-2011
22-FEB-2011   1250   1250    22-FEB-2011
02-APR-2011   2975   1600    20-FEB-2011
02-APR-2011   2975   1250    22-FEB-2011
02-APR-2011   2975   2975    02-APR-2011
01-MAY-2011   2850   1600    20-FEB-2011
01-MAY-2011   2850   1250    22-FEB-2011
01-MAY-2011   2850   2975    02-APR-2011
01-MAY-2011   2850   2850    01-MAY-2011
09-JUN-2011   2450   2975    02-APR-2011
09-JUN-2011   2450   2850    01-MAY-2011
09-JUN-2011   2450   2450    09-JUN-2011
08-SEP-2011   1500   1500    08-SEP-2011
28-SEP-2011   1250   1500    08-SEP-2011
28-SEP-2011   1250   1250    28-SEP-2011
17-NOV-2011   5000   1500    08-SEP-2011
17-NOV-2011   5000   1250    28-SEP-2011
17-NOV-2011   5000   5000    17-NOV-2011
03-DEC-2011   950    1500    08-SEP-2011
03-DEC-2011   950    1250    28-SEP-2011
03-DEC-2011   950    5000    17-NOV-2011
03-DEC-2011   950    950     03-DEC-2011
03-DEC-2011   950    3000    03-DEC-2011
03-DEC-2011   3000   1500    08-SEP-2011
03-DEC-2011   3000   1250    28-SEP-2011
03-DEC-2011   3000   5000    17-NOV-2011
03-DEC-2011   3000   950     03-DEC-2011
03-DEC-2011   3000   3000    03-DEC-2011
23-JAN-2012   1300   5000    17-NOV-2011
23-JAN-2012   1300    950    03-DEC-2011
23-JAN-2012   1300   3000    03-DEC-2011
23-JAN-2012   1300   1300    23-JAN-2012
09-DEC-2012   3000   3000    09-DEC-2012
12-JAN-2013   1100   3000    09-DEC-2012
12-JAN-2013   1100   1100    12-JAN-2013

```

Agora que você sabe quais SALs serão incluídas na janela móvel de soma, basta usar a função agregada SUM para produzir um conjunto de resultados mais expressivo:

```

sum(d.sal) as spending_pattern
from emp e, emp d
where d.hiredate
    between e.hiredate-90 and e.hiredate
group by e.hiredate,e.sal
order by 1

```

Se você comparar o conjunto de resultados da consulta anterior e o conjunto de resultados da consulta mostrada aqui (que é a solução original apresentada), verá que eles são iguais:

```

select e.hiredate,
       e.sal,
       (select sum(sal) from emp d
        where d.hiredate between e.hiredate-90
                           and e.hiredate) as spending_pattern
  from emp e
 order by 1

```

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-2010	800	800
20-FEB-2011	1600	2400
22-FEB-2011	1250	3650
02-APR-2011	2975	5825
01-MAY-2011	2850	8675
09-JUN-2011	2450	8275
08-SEP-2011	1500	1500
28-SEP-2011	1250	2750
17-NOV-2011	5000	7750
03-DEC-2011	950	11700
03-DEC-2011	3000	11700
23-JAN-2012	1300	10250
09-DEC-2012	3000	3000
12-JAN-2013	1100	4100

## 12.20 Dinamizando um conjunto de resultados com subtotais

### Problema

Você deseja criar um relatório contendo subtotais e depois transpor os resultados para fornecer um relatório mais legível. Por exemplo, você foi solicitado a criar um relatório que exiba, para cada departamento, os gerentes do departamento e uma soma dos salários dos funcionários que trabalham para esses gerentes. Além disso, você deseja retornar dois subtotais: a soma de todos os salários em cada departamento para os funcionários que têm gerentes e uma soma de todos os salários no conjunto de resultados (a soma dos subtotais do departamento). Atualmente você tem o seguinte relatório:

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
10		3750
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
20		10875
30	7698	6550
30	7839	2850
30		9400
		24025

Você deseja fornecer um relatório mais legível e transformar o conjunto de resultados anterior no seguinte, o que torna o significado do relatório muito mais claro:

MGR	DEPT10	DEPT20	DEPT30	TOTAL
7566	0	6000	0	
7698	0	0	6550	
7782	1300	0	0	
7788	0	1100	0	
7839	2450	2975	2850	
7902	0	800	0	
	3750	10875	9400	24025

## Solução

O primeiro passo é gerar subtotais utilizando a extensão ROLLUP para GROUP BY. O próximo passo é realizar um pivô clássico (agregado e expressão CASE) para criar as colunas desejadas para o seu relatório. A função GROUPING permite determinar facilmente quais valores são subtotais (ou seja, existem por causa de ROLLUP e normalmente não estariam lá). Dependendo de como o seu RDBMS classifica os valores NULL, pode ser necessário adicionar um ORDER BY à solução para permitir que ela se pareça com o conjunto de resultados de destino anterior.

### DB2 e Oracle

Use a extensão ROLLUP para GROUP BY e depois use uma expressão CASE para formatar os dados em um relatório mais legível:

```

1 select mgr,
2      sum(case deptno when 10 then sal else 0 end) dept10,
3      sum(case deptno when 20 then sal else 0 end) dept20,
4      sum(case deptno when 30 then sal else 0 end) dept30,
5      sum(case flag when '11' then sal else null end) total
6  from (
7 select deptno,mgr,sum(sal) sal,
1      cast(grouping(deptno) as char(1))||
```

```

8      cast(grouping(mgr) as char(1)) flag
9  from emp
10 where mgr is not null
11 group by rollup(deptno,mgr)
12      ) x
13 group by mgr

```

## SQL Server

Use a extensão ROLLUP para GROUP BY e depois use uma expressão CASE para formatar os dados em um relatório mais legível:

```

1 select mgr,
2      sum(case deptno when 10 then sal else 0 end) dept10,
3      sum(case deptno when 20 then sal else 0 end) dept20,
4      sum(case deptno when 30 then sal else 0 end) dept30,
5      sum(case flag when '11' then sal else null end) total
6  from (
7 select deptno,mgr,sum(sal) sal,
8      cast(grouping(deptno) as char(1))+
9      cast(grouping(mgr) as char(1)) flag
10 from emp
11 where mgr is not null
12 group by deptno,mgr with rollup
13      ) x
14 group by mgr

```

## PostgreSQL

Use a extensão ROLLUP para GROUP BY e depois use uma expressão CASE para formatar os dados em um relatório mais legível:

```

1 select mgr,
2      sum(case deptno when 10 then sal else 0 end) dept10,
3      sum(case deptno when 20 then sal else 0 end) dept20,
4      sum(case deptno when 30 then sal else 0 end) dept30,
5      sum(case flag when '11' then sal else null end) total
6  from (
7 select deptno,mgr,sum(sal) sal,
8      concat(cast (grouping(deptno) as char(1)),
9      cast(grouping(mgr) as char(1))) flag
10 from emp
11 where mgr is not null
12 group by rollup (deptno,mgr)
13      ) x
14 group by mgr

```

## MySQL

Use a extensão ROLLUP para GROUP BY e depois use uma expressão CASE para formatar os dados em um relatório mais legível:

```

1 select mgr,
2      sum(case deptno when 10 then sal else 0 end) dept10,
3      sum(case deptno when 20 then sal else 0 end) dept20,
4      sum(case deptno when 30 then sal else 0 end) dept30,
5      sum(case flag when '11' then sal else null end) total
6  from (
7  select deptno,mgr,sum(sal) sal,
8        concat( cast(grouping(deptno) as char(1)) ,
9        cast(grouping(mgr) as char(1))) flag
10 from emp
11 where mgr is not null
12 group by deptno,mgr with rollup
13      ) x
14 group by mgr;

```

## Discussão

As soluções fornecidas aqui são idênticas, exceto pela concatenação de strings e como GROUPING é especificado. Como as soluções são muito semelhantes, a discussão a seguir se referirá à solução SQL Server para destacar os conjuntos de resultados intermediários (a discussão também é relevante para DB2 e Oracle).

O primeiro passo é gerar um conjunto de resultados que some o SAL dos funcionários em cada DEPTNO por MGR. A ideia é mostrar quanto ganham os funcionários de um determinado gestor em um determinado departamento. Por exemplo, a consulta a seguir permitirá comparar os salários dos funcionários que trabalham para KING no DEPTNO 10 em comparação com aqueles que trabalham para KING no DEPTNO 30:

```

select deptno,mgr,sum(sal) sal
  from emp
 where mgr is not null
 group by mgr,deptno
 order by 1,2

```

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
30	7698	6550
30	7839	2850

O próximo passo é usar a extensão ROLLUP para GROUP BY para criar subtotais para cada DEPTNO e para todos os funcionários (que possuem um gerente):

```

select deptno,mgr,sum(sal) sal
  from emp
 where mgr is not null

```

```
group by deptno,mgr with rollup
```

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
10		3750
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
20		10875
30	7698	6550
30	7839	2850
30		9400
		24025

Com os subtotais criados, você precisa determinar quais valores são de fato subtotais (criados por ROLLUP) e quais são resultados do GROUP BY regular. Use a função GROUPING para criar bitmaps para ajudar a identificar os valores subtotais dos valores agregados regulares:

```
select deptno,mgr,sum(sal) sal,
       cast(grouping(deptno) as char(1))+
       cast(grouping(mgr) as char(1)) flag
  from emp
 where mgr is not null
 group by deptno,mgr with rollup
```

DEPTNO	MGR	SAL	FLAG
10	7782	1300 00	
10	7839	2450 00	
10		3750 01	
20	7566	6000 00	
20	7788	1100 00	
20	7839	2975 00	
20	7902	800 00	
20		10875 01	
30	7698	6550 00	
30	7839	2850 00	
30		9400 01	
		24025 11	

Se não for imediatamente óbvio, as linhas com valor 00 para FLAG são os resultados de agregação regular. As linhas com valor 01 para FLAG são os resultados do ROLLUP agregando SAL por DEPTNO (uma vez que DEPTNO é listado primeiro no ROLLUP; se você mudar a ordem, por exemplo, GROUP BY MGR, DEPTNO COM ROLLUP, você verá bastante resultados diferentes). A linha com valor 11 para FLAG é o resultado da agregação de SAL por ROLLUP em todas as linhas.

Neste ponto você tem tudo que precisa para criar um relatório embelezado simplesmente usando expressões CASE. O objetivo é fornecer um relatório que mostre os salários dos funcionários de cada gerente em todos os departamentos. Caso um gestor não possua subordinados em determinado departamento, deverá ser retornado zero; caso contrário, você deseja retornar a soma de todos os salários dos subordinados desse gerente nesse departamento. Além disso, você deseja adicionar uma coluna final, TOTAL, representando a soma de todos os salários do relatório. A solução que satisfaz todos esses requisitos é mostrada aqui:

```
select mgr,
       sum(case deptno when 10 then sal else 0 end) dept10,
       sum(case deptno when 20 then sal else 0 end) dept20,
       sum(case deptno when 30 then sal else 0 end) dept30,
       sum(case flag when '11' then sal else null end) total
  from (
    select deptno,mgr,sum(sal) sal,
           cast(grouping(deptno) as char(1))+
           cast(grouping(mgr) as char(1)) flag
      from emp
     where mgr is not null
   group by deptno,mgr with rollup
      ) x
  group by mgr
 order by coalesce(mgr,9999)
```

MGR	DEPT10	DEPT20	DEPT30	TOTAL
7566	0	6000	0	
7698	0	0	6550	
7782	1300	0	0	
7788	0	1100	0	
7839	2450	2975	2850	
7902	0	800	0	
	3750	10875	9400	24025

## 12.21 Resumindo

Os bancos de dados servem para armazenar dados, mas eventualmente alguém precisa recuperá-los e apresentá-los em algum lugar. As receitas deste capítulo mostram uma variedade de maneiras importantes pelas quais os dados podem ser remodelados ou formatados para atender às necessidades dos usuários. Além de sua utilidade geral em fornecer dados aos usuários na forma que eles necessitam, essas técnicas desempenham um papel importante ao dar ao proprietário do banco de dados a capacidade de criar um datawarehouse.

À medida que você ganha mais experiência no suporte aos usuários no negócio, você se tornará mais hábil e ampliará as ideias aqui em apresentações mais elaboradas.

# CAPÍTULO 13

## Consultas hierárquicas

Este capítulo apresenta receitas para expressar relacionamentos hierárquicos que você pode ter em seus dados. É comum, ao trabalhar com dados hierárquicos, ter mais dificuldade em recuperar e exibir os dados (como uma hierarquia) do que armazená-los.

Embora tenham se passado apenas alguns anos desde que o MySQL adicionou CTEs recursivos, agora que eles estão disponíveis significa que CTEs recursivos estão disponíveis em praticamente todos os RDBMS. Como resultado, eles são o padrão ouro para lidar com consultas hierárquicas, e este capítulo fará uso liberal dessa capacidade para fornecer receitas que o ajudarão a desvendar a estrutura hierárquica dos seus dados.

Antes de começar, examine a tabela EMP e o relacionamento hierárquico entre EMPNO e MGR:

```
select empno,mgr
  from emp
 order by 2
```

EMPNO	MGR
7788	7566
7902	7566
7499	7698
7521	7698
7900	7698
7844	7698
7654	7698
7934	7782
7876	7788
7566	7839
7782	7839
7698	7839
7369	7902
7839	

Se você olhar com atenção, verá que cada valor de MGR também é um EMPNO, ou seja, o gerente de cada funcionário na tabela EMP também é um funcionário da tabela EMP e não está armazenado em outro lugar. A relação entre MGR e EMPNO é uma relação pai-filho, pois o valor para MGR é o pai mais imediato para um determinado EMPNO (também é possível que o gerente de um funcionário específico também possa ter um gerente, e esses gerentes possam por sua vez, têm gerentes, e assim por diante, criando uma hierarquia  $n$ - de níveis). Se um funcionário não tiver gerente, então MGR será NULL.

## 13.1 Expressando um relacionamento pai-filho

### Problema

Você deseja incluir informações dos pais junto com os dados dos registros dos filhos. Por exemplo, você deseja exibir o nome de cada funcionário junto com o nome de seu gerente. Você deseja retornar o seguinte conjunto de resultados:

```
EMPS_AND_MGRS
-----
FORD works for JONES
SCOTT works for JONES
JAMES works for BLAKE
TURNER works for BLAKE
MARTIN works for BLAKE
WARD works for BLAKE
ALLEN works for BLAKE
MILLER works for CLARK
ADAMS works for SCOTT
CLARK works for KING
BLAKE works for KING
JONES works for KING
SMITH works for FORD
```

### Solução

Cadastre-se no EMP no MGR e no EMPNO para encontrar o nome do gerente de cada funcionário. Em seguida, use as funções fornecidas pelo seu RDBMS para concatenação de strings para gerar as strings no conjunto de resultados desejado.

#### DB2, Oracle e PostgreSQL

Inscreve-se automaticamente no EMP. Em seguida, use o operador de concatenação de barra vertical dupla (||):

```
1 select a.ename || ' works for ' || b.ename as emps_and_mgrs
2   from emp a, emp b
1 where a.mgr = b.empno
```

## MySQL

Inscreve-se automaticamente no EMP. Em seguida, use a função de concatenação CONCAT:

```
1 select concat(a.ename, ' works for ',b.ename) as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

## SQL Server

Inscreve-se automaticamente no EMP. Em seguida, use o sinal de mais (+) como operador de concatenação:

```
1 select a.ename + ' works for ' + b.ename as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

## Discussão

A implementação é essencialmente a mesma para todas as soluções. A diferença está apenas no método de concatenação de strings e, portanto, uma discussão cobrirá todas as soluções.

A chave é a união entre MGR e EMPNO. O primeiro passo é construir um produto cartesiano unindo o EMP a ele mesmo (apenas uma parte das linhas retornadas pelo produto cartesiano é mostrada aqui):

```
select a.empno, b.empno
  from emp a, emp b
```

EMPNO	MGR
7369	7369
7369	7499
7369	7521
7369	7566
7369	7654
7369	7698
7369	7782
7369	7788
7369	7839
7369	7844
7369	7876
7369	7900
7369	7902
7369	7934
7499	7369
7499	7499
7499	7521
7499	7566
7499	7654
7499	7698
7499	7782

7499	7788
7499	7839
7499	7844
7499	7876
7499	7900
7499	7902
7499	7934

Como você pode ver, ao usar um produto cartesiano você está retornando todas as combinações possíveis EMPNO/EMPNO (de forma que pareça que o gerente do EMPNO 7369 são todos os outros funcionários da tabela, incluindo o EMPNO 7369).

O próximo passo é filtrar os resultados de forma que você retorne apenas o EMPNO de cada funcionário e seu gestor. Faça isso ingressando no MGR e no EMPNO:

```
1 select a.empno, b.empno mgr
2   from emp a, emp b
3  where a.mgr = b.empno
```

EMPNO	MGR
7902	7566
7788	7566
7900	7698
7844	7698
7654	7698
7521	7698
7499	7698
7934	7782
7876	7788
7782	7839
7698	7839
7566	7839
7369	7902

Agora que você tem cada funcionário e o EMPNO de seu gestor, você pode retornar o nome de cada gestor simplesmente selecionando B.ENAME em vez de B.EMPNO. Se depois de alguma prática você tiver dificuldade em entender como isso funciona, você pode usar uma subconsulta escalar em vez de uma auto-junção para obter a resposta:

```
select a.ename,
       (select b.ename
        from emp b
       where b.empno = a.mgr) as mgr
  from emp a
```

ENAME	MGR
SMITH	FORD
ALLEN	BLAKE
WARD	BLAKE
JONES	KING

MARTIN	BLAKE
BLAKE	KING
CLARK	KING
SCOTT	JONES
KING	
TURNER	BLAKE
ADAMS	SCOTT
JAMES	BLAKE
FORD	JONES
MILLER	CLARK

A versão da subconsulta escalar é equivalente à autojunção, exceto por uma linha: o funcionário KING está no conjunto de resultados, mas esse não é o caso da autojunção. "Por que não?" você pode perguntar. Lembre-se, NULL nunca é igual a nada, nem mesmo a si mesmo. Na solução de auto-associação, você utiliza uma associação equitativa entre EMPNO e MGR, filtrando assim quaisquer funcionários que tenham NULL para MGR. Para ver o funcionário KING ao usar o método de auto-associação, você deve fazer a associação externa conforme mostrado nas duas consultas a seguir. A primeira solução usa a junção externa ANSI, enquanto a segunda usa a sintaxe de junção externa Oracle. A saída é a mesma para ambos e é mostrada após a segunda consulta:

```
/* ANSI */
select a.ename, b.ename mgr
  from emp a left join emp b
    on (a.mgr = b.empno)

/* Oracle */
select a.ename, b.ename mgr
  from emp a, emp b
 where a.mgr = b.empno (+)
```

ENAME	MGR
FORD	JONES
SCOTT	JONES
JAMES	BLAKE
TURNER	BLAKE
MARTIN	BLAKE
WARD	BLAKE
ALLEN	BLAKE
MILLER	CLARK
ADAMS	SCOTT
CLARK	KING
BLAKE	KING
JONES	KING
SMITH	FORD
KING	

## 13.2 Expressando uma relação filho-pai-avó

### Problema

O funcionário CLARK trabalha para a KING, e para expressar esse relacionamento você pode usar a primeira receita deste capítulo. E se o funcionário CLARK fosse, por sua vez, gerente de outro funcionário? Considere a seguinte consulta:

```
select ename,empno,mgr
  from emp
 where ename in ('KING','CLARK','MILLER')
```

ENAME	EMPNO	MGR
CLARK	7782	7839
KING	7839	
MILLER	7934	7782

Como você pode ver, o funcionário MILLER trabalha para CLARK, que por sua vez trabalha para KING. Você deseja expressar a hierarquia completa de MILLER a KING. Você deseja retornar o seguinte conjunto de resultados:

```
LEAF    BRANCH    ROOT
-----
MILLER-->CLARK-->KING
```

No entanto, a abordagem de auto-junção única da receita anterior não será suficiente para mostrar todo o relacionamento de cima para baixo. Você poderia escrever uma consulta que faça duas auto-junções, mas o que você realmente precisa é de uma abordagem geral para atravessar essas hierarquias.

### Solução

Esta receita difere da primeira porque agora existe uma relação de três níveis, como o título sugere. Se o seu RDBMS não fornece funcionalidade para percorrer dados estruturados em árvore, como é o caso do Oracle, então você pode resolver esse problema usando os CTEs.

#### DB2, PostgreSQL e SQL Server

Use a cláusula recursiva COM para encontrar o gerente de MILLER, CLARK, e depois o gerente de CLARK, KING. O operador de concatenação de strings do SQL Server + é usado nesta solução:

```
1   with x (tree,mgr,depth)
2     as (
3 select cast(ename as varchar(100)),
4        mgr, 0
5   from emp
```

```

6   where ename = 'MILLER'
7   union all
8 select cast(x.tree+'-->'+e.ename as varchar(100)),
9       e.mgr, x.depth+1
10  from emp e, x
11  where x.mgr = e.empno
12 )
13 select tree leaf branch root
14  from x
15 where depth = 2

```

Esta solução pode funcionar em outros bancos de dados se o operador de concatenação for alterado. Portanto, mude para || para DB2 ou CONCAT para PostgreSQL.

## MySQL

Isto é semelhante à solução anterior, mas também precisa da palavra-chave RECURSIVE:

```

1   with recursive x (tree,mgr,depth)
2     as (
3 select cast(ename as varchar(100)),
4         mgr, 0
5   from emp
6  where ename = 'MILLER'
7  union all
8 select cast(concat(x.tree,'-->',emp.ename) as char(100)),
9         e.mgr, x.depth+1
10  from emp e, x
11  where x.mgr = e.empno
12 )
13 select tree leaf branch root
14  from x
15 where depth = 2

```

## Oráculo

Use a função SYS\_CONNECT\_BY\_PATH para retornar MILLER; O gerente da MILLER, CLARK; e depois o empresário de CLARK, KING. Use a cláusula CONNECT BY para percorrer a árvore:

```

1 select ltrim(
2         sys_connect_by_path(ename,'-->'),
3         '-->') leaf branch root
4   from emp
5  where level = 3
6  start with ename = 'MILLER'
1 connect by prior mgr = empno

```

### DB2, SQL Server, PostgreSQL e MySQL

A abordagem aqui é começar no nó folha e caminhar até a raiz (como prática útil, tente caminhar na outra direção). A parte superior de UNION ALL simplesmente encontra a linha do funcionário MILLER (o nó folha). A parte inferior de UNION ALL encontra o funcionário que é o gerente de MILLER e depois encontra o gerente dessa pessoa, e esse processo de localização do “gerente do gerente” se repete até que o processamento pare no gerente de nível mais alto (o nó raiz). O valor de DEPTH começa em 0 e aumenta automaticamente em 1 cada vez que um gerenciador é encontrado. DEPTH é um valor que o DB2 mantém para você quando você executa uma consulta recursiva.



Para uma introdução interessante e aprofundada à cláusula WITH com foco em seu uso recursivamente, consulte o artigo de Jonathan Gennick [“Comprendendo a cláusula WITH”](https://www.gennick.com/outside-the-box/understanding-the-with-clause/) (<https://www.gennick.com/outside-the-box/understanding-the-with-clause/>).

A seguir, a segunda consulta do UNION ALL une a visão recursiva X à tabela EMP, para definir o relacionamento pai-filho. A consulta neste ponto, usando o operador de concatenação do SQL Server, é a seguinte:

```
with x (tree,mgr,depth)
  as (
select cast(ename as varchar(100)),
       mgr, 0
  from emp
 where ename = 'MILLER'
 union all
select cast(x.tree+'-->'+e.ename as varchar(100)),
       e.mgr, x.depth+1
  from emp e, x
 where x.mgr = e.empno
)
select tree leaf____branch____root
  from x

TREE          DEPTH
-----  -----
MILLER        0
CLARK         1
KING          2
```

Neste ponto, o cerne do problema foi resolvido; começando por MILLER, retorne o relacionamento hierárquico completo de baixo para cima. O que resta então é apenas formatação. Como a travessia da árvore é recursiva, basta concatenar o ENAME atual do EMP com o anterior, o que fornece o seguinte conjunto de resultados:

```

with x (tree,mgr,depth)
  as (
select cast(ename as varchar(100)),
       mgr, 0
  from emp
 where ename = 'MILLER'
 union all
select cast(x.tree+'-->'||e.ename as varchar(100)),
       e.mgr, x.depth+1
  from emp e, x
 where x.mgr = e.empno
)
select depth, tree
  from x

```

DEPTH TREE

---

```

0 MILLER
1 MILLER-->CLARK
2 MILLER-->CLARK-->KING

```

A etapa final é manter apenas a última linha da hierarquia. Existem várias maneiras de fazer isso, mas a solução usa DEPTH para determinar quando a raiz é alcançada (obviamente, se CLARK tiver um gerenciador diferente de KING, o filtro em DEPTH teria que mudar; para uma solução mais genérica que não requer tal filtrar, veja a próxima receita).

### Oracle

A cláusula CONNECT BY faz todo o trabalho na solução Oracle. Começando com MILLER, você percorre todo o caminho até KING sem a necessidade de junções. A expressão na cláusula CONNECT BY define o relacionamento dos dados e como a árvore será percorrida:

```

select ename
  from emp
 start with ename = 'MILLER'
connect by prior mgr = empno

ENAME
-----
MILLER
CLARK
KING

```

A palavra-chave PRIOR permite acessar valores do registro anterior na hierarquia. Assim, para qualquer EMPNO, você pode usar PRIOR MGR para acessar o número do gerente daquele funcionário. Ao ver uma cláusula como CONNECT BY PRIOR MGR = EMPNO, pense nessa cláusula como expressando uma junção entre, neste caso, pai e filho.



Para saber mais sobre CONNECT BY e seu uso em consultas hierárquicas, “[Consultas hierárquicas no Oracle \(https://oracle-base.com/articles/misc/hierarchical-queries\)](https://oracle-base.com/articles/misc/hierarchical-queries)” é uma boa visão geral.

Neste ponto, você exibiu com êxito a hierarquia completa começando em MILLER e terminando em KING. O problema está em grande parte resolvido. Tudo o que resta é a formatação. Use a função SYS\_CONNECT\_BY\_PATH para anexar cada ENAME ao anterior:

```
select sys_connect_by_path(ename,'-->') tree
      from emp
     start with ename = 'MILLER'
connect by prior mgr = empno
```

TREE  
-----  
-->MILLER  
-->MILLER-->CLARK  
-->MILLER-->CLARK-->KING

Como você está interessado apenas na hierarquia completa, você pode filtrar na pseudocoluna LEVEL (uma abordagem mais genérica é mostrada na próxima receita):

```
select sys_connect_by_path(ename,'-->') tree
      from emp
     where level = 3
     start with ename = 'MILLER'
connect by prior mgr = empno
```

TREE  
-----  
-->MILLER-->CLARK-->KING

A etapa final é usar a função LTRIM para remover o início -> do conjunto de resultados.

## 13.3 Criando uma visão hierárquica de uma tabela

### Problema

Você deseja retornar um conjunto de resultados que descreva a hierarquia de uma tabela inteira. No caso da tabela EMP, o funcionário KING não tem gerente, então KING é o nó raiz. Você deseja exibir, começando por KING, todos os funcionários da KING e todos os funcionários (se houver) dos subordinados da KING. Por fim, você deseja retornar o seguinte conjunto de resultados:

```
EMP_TREE
-----
KING
KING - BLAKE
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK
KING - CLARK - MILLER
KING - JONES
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

## Solução

### DB2, PostgreSQL e SQL Server

Use a cláusula recursiva COM para começar a construir a hierarquia no KING e, por fim, exibir todos os funcionários. A solução a seguir usa o operador de concatenação do DB2 (||). Os usuários do SQL Server usam o operador de concatenação (+) e o MySQL usa a função CONCAT. Além dos operadores de concatenação, a solução funcionará como está em ambos os RDBMSs:

```
1  with x (ename,empno)
2    as (
3 select cast(ename as varchar(100)),empno
4   from emp
5  where mgr is null
6  union all
7 select cast(x.ename||' - '||e.ename as varchar(100)),
8       e.empno
9   from emp e, x
10  where e.mgr = x.empno
11 )
12 select ename as emp_tree
13   from x
14  order by 1
```

### MySQL

O MySQL também precisa da palavra-chave RECURSIVE:

```
1  with recursive x (ename,empno)
2    as (
3 select cast(ename as varchar(100)),empno
4   from emp
5  where mgr is null
```

```
5 union all
6 select cast(concat(x.ename,' - ',e.ename) as varchar(100)),
7      e.empno
8   from emp e, x
9  where e.mgr = x.empno
11 )
12 select ename as emp_tree
13   from x
14  order by 1
```

## Oracle

Use a função CONNECT BY para definir a hierarquia. Use a função SYS\_CONNECT\_BY\_PATH para formatar a saída de acordo:

```
1 select ltrim(
2           sys_connect_by_path(ename,' - '),
3           ' - ') emp_tree
4   from emp
5  start with mgr is null
6 connect by prior empno=mgr
7  order by 1
```

Esta solução difere da receita anterior porque não inclui nenhum filtro na pseudocoluna LEVEL. Sem o filtro, todas as árvores possíveis (onde PRIOR EMPNO=MGR) são exibidas.

## Discussão

### DB2, MySQL, PostgreSQL e SQL Server

O primeiro passo é identificar a linha raiz (funcionário KING) na parte superior de UNION ALL na visão recursiva X. O próximo passo é encontrar os subordinados de KING, e seus subordinados, se houver, juntando a visão recursiva X a tabela EMP. A recursão continuará até que você devolva todos os funcionários. Sem a formatação que você vê no conjunto de resultados final, o conjunto de resultados retornado pela visualização recursiva X é mostrado aqui:

```
withx (ename,empno)
  as (
select cast(ename as varchar(100)),empno
  from emp
 where mgr is null
union all
select cast(e.ename as varchar(100)),e.empno
  from emp e, x
 where e.mgr = x.empno
)
select ename emp_tree
  from x
```

```
EMP_TREE
-----
KING
JONES
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
WARD
MARTIN
TURNER
JAMES
CLARK
MILLER
```

Todas as linhas da hierarquia são retornadas (o que pode ser útil), mas sem a formatação você não consegue saber quem são os gerentes. Ao concatenar cada funcionário ao seu gerente, você retorna resultados mais significativos. Produza a saída desejada simplesmente usando o seguinte:

```
cast(x.ename+' '+e.ename as varchar(100))
```

na cláusula SELECT da parte inferior de UNION ALL na visão recursiva X.

A cláusula WITH é extremamente útil na resolução deste tipo de problema, pois a hierarquia pode mudar (por exemplo, nós folha tornam-se nós ramificados) sem qualquer necessidade de modificar a consulta.

### Oracle

A cláusula CONNECT BY retorna as linhas na hierarquia. A cláusula START WITH define a linha raiz. Se você executar a solução sem SYS\_CONNECT\_BY\_PATH, poderá ver que as linhas corretas são retornadas (o que pode ser útil), mas não formatadas para expressar o relacionamento das linhas:

```
select ename emp_tree
  from emp
 start with mgr is null
connect by prior empno = mgr
```

```
EMP_TREE
-----
KING
JONES
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
```

```
WARD  
MARTIN  
TURNER  
JAMES  
CLARK  
MILLER
```

Usando a pseudocoluna LEVEL e a função LPAD, você pode ver a hierarquia mais claramente e, em última análise, ver por que SYS\_CONNECT\_BY\_PATH retorna os resultados que você vê na saída desejada mostrada anteriormente:

```
select lpad('.',2*level,'.')||ename emp_tree  
  from emp  
 start with mgr is null  
connect by prior empno = mgr
```

```
EMP_TREE  
-----  
.KING  
...JONES  
....SCOTT  
.....ADAMS  
.....FORD  
.....SMITH  
....BLAKE  
.....ALLEN  
.....WARD  
.....MARTIN  
.....TURNER  
.....JAMES  
....CLARK  
.....MILLER
```

O recuo nesta saída indica quem são os gerentes, aninhando subordinados sob seus superiores. Por exemplo, KING não funciona para ninguém. JONES trabalha para o KING. SCOTT trabalha para JONES. ADAMS trabalha para SCOTT.

Se você observar as linhas correspondentes da solução ao usar SYS\_CONNECT\_BY\_PATH, verá que SYS\_CONNECT\_BY\_PATH acumula a hierarquia para você. Ao chegar a um novo nó, você também verá todos os nós anteriores:

```
KING  
KING - JONES  
KING - JONES - SCOTT  
KING - JONES - SCOTT - ADAMS
```

## 13.4 Encontrando todas as linhas filhas para uma determinada linha pai

### Problema

Você deseja encontrar todos os funcionários que trabalham para a JONES, direta ou indiretamente (ou seja, trabalham para alguém que trabalha para a JONES). A lista de funcionários de JONES é mostrada aqui (JONES está incluído no conjunto de resultados):

```
ENAME
-----
JONES
SCOTT
ADAMS
FORD
SMITH
```

### Solução

Ser capaz de mover-se para o topo ou para a base de uma árvore é extremamente útil. Para esta solução, não é necessária nenhuma formatação especial. O objetivo é simplesmente devolver todos os funcionários que trabalham sob o comando do funcionário JONES, incluindo o próprio JONES. Esse tipo de consulta realmente mostra a utilidade de extensões SQL recursivas, como CONNECT BY do Oracle e cláusula WITH do SQL Server/DB2.

#### DB2, PostgreSQL e SQL Server

Use a cláusula recursiva WITH para encontrar todos os funcionários de JONES. Comece com JONES especificando WHERE ENAME = JONES na primeira das duas consultas de união:

```
1  with x (ename,empno)
2    as (
3 select ename,empno
4   from emp
5  where ename = 'JONES'
6  union all
7 select e.ename, e.empno
8   from emp e, x
9  where x.empno = e.mgr
10 )
11 select ename
12   from x
```

#### Oracle

Use a cláusula CONNECT BY e especifique START WITH ENAME = JONES para encontrar todos os funcionários de JONES:

```
1 select ename
1 from emp
```

```
2 start with ename = 'JONES'  
3 connect by prior empno = mgr
```

## Discussão

### DB2, MySQL, PostgreSQL e SQL Server

A cláusula recursiva WITH torna este um problema relativamente fácil de resolver. A primeira parte da cláusula WITH, a parte superior de UNION ALL, retorna a linha do funcionário JONES. Você precisa retornar ENAME para ver o nome e EMPNO para poder usá-lo para ingressar. A parte inferior do UNION ALL une recursivamente EMP.MGR a X.EMPNO. A condição de junção será aplicada até que o conjunto de resultados se esgote.

### Oracle

A cláusula START WTH informa à consulta para tornar JONES o nó raiz. A condição na cláusula CONNECT BY impulsiona a caminhada na árvore e será executada até que a condição não seja mais verdadeira.

## 13.5 Determinando quais linhas são nós folha, ramificação ou raiz

### Problema

Você deseja determinar que tipo de nó é uma determinada linha: uma folha, ramo ou raiz. Neste exemplo, um nó folha é um funcionário que não é gerente. Um nó de filial é um funcionário que é gerente e também possui um gerente. Um nó raiz é um funcionário sem gerente. Você deseja retornar 1 (TRUE) ou 0 (FALSE) para refletir o status de cada linha na hierarquia. Você deseja retornar o seguinte conjunto de resultados:

ENAME	IS_LEAF	IS_BRANCH	IS_ROOT
KING	0	0	1
JONES	0	1	0
SCOTT	0	1	0
FORD	0	1	0
CLARK	0	1	0
BLAKE	0	1	0
ADAMS	1	0	0
MILLER	1	0	0
JAMES	1	0	0
TURNER	1	0	0
ALLEN	1	0	0
WARD	1	0	0
MARTIN	1	0	0
SMITH	1	0	0

## Solução

É importante perceber que a tabela EMP é modelada em uma hierarquia de árvore, não em uma hierarquia recursiva, e o valor de MGR para nós raiz é NULL. Se o EMP fosse modelado para usar uma hierarquia recursiva, os nós raiz seriam auto-referenciados (ou seja, o valor do MGR para o funcionário KING seria o EMPNO do KING). Consideramos a auto-referência contra-intuitiva e, portanto, estamos usando valores NULL para o MGR dos nós raiz. Para usuários Oracle que usam CONNECT BY e usuários de DB2/SQL Server que usam WITH, você achará hierarquias de árvore mais fáceis de trabalhar e potencialmente mais eficientes do que hierarquias recursivas. Se você estiver em uma situação em que possui uma hierarquia recursiva e estiver usando CONNECT BY ou WITH, cuidado: você pode acabar com um loop no seu SQL. Você precisará codificar esses loops se estiver preso a hierarquias recursivas.

### DB2, PostgreSQL, MySQL e SQL Server

Use três subconsultas escalares para determinar o valor “Booleano” correto (1 ou 0) a ser retornado para cada tipo de nó:

```

1 select e.ename,
2       (select sign(count(*)) from emp d
3        where 0 =
4          (select count(*) from emp f
5           where f.mgr = e.empno)) as is_leaf,
6       (select sign(count(*)) from emp d
7        where d.mgr = e.empno
8           and e.mgr is not null) as is_branch,
9       (select sign(count(*)) from emp d
10      where d.empno = e.empno
11         and d.mgr is null) as is_root
12   from emp e
13 order by 4 desc,3 desc

```

### Oracle

A solução de subconsulta escalar também funcionará para Oracle e deverá ser usada se você estiver em uma versão do Oracle anterior ao Oracle Database 10g. A solução a seguir destaca funções integradas fornecidas pela Oracle (que foram introduzidas no Oracle Database 10g) para identificar linhas raiz e folha. As funções são CONNECT\_BY\_ROOT e CONNECT\_BY\_ISLEAF, respectivamente:

```

1 select ename,
2       connect_by_isleaf is_leaf,
3       (select count(*) from emp e
4        where e.mgr = emp.empno
5           and emp.mgr is not null
6           and rownum = 1) is_branch,
7       decode(ename,connect_by_root(ename),1,0) is_root
1   from emp

```

```

2 start with mgr is null
3 connect by prior empno = mgr
4 order by 4 desc, 3 desc

```

## Discussão

### DB2, PostgreSQL, MySQL e SQL Server

Esta solução simplesmente aplica as regras definidas na seção “Problema” para determinar folhas, galhos e raízes. A primeira etapa é determinar se um funcionário é um nó folha. Se o funcionário não for gerente (ninguém trabalha sob ele), então ele é um nó folha. A primeira subconsulta escalar, IS\_LEAF, é mostrada aqui:

```

select e.ename,
       (select sign(count(*)) from emp d
        where 0 =
              (select count(*) from emp f
               where f.mgr = e.empno)) as is_leaf
  from emp e
 order by 2 desc

```

ENAME	IS_LEAF
SMITH	1
ALLEN	1
WARD	1
ADAMS	1
TURNER	1
MARTIN	1
JAMES	1
MILLER	1
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0
KING	0

Como a saída para IS\_LEAF deve ser 0 ou 1, é necessário pegar o SIGN da operação COUNT(\*). Caso contrário, você obteria 14 em vez de 1 para linhas de folhas. Como alternativa, você pode usar uma tabela com apenas uma linha para contar, porque deseja retornar apenas 0 ou 1. Por exemplo:

```

select e.ename,
       (select count(*) from t1 d
        where not exists
              (select null from emp f
               where f.mgr = e.empno)) as is_leaf
  from emp e
 order by 2 desc

```

ENAME	IS_LEAF
SMITH	1
ALLEN	1
WARD	1
ADAMS	1
TURNER	1
MARTIN	1
JAMES	1
MILLER	1
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0
KING	0

A próxima etapa é encontrar nós de ramificação. Se um funcionário for um gerente (alguém trabalha para ele) e também trabalhar para outra pessoa, então o funcionário é um nó de filial. Os resultados da subconsulta escalar IS\_BRANCH são mostrados aqui:

```
select e.ename,
       (select sign(count(*)) from emp d
        where d.mgr = e.empno
        and e.mgr is not null) as is_branch
  from emp e
 order by 2 desc
```

ENAME	IS_BRANCH
JONES	1
BLAKE	1
SCOTT	1
CLARK	1
FORD	1
SMITH	0
TURNER	0
MILLER	0
JAMES	0
ADAMS	0
KING	0
ALLEN	0
MARTIN	0
WARD	0

Novamente, é necessário pegar o SINAL da operação COUNT(\*). Caso contrário, você obterá (potencialmente) valores maiores que 1 quando um nó for uma ramificação. Assim como a subconsulta escalar IS\_LEAF, você pode usar uma tabela com uma linha para evitar o uso de SIGN. A solução a seguir usa a tabela T1:

```

select e.ename,
       (select count(*) from t1 t
        where exists (
          select null from emp f
          where f.mgr = e.empno
            and e.mgr is not null)) as is_branch
  from emp e
 order by 2 desc

```

ENAME	IS_BRANCH
JONES	1
BLAKE	1
SCOTT	1
CLARK	1
FORD	1
SMITH	0
TURNER	0
MILLER	0
JAMES	0
ADAMS	0
KING	0
ALLEN	0
MARTIN	0
WARD	0

A última etapa é encontrar os nós raiz. Um nó raiz é definido como um funcionário que é gerente, mas não trabalha para mais ninguém. Na tabela EMP, apenas KING é um nó raiz. A subconsulta escalar IS\_ROOT é mostrada aqui:

```

select e.ename,
       (select sign(count(*)) from emp d
        where d.empno = e.empno
          and d.mgr is null) as is_root
  from emp e
 order by 2 desc

```

ENAME	IS_ROOT
KING	1
SMITH	0
ALLEN	0
WARD	0
JONES	0
TURNER	0
JAMES	0
MILLER	0
FORD	0
ADAMS	0
MARTIN	0

BLAKE	0
CLARK	0
SCOTT	0

Como EMP é uma pequena tabela de 14 linhas, é fácil ver que o funcionário KING é o único nó raiz, portanto, neste caso, tomar o SIGN da operação COUNT(\*) não é estritamente necessário. Se puder haver vários nós raiz, você poderá usar SIGN ou uma tabela de uma linha na subconsulta escalar, como mostrado anteriormente para IS\_BRANCH e IS\_LEAF.

### Oracle

Para aqueles que usam versões do Oracle anteriores ao Oracle Database 10g, você pode acompanhar a discussão para os outros RDBMSs, pois essa solução funcionará (sem modificações) no Oracle. Se você estiver no Oracle Database 10g ou posterior, você pode querer aproveitar duas funções para tornar a identificação dos nós raiz e folha uma tarefa simples: elas são CONNECT\_BY\_ROOT e CONNECT\_BY\_ISLEAF, respectivamente. No momento em que este livro foi escrito, era necessário usar CONNECT BY em sua instrução SQL para que você pudesse usar CONNECT\_BY\_ROOT e CONNECT\_BY\_ISLEAF. A primeira etapa é encontrar os nós folha usando CONNECT\_BY\_ISLEAF da seguinte maneira:

```
select ename,
       connect_by_isleaf is_leaf
  from emp
 start with mgr is null
connect by prior empno = mgr
order by 2 desc
```

ENAME	IS_LEAF
ADAMS	1
SMITH	1
ALLEN	1
TURNER	1
MARTIN	1
WARD	1
JAMES	1
MILLER	1
KING	0
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0

A próxima etapa é usar uma subconsulta escalar para encontrar os nós da ramificação. Nós de filial são funcionários que são gerentes, mas que também trabalham para outra pessoa:

```

select ename,
       (select count(*) from emp e
        where e.mgr = emp.empno
        and emp.mgr is not null
        and rownum = 1) is_branch
  from emp
 start with mgr is null
connect by prior empno = mgr
order by 2 desc

```

ENAME	IS_BRANCH
JONES	1
SCOTT	1
BLAKE	1
FORD	1
CLARK	1
KING	0
MARTIN	0
MILLER	0
JAMES	0
TURNER	0
WARD	0
ADAMS	0
ALLEN	0
SMITH	0

O filtro em ROWNUM é necessário para garantir que você retorne uma contagem de 1 ou 0 e nada mais.

A última etapa é identificar os nós raiz usando a função CONNECT\_BY\_ROOT. A solução encontra o ENAME do nó raiz e o compara com todas as linhas retornadas pela consulta. Se houver uma correspondência, essa linha será o nó raiz:

```

select ename,
       decode(ename,connect_by_root(ename),1,0) is_root
  from emp
 start with mgr is null
connect by prior empno = mgr
order by 2 desc

```

ENAME	IS_ROOT
KING	1
JONES	0
SCOTT	0
ADAMS	0
FORD	0
SMITH	0
BLAKE	0
ALLEN	0
WARD	0

MARTIN	0
TURNER	0
JAMES	0
CLARK	0
MILLER	0

A função SYS\_CONNECT\_BY\_PATH acumula uma hierarquia começando no valor raiz, conforme mostrado aqui:

```
select ename,
       ltrim(sys_connect_by_path(ename,'',''),'') path
  from emp
 start with mgr is null
 connect by prior empno=mgr
```

ENAME	PATH
KING	KING
JONES	KING,JONES
SCOTT	KING,JONES,SCOTT
ADAMS	KING,JONES,SCOTT,ADAMS
FORD	KING,JONES,FORD
SMITH	KING,JONES,FORD,SMITH
BLAKE	KING,BLAKE
ALLEN	KING,BLAKE,ALLEN
WARD	KING,BLAKE,WARD
MARTIN	KING,BLAKE,MARTIN
TURNER	KING,BLAKE,TURNER
JAMES	KING,BLAKE,JAMES
CLARK	KING,CLARK
MILLER	KING,CLARK,MILLER

Para obter a linha raiz, simplesmente substring o primeiro ENAME em PATH:

```
select ename,
       substr(root,1,instr(root,',')-1) root
  from (
select ename,
       ltrim(sys_connect_by_path(ename,'',''),'') root
  from emp
 start with mgr is null
 connect by prior empno=mgr
      )
```

ENAME	ROOT
KING	KING
JONES	KING
SCOTT	KING
ADAMS	KING
FORD	KING
SMITH	KING
BLAKE	KING

ALLEN	KING
WARD	KING
MARTIN	KING
TURNER	KING
JAMES	KING
CLARK	KING
MILLER	KING

A última etapa é sinalizar o resultado da coluna ROOT; se for NULL, essa é sua linha raiz.

## 13.6 Resumindo

A disseminação de CTEs por todos os fornecedores tornou as abordagens padronizadas para consultas hierárquicas muito mais viáveis. Este é um grande avanço, pois os relacionamentos hierárquicos aparecem em muitos tipos de dados, mesmo em dados onde o relacionamento não é necessariamente planejado, portanto, as consultas precisam considerá-lo.

# CAPÍTULO 14

## Probabilidades e fins

Este capítulo contém perguntas que não cabem em nenhum outro capítulo, seja porque o capítulo ao qual pertenceriam já é longo o suficiente, ou porque os problemas que resolvem são mais divertidos do que realistas. Este capítulo pretende ser um capítulo “divertido”, pois as receitas aqui podem ou não ser receitas que você realmente usaria; no entanto, as questões são interessantes e queríamos incluí-las neste livro.

### 14.1 Criando relatórios Cross-Tab usando o operador PIVOT do SQL Server

#### Problema

Você deseja criar um relatório de tabela cruzada para transformar as linhas do seu conjunto de resultados em colunas. Você conhece os métodos tradicionais de pivotamento, mas gostaria de tentar algo diferente. Em particular, você deseja retornar o seguinte conjunto de resultados sem usar expressões CASE ou junções:

DEPT_10	DEPT_20	DEPT_30	DEPT_40
3	5	6	0

#### Solução

Use o operador PIVOT para criar o conjunto de resultados necessário sem expressões CASE ou junções adicionais:

```

1 select [10] as dept_10,
2      [20] as dept_20,
3      [30] as dept_30,
4      [40] as dept_40
5  from (select deptno, empno from emp) driver

```

```

6 pivot (
7 count(driver.empno)
8 for driver.deptno in ( [10],[20],[30],[40] )
9 ) as empPivot

```

## Discussão

O operador PIVOT pode parecer estranho à primeira vista, mas a operação que ele executa na solução é tecnicamente igual à consulta de transposição mais familiar mostrada aqui:

```

select sum(case deptno when 10 then 1 else 0 end) as dept_10,
       sum(case deptno when 20 then 1 else 0 end) as dept_20,
       sum(case deptno when 30 then 1 else 0 end) as dept_30,
       sum(case deptno when 40 then 1 else 0 end) as dept_40
  from emp

```

DEPT_10	DEPT_20	DEPT_30	DEPT_40
3	5	6	0

Agora que você sabe o que está essencialmente acontecendo, vamos detalhar o que o operador PIVOT está fazendo. A linha 5 da solução mostra uma visualização inline chamada DRIVER:

```
from (select deptno, empno from emp) driver
```

Usamos o alias DRIVER porque as linhas desta visualização embutida (ou expressão de tabela) alimentam diretamente a operação PIVOT. O operador PIVOT gira as linhas em colunas avaliando os itens listados na linha 8 da lista FOR (mostrada aqui):

```
for driver.deptno in ( [10],[20],[30],[40] )
```

A avaliação é mais ou menos assim:

1. Se houver DEPTNOs com valor 10, execute a operação agregada definida (COUNT(DRIVER.EMPNO)) para essas linhas.
2. Repita para DEPTNOs 20, 30 e 40.

Os itens listados entre colchetes na linha 8 servem não apenas para definir valores para os quais a agregação é realizada; os itens também se tornam os nomes das colunas no conjunto de resultados (sem os colchetes). Na cláusula SELECT da solução, os itens da lista FOR são referenciados e têm alias. Se você não criar nomes alternativos para os itens da lista FOR, os nomes das colunas se tornarão os itens da lista FOR sem colchetes.

Curiosamente, como a visualização inline DRIVER é apenas isso - uma visualização inline - você pode colocar SQL mais complexo lá. Por exemplo, considere a situação em que você deseja modificar o conjunto de resultados de forma que o nome real do departamento seja o nome da coluna. Aqui estão listadas as linhas na tabela DEPT:

```
select * from dept
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Você deseja usar PIVOT para retornar o seguinte conjunto de resultados:

ACCOUNTING	RESEARCH	SALES	OPERATIONS
3	5	6	0

Como a visualização embutida DRIVER pode ser praticamente qualquer expressão de tabela válida, você pode realizar a junção da tabela EMP com a tabela DEPT e depois fazer com que o PIVOT avalie essas linhas. A consulta a seguir retornará o conjunto de resultados desejado:

```
select [ACCOUNTING] as ACCOUNTING,
       [SALES]      as SALES,
       [RESEARCH]    as RESEARCH,
       [OPERATIONS] as OPERATIONS
  from (
    select d.dname, e.empno
      from emp e,dept d
     where e.deptno=d.deptno
       ) driver
 pivot (
  count(driver.empno)
  for driver.dname in ([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
 ) as empPivot
```

Como você pode ver, PIVOT oferece uma abordagem interessante na rotação de conjuntos de resultados. Independentemente de você preferir usá-lo em vez dos métodos tradicionais de dinamização, é bom ter outra ferramenta em sua caixa de ferramentas.

## 14.2 Destornando um relatório de tabela cruzada usando o operador UNPIVOT do SQL Server

### Problema

Você tem um conjunto de resultados dinâmico (ou simplesmente uma tabela de fatos) e deseja desdinamizá-lo. Por exemplo, em vez de ter um conjunto de resultados com uma linha e quatro colunas, você deseja retornar um conjunto de resultados com duas colunas e quatro linhas. Usando o conjunto de resultados da receita anterior, você deseja convertê-lo a partir disto:

ACCOUNTING	RESEARCH	SALES	OPERATIONS
3	5	6	0

para isso:

DNAME	CNT
ACCOUNTING	3
RESEARCH	5
SALES	6
OPERATIONS	0

## Solução

Você não achou que o SQL Server lhe daria a capacidade de PIVOT sem poder UNPIVOT, não é? Para desdinamizar o conjunto de resultados, basta usá-lo como driver e deixar o operador UNPIVOT fazer todo o trabalho. Tudo que você precisa fazer é especificar os nomes das colunas:

```

1  select DNAME, CNT
2    from (
3      select [ACCOUNTING] as ACCOUNTING,
4            [SALES]      as SALES,
5            [RESEARCH]    as RESEARCH,
6            [OPERATIONS] as OPERATIONS
7      from (
8        select d.dname, e.empno
9          from emp e,dept d
10         where e.deptno=d.deptno
11
12       ) driver
13     pivot (
14       count(driver.empno)
15       for driver.dname in ([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
16     ) as empPivot
17   ) new_driver
18 unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
19   ) as un_pivot

```

Idealmente, antes de ler esta receita você leu a anterior, porque a visualização inline NEW\_DRIVER é simplesmente o código da receita anterior (se você não o entende, consulte a receita anterior antes de olhar para esta receita). um). Como as linhas 3 a 16 consistem em código que você já viu, a única sintaxe nova está na linha 18, onde você usa UNPIVOT.

O comando UNPIVOT simplesmente analisa o conjunto de resultados de NEW\_DRIVER e avalia cada coluna e linha. Por exemplo, o operador UNPIVOT avalia os nomes das colunas de NEW\_DRIVER. Ao encontrar ACCOUNTING, ele transforma o nome da coluna ACCOUNTING em um valor de linha (na coluna DNAME). Também recebe o valor de ACCOUNTING de NEW\_DRIVER (que é 3) e retorna isso também como parte da linha ACCOUNTING

(na coluna CNT). UNPIVOT faz isso para cada um dos itens especificados na lista FOR e simplesmente retorna cada um deles como uma linha.

O novo conjunto de resultados agora é reduzido e possui duas colunas, DNAME e CNT, com quatro linhas:

```

select DNAME, CNT
  from (
    select [ACCOUNTING] as ACCOUNTING,
           [SALES]      as SALES,
           [RESEARCH]    as RESEARCH,
           [OPERATIONS] as OPERATIONS
      from (
        select d.dname, e.empno
          from emp e,dept d
         where e.deptno=d.deptno

      ) driver
 pivot (
  count(driver.empno)
  for driver.dname in ( [ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS] )
 ) as empPivot
) new_driver
unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
) as un_pivot

DNAME          CNT
-----
ACCOUNTING      3
RESEARCH         5
SALES            6
OPERATIONS       0

```

## 14.3 Transpondo um conjunto de resultados usando a cláusula MODEL da Oracle

### Problema

Assim como na primeira receita deste capítulo, você deseja encontrar uma alternativa às técnicas tradicionais de pivotamento que já viu. Você quer experimentar a cláusula MODEL da Oracle. Ao contrário do operador PIVOT do SQL Server, a cláusula MODEL do Oracle não existe para transpor conjuntos de resultados; na verdade, seria bastante correto dizer que a aplicação da cláusula MODEL para pivotamento seria um uso indevido e claramente não era o objetivo da cláusula MODEL. No entanto, a cláusula MODEL proporciona uma abordagem interessante para um problema comum. Para este problema específico, você deseja transformar o seguinte conjunto de resultados:

```
select deptno, count(*) cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

para isso:

D10	D20	D30
3	5	6

## Solução

Use agregação e expressões CASE na cláusula MODEL da mesma forma que você as usaria se estivesse girando com técnicas tradicionais. A principal diferença neste caso é que você usa arrays para armazenar os valores da agregação e retornar os arrays no conjunto de resultados:

```
select max(d10) d10,
       max(d20) d20,
       max(d30) d30
  from (
select d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
model
dimension by(deptno d)
measures(deptno, cnt d10, cnt d20, cnt d30)
rules(
      d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
      d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
      d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
    )
  )
```

## Discussão

A cláusula MODEL é uma adição poderosa à caixa de ferramentas Oracle SQL. Depois de começar a trabalhar com MODEL, você notará recursos úteis, como iteração, acesso de array a valores de linha, a capacidade de “inserir” linhas em um conjunto de resultados e a capacidade de construir modelos de referência. Você verá rapidamente que esta receita não tira vantagem de nenhum dos recursos interessantes que a cláusula MODEL oferece, mas é bom poder olhar para um problema de vários ângulos e usar diferentes recursos de maneiras inesperadas (se não for por outro motivo). razão do que aprender onde certos recursos são mais úteis que outros).

O primeiro passo para entender a solução é examinar a visualização embutida na cláusula FROM. A visualização embutida simplesmente conta o número de funcionários em cada DEPTNO na tabela EMP. Os resultados são mostrados aqui:

```
select deptno, count(*) cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

Este conjunto de resultados é fornecido ao MODEL para trabalhar. Examinando a cláusula MODEL, você vê três subseções que se destacam: DIMENSION BY, MEASURES e RULES. Vamos começar com MEDIDAS.

Os itens na lista MEDIDAS são simplesmente os arrays que você está declarando para esta consulta. A consulta usa quatro matrizes: DEPTNO, D10, D20 e D30. Assim como as colunas em uma lista SELECT, os arrays na lista MEASURES podem ter aliases. Como você pode ver, três dos quatro arrays são, na verdade, CNT na visualização embutida.

Se a lista MEASURES contém nossos arrays, então os itens na subseção DIMENSION BY são os índices do array. Considere o seguinte: o array D10 é simplesmente um apelido para CNT. Se você observar o conjunto de resultados da visualização embutida anterior, verá que CNT tem três valores: 3, 5 e 6. Ao criar uma matriz de CNT, você está criando uma matriz com três elementos, a saber, os três inteiros: 3, 5 e 6. Agora, como você acessa esses valores do array individualmente? Você usa o índice da matriz. O índice, definido na subseção DIMENSION BY, possui os valores 10, 20 e 30 (do conjunto de resultados acima). Assim, por exemplo, a seguinte expressão:

`d10[10]`

seria avaliado como 3, pois você está acessando o valor de CNT na matriz D10 para DEPTNO 10 (que é 3).

Como todas as três matrizes (D10, D20, D30) contêm os valores do CNT, todas as três têm os mesmos resultados. Como então colocamos a contagem adequada na matriz correta? Insira a subcláusula RULES. Se você observar o conjunto de resultados da visualização embutida mostrada anteriormente, verá que os valores para DEPTNO são 10, 20 e 30. As expressões que envolvem CASE na cláusula RULES simplesmente avaliam cada valor na matriz DEPTNO:

- Se o valor for 10, armazene o CNT para DEPTNO 10 em D10[10] ou então armazene 0.
- Se o valor for 20, armazene o CNT para DEPTNO 20 em D20[20] ou então armazene 0.
- Se o valor for 30, armazene o CNT para DEPTNO 30 em D30[30] ou então armazene 0.

Se você se sentir um pouco como Alice caindo na toca do coelho, não se preocupe; apenas pare e execute o que foi discutido até agora. O conjunto de resultados a seguir representa o que foi discutido. Às vezes é mais fácil ler um pouco, olhar o código que realmente executa o que você acabou de ler e depois voltar e lê-lo novamente. O seguinte é bastante simples quando você o vê em ação:

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
model
dimension by(deptno d)
measures(deptno, cnt d10, cnt d20, cnt d30)
rules(
  d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
  d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
  d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
)

DEPTNO      D10      D20      D30
-----  -----  -----  -----
  10          3          0          0
  20          0          5          0
  30          0          0          6
```

Como você pode ver, a subseção RULES é o que alterou os valores em cada array. Se você ainda não entendeu, simplesmente execute a mesma consulta, mas comente as expressões na subclasse RULES:

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
model
dimension by(deptno d)
measures(deptno, cnt d10, cnt d20, cnt d30)
rules(
/*
  d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
  d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
  d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
*/
)

DEPTNO      D10      D20      D30
-----  -----  -----  -----
  10          3          3          3
  20          5          5          5
  30          6          6          6
```

Deve ficar claro agora que o conjunto de resultados da cláusula MODEL é o mesmo da visualização embutida, exceto que a operação COUNT tem os alias D10, D20 e D30. A seguinte consulta comprova isso:

```
select deptno, count(*) d10, count(*) d20, count(*) d30
  from emp
 group by deptno
```

DEPTNO	D10	D20	D30
10	3	3	3
20	5	5	5
30	6	6	6

Portanto, tudo o que a cláusula MODEL fez foi pegar os valores de DEPTNO e CNT, colocá-los em arrays e então certificar-se de que cada array represente um único DEPTNO. Neste ponto, as matrizes D10, D20 e D30 têm, cada uma, um único valor diferente de zero que representa o CNT para um determinado DEPTNO. O conjunto de resultados já está transposto, e tudo o que resta é usar a função agregada MAX (você poderia ter usado MIN ou SUM; não faria diferença neste caso) para retornar apenas uma linha:

```
seleccione máximo (d10) d10,
máx(d20) d20,
máx(d30) d30

de (select max(d10) d10,
        max(d20) d20,
        max(d30) d30
     from (
select d10,d20,d30
     from ( select deptno, count(*) cnt from emp group by deptno )
model
dimension by(deptno d)
measures(deptno, cnt d10, cnt d20, cnt d30)
rules(
      d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
      d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
      d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
    )
  )
```

D10	D20	D30
3	5	6

## 14.4 Extraindo elementos de uma string de Locais não fixos

### Problema

Você tem um campo de string que contém dados de log serializados. Você deseja analisar a string e extrair as informações relevantes. Infelizmente, as informações relevantes não estão em pontos fixos da string. Em vez disso, você deve usar o fato de que existem certos caracteres em torno das informações que você precisa, para extrair essas informações. Por exemplo, considere as seguintes sequências:

```

xxxxxabc[867]xxx[-]xxxx[5309]xxxxx
xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx
call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx
film:[non_marked]qq:[unit]tailpipe:[withabana?]80sxxxxx

```

Você deseja extrair os valores entre colchetes, retornando o seguinte conjunto de resultados:

FIRST_VAL	SECOND_VAL	LAST_VAL
867	-	5309
11271978	4	Joe
F_GET_ROWS()	ROSEWOOD...SIR	44400002
non_marked	unit	withabana?

## Solução

Apesar de não saber a localização exata dos valores interessantes na sequência, você sabe que eles estão localizados entre colchetes [], e sabe que existem três deles. Use a função interna INSTR do Oracle para encontrar a localização dos colchetes. Use a função integrada SUBSTR para extrair os valores da string. A visualização V conterá as strings a serem analisadas e é definida da seguinte forma (seu uso é estritamente para facilitar a leitura):

```

create view V
as
select 'xxxxxabc[867]xxx[-]xxxx[5309]xxxxx' msg
  from dual
 union all
select 'xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx' msg
  from dual
 union all
select 'call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx' msg
  from dual
 union all
select 'film:[non_marked]qq:[unit]tailpipe:[withabana?]80sxxxxx' msg
  from dual

1 select substr(msg,
2      instr(msg,'[',1,1)+1,
3      instr(msg,']',1,1)-instr(msg,['',1,1)-1) first_val,
4      substr(msg,
5      instr(msg,['',1,2)+1,
6      instr(msg,']',1,2)-instr(msg,['',1,2)-1) second_val,
7      substr(msg,
8      instr(msg,['',-1,1)+1,
9      instr(msg,']',-1,1)-instr(msg,['',-1,1)-1) last_val
10  from V

```

## Discussão

Usar a função interna INSTR do Oracle torna esse problema bastante simples de resolver. Como você sabe que os valores que procura estão entre [], e que existem três conjuntos de [], o primeiro passo para esta solução é simplesmente usar INSTR para encontrar as posições numéricas de [] em cada string. O exemplo a seguir retorna a posição numérica dos colchetes de abertura e fechamento em cada linha:

```
select instr(msg,['',1,1) "1st_[",
                instr(msg,']',1,1) "]_1st",
                instr(msg,['',1,2) "2nd_[",
                instr(msg,']',1,2) "]_2nd",
                instr(msg,['',-1,1) "3rd_[",
                instr(msg,']',-1,1) "]_3rd"
from V
```

1st_[ ]_1st	2nd_[ ]_2nd	3rd_[ ]_3rd
9      13	17    19	24    29
11     20	28    30	34    38
6      19	23    38	42    51
6      17	21    26	36    49

Neste ponto, o trabalho árduo está concluído. Tudo o que resta é inserir as posições numéricas em SUBSTR para analisar MSG nesses locais. Você notará que na solução completa há alguma aritmética simples nos valores retornados por INSTR, particularmente +1 e -1; isso é necessário para garantir que o colchete de abertura, [, não seja retornado no conjunto de resultados final. Listada aqui está a solução menos adição e subtração de 1 nos valores de retorno do INSTR; observe como cada valor tem um colchete inicial:

```
select substr(msg,
              instr(msg,['',1,1),
              instr(msg,']',1,1)-instr(msg,['',1,1)) first_val,
              substr(msg,
              instr(msg,['',1,2),
              instr(msg,']',1,2)-instr(msg,['',1,2)) second_val,
              substr(msg,
              instr(msg,['',-1,1),
              instr(msg,']',-1,1)-instr(msg,['',-1,1)) last_val
from V
```

FIRST_VAL	SECOND_VAL	LAST_VAL
[867	[ -	[5309
[11271978	[ 4	[Joe
[F_GET_ROWS()	[ROSEWOOD...SIR	[44400002
[non_marked	[unit	[withabanana?

No conjunto de resultados anterior, você pode ver que o colchete aberto está lá. Você pode estar pensando: “OK, coloque a adição de 1 em INSTR de volta e o colchete inicial desaparece. Por que precisamos subtrair 1?”

A razão é esta: se você colocar a adição de volta, mas deixar de fora a subtração, você acaba incluindo o colchete de fechamento, como mostrado aqui:

```
select substr(msg,
    instr(msg,'[',1,1)+1,
    instr(msg,']',1,1)-instr(msg,['',1,1)) first_val,
    substr(msg,
    instr(msg,['',1,2)+1,
    instr(msg,']',1,2)-instr(msg,['',1,2)) second_val,
    substr(msg,
    instr(msg,['',-1,1)+1,
    instr(msg,']',-1,1)-instr(msg,['',-1,1)) last_val
from V

FIRST_VAL      SECOND_VAL      LAST_VAL
-----  -----  -----
867]          - ]            5309]
11271978]       4]            Joe]
F_GET_ROWS()] ROSEWOOD...SIR] 44400002]
non_marked]     unit]        withabanana?]
```

Neste ponto, deve estar claro: para garantir que você não inclua nenhum dos colchetes, você deve adicionar um ao índice inicial e subtrair um do índice final.

## 14.5 Encontrando o número de dias em um ano (uma solução alternativa para Oracle)

### Problema

Você deseja encontrar o número de dias em um ano.



Esta receita apresenta uma solução alternativa para “Determinar o número de dias em um ano” de [Capítulo 9](#). Esta solução é específica para Oracle.

### Solução

Use a função TO\_CHAR para formatar a última data do ano em um número do dia do ano de três dígitos:

```
1 select 'Days in 2021: ' ||
2      to_char(add_months(trunc(sysdate,'y'),12)-1,'DDD')
3      as report
4  from dual
5 union all
6 select 'Days in 2020: ' ||
1      to_char(add_months(trunc (
```

```

8      to_date('01-SEP-2020'), 'y'), 12)-1, 'DDD')
9  from dual

REPORT
-----
Days in 2021: 365
Days in 2020: 366

```

## Discussão

Comece usando a função TRUNC para retornar o primeiro dia do ano para a data especificada, como segue:

```

select trunc(to_date('01-SEP-2020'), 'y')
  from dual

TRUNC(TO_DA
-----
01-JAN-2020

```

A seguir, use ADD\_MONTHS para adicionar um ano (12 meses) à data truncada. Em seguida, subtraia um dia, chegando ao final do ano em que cai sua data original:

```

select add_months(
      trunc(to_date('01-SEP-2020'), 'y'),
      12) before_subtraction,
      add_months(
      trunc(to_date('01-SEP-2020'), 'y'),
      12)-1 after_subtraction
  from dual

BEFORE_SUBT AFTER_SUBTR
-----
01-JAN-2021 31-DEC-2020

```

Agora que você encontrou o último dia do ano com o qual está trabalhando, basta usar TO\_CHAR para retornar um número de três dígitos representando em qual dia (1º, 50º, etc.) do ano é o último dia:

```

select to_char(
      add_months(
      trunc(to_date('01-SEP-2020'), 'y'),
      12)-1, 'DDD') num_days_in_2020
  from dual

NUM
---
366

```

## 14.6 Procurando por strings alfanuméricas mistas

### Problema

Você tem uma coluna com dados alfanuméricos mistos. Você deseja retornar as linhas que possuem caracteres alfabéticos e numéricos; em outras palavras, se uma string tiver apenas números ou apenas letras, não a retorne. Os valores de retorno devem ter uma mistura de letras e números. Considere os seguintes dados:

```
STRINGS
-----
1010 switch
333
3453430278
ClassSummary
findRow 55
threes
```

O conjunto de resultados final deve conter apenas as linhas que possuem letras e números:

```
STRINGS
-----
1010 switch
findRow 55
```

### Solução

Use a função interna TRANSLATE para converter cada ocorrência de uma letra ou dígito em um caractere específico. Em seguida, mantenha apenas as strings que possuem pelo menos uma ocorrência de ambos. A solução usa sintaxe Oracle, mas tanto o DB2 quanto o PostgreSQL suportam TRANSLATE, portanto, modificar a solução para funcionar nessas plataformas deve ser trivial:

```
with v as (
  select 'ClassSummary' strings from dual union
  select '3453430278'      from dual union
  select 'findRow 55'        from dual union
  select '1010 switch'      from dual union
  select '333'               from dual union
  select 'threes'            from dual
)
select strings
  from (
    select strings,
           translate(
             strings,
             'abcdefghijklmnopqrstuvwxyz0123456789',
             rpad('#',26,'#')||rpad('*',10,'*')) translated
    from v
```

```
) x
whereinstr(translated,'#') > 0
and instr(translated,'*') > 0
```



Como alternativa à cláusula WITH, você pode usar uma visualização embutida ou simplesmente criar uma visualização.

## Discussão

A função TRANSLATE torna esse problema extremamente fácil de resolver. O primeiro passo é usar TRANSLATE para identificar todas as letras e todos os dígitos por caracteres de cerquilha (#) e asterisco (\*), respectivamente. Os resultados intermediários (da visualização in-line X) são os seguintes:

```
with v as (
  select 'ClassSummary' strings from dual union
  select '3453430278'      from dual union
  select 'findRow 55'       from dual union
  select '1010 switch'     from dual union
  select '333'              from dual union
  select 'threes'           from dual
)
select strings,
       translate(
         strings,
         'abcdefghijklmnopqrstuvwxyz0123456789',
         rpad('#',26,'#')||rpad('*',10,'*')) translated
from v
```

STRINGS	TRANSLATED
1010 switch	***** #####
333	***
3453430278	*****
ClassSummary	C#####S#####
findRow 55	####R## **
threes	#####

Neste ponto, é apenas uma questão de manter as linhas que possuem pelo menos uma instância de # e \*. Use a função INSTR para determinar se # e \* estão em uma string. Se esses dois caracteres estiverem, de fato, presentes, o valor retornado será maior que zero. As strings finais a serem retornadas, juntamente com seus valores traduzidos, são mostradas a seguir para maior clareza:

```
with v as (
  select 'ClassSummary' strings from dual union
  select '3453430278'      from dual union
  select 'findRow 55'       from dual union
```

```

select '1010 switch'      from dual union
select '333'              from dual union
select 'threes'            from dual
)
select strings, translated
  from (
select strings,
       translate(
         strings,
         'abcdefghijklmnopqrstuvwxyz0123456789',
         rpad('#',26,'#')||rpad('*',10,'*')) translated
  from v
)
where instr(translated,'#') > 0
  and instr(translated,'*') > 0

STRINGS      TRANSLATED
-----
1010 switch **** ######
findRow 55    #####R## **

```

## 14.7 Convertendo números inteiros em binários usando Oracle

### Problema

Você deseja converter um número inteiro em sua representação binária em um sistema Oracle. Por exemplo, você gostaria de retornar todos os salários na tabela EMP em binário como parte do seguinte conjunto de resultados:

ENAME	SAL	SAL_BINARY
SMITH	800	1100100000
ALLEN	1600	11001000000
WARD	1250	10011100010
JONES	2975	101110011111
MARTIN	1250	10011100010
BLAKE	2850	101100100010
CLARK	2450	100110010010
SCOTT	3000	101110111000
KING	5000	1001110001000
TURNER	1500	10111011100
ADAMS	1100	10001001100
JAMES	950	1110110110
FORD	3000	101110111000
MILLER	1300	10100010100

### Solução

Devido à capacidade do MODEL de iterar e fornecer acesso de array aos valores de linha, é uma escolha natural para esta operação

(supondo que você seja forçado a resolver o problema em SQL, pois uma função armazenada é mais apropriada aqui). Assim como o restante das soluções deste livro, mesmo que você não encontre uma aplicação prática para esse código, concentre-se na técnica. É útil saber que a cláusula MODEL pode executar tarefas procedimentais enquanto ainda mantém a natureza e o poder baseados em conjunto do SQL. Portanto, mesmo que você diga: “Eu nunca faria isso em SQL”, tudo bem. Não estamos de forma alguma sugerindo que você deva ou não. Lembramos que você deve focar na técnica, para que possa aplicá-la no que considerar uma aplicação mais “prática”.

A solução a seguir retorna todos os ENAME e SAL da tabela EMP, enquanto chama a cláusula MODEL em uma subconsulta escalar (desta forma, ela serve como uma espécie de função autônoma da tabela EMP que simplesmente recebe uma entrada, a processa e retorna um valor, muito mais como uma função faria):

```

1 select ename,
2      sal,
3      (
4          select bin
5              from dual
6          model
7              dimension by ( 0 attr )
8              measures ( sal num,
9                  cast(null as varchar2(30)) bin,
10                 '0123456789ABCDEF' hex
11                 )
12             rules iterate (10000) until (num[0] <= 0) (
13                 bin[0] = substr(hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
14                 num[0] = trunc(num[cv()]/2)
15             )
16         ) sal_binary
17     from emp

```

## Discussão

Mencionamos na seção “Solução” que esse problema provavelmente é melhor resolvido por meio de uma função armazenada. Na verdade, a ideia desta receita surgiu de uma função. Na verdade, esta receita é uma adaptação de uma função chamada TO\_BASE, escrita por Tom Kyte da Oracle Corporation. Como outras receitas neste livro que você pode decidir não usar, mesmo que não use esta receita, ela faz um bom trabalho ao mostrar alguns dos recursos da cláusula MODEL, como iteração e acesso ao array de linhas.

Para facilitar a explicação, focamos em uma ligeira variação da subconsulta que contém a cláusula MODEL. O código a seguir é essencialmente a subconsulta da solução, exceto que foi programado para retornar o valor 2 em binário:

```

select bin
  from dual
model
dimension by ( 0 attr )
measures ( 2 num,

```

```

        cast(null as varchar2(30)) bin,
        '0123456789ABCDEF' hex
    )
rules iterate (10000) until (num[0] <= 0) (
    bin[0] = substr (hex[cv()],mod(num[cv()]),2)+1,1)||bin[cv()],
    num[0] = trunc(num[cv()]/2)
)

```

BIN  
-----  
10

A consulta a seguir gera os valores retornados de uma iteração das RULES definidas na consulta anterior:

```

select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
  from dual

```

START_VAL	HEX	BIN	NUM
2	0123456789ABCDEF	0	1

START\_VAL representa o número que você deseja converter em binário, que neste caso é 2. O valor de BIN é o resultado de uma operação de substring em *0123456789ABCDEF* (HEX, na solução original). O valor de NUM é o teste que determinará quando você sair do loop.

Como você pode ver no conjunto de resultados anterior, na primeira vez que o loop passa, BIN é 0 e NUM é 1. Como NUM não é menor ou igual a 0, ocorre outra iteração do loop. A seguinte instrução SQL mostra os resultados da próxima iteração:

```

select num start_val,
       substr('0123456789ABCDEF',mod(1,2)+1,1) ||
       bin bin,
       trunc(1/2) num
  from (
select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
  from dual
)

```

START_VAL	BIN	NUM
1	10	0

Na próxima vez que passar pelo loop, o resultado da operação de substring em HEX retornará 1 e o valor anterior de BIN, 0, será anexado a ele. O teste, NUM, agora é 0; portanto, esta é a última iteração e o valor de retorno “10” é a representação binária do número 2. Quando estiver confortável com o que está acontecendo, você pode remover a iteração da cláusula MODEL e percorre-la linha por linha para acompanhar como as regras são aplicadas para chegar ao conjunto de resultados final, conforme mostrado aqui:

```

select 2 orig_val, num, bin
  from dual
model
dimension by ( 0 attr )
measures ( 2 num,
           cast(null as varchar2(30)) bin,
           '0123456789ABCDEF' hex
         )
rules (
  bin[0] = substr (hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
  num[0] = trunc(num[cv()]/2),
  bin[1] = substr (hex[0],mod(num[0],2)+1,1)||bin[0],
  num[1] = trunc(num[0]/2)
)
-----
```

ORIG_VAL	NUM	BIN
2	1 0	
2	0 10	

## 14.8 Dinamizando um conjunto de resultados classificados

### Problema

Você deseja classificar os valores em uma tabela e, em seguida, dinamizar o conjunto de resultados em três colunas. A ideia é mostrar os três primeiros, os três seguintes e depois todo o resto. Por exemplo, você deseja classificar os funcionários na tabela EMP por SAL e depois dinamizar os resultados em três colunas. O conjunto de resultados desejado é o seguinte:

TOP_3	NEXT_3	REST
KING (5000)	BLAKE (2850)	TURNER (1500)
FORD (3000)	CLARK (2450)	MILLER (1300)
SCOTT (3000)	ALLEN (1600)	MARTIN (1250)
JONES (2975)		WARD (1250)
		ADAMS (1100)
		JAMES (950)
		SMITH (800)

A chave para esta solução é primeiro usar a função de janela DENSE\_RANK OVER para classificar os funcionários por SAL, permitindo ao mesmo tempo empates. Ao usar DENSE\_RANK OVER, você pode ver facilmente os três principais salários, os próximos três salários e todo o resto.

Em seguida, use a função de janela ROW\_NUMBER OVER para classificar cada funcionário em seu grupo (os três primeiros, os próximos três ou o último grupo). A partir daí, basta realizar uma transposição clássica, enquanto usa as funções de string integradas disponíveis em sua plataforma para embelezar os resultados. A solução a seguir usa sintaxe Oracle. Como todos os fornecedores agora oferecem suporte a funções de janela, converter a solução para funcionar em outras plataformas é trivial:

```
1 select max(case grp when 1 then rpad(ename,6) ||
2           ' ('|| sal ||')' end) top_3,
3       max(case grp when 2 then rpad(ename,6) ||
4           ' ('|| sal ||')' end) next_3,
5       max(case grp when 3 then rpad(ename,6) ||
6           ' ('|| sal ||')' end) rest
7   from (
8 select ename,
9       sal,
10      rnk,
11      case when rnk <= 3 then 1
12        when rnk <= 6 then 2
13        else                 3
14    end grp,
15    row_number()over(
16        partition by case when rnk <= 3 then 1
17                      when rnk <= 6 then 2
18                      else                 3
19        end
20        order by sal desc, ename
21    ) grp_rnk
22  from (
23 select ename,
24       sal,
25       dense_rank()over(order by sal desc) rnk
26  from emp
27    ) x
28    ) y
29 group by grp_rnk
```

## Discussão

Esta receita é um exemplo perfeito de quanto você pode realizar com tão pouco, com a ajuda das funções de janela. A solução pode parecer complicada, mas ao analisá-la de dentro para fora, você ficará surpreso com o quão simples ela é. Vamos começar executando primeiro a visualização in-line X:

```
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
  from emp
```

ENAME	SAL	RNK
KING	5000	1
SCOTT	3000	2
FORD	3000	2
JONES	2975	3
BLAKE	2850	4
CLARK	2450	5
ALLEN	1600	6
TURNER	1500	7
MILLER	1300	8
WARD	1250	9
MARTIN	1250	9
ADAMS	1100	10
JAMES	950	11
SMITH	800	12

Como você pode ver no conjunto de resultados anterior, a visualização in-line X simplesmente classifica os funcionários por SAL, ao mesmo tempo que permite empates (como a solução usa DENSE\_RANK em vez de RANK, há vínculos sem lacunas). A próxima etapa é pegar as linhas da visualização in-line X e criar grupos usando uma expressão CASE para avaliar a classificação de DENSE\_RANK. Além disso, utilize a função de janela ROW\_NUMBER OVER para classificar os funcionários por SAL dentro de seu grupo (dentro do grupo que você está criando com a expressão CASE). Tudo isso acontece na visualização inline Y e é mostrado aqui:

```
select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
             when rnk <= 6 then 2
             else                  3
        end grp,
       row_number()over (
           partition by case when rnk <= 3 then 1
                             when rnk <= 6 then 2
                             else                  3
                        end
           order by sal desc, ename
        ) grp_rnk
  from (
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
  from emp
        ) x
```

ENAME	SAL	RNK	GRP	GRP_RNK
KING	5000	1	1	1
FORD	3000	2	1	2
SCOTT	3000	2	1	3
JONES	2975	3	1	4
BLAKE	2850	4	2	1
CLARK	2450	5	2	2
ALLEN	1600	6	2	3
TURNER	1500	7	3	1
MILLER	1300	8	3	2
MARTIN	1250	9	3	3
WARD	1250	9	3	4
ADAMS	1100	10	3	5
JAMES	950	11	3	6
SMITH	800	12	3	7

Agora a consulta está começando a tomar forma, e se você acompanhou desde o início (a partir da visualização inline X), verá que não é tão complicado. A consulta até agora retorna cada funcionário; seu SAL; seu RNK, que representa a classificação do seu SAL entre todos os funcionários; seu GRP, que indica o grupo em que cada funcionário está inserido (com base no SAL); e por fim GRP\_RANK, que é um ranking (baseado no SAL) dentro do seu GRP.

Neste ponto, execute um pivô tradicional no ENAME enquanto usa o operador de concatenação Oracle || para anexar o SAL. A função RPAD garante que os valores numéricos entre parênteses estejam bem alinhados. Por fim, use GROUP BY em GRP\_RANK para garantir que você mostre cada funcionário no conjunto de resultados. O conjunto de resultados final é mostrado aqui:

```

select max(case grp when 1 then rpad(ename,6) ||
           ' ('|| sal ||')' end) top_3,
       max(case grp when 2 then rpad(ename,6) ||
           ' ('|| sal ||')' end) next_3,
       max(case grp when 3 then rpad(ename,6) ||
           ' ('|| sal ||')' end) rest
  from (
select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
             when rnk <= 6 then 2
             else                      3
        end grp,
       row_number()over (
          partition by case when rnk <= 3 then 1
                            when rnk <= 6 then 2
                            else                      3
                       end
          Order by sal desc, ename
       ) grp_rnk
  from (

```

```

select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
  from emp
   ) x
   ) y
group by grp_rnk

TOP_3          NEXT_3          REST
-----
KING    (5000)  BLAKE   (2850)  TURNER (1500)
FORD    (3000)  CLARK    (2450)  MILLER (1300)
SCOTT   (3000)  ALLEN    (1600)  MARTIN (1250)
JONES   (2975)           WARD    (1250)
ADAMS   (1100)           JAMES    (950)
                           SMITH    (800)

```

Se você examinar as consultas em todas as etapas, notará que a tabela EMP é acessada exatamente uma vez. Uma das coisas notáveis sobre as funções de janela é a quantidade de trabalho que você pode realizar em apenas uma passagem pelos seus dados. Não há necessidade de autojunções ou tabelas temporárias; basta obter as linhas necessárias e deixar que as funções da janela façam o resto. Somente na visualização inline X você precisa acessar o EMP. A partir daí, é simplesmente uma questão de massgear o conjunto de resultados para ficar do jeito que você deseja. Considere o que tudo isso significa para o desempenho se você puder criar esse tipo de relatório com um único acesso à tabela. Muito legal.

## 14.9 Adicionando um cabeçalho de coluna em um conjunto de resultados com dinamização dupla

### Problema

Você deseja empilhar dois conjuntos de resultados e depois dinamizá-los em duas colunas. Além disso, você deseja adicionar um “cabeçalho” para cada grupo de linhas em cada coluna. Por exemplo, você tem duas tabelas contendo informações sobre funcionários que trabalham em diferentes áreas de desenvolvimento da sua empresa (digamos, em pesquisas e aplicações):

```
select * from it_research
```

```

DEPTNO ENAME
-----
100 HOPKINS
100 JONES
100 TONEY
200 MORALES
200 P.WHITAKER
200 MARCIANO
200 ROBINSON

```

300 LACY  
300 WRIGHT  
300 J.TAYLOR

```
select * from it_apps
```

```
DEPTNO ENAME
```

```
-----  
400 CORRALES  
400 MAYWEATHER  
400 CASTILLO  
400 MARQUEZ  
400 MOSLEY  
500 GATTI  
500  
CALZAGHE  
600 LAMOTTA  
600 HAGLER  
600 HEARNS  
600 FRAZIER  
700 GUINN  
700 JUDAH  
700 MARGARITO
```

Você gostaria de criar um relatório listando os funcionários de cada tabela em duas colunas. Você deseja retornar o DEPTNO seguido de ENAME para cada um. Por fim, você deseja retornar o seguinte conjunto de resultados:

RESEARCH	APPS
100	400
JONES	MAYWEATHER
TONEY	CASTILLO
HOPKINS	MARQUEZ
200	MOSLEY
P.WHITAKER	CORRALES
MARCIANO	500
ROBINSON	CALZAGHE
MORALES	GATTI
300	600
WRIGHT	HAGLER
J.TAYLOR	HEARNS
LACY	FRAZIER
	LAMOTTA
700	
	JUDAH
	MARGARITO
	GUINN

## Solução

Na maior parte, esta solução não requer nada mais do que um simples stack 'n' pivot (união e depois pivô) com uma peculiaridade adicional: o DEPTNO deve preceder o ENAME para cada funcionário retornado. A técnica aqui usa um produto cartesiano para gerar uma linha extra para cada DEPTNO, para que você tenha as linhas necessárias para mostrar todos os funcionários, além de espaço para o DEPTNO. A solução usa a sintaxe Oracle, mas como o DB2 suporta funções de janela que podem calcular janelas móveis (a cláusula de enquadramento), converter essa solução para funcionar no DB2 é trivial. Como as tabelas IT\_RESEARCH e IT\_APPS existem apenas para esta receita, suas instruções de criação de tabela são mostradas junto com esta solução:

```

create table IT_research (deptno number, ename varchar2(20))

insert into IT_research values (100,'HOPKINS')
insert into IT_research values (100,'JONES')
insert into IT_research values (100,'TONY')
insert into IT_research values (200,'MORALES')
insert into IT_research values (200,'P.WHITAKER')
insert into IT_research values (200,'MARCIANO')
insert into IT_research values (200,'ROBINSON')
insert into IT_research values (300,'LACY')
insert into IT_research values (300,'WRIGHT')
insert into IT_research values (300,'J.TAYLOR')

create table IT_apps (deptno number, ename varchar2(20))

insert into IT_apps values (400,'CORRALES')
insert into IT_apps values (400,'MAYWEATHER')
insert into IT_apps values (400,'CASTILLO')
insert into IT_apps values (400,'MARQUEZ')
insert into IT_apps values (400,'MOSLEY')
insert into IT_apps values (500,'GATTI')
insert into IT_apps values (500,'CALZAGHE')
insert into IT_apps values (600,'LAMOTTA')
insert into IT_apps values (600,'HAGLER')
insert into IT_apps values (600,'HEARNS')
insert into IT_apps values (600,'FRAZIER')
insert into IT_apps values (700,'GUINN')
insert into IT_apps values (700,'JUDAH')
insert into IT_apps values (700,'MARGARITO')

1 select max(decode(flag2,0,it_dept)) research,
2       max(decode(flag2,1,it_dept)) apps
3   from (
4 select sum(flag1)over(partition by flag2
5                           order by flag1, rownum) flag,
6       it_dept, flag2
7   from (

```

```
8 select 1 flag1, 0 flag2,
9      decode(rn,1,to_char(deptno),''||ename) it_dept
10     from (
11 select x.* , y.id,
12       row_number()over(partition by x.deptno order by y.id) rn
13   from (
14 select deptno,
15       ename,
16       count(*)over(partition by deptno) cnt
17   from it_research
18       ) x,
19       (select level id from dual connect by level <= 2) y
20
21 where rn <= cnt+1
22 union all
23 select 1 flag1, 1 flag2,
24      decode(rn,1,to_char(deptno),''||ename) it_dept
25     from (
26 select x.* , y.id,
27       row_number()over(partition by x.deptno order by y.id) rn
28   from (
29 select deptno,
30       ename,
31       count(*)over(partition by deptno) cnt
32   from it_apps
33       ) x,
34       (select level id from dual connect by level <= 2) y
35
36 where rn <= cnt+1
37       ) tmp1
38       ) tmp2
39 group by flag
```

## Discussão

Como muitas outras consultas do tipo armazenamento/relatório, a solução apresentada parece bastante complicada, mas uma vez detalhada, você verá que nada mais é do que uma pilha e um pivô com um toque cartesiano (com gelo, com um pequeno guarda-chuva). A maneira de decompor essa consulta é trabalhar primeiro em cada parte do UNION ALL e depois reuni-las para o pivô. Vamos começar com a parte inferior do UNION ALL:

```
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),''||ename) it_dept
     from (
select x.* , y.id,
       row_number()over(partition by x.deptno order by y.id) rn
     from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
   from it_apps
```

```

) x,
(select level id from dual connect by level <= 2) y
) z
where rn <= cnt+1

FLAG1      FLAG2 IT_DEPT
-----
- 1        1 400
1          1 MAYWEATHER
1          1 CASTILLO
1          1 MARQUEZ
1          1 MOSLEY
1          1 CORRALES
1          1 500
1          1 CALZAGHE
1          1 GATTI
1          1 600
1          1 HAGLER
1          1 HEARNS
1          1 FRAZIER
1          1 LAMOTTA
1          1 700
1          1 JUDAH
1          1 MARGARITO
1          1 GUINN

```

Vamos examinar exatamente como esse conjunto de resultados é montado. Dividindo a consulta anterior em seus componentes mais simples, você tem a visualização in-line X, que simplesmente retorna cada ENAME e DEPTNO e o número de funcionários em cada DEPTNO da tabela IT\_APPS. Os resultados são os seguintes:

```

select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps

```

DEPTNO	ENAME	CNT
400	CORRALES	5
400	MAYWEATHER	5
400	CASTILLO	5
400	MARQUEZ	5
400	MOSLEY	5
500	GATTI	2
500	CALZAGHE	2
600	LAMOTTA	4
600	HAGLER	4
600	HEARNS	4
600	FRAZIER	4
700	GUINN	3
700	JUDAH	3
700	MARGARITO	3

A próxima etapa é criar um produto cartesiano entre as linhas retornadas da visualização inline X e duas linhas geradas a partir de DUAL usando CONNECT BY. Os resultados desta operação são os seguintes:

```
select *
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
   ) x,
       (select level id from dual connect by level <= 2) y
order by 2
```

DEPTNO	ENAME	CNT	ID
500	CALZAGHE	2	1
500	CALZAGHE	2	2
400	CASTILLO	5	1
400	CASTILLO	5	2
400	CORRALES	5	1
400	CORRALES	5	2
600	FRAZIER	4	1
600	FRAZIER	4	2
500	GATTI	2	1
500	GATTI	2	2
700	GUINN	3	1
700	GUINN	3	2
600	HAGLER	4	1
600	HAGLER	4	2
600	HEARNS	4	1
600	HEARNS	4	2
700	JUDAH	3	1
700	JUDAH	3	2
600	LAMOTTA	4	1
600	LAMOTTA	4	2
700	MARGARITO	3	1
700	MARGARITO	3	2
400	MARQUEZ	5	1
400	MARQUEZ	5	2
400	MAYWEATHER	5	1
400	MAYWEATHER	5	2
400	MOSLEY	5	1
400	MOSLEY	5	2

Como você pode ver nesses resultados, cada linha da visualização inline X agora é retornada duas vezes devido ao produto cartesiano com a visualização inline Y. O motivo pelo qual um cartesiano é necessário ficará claro em breve. A próxima etapa é pegar o conjunto de resultados atual e classificar cada funcionário dentro de seu DEPTNO por ID (ID tem valor 1 ou 2 conforme retornado pelo produto cartesiano). O resultado dessa classificação é mostrado na saída da seguinte consulta:

```

select x.* , y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
      ) x,
       (select level id from dual connect by level <= 2) y

```

DEPTNO	ENAME	CNT	ID	RN
400	CORRALES	5	1	1
400	MAYWEATHER	5	1	2
400	CASTILLO	5	1	3
400	MARQUEZ	5	1	4
400	MOSLEY	5	1	5
400	CORRALES	5	2	6
400	MOSLEY	5	2	7
400	MAYWEATHER	5	2	8
400	CASTILLO	5	2	9
400	MARQUEZ	5	2	10
500	GATTI	2	1	1
500	CALZAGHE	2	1	2
500	GATTI	2	2	3
500	CALZAGHE	2	2	4
600	LAMOTTA	4	1	1
600	HAGLER	4	1	2
600	HEARN	4	1	3
600	FRAZIER	4	1	4
600	LAMOTTA	4	2	5
600	HAGLER	4	2	6
600	FRAZIER	4	2	7
600	HEARN	4	2	8
700	GUINN	3	1	1
700	JUDAH	3	1	2
700	MARGARITO	3	1	3
700	GUINN	3	2	4
700	JUDAH	3	2	5
700	MARGARITO	3	2	6

Cada funcionário é classificado; então sua duplicata é classificada. O conjunto de resultados contém duplicatas para todos os funcionários na tabela IT\_APP, juntamente com sua classificação dentro do DEPTNO. A razão pela qual você precisa gerar essas linhas extras é porque você precisa de um slot no conjunto de resultados para inserir o DEPTNO na coluna ENAME. Se você juntar IT\_APPS cartesianamente com uma tabela de uma linha, não obterá linhas extras (porque cardinalidade de qualquer tabela  $\times 1$  = cardinalidade dessa tabela).

A próxima etapa é pegar os resultados retornados até agora e dinamizar o conjunto de resultados de forma que todos os ENAMES sejam retornados em uma coluna, mas sejam precedidos pelo DEPTNO em que estão. A consulta a seguir mostra como isso acontece:

```

select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
      ) x,
      (select level id from dual connect by level <= 2) y
      ) z
 where rn <= cnt+1

```

FLAG1	FLAG2	IT_DEPT
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO
1	1	MARQUEZ
1	1	MOSLEY
1	1	CORRALES
1	1	500
1	1	CALZAGHE
1	1	GATTI
1	1	600
1	1	HAGLER
1	1	HEARN
1	1	FRAZIER
1	1	LAMOTTA
1	1	700
1	1	JUDAH
1	1	MARGARITO
1	1	GUINN

FLAG1 e FLAG2 entram em jogo mais tarde e podem ser ignorados por enquanto. Concentre sua atenção nas linhas em IT\_DEPT. O número de linhas retornadas para cada DEPTNO é CNT\*2, mas tudo o que é necessário é CNT+1, que é o filtro na cláusula WHERE. RN é a classificação de cada funcionário. As linhas mantidas são todas aquelas com classificação menor ou igual a CNT+1; ou seja, todos os funcionários de cada DEPTNO mais um (esse funcionário extra é o funcionário que ocupa o primeiro lugar no seu DEPTNO). Esta linha extra é onde o DEPTNO entrará. Usando DECODE (uma função Oracle mais antiga que fornece mais ou menos o equivalente a uma expressão CASE) para avaliar o valor de RN, você pode inserir o valor de DEPTNO no conjunto de resultados. O funcionário que estava na posição um (com base no valor de RN) ainda é mostrado no conjunto de resultados, mas agora é o último em cada DEPTNO (como a ordem é irrelevante, isso não é problema). Isso cobre praticamente a parte inferior do UNION ALL.

A parte superior do UNION ALL é processada da mesma forma que a parte inferior, portanto não há necessidade de explicar como funciona. Em vez disso, vamos examinar o conjunto de resultados retornado ao empilhar as consultas:

```

select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),''||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
      ) x,
       (select level id from dual connect by level <= 2) y
      )
 where rn <= cnt+1
union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),''||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
      ) x,
       (select level id from dual connect by level <= 2) y
      )
 where rn <= cnt+1

```

FLAG1	FLAG2	IT_DEPT
1	0	100
1	0	JONES
1	0	TONEY
1	0	HOPKINS
1	0	200
1	0	P.WHITAKER
1	0	MARCIANO
1	0	ROBINSON
1	0	MORALES
1	0	300
1	0	WRIGHT
1	0	J.TAYLOR
1	0	LACY
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO

```
1      1  MARQUEZ
1      1  MOSLEY
1      1  CORRALES
1      1  500
1      1  CALZAGHE
1      1  GATTI
1      1  600
1      1  HAGLER
1      1  HEARNS
1      1  FRAZIER
1      1  LAMOTTA
1      1  700
1      1  JUDAH
1      1  MARGARITO
1      1  GUINN
```

Neste ponto, não está claro qual é o propósito do FLAG1, mas você pode ver que o FLAG2 identifica quais linhas vêm de qual parte do UNION ALL (0 para a parte superior, 1 para a parte inferior).

A próxima etapa é agrupar o conjunto de resultados empilhados em uma visualização inline e criar um total acumulado em FLAG1 (finalmente, seu propósito é revelado!), que atuará como uma classificação para cada linha em cada pilha. Os resultados da classificação (total acumulado) são mostrados aqui:

```
select sum(flag1)over(partition by flag2
                      order by flag1, rownum) flag,
       it_dept, flag2
  from (
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),''||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
     ) x,
     (select level id from dual connect by level <= 2) y
    )
   where rn <= cnt+1
union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),''||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
```

```

from it_apps
  ) x,
  (select level id from dual connect by level <= 2) y
  )
where rn <= cnt+1
  ) tmp1

```

FLAG	IT_DEPT	FLAG2
1	100	0
2	JONES	0
3	TONEY	0
4	HOPKINS	0
5	200	0
6	P.WHITAKER	0
7	MARCIANO	0
8	ROBINSON	0
9	MORALES	0
10	300	0
11	WRIGHT	0
12	J.TAYLOR	0
13	LACY	0
1	400	1
2	MAYWEATHER	1
3	CASTILLO	1
4	MARQUEZ	1
5	MOSLEY	1
6	CORRALES	1
7	500	1
8	CALZAGHEe	1
9	GATTI	1
10	600	1
11	HAGLER	1
12	HEARNs	1
13	FRAZIER	1
14	LAMOTTA	1
15	700	1
16	JUDAH	1
17	MARGARITO	1
18	GUINN	1

A última etapa (finalmente!) É dinamizar o valor retornado por TMP1 em FLAG2 enquanto agrupa por FLAG (o total acumulado gerado em TMP1). Os resultados do TMP1 são agrupados em uma visualização inline e dinamizados (envolvidos em uma visualização inline final chamada TMP2). A solução final e o conjunto de resultados são mostrados aqui:

```

select max(decode(flag2,0,it_dept)) research,
       max(decode(flag2,1,it_dept)) apps
  from (
select sum(flag1)over(partition by flag2
                     order by flag1, rownum) flag,

```

```

    it_dept, flag2
  from (
  select 1 flag1, 0 flag2,
         decode(rn,1,to_char(deptno),''||ename) it_dept
  from (
  select x.*, y.id,
         row_number()over(partition by x.deptno order by y.id) rn
  from (
  select deptno,
         ename,
         count(*)over(partition by deptno) cnt
  from it_research
      ) x,
      (select level id from dual connect by level <= 2) y
  )
  where rn <= cnt+1
union all
  select 1 flag1, 1 flag2,
         decode(rn,1,to_char(deptno),''||ename) it_dept
  from (
  select x.*, y.id,
         row_number()over(partition by x.deptno order by y.id) rn
  from (
  select deptno deptno,
         ename,
         count(*)over(partition by deptno) cnt
  from it_apps
      ) x,
      (select level id from dual connect by level <= 2) y
  )
  where rn <= cnt+1
      ) tmp1
      ) tmp2
group by flag

```

RESEARCH	APPS
100	400
JONES	MAYWEATHER
TONEY	CASTILLO
HOPKINS	MARQUEZ
200	MOSLEY
P.WHITAKER	CORRALES
MARCIANO	500
ROBINSON	CALZAGHE
MORALES	GATTI
300	600
WRIGHT	HAGLER
J.TAYLOR	HEARN
LACY	FRAZIER
	LAMOTTA

```

700
JUDAH
MARGARITO
GUINN

```

## 14.10 Convertendo uma subconsulta escalar em uma subconsulta composta no Oracle

### Problema

Você deseja ignorar a restrição de retornar exatamente um valor de uma subconsulta escalar. Por exemplo, você tenta executar a seguinte consulta:

```

select e.deptno,
       e.ename,
       e.sal,
       (select d.dname,d.loc,sysdate today
        from dept d
        where e.deptno=d.deptno)
  from emp e

```

mas recebe um erro porque as subconsultas na lista SELECT podem retornar apenas um único valor.

### Solução

É certo que este problema é bastante irrealista, porque uma simples junção entre as tabelas EMP e DEPT permitiria retornar quantos valores desejar do DEPT. No entanto, o segredo é focar na técnica e entender como aplicá-la a um cenário que você considere útil. A chave para contornar o requisito de retornar um único valor ao colocar um SELECT dentro de SELECT (subconsulta escalar) é aproveitar as vantagens dos tipos de objetos do Oracle. Você pode definir um objeto para ter vários atributos e, em seguida, trabalhar com ele como uma única entidade ou fazer referência a cada elemento individualmente. Na verdade, você realmente não ignora a regra. Você simplesmente retorna um valor — um objeto [.keep-together]# — # que por sua vez contém muitos atributos.

Esta solução utiliza o seguinte tipo de objeto:

```

create type generic_obj
  as object (
    val1 varchar2(10),
    val2 varchar2(10),
    val3 date
  );

```

Com esse tipo instalado, você pode executar a seguinte consulta:

```

1 select x.deptno,
2        x.ename,

```

```

1      x.multival.val1 dname,
2      x.multival.val2 loc,
3      x.multival.val3 today
4  from (
7select e.deptno,
8      e.ename,
9      e.sal,
10     (select generic_obj(d.dname,d.loc,sysdate+1)
11       from dept d
12      where e.deptno=d.deptno) multival
13  from emp e
14      ) x

```

DEPTNO	ENAME	DNAME	LOC	TODAY
20	SMITH	RESEARCH	DALLAS	12-SEP-2020
30	ALLEN	SALES	CHICAGO	12-SEP-2020
30	WARD	SALES	CHICAGO	12-SEP-2020
20	JONES	RESEARCH	DALLAS	12-SEP-2020
30	MARTIN	SALES	CHICAGO	12-SEP-2020
30	BLAKE	SALES	CHICAGO	12-SEP-2020
10	CLARK	ACCOUNTING	NEW YORK	12-SEP-2020
20	SCOTT	RESEARCH	DALLAS	12-SEP-2020
10	KING	ACCOUNTING	NEW YORK	12-SEP-2020
30	TURNER	SALES	CHICAGO	12-SEP-2020
20	ADAMS	RESEARCH	DALLAS	12-SEP-2020
30	JAMES	SALES	CHICAGO	12-SEP-2020
20	FORD	RESEARCH	DALLAS	12-SEP-2020
10	MILLER	ACCOUNTING	NEW YORK	12-SEP-2020

## Discussão

A chave para a solução é usar a função construtora do objeto (por padrão, a função construtora tem o mesmo nome do objeto). Como o objeto em si é um valor escalar único, ele não viola a regra de subconsulta escalar, como você pode ver a seguir:

```

select e.deptno,
      e.ename,
      e.sal,
      (select generic_obj(d.dname,d.loc,sysdate-1)
        from dept d
       where e.deptno=d.deptno) multival
  from emp e

```

DEPTNO	ENAME	SAL	MULTIVAL(VAL1, VAL2, VAL3)
20	SMITH	800	GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
30	ALLEN	1600	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
30	WARD	1250	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
20	JONES	2975	GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
30	MARTIN	1250	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')

```

30 BLAKE  2850  GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
10 CLARK   2450  GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2020')
20 SCOTT   3000  GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
10 KING    5000  GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2020')
30 TURNER  1500  GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
20 ADAMS   1100  GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
30 JAMES   950   GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
20 FORD    3000  GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
10 MILLER  1300  GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2020')

```

A próxima etapa é simplesmente agrupar a consulta em uma visualização embutida e extrair os atributos.



Na Oracle, diferentemente de outros fornecedores, geralmente você não precisa nomear suas visualizações in-line. Neste caso específico, entretanto, você precisa nomear sua visualização embutida. Caso contrário, você não conseguirá fazer referência aos atributos do objeto.

## 14.11 Analisando dados serializados em linhas

### Problema

Você serializou dados (armazenados em strings) que deseja analisar e retornar como linhas. Por exemplo, você armazena os seguintes dados:

```

STRINGS
-----
entry:stewiegriffin:lois:brian:
entry:moe::sizlack:
entry:petergriffin:meg:chris:
entry:willie:
entry:quagmire:mayorwest:cleveland:
entry:::flanders:
entry:robo:tchi:ken:

```

Você deseja converter essas strings serializadas no seguinte conjunto de resultados:

VAL1	VAL2	VAL3
moe		sizlack
petergriffin	meg	chris
quagmire	mayorwest	cleveland
robo	tchi	ken
stewiegriffin	lois	brian
willie		flanders

Cada string serializada neste exemplo pode armazenar até três valores. Os valores são delimitados por dois pontos e uma string pode ou não ter todas as três entradas. Se uma sequência não tiver todas as três entradas, você deverá ter cuidado ao colocar as entradas disponíveis na coluna correta do conjunto de resultados. Por exemplo, considere a seguinte linha:

```
entry:::flanders:
```

Esta linha representa uma entrada com os dois primeiros valores ausentes e apenas o terceiro valor disponível. Portanto, se você examinar o conjunto de resultados de destino na seção “Problema”, notará que, para a linha em que FLANDERS está, VAL1 e VAL2 são NULL.

A chave para esta solução nada mais é do que percorrer uma string com alguma análise de string, seguida por um simples pivô. Esta solução usa linhas da visão V, que é definida a seguir. O exemplo usa sintaxe Oracle, mas como nada além de funções de análise de string são necessárias para esta receita, a conversão para outras plataformas é simples:

```
create view V
  as
select 'entry:stewiegriffin:lois:brian:' strings
  from dual
union all
select 'entry:moe::sizlack:'
  from dual
union all
select 'entry:petergriffin:meg:chris:'
  from dual
union all
select 'entry:willie:'
  from dual
union all
select 'entry:quagmire:mayorwest:cleveland:'
  from dual
union all
select 'entry:::flanders:'
  from dual
union all
select 'entry:robo:tchi:ken:'
  from dual
```

Usando a visualização V para fornecer os dados de exemplo a serem analisados, a solução é a seguinte:

```
1 with cartesian as (
2 select level id
3   from dual
4  connect by level <= 100
5 )
6 select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
7        max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
8        max(decode(id,3,substr(strings,p1+1,p2-1))) val3
```

```

8   from (
9 select v.strings,
10      c.id,
11      instr(v.strings,':',1,c.id) p1,
12      instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
13  from v, cartesian c
14 where c.id <= (length(v.strings)-length(replace(v.strings,:')))-1
15 )
16
17 group by strings
18 order by 1

```

## Discussão

O primeiro passo é percorrer as strings serializadas:

```

with cartesian as (
  select level id
    from dual
   connect by level <= 100
)
select v.strings,
       c.id
  from v, cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,:')))-1

```

STRINGS	ID
entry:::flanders:	1
entry:::flanders:	2
entry:::flanders:	3
entry:moe:::sizlack:	1
entry:moe:::sizlack:	2
entry:moe:::sizlack:	3
entry:petergriffin:meg:chris:	1
entry:petergriffin:meg:chris:	3
entry:petergriffin:meg:chris:	2
entry:quagmire:mayorwest:cleveland:	1
entry:quagmire:mayorwest:cleveland:	3
entry:quagmire:mayorwest:cleveland:	2
entry:robo:tchi:ken:	1
entry:robo:tchi:ken:	2
entry:robo:tchi:ken:	3
entry:stewiegriffin:lois:brian:	1
entry:stewiegriffin:lois:brian:	3
entry:stewiegriffin:lois:brian:	2
entry:willie:	1

A próxima etapa é usar a função INSTR para encontrar a posição numérica de cada dois pontos em cada string. Como cada valor que você precisa extrair está entre dois pontos, os valores numéricos têm os alias P1 e P2, para “posição um” e “posição dois”:

```

with cartesian as (
  select level id
    from dual
   connect by level <= 100
)
select v.strings,
       c.id,
      instr(v.strings,':',1,c.id) p1,
      instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
     from v,carthesian c
    where c.id <= (length(v.strings)-length(replace(v.strings,:')))-1
      order by 1
  
```

STRINGS	ID	P1	P2
entry:::flanders:	1	6	1
entry:::flanders:	2	7	1
entry:::flanders:	3	8	9
entry:moe::sizlack:	1	6	4
entry:moe::sizlack:	2	10	1
entry:moe::sizlack:	3	11	8
entry:petergriffin:meg:chris:	1	6	13
entry:petergriffin:meg:chris:	3	23	6
entry:petergriffin:meg:chris:	2	19	4
entry:quagmire:mayorwest:cleveland:	1	6	9
entry:quagmire:mayorwest:cleveland:	3	25	10
entry:quagmire:mayorwest:cleveland:	2	15	10
entry:robo:tchi:ken:	1	6	5
entry:robo:tchi:ken:	2	11	5
entry:robo:tchi:ken:	3	16	4
entry:stewiegriffin:lois:brian:	1	6	14
entry:stewiegriffin:lois:brian:	3	25	6
entry:stewiegriffin:lois:brian:	2	20	5
entry:willie:	1	6	7

Agora que você conhece as posições numéricas de cada par de dois pontos em cada string, basta passar as informações para a função SUBSTR para extrair os valores. Como você deseja criar um conjunto de resultados com três colunas, use DECODE para avaliar o ID do produto cartesiano:

```

with cartesian as (
  select level id
    from dual
   connect by level <= 100
)
select decode(id,1,substr(strings,p1+1,p2-1)) val1,
       decode(id,2,substr(strings,p1+1,p2-1)) val2,
       decode(id,3,substr(strings,p1+1,p2-1)) val3
     from (
select v.strings,
       c.id,
      instr(v.strings,':',1,c.id) p1,
  
```

```

instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
from v, cartesian c
where c.id <= (length(v.strings)-length(replace(v.strings,:')))-1
)
order by 1

VAL1          VAL2          VAL3
-----
moe
petergriffin
quagmire
robo
stewiegriffin
willie
                lois

                meg
                mayorwest

                tchi
                                brian
                                sizlack
                                chris
                                cleveland
                                flanders ken

```

A última etapa é aplicar uma função agregada aos valores retornados por SUBSTR durante o agrupamento por ID, para criar um conjunto de resultados legível por humanos:

```

with cartesian as (
  select level id
    from dual
   connect by level <= 100
)
select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
       max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
       max(decode(id,3,substr(strings,p1+1,p2-1))) val3
  from (
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
  from v,cartesian c
where c.id <= (length(v.strings)-length(replace(v.strings,:)))-1
      )
group by strings
order by 1

VAL1          VAL2          VAL3
-----
moe           sizlack
petergriffin    meg        chris

```

quagmire	mayorwest	cleveland
robo	tchi	ken
stewiegriffin	lois	brian
		willie
		flanders

## 14.12 Cálculo da porcentagem em relação ao total

### Problema

Você deseja relatar um conjunto de valores numéricos e mostrar cada valor como uma porcentagem do total. Por exemplo, você está em um sistema Oracle e deseja retornar um conjunto de resultados que mostre o detalhamento dos salários por JOB para poder determinar qual cargo custa mais dinheiro para a empresa. Você também deseja incluir o número de funcionários por JOB para evitar que os resultados sejam enganosos. Você deseja produzir o seguinte relatório:

JOB	NUM_EMPS	PCT_OF_ALL_SALARIES
CLERK	4	14
ANALYST	2	20
MANAGER	3	28
SALESMAN	4	19
PRESIDENT	1	17

Como você pode ver, se o número de funcionários não estiver incluído no relatório, parece que o cargo de presidente representa muito pouco do salário total. Ver que há apenas um presidente ajuda a colocar em perspectiva o que significam esses 17%.

### Solução

Somente a Oracle permite uma solução decente para este problema, que envolve o uso da função integrada RATIO\_TO\_REPORT. Para calcular porcentagens do total para outros bancos de dados, você pode usar a divisão mostrado em [Receita 7.11](#):

```
1 select job,num_emps,sum(round(pct)) pct_of_all_salaries
2   from (
3 select job,
4       count(*)over(partition by job) num_emps,
5       ratio_to_report(sal)over()*100 pct
6   from emp
7      )
8 group by job,num_emps
```

## Discussão

O primeiro passo é usar a função de janela COUNT OVER para retornar o número de funcionários por JOB. Em seguida, use RATIO\_TO\_REPORT para retornar a porcentagem que cada salário conta em relação ao total (o valor é retornado em decimal):

```
select job,
       count(*)over(partition by job) num_emps,
       ratio_to_report(sal)over()*100 pct
  from emp
```

JOB	NUM_EMPS	PCT
ANALYST	2	10.3359173
ANALYST	2	10.3359173
CLERK	4	2.75624462
CLERK	4	3.78983635
CLERK	4	4.4788975
CLERK	4	3.27304048
MANAGER	3	10.2497847
MANAGER	3	8.44099914
MANAGER	3	9.81912145
PRESIDENT	1	17.2265289
SALESMAN	4	5.51248923
SALESMAN	4	4.30663221
SALESMAN	4	5.16795866
SALESMAN	4	4.30663221

A última etapa é usar a função agregada SUM para somar os valores retornados por RATIO\_TO\_REPORT. Certifique-se de agrupar por JOB e NUM\_EMPS. Multiplique por 100 para retornar um número inteiro que represente uma porcentagem (por exemplo, para retornar 25 em vez de 0,25 para 25%):

```
select job,num_emps,sum(round(pct)) pct_of_all_salaries
  from (
select job,
       count(*)over(partition by job) num_emps,
       ratio_to_report(sal)over()*100 pct
  from emp
      )
 group by job,num_emps
```

JOB	NUM_EMPS	PCT_OF_ALL_SALARIES
CLERK	4	14
ANALYST	2	20
MANAGER	3	28
SALESMAN	4	19
PRESIDENT	1	17

## 14.13 Testando a existência de um valor dentro de um grupo

### Problema

Você deseja criar um sinalizador booleano para uma linha dependendo se alguma linha em seu grupo contém um valor específico. Considere o exemplo de um aluno que fez um certo número de exames durante um período de tempo. Um aluno fará três exames em três meses. Se um aluno for aprovado em um desses exames, o requisito será atendido e uma bandeira deverá ser devolvida para expressar esse fato. Caso o aluno não tenha passado em nenhuma das três provas no período de três meses, deverá ser devolvida uma bandeira adicional para expressar esse fato também. Considere o seguinte exemplo (usando a sintaxe Oracle para criar linhas para este exemplo; pequenas modificações são necessárias para os outros fornecedores, tornando o usuário das funções de janela):

```
create view V
as
select 1 student_id,
       1 test_id,
       2 grade_id,
       1 period_id,
       to_date('02/01/2020','MM/DD/YYYY') test_date,
       0 pass_fail
  from dual union all
select 1, 2, 2, 1, to_date('03/01/2020','MM/DD/YYYY'), 1 from dual union all
select 1, 3, 2, 1, to_date('04/01/2020','MM/DD/YYYY'), 0 from dual union all
select 1, 4, 2, 2, to_date('05/01/2020','MM/DD/YYYY'), 0 from dual union all
select 1, 5, 2, 2, to_date('06/01/2020','MM/DD/YYYY'), 0 from dual union all
select 1, 6, 2, 2, to_date('07/01/2020','MM/DD/YYYY'), 0 from dual

select *
  from V

STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE      PASS_FAIL
----- ----- ----- ----- 10-APR-2020      0
          1      1      2      1 01-FEB-2020      0
          1      2      2      1 01-MAR-2020      1
          1      3      2      1 01-APR-2020      0
          1      4      2      2 01-MAY-2020      0
          1      5      2      2 01-JUN-2020      0
          1      6      2      2 01-JUL-2020      0
```

Examinando o conjunto de resultados anterior, você vê que o aluno fez seis testes em períodos de dois e três meses. O aluno passou em um teste (1 significa “aprovado”; 0 significa “reprovado”); assim, o requisito é satisfeito durante todo o primeiro período. Como o aluno não passou em nenhum exame durante o segundo período (os próximos três meses), PASS\_FAIL é 0 para todos os três exames. Você deseja retornar um conjunto de resultados que destaque se um aluno passou em um teste em um determinado período. Em última análise, você deseja retornar o seguinte conjunto de resultados:

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	METREQ	IN_PROGRESS
1	1	2	1	01-FEB-2020	+	0
1	2	2	1	01-MAR-2020	+	0
1	3	2	1	01-APR-2020	+	0
1	4	2	2	01-MAY-2020	-	0
1	5	2	2	01-JUN-2020	-	0
1	6	2	2	01-JUL-2020	-	1

Os valores para METREQ (“requisito atendido”) são + e -, significando que o aluno satisfez ou não satisfez o requisito de aprovação em pelo menos um teste em um período (período de três meses), respectivamente. O valor para IN\_PROGRESS deverá ser 0 se o aluno já tiver passado em uma prova em determinado período. Se um aluno não tiver passado em um teste em um determinado período, a linha que contém a data do último exame desse aluno terá o valor 1 para IN\_PROGRESS.

## Solução

Esse problema parece complicado porque você precisa tratar as linhas de um grupo como um grupo e não como indivíduos. Considere os valores para PASS\_FAIL na seção “Problema”. Se você avaliar linha por linha, parece que o valor de METREQ para cada linha, exceto TEST\_ID 2, deveria ser -, quando não for o caso. Você deve avaliar as linhas como um grupo. Usando a função de janela MAX OVER, você pode determinar facilmente se um aluno passou em pelo menos um teste durante um determinado período. Depois de ter essas informações, os valores “booleanos” são uma simples questão de usar expressões CASE:

```

1 select student_id,
2      test_id,
3      grade_id,
4      period_id,
5      test_date,
6      decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
7      decode( grp_p_f,1,0,
8              decode( test_date,last_test,1,0 ) ) in_progress
9  from (
10 select V.*,
11        max(pass_fail)over(partition by
12                           student_id,grade_id,period_id) grp_p_f,
13        max(test_date)over(partition by
14                           student_id,grade_id,period_id) last_test
15  from V
16      ) x

```

## Discussão

A chave para a solução é usar a função de janela MAX OVER para retornar o maior valor de PASS\_FAIL para cada grupo.

Como os valores para PASS\_FAIL são apenas 1 ou 0, se um aluno for aprovado em pelo menos um exame, MAX OVER retornará 1 para todo o grupo. Como isso funciona é mostrado aqui:

```
select V.*,
       max(pass_fail)over(partition by
                           student_id,grade_id,period_id) grp_pass_fail
  from V

STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE      PASS_FAIL GRP_PASS_FAIL
-----  -----  -----  -----  -----  -----
1        1        2        1 01-FEB-2020          0          1
1        2        2        1 01-MAR-2020          1          1
1        3        2        1 01-APR-2020          0          1
1        4        2        2 01-MAY-2020          0          0
1        5        2        2 01-JUN-2020          0          0
1        6        2        2 01-JUL-2020          0          0
```

O conjunto de resultados anterior mostra que o aluno passou em pelo menos uma prova no primeiro período; portanto, todo o grupo tem o valor 1 ou “aprovado”. O próximo requisito é que caso o aluno não tenha passado em nenhuma prova em um período, retorne o valor 1 para a flag IN\_PROGRESS para a última data da prova naquele grupo. Você também pode usar a função de janela MAX OVER para fazer isso:

```
select V.*,
       max(pass_fail)over(partition by
                           student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
                           student_id,grade_id,period_id) last_test
  from V

STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE      PASS_FAIL GRP_P_F LAST_TEST
-----  -----  -----  -----  -----  -----
1        1        2        1 01-FEB-2020          0          1 01-APR-2020
1        2        2        1 01-MAR-2020          1          1 01-APR-2020
1        3        2        1 01-APR-2020          0          1 01-APR-2020
1        4        2        2 01-MAY-2020          0          0 01-JUL-2020
1        5        2        2 01-JUN-2020          0          0 01-JUL-2020
1        6        2        2 01-JUL-2020          0          0 01-JUL-2020
```

Agora que você determinou em qual período o aluno passou no teste e qual é a última data do teste para cada período, a última etapa é simplesmente uma questão de aplicar um pouco de mágica de formatação para deixar o conjunto de resultados bonito. A solução definitiva usa a função DECODE da Oracle (apoiares do CASE, comam tudo) para criar as colunas METREQ e IN\_PROGRESS. Use a função LPAD para justificar à direita os valores de METREQ:

```
select student_id,
       test_id,
       grade_id,
       period_id,
       test_date,
```

```

decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
decode( grp_p_f,1,0,
        decode( test_date,last_test,1,0 ) ) in_progress
from (
select V.*,
       max(pass_fail)over(partition by
                           student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
                           student_id,grade_id,period_id) last_test
from V
) x

```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	METREQ	IN_PROGRESS
1	1	2	1	01-FEB-2020	+	0
1	2	2	1	01-MAR-2020	+	0
1	3	2	1	01-APR-2020	+	0
1	4	2	2	01-MAY-2020	-	0
1	5	2	2	01-JUN-2020	-	0
1	6	2	2	01-JUL-2020	-	1

## 14.14 Resumindo

SQL é mais poderoso do que muitos acreditam. Ao longo deste livro tentamos desafiá-lo a ver mais aplicações do que normalmente são observadas. Neste capítulo, fomos direto aos casos extremos e tentamos mostrar como você pode enviar SQL, tanto com recursos padrão quanto com determinados recursos específicos do fornecedor.

## APÊNDICE A

# Atualização de função de Window

As receitas deste livro aproveitam ao máximo as funções de Window adicionadas ao padrão ISO SQL em 2003, bem como as funções de Window específicas do fornecedor. Este apêndice pretende servir como uma breve visão geral de como funcionam as funções da Window. As funções de Window tornam muitas tarefas normalmente difíceis (isto é, difíceis de resolver usando SQL padrão) bastante fáceis. Para obter uma lista completa das funções de Window disponíveis, sintaxe completa e cobertura detalhada de como elas funcionam, consulte a documentação do seu fornecedor.

## Agrupamento

Antes de passar para as funções de Window, é crucial que você entenda como o agrupamento funciona em SQL – o conceito de agrupamento de resultados em SQL pode ser difícil de dominar. Os problemas decorrem do não entendimento completo de como funciona a cláusula GROUP BY e por que certas consultas retornam determinados resultados ao usar GROUP BY.

Simplificando, agrupar é uma forma de organizar linhas semelhantes. Ao usar GROUP BY em uma consulta, cada linha no conjunto de resultados é um grupo e representa uma ou mais linhas com os mesmos valores em uma ou mais colunas especificadas. Essa é a essência da questão.

Se um grupo é simplesmente uma instância única de uma linha que representa uma ou mais linhas com o mesmo valor para uma determinada coluna (ou colunas), então exemplos práticos de grupos da tabela EMP incluem *todos os funcionários do departamento 10*(o valor comum para esses funcionários que permite que eles estejam no mesmo grupo é DEPTNO=10) ou *todos os funcionários*(o valor comum para esses funcionários que permite que eles estejam no mesmo grupo é JOB=CLERK). Considere as seguintes perguntas. A primeira mostra todos os funcionários do departamento 10; a segunda consulta agrupa os funcionários do departamento 10 e retorna as seguintes informações sobre o grupo: o número de linhas (membros) do grupo, o maior salário e o menor salário:

```
select deptno,ename
  from emp
 where deptno=10
```

DEPTNO ENAME

```
-----  
10 CLARK  
10 KING  
10 MILLER
```

```
select deptno,
       count(*) as cnt,
       max(sal) as hi_sal,
       min(sal) as lo_sal
  from emp
 where deptno=10
 group by deptno
```

DEPTNO	CNT	HI_SAL	LO_SAL
10	3	5000	1300

Se você não conseguisse agrupar os funcionários do departamento 10, para obter as informações na segunda consulta, você teria que inspecionar manualmente as linhas desse departamento (trivial se houver apenas três linhas, mas e se houvesse três milhões linhas?). Então, por que alguém iria querer se agrupar? As razões para isso variam; talvez você queira ver quantos grupos diferentes existem ou quantos membros (linhas) existem em cada grupo. Como você pode ver neste exemplo simples, o agrupamento permite obter informações sobre muitas linhas em uma tabela sem precisar inspecioná-las uma por uma.

## Definição de um grupo SQL

Em matemática, um grupo é definido, em sua maior parte, como  $(G, \cdot, e)$ , onde  $G$  é um conjunto,  $\cdot$  é uma operação binária em  $G$ , e  $e$  é membro de  $G$ . Usaremos esta definição como base para o que é um grupo SQL. Um grupo SQL será definido como  $(G, e)$ , onde  $G$  é um conjunto de resultados de uma consulta única ou independente que usa GROUP BY,  $e$  é membro de  $G$ , e os seguintes axiomas são satisfeitos:

- Para cada  $e$  em  $G$ ,  $e$  é distinto e representa uma ou mais instâncias de  $e$ .
- Para cada  $e$  em  $G$ , a função agregada COUNT retorna um valor  $> 0$ .



O conjunto de resultados está incluído na definição de um grupo SQL para reforçar o fato de que estamos definindo o que são grupos ao trabalhar apenas com consultas. Assim, seria correto substituir  $e$  em cada axioma com a palavra *row* porque as linhas no conjunto de resultados são tecnicamente os grupos.

Como essas propriedades são fundamentais para o que consideramos um grupo, é importante provarmos que são verdadeiras (e continuaremos a fazê-lo através do uso de alguns exemplos de consultas SQL).

### Os grupos não estão vazios

Pela sua própria definição, um grupo deve ter pelo menos um membro (ou linha). Se aceitarmos isso como verdade, então pode-se dizer que um grupo não pode ser criado a partir de uma mesa vazia. Para provar que essa proposição é verdadeira, basta tentar provar que ela é falsa. O exemplo a seguir cria uma tabela vazia e tenta criar grupos por meio de três consultas diferentes nessa tabela vazia:

```
create table fruits (name varchar(10))

select name
  from fruits
group by name

(no rows selected)

select count(*) as cnt
  from fruits
group by name

(no rows selected)

select name, count(*) as cnt
  from fruits
group by name

(no rows selected)
```

Como você pode ver nessas consultas, é impossível criar o que o SQL considera um grupo a partir de uma tabela vazia.

### Grupos são distintos

Agora vamos provar que os grupos criados via consultas com cláusula GROUP BY são distintos. O exemplo a seguir insere cinco linhas na tabela FRUITS e cria grupos a partir dessas linhas:

```
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Apple')
insert into fruits values ('Peach')

select *
  from fruits
```

NAME

```
-----  
Oranges  
Oranges  
Oranges  
Apple  
Peach
```

```
select name  
      from fruits  
     group by name
```

NAME

```
-----  
Apple  
Oranges  
Peach
```

```
select name, count(*) as cnt  
      from fruits  
     group by name
```

NAME	CNT
Apple	1
Oranges	3
Peach	1

A primeira consulta mostra que “Oranges” ocorre três vezes na tabela FRUITS. No entanto, a segunda e a terceira consultas (usando GROUP BY) retornam apenas uma instância de “Oranges”. Juntas, essas consultas provam que as linhas no conjunto de resultados (*e* em *G*, de nossa definição) são distintos e cada valor de NAME representa uma ou mais instâncias de si mesmo na tabela FRUITS.

Saber que os grupos são distintos é importante porque significa que, normalmente, você não usaria a palavra-chave DISTINCT em sua lista SELECT ao usar GROUP BY em suas consultas.



Não fingimos que GROUP BY e DISTINCT são iguais. Eles representam dois conceitos completamente diferentes. Afirmamos que os itens listados na cláusula GROUP BY serão distintos no conjunto de resultados e que usar DISTINCT e também GROUP BY é redundante.

## Axioma de Frege e Paradoxo de Russell

Para aqueles que estão interessados, Frege's *axioma da abstração*, baseado na solução de Cantor para definir a adesão a conjuntos infinitos ou incontáveis, afirma que, dada uma propriedade de identificação específica, existe um conjunto cujos membros são apenas os itens que possuem essa propriedade. A fonte do problema, conforme colocado por Robert Stoll, “é o uso irrestrito do princípio da abstração”. Bertrand Russell pediu a Gottlob Frege que considerasse um conjunto cujos membros são conjuntos e têm a propriedade definidora de não serem membros de si mesmos.

Como Russell apontou, o axioma da abstração dá muita liberdade porque você está simplesmente especificando uma condição ou propriedade para definir a pertença a um conjunto; assim, uma contradição pode ser encontrada. Para explicar melhor como uma contradição pode ser encontrada, ele criou o “quebra-cabeça de Barber”. O quebra-cabeça do Barbeiro afirma:

Numa certa cidade há um barbeiro que faz a barba de todos aqueles homens, e apenas dos homens que não se barbeiam. Se isso for verdade, quem faz a barba do barbeiro?

Para um exemplo mais concreto, considere o conjunto que pode ser descrito como:

Para todos os membros  $x$  em  $e$  que satisfazem uma condição específica ( $P$ ).

A notação matemática para esta descrição é:

$$\{x \in y \mid P(x)\}$$

Como o conjunto anterior considera *apenas aqueles x em y que satisfazem uma condição* ( $P$ ), você pode achar mais intuitivo descrever o conjunto como *x é um membro de y se e somente se x satisfaz uma condição* ( $P$ ).

Neste ponto vamos definir esta condição  $P(x)$  como *x não é membro de x*:

$$(x \notin x)$$

O conjunto agora é definido como *x é membro de y se e somente se x não é membro de x*:

$$\{x \in y \mid (x \notin x)\}$$

O paradoxo de Russell pode ainda não estar claro para você, mas pergunte-se o seguinte: o conjunto anterior pode ser membro de si mesmo? Vamos supor que  $x=y$  e olhe para o conjunto novamente. O seguinte conjunto pode ser definido como *y é membro de y se e somente se y não for membro de y*:

$$\{y \in y \mid (y \notin y)\}$$

Simplificando, o paradoxo de Russell deixa-nos numa posição de ter um conjunto que é simultaneamente membro de si mesmo e não membro de si mesmo, o que é uma contradição. O pensamento intuitivo levaria alguém a acreditar que isso não é um problema; na verdade, como pode um conjunto ser membro de si mesmo? Afinal, o conjunto de todos os livros não é um livro. Então, por que existe esse paradoxo e como pode ser um problema? Torna-se um problema quando se consideram aplicações mais abstratas da teoria dos conjuntos. Por exemplo, uma aplicação “prática” do paradoxo de Russell pode ser demonstrada considerando o conjunto de todos os conjuntos.

Se permitirmos que tal conceito exista, então, pela sua própria definição, ele deve ser um membro de si mesmo (é, afinal, o conjunto de todos os conjuntos). O que acontece então quando você aplica o anterior  $P(x)$  ao conjunto de todos os conjuntos? Dito de forma simples, o paradoxo de Russell afirmaria que o conjunto de todos os conjuntos é membro de si mesmo se e somente se não for membro de si mesmo — claramente uma contradição.

Para aqueles que estão interessados, Ernst Zermelo desenvolveu o esquema axiomático da separação (também conhecido como *esquema axiomático de subconjuntos* ou o *axioma da especificação*) para contornar elegantemente o paradoxo de Russell na teoria axiomática dos conjuntos.

### COUNT nunca é zero

As consultas e os resultados da seção anterior também provam o axioma final de que a função agregada COUNT nunca retornará zero quando usada em uma consulta com GROUP BY em uma tabela não vazia. Não deveria ser surpresa que você não possa retornar uma contagem zero para um grupo. Já provamos que um grupo não pode ser criado a partir de uma mesa vazia; portanto, um grupo deve ter pelo menos uma linha. Se existir pelo menos uma linha, a contagem sempre será pelo menos uma.



Lembre-se, estamos falando sobre usar COUNT com GROUP BY, e não COUNT sozinho. Uma consulta usando COUNT sem GROUP BY em uma tabela vazia retornará, é claro, zero.

## Paradoxos

A seguinte citação é de Gottlob Frege em resposta à descoberta de Bertrand Russell de uma contradição ao axioma da abstração de Frege na teoria dos conjuntos:

Dificilmente algo mais infeliz pode acontecer a um escritor científico do que ter um dos alicerces de seu edifício abalado após a conclusão do trabalho.... Esta foi a posição que me foi colocada por uma carta do Sr. Bertrand Russell, justamente quando a impressão deste volume estava quase concluída.

Os paradoxos muitas vezes fornecem cenários que parecem contradizer teorias ou ideias estabelecidas. Em muitos casos, estas contradições são localizadas e podem ser “contornadas”, ou são aplicáveis a casos de teste tão pequenos que podem ser ignoradas com segurança.

Você já deve ter adivinhado que o ponto principal de toda essa discussão sobre paradoxos é que existe um paradoxo em relação à nossa definição de grupo SQL, e esse paradoxo deve ser abordado. Embora nosso foco agora esteja em grupos, em última análise estamos discutindo consultas SQL. Na cláusula GROUP BY, uma consulta pode ter uma ampla gama de valores, como constantes, expressões ou, mais comumente, colunas de uma tabela. Pagamos um preço por essa flexibilidade, porque NULL é um “valor” válido em SQL. NULLs apresentam problemas porque são efetivamente ignorados por funções agregadas.,

Dito isso, se uma tabela consistir em uma única linha e seu valor for NULL, o que a função agregada COUNT retornaria quando usada em uma consulta GROUP BY? Pela nossa própria definição, ao usar GROUP BY e a função agregada COUNT, um valor  $\geq 1$  deve ser retornado. O que acontece, então, no caso de valores ignorados por funções como COUNT, e o que isso significa para a nossa definição de GROUP? Considere o exemplo a seguir, que revela o paradoxo do grupo NULL (usando a função COALESCE quando necessário para facilitar a leitura):

```
select *
  from fruits

NAME
-----
Oranges
Oranges
Oranges
Apple
Peach

insert into fruits values (null)

select coalesce(name,'NULL') as name
      from fruits

NAME
-----
Oranges
Oranges
Oranges
Apple
Peach
NULL
NULL
NULL
NULL
NULL
NULL

select coalesce(name,'NULL') as name,
      count(name) as cnt
  from fruits
group by name

NAME          CNT
-----  -----
Apple           1
NULL            0
```

Oranges	3
Peach	1

Parece que a presença de valores NULL em nossa tabela introduz uma contradição, ou paradoxo, em nossa definição de grupo SQL. Felizmente, esta contradição não é um verdadeiro motivo de preocupação, porque o paradoxo tem mais a ver com a implementação de funções agregadas do que com a nossa definição. Considere a consulta final do conjunto anterior; uma declaração geral do problema para essa consulta seria:

Conte o número de vezes que cada nome ocorre na tabela FRUITS ou conte o número de membros de cada grupo.

Examinando as instruções INSERT anteriores, fica claro que existem cinco linhas com valores NULL, o que significa que existe um grupo NULL com cinco membros.



Embora NULL certamente possua propriedades que o diferenciem de outros valores, ele é, no entanto, um valor e pode de fato ser um grupo.

Como, então, podemos escrever a consulta para retornar uma contagem de 5 em vez de 0, retornando assim a informação que procuramos e ao mesmo tempo em conformidade com nossa definição de grupo? O exemplo a seguir mostra uma solução alternativa para lidar com o paradoxo do grupo NULL:

```
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
```

NAME	CNT
Apple	1
Oranges	3
Peach	1
NULL	5

A solução alternativa é usar COUNT(\*) em vez de COUNT(NAME) para evitar o paradoxo do grupo NULL. As funções agregadas ignorarão valores NULL se existir algum na coluna passada a elas. Assim, para evitar zero ao usar COUNT, não passe o nome da coluna; em vez disso, passe um asterisco (\*). Isso faz com que \* a função COUNT conte linhas em vez dos valores reais das colunas, portanto, se os valores reais são NULL ou não NULL é irrelevante.

Mais um paradoxo tem a ver com o axioma de que cada grupo num conjunto de resultados (para cada  $e$  em  $G$ ) é distinto. Devido à natureza dos conjuntos de resultados e tabelas SQL, que são definidos com mais precisão como multisets ou “bags”, e não como conjuntos (porque linhas duplicadas são permitidas), é possível retornar um conjunto de resultados com grupos duplicados. Considere as seguintes consultas:

```

select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
 union all
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name

```

NAME	CNT
Apple	1
Oranges	3
Peach	1
NULL	5
Apple	1
Oranges	3
Peach	1
NULL	5

```

select x.*
  from (
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
 ) x,
 (select deptno from dept) y

```

NAME	CNT
Apple	1
Oranges	3
Peach	1
NULL	5

Como você pode ver nessas consultas, os grupos se repetem de fato nos resultados finais. Felizmente, isto não é motivo de grande preocupação porque representa apenas um paradoxo parcial.

A primeira propriedade de um grupo afirma que para  $(G, e)$ ,  $G$  é um conjunto de resultados de uma consulta única ou independente que usa GROUP BY. Simplificando, o conjunto de resultados de qualquer consulta GROUP BY está em conformidade com nossa definição de grupo. Somente quando você combina os conjuntos de resultados de duas consultas GROUP BY para criar um multiconjunto é que os grupos podem se repetir. A primeira consulta no exemplo anterior usa UNION ALL, que não é uma operação de conjunto, mas uma operação multiset, e invoca GROUP BY duas vezes, executando efetivamente duas consultas.



Se você usar UNION, que é uma operação definida, você não verá grupos repetidos.

A segunda consulta do conjunto anterior usa um produto cartesiano, que só funciona se você primeiro materializar o grupo e depois realizar o cartesiano. Assim, a consulta GROUP BY quando independente está em conformidade com nossa definição. Nenhum dos dois exemplos prejudica a definição de um grupo SQL. Eles são mostrados para fins de integridade e para que você saiba que quase tudo é possível em SQL.

## Relacionamento entre SELECT e GROUP BY

Com o conceito de grupo definido e comprovado, é hora de passar para questões mais práticas relativas a consultas usando GROUP BY. É importante entender o relacionamento entre a cláusula SELECT e a cláusula GROUP BY ao agrupar em SQL. É importante ter em mente, ao usar funções agregadas como COUNT, que qualquer item da sua lista SELECT que não seja usado como argumento para uma função agregada deve fazer parte do seu grupo. Por exemplo, se você escrever uma cláusula SELECT como esta:

```
select deptno, count(*) as cnt
  from emp
```

então você deve listar DEPTNO em sua cláusula GROUP BY:

```
select deptno, count(*) as cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

Constantes, valores escalares retornados por funções definidas pelo usuário, funções de Window e subconsultas escalares não correlacionadas são exceções a esta regra.

Como a cláusula SELECT é avaliada após a cláusula GROUP BY, essas construções são permitidas na lista SELECT e não precisam (e em alguns casos não podem) ser especificadas na cláusula GROUP BY. Por exemplo:

```
select 'hello' as msg,
       1 as num,
       deptno,
       (select count(*) from emp) as total,
       count(*) as cnt
  from emp
 group by deptno
```

MSG	NUM	DEPTNO	TOTAL	CNT
hello	1	10	14	3
hello	1	20	14	5
hello	1	30	14	6

Não deixe esta pergunta confundir você. Os itens da lista SELECT não listados na cláusula GROUP BY não alteram o valor de CNT para cada DEPTNO, nem os valores de DEPTNO mudam. Com base nos resultados da consulta anterior, podemos definir a regra sobre a correspondência de itens na lista SELECT e a cláusula GROUP BY ao usar agregações com um pouco mais de precisão:

Os itens em uma lista SELECT que podem potencialmente alterar o grupo ou alterar o valor retornado por uma função agregada devem ser incluídos na cláusula GROUP BY.

Os itens adicionais na lista anterior SELECT não alteraram o valor de CNT para nenhum grupo (cada DEPTNO), nem alteraram os próprios grupos.

Agora é justo perguntar: exatamente quais itens de uma lista SELECT podem alterar um agrupamento ou o valor retornado por uma função agregada? A resposta é simples: outras colunas da(s) tabela(s) que você está selecionando. Considere a perspectiva de adicionar a coluna JOB à consulta que estamos analisando:

```
select deptno, job, count(*) as cnt
      from emp
 group by deptno, job
```

DEPTNO	JOB	CNT
10	CLERK	1
10	MANAGER	1
10	PRESIDENT	1
20	CLERK	2
20	ANALYST	2
20	MANAGER	1
30	CLERK	1
30	MANAGER	1
30	SALESMAN	4

Ao listar outra coluna, JOB, da tabela EMP, estamos alterando o grupo e alterando o conjunto de resultados. Assim, devemos agora incluir JOB na cláusula GROUP BY junto com DEPTNO; caso contrário, a consulta falhará. A inclusão de JOB nas cláusulas SELECT/GROUP BY altera a consulta de “Quantos funcionários há em cada departamento?” para “Quantos tipos diferentes de funcionários existem em cada departamento?” Observe novamente que os grupos são distintos; os valores para DEPTNO e JOB *individualmente* não são distintos, mas a combinação dos dois (que é o que está no GROUP BY e na lista SELECT e, portanto, no grupo) é distinta (por exemplo, 10 e CLERK aparecem apenas uma vez).

Se você optar por não colocar itens que não sejam funções agregadas na lista SELECT, poderá listar qualquer coluna válida desejada na cláusula GROUP BY. Considere as duas perguntas a seguir, que destacam esse fato:

```
select count(*)
  from emp
 group by deptno

COUNT(*)
-----
 3
 5
 6

select count(*)
  from emp
 group by deptno,job

COUNT(*)
-----
 1
 1
 1
 2
 2
 1
 1
 1
 4
```

A inclusão de itens que não sejam funções agregadas na lista SELECT não é obrigatória, mas geralmente melhora a legibilidade e a usabilidade dos resultados.



Como regra, ao usar GROUP BY e funções agregadas, quaisquer itens na lista SELECT (da(s) tabela(s) na cláusula FROM) não usados como argumento para uma função agregada devem ser incluídos na cláusula GROUP BY. No entanto, o MySQL tem um “recurso” que permite que você se desvie desta regra, permitindo que você coloque itens em sua lista SELECT (que são colunas na(s) tabela(s) que você está selecionando) que não são usados como argumentos para uma agregação. função e que não estão presentes em sua cláusula GROUP BY. Usamos o termo *recurso* vagamente aqui, pois seu uso é um bug esperando para acontecer. Na verdade, se você usa MySQL e se preocupa com a precisão de suas consultas, sugerimos que você remova esse, ahem, “recurso”.

## Windows

Depois de entender o conceito de agrupamento e uso de agregações em SQL, será fácil entender as funções da Window. As funções de Window, como as funções agregadas, realizam uma agregação em um conjunto definido (um grupo) de linhas, mas em vez de retornar um valor por grupo, as funções de Window podem retornar vários valores para cada grupo. O grupo de linhas para realizar a agregação é o *Window*. O DB2 realmente chama essas funções *funções de processamento analítico online (OLAP)*, e a Oracle os chama *funções analíticas*, mas o padrão ISO SQL as chama de funções de Window, então esse é o termo usado neste livro.

### Um exemplo simples

Digamos que você queira contar o número total de funcionários em todos os departamentos. O método tradicional para fazer isso é emitir uma consulta COUNT(\*) contra toda a tabela EMP:

```
select count(*) as cnt
  from emp
```

CNT
-----
14

Isso é bastante fácil, mas muitas vezes você desejará acessar esses dados agregados de linhas que não representam uma agregação ou que representam uma agregação diferente. As funções da Window facilitam o trabalho com esses problemas. Por exemplo, a consulta a seguir mostra como você pode usar uma função de Window para acessar dados agregados (a contagem total de funcionários) de linhas de detalhes (uma por funcionário):

```
select ename,
       deptno,
       count(*) over() as cnt
  from emp
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	14
KING	10	14
MILLER	10	14
SMITH	20	14
ADAMS	20	14
FORD	20	14
SCOTT	20	14
JONES	20	14
ALLEN	30	14
BLAKE	30	14
MARTIN	30	14
JAMES	30	14
TURNER	30	14
WARD	30	14

A invocação da função de Window neste exemplo é COUNT (\*) OVER (). A presença da palavra-chave OVER indica que a invocação de COUNT será tratada como uma função de Window, não como uma função agregada. Em geral, o padrão SQL permite que todas as funções agregadas também sejam funções de Window, e a palavra-chave OVER é como a linguagem distingue entre os dois usos.

Então, o que a função de Window COUNT (\*) OVER () fez exatamente? Para cada linha retornada na consulta, ela retornou a contagem de *todas as linhas* na tabela. Como sugerem os parênteses vazios, a palavra-chave OVER aceita cláusulas adicionais para afetar o intervalo de linhas que uma determinada função de Window considera. Na ausência de tais cláusulas, a função de Window examina todas as linhas do conjunto de resultados, e é por isso que você vê o valor 14 repetido em cada linha da saída.

Esperamos que você esteja começando a ver a grande utilidade das funções de Window, que permitem trabalhar com vários níveis de agregação em uma linha. À medida que você avança neste apêndice, você começará a ver ainda mais como essa habilidade pode ser incrivelmente útil.

## Ordem de Avaliação

Antes de se aprofundar na cláusula OVER, é importante observar que as funções de Window são executadas como a última etapa do processamento SQL antes da cláusula ORDER BY. Como exemplo de como as funções de Window são processadas por último, vamos pegar a consulta da seção anterior e usar uma cláusula WHERE para filtrar funcionários do DEPTNO 20 e 30:

```
select ename,
       deptno,
       count(*) over() as cnt
  from emp
 where deptno = 10
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3

O valor de CNT para cada linha não é mais 14, agora é 3. Neste exemplo, é a cláusula WHERE que restringe o conjunto de resultados a três linhas; portanto, a função de window contará apenas três linhas (há apenas três linhas disponíveis para a função de window quando o processamento atingir a parte SELECT da consulta). Neste exemplo você pode ver que as funções de window realizam seus cálculos após cláusulas como WHERE e GROUP BY serem avaliadas.

## Partições

Use a cláusula PARTITION BY para definir um *partição* ou grupo de linhas para realizar uma agregação. Como já vimos, se você usar parênteses vazios, todo o conjunto de resultados será a partição sobre a qual uma agregação de função de window será calculada. Você pode pensar na cláusula PARTITION BY como um “GROUP BY em movimento” porque, diferentemente de um GROUP BY tradicional, um grupo criado por PARTITION BY não é distinto em um conjunto de resultados. Você pode usar PARTITION BY para calcular uma agregação em um grupo definido de linhas (redefinindo quando um novo grupo é encontrado) e, em vez de ter um grupo representando todas as instâncias desse valor na tabela, cada valor (cada membro em cada grupo) é retornado. Considere a seguinte consulta:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as cnt
  from emp
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3
SMITH	20	5
ADAMS	20	5
FORD	20	5
SCOTT	20	5
JONES	20	5

ALLEN	30	6
BLAKE	30	6
MARTIN	30	6
JAMES	30	6
TURNER	30	6
WARD	30	6

Esta consulta ainda retorna 14 linhas, mas agora o COUNT é executado para cada departamento como resultado da cláusula PARTITION BY DEPTNO. Cada funcionário do mesmo departamento (na mesma partição) terá o mesmo valor para CNT, porque a agregação não será redefinida (recalculada) até que um novo departamento seja encontrado. Observe também que você está retornando informações sobre cada grupo, juntamente com os membros de cada grupo. Você pode pensar na consulta anterior como uma versão mais eficiente do seguinte:

```
select e.ename,
       e.deptno,
       (select count(*) from emp d
        where e.deptno=d.deptno) as cnt
  from emp e
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3
SMITH	20	5
ADAMS	20	5
FORD	20	5
SCOTT	20	5
JONES	20	5
ALLEN	30	6
BLAKE	30	6
MARTIN	30	6
JAMES	30	6
TURNER	30	6
WARD	30	6

Além disso, o que é interessante na cláusula PARTITION BY é que ela executa seus cálculos independentemente de outras funções da window, particionando por diferentes colunas na mesma instrução SELECT. Considere a seguinte consulta, que retorna cada funcionário, seu departamento, o número de funcionários em seu respectivo departamento, seu cargo e o número de funcionários com o mesmo cargo:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as dept_cnt,
       job,
       count(*) over(partition by job) as job_cnt
```

```
from emp
order by 2
```

ENAME	DEPTNO	DEPT_CNT	JOB	JOB_CNT
MILLER	10	3	CLERK	4
CLARK	10	3	MANAGER	3
KING	10	3	PRESIDENT	1
SCOTT	20	5	ANALYST	2
FORD	20	5	ANALYST	2
SMITH	20	5	CLERK	4
JONES	20	5	MANAGER	3
ADAMS	20	5	CLERK	4
JAMES	30	6	CLERK	4
MARTIN	30	6	SALESMAN	4
TURNER	30	6	SALESMAN	4
WARD	30	6	SALESMAN	4
ALLEN	30	6	SALESMAN	4
BLAKE	30	6	MANAGER	3

Neste conjunto de resultados, você pode ver que os funcionários do mesmo departamento têm o mesmo valor para DEPT\_CNT e que os funcionários que têm o mesmo cargo têm o mesmo valor para JOB\_CNT.

Até agora deve estar claro que a cláusula PARTITION BY funciona como uma cláusula GROUP BY, mas o faz sem ser afetado pelos outros itens da cláusula SELECT e sem exigir que você escreva uma cláusula GROUP BY.

## Efeito de NULLs

Assim como a cláusula GROUP BY, a cláusula PARTITION BY agrupa todos os NULLs em um grupo ou partição. Assim, o efeito de NULLs ao usar PARTITION BY é semelhante ao uso de GROUP BY. A consulta a seguir usa uma função de window para contar o número de funcionários com cada comissão distinta (retornando -1 no lugar de NULL para facilitar a leitura):

```
select coalesce(comm,-1) as comm,
       count(*)over(partition by comm) as cnt
  from emp
```

COMM	CNT
0	1
300	1
500	1
1400	1
-1	10
-1	10
-1	10
-1	10
-1	10

-1	10
-1	10
-1	10
-1	10
-1	10

Como COUNT(\*) é usado, a função conta linhas. Você pode ver que existem 10 funcionários com comissões NULAS. Use COMM em vez de \*, entretanto, e você obterá resultados bem diferentes:

```
select coalesce(comm,-1) as comm,
       count(comm)over(partition by comm) as cnt
  from emp
```

COMM	CNT
0	1
300	1
500	1
1400	1
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0

Esta consulta utiliza COUNT (COMM), o que significa que apenas os valores não NULL na coluna COMM são contados. Há um funcionário com comissão de 0, um funcionário com comissão de 300 e assim por diante. Mas observe a contagem para quem tem comissões NULL! Essas contagens são 0. Por quê? Como as funções agregadas ignoram valores NULL ou, mais precisamente, as funções agregadas contam apenas valores não NULL.



Ao usar COUNT, considere se deseja incluir NULLs. Use COUNT(column) para evitar contar NULLs. Use COUNT(\*) se desejar incluir NULLs (já que você não está mais contando os valores reais das colunas, você está contando as linhas).

## Quando o pedido é importante

Às vezes, a ordem na qual as linhas são tratadas por uma função de window é importante para os resultados que você deseja obter de uma consulta. Por esse motivo, a sintaxe da função de window inclui uma subcláusula ORDER BY que você pode colocar dentro de uma cláusula OVER.

Use a cláusula ORDER BY para especificar como as linhas são ordenadas com uma partição (lembre-se, “partição” na ausência de uma cláusula PARTITION BY significa todo o conjunto de resultados).



Algumas funções de window *require* você impõe ordem nas partições de linhas afetadas. Assim, para algumas funções de window, uma cláusula ORDER BY é obrigatória. No momento em que este artigo foi escrito, o SQL Server não permitia ORDER BY na cláusula OVER quando usado com funções de window agregadas. O SQL Server permite ORDER BY na cláusula OVER quando usado com funções de classificação de window.

Ao usar uma cláusula ORDER BY na cláusula OVER de uma função de window, você está especificando duas coisas:

- Como as linhas na partição são ordenadas
- Quais linhas estão incluídas no cálculo

Considere a seguinte consulta, que soma e calcula um total acumulado de salários para funcionários no DEPTNO 10:

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate) as running_total
  from emp
 where deptno=10
```

DEPTNO	ENAME	HIREDATE	SAL	TOTAL1	TOTAL2	RUNNING_TOTAL
10	CLARK	09-JUN-1981	2450	8750	8750	2450
10	KING	17-NOV-1981	5000	8750	8750	7450
10	MILLER	23-JAN-1982	1300	8750	8750	8750



Apenas para mantê-lo alerta, inclui uma soma entre parênteses vazios. Observe como TOTAL1 e TOTAL2 têm os mesmos valores. Por que? Mais uma vez, a ordem na qual as funções da window são avaliadas responde à questão. A cláusula WHERE filtra o conjunto de resultados de forma que apenas os salários do DEPTNO 10 sejam considerados para somatório. Neste caso, existe apenas uma partição – todo o conjunto de resultados, que consiste apenas em salários do DEPTNO 10. Assim, TOTAL1 e TOTAL2 são iguais.

Observando os valores retornados pela coluna SAL, você pode ver facilmente de onde vêm os valores de RUNNING\_TOTAL. Você pode observar os valores e adicioná-los você mesmo para calcular o total acumulado.

Mas o mais importante é que, em primeiro lugar, a inclusão de um ORDER BY na cláusula OVER criou um total acumulado? A razão é que, quando você usa ORDER BY na cláusula OVER, você está especificando uma window padrão “móvel” ou “deslizante” dentro da partição, mesmo que não a veja. A cláusula ORDER BY HIREDATE termina a soma em HIREDATE na linha atual.

A consulta a seguir é igual à anterior, mas usa a cláusula RANGE BETWEEN (sobre a qual você aprenderá mais adiante) para especificar explicitamente o comportamento padrão resultante de ORDER BY HIREDATE:

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate
                     range between unbounded preceding
                     and current row) as running_total
  from emp
 where deptno=10
```

DEPTNO	ENAME	HIREDATE	SAL	TOTAL1	TOTAL2	RUNNING_TOTAL
10	CLARK	09-JUN-1981	2450	8750	8750	2450
10	KING	17-NOV-1981	5000	8750	8750	7450
10	MILLER	23-JAN-1982	1300	8750	8750	8750

A cláusula RANGE BETWEEN que você vê nesta consulta é denominada *cláusula de enquadramento* pela ANSI, e usaremos esse termo aqui. Agora, deve ser fácil ver por que especificar um ORDER BY na cláusula OVER criou um total acumulado; (por padrão) instruímos a consulta a somar todas as linhas começando na linha atual e incluir todas as linhas anteriores (“anteriores” conforme definido em ORDER BY, neste caso ordenando as linhas por HIREDATE).

## A cláusula de enquadramento

Vamos aplicar a cláusula de enquadramento da consulta anterior ao conjunto de resultados, começando com o primeiro funcionário contratado, que se chama CLARK:

1. Começando com o salário de CLARK, 2.450, e incluindo todos os funcionários contratados antes de CLARK, calcule uma soma. Como CLARK foi o primeiro funcionário contratado no DEPTNO 10, a soma é simplesmente o salário de CLARK, 2.450, que é o primeiro valor retornado por RUNNING\_TOTAL.
2. Vamos passar para o próximo funcionário baseado em HIREDATE, chamado KING, e aplicar a cláusula de enquadramento mais uma vez. Calcule uma soma no SAL começando pela linha atual, 5.000 (salário de KING), e inclua todas as linhas anteriores (todos os funcionários contratados antes REI).

CLARK é o único contratado antes de KING, então a soma é  $5000 + 2450$ , que é 7450, segundo valor retornado por RUNNING\_TOTAL.

3. Passando para MILLER, o último funcionário da partição baseada em HIREDATE, vamos aplicar mais uma vez a cláusula de enquadramento. Calcule uma soma no SAL começando pela linha atual, 1300 (salário de MILLER), e inclua todas as linhas anteriores (todos os funcionários contratados antes de MILLER). CLARK e KING foram ambos contratados antes da MILLER e, portanto, seus salários estão incluídos no RUNNING\_TOTAL da MILLER:  $2450 + 5000 + 1300$  é 8750, que é o valor de RUNNING\_TOTAL para a MILLER.

Como você pode ver, é realmente a cláusula de enquadramento que produz o total corrente. O ORDER BY define a ordem de avaliação e também implica um enquadramento padrão.

Em geral, a cláusula de enquadramento permite definir diferentes “subjanelas” de dados para incluir em seus cálculos. Há muitas maneiras de especificar essas subjanelas. Considere a seguinte consulta:

```
select deptno,
       ename,
       sal,
       sum(sal)over(order by hiredate
                     range between unbounded preceding
                           and current row) as run_total1,
       sum(sal)over(order by hiredate
                     rows between 1 preceding
                           and current row) as run_total2,
       sum(sal)over(order by hiredate
                     range between current row
                           and unbounded following) as run_total3,
       sum(sal)over(order by hiredate
                     rows between current row
                           and 1 following) as run_total4
  from emp
 where deptno=10
```

DEPTNO	ENAME	SAL	RUN_TOTAL1	RUN_TOTAL2	RUN_TOTAL3	RUN_TOTAL4
10	CLARK	2450	2450	2450	8750	7450
10	KING	5000	7450	7450	6300	6300
10	MILLER	1300	8750	6300	1300	1300

Não se deixe intimidar aqui; esta consulta não é tão ruim quanto parece. Você já viu RUN\_TOTAL1 e os efeitos da cláusula de enquadramento UNBOUNDED PRECEDING AND CURRENT ROW. Aqui está uma rápida descrição do que está acontecendo nos outros exemplos:

Em vez da palavra-chave RANGE, esta cláusula de enquadramento especifica ROWS, o que significa o *frame*, ou janela, será construída contando um certo número de linhas. O 1 PRECEDING significa que o quadro começará com a linha imediatamente anterior à linha atual. A faixa continua pela CURRENT ROW. Então o que você obtém em RUN\_TOTAL2 é a soma do salário do funcionário atual e do funcionário anterior, com base em HIREDATE.

[[sqlckbk-APP-A-NOTE-11]]



Acontece que RUN\_TOTAL1 e RUN\_TOTAL2 são iguais para CLARK e KING. Por que? Pense em quais valores estão sendo somados para cada um desses funcionários, para cada uma das duas funções da window. Pense com cuidado e você obterá a resposta.

#### *RUN\_TOTAL3*

A função de window para RUN\_TOTAL3 funciona exatamente de forma oposta à de RUN\_TOTAL1; em vez de começar na linha atual e incluir todas as linhas anteriores no somatório, o somatório começa na linha atual e inclui todas as linhas subsequentes no somatório.

#### *RUN\_TOTAL4*

Este é o inverso de RUN\_TOTAL2; em vez de começar na linha atual e incluir uma linha anterior no somatório, comece com a linha atual e inclua uma linha subsequente no somatório.



Se você conseguir entender o que foi explicado até agora, não terá problemas com nenhuma das receitas deste livro. Se você não está entendendo, tente praticar com seus próprios exemplos e seus próprios dados. Geralmente é mais fácil aprender codificando novos recursos do que apenas lendo sobre eles.

## Um final de enquadramento

Como exemplo final do efeito da cláusula de enquadramento na saída da consulta, considere a seguinte consulta:

```
select ename,
       sal,
       min(sal)over(order by sal) min1,
       max(sal)over(order by sal) max1,
       min(sal)over(order by sal
                     range between unbounded preceding
                     and unbounded following) min2,
       max(sal)over(order by sal
```

```

range between unbounded preceding
and unbounded following) max2,
min(sal)over(order by sal
range between current row
and current row) min3,
max(sal)over(order by sal
range between current row
and current row) max3,
max(sal)over(order by sal
rows between 3 preceding
and 3 following) max4
from emp

```

ENAME	SAL	MIN1	MAX1	MIN2	MAX2	MIN3	MAX3	MAX4
SMITH	800	800	800	800	5000	800	800	1250
JAMES	950	800	950	800	5000	950	950	1250
ADAMS	1100	800	1100	800	5000	1100	1100	1300
WARD	1250	800	1250	800	5000	1250	1250	1500
MARTIN	1250	800	1250	800	5000	1250	1250	1600
MILLER	1300	800	1300	800	5000	1300	1300	2450
TURNER	1500	800	1500	800	5000	1500	1500	2850
ALLEN	1600	800	1600	800	5000	1600	1600	2975
CLARK	2450	800	2450	800	5000	2450	2450	3000
BLAKE	2850	800	2850	800	5000	2850	2850	3000
JONES	2975	800	2975	800	5000	2975	2975	5000
SCOTT	3000	800	3000	800	5000	3000	3000	5000
FORD	3000	800	3000	800	5000	3000	3000	5000
KING	5000	800	5000	800	5000	5000	5000	5000

OK, vamos analisar esta consulta:

### MIN1

A função de window que gera esta coluna não especifica uma cláusula de enquadramento, portanto, a cláusula de enquadramento padrão UNBOUNDED PRECEDING AND CURRENT ROW entra em ação. Por que MIN1 800 para todas as linhas? É porque o salário mais baixo vem primeiro (ORDER BY SAL), e continua sendo o salário mais baixo, ou mínimo, para sempre.

### MAX1

Os valores de MAX1 são muito diferentes daqueles de MIN1. Por que? A resposta (novamente) é a cláusula de enquadramento padrão UNBOUNDED PRECEDING AND CURRENT ROW. Em conjunto com ORDER BY SAL, esta cláusula enquadrante garante que o salário máximo também corresponderá ao da linha atual.

Considere a primeira linha, para SMITH. Ao avaliar o salário do SMITH e todos os salários anteriores, MAX1 para o SMITH é o salário do SMITH, porque não há salários anteriores. Passando para a próxima linha, JAMES, ao comparar o salário de JAMES com todos os salários anteriores, neste caso comparando com o salário de SMITH, o salário de JAMES é o maior dos dois, e portanto é o máximo.

Se você aplicar esta lógica a todas as linhas, verá que o valor de MAX1 para cada linha é o salário do funcionário atual.

### *MIN2 e MAX2*

A cláusula de enquadramento fornecida para estes é UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING, que é o mesmo que especificar parênteses vazios. Assim, todas as linhas do conjunto de resultados são consideradas no cálculo de MIN e MAX. Como seria de esperar, os valores MIN e MAX para todo o conjunto de resultados são constantes e, portanto, o valor dessas colunas também é constante.

### *MIN3 e MAX3*

A cláusula de enquadramento para estes é LINHA ATUAL E LINHA ATUAL, o que significa simplesmente usar apenas o salário do funcionário atual ao procurar o salário MIN e MAX. Assim, MIN3 e MAX3 são iguais a SAL para cada linha. Isso foi fácil, não foi?

### *MAX4*

A cláusula de enquadramento definida para MAX4 é 3 PRECEDING AND 3 FOLLOWING, o que significa que, para cada linha, considere as três linhas anteriores e as três linhas depois da linha atual, bem como a própria linha atual. Esta invocação específica de MAX(SAL) retornará dessas linhas o valor do salário mais alto.

Se você observar o valor de MAX4 para o funcionário MARTIN, poderá ver como a cláusula de enquadramento é aplicada. O salário de MARTIN é 1.250, e os três salários dos funcionários anteriores ao de MARTIN são WARD (1.250), ADAMS (1.100) e JAMES (950). Os três salários dos funcionários depois de MARTIN são MILLER (1300), TURNER (1500) e ALLEN (1600). De todos esses salários, incluindo o de MARTIN, o mais alto é o de ALLEN e, portanto, o valor de MAX4 para MARTIN é 1600.

## **Legibilidade + Desempenho = Potência**

Como você pode ver, as funções de window são extremamente poderosas, pois permitem escrever consultas que contêm informações detalhadas e agregadas. O uso de funções de window permite que você escreva consultas menores e mais eficientes em comparação ao uso de várias subconsultas escalares e/ou de autojunção. Considere a seguinte consulta, que responde facilmente a todas as seguintes questões: “Qual é o número de funcionários em cada departamento? Quantos tipos diferentes de funcionários existem em cada departamento (por exemplo, quantos funcionários existem no departamento 10)? Quantos funcionários no total estão na tabela EMP?”

```
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over () as total
  from emp
```

DEPTNO	JOB	EMP_CNT	JOB_CNT	TOTAL
10	CLERK	3	1	14
10	MANAGER	3	1	14
10	PRESIDENT	3	1	14
20	ANALYST	5	2	14
20	ANALYST	5	2	14
20	CLERK	5	2	14
20	CLERK	5	2	14
20	MANAGER	5	1	14
30	CLERK	6	1	14
30	MANAGER	6	1	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14

Retornar o mesmo conjunto de resultados sem usar funções de window exigiria um pouco mais de trabalho:

```
select a.deptno, a.job,
       (select count(*) from emp b
        where b.deptno = a.deptno) as emp_cnt,
       (select count(*) from emp b
        where b.deptno = a.deptno and b.job = a.job) as job_cnt,
       (select count(*) from emp) as total
  from emp a
 order by 1,2
```

DEPTNO	JOB	EMP_CNT	JOB_CNT	TOTAL
10	CLERK	3	1	14
10	MANAGER	3	1	14
10	PRESIDENT	3	1	14
20	ANALYST	5	2	14
20	ANALYST	5	2	14
20	CLERK	5	2	14
20	CLERK	5	2	14
20	MANAGER	5	1	14
30	CLERK	6	1	14
30	MANAGER	6	1	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14

A solução sem janela obviamente não é difícil de escrever, mas certamente não é tão limpa ou eficiente (você não verá diferenças de desempenho com uma tabela de 14 linhas, mas tente essas consultas com, digamos, uma tabela de 1.000 ou 10.000 linhas, e então você verá o benefício de usar funções de window em várias auto-junções e subconsultas escalares).

## Fornecendo uma base

Além da legibilidade e do desempenho, as funções de window são úteis para fornecer uma “base” para consultas mais complexas do “estilo de relatório”. Por exemplo, considere a seguinte consulta “estilo de relatório” que usa funções de window em uma visualização embutida e depois agrega os resultados em uma consulta externa. O uso de funções de window permite retornar dados detalhados e agregados, o que é útil para relatórios. A consulta a seguir usa funções de window para localizar contagens usando partições diferentes. Como a agregação é aplicada a diversas linhas, a visualização embutida retorna todas as linhas do EMP, que as expressões externas CASE podem usar para transpor e criar um relatório formatado:

```

select deptno,
       emp_cnt as dept_total,
       total,
       max(case when job = 'CLERK'
                 then job_cnt else 0 end) as clerks,
       max(case when job = 'MANAGER'
                 then job_cnt else 0 end) as mgrs,
       max(case when job = 'PRESIDENT'
                 then job_cnt else 0 end) as prez,
       max(case when job = 'ANALYST'
                 then job_cnt else 0 end) as anals,
       max(case when job = 'SALESMAN'
                 then job_cnt else 0 end) as smen
  from (
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over () as total
  from emp
   ) x
 group by deptno, emp_cnt, total

```

DEPTNO	DEPT_TOTAL	TOTAL	CLERKS	MGRS	PREZ	ANALS	SMEN
10	3	14	1	1	1	0	0
20	5	14	2	1	0	2	0
30	6	14	1	1	0	0	4

A consulta anterior retorna cada departamento, o número total de funcionários em cada departamento, o número total de funcionários na tabela EMP e um detalhamento do número de diferentes tipos de cargos em cada departamento. Tudo isso é feito em uma consulta, sem junções adicionais ou tabelas temporárias!

Como exemplo final de como é fácil responder a várias perguntas usando funções de window, considere a seguinte consulta:

```

select ename as name,
       sal,
       max(sal)over(partition by deptno) as hiDpt,
       min(sal)over(partition by deptno) as loDpt,
       max(sal)over(partition by job) as hiJob,
       min(sal)over(partition by job) as loJob,
       max(sal)over() as hi,
       min(sal)over() as lo,
       sum(sal)over(partition by deptno
                    order by sal,empno) as dptRT,
       sum(sal)over(partition by deptno) as dptSum,
       sum(sal)over() as ttl
  from emp
 order by deptno,dptRT

```

NAME	SAL	HIDPT	LODPT	HIJOB	LOJOB	HI	LO	DPTRT	DPTSUM	TTL
MILLER	1300	5000	1300	1300	800	5000	800	1300	8750	29025
CLARK	2450	5000	1300	2975	2450	5000	800	3750	8750	29025
KING	5000	5000	1300	5000	5000	5000	800	8750	8750	29025
SMITH	800	3000	800	1300	800	5000	800	800	10875	29025
ADAMS	1100	3000	800	1300	800	5000	800	1900	10875	29025
JONES	2975	3000	800	2975	2450	5000	800	4875	10875	29025
SCOTT	3000	3000	800	3000	3000	5000	800	7875	10875	29025
FORD	3000	3000	800	3000	3000	5000	800	10875	10875	29025
JAMES	950	2850	950	1300	800	5000	800	950	9400	29025
WARD	1250	2850	950	1600	1250	5000	800	2200	9400	29025
MARTIN	1250	2850	950	1600	1250	5000	800	3450	9400	29025
TURNER	1500	2850	950	1600	1250	5000	800	4950	9400	29025
ALLEN	1600	2850	950	1600	1250	5000	800	6550	9400	29025
BLAKE	2850	2850	950	2975	2450	5000	800	9400	9400	29025

Esta consulta responde às seguintes perguntas de forma fácil, eficiente e legível (e sem junções adicionais ao EMP!). Basta combinar o funcionário e seu salário com as diferentes linhas do conjunto de resultados para determinar:

- Quem ganha o salário mais alto de todos os funcionários (HI)
- Quem ganha o menor salário de todos os funcionários (LO)
- Quem ganha o salário mais alto do departamento (HIDPT)
- Quem ganha o salário mais baixo do departamento (LODPT)
- Quem ganha o salário mais alto no trabalho (HIJOB)
- Quem ganha o salário mais baixo no trabalho (LOJOB)
- Qual é a soma de todos os salários (TTL)
- Qual é a soma dos salários por departamento (DPTSUM)
- Qual é o total acumulado de todos os salários por departamento (DPTRT)

## APÊNDICE B

# Expressões de tabela comuns (CTE)

Muitas das consultas apresentadas neste livro de receitas vão além do que é possível usando tabelas, pois normalmente estão disponíveis em um banco de dados, especialmente em relação a funções agregadas e funções de janela. Portanto, para algumas consultas, você precisa criar uma tabela derivada – uma subconsulta ou uma expressão de tabela comum (CTE - common table expression).

## Subconsultas

Provavelmente a maneira mais simples de criar uma tabela virtual que permite executar consultas em funções de janela ou funções agregadas é uma subconsulta. Tudo o que é necessário aqui é escrever a consulta necessária entre parênteses e, em seguida, escrever uma segunda consulta que a utilize. A tabela a seguir ilustra o uso de subconsultas com um simple *sagregado duplo*— você deseja encontrar não apenas a contagem de funcionários em cada cargo, mas também identificar o número mais alto, mas não pode aninhar funções agregadas diretamente em uma consulta padrão.

Uma armadilha é que alguns fornecedores exigem que você forneça a tabela e o alias da subconsulta, mas outros não. O exemplo a seguir foi escrito em MySQL, o que requer um alias. O alias aqui é HEAD\_COUNT\_TAB após o parêntese de fechamento.

Outros que exigem um alias são PostgreSQL e SQL Server, enquanto o Oracle não:

```
select max(HeadCount) as HighestJobHeadCount from
(select job,count(empno) as HeadCount
from emp
group by job) head_count_tab
```

## Expressões de tabela comuns (CTE)

Os CTEs foram planejados para superar alguns dos limites das subconsultas e podem ser mais conhecidos por permitir o uso de consultas recursivas no SQL. Na verdade, habilitar a recursão no SQL foi a principal inspiração para os CTEs.

Este exemplo alcança o mesmo resultado que a subconsulta que vimos anteriormente – ele encontra um *agregado duplo*:

```
with head_count_tab (job,HeadCount) as  
  
(select job,count(empno)  
from emp  
group by job)  
  
select max(HeadCount) as HighestJobHeadCount  
from head_count_tab
```

Embora esta consulta resolva um problema simples, ela ilustra as características essenciais de um CTE. Apresentamos a tabela derivada usando a cláusula WITH, especificando os títulos das colunas entre parênteses, e usamos parênteses ao redor da própria consulta da tabela derivada. Se quisermos adicionar mais tabelas derivadas, podemos adicionar mais, desde que separemos cada uma delas com uma vírgula e forneçamos seu nome antes de sua consulta (o inverso de como o alias geralmente funciona em SQL).

Como as consultas internas são apresentadas antes da consulta externa, em muitas circunstâncias elas também podem ser consideradas mais legíveis – elas facilitam o estudo de cada elemento lógico da consulta separadamente para compreender o fluxo lógico. É claro que, como acontece com todas as coisas na codificação, isso irá variar de acordo com as circunstâncias e, às vezes, a subconsulta será mais legível.

Considerando que a recursão é a principal razão da existência dos CTEs, a melhor maneira de demonstrar sua capacidade é por meio de uma consulta recursiva.

A consulta a seguir calcula os primeiros 20 números da sequência de Fibonacci usando um CTE recursivo. Observe que na primeira parte da consulta âncora, podemos inicializar os valores na primeira linha da tabela virtual:

```
with recursive workingTable ( fibNum, NextNumber , index1)  
as  
(select 0,1,1  
union all  
select fibNum+nextNumber,fibNum,index1+1  
from anchor  
where index1<20)  
  
select fibNum from workingTable as fib
```

A sequência de Fibonacci encontra o próximo número somando os números atuais e anteriores; você também pode usar o LAG para obter esse resultado. Porém, neste caso criamos um pseudo-LAG usando duas colunas para contabilizar o número atual e o anterior. Observe a palavra-chave RECURSIVE, que é obrigatória no MySQL, Oracle e PostgreSQL, mas não no SQL Server ou DB2. Nesta consulta, a coluna index1 é amplamente redundante no sentido de não ser usada para o cálculo de Fibonacci. Em vez disso, incluímos isso para simplificar a definição do número de linhas retornadas por meio da cláusula WHERE. Em um CTE recursivo, a cláusula WHERE se torna crucial, pois sem ela a consulta não terminaria (embora neste caso específico, se você tentar excluí-la, provavelmente descobrirá que seu banco de dados gera um erro de estouro quando os números se tornam muito grandes para o tipo de dados).

No final do espectro, não há muita diferença entre uma subconsulta e CTE em termos de usabilidade. Ambos permitem aninhar ou escrever consultas mais complicadas que se referem a outras tabelas derivadas. No entanto, quando você começa a aninhar muitas subconsultas, a legibilidade diminui porque o significado de diferentes variáveis fica oculto em camadas de consulta sucessivas. Em contrapartida, como um CTE organiza cada elemento verticalmente, é mais fácil compreender o significado de cada elemento.

## Resumindo

O uso de tabelas derivadas amplia drasticamente o alcance do SQL. Tanto as subconsultas quanto o CTES são usados muitas vezes ao longo do livro, por isso é importante entender como funcionam, especialmente porque cada um deles tem uma sintaxe específica que você precisa dominar para garantir o sucesso. O CTE recursivo, que agora está disponível nas ofertas dos fornecedores neste livro, é uma das maiores extensões que já ocorreram no SQL, permitindo muitas possibilidades extras.